DB2 Data Management Software

A Technical Discussion of Multi Version Read Consistency

IBM Software Group
Toronto Laboratory

Contents

Introduction

There has been a lot of debate in recent years regarding the various RDBMS implementations of concurrency models. Oracle claims that because readers don't block writers and writers don't block readers that they have a better solution for concurrency and that applications run better on Oracle[1]. The reality is that with Oracle, as with any other database, you design and code your application with an understanding of the underlying isolation and concurrency model. DB2 implements the ANSI standard isolation levels (RR, RS, CS and UR). No other database vendor has implemented Oracle's Multi Version Read Consistency isolation nor has it proven to be a performance advantage in industry standard, ISV or real life customer benchmarks. Simply stated; Oracle is taking an old architectural decision and trying to showcase it as a differentiator, when in fact it is simply a concurrency model that developers must code around and one that adds an extra burden of management on the DBA as described below.

How Multi Version Read Consistency Works

In Oracle, when a block of data is updated, the old version of the block is stored in what is known as a Rollback Segment. Rollback segments are now called Undo Tablespaces in Oracle 9i but the two names refer to virtually the same entity with slightly different management characteristics so for the purposes of this paper we will refer to Rollback Segments or RBS. Note that these rollback segments are not for crash recovery but rather they are for transactional purposes. This means that, not only is the old block of data stored in the rollback segment, but the undo and redo images of the changed data are also stored in the redo logs for recovery purposes.

Each Oracle block (known as a page in DB2) has what is called a System Change Number (SCN) associated with it. Logically you can think of this as a timestamp for the data page to indicate the version of the page. So when a row on a page is updated, the current version of the block with the current SCN is copied into the rollback segment, then the data page is updated and given a new SCN (which is stored in the block header).

Highlights

With Oracle, the data you see in any query is the data as it existed at the start of that query. This does not conform to any ANSI standard isolation level, nor is it the way other RDBMSs work. To accomplish this, each statement picks up the SCN from the time the statement started (i.e. it picks up the data timestamp of the statement start time). Then as it reads data blocks, it checks to see if the data block has a higher (newer) SCN than the statement SCN. If the SCN of the block is higher, then Oracle knows that this block must have been updated since this statement first started so the transaction then goes to the rollback segment to try to find an older version of that block to satisfy the query. But it's a bit more onerous than it sounds. The way this works is as follows:

1. *statement A starts and is assigned SCN 105*
2. *the statement starts a table scan of table X (for example)*
3. *for every block that is read in from disk into the bufferpool, the SCN of that block is checked*
4. *if the block SCN is greater than SCN 105 then the block is duplicated (cloned) inside the bufferpool*
5. *lets assume that the block SCN is 108*
6. *the statement then sidetracks into the rollback segment looking for the last transaction to update this page. The transaction that updated this block to 108 is undone on the clone block using the information found in the rollback segment. So the block now has SCN 107 (for example).*
7. *continue cloning the block and applying the rollback segment images as in step 6 until you have a block that has an SCN < 105 (your transactions SCN)*
8. *then continue to read the next block from the table into the bufferpool (and go to step 6 as needed)*

... [A]ny statement that is trying to read a page that either has been updated or is currently in the process of being updated, must go to the rollback segment to get an older version of the data so that it does not wait on the updaters lock.

So in essence, any statement that is trying to read a page that either has been updated or is currently in the process of being updated, must go to the rollback segment to get an older version of the data so that it does not wait on the updaters lock. It is true that a pure reader will not wait on a writer but it is reading old, and possibly out of date data, performing additional I/O, filling the bufferpool with non-reusable pages, and using up extra CPU intructions. Even committed transactions are not seen by the reader if the reading statement started before the updating statement.

Highlights

How DB2 handles Locking and Concurrency

DB2 supports the following standard isolation levels:

Repeatable Read

Repeatable read (RR) locks all the rows an application references within a unit of work. Using repeatable read, a SELECT statement issued by an application twice within the same unit of work in which the cursor was opened, gives the same result each time. With repeatable read, lost updates, access to uncommitted data, and phantom rows are not possible.

