From: Spitzer, Andras ([redacted])
Sent: Wednesday, April 14, 2010 11:37 AM
To: [redacted] ([redacted])
Cc: [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted])
Subject: RE: I/O profiling on cseloukpuapp13

Anders,

Let me give you a heads up about what I found since my last e-mail. I created a separate environment where I can analyze the behaviour of the batch program, and if I see an option to be safe to try on the production, I make the changes.

Today's run is still running, and I changed one parameter last night. I raised the V_BUFFERS value for this particular batch program. You can see the result, as it used to have ~20M memory, for example :

```
 14365 fabriken    14M   12M cpu1    27    1   0:35:43  11% runcbl/1
```

Since yesterday it's using ~440M :

```
 14565 fabriken  438M  436M cpu17    7    1   7:25:24  12% runcbl/1
```

I tested this in my separated environment, and I was happy to see that the runcbl process started to use more CPU (!), which means that the fcntl()s and pread()s are way too heavy in the process (we can consider that many syscalls as some sort of bottleneck). When I ran the batch having the default V_BUFFERS using ~20M, the CPU usage was around ~20%. When I raised the memory buffer to the amount I set yesterday, the CPU usage moved up to ~50%, which is a pretty nice increase. So increasing the V_BUFFERS does work (on test, not on production..).

I also experienced that since I raised the V_BUFFERS, the I/O workload substantially decreased (!), again that's a brilliant point because it confirms my previous theory that not the I/O is the bottleneck really. I barely saw a read/write from this batch process.

Please find the syscall times & summary with the original V_BUFFERS :

```
syscall               seconds   calls  errors
read                     .180   11234
write                    .019    1216
open                     .174   10023    6370
close                    .046    3654
time                     .010    1250
stat                     .077    3653
lseek                    .117   13058
fstat                    .034    3668
fcntl                   2.222  232998
pread                   3.524  193968
pwrite                   .174   11767
getcwd                   .022     910
                      --------  ------   ----
sys totals:             6.605  487399   6370
usr time:              32.869
elapsed:               62.110
```

And with the new, raised V_BUFFERS value :

```
syscall               seconds   calls  errors
read                     .017    1204
write                    .040    2067
open                     .014     314
close                    .014     314
time                     .016    1969
stat                     .013     262
lseek                    .029    3404
fstat                    .005     420
fcntl                    .211   19871
pread                    .196   12187
pwrite                   .049    2303
                      --------  ------   ----
sys totals:              .609   44315      0
usr time:              64.018
elapsed:               68.310
```

Even the number of syscalls is way lower, not just the time spent executing them. (you can see we have way less fcntl() and pread())

So the question is now why/how is that possible that none of these increased the performance on the production? Unfortunately doesn't have the answer yet, but I have my next theory. It's really getting proved that the bottleneck is in the application space, not in the resource space. Tonight I will enable profiling on the batch job, although I'm not sure it will show me what I'm worried about. My idea is that the reason why still we having this late finish with the job is file locking.

I realized that it's true though that the I/O pattern is similar and the number of records processed is similar, one metric does change. The amount of cumulative execution time.. Please look at the followings :

This example is the Friday when we had the good finish date :

0326/04:41
```
fabriken 14365  1.3  0.11472012304 ?        S 02:01:00 73:24 runcbl -c
/appl/etc/640_config c01p60 1
```

The process' cumulative execution time was ~73 minutes !

0330/07:14
```
fabriken  2026  2.3  0.22291220288 ?        S 02:01:00 257:40 runcbl -c
/appl/etc/640_config c01p60 1
```

A few days later, same workload, approximately same amount of records, still see the cumulative execution time, it's 257 minutes !

0405/07:38
```
fabriken 12638  1.4  0.22700824608 ?        S 02:01:00 251:55 runcbl -c
/appl/etc/640_config c01p60 1
```

On April 05, same job, same everything it's 251 minutes..

0406/08:59
```
fabriken 28676  8.7  0.11880816168 ?        O 02:01:00 300:30 runcbl -c
/appl/etc/640_config c01p60 1
```

On April 06, it's 300 minutes.

Today's run, which is still running :

```
fabriken 14565 12.3  2.8448888446080 ?        O 02:01:01 480:51 runcbl -c
/appl/etc/640_config c01p60 1
```

480 minutes of cumulative execution time!

So the time difference why the batch job finish sometimes in time or sometimes late is really inside the application and it doesn't seem to be a resource issue. My current assumption (which is again, a theory only yet) is that this means file locking. The reason why process is running late is because the other runcbl commands are locking the same files this job is using, so it has to wait (and wait means loop and loop means execution time nside the application) a lot in order to get read/write access to these files. That also would explain the randomness of the finish date, as these differenc runcbl programs are not snychronizing who is locking first in what order, so it's really random which process have to wait how much exactly.

So the problem is that these runcbl processes are not using these data files via a centralized locking mechanism, but each separatedly using it's own fcntl to lock for read/write. Anyway, hopefully the profiling will give us new information, also I'm working on to analyze the locking as a possible issue further.

Regards,
sendai