DB2 for z/OS Best Practices
DB2 10 Migration Planning and Very Early Experiences
Part 1

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 for z/OS development. Welcome to another Web lecture, in this series on DB2 for z/OS Best Practices.

This particular web lecture is about DB2 10 for z/OS migration planning and very early experiences. And my main objective of this web lecture here is to help you migrate to DB2 Version 10 as fast as possible, but most importantly also as safely as possible.

Now let’s turn to slide 2.

Slide 2 (00:30)

Slide 2 is a disclaimer on trademarks that are actually used within this particular presentation.

Now, let’s turn to slide 3.

Slide 3 (00:38)

So, what are the objectives of this web lecture?

First of all I want to share lessons learned with other customers: the surprises and pitfalls that people have run into; provide some hints and tips; address some myths that may have grown up around Version 10; provide additional planning information; but also provide usage guidelines and positioning for new enhancements that were available in Version 10, and some new enhancements that have come into Version 10 after general availability was announced.

Now let’s turn to slide 4.

Slide 4 (01:12)

I want to make a few comments on slide 4 [slide 5] about the beta program for DB2 Version 10. The actual beta was announced in February 2010, and we shipped the beta, or started the beta on March the twelfth 2010. It was the largest beta program ever, and it was the strongest ever customer demand to participate in the program.

There were primarily two main reasons for that. One was the rumours got out about the price/performance improvements, which are most welcome in Version 10, and also because we’ve also solved the scalability issue related to DBM1 31-bit storage.

So, there was very strong customer demand to get into the program. Eventually, we settled on 24 worldwide customers across many industries, some migrating from 9 to 10, others migrating from 8 to 10, in the core program. And then in third-quarter 2010, we extended the beta to other customers. So, all-in-all here, there were 73 parties in the vendor program as well.

So what were the customer focus areas?

Regression testing, and this is pretty much important here, because… Basically, what you want to try and do in any situation is to make or allow the given customer to have a very good day-one experience.

So, one of the things that we wanted to encourage among the customers was to test like you do in production, in the order you do in production. So, for example, the idea was for them to create a performance baseline on Version 9, then migrate into Version 10 CM mode, and test without the rebind, and create a new performance profile, then to rebind in conversion mode and repeat the exercise again, and finally migrate to new-function mode, and repeat the exercise once more.

Another focus area was obviously to validate the out-of-the-box performance improvements, and to explore some of the additional performance opportunities, and also to focus on other scalability improvements and general new function.

Now, let’s turn to slide 6.

Slide 6 (03:16)

What I'd like to do here on slide 6 is just to give some general highlights, and we'll drill down into more detail later.

When I talk here about "good results" or "mixed results," then what I'm talking about is the status as at the program, after the beta program had finished, and after the product has gone general availability.

One of the big success stories in Version 10 is the virtual storage constraint relief, related to the DBM1 address space 31-bit virtual storage. We did have some issues during the program, but by the end of the program we were providing some relief without the rebind. And once the customers actually rebind their plans and packages under Version 10 CM mode, then we have very very generous virtual storage constraint relief.

So, while this is important from a scalability viewpoint, it also opens up opportunities for further price/performance improvements that I'll talk about later.

The second one was INSERT performance. What we wanted to do in Version 10 was provide some general performance improvements for insert across all the table space types, including partitioned, segmented, and UTS. But we also had a goal to improve the performance of universal table space, both PBG and PBR, so that performance was equal or better than the classic partitioned table space.

The good news is that we have provided insert performance improvements across each of the table space types, and in many cases now the performance of UTS is equal or better than partitioned.

But there are some cases, which I'll talk about later, where insert performance for UTS is still not as good as classic table space types.

There was a lot of interest in hash data access as an alternative to clustered index access, to access rows in your DB2 tables. Now the good news is that it works well. However, the sweet spot in terms of use case was a lot smaller than what we expected. And I'll comment later that maybe you have to have at least three levels or more in your index in order to start benefitting from hash access.

There's also a quite long list of improvements that will help the performance of complex queries.

In Version 10, we provide support for inline LOBs. And I am going to talk here about this term "SLOBs" and what I mean is it's just short here for "short large objects." So prior to version 10 here, when you have a LOB data column, then the LOB data column is stored in an auxiliary table space. And in order to access the LOB column value, you have to navigate from the base table row over to the auxiliary table space via an index.

Now there are many types of LOB applications where the LOBs are in fact SLOBs, short large objects. Or it might be the case the 70, 80, or 90 percent of the LOB column values are very short in length. So, in version 10, you have the opportunity to store a certain portion of the LOB data in the base table row, and then once you've gone beyond that value, then the rest of the LOB is stored in the auxiliary table space.

And clearly there is an opportunity here, that if the LOBs are short in size, and you can inline all of the LOB column value

in the base table row, you can generate some pretty significant performance improvements. And so, inline LOBs have proven to be very beneficial to several customers' use cases.

As part of the release, we were trying to focus on issues and problems that were related to scalability. So, once we had addressed the DBM1 31-bit storage, we were also looking around for other areas of contention.

So, there's been a number of changes made to, first of all, improving latch contention management, but also removing some of the latches, and also holding those latches for a shorter periods of time, or some cases, have more granularity in the latch.

From my own personal viewpoint, I view the Version 10 program basically a significant improvement over what we had in Version 9 and Version 8. Why?

First of all in terms of the qualities of the problems found, and also the issues found as well. There were a number of issues that we found, that if it had escaped into the field after GA, or it had been discovered after GA for version 10, they would have caused quite a few problems.

And the last thing was, as the program developed, we saw improved reliability and confidence amongst the customers in terms of being able to successfully run their workload and achieve good performance results.

Now let's turn on to slide 7.

Slide 7 (07:45)

Now I want to talk about mixed results. “Mixed results” means good, but also some bad.

If you look at OLTP performance, we had an aggressive goal for OLTP performance in this release to deliver a 5 to 10 percent, for most cases, reduced CPU. And in many cases customers ran measurements where able to achieve 5 to 10 percent. In a few cases, people were able to over-perform as well. But there were some cases, a few cases, where people had very very simple SQL, where the cost of a transaction meant that the performance improvements in the SQL got amortized away.

So these are situations where you’ve got very skinny transactions, and very skinny packages, and the cost of package allocation outweigh the optimizations that we’ve made in SQL processing performance.

Another area of mixed results was in the area of single-thread bind and rebind. But in conversion mode, and also in new function mode, there is degradation in the CPU and elapsed time for bind and rebind performance. There are several reasons behind that I’ll go into a lot of detail later on.

