Time (and how to get rid of it)

Gus Björklund,
Head Groundskeeper, The Parmington Foundation,

Americas PUG Challenge
Manchester, NH, USA
4-7 June 2017
Abstract

In this talk, we examine the various ways in which time is used during the execution of a transaction by multiple concurrent users. One of these is "lock latency".

We then look at how latency can be reduced to quite small intervals by careful tuning.
Notices

- Please ask questions as we go

- YMMV (Your mileage may vary, transportation, meals, and accommodations not included).
"Time is what we want most, but... what we use worst."

-- William Penn
<table>
<thead>
<tr>
<th>thing</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read or write L1 cache memory</td>
<td>0.5 ns</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5 ns</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>100 ns</td>
</tr>
<tr>
<td><strong>Read 1 byte from main memory</strong></td>
<td><strong>100 ns</strong></td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000 ns</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000 ns</td>
</tr>
<tr>
<td>Round trip packet within same datacenter</td>
<td>500,000 ns</td>
</tr>
<tr>
<td>1 millisecond</td>
<td>1,000,000 ns</td>
</tr>
<tr>
<td><strong>Disk seek</strong></td>
<td><strong>10,000,000 ns</strong></td>
</tr>
<tr>
<td>Read 1 MB sequentially from network</td>
<td>10,000,000 ns</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>30,000,000 ns</td>
</tr>
<tr>
<td>Send packet CA -&gt; Netherlands -&gt; CA</td>
<td>150,000,000 ns</td>
</tr>
<tr>
<td>1 second</td>
<td>1,000,000,000 ns</td>
</tr>
</tbody>
</table>
**More numbers you should know. Trust the big B !!!**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Time (sec)</th>
<th># of Recs</th>
<th># of Ops</th>
<th>Time per op (nsec)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GL to -B</td>
<td>0.96</td>
<td>100,000</td>
<td>203,473</td>
<td>4,718</td>
<td>1</td>
</tr>
<tr>
<td>-B to FS Cache</td>
<td>10.24</td>
<td>100,000</td>
<td>26,711</td>
<td>383,362</td>
<td>81</td>
</tr>
<tr>
<td>FS Cache to SAN</td>
<td>5.93</td>
<td>100,000</td>
<td>26,711</td>
<td>222,006</td>
<td>47</td>
</tr>
<tr>
<td>-B to SAN Cache**</td>
<td>11.17</td>
<td>100,000</td>
<td>26,711</td>
<td>418,180</td>
<td>89</td>
</tr>
<tr>
<td>SAN Cache to Disk</td>
<td>200.35</td>
<td>100,000</td>
<td>26,711</td>
<td>7,500,655</td>
<td>1590</td>
</tr>
<tr>
<td>-B to Disk</td>
<td>211.52</td>
<td>100,000</td>
<td>26,711</td>
<td>7,918,834</td>
<td>1678</td>
</tr>
</tbody>
</table>

**Used concurrent IO to eliminate FS cache effects**

---

actual measurements made by Tom Bascom on customer AIX system
Test environment: ATM

- Same as the one in Secret Bunkers
  - database is about 12 GB

- Simulates ATM withdrawal transaction

- 150 concurrent users
  - execute as many transactions as possible in given time
  - result reported as "transactions per second".

- Highly update intensive
  - fetch 3 rows
  - update 3 rows
  - create 1 row with 1 index entry
our test machine

- 4 quad-core 2.4 GHz intel processors
- 64 GB memory
- 16 x 300 GB 10,000 rpm sas drives in RAID 10
- Centos 6 Linux (2.6.32-504.12.2.el6.x86_64)
- OpenEdge 11.7
- ATM 7
initial configuration

OE 11.7

database size 12 GB
150 self-serving clients

-db atm
-maxAreas 50
-omsiz 4096
-n 200
-spin 5000
-L 10240
-B 64000
-bibufs 64
let's run some tests
What is going on for 51 of 82 milliseconds?
nothing at all.
for more than half the time.
nothing at all.
for more than half the time.
what can we do about it ??
The transaction does the following (for 150 users):

0) execute 4GL code
1) fetch records from db, reading from cache
2) generate BI notes
3) update and create records
4) create index entries
5) get and release various kinds of locks
kinds of locks:
0) record locks
1) MTX lock
2) TXE lock
3) data buffer locks
4) bi buffer locks
5) latches
Latches are typically held for very short times.

maybe 100 nanoseconds

on modern computers
Lock latency:

time from when holder releases lock until waiting acquirer has locked it.

No useful work done while waiting.
Spinlock latches:

- test and set
- spin and test
- take a nap
- spin and test
- nap longer
- spin and test
- nap even longer
Tuning -napmax

- spin
- nap
- spin
- nap
- spin
- nap
The dawn rises only when the rooster crows.

Burmese proverb
-spin 5,000 vary -napmax

Value Axis

<table>
<thead>
<tr>
<th>nap 1</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>82</td>
</tr>
<tr>
<td>100</td>
<td>86</td>
</tr>
<tr>
<td>50</td>
<td>83</td>
</tr>
<tr>
<td>25</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
</tr>
</tbody>
</table>
Change -spin to 50,000
Tune –napmax again
-spin 50,000: vary -napmax
-napmax 10: vary -spin

4,545

milliseconds

(thousands)
Longer nap times => higher latch latency
Higher spin => lower latch latency
Higher contention => higher latch latency
Buffer Pool LRU Chain

- Aging "newest" to "oldest"
- Replacing "oldest" to "newest"

Hash Table
Checkpoint Queue
Database
Page Writer Queue
Every buffer access causes an LRU chain update
Can we reduce LRU chain overhead
and associated latch contention?
Tuning -Iruskip
napmax 250 (default), spin 5,000: vary lrskips

milliseconds

<table>
<thead>
<tr>
<th>none</th>
<th>5</th>
<th>10</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>250</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>86</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>88</td>
<td>87</td>
<td>89</td>
<td>88</td>
</tr>
</tbody>
</table>
napmax 250 (default), spin 50,000: vary irusksips
napmax 10, spin 50,000: vary lruskips

milliseconds

30
35
40

none 5 10 25 50 100 200 500

32 32 32 32 32 32 32 32
napmax 10, spin 50,000: vary lruskips
By tuning, we got rid of 51 milliseconds of wasted time
"Experience is a brutal teacher because she gives the test first and the lesson afterwards."

-- Vernon Sanders Law
What do we learn from all this?
0) small changes have small effects
1) sometimes big changes have small effects
2) proper use of -spin has yuuge effects
3) -spin should be higher than we thought
4) -napmax should be low
5) spin, napmax, Iruskips interact
6) Iruskips 25 to 100 seems sufficient
Want Answers

gus642@gmail.com