Robert Catterall, IBM
rfcatter@us.ibm.com

Key Metrics for DB2 for z/OS Subsystem and Application Performance Monitoring (Part 1)

New England DB2 Users Group
September 17, 2015

The genesis of this presentation

- Mainframe DB2 people have an abundance of data fields they can look at for performance monitoring purposes
  - In DB2 monitor displays and reports
  - In z/OS monitor displays and reports
  - In various DB2 -DISPLAY commands
  - In CICS (DSNC) DISPLAY STATISTICS command output
- With all of these numbers staring back at you, you could:
  - Freeze up (sometimes referred to as “analysis paralysis”)
  - Try to analyze everything, all the time (maybe OK if you have a LOT of free time on your hands)
  - Focus too much on “FYI” and “level 2” numbers (the latter being fields that you should check if a “level 1” number is not what it should be), and overlook what’s really important

My goal

- Through this presentation, I want to help you to be more effective and efficient in monitoring DB2 subsystem and application performance
- How?
  - By spotlighting the relatively small set of metrics that are your most important indicators of good (or not) performance

Agenda

- Part 1
  - DB2 monitor-generated reports versus online displays
  - Application performance: DB2 monitor accounting reports (and displays)
- Part 2
  - Subsystem performance: DB2 monitor statistics reports (and displays)
  - The best bits in DB2 and CICS DISPLAY command output
  - Important DB2-related stuff in z/OS monitor reports and displays

DB2 monitor-generated reports versus online displays

Ongoing tuning versus putting out fires

- Many sites use their DB2 for z/OS monitor exclusively in online mode
  - Online monitoring is valuable, especially when you need to see what's happening right now in order to diagnose a performance problem
  - For in-depth, ongoing analysis of the performance "health" of a DB2 for z/OS subsystem and associated applications, I prefer to use DB2 monitor-generated reports
    - If you've only used your DB2 monitor in online mode, look into the product's batch reporting capabilities
    - In this presentation, I'll show a lot of information excerpted from DB2 monitor-generated reports – you should be able to find most of this information in online displays, as well

Generating reports with your DB2 monitor

- Usually involves executing a batch job that includes a DD statement pointing to a data set containing DB2 trace records (these records are usually written to SMF)
  - Batch job has a control statement in SYSIN, in which you specify things such as:
    - “From” and “to” dates/times
    - Report type (e.g., ACCOUNTING LONG)
    - Filtering criteria (e.g., include or exclude a DB2 plan name)
    - Report data organization options (e.g., order by connection type)

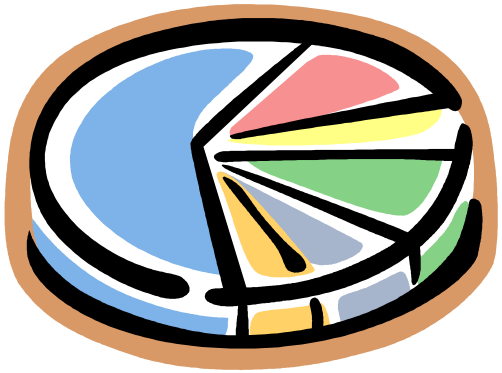
The two most useful DB2 monitor reports

- Accounting long (aka “accounting detail”), with:
  - “From” and “to” times encompassing either a busy 1- or 2-hour time period, or a 24-hour time period
  - Data ordered by (or “grouped by”) connection type
    - Gives you a detailed report for each DB2 connection type: CICS, IMS, DRDA, TSO, call attach, utility, etc.
    - If you need more granularity, can get data at correlation-name level (e.g., CICS transaction ID or batch job name), primary auth ID level, etc.
- Statistics long (aka “statistics detail”), with:
  - Same “from” and “to” times as accounting reports (see above)
- In addition to providing very useful information, these two reports are pretty inexpensive (records on which the reports are based are generated by low-overhead DB2 traces)

Application performance: DB2 monitor accounting reports (and displays)

Understanding your DB2 application workload

- What's the biggest component of your DB2 workload?
  - Seems simple enough, but I've found that plenty of DB2 people cannot readily answer this question as it pertains to their site
- "Biggest" – biggest in terms of aggregate class 2 CPU time
  - Information comes from DB2 accounting trace class 2
  - Also known as "in-DB2" CPU time
  - Indicates the CPU cost of SQL statement execution
- "Component" – connection type (e.g., CICS, batch, DRDA, etc.)



