Exploiting Large Memory to reduce CPU resource consumption and improve application elapsed time

John Campbell
DB2 for z/OS Development
IBM Silicon Valley Lab
2nd March 2015
DB2 10/11: Tuning with Larger Memory

- **DB2 local and group buffer pools**
  - Reduction of elapsed time and CPU time by avoiding I/Os
- **Large page frames**
  - CPU reduction through better TLB efficiency
- **Thread reuse with IMS or CICS applications**
  - Reduction of CPU time by avoiding thread allocation and deallocation
- **Thread reuse and RELEASE(DEALLOCATE)**
  - Reduction of CPU time by avoiding package allocation and parent locks
  - DDF High performance DBATs support with DB2 10
  - Ability to break-in to persistent threads with DB2 11
- **Global dynamic statement cache**
  - EDMSTMTC up to 4G with DB2 11, default 110MB
  - Reduction of CPU time by avoiding full prepare
- **Local statement cache**
  - MAXKEEPD up to 200K statements with DB2 11, default 5000
  - Reduction of CPU time by avoiding short prepare
- **In-memory data cache**
  - MXDTCACH up to 512MB per thread, default 20MB
  - Reduce CPU/elapsed time with potentially better access path selection with DB2 11
Evaluating local buffer pool performance

- Use the random buffer pool hit ratio, better still use page residency time
- Unlike buffer pool hit ratio, page residency time is not affected by “extraneous” Getpage buffer hits
  - e.g., If a thread does two rapid Getpages and page releases of the same page, the second Getpage is an extraneous buffer hit, because it has no risk of incurring a sync I/O
- Random page residency time
  - Maximum of the following two formulae:
    - Ratio of VPSIZE to total # pages read/sec (including prefetched pages)
    - Ratio of VPSIZE * (1-VPSEQT/100) to random synch IO/sec
Large memory

- “Memory is cheap. CPUs are expensive”
- For every I/O that you save, you avoid the software charge for the CPU that it took to otherwise do that I/O
CPU Cost Saving by Reducing DB2 Sync I/Os

- Banking (60M account) workload with 2 way data sharing:
  - 11% response and 6% CPU reduction from 52 GB GBP to 398 GB for both members with same LBP size (60GB)
  - 40% response and 11% CPU reduction from 30GB LBP to 236GB LBP for both members with same reasonable GBP size (60GB)

![Graph showing CPU cost saving by reducing DB2 Sync I/Os]

- Graphs showing CPU time and I/O per commit vs. Local Buffer Pool size (GB) and Group Buffer Pool size (GB).
Findings Summary

- Sync I/O reduction ties closely with CPU reduction
  - This was done in z/OS 2.1, zEC12, DB2 11
  - Expect larger saving with z/OS 1.13
- For this particular workload, the higher benefit on investing on local buffer pools
  - Not all the objects become GBP-dependent
  - Large tablespaces defined with Member Cluster
  - Most of access pattern is fairly random
- GBP has to be big enough to support enough directory entries
- The best performer was with both large LBP and large GBP
IBM Brokerage Workload – Enlarging Local Buffer Pool

- 10 -> 24GB 14% CPU reduction, 3% Throughput improvement (26 -> 13 sync I/O)
- 24 -> 70GB 3% CPU reduction, 15% Throughput improvement (13 -> 8 sync I/O)
DB2 Buffer Pool Simulation - Why?

- Larger local buffer pools can potentially reduce CPU usage by reducing sync I/Os
  - The benefit depends on the size of active workload and access pattern
    - May not see any benefit if working set size is very small and already fit in the buffer pools today
    - May not see any benefit if working set is too large and increment is not large enough
  - Pages have to be re-referenced – not for one time sequential read
- Try and validate may not work well with customer workload with high variations
- Existing available tooling requires expensive set of traces and intensive analysis
Buffer Pool Simulation

- Simulation provides accurate benefit of increasing buffer pool size from production environment

- ALTER BUFFERPOOL command will support
  - SPSIZE (simulated pool size)
  - SPSEQT (sequential threshold for simulated pool)

- DISPLAY BPOOL DETAIL and Statistics Trace will include
  - # Sync and Async DASD I/Os that could have been avoided
  - Sync I/O delay that could have avoided

- Cost of simulation
  - CPU cost: approximate 1-2% per buffer pool
  - Real storage cost: approximate 2% of simulating pool size for 4K pages (1% for 8K, so on…)
    - For example, to simulate SPSIZE(1000K) 4K pools requires approx. 78MB additional real storage

- V11 APAR PI22091 for Buffer Pool Simulation now available
### Statistics Class 8 – Dataset Statistics for I/O Tuning

- **Statistics class 8 (IFCID 199)**

<table>
<thead>
<tr>
<th>BPOOL</th>
<th>DATABASE</th>
<th>SPACENAM</th>
<th>TYPE</th>
<th>GBP</th>
<th>SYNCH I/O AVG</th>
<th>ASYNC I/O AVG</th>
<th>ASY I/O PGS AVG</th>
<th>SYN I/O AVG DELAY</th>
<th>SYN I/O MAX DELAY</th>
<th>CURRENT PAGES (VP)</th>
<th>CHANGED PAGES (VP)</th>
<th>CURRENT PAGES (HP)</th>
<th>NUMBER OF GETPAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>----------</td>
<td>----------</td>
<td>------</td>
<td>-----</td>
<td>---------------</td>
<td>---------------</td>
<td>----------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>BP10</td>
<td>KAGURA24</td>
<td>TSP</td>
<td>----</td>
<td>----</td>
<td>23.35</td>
<td>0.01</td>
<td>32.00</td>
<td>8</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td>3433</td>
</tr>
<tr>
<td></td>
<td>TETHTS</td>
<td>N</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>BP11</td>
<td>KAGURA24</td>
<td>IDX</td>
<td>----</td>
<td>----</td>
<td>102.59</td>
<td>4.04</td>
<td>5.98</td>
<td>1</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td>18991</td>
</tr>
<tr>
<td></td>
<td>TETHIX1</td>
<td>N</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>74</td>
</tr>
</tbody>
</table>

- Count of Sync I/O per second
- Average Sync I/O (ms)
CPU reduction from IO Avoidance

- We have measured very wide range between 20 usec to 70 usec CPU saving on zEC12 from the various workloads with steady 70-80% CPU utilization
- z13 is 5-10% better
- The variation depends on SQL workload and technical configuration
  - # of concurrent threads
  - Access pattern
  - Dedicated CPs
  - I/O saved came from GBP dependent getpage or not
- On z13, range is 20-40 usec