Read Stability

Read stability (RS) locks only those rows that an application retrieves within a unit of work. It ensures that any qualifying row read during a unit of work is not changed by other application processes until the unit of work completes, and that any row changed by another application process is not read until the change is committed by that process. That is, "nonrepeatable read" behavior is not possible.

Cursor Stability

Cursor stability (CS) locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.

No other applications can update or delete a row that a cursor stability application has retrieved while any updatable cursor is positioned on the row. Cursor stability applications cannot see uncommitted changes of other applications.

Uncommitted Read

The [uncommitted read] application also does not lock other applications out of the row it is reading

Uncommitted read (UR) allows an application to access uncommitted changes of other transactions. The application also does not lock other applications out of the row it is reading, unless the other application attempts to drop or alter the table. Uncommitted read works differently for read-only and updatable cursors.

Highlights

Read-only cursors can access most uncommitted changes of other transactions. However, tables, views, and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes by other transactions can be read before they are committed or rolled back.

Issues with Multi Version Read Consistency

There are several issues with multi version read consistency

Management of rollback segments.

Rollback segments are simply disk space set aside to store old images of data. They are physically a set of operating system files of a predefined size that work in a circular fashion. That is, old data is put into rbs 1 followed by rbs 2 and so on. When all of the rollback segments are full, the process wraps back around to the first segment. Undo Tablespaces in Oracle 9i work in a similar fashion except that the information is stored in an Oracle managed tablespace which is made up of operating system data files so the process is almost identical. There is a DBA management burden to configure and maintain rollback segments. How big should your rollback segments be? What if you have a single transaction that does not fit into your rollback segments? What happens if you run out of space?

... if there is not enough rollback segment space for your transaction, the transaction fails.

Well running out of space is not a problem...Oracle simply cancels your transaction! Yes that's right, if there is not enough rollback segment space for your transaction, the transaction fails. Similarly what happens if the rollback segment has filled up and wrapped on itself, overwriting an old image that you need (i.e. your statement is long running and you need an image that is older but no longer in the rollback segment). That read only transaction also fails with an ORA-1555 "Snapshot too old". Ask any Oracle DBA what ORA-1555 means and they will be all too aware of it. Many DBAs have spent significant amounts of time trying to manage the size of their rollback segments or undo tablespaces to avoid this problem. This is not an issue with DB2 as there are no rollback segments to worry about so you would never get a Snapshot Too Old error.

Highlishts

Each implementation does what it is designed to do and it's the application development that either makes the best use of the technology or runs into issues.

ORA-1555 Snapshot Too Old

As shown above, transactions that need older versions of data that are no longer around can fail with an ORA-1555. To get around this well known Oracle problem, application developers using Oracle will write their applications and commit as infrequently as possible. Why? Because uncommitted changes do not get overwritten in the rollback segments so this type of application coding attempts to alleviate the ORA-1555 problem. What you will often find is that if you directly port an Oracle application to DB2, you will find an excess of locks being held and concurrency will suffer because the application is not designed to take advantage of the underlying concurrency model. Similarly if you took a DB2 application and ported it directly to Oracle, you would find that it may have problems with ORA-1555 because of the frequent commits in the application. Each implementation does what it is designed to do and it's the application development that either makes the best use of the technology or runs into issues.

Here is a quote from the Oracle Application Developer's Guide – Fundamentals

> *Long running read-only transactions sometimes receive a "snapshot too old" error (ORA-01555). Create more, or larger, rollback segments to avoid this. You can also issue long-running queries when online transaction processing is at a minimum, or you can obtain a shared lock on the table before querying it, preventing any other modifications during the transaction.* [2]

Oracle's concurrency model is page based not row based

In Oracle, the SCN is stored in the header of each data block (a.k.a. page). So if any record on that block is modified, the SCN for that block is updated. If a transaction is looking for record 5 on block 106, it may have to clone and reconstruct several different version of block 106 even if record 5 has never changed. With DB2, the concurrency mechanisms work on the row level so if one transaction is locking row Y and another transaction wants to look at row X (even if they are both on the same page), then both transaction would be able to proceed.

How do you force readers to see current data rather than outdated information?
In real world applications, transactions often need to be serialized. That is, the results of two independent transactions should be the same regardless of when they were run (either independently or overlapping). Here is an example to show you how Oracle would behave if you simply coded two transactions without taking into account the multi version read consistency of Oracle.