Answering the "biggest component" question

- Accounting long report, with data ordered by connection type
- For each connection type, perform a simple calculation (referring to sample report output on following slide):
  - (average class 2 CPU time) X (number of occurrences)
  - "Number of occurrences" = number of trace records
    - Usually one per transaction for online, one per job for batch
    - DB2 can "roll up" accounting records for DRDA transactions (ACCUMACC – default is 10 – and ACCUMUID parameters in ZPARM)
  - Reports generated by different monitors can look a little different
    - Samples in this presentation are from reports generated by IBM's Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS
    - Fields in reports can usually be found in online monitor displays
  - Note: I'm leaving out some report lines and columns because putting all on a slide would require a too-small font size

Sample report output (2-hour time period)

```
CONNTYPE: DRDA

AVERAGE          DB2 (CL.2)       HIGHLIGHTS
-----------      ----------       ------------------------
CP CPU TIME        0.003614       #OCCURRENCES  :  3087344

SE CPU TIME        0.003348
```

Don't forget this! (SE = "specialty engine," which usually means zIIP)

(avg CL 2 CPU) X (# of occurrences) = 0.006962 X 3,087,344

= 21,494 seconds

In a DB2 data sharing environment, do this for each member of the group to get TOTAL DRDA SQL cost, TOTAL CICS-DB2 SQL cost, etc.

The DRDA part of the overall DB2 workload

- Often, DRDA-related activity is the fastest-growing component of an organization's DB2 for z/OS workload
- At some sites, DRDA-related activity is the largest component of the DB2 for z/OS workload – bigger than CICS-DB2, bigger than batch-DB2
  - Again, "largest" refers to total class 2 CPU time
- I have found that people – even mainframe DB2 people – are often unaware of this
  - Not uncommon for senior IT managers to think of the mainframe as just the server where the "legacy" applications run
  - In fact, the mainframe DB2 platform is evolving to become a "super-sized" (and super-available, super-secure) data server for multi-tier apps

Another important workload characteristic

- Is the DB2 workload CPU-constrained?
- A good place to check: “not accounted for” time in the DB2 monitor Accounting Long report
  - What it is: in-DB2 (i.e., class 2) elapsed time that is not CPU time, not suspension time (the latter being class 3, or “waiting for” time)
  - Basically DB2 saying, “this was time, related to SQL statement execution, that I can’t account for”
  - In my experience, usually associated with DB2 wait-for-dispatch time
    - In other words, DB2 (vs. application) tasks are not being readily dispatched
  - DB2 address spaces usually have a high priority in the system, so if not-accounted-for time is relatively high for a transactional workload, it could be that you’ve hit a processing capacity wall

DB2 not-accounted-for time (1)

```
CONNTYPE: CICS

CLASS 2 TIME DISTRIBUTION
-----------------------------------------------
CPU     |===============> 30%
SECPU   |
NOTACC  |==> 5%
SUSP    |==============================> 65%
```

- I get concerned if not-accounted-for time is greater than 10% for a high-priority transactional workload such as CICS-DB2 (or, often, DRDA)
  - Not so concerned if this time exceeds 10% for batch DB2 workload – that's not uncommon

DB2 not-accounted-for time (2)

```
CONNTYPE: CICS

AVERAGE          DB2 (CL.2)
-------------    -----------
ELAPSED TIME       0.085225 (A)

CP CPU TIME        0.025313 (B)

SE CPU TIME        0.000000 (C)

SUSPEND TIME       0.055708 (D)

NOT ACCOUNT.       0.004204
```

- If your monitor report does not have the “bar chart” elapsed time breakdown shown on the preceding slide, it will likely have a “not accounted for” field in the “class 2” time column (in red at left)
- If “not accounted for” time is not provided, calculate it yourself:
- A – (B + C + D)

What if not-accounted-for time is high?

- Add capacity (could just be an LPAR configuration change)
- If that's not feasible…
  - May see what you can do to reduce CPU consumption of the DB2 workload (more on that to come in this presentation)
  - Ensure that dispatching priorities are optimized for throughput in a CPU-constrained environment
    - IRLM should be in the SYSSTC service class (very high priority)
    - DB2 MSTR, DBM1, DIST, and stored procedure address spaces should be assigned to a high-importance service class (my opinion: somewhat higher priority than CICS AORs)
      - If system is really busy, you may need to go with PRIORITY(LOW) for CICS-DB2 transaction TCBs (relative to priority of CICS AOR main task – default is HIGH)
    - Classify DRDA transactions (in WLM policy) so they won't run as "discretionary" work

