Deadlocks - types and solutions

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:31 AM PST

Contents

Prerequisites

Make sure that you are familiar with the concepts of locks, blocking and lock compatibility. Take a look at:
Blocking and deadlocks
Blocking
Deadlocks
Lock Compatibility

The first three wiki articles contain very important and useful information on basic concepts and other useful information on how to capture this events on telemetry and on customer side.

What is a deadlock

In short, occurs when two different transactions will wait to acquire a lock on a resource that is being held by the other transaction. Since both transactions are waiting on each other, one of them needs to be killed so the other can proceed.

The rules for the engine to choose the deadlock victim are:

- one of the queries has SET DEADLOCK_PRIORITY HIGH or SET DEADLOCK_PRIORITY LOW. For example, if one the transactions has SET DEADLOCK_PRIORITY LOW it will be the deadlock victim.
- if there is no SET on any of the transactions, the deadlock victim will be the transaction with the least expensive rollback.

The handling of the deadlocks is made by a server process called LOCK_MONITOR. This process runs every 5 seconds. After finding one, it will adopt a pessimist approach lowering to a check of every 100 miliseconds. It will come back to 5 seconds if no deadlocks are found.

Like so, a deadlock will not last more than 5 seconds (excepting the eventual rollback of the transaction that was chosen has deadlock victim).

About SET DEADLOCK_PRIORITY

This is an option that the customer can set to control if a particular transaction will be victim or not of deadlock, if that specific process comes involved on a deadlock situation. But, like mentioned above, in normal circunstances the LOCK MONITOR will choose has deadlock victim the transaction with the least expensive rollback. Changing this behavior the customer might see in some cases long rollbacks. In conclusion, the usage of this setting neeeds to be carefuly assessed.

Deadlock possible root causes and solutions

Deadlocks, like mentioned earlier, is when two transactions are waiting for each other and one of them is killed by the engine so that the other one can proceed. But deadlocks can have different root causes and different solutions.

The purpose of this TSG is to give an overview of some common types od deadlocks and solutions.

Lets see a few example belows.

Read Write Deadlock

One transaction is reading and the other one is writing. Note that in some cases one the transactions might not be reading on explicit manner. It might be reading due to an FK relationship (when insert or updating a record on one table, needs to check if the value exists on the parent table)

Example of Read Write Deadlock:

Im cutting some parts of the XML. Just leaving the most interesting parts

```
<deadlock>
<victim-list>
<victimProcess id="process2a6c0073848" />
</victim-list>
<keylock hobtid="72057594047430656" dbid="9" objectname="7aaddf14-3f19-4a30-bac2-386c9e4110c2.dbo.Product" ind
<owner-list>
<owner id="process2a688c68108" mode="X" />
</owner-list>
<waiter-list>
<waiter id="process2a6c0073848" mode="S" requestType="wait" />
</waiter-list>
</keylock>
<keylock hobtid="72057594047692800" dbid="9" objectname="7aaddf14-3f19-4a30-bac2-386c9e4110c2.dbo.ProductModel
<owner-list>
<owner id="process2a6c0073848" mode="X" />
</owner-list>
<waiter-list>
<waiter id="process2a688c68108" mode="S" requestType="wait" />
</waiter-list>
</keylock>
</resource-list>
</deadlock>
```

Operations that are being done:

- One transaction that writes on ProductModel table and reads from Product table, inside a transaction (BEGIN TRANSACTION...).
- A second transaction that writes on Product table and reads from ProductModel table, inside a transaction (BEGIN TRANSACTION...).

Describing what is happening here:

1- process2a688c68108 - holds a X lock on Product table, PK__Product__B40CC6ED2401060E index

2- process2a6c0073848 - holds a X lock on ProductModel table, ProductModel_ix index

3- process2a688c68108 - tries to acquire an S lock on ProductModel table, ProductModel_ix index - **Blocked by process2a6c0073848 X Lock**

4- process2a6c0073848 - tries to acquire an S lock on Product table, ProductModel_ix index - **Blocked by process2a688c68108 X Lock**

5- process2a688c68108 - is victim for deadlock (logused="408" vs logused="37996")

Solutions for Read Write Deadlock

- Change to read commited snapshot isolation
- Remove the SELECT out of the transaction
- Change the order of access of objects (if needed, perform the select first into a temp table)
- On case of a FK relationship, might be worth to check if the access to the parent table is optimized (index seek)