Part of it is to do with catalog restructure. Part of it is as well is that plan stability is on by default, and the default value is extended. And also DB2 has single-path code for a given release, and even in conversion mode, ahead of the catalog tables in ENFM, which improve concurrent bind and rebind performance. Because we have a single code path, then basically, we’re following a similar access path compared to what we had in Version 8 and Version 9.

Later on, I will talk about the very good results that we achieved in terms of concurrency when it came to both bind and rebind performance.

So, general message here is CPU elapsed time here for bind and rebind is degraded in all modes of Version 10, but once you actually get post the ENFM process, after the catalog restructure, then we've achieved pretty good results when it comes to concurrent bind and rebind performance.

So, recommendation is, having got to that particular stage here, is to change your procedures so that you do binds and rebinds in parallel concurrently to reduce the overall time that the application has to be downs to do all of the bind and rebind activity.

Another mixed result was in the area of DDL concurrency. Again, we hoped to benefit from the catalog restructure. And in some cases there is some benefit in terms of improved DDL concurrency, but in other cases there are no particular improvements coming from the catalog restructure. Part of the reason for that is that many of you customers have multiple DDL within a single commit scope, and they are actually going after different database descriptors and in different orders. And therefore, there is still potential for deadlock.

When it comes to access path lockdown, when we shipped DB2 Version 10 at GA, we actually disabled this particular piece of functions called APREUSE and APCOMPARE. And basically, this was aimed at a very conservative customer who basically wants to generate a new SQL runtime with BIND REPLACE and REBIND, but at the same time wanted to keep the same access path.

Now, because of some problems here with the underlying infrastructure, related to OPTHINTS, then APREUSE and APCOMPARE where disabled at GA. But I'm pleased to say, and you'll see this in more detail later on in this presentation here, that APREUSE and APCOMPARE have now been re-enabled and have been available through APARs delivered through the service stream.

Now let's turn to slide 8.

Slide 8 (11:18)

So, continuing on the theme of highlights here. Basically if you look at the feedback across the experience, and feedback across the customers in the program, most of that experience was fairly positive. The majority of the customers who participated in the program are going to start their migration to Version 10 in 2011. Some of them it may as much as 9, 12, 15 months to complete the migration to V10 CM across the many DB2 subsystems they've got on their landscape.

From a personal viewpoint, I see incremental improvement in the achievement over Version 8 and Version 9 programs.

When you look across all the customers, there's probably not a single voice or message coming trough across the board. I think you also need to appreciate that when customers are in an early support program, they're having to put in a tremendous amount of effort, and to sustained effort over a 6-month period, and react to and fit in with business and technical priorities. And in an environment, most of our

customers, people and hardware resources are constrained, as is their time.

And I'm particularly grateful for all the customers that worked weekends, at night, and in their own private time, in order to get the projects complete.

There's a significant variation in terms of the customer commitment and achievement. A subset of customers did a very good job in regression and new function testing, and good give back. Other customers basically provided limited qualification about what they where going to do, and also limited qualification about what they where actually able to achieve.

By the end of the Version 10 QPP/beta program, none of the customers where in true business production. And for those of you who are familiar with IMS, you need to appreciate the difference here between a QPP and a beta, versus an early support program. A product like IMS runs an early support program, where all the development and testing is complete by the start of the program. Whereas, when you have a QPP/beta program, as run by DB2, we continue to develop and test in parallel with the program at the customers.

Now, let's turn on to slide 9.

Slide 9 (13:18)

One of the main things about DB2 Version 10 is that there are many opportunities for price/performance improvements. In other words, "cost reduction." It's a major theme of the release, and it's most welcome to all our customers.

However, there's a lot of speculation about what the improvements are, sometimes expressed in a very general sense. Some of my customers are actually intimidated by the marketing noise about those improvements to performance. The expectations of the CIO may have been set to high. For some of the workloads they've evaluated they may not be seeing the improvements in CPU or elapsed time they expected. And they are concerned that they are not able to see that.

Conversely, some customers have seen very big improvements, or big improvements, for certain workloads. So, the performance improvements you see are very much workload-dependent. And for any customer both n the beta program, and also after general availability, one needs to be careful because some small workloads may skew expectations on the savings, either in a very positive, or also in a very negative way. Some of the measurements and quotes that you hear quoted are insanely positive and should be ignored.

But a key question for all of us here is basically, how do you extrapolate and estimate for a production mixed workload?

And this is a very difficult thing to do. We've never been able to do it before with other releases of DB2, and these sort of modelling techniques, with analytical models, really do not deliver the accuracy and the confidence that you'd expect, or you would want, in trying to project forward. And really a serious benchmarking effort is required in order to have a good handle on what the CPU improvements would be for your total production mixed workload.

Now let's turn on to slide 10.

Slide 10 (15:10)

So on slide 10, I'm going to talk here about performance and scalability. And one of the key messages that I'd like you to take a way from this presentation is the need to plan on additional real memory, and to monitor the real memory usage afterword, and to tune based on available real memory on a system.

General estimate here is to plan on a general 10 to 30 percent increase in real memory. It's simply a very rough cut first estimate on the amount of real memory required. For those customers running very small systems with tiny buffer pools, the increase in real memory may be as high as 30 percent. On the other hand those with very large systems with very large DB2 buffer pools, then the increase may tend toward the lower end of this range, maybe 10 percent.

Many traditional OLTP workloads have seen a 5 to 10 percent reduction as soon as conversion mode in Version 10. Some have seen more. Some have seen less. But there are two particular prerequisites in order to be able to maximize the CPU reduction benefits for these OLTP workloads.

First of all, rebinding the packages to generate a new SQL run time, and the second thing is to use the new 1-megabyte real storage page frame sizes. Now, the prerequisite to using 1-megabyte real storage frames, is first of all your buffer pools have to be defined as PGFIX=YES, and the second thing is you have to be running on a configuration here, where you've actually carved out enough memory to support 1-megabyte real storage frames. The 1-megabyte real storage frames are available on the z10 and the z196, and in or-

der to get the most benefit here, you wan to have your long term page fix buffer pools 100 percent backed by these 1-megabyte real storage frames.

But there were some exceptions where customers didn't see the 5 percent CPU saving for OLTP, and like I said earlier, this is related to very light transactions with skinny packages and a few simple SQL. And the bottom line here is that package allocation cost overrode the benefit from the SQL optimizations that we made.

APAR PM31614 helps solve part of this here, by improving, or reducing, the package allocation overhead. Another way of compensating for this, is to make more use of persistent threads with RELEASE(DEALLOCATE).

Now, what do I mean by persistent threads here? I mean things like a CICS protected entry thread or an IMS wait-for-input or pseudo-WFI regions.

