Keeping Track of DB2 Locking Conflicts
Nancy Perkinson

DB2 locking conflicts happen all the time at every shop. Mostly they go by unnoticed because they are resolved quickly and efficiently behind the scenes, but occasionally locking conflicts can cause frustrating abends and/or greatly lengthen clock response times. The easiest way to see them is to view them in action in a DB2 monitoring tool like Omegamon, Mainview, InSight, or other product. You can also see the full-blown completed deadlocks and timeouts in the DB2 master address space syslog but only after the fight is over and the crying has started – if the fight is spontaneously resolved before the IRLM has to declare a winner, then without an accounting trace report no one knows it even happened. But today we’re just going to look at the full-blown deadlocks and timeouts, and give a few ideas for why and how you might want to monitor and track them.

Why track deadlocks and timeouts? Everybody loves pointing the finger, and by tracking deadlocks and timeouts you effectively have something at which to point. Whether they are truly not playing fair, and how to ultimately resolve the problem is for someone at your shop to call. This data will show if something (a batch job, a CICS transid, a distributed thread, etc) is involved in more than expected locking or timeout situations, or if a resource surprisingly is a source of contention. You'll be able to track if certain days of the week or times of the monthly business cycle are more prone to problems. And you might just find that you have definitive proof that the big update job on the last day of the month is what is causing many users to experience poor response time between 9:15 and 9:30 in the morning. The opportunities for sleuthing with this data are many, as are the opportunities to improve the even and regular flow of work in your system.

Locking conflicts encompass two things: Deadlocks and Timeouts. Deadlocks are a situation where one guy has resource A and wants resource B, and another guy has resource B and wants resource A. Typically deadlocks are a quick death for one of the participants with the IRLM recognizing very quickly that it is an unresolvable conflict and someone must be forced to surrender while the other participant is allowed to continue. The guy who wins the fight doesn’t even realize there was a fight, but the guy who lost is issued an SQLCODE -911 and all his work is rolled back to his last commit point. Timeouts are different, but produce somewhat the same results. You'll get a timeout when you want resource A, and the other guy has resource A so you just patiently wait your turn. But timeouts aren’t always quick because the other guy might be doing loads of work without committing, thus keeping resource A tied up for a very long time. Finally, after a specified amount of time, the IRLM says “Sorry, but it doesn’t look like the other guy is going to let go” and he fails your SQL call with an SQLCODE
-911, and all your work is rolled back to the last commit point. When you keep track of these deadlocks and timeouts, who won the fight and who lost, what resources were involved, and when it happened you are armed with the ability to spot patterns and behaviors that negatively impact the performance of 50% of the participants in these situations. As the Performance Analyst you can share these findings with the responsible parties, and guide them in a solution to the conflict.

**Setting up a Repository for the Locking Conflict Data**

First off, you need a repository for the locking conflict data you'll be gathering every day – we choose to use a DB2 table and called it LOCKING_CONFLICTS. Here are the key columns we set up to save the info:

- **WINNER** – The guy that didn’t get the abend
- **LOSER** – The guy that got the timeout or deadlock
- **RESOURCE_NAME_DB** – the first resource database name involved
- **RESOURCE_NAME_SP** – the first resource tablespace or indexspace name involved
- **RESOURCE_CREATOR** – the first resource creator name involved
- **RESOURCE_NAME** – the first resource table or index name involved
- **RESOURCE_TYPE** - was it a datapage, a row, a table?
- **RESOURCE_NAME_DB2** – the second resource database name involved
- **RESOURCE_NAME_SP2** – the second resource tablespace or indexspace name
- **RESOURCE_CREATOR2** – the second resource creator name involved
- **RESOURCE_NAME2** – the second resource table or index name involved
- **RESOURCE_TYPE2** – what was the second resource (datapage, row, table?)
- **REASON_CODE** – was it a DEADLOCK or a TIMEOUT
- **CONFLICT_TS** – timestamp of event

**How to get the Deadlock and Timeout information**

There are quite a few ways you can grab Deadlock and Timeout information for insertion into your historical repository. Most of the DB2 monitors (Omegamon, Mainview, DB2 PM, etc) have the ability to produce Locking Conflict reports, and these can be run daily and then fed into a simple routine that scrapes them for the LOSER/WINNER information. But probably the best long term solution is to pull the information straight from SMF data where you’d pull IFCIDs 172 (deadlocks) and 196 (timeouts) from the MVS SMF 100 records. No matter how you get the information, once a day you just need to pull the data for the entire 24-hour period, parse through it for the applicable fields, make it pretty, and then store it in the repository.
How and When to analyze the Data

