Proposals to prevent timeouts by reducing the number of threads in non-sales-sync

May 19, 2017

Timeouts and deadlocks

A lot of the timeouts in GCM seem to be related with threads blocking each other. In many cases it is a deadlock where two or more threads are blocked waiting for resources that are blocked by the other threads.

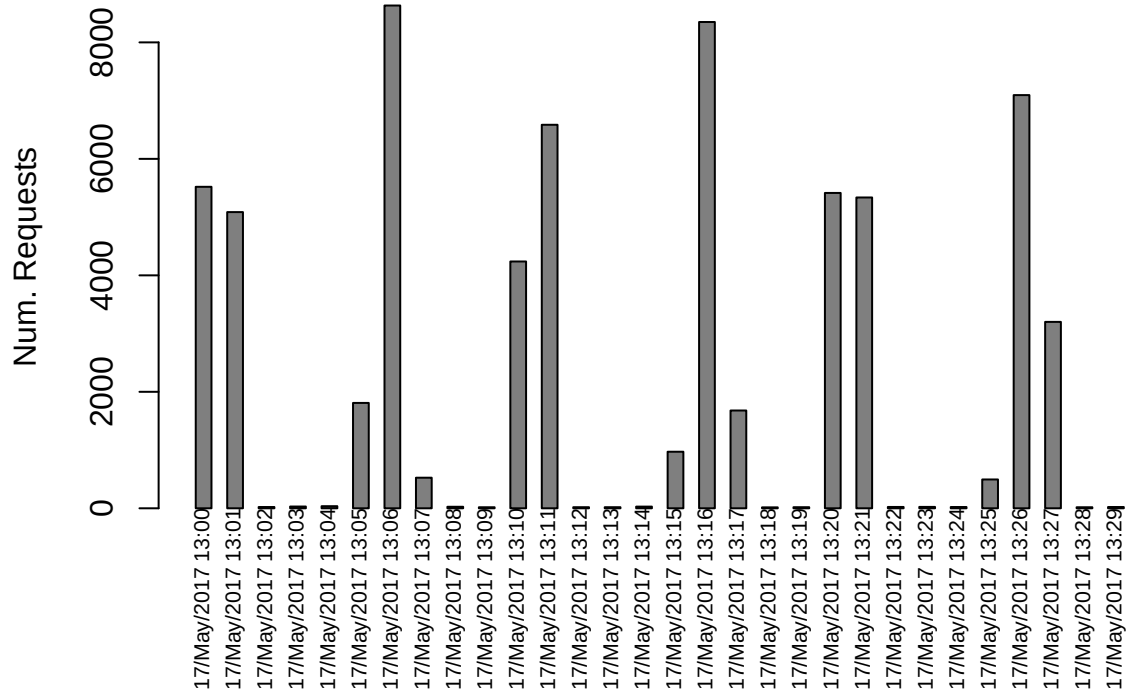
There are two conditions that make the deadlocks more likely.

1. Long transactions.
2. Concurrently access the same records.

Reducing either one will also reduce the deadlock occurrence. This document argues that it is possible to have a smaller number of threads and therefore reducing the likelihood of having timeouts and deadlocks.

Number of Requests on GCM per minute.

This chart shows the HTTP requests on p-bussvc-app01.




```
##               minute num_requests
## 1  17/May/2017 13:00         5520
## 2  17/May/2017 13:01         5086
## 3  17/May/2017 13:02           22
## 4  17/May/2017 13:03           30
## 5  17/May/2017 13:04           36
## 6  17/May/2017 13:05         1809
## 7  17/May/2017 13:06         8633
```

```
## 8  17/May/2017 13:07          525
## 9  17/May/2017 13:08           26
## 10 17/May/2017 13:09           18
## 11 17/May/2017 13:10         4238
## 12 17/May/2017 13:11         6585
## 13 17/May/2017 13:12           19
## 14 17/May/2017 13:13           19
## 15 17/May/2017 13:14           29
## 16 17/May/2017 13:15          971
## 17 17/May/2017 13:16         8350
## 18 17/May/2017 13:17         1679
## 19 17/May/2017 13:18           16
## 20 17/May/2017 13:19           19
## 21 17/May/2017 13:20         5415
## 22 17/May/2017 13:21         5337
## 23 17/May/2017 13:22           23
## 24 17/May/2017 13:23           24
## 25 17/May/2017 13:24           22
## 26 17/May/2017 13:25          495
## 27 17/May/2017 13:26         7095
## 28 17/May/2017 13:27         3201
## 29 17/May/2017 13:28           18
## 30 17/May/2017 13:29           22
```

It can be observed that GCM has usage spikes every 5 minutes. They start at 13:00, 13:05, 13:10, 13:20 and 13:25 and so forth. The usage peaks last 2 o 3 minutes and the rest of the 5 minute period has a very tiny load. The spikes align with the non-sales-sync process running times.