Now let's turn on to slide 11.

Slide 11 (17:40)

What I have on the next few slides here, is to basically identify where these performance improvements are actually introduced. For example, on this particular chart here, no rebind is required for the following list of performance improvements, and I will highlight just a few of them, one of which is index list prefetch. What we tried to do in this release is to reduce the requirement to reorganize indexes.

So, one of the things we do now, is we utilize in the list prefetch technology that we introduced for data pages back in Version 2 release 2, and we're now using it to access index leaf pages. We use the information in the non-leaf pages, and then use list prefetch to prefetch the index leaf pages, and therefore better tolerate disorganized indexes.

INSERT index I/O parallelism: if you've got three or more indexes now, on a table, and we're doing insert processing, we will overlap the reads of the data from the indexes there to reduce the overall elapsed time.

We've also introduced a thing called high-performance DBATs, a new type of DBAT which provides thread reuse for distributed threads. And also enables us to honor RELEASE(DEALLOCATE).

Now, let's turn on to slide 12.

Slide 12 (19:02)

Now we have some performance improvements here, which require a rebind first. And one of those is if you're going to switch to using RELEASE(DEALLOCATE), and basically achieve the benefit when using persistent threads and RELEASE(DEALLOCATE) to improve performance.

If you want early evaluation of residual predicates, it also requires a rebind.

There are a number of improvements in the area of IN-list processing, with a new access method. We also have a

thing called SQL pagination, again which is a new access method.

One of the things that became very useful during the early support program was the use of index include columns. You may have a situation today where you have one index on let's say columns A and B for uniqueness, and then you another index on A, B, X, Y, and Z columns, and that extra index is to improve performance.

What you can now do in this version of DB2 is have a unique index with include columns. So, therefore you can say, "I want to have unique index on columns A and B, but also include columns X, Y, and Z." And this single index can be used to still enforce uniqueness constraint, but also can be used for performance. So, this means that we can we can reduce the number of indexes, which obviously is DASD savings, but also more importantly than that, this is one less index to be maintained during insert and delete processing.

Now, one point that I'll go into in a lot more detail later, is about executing RUNSTATS before rebind. Certainly, if you are a customer who is coming from Version 8, directly to Version 10, it is very important to collect the improved index statistics before you actually do the rebind of static SQL. So, this is referring to an improvement that we made in Version 9, where we changed the calculation, or the algorithm, for cluster ratio, and also introduced a new statistic called the data repeat factor.

So, strong recommendation here is to collect these new statistics in Version 10 before you actually do the rebind of static SQL.

Now, if you're coming from Version 9 to Version 10, and

you're not already using KEYCARD as a default option on your RUNSTATS, then again, the recommendation is to run RUNSTATS with KEYCARD before you actually rebind under Version 10.

I'd like to point out now that KEYCARD is implicitly on in Version 10 RUNSTATS, and in fact it does not even matter whether you specify the KEYCARD on the RUNSTATS control statements. You'll always run the RUNSTATS in Version 10 and collect the KEYCARD statistics.

Now let's turn to slide 13.

Slide 13 (21:40)

Now, I want to go into a bit more detail here now, about high performance DBATs. This is a new type of distributed thread. And in order to benefit from high performance DBATs, you must set the systems parameter or ZPARM CMTSTAT=INACTIVE so that threads can be pooled and reused.

The key to the door is that one or more packages on the connection need to be bound with RELEASE(DEALLOCATE). And the second prerequisite is that you must have issued a MODIFY command to DDF to basically honor the BNDOPT option that was specified on the package.

So, to repeat myself there, the two keys to the door here are to have one or more packages bound with RELEASE(DEALLOCATE), and to have done the MODIFY to DDF with the command on this chart here where you specify

PKGREL(BNDOPT), so that way DB2 DDF will honor the bind option.

So, let's now drill down into some detail. When a DBAT can be pooled is at the end of a client's unit of work. Now, what's different about a high performance DBAT, is that the database access for the DBAT and the client connection will remain active together. We still cut the accounting record and end the enclave at the end of the unit of work. But basically the DBAT and the client connection stay together.

Without a high performance DBAT, then what happens is the client connection is parked and the DBAT is pooled, or put back in the pool, for use by another connection. So the key point is with a high performance DBAT, the DBAT and the client connection remain active together.

Also after the high performance DBAT has been reused 200 times, the high performance DBAT is purged and the client connection will again go fully inactive as before. All the interactions with the client will still be the same, in that if the client is part of a sysplex workload balancing setup, it will still receive indications that the connection can be multiplexed across many different client connections.

The idle thread timeout parameter IDTHTOIN will not apply if the high performance DBAT is waiting for the next client unit of work. If the High performance DBAT has not received any new work for the period of time defined by POOLINAC, then again, the DBAT will be purged, and the connection will go inactive.

If the number or percentage of the high performance DBATs exceeds 50 percent of the MAXDBAT threshold then the DBATs will be pooled at commit and the and the package

resources copied or allocated as if they where RELEASE(DEALLOCATE). So, basically up to 50 percent of your DBAT threshold can be allocated as high performance DBATs.

One important feature of the new DB2 support for high performance DBATs is that these DBATs can be purged to allow DDL, bind, and utilities to break in. And this is achieved again with the MODIFY DDF command. You can actually switch from, basically PKGREL(BINDOPT) over to PKGREL(COMMIT). When you say MODIFY DDF PKGREL(COMMIT), this forces all the packages to start executing with COMMIT, and thereby allow the DDL, and the bind, and the utilities too break in.

So, the sequence of events here would be to do, under normal operation to be running with MODIFY DDF PKGREL(BINDOPT) but before doing the DDL, the bind etcetera, you'd switch to COMMIT, then run your DDL, bind, or utilities, and when that's finish switch back to PKGREL(BINDOPT).

Now let's turn to slide 14.

Slide 14 (25:19)

I want to make a few comments about customer measurements before I show some results with you. First of all, customer measurements are not always consistent and repeatable. And part of the reason for that is that in real customer environments, you're not able to run with dedicated resources: dedicated CPU, dedicated storage, dedicated DASD. So, it's very important in the customer environment to

run the measurements several times to make sure you have some level of consistency or repeatability, and to make sure the results you're not achieving are wild.

In many cases in customer environments, you see a wide variation in terms of measurement noise, especially in terms of elapsed time.

Many cases, customers are only running a subset of the production workload, and a key question is whether that workload is representative of the total mixed workload that the customer runs.

Sometimes use of synthetic workload is used to study specific enhancements, and this can lead to very optimistic or pessimistic results. In some cases we don't trust some of the very large numbers on CPU reduction, and especially some of the elapsed time savings.

