DB2 Data Management Software

A Technical Discussion of Row Level Locking

IBM Software Group
Toronto Laboratory

Contents

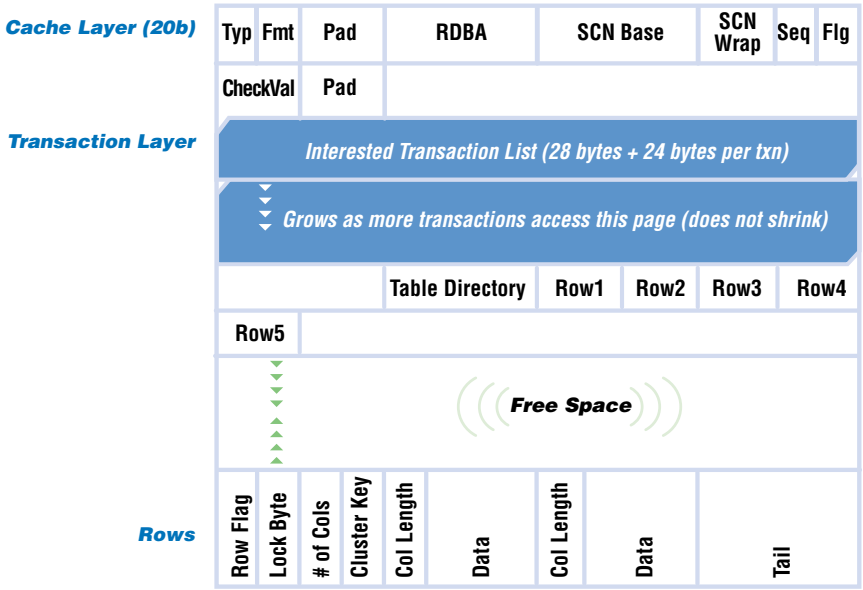
Introduction

Concurrency has always been an important issue with database systems. As well, in todays systems the trend is towards self managing of memory to reduce DBA cost of management. DB2 has designed its locking mechanisms to address both issues simultaneously. Oracle is claiming that DB2s lock memory managment mechanism is inferior[1]. The truth is that Oracle's locking mechanism is not designed for the high performance servers that are available today.

How Oracle Manages Locks

Oracle actually stores lock information on the same data page (called a block in Oracle terminology) that the data rows exist on. On every data and index page, Oracle has a section of the header called the transaction layer. This section is variable in size and will grow as more transactions simultaneously access records on the data page. Every time a transaction accesses a row on a data page, an entry is put in the Interested Transaction List (ITL) on that page. For every transaction, 24 bytes on the page are used up for an ITL. As more transactions access the same data page, the transaction layer grows in size (using up more disk space for transactional information).

Oracle Data Block (page) Format



Highlights

Oracle uses 24bytes of disk space per transaction per page.

Transactions sleep waiting for free space.

Issues With This Locking Mechanism

Overhead using a disk based locking mechanism

Each ITL uses 24 bytes of disk space. By default each data page has 1 ITL and each index page has 2 ITL slots reserved. So by default 24 bytes of every data page (and 48 bytes of every index page) is not usable for user data. In a more realistic environment where several concurrent transactions are accessing the same data blocks, it would not be unusual to find 10 ITL slots on each data page. That's 240 bytes on each data page wasted or 12% of all data space (assuming 2k page for an OLTP system)

What happens if I have a hot data page or hot table

If a table has a lot of concurrent transactions then it will use up more disk space for transactions. In many applications there exits a set of hot records or a hot table. So what happens if you have 1000 concurrent update transactions on a small table? How do you limit the growth of the transaction layer from using up all the space on a page? Oracle provides a MAXTRANS parameter on each table to limit the growth of the transaction layer of the data page.

How does the DBA know what tables will be hot and how much space should be reserved?

Good question. For peak performance, this type of locking mechanism requires the DBA to determine how much space on each data page needs to be set aside for lock information which is dependent on the application. The transaction layer can grow if more transaction space is required but it does not automatically shrink. This is also an additional item for DBA's to monitor to see if pages are becoming too full of ITL slots thus leaving less (or no) room for new data

What happens if there is no more space on the data page for a new transaction or you hit MAXTRANS?