Probably the best way to start analyzing the Locking Conflicts data is to just start browsing though the most recent days’ entries. You can graph them by timestamp and see if you have spikes around certain time periods, or you can group them by WINNER or LOSER, or a combination of the two, and see if there are some definite bad boys or proverbial victims. If you have one entry that keeps showing up in the WINNER column, then there is a good chance that this guy is in need of some COMMIT logic or some change to his logic or run time to keep him from stepping on everybody else’s toes.

The ways to analyze this data are many, and it typically will provide quite a few answers to the frustrations that people have just accepted over time.

No shop is ever going to completely avoid Deadlocks and Timeouts, but most of us can go a long way in edging toward zero-tolerance of these situations by analyzing when they are happening and the major players involved. Once you show your applications people the proof of what is causing poor performance hopefully they’ll be more than willing to make the changes in their code so that it will be more polite and happily co-existing with the other systems in your shop. Thus, you might need to educate your applications people about the best rules to avoid locking conflicts.

The #1 Rule is to avoid Locking Conflicts

The most important thing is to avoid locking conflicts entirely, but when you notice regular trends or certain things becoming popular entries in your locking conflicts database then you’ll need to approach the responsible parties and offer some suggestions. Here are the three main things that are important in avoiding locking conflicts:

- Keep units of work small, and COMMIT or syncpoint at a reasonable frequency to reduce the amount of time that locks are held. COMMITs cost a little CPU, but they also free up resources – find a happy medium.
- When operations must be simultaneous, make an effort to access the resources in the same order. For example, if you have two processes that update Table A and Table B, make sure they both update Table A first, then Table B – this will greatly lessen the possibility of deadlocks between the two processes.
- Be careful when your DBAs change isolation levels (a bind parameter) or locksize options (switching to row-level locking, etc) or lock timeout values because these change the nature and behavior of your locking conflicts sometimes producing unexpected results. For example, people might love the idea of row-level locking and think it will solve all their problems, but these can escalate to entire tablespace locks much faster and could cause even bigger problems.
Conclusion

Locking conflicts are something that we all deal with. Certain steps minimize their impact on our systems, but in order to know who/what to target for these steps we have to be able to analyze the major winners and losers participating in the worst locking conflicts. A locking conflicts historical repository allows the Performance Analyst to continually monitor the situation, and provides necessary documentation for identifying the applications that might be ripe for changes to make them more compatible with the other applications in your shop.

And just two disclaimers this time:

- This article is written at a very basic level, as not all locks are the same in reality. There are different kinds/levels of locks and some are compatible with others while others are not. Locking discussions and analysis can get very complicated when you start considering tablespace, table, and page/row locks, not to mention bind parameters. And of course things change from release to release. Never forget that a good DBA is a Performance Analyst’s best friend.
- This Locking Conflicts Repository wasn’t our original idea. I met a really smart DBA from JB Hunt back at a regional meeting (maybe 1992?), and I got this idea from him. I feel terrible because I can’t recall his name, but I do remember specifically that it was JB Hunt, and I’m sure he’s gone on to much bigger and better things now. But, once again, this brings to light the ongoing benefits of attending regional meetings and networking!

---

1 The Deadlock time check is a parameter in the start up of the IRLM. If set to 5 seconds, then it checks every 5 clock seconds to see if any exist. So, in this example, the longest a deadlock would be allowed to exist would be 5 clock seconds before one of the resources is forcefully freed.

2 The Timeout value is a zparm and can be adjusted there. 60 clock seconds is common, but a shop can easily choose to be more or less patient with that value.

3 You will need DB2 Statistics trace class 3 for this information; most shops have this statistics trace class on already, but you can check with a DBA to be certain.

4 There are a number of common things DBAs do to lessen the instances of lock issues including the adjustment of lock escalation settings (zparms), changing bind option CURRENTDATA which allows some lock avoidance techniques to be used, or changing the bind option ISOLATION (Cursor Stability – CS is most common). I’m sure your DBAs are on the mark with all of these, but it never hurts for the Performance Analyst to know more about these controls.