So, general recommendation to all the customers that listen to this web lecture here, is that customers should not spend the CPU savings until they've actually seen them in production first.

Now let's turn to slide 15.

Slide 15 (26:38)

On slide 15 here, are some examples of some performance improvements that customers were able to see during the beta program. If we look at the first three rows here, these were customers who ran CICS online transaction workloads and saw performance improvements in line with our aggres-

sive objective of 5 to 10 percent improvement. In the first case here, the customer saw 7 percent CPU reduction in Version 10 CM mode after rebind. And they saw an additional 6 percent after this define the buffer pools as long-term page fixed and used the 1-megabyte page frames.

Now notice carefully what I said there. This customer previously was not using long-term page fix buffer pools. Long-term page fix buffer pools was introduced back in Version 8 in order to help reduce CPU regression with Version 8. So, when I say the customer got a 6 percent performance improvement over and above the 7 percent, it was from the combination of using long-term page fix, which was introduced in Version 8, plus using the 1-megabyte real page frames that came along in Version 10.

Another customer saw 10 percent CPU reduction after migration to Version 10 from Version 9. And a third customer saw 5 percent.

However, we did see a few cases where customers saw 5 to 10 percent CPU degradation, and as repeated earlier, this is where we have very simple SQL with very skinny packages.

Customers using high performance DBATs, some of them, saw fairly good reduction in CPU and also in elapsed time. Customers with heavy concurrent insert in data sharing have seen some very generous improvements.

Now let’s turn to slide 16.

Slide 16 (28:25)

On slide 16 are some very special cases, or some niche improvements in Version 10.

The first one here is multi-row insert, OK? And this is a particular workload, a synthetic workload, that the customer built and used previously under Version 8 and Version 9, to do what is called “log record sequence number contention.” Going back to Version 8, previously, every log record on a member of a data sharing group, had to have a unique log record sequence number. And bascially, the clock only moved forward every 16 microseconds because the LRSN is the higher six bytes of the store clock.

So, we made some improvements in Version 9, to reduce the number of times we had to spin, and to hold the log latch for a shorter period of time, and in some case not to hold the latch at all. But there are some further improvements in Version 10.

And what you can see, even after the version 9 improvements, this customer was able to see a 33 percent CPU reduction compared to Version 9, and a four-times improvement compared to Version 8. But be careful here, this is a particular niche case here, and a particular workload that was used to magnify a particular problem, and to study it.

Another customer did a special case study where they had multiple indexes on a table, and they did and insert-intensive workload, and they saw the benefit of the parallel index update, the parallel read. Other customers saw big improvements from inline LOBs and include indexes.

So, I won’t read the number out here, but you can see that these are particular niche workloads here, but there are some particularly good number achieved. And the real ques-

tion is how much does these mean in the overall sense? And you do need to be careful, because these sort of improvements are only going to apply to a small part of your workload.

Now let's turn to slide 17

Slide 17 (30:15)

I now want to drill down here about the 1-megabyte real storage page frames that are available on the z10 and z196 processor. I get a lot of questions coming in about this particular area through previous webcasts and presentations at conferences. So, what's the objective of us using the 1-megabyte real storage page frames?

The benefit here is to reduce CPU through less TLB misses: translation look-aside buffer misses. This is where virtual address get translated in to real storage addresses. So, this is very different from long-term page fix. Long-term page fix was introduced in Version 8, and what long-term page fix on the buffer pools tries to do is to avoid the repetitive cost of page fix and page free around per-page when you have to an I/O. But one thing here is, if you want to use the 1 megabyte real switch page frames, your buffer pools must be defined as PGFIX=YES, long-term page fix.

But, there are different objectives here. As I said, the long-term page fix benefit is directly related to I/O-intensity, so if you have no I/Os going on, there's no benefit from long-term page fix, and if you have lots of I/Os going on, then you have the potential to get some pretty good benefit from long-term

page fix. Whereas, the use of the 1-megabyte real storage page frames is basically trying to reduce TLB misses.

Now, if I look around my social circle, many customers so far have been reluctant to use PGFIX=YES that came in with Version 8, because they are concerned about the amount of real storage. Why could that be?

Well, they may be running to close to the edge of the amount of real storage, but once that they do understand the benefit of long term page fix, either themselves, or because of pressure from the z/OS systems programmers, they're not prepared to commit to using long-term page fix.

So, you may have to reconsider whether or not you want to use long-term page fix under Version 10, to also be able to get the benefit of the 1-megabyte real storage page frames.

And if you are concerned about the amount of real memory, then what you might consider doing is actually reducing the size of the buffer pool in order to generate some relief on the amount of real memory and to make some available in order to use the 1-megabyte real storage page frames.

Note: long-term page fix of buffer pools is a long-term decision. When you make the switch to go PGFIX=YES or PGFIX=NO the actual change goes pending, and the buffer pools need to go through reallocation in order to change the attribute. And whilst in some cases you could achieve this by setting the VPSIZE to 0 and then increasing it back to its former value, in most cases people will have to re-cycle DB2 in order to change the attribute.

A lot of customers are concerned about the cost of real storage on their System z systems. One thing I'd like to adver-

tise here, is a reduction in the cost of real storage on the z196 processor. I understand there is a 75 percent cost reduction, and it's roughly about 1.5K per gig versus about 6K per gig that you used to have on the earlier releases.

Now let's turn to slide 18.

Slide 18 (33:38)

I'm now going to give some more information here about the 1-megabyte real storage page frames. And the principle here, is that the systems programmer must partition their real storage between the existing type of 4K frames and the 1-meg' frames. So how is this done?

It's basically done at IPL time. There's a parameter called LFAREA in the IESYSnn parmlib member, which specifies what percentage of the real memory should be configured as 1-megabyte frames. So it's LFAREA percentage that determines the amount of storage that is reserved for the 1-meg' frames, and to repeat myself here, this is only changeable via an IPL.

So, another common question that comes in is, "how much 1-megabyte page frames do I need?" And the answer is, add up the some of all your buffer pools that are defined as long-term page fix, and maybe another 10 to 20 percent on top of that for some growth and tuning.

One thing that I would say is that we did identify a number of critical problems during the Version 10 QPP beta program related to the z/OS support for 1-megabyte real storage page frames. So make sure that you have the critical z/OS

corrective and preventive service applied before you start using the 1-megabyte page frames.

What we've observed across customers in terms of benefit unique to the 1-megabyte real storage page frames is basically between 0 and 6 percent.

Some of the customers have a requirement for us to have a new parameter to separate the use of PGFIX=YES from the use of the 1-megabyte page size. And we plan to address this in a future release of DB2.