Related: are you zIIP-constrained?

```
AVERAGE          APPL(CL.1)
------------     ----------
CP CPU TIME        0.012106
 SECP CPU          0.000166   A
SE CPU TIME        0.018054   B
```

- This field shows you how much zIIP-eligible work actually ran on a general-purpose engine
- That can happen when a zIIP engine is not available at the time zIIP-eligible work is ready for dispatch

- Referring to the report snippet above, you want the percentage of zIIP-eligible CPU time consumed on a general-purpose engine to be low – calculation is A / (A + B)
  - In this case, the figure is less than 1% – that's good
  - If the figure is > 5%, I'd be concerned about contention for zIIP MIPs
    - That kind of "zIIP-eligible, but not zIIP-executed" figure might be seen on a system on which zIIP engine utilization is around 60% – or even less if a system has only one zIIP engine

How is your DB2 I/O performance?

Sample report output

```
CONNTYPE: DB2CALL
                                        A              B
CLASS 3 SUSPENSIONS    AVERAGE TIME    AV.EVENT
-------------------    ------------    --------
SYNCHRON. I/O               6.520800     6133.32
```

- Average service time for synchronous I/Os = A / B
- Times are getting to be <u>really</u> low (in this case, 1.06 ms)
  - Has much to do with advances in I/O hardware and software
  - Lowest I've seen: 0.25 ms per synch read (at site using solid state drives)
- A time > 5 ms represents opportunity for improvement
- A time > 10 ms could indicate a performance problem

How CPU-efficient are your DB2 applications?

- Usually, you're aiming to reduce A (referring to sample report below), which is in-DB2 CPU time (CPU cost of SQL statement execution)
  - Note that, sometimes, reducing A can be accomplished by increasing B (recall that "SE" is short for "specialty engine," which usually is a zIIP engine – more on this to come)

Sample accounting report output

```
AVERAGE          DB2  (CL.2)
------------     -----------
CP CPU TIME       28.311773 (A)

SE CPU TIME        0.000000 (B)
```

Average CPU time – per what and for what?

- Depends on scope of information in accounting report (specified by you)

If DRDA accounting records rolled up, number of commits is good indicator of number of transactions

- Could be average:
  - Per transaction/job for connection type (e.g., all DRDA, all call attach)
  - Per transaction for a CICS AOR (an example of a connection ID)
  - For a given batch job or CICS tran (examples of correlation names)
  - Per transaction or job for a given DB2 authorization ID
- Larger scope can be appropriate when planning change of the "rising tide lifts all boats" variety (e.g., page-fixed buffer pool)
  - Largest scope: DB2 subsystem ID

```
AVERAGE          DB2 (CL.2)
-----------      ----------
CP CPU TIME       28.311773

SE CPU TIME        0.000000
```

Information at the program (package) level

Package name

Sample report output

```
M123456B                    TIMES
-----------          -------------
CP CPU TIME          13:35.566002

SE CPU TIME               0.000000
```

- Very useful if a batch job or transaction involves execution of multiple programs
- Requires data from DB2 accounting trace classes 7 and 8

- May be LOTS of packages in the report – where do you start?
  - Your monitor may show in the Accounting Long report the top programs by elapsed time (class 7)
  - High elapsed time often points to high CPU time

```
PROGRAM NAME    CLASS 7 CONSUMERS
D789123Y       |=> 3%
M123092G       |=======> 15%
I273459Z       |> 1%
```

Application efficiency: thread reuse

(data in this report sample happens to be for a CICS-DB2 workload)

Thread reused, auth ID changed

Thread not reused

Thread reused, no auth ID change

```
NORMAL TERM.        AVERAGE
-------------      --------
NEW USER                0.79
DEALLOCATION            0.01

RESIGNON                0.20
```

- Sample above shows a thread reuse rate of 99% -- very good
- Boost CICS-DB2 thread reuse via protected entry threads for high-use trans (PROTECTNUM in DB2ENTRY RDO resource)
  - Protected thread will stick around for 45 seconds (default) after transaction completes – can be reused by another transaction associated with <u>same DB2ENTRY if plan name doesn't change</u>
- Boost IMS-DB2 thread reuse via pseudo-WFI regions