Transaction 1	Transaction 2
Begin transaction.	
	Begin transaction.
Select available seats on flight ABC111. See seat 23F is the last seat available Reserve this seat.	
	Select available seats on flight ABC111. Also sees 23F as Oracle will go to the rollback segment to get the old version of that block.
Commit Transaction.	
	Reserve this seat.
	Commit Transaction. Successful but now the flight is oversold.

Highlishts

Here is how you must code your Oracle application in order to be able to properly serialize your transactions.

Transaction 1	Transaction 2
Begin transaction.	
	Begin transaction.
Select available seats on flight ABC111 using FOR UPDATE clause. See seat 23F is the last seat available	
Reserve this seat.	
	Select available seats on flight ABC111 using FOR UPDATE clause
	Blocks waiting on the lock from the first transaction
Commit Transaction.	
	Select Returns with no seats left so you book your seat on another flight.
	Commit Transaction

Note that this last example is the default behavior with DB2. That is, a second transaction will wait for the first transaction to commit before it sees changed data. That way you get current results not old results. If you do not want to wait on a lock then there is the other ANSI standard isolation level called uncommitted read which will see the current value of the data even if that data is not yet committed.

Wasting space in the buffercache

The whole point behind a buffer cache is to store many frequently used pages in memory to avoid disk I/O and share data between multiple users.

As demonstrated in section 1 above, in order for Oracle to rebuild the old versions of the data, it must duplicate or clone the newer page inside of the buffer cache (called a bufferpool in DB2). This cloning has the unwanted side effect of flooding the buffer cache with non-reusable information. The whole point behind a buffer cache is to store many frequently used pages in memory to avoid disk I/O and share data between multiple users. Because of the way Oracle has implemented their multi version read consistency, the bufferpool fills with pages that are not shared by multiple transactions and thus more physical I/Os are required thus impacting the performance of the system.

Highlights

DB2 has shown consistent leadership in the SAP three-tier SD benchmark

Benchmarks Prove it

TPC-H

In data warehousing benchmarks, namely TPC-H, readers and writers do not run concurrently. That is, the benchmark is designed to run query streams and update streams at different times so multi-version read consistency does not play a role.

SAP

Oracle often mentions that the best example of multi version read consistency is SAP R/3. However, here are the current SAP three-tier Standard Application Sales and Distribution (SD) Benchmark results (as of September 4, 2002). Although benchmarks are sometimes a leapfrog game, Oracle is definitely not the leader in SAP three-tier SD benchmark results even using multi version read concurrency. In fact, DB2 has shown consistent leadership in the SAP three-tier SD benchmark.

SAP SD three-tier benchmark results

1. DB2 V7.2 running on AIX 5.1 on 32-way IBM pSeries p690
2. Microsoft SQL Server 2000 running on Windows Datacenter Server on 32-way Unisys ES7000 server
3. DB2 V7.2 running on AIX 4.3.3 on 24-way IBM pSeries p680
4. DB2 V7.2 running on AIX 4.3.3 on 24-way Bull Escala EPC2450
5. Microsoft SQL Server 2000 running on Windows 2000 Server on 32-way Unisys ES7000 server
6. Oracle 8.1.6 running on Solaris 8 on 64-way Fujitsu Siemens Primepower

Peoplesoft and Baan

The same holds true for Peoplesoft and Baan sizing benchmarks. Oracle and ISV restrictions do no permit us to share the details, but feel free to ask PeopleSoft and Baan for solution sizings that show DB2 and Oracle on the same system.

TPC-C

TPC-C benchmarks require read and write transactions simultaneously. In this benchmark however, Oracle codes their driver to bypass multi version read consistency errors by using serializable isolation level. Here is a line of code from Oracle's full disclosure report which you can download from http://www.tpc.org. It shows that Oracle is using serializable isolation level.

```
#define SQLTXT "alter session set
isolation_level = serializable"
```

But this does not mean that Oracle won't run into locking and concurrency problems so they also catch these errors.

```
#define DEADLOCK 60 /* ORA-00060: deadlock */

#define NOT_SERIALIZABLE 8177 /* ORA-08177:
transaction not serializable */

#define SNAPSHOT_TOO_OLD 1555 /* ORA-01555:
snapshot too old */

WHEN not_serializable OR deadlock OR
snapshot_too_old THEN ROLLBACK;

:retry := :retry + 1;
```

So in the TPC-C benchmark, they must code their application driver to both bypass multi version reads as well as code to catch snapshot too old errors, rollback that unit of work and try again.