Now let's turn to slide 19.

Slide 19 (35:31)

Now, I'd like to switch gears on this topic of performance and scalability and talk about the virtual storage constraint relief below the bar in the DBM1 address space.

This virtual storage constraint relief, which his much needed, is available as soon as conversion mode. And to maximize the benefit, you need to rebind the static SQL packages to get some maximum benefit. The result that we've achieved with many programs during the QPP/beta program has been very significant, and there is a high degree of confidence that the problem has been solved.

Now we can now support up to 20,000 active threads, but let's talk about it from a real-world perspective. There are many customers today in my social circle, and they support about maybe 500 threads per DB2 subsystem or DB2 member of a data sharing group. And I'm quite confident now in

Version 10 that we can scale maybe up to 2 and a half thousand, 3,000 threads, quite comfortably and that will make a huge difference in terms of the impact on our customers.

So, what are the limiting factors on vertical scalability in a post-V10 environment, in terms of the number of threads and the thread storage foot print?

Well, the number one thing is that amount of real storage provisioned. And that's a key factor now in terms of how much is provisioned that a customer needs to monitor it and tune and capacity-plan from the right amount of real storage. Secondary consideration is the amount of ESQA and ECSA 31-bit storage on the LPAR, and finally active log write.

Now let's turn to slide 20.

Slide 20 (37:10)

On slide 20 here, is an example of a problem that we found during the QPP/beta program, and which we fixed before GA for Version 10. So, I earlier said on one of my earlier slides, that I wanted customers to test in the order and the way they would do things in a real production environment.

So, this graph here is taken from one of the customers, who I worked very closely with during the program. And basically there are three columns here: one related to Version 9, Version 10 CM without the rebind, and then Version 10 CM with the rebind. And what it's showing here is the thread foot print below the 2-gigabyte bar in each of these modes.

So, we're looking at the Version 9 column here, the thread storage below the bar is between 3 and 3-and-a-half megabytes. And it's basically made up of the EDM pool and also the agent local user, plus some allowance here for EDM pool fragmentation. Now what this shows here in the second column is the result of Version 10 conversion mode, without the rebind. And what you can now see is that the thread storage foot print has gone up to between 3-and-a-half and 4 meg'. And this is a very unhealthy situation, because most customers, when they migrate to a new release, will not rebind all their plans and packages straight-away. So, therefore this is a regression on the virtual storage utilization below the bar, and this could've killed many customers.

And the third column shows what happens after the rebind in Version 10 CM. And what you can now see is the thread storage footprint is much much smaller, and you can see the very generous virtual storage constraint relief.

So, repeat that again, the chart here in the middle column which shows these bad results from Version 10 CM was during the program before general availability.

Now let's look after the solution.

Slide 22 (39:12)

So again now on this chart here we have our three columns on a different scale. Now we have a workload here that in Version 9 we're using about 2 to 2-and-a-half meg' per thread, and then you can see Version 10 CM mode without the rebind, and without the fix, and you can see that thread storage footprint is between 2-and-a-half and 4 meg'. But

now look at what happens after the fix. If you look at that third column there, effectively columns 3 and 4 here, you can now see Version 10 CM mode without the rebind but with the fix on, and you can see that we deliver some virtual storage constraint relief even without the rebind in Version 10 CM mode.

But if you do want to maximize the benefit in terms of virtual storage constraint relief, then the recommendation is to re-bind your static SQL packages in CM mode.

Now let's turn to slide 22.

Slide 22 (40:04)

This picture here just really shows you the trends and summarizes the trends over the versions, starting with Version 8 going all the way through to Version 8, Version 9, and Version 10.

You can see back in Version 7 here that we provided some virtual storage constraint relief, by basically using data space buffer pools.

Then in Version [8] we moved the buffer pools above the bar, castout buffers, compression dictionaries, global dynamic statement cache, and so on.

And then in Version 9, what we did was move some DDF control blocks above the bar. We moved 100 percent of the skeleton cursor tables and package tables up there, and also portions of the CTs and PTs.

But finally you can see here in Version 10, that not only did we keep the things that were there before above the bar, what we've now done is moved a huge portion of the SQL run time above the bar, and that has provided us very generous relief.

So, now in Version 10, theoretically it's possible to go up to 20,000 active threads. But in most cases, it's probably going to be a few thousands, compared to a few hundreds that we were limited to in Version 8 and Version 9.

Now let's turn to slide 23.

Slide 23 (41:11)

So, on one hand here, the virtual storage constraint relief achieved now can provide very generous scalability improvement, it also provides opportunities for improved price/performance. Many customers in my social circle had been on one I called the "JC diet plan," where previously in order to generate storage constraint relief, they had to basically give up CPU cycles. So, for example, customers stopped used RELEASE(DEALLOCATE) and maybe stopped using persistent threads like using CICS protected entry threads.

Well, now, assuming you have enough real memory, one can reverse the equation, and actually trade additional real memory, in order to reduce CPU.

Now when I talk about additional real storage here, I'm talking about additional real storage requirement over and above the 10 to 30 percent that I already mentioned.

So, anyway, let's take some examples here of how this can be achieved. If you've got the real memory, what could you do to use this real memory and reduce your transaction cost? Well, in the area of CICS for example you could make more use of protected entry threads. In the case of IMS, make more use of, basically, wait-for-input regions, and pseudo-WFI regions.

Now, the benefit from using thread reuse and using these protected threads, or persistent threads, this avoids the repetitive cost of creating and terminating a thread.

And this probably, you know, is relatively small, but it is an improvement. But it becomes the key to the door of using RELEASE(DEALLOCATE). If you're not using persistent threads, then having a bind parameter of COMMIT or DEALLOCATE makes no difference whatsoever. It is only when RELEASE(DEALLOCATE) is used in combination with persistent threads, that the opportunity opens up, and therefore avoid the repetitive cost, not just of creating and terminating the thread, but of also allocating and freeing of storage, of acquiring parent locks, and so on.

The second opportunity now, is related to DDF. Now for the first time in Version 10, we have this concept of high performance DBATs, and we start respecting the release parameter RELEASE(DEALLOCATE).

One thing I would like to say here is, "Be wary here of a one-size-fits-all strategy." Because if you decide to use RELEASE(DEALLOCATE) in all of your packages, this could very seriously drive up your real storage requirement, and also drive up a huge portion of your DBAT requirement, and therefore run into the MAXDBAT threshold.

So, what I would suggest you do here, is just like what you've done previously in the CICS and the IMS environment, that you focus the use of high performance DBATs on basically the high-volume transactions.