Maximizing performance benefit of thread reuse

- RELEASE(DEALLOCATE) package bind option
  - Table space locks, package sections allocated to thread retained until thread deallocation, vs. being released at commit
  - If package is executed repeatedly via the same thread, these resources won't have to be repeatedly reacquired – that improves CPU efficiency
- Can reduce CPU consumption by several percentage points
  - Biggest impact for transactions that have low in-DB2 time
  - Can also be beneficial for batch jobs that issue lots of commits
- Considerations:
  - Not good bind option for programs that get <u>exclusive</u> table space locks
  - Can impact scheduling of DDL, utilities, bind/rebind operations
    - DB2 11 NFM: use of a RELEASE(DEALLOCATE) package executed via a <u>local</u> persistent thread will be drained, and RELEASE will be temporarily changed to COMMIT, so as not to block utility or DDL or bind/rebind operation

RELEASE(DEALLOCATE) for DDF threads

- DB2 10 introduced high-performance DBATs
  - Instantiated when RELEASE(DEALLOCATE) package executed via DBAT
    - Prior releases of DB2 treated RELEASE(DEALLOCATE) packages as though bound with RELEASE(COMMIT) when executed via DBAT
  - Once instantiated, high performance DBAT will remain dedicated to instantiating connection and can be reused for 200 units of work
  - Best used for simple, high-volume DRDA transactions
  - Recommendation: bind IBM Data Server Driver or DB2 Connect packages into NULLID collection with RELEASE(COMMIT), and into another collection (e.g., NULLID2) with RELEASE(DEALLOCATE)
    - Then, via client-side data source property, point to one collection or the other to control which applications use high-performance DBATs
  - Can “turn off” high-performance DBATs (perhaps to get some ALTERs, REBINDs, etc. done) via command: -MODIFY DDF PKGREL(COMMIT)
    - Turn back on with -MODIFY DDF PKGREL(BNDOPT)

Application efficiency: GETPAGES

- #1 determinant of CPU time for DB2-accessing job/transaction
- Ways to reduce GETPAGE activity:
  - Change query access paths
    - Often involves adding indexes or modifying existing indexes
    - Might involve rewriting the query to get a better-performing access path
  - Re-cluster data
    - ALTER INDEX CLUSTER / NOT CLUSTER
    - Table-controlled partitioning: can have different clustering, partitioning keys
  - Archive/purge “cold” data, so “warm” data not so spread out in table
    - DB2 11 provides automatic data archiving capability
      http://robertsdb2blog.blogspot.com/2015/03/the-db2-managed-data-archiving-feature.html

```
TOTAL BPOOL ACTIVITY     AVERAGE
--------------------    --------
GETPAGES                  359.66
```

Application efficiency: dynamic SQL cache

- Tends to be particularly important for client-server transactions (DRDA workload) – often involve execution of dynamic SQL
  - Recall that when programs issue JDBC or ODBC calls, these are executed as dynamic SQL statements on the DB2 for z/OS server
  - CPU cost of full PREPARE of a statement can be several times the cost of statement execution
- One way to boost statement cache hits: enlarge the dynamic statement cache (it's been above 2 GB "bar" since DB2 V8)
- Also: use parameter markers (vs. literal values) in dynamic SQL statements (cache "hit" requires byte-for-byte match)

DYNAMIC SQL STMT	AVERAGE
NOT FOUND IN CACHE	A 0.26
FOUND IN CACHE	B 1.05

You want to maximize B / (A + B)

DB2 10 and dynamic statement caching

- CONCENTRATE STATEMENTS WITH LITERALS attribute of PREPARE statement (can also be enabled on DB2 client side by specifying keyword in data source or connection property)
  - If match for dynamic statement with literals not found in cache, literals replaced with & and cache is searched to find match for new statement
    - If not found, new statement is prepared and placed in the cache
- Not quite as CPU-efficient as traditional dynamic statement caching and parameterized SQL, but less costly than full prepares of dynamic statements containing literals
  - Note: may WANT optimization using literals for range predicates

```
DYNAMIC SQL STMT                AVERAGE
--------------------          --------
CSWL - MATCHES FOUND               0.24
```

Application efficiency: shifting work to zIIPs

- zIIP offload reduces cost of computing
- Options for increasing zIIP utilization:
  - For DRDA workload, if using traditional DB2 stored procedures, switch to native SQL procedures
  - If it's a batch workload, consider binding some packages with DEGREE(ANY) to enable query parallelization
    - May want to limit degree of parallelization via PARAMDEG in ZPARM (or through the DB2 10-introduced SYSQUERYOPTS catalog table)

```
AVERAGE          DB2  (CL.2)
------------     -----------
CP CPU TIME       28.311773 (A)

SE CPU TIME        0.000000 (B)
```

← Aim: reduce A by increasing B

Thanks for your time

Robert Catterall
rfcatter@us.ibm.com