It would be highly desirable that we didn't have the usage spikes. It is much better to have a steady and uniform load.

Warranty and loyalty records.

These are the non sales sync records processed May 17, 2017 by hour.

```
nonsales = read.csv("t3.out");
nonsales
```

```
##    hour loyalty warranty
## 1     0    8475   400891
## 2     1    8443   401919
## 3     2    8430   402101
## 4     3    8426   402067
## 5     4    8430   402043
## 6     5    8433   402038
## 7     6    8722   402340
## 8     7    9903   403681
## 9     8   11334   405593
## 10    9   12153   407298
## 11   10   13155   409434
## 12   11   13456   410333
## 13   12   13408   410869
## 14   13   13354   411076
## 15   14   13752   411594
## 16   15   13996   412059
```

```
## 17   16   14039   412377
## 18   17   14011   412474
## 19   18   13411   411737
## 20   19   12772   411816
## 21   20   10978   409565
## 22   21    9803   408054
## 23   22    9147   406985
## 24   23    8712   406398
```

It can be shown that 97% of the processed records are warranty records.

```
sum(nonsales$warranty)/(sum(nonsales$loyalty)+sum(nonsales$warranty))
```

```
## [1] 0.9734359
```

Turns out that most of this warranty records are old records in queue that we are processing over and over.

```
SELECT to_char(REC_DT, 'YYYY-MM'), count(*) FROM "SYNC_QUE_BDO"."WAR_CUST_TRANS_QUE" group by to_char(RE
```

Month count

2017-02 3,304

2017-03 11,194

2017-04 11,882

2017-05 8,022

```
s = sum(c(3304, 11194, 11882, 8022));
s
```

```
## [1] 34402
```

So we have 34402 records that are processed over and over. Every five minutes. That is a total 10320600 records processed a day!

Recomendation

Stop re-processing the old warranty records. Either archive them and remove them from the WAR_CUST_TRANS_QUE table or only process those records with in a smaller time window. Let's say one week. It has to be considered that we have around 12,000 records that can't be processed a month. As long as the root cause is not fixed we will need an archiving and deleting mechanism that can be either automatic, or requires operations intervention.

Shouldn't we fix the root cause and not have so many rejected records? Of course. But that requires a bigger time and resource commitment. In the mean time we need to mitigate this problem.

Scheduling the non sales sync.

The current non-sales-sync process executes every five minutes. This works well as long as the executing process takes less than five minutes. What would happen if it takes longer than that ?

Consider this hypothetical example:

Assumption 1.- We are receiving 100 records per minute.

Assumption 2.- The non-sales-sync processes 200 records per minute.

Assumption 3.- We attempt to launch a process at 5 minutes intervals.

Which is simulated by the following code..

```
public static void main(String[] args) {

    int queue = 0;  // Total records in the queue.
    int inproc = 0; // Records that will be processed by the current process.
    int in = 100;   // New records in the queue every minute.
    for(int i = 0; i < 30; i++) {

        if (i % 5 == 0 && inproc == 0) {
                inproc = queue;
        }

        int out = Math.min(200, inproc);

        queue += in;

        queue  -= out;
        inproc -= out;
        println("minute=%d .- queue=%d, inproc=%d, in=%d, out=%d", i, queue, inproc, in, out);
    }
}
```