So, I can use an example here based on SQLJ and JDBC. What you probably want to do is take your packages and bind them twice. First of all bind them into the first collection with RELEASE(COMMIT) and then into the second collection with RELEAE(DEALLOCATE). And so from an application perspective, you want the applications which are high performance, and where you want to benefit from the high performance DBATs with RELEASE(DEALLOCATE), you want them to connect to a data source which points to the collection which uses RELEASE(DEALLOCATE) and what you want the other applications to continue using RELEASE(COMMIT), and the way that works is they commit, or connect I should say, to the alternative data source that points to the collection where the packages were bound with RELEASE(COMMIT).

So, when it comes to the default JDBC and ODBC packages, this is what you want to do. You want to bind them into two separate collections and let the applications connect to the appropriate collection. Or should I say, "the appropriate data source that points to the specific connection."

Now let's turn to slide 24.

Slide 24 (45:07)

Now, one thing I want to make very clear here is more use of persistent threads with RELEASE(DEALLOCATE) is a trade off with bind, rebind, and DDL concurrency. One of the advantages of the old world of giving up on persisted threads, and not using RELEASE(DEALLOCATE) was it made it a lot easier to break in to do binds and rebinds and do things like DDL, adding new indexes. So, there will be a trade off with using more persistent threads with RELEASE(DEALLOCATE).

A common question I get is customers say they cannot find the cost of the thread create and terminate, or the benefit of actually avoiding it. An important point here is the cost of thread create and terminate is outside the DB2 accounting interval. So, both the cost of thread create and terminate, or the savings by avoiding it, are not apparent when you look into the DB2 accounting record.

So, basically you need to look in the CICS accounting record. CICS uses the L8 TCB, to access DB2, irrespective of whether the application is thread-safe or not. So, the cost of thread create and terminate is clocked against the L8 TCB, and you can find that in the CICS SMF type 110 record.

Note: prior to OTE, this cost of thread create and terminate was not captured even in the CICS 110 record, and it was basically un-captured CPU.

Now, one you use RELEASE(DEALLOCATE) in conjunction with the persistent thread, then the benefit of using RELEASE(DEALLOCATE) will actually show up in the DB2 accounting record.

So, to repeat myself here, if you decide to use more persistent threads with CICS, and you decide to use RE-

LEASE(DEALLOCATE), then the benefit of RELEASE(DEALLOCATE) on the persistent threads will show up in the DB2 accounting record. But the cost, or the avoidance, of the thread create and terminate will show up in the CICS accounting record.

Please do be aware however, that when you do use RELEASEE(DEALLOCATE) some locks will be held beyond the commit scope and held until thread termination. So, for example a mass-delete lock where you've got an SQL DELETE without a WHERE clause, or a gross-level lock acquired on behalf of an SQL LOCK TABLE request.

Note: this is no longer a problem for gross-level lock acquired by lock escalation.

So, basically when you have transactions where you want to use persistent threads with RELEASE(DEALLOCATE) you need to avoid mass-deletes, and you also need to avoid use of the SQL LOCK TABLE request.

Now, let's turn to slide 25.

Slide 25 (47:40)

On slide 25 here, is an example of what we were able to do with our IRWW. So, we have an IMS version of the IRWW workload that we tried in data sharing. And what you see here, on this graph here, is various measurements we've made. So, the base measurement here is based on DB2 Version 9 new function mode, with a rebind where you are already using plan management, or rather the EXTENDED version of PLANMGMT. The next measurement point was to

go from Version 9 NFM to Version 10 CM mode, without the rebind. The third point, was basically to have Version 10 CM mode with a rebind, and then finally we go for NFM, and the very last point is then combine that with RELEASE(DEALLOCATE).

So, what you can see on the graph here, is that in Version 10 CM mode, without the rebind, there was seen a 1.3 percent reduction in CPU relative to Version 9. It's a pretty small improvement. But then, when you do the rebind in CM mode, we can see a 4.8 percent CPU reduction. When we repeat the exercise in NFM and rebind there, then there's a very tiny improvement there beyond the 4.8. We've now got a 5.1 percent CPU reduction. But the next big step up in terms of reduced CPU comes with rebinding with RELEASE(DEALLOCATE).

So, measurement points 2 and 3 on this chart here were done with RELEASE(COMMIT). So we had thread reuse with RELEASE(COMMIT), but the last measurement in point 4 is when we combine RELEASE(DEALLOCATE) with the thread reuse, and then we get the second step up, in terms of CPU reduction.

So, this is just an example with the IBM Relational Warehousing Workload, and results will vary in your installation.

Now, let's turn to slide 26.

Slide 26 (49:39)

Now other potential benefits here, as a result of the virtual storage constraint relief, is the potential to reduce the num-

ber of DB2 subsystems. Today many customers have multiple DB2 members running on the same LPAR. And what I am talking about here is multiple members of the same data sharing group. And the reason why this was done was they needed additional members to handle the workload, because of the previous virtual storage constraint. So now they have the ability now to actually collapse the number of systems, and therefore reduce the total number of DB2 subsystems, and specifically the number of DB2 members running on a single LPAR.

And this may result further in the ability to reduce the number of LPARs. One of my customers who today is running a 12-way data sharing group in a post Version 10 environment are planning to reduce the size of that data sharing group down to 5 members. And they will be able to get savings from reducing the number of members, but also will be able to reduce the number of LPARs.

When you do increase the amount of workload going through an individual DB2 member, you need to consider the potential here for the increase in the logging rate per DB2 member. And if you decide to reduce the number of LPARs, you need to consider the increase in the SMF data volume per LPAR.

So, there are some things that can help you. So, one of the things that we've provided in DB2 Version 10 is the DB2 compression of the SMF data, to reduce the SMF data volume. And we've had some pretty good experience here at the lab here by using this DB2 compression of the SMF data, and we've seen about a 70 to 80 percent reduction in the SMF data volume related to the DB2 accounting records. And the CPU cost of doing this is fairly tiny, at about approximately 1 percent.

Now, you may want to consider the use of accounting rollup in Version 10 for DDF and also RRS work. Now, I'm not a big fan of accounting rollup prior to Version 10 because it compromises performance problem determination and problem source identification, because when you have accounting rollup, you lose information on the outlying transactions.

Now what you may decide to do now in Version 10, because you've actually got the SMF compression, maybe it would be better to turn on SMF compression and give up on the accounting rollup. In that way you still get reduction in the SMF data volume, but you don't lose the outlying transactions.

I'd also like to point out, that even if you are using accounting rollup, there are some significant improvements in package level accounting in Version 10 as well.