If a new transaction cannot allocate space for an ITL, the transaction will go to sleep. Even if the row that this transaction wants to update is not being accessed by any other transaction, a lock cannot be acquired because there is not enough free space on the page to write an ITL (a locking record). Even if there is lots of free space on other pages in the table, if there is no free space on this particular page, the transaction must wait. When it wakes up, it looks to see if there is a free ITL slot. If there is still no space, it goes back to sleep.

Highlights

The problem with this algorithm is that while one transaction is sleeping, another transaction can come in and get an ITL slot that was made available while the first transaction was still sleeping. This means that there is no guarantee that a transaction can acquire the lock it wants and locks are not allocated to transactions in the order they are requested (i.e. a transaction can jump in front of another transaction that has been waiting).

How DB2 Manages Locks

DB2 employs a patented[2] locking mechanism that stores locks in memory. Since locks are transient objects that do not require persistence across server failures, there is no need to write lock information to disk. If there is a database outage for example, then a database restart does not need to know what row locks were granted at the time of the crash. This is why memory is a better resource to use for lock information as you do not need to waste disk space to store these nonpersistent objects.

Memory is a better resource to use for nonpersistent objects like locks.

The way DB2 works is that every lock requested has a name (known as a Lock Request Block or LRB). When a lock is requested, that locks LRB is stored in a memory area known as the lock list. The size of this locklist is determined by a database configuration parameter called LOCKLIST. By default, 200k of memory is set aside for locks and the DB2 Performance Configuration Wizard can assist the DBA in determining an appropriate size for the locklist given the application type and usage requirements. If multiple applications try to lock the same record in exclusive mode (for update) then DB2 creates a linked list of these lock requests (first come first served) in memory. Multiple transactions may be sharing a lock at the same time, given that they are both reading the record and not updating it. The linked list of lock requests contains both transactions using the lock and those transactions waiting for the lock. When a transaction is waiting on a lock it is actually waiting on a semaphore (not on a sleep timer that Oracle uses). When the first transaction on the list releases its lock, it removes itself from the lock chain and posts a message to the next transaction(s) in the sequence to allow the waiting transaction(s) to continue and take over ownership of that lock. So with DB2 a waiting lock request will not be pre-empted by another lock request, they are processed in the sequence they are requested.

Highlights

Now what happens if one application starts acquiring an unusually large number of locks and starts using up too much memory. In this case, DB2 has an automatic (SMART) algorithm to reduce the memory requirements on the system. The process is called lock escalation and should be an extremely infrequent event if a system has sufficient physical memory and is tuned appropriately. If the LOCKLIST memory area is full or if one application acquires "too many" locks then DB2 invokes a memory requirement reduction algorithm known as lock escalation. "Too many" locks is defined by the DBA using a second configuration parameter called MAXLOCKS. MAXLOCKS defines the percentage of the LOCKLIST that any one application can acquire before a lock escalation will take place. By default MAXLOCKS is set to 22% of the LOCKLIST so by default if one application acquires more than 22% of the locklist, DB2 will escalate. When a lock escalation occurs, the application holding the most locks (and therefore using up the most memory) will convert its row level locks to a table level lock, thus reducing memory requirements on the system. As well this has the added advantage that the lock hogging application is no longer required to acquire row level locks for the table it already has locked and can therefore finish its work faster and get out of everyone else's way.

Conclusions

Using memory for transient and nonpersistent locking information is a solution that matches well with today's database environments where servers have a significant amount of memory.

As the size of databases moves into the Terabyte range, it is important to consider disk space utilization and to use disk space wisely. Adding an extra 100 bytes to a page that cannot be used for data but instead is used for locks may seem like a small amount but when you get into systems that have millions of data pages, the wasted disk space not only adds up to a lot of unused space but means real dollars lost and extra dollars spent monitoring this unnecessary space usage. Using memory for transient and nonpersistent locking information is a solution that matches well with today's database environments where servers have a significant amount of memory. As well the automatic facilities available in DB2 like the Configuration Advisor and the Lock Escalation Algorithm allow DBAs to utilize the available memory on the machine while not adding the extra burden of having to reorganize data to reclaim unused lock space.

References

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

2. Method for managing lock escalation in a multiprocessing, multiprogramming environment US Patent #4,716,528 (Crus; Richard A.; Haderle; Donald J; Herron; Howard W)

Note that the above is accurate for DB2 V7.2, DB2 V8.1 and Oracle 9i. Future versions may or may not change this behavior.

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

Printed in United States of America
8-02