This shows that even though the rate out is twice as fast as the rate in. The queue always has between 300 and 500 records. Which means records stay longer in the queue. 3.75 minutes in the average case. Furthermore, it causes a lot of processing spikes. Sometimes it works at a rate of 200 per minute and other times is idle processing 0 records per minute.

```
minute=0 .- queue=100, inproc=0, in=100, out=0
minute=1 .- queue=200, inproc=0, in=100, out=0
minute=2 .- queue=300, inproc=0, in=100, out=0
minute=3 .- queue=400, inproc=0, in=100, out=0
minute=4 .- queue=500, inproc=0, in=100, out=0
minute=5 .- queue=400, inproc=300, in=100, out=200
minute=6 .- queue=300, inproc=100, in=100, out=200
minute=7 .- queue=300, inproc=0, in=100, out=100
minute=8 .- queue=400, inproc=0, in=100, out=0
minute=9 .- queue=500, inproc=0, in=100, out=0
minute=10 .- queue=400, inproc=300, in=100, out=200
minute=11 .- queue=300, inproc=100, in=100, out=200
minute=12 .- queue=300, inproc=0, in=100, out=100
minute=13 .- queue=400, inproc=0, in=100, out=0
minute=14 .- queue=500, inproc=0, in=100, out=0
minute=15 .- queue=400, inproc=300, in=100, out=200
minute=16 .- queue=300, inproc=100, in=100, out=200
minute=17 .- queue=300, inproc=0, in=100, out=100
minute=18 .- queue=400, inproc=0, in=100, out=0
minute=19 .- queue=500, inproc=0, in=100, out=0
minute=20 .- queue=400, inproc=300, in=100, out=200
minute=21 .- queue=300, inproc=100, in=100, out=200
minute=22 .- queue=300, inproc=0, in=100, out=100
minute=23 .- queue=400, inproc=0, in=100, out=0
minute=24 .- queue=500, inproc=0, in=100, out=0
minute=25 .- queue=400, inproc=300, in=100, out=200
minute=26 .- queue=300, inproc=100, in=100, out=200
minute=27 .- queue=300, inproc=0, in=100, out=100
minute=28 .- queue=400, inproc=0, in=100, out=0
```

```
minute=29 .- queue=500, inproc=0, in=100, out=0
```

The recommendation is to run the non-sales-script with a delay after the process completes.

```
#!bash
while true:
do
   java nonsalessync
   sleep 30;
done
```

Which should maintain a steady and uniform load...

```
minute=0 .- queue=100, inproc=0, in=100, out=0
minute=1 .- queue=100, inproc=0, in=100, out=100
minute=2 .- queue=100, inproc=0, in=100, out=100
minute=3 .- queue=100, inproc=0, in=100, out=100
minute=4 .- queue=100, inproc=0, in=100, out=100
minute=5 .- queue=100, inproc=0, in=100, out=100
minute=6 .- queue=100, inproc=0, in=100, out=100
minute=7 .- queue=100, inproc=0, in=100, out=100
minute=8 .- queue=100, inproc=0, in=100, out=100
minute=9 .- queue=100, inproc=0, in=100, out=100
minute=10 .- queue=100, inproc=0, in=100, out=100
minute=11 .- queue=100, inproc=0, in=100, out=100
minute=12 .- queue=100, inproc=0, in=100, out=100
minute=13 .- queue=100, inproc=0, in=100, out=100
minute=14 .- queue=100, inproc=0, in=100, out=100
minute=15 .- queue=100, inproc=0, in=100, out=100
minute=16 .- queue=100, inproc=0, in=100, out=100
minute=17 .- queue=100, inproc=0, in=100, out=100
minute=18 .- queue=100, inproc=0, in=100, out=100
minute=19 .- queue=100, inproc=0, in=100, out=100
minute=20 .- queue=100, inproc=0, in=100, out=100
minute=21 .- queue=100, inproc=0, in=100, out=100
minute=22 .- queue=100, inproc=0, in=100, out=100
minute=23 .- queue=100, inproc=0, in=100, out=100
minute=24 .- queue=100, inproc=0, in=100, out=100
minute=25 .- queue=100, inproc=0, in=100, out=100
minute=26 .- queue=100, inproc=0, in=100, out=100
minute=27 .- queue=100, inproc=0, in=100, out=100
minute=28 .- queue=100, inproc=0, in=100, out=100
minute=29 .- queue=100, inproc=0, in=100, out=100
```

Threads

We currently have 20 threads in the non-sales-sync. As stated earlier, 97% of the volume are warranty records which are rejected over and over. If we resolve that issue it should be possible to reduce the number of threads. (e.g. 8 or 10)That will prevent the frequency of the of the timeouts and deadlock issues. It will maintain a steady and uniform flow. The processing time is not really directly proportional to the number of threads as the bulk of the load is on the database.

Summary of recomendations.

1- Only process warranty records within a time window. (e.g. Last week, Last month or last three days)

2.- Reduce the number of threads and reevaluate to determine the optimal value.

3.- Continuously executing the job after a small delay will create a steadier and uniform flow.