Another thing to say about real storage is that if you start increasing the size of an individual DB2 subsystem in terms of the workload going through it, you need to consider the increased storage for DUMPSRV, and you need to re-evaluate your MAXSPACE setting as well.

Now, there's potential here that people may say that because of these sorts of savings in being able to reduce the number of members in a data sharing group that they may decide to either give up on data sharing, or reduce the number of members, and actually compromise the availability.

I'd just like to point out here that data sharing is a major technology here to support the avoidance of planned outages. And it's a key technology to avoid humongous single points of failure. So, there's still continued value in using data sharing going forward. And, as before, we recommend

a minimum of 4-way data sharing assuming you have a two-box or 2-processor configuration. The idea of having a 4-way configuration is that you can have 2 LPARs per processor and a DB2 on each of those LPARs, and that's where the 4-way comes from.

And the idea here of having 4-way is that if you lose an LPAR or lose a DB2 member, the surviving DB2 members on the LPARs can take over 100 percent of the CPU capacity.

Now let's turn to slide 27.

Slide 27 (53:39)

Another common question that comes in from customers is about the 3 large areas that are allocated at IPL time for 64-bit storage. So, one thing to indicate or discuss about this is that these larger areas that are allocated here… We're not really allocating these areas. What we're actually doing is reserving the virtual storage.

So, we have the common storage area which is the z/OS default, is now 6 gigabytes. And when you have the common area, this is a bit like ECSA, and it's addressable by all authorized programs running on the LPAR. We in DB2 use it for IFC for the accounting. The private area is 1 terabyte. And in that private area, mainly the DB2 buffer pools, but also things like XML and LOB data are huge users of the area with things like RTS blocks, trace buffers, and so on.

We also have the shared private area of 128 gigabytes. A shared private area is addressable by all the authorized

processes which have registered their interest to z/OS using the unique object token created when the memory object was created. And DB2 Version 9 introduced this 64-bit shared private storage, but in Version 10 almost all of the DB2 storage is now 64-bit shared private.

So, to repeat again, don't be scared by these very large numbers. All simply DB2 is doing is reserving virtual storage. It does not mean that it is being used. So these areas do not have to be backed by real storage. That storage is not even being allocated from a virtual storage perspective. We're just simply reserving a range.

In some respects this is a bit of a lazy design, but it does make it a lot easier from a design perspective, in terms of not having to cater to serialization when trying to extend. So it's very much a high performance design.

So, these storage areas only need to be backed by real storage when they are allocated within the large reference area.

Now let's turn to slide 28.

Slide 28 (55:41)

A major theme about this presentation is about the need to carefully plan, provision, and monitor real storage consumption. So, first of all, let me talk about, or share with you, about a parameter called SPRMRSMX. I call this the "real storage kill switch."

This was available prior to DB2 Version 10, and it was only used by a small number of customers who were running multiple DB2 subsystems from the same or different groups on an LPAR, and basically it was a safety switch. It was an idea that if you had a runaway DB2 subsystem that was allocating more and more storage, and particularly consume all the real storage, and consume all the aux storage, you then take the LPAR down.

So the idea of this real storage kill switch was basically to provide a safety device. And the idea was you'd have a rough estimate as to the normal working set size of a DB2 subsystem, multiplied by 2, and then you set the ZPARM. And what would happen is when this value for SPRMRSMX was reached, then that DB2 subsystem would be cancelled, cancelled out. And this is way of protecting the overall availability of the LPAR and other subsystems and DB2 subsystems that are running on there.

With the advent of Version 10, customers using the real storage kill switch will need to re-evaluate that value and factor in the amount of storage being used in 64-bit shared and 64-bit common storage.

Now, with the advent of Version 10, DB2 is making much more use of 64-bit storage, and there are some problems that needed to be solved. One is that you needed the ability to monitor real and auxiliary storage usage for 64-bit storage.

Now, in version 9 this wasn't a big issue because the use of 64-bit shared was fairly limited. But now, Version 10 makes extensive use of 64-bit shared. Also, the buckets we reported previously for real and aux usage were LPAR level instrumentation buckets. If you had more than one DB2 sub-

system running on the LPAR, these numbers were actually inaccurate, and you were not able to isolate down and find out how much real and auxiliary storage was being used by the individual DB2 subsystems. And the only way to get a reliable number was to have one DB2 subsystem per LPAR.

And then there was also a lack of monitoring for what is called an ENF 55 condition. This is when more than 50 percent of the auxiliary storage is usage.

So, as I said, these are some of the problems or challenges introduced with DB2 Version 10, and the question is, “what are we going to do about it?”

Well, if you now turn to slide 29…

Slide 29 (58:30)

I want to advertise a new function APAR PM24723 that’s come out for DB2 Version 10, after GA. And this is very important, OK? It addresses the monitoring issue, and it also provides a new extension to IFCID 225 for storage monitoring. The prerequisite for the monitoring piece is the MVS APAR OA35885. This MVS APAR provides a new callable service to real storage manager that is used by DB2, for example, to provide real and auxiliary storage use for an addressing range for any shared object.

Secondly, with this APAR, the hidden ZPARM for the real storage kill switch has now become opaque. It is called REALSTORAGE_MAX.

But another major thing that was introduced with this APAR was a thing called DISCARD mode. And it's controlled by a new ZPARM called REALSTORAGE_MANAGEMENT. And this controls when DB2 will discard or free storage frames back to z/OS real storage manager. There are three values for this ZPARM: ON, OFF, and AUTO. And AUTO being the default, and strongly recommended.

When you have REALSTORAGE_MANAGEMENT set to ON, then DB2 will discard unused frames back to real storage manager all the time. So it will discard stack, thread storage and… Basically what it's trying to do is keep the storage footprint small.

On the other hand, if you turn it to OFF, then you basically turn off discard processing. However, when things do get really out of hand, then we'll still go into discard mode processing.

Now what then happens with AUTO, which is the default, is that basically this discard mode processing will occur when heavy paging occurs and is imminent, and we try to reduce the frame count to avoid system paging. So, it's a bit like a thermostat on your central heating system or air-conditioning system. It'll cut in when there's a problem, and then when the condition is relieved, it will turn off discard mode processing. With AUTO, which is the default, DB2 monitors paging rates, switches between ON and OFF, and decides when to discard frames based on reaching 80 percent of the real storage kill switch, 50 percent of aux used, and when we have a condition in z/OS called AVQLOW "average q low." This is a count of the available real storage frames.

When actually the discard mode processing is operational, there are some new messages: DSNV516I and 517I, when the paging rates cause DB2 to free real frames.

So, strong recommendation for true business production use is to apply the APAR PM24723 and the prerequisite MVS APAR OA35885.