Write Write Deadlock

Both transactions are writing.

```
<deadlock>
<victim-list>
<victimProcess id="process2a6c0073848" />
</victim-list>
<process-list>
(...)
<resource-list>
<keylock hobtid="72057594047627264" dbid="9" objectname="7aaddf14-3f19-4a30-bac2-386c9e4110c2.dbo.ProductModel
<owner-list>
<owner id="process2a6acfa08c8" mode="X" />
</owner-list>
<waiter-list>
<waiter id="process2a6c0073848" mode="X" requestType="wait" />
</waiter-list>
</keylock>
<keylock hobtid="72057594047430656" dbid="9" objectname="7aaddf14-3f19-4a30-bac2-386c9e4110c2.dbo.Product" ind
<owner-list>
<owner id="process2a6c0073848" mode="X" />
</owner-list>
<waiter-list>
<waiter id="process2a6acfa08c8" mode="X" requestType="wait" />
</waiter-list>
</keylock>
</resource-list>
</deadlock>
```

Operations that are being done:

- One transaction that writes on ProductModel table and write on Product table, inside a transaction (BEGIN TRANSACTION...).
- A second transaction that writes on Product table and write on ProductModel table, inside a transaction (BEGIN TRANSACTION...).

Describing what is happening here:

1- process2a6acfa08c8 - holds a X lock on ProductModel table, index PK__ProductM__E8D7A1CC6F7986A5

2- process2a6c0073848 - owns a X lock on Product table, index PK__Product__B40CC6ED2401060E

3- process2a6acfa08c8 - tries to acquire a X lock on Product table, index PK__Product__B40CC6ED2401060E. Blocked by process2a6c0073848 X lock

4- process2a6c0073848 - tries to acquire a X lock on ProductModel table, index PK__ProductM__E8D7A1CC6F7986A5. Blocked by process2a6acfa08c8 X lock

5- process2a6c0073848 - chosen as deadlock victim (logused="348" vs logused="440")

Solutions for Write Write Deadlock

- Change the order of access to objects. For example, instead of having a transaction that writes on A and then on B and another transaction writting on B and than A, make all transactions access the objects on the same order. For example A and B on both.

Key or RID Lookup Deadlock

Happens when a query holds a lock on index and tries to access the leaf level (clustered or Heap) that has lock from a transaction that is changing data. Deadlock occurs with 2 transactions on the same table - we will see that objects and query associated with the deadlock are all on the same table.

```
<deadlock>
<victim-list>
<victimProcess id="process2a6b150c8c8" />
</victim-list>(...)
<keylock hobtid="72057594048544768" dbid="9" objectname="7aaddf14-3f19-4a30-bac2-386c9e4110c2.dbo.Orders" inde
<owner-list>
<owner id="process2a6a30cb468" mode="X" />
</owner-list>
<waiter-list>
<waiter id="process2a6b150c8c8" mode="S" requestType="wait" />
</waiter-list>
</keylock>
<keylock hobtid="72057594048610304" dbid="9" objectname="7aaddf14-3f19-4a30-bac2-386c9e4110c2.dbo.Orders" inde
<owner-list>
<owner id="process2a6b150c8c8" mode="S" />
</owner-list>
<waiter-list>
<waiter id="process2a6a30cb468" mode="X" requestType="wait" />
</waiter-list>
</keylock>
</resource-list>
</deadlock>
```

Operations that are being done:

- One query reads from Orders table. It uses a nonclustered index, but doesn`t contain all the columns, so it will have to do a key lookup.
- Another query inserts on Orders tables. To insert into a table (on this case, on the clustered index), it has also to write on all nonclustered indexes

Describing what is happening here:

1- process2a6a30cb468 - holds an X lock on Orders, PK_Sales_OrderLines

2- process2a6b150c8c8 - holds an S lock on Orders, IX_Sales_OrderLines_Perf_20160301_01

3- process2a6a30cb468 - tries to acquire a S lock on Orders, PK_Sales_OrderLines (Key lookup) - Blocked by X lock

4- process2a6b150c8c8 - tries to acquire a X lock on Orders, IX_Sales_OrderLines_Perf_20160301_01 - Blocked by S lock

5- process2a6b150c8c8 - deadlock victim (logused="0" vs logused="272)

Solutions for Key or RID Lookup Deadlock

- Create a covered index, or change the query to eliminate the key lookup (for example, an implicit conversion. In order words, check plan quality).
- Change to read commited snapshot isolation.

How good have you found this content?