Contents

With DB2 the default behavior is to serialize transactions such that each transaction sees the current committed data.

Conclusions

With Oracle, as with any other database, you design and code your application with an understanding of the underlying isolation and concurrency model. With DB2 the default behavior is to serialize transactions such that each transaction sees the current committed data. With Oracle, the default behavior is to return old and possibly out of date information and therefore application developers must code around this default behavior to obtain serializable transactions.

Other database vendors have not implemented Oracle's Multi Version Read Consistency isolation nor has it proven to be a performance advantage in the industry standard, ISV or real life customer benchmarks. Simply stated; Oracle has taking an old architectural decision and is now trying to showcase it as a differentiator, when in fact it is simply a concurrency model that developers must code around and one that adds an extra burden of management on the DBA.

Appendix A: Required Benchmark Information

These SAP standard application benchmarks fully comply with SAP's issued benchmark regulations and have been audited and certified by SAP.

(1) 47,008 SAP SD benchmark users; 1.97 seconds average dialog response time; 4,713,000 processed order line items/hour; OS: AIX 5.1; RDBMS: DB2 UDB 7.2; SAP R/3 Release: 4.6C; Database server: IBM eServer pSeries p690, 32-processors SMP, Power4 1.3GHz, 24 MB L2 cache, 64 GB main memory (cert #2002046).

(2) 26,000 SAP SD benchmark users; 1.97 seconds average dialog response time; 2,606,000 processed order line items/hour; OS: Microsoft Windows Datacenter Server Limited Edition; RDBMS: Microsoft SQL Server 2000; SAP R/3 Release: 4.6C; Database server: Unisys e-@ction Enterprise Server ES7000, 32-processors, Pentium III 900 MHz, 2 MB L2 cache, 12 GB main memory (cert #2002007).

(3) 25,560 SAP SD benchmark users; 1.92 seconds average dialog response time; 2,573,330 processed order line items/hour; OS: AIX 4.3.3; RDBMS: DB2 UDB 7.2; SAP R/3 Release: 4.6C; Database server: IBM eServer pSeries p680, 24-processors SMP, RS64 IV 600 MHz, 16 MB L2 cache, 64 GB main memory (cert #2001046).

(4) 25,560 SAP SD benchmark users; 1.95 seconds average dialog response time; 2,566,000 processed order line items/hour; OS: AIX 4.3.3; RDBMS: DB2 UDB 7.2; R/3 Release: 4.6C; 3,800Gb total disk space; Database server: Bull Escala EPC2450, 24-processors SMP, RS64 IV 600 MHz, 16 MB L2 cache, 64 GB main memory (cert #2001049).

(5) 24,000 SAP SD benchmark users; 1.94 seconds average dialog response time; 2,411,330 processed order line items/hour; OS: Windows 2000; RDBMS: Microsoft SQL Server 2000 SP1; R/3 Release: 4.6C; Database server: Unisys e-@action Enterprise Server ES7000, 32-processors, Pentium III Xeon 900 MHz, 2 MB L2 cache, 12 GB main memory (cert #2001039)

(6) 23,000 SAP SD benchmark users; 1.73 seconds average dialog response time; 2,353,670 processed order line items/hour; OS: Solaris 8; RDBMS: Oracle 8.1.6; SAP R/3 Release: 4.6B; Database server: Fujitsu Siemens PRIMEPOWER 2000, 64-processor SMP, Sparc64 450 MHz, 8 MB L2 cache, 64 GB main memory (cert #2000029)

References

1. Oracle comparison to DB2 (page 5)
http://otn.oracle.com/products/oracle9i/pdf/CWP_9IVSDB2_PERF.PDF

2. Oracle 9i Application Developer's Guide – Fundamentals Release 2 (9.2)
Page 7-8

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

Printed in United States of America
8-02