Now let's turn to slide 30.

Slide 30 (1:01:29)

I now want to switch tacks onto high INSERT performance.

As I said as part of my introductory comments, one of the things that we're trying to do here is provide performance improvements for universal table space, and we want the INSERT performance for UTS to be equal or better compared to classic partitioned.

And significant changes have occurred in DB2 version 10. First of all, universal table space now supports MEMBER CLUSTER, and there is the ability to turn it both on and off. There have also been some changes to the space search algorithm, so it's now somewhat similar to classic partitioned. So the goal was for UTS to be equal or better than classic partitioned. In absolute terms, we're not there yet, but there have been significant improvements, and we're much closer.

Now, how close we are, or how good we are, is very workload dependent. Some workloads are definitely better, a few of them are actually worse. INSERT performance in general is still a trade off between space reuse versus throughput

and reduced contention. So, the one are area we are still a bit weak on is when you have UTS PBR/PBG with row-level locking and sequential insert. And I'll be talking through some charts here, which will illustrate the benefits, and also illustrate this weak point as well.

Now let's turn to slide 31.

Slide 31 (1:02:50)

Now, let's talk about some general INSERT performance improvements across all the table space types.

The first one is reduced LRSN spin for inserts to the same page. This works very well when you use multi-row insert, and also when you have simple insert within a loop in a data sharing environment. So, now we no longer have to have a unique LRSN for each of these log records.

There's also an optimization for pocket sequential. Sometimes this is referred to as "spot sequential," and the change here is that index manager now picks the candidate RID during sequential insert, to be the next lowest key RID value. Previously, it was the next highest key RID value. The end result of this is a high chance to find space and avoid a space search.

And the third improvement is parallel index I/O. It works very well when activated for random key inserts. It occurs when you have three or more indexes, and to compensate for the potential increased CPU cost of having parallel index I/O, then all prefetch and deferred writes are now eligible for zIIP offload. So, this is a compensation feature.

Now, let’s turn to slide 32.

Slide 32 (1:04:02)

So, what I want to do here is talk you through some insert measurements that we’ve done at the DB2 lab. These are based on two-way data sharing. They are based on a database schema where we have three tables, with a total six indexes (4 unique, 2 non-unique, 2 secondary indexes) and we test various table space types: partitioned, segmented, UTS PBR/PBG.

And basically the workload issues SQL INSERTs. These INSERTS may have 5, 9, or 46 columns of various column data types. For interest purposes the application is implemented in Java, and we have two general types of the workload. We have sequential insert into empty tables, we have up to 240 concurrent threads and we use multi-row insert with a row-set size of 100. And the other type of test we have is random inserts into populated tables, where we have single-row inserts and have up to 200 concurrent threads.

Now let’s turn on to slide 33.

Slide 33 (1:05:03)

Now what we have on this slide 33 here is a set of measurements related to range-defined table spaces. So what we’re doing here is comparing like with like. We’re compar-

ing classic partitioned table spaces with UTS PBR with or without MEMBER CLUSTER.

Now, the top half of the chart here covers random inserts, and the second half, or bottom half, of the chart covers sequential inserts. So let's just look at the top half of the chart here about random inserts. On the left-hand side is the throughput. On the right-hand side is the CPU time. The columns in red are showing page-level locking, and the columns in blue are showing row-level locking.

So, if you now look on that top left-hand side here, the throughput: on the Y-axis we have rows per second, and on the X-axis we have PTS with and without MEMBER CLUSTER, and we have PBR with and without MEMBER CLUSTER. And what you can see here now is that PBR competes very favourably now with PTS. In terms of throughput here, they are very much the same. And there is also not a big gap between page and row-level locking, OK? And there doesn't seem to be a lot of big difference here between with and without using MEMBER CLUSTER.

When we go to the right-hand size of that chart here, and we look at CPU time, again now you can see that the CPU time is very much same for PBR with and without MEMBER CLUSTER versus PTS.

So the top half of this chart here for random inserts is extremely positive and not a lot of difference between page- and row-level locking.

Now let's come to the bottom half of the chart. So again, let's look at throughput. We can see here now if we look at page-level locking to start with now, that in terms of throughput here PBR now, with or without MEMBER CLUSTER com-

petes very favourably with page-level locking with PTS. However, when you use row-level locking with PBR, and without using MEMBER CLUSTER, then there is a very significant difference with row-level locking between using MEMBER CLUSTER and not using MEMBER CLUSTER. When we are able to use MEMBER CLUSTER with row-level locking on PBR then the throughput is significantly increased.

If you look on the bottom right hand side of the chart for sequential inserts, we see CPU time. And what you can see here with sequential inserts is the CPU time per commit is significantly lower here with page-level locking versus row-level locking. And this is not a surprise when using… the key difference between page-level locking and row-level locking is when you have sequential INSERTs, UPDATEs, and DELETEs.

Now let’s turn to slide 34.

Slide 34 (1:07:57)

On slide 34 we have a similar sort of chart, but what’s different about this chart is non-range-defined table space. Again trying to compare like with like. So what we’re doing here is comparing segmented table spaces where we don’t have MEMBER CLUSTER and don’t have MEMBER CLUSTER in Version 10 versus PBG.

And PBG in Version 10 now supports MEMBER CLUSTER. So, the top half of the chart is related to random inserts and the bottom half of the chart is related to sequential inserts.

So let’s talk about the top half of the chart first, related to random inserts. So, top left-hand side talks about throughput. And if you now look at the throughput with page-level locking, you see the highest throughput is now achieved with PBG with and without MEMBER CLUSTER, and the very best result is PBG with MEMBER CLUSTER. If we use row-level locking, then the best result is with PBG and MEMBER CLUSTER.

If we look at the CPU time, OK? Then the best result is achieved here now in this case with UTS PBG. And in fact with row-level locking it comes down still further. So again, a good set of results here for random inserts.

Now let’s look at the bottom half of the chart and look at sequential inserts. So first of all let’s look at throughput in the bottom left-hand side. And now we can see that the level of throughput is much better, or simply better with PBG. It’s equal or better than what we had with segmented.

Now however, when you look at row-level locking, then basically the results with row-level locking are poor with both segmented and also PBG without member cluster. Once you actually use row-level locking with MEMBER CLUSTER on PBG then you get pretty good results on sequential inserts.

On the bottom right-hand side we see the CPU time for sequential inserts. And obviously, the lower the number the better. And you can see in terms of CPU time the best results here are used with page-level locking.

So, this completes part 1 of this web lecture on DB2 Version 10 migration planning and very early experiences. Thank you for listening to this web lecture.

(1:10:14)