Robert Catterall
IBM Senior Consulting Db2 for z/OS Specialist
rfcatter@us.ibm.com

Managing, Monitoring, Tuning and Architecting a Db2 for z/OS DDF Workload

Wisconsin Db2 Users Group
September 11, 2019

Agenda

- DDF workload trends
- Monitoring and tuning DDF and DDF-using applications
- Granular management of a DDF application workload
- Some DDF application architecture considerations

DDF workload trends

Much development of new DDF-using applications

- Fueling that trend:
  - Developer productivity: with DDF, application developers can write Db2 for z/OS-accessing code *that is not Db2 for z/OS-specific*
    - Example: non-DBMS-specific SQL interfaces such as JDBC and ODBC
    - Example: REST calls (Db2's REST interface is an extension of DDF functionality)
  - Economics: SQL statements that execute under preemptable SRBs in the DDF address space are up to 60% zIIP offload-able
    - Those would be SQL statements issued by DRDA requester applications, or invoked by REST clients, or issued by native SQL procedures called by DRDA or REST clients
  - Application vendor support: vendor-supplied applications that support Db2 for z/OS as a data server typically utilize DDF for Db2 data access

Much growth of existing DDF-using applications

- A key factor: DDF scalability
  - A single Db2 subsystem can support thousands of DDF transactions per second
    - I've seen > 4000 tps with my own eyes – check count of commits in Db2 monitor accounting long report, with data ordered by connection type, in DRDA part of report
  - A single Db2 subsystem can support up to 150,000 client application connections, up to 20,000 of which can be concurrently in-use
    - 20,000 actively in-use connections would require a lot of below-the-2-GB-bar virtual storage in DBM1 address space; 5000-10,000 should not be a problem
  - These single-subsystem numbers can be multiplied by "n" in an n-way Db2 data sharing group on a Parallel Sysplex cluster of IBM Z servers
    - "n" can be up to 32

Another DDF workload-growth booster: security

- By default, Db2 requires DDF applications to authenticate with ID and password (TCPALVER=NO in ZPARM)
- Roles + trusted contexts prevent mis-use of application's ID + password
  - Db2 privileges required by application granted to Db2 role, <u>not</u> to application's ID
  - Db2 trusted context says, "Privileges granted to role XYZ can be used by application that connects to Db2 using ID ABC, *from one of these IP addresses*" ← (app servers)
- Db2 also supports authentication via ID + certificate (vs. ID + password)
- DDF supports data encryption "on the wire" (i.e., between client, Db2)
- Same security protections available for SQL-issuing and REST clients

Also boosting DDF workload growth: availability

- DDF-using applications benefit from the qualities of service delivered by z/OS and IBM Z
- In a Db2 data sharing group, software and hardware maintenance can be applied, *and even Db2 version-to-version upgrades can be accomplished* without ever stopping the DDF application workload
- Db2 data sharing: if DDF-using application connects using IP address of *Sysplex Distributor* (IP address of data sharing group), connection will be successful as long as <u>any</u> <u>one</u> member of group is up and running

Another trend: changing nature of DDF workloads

- Db2's distributed data facility has been around for > 25 years
- Initially, DDF workloads were often of business intelligence variety
- DDF still popular for analytical applications, but over the past decade there has been a shift towards more operational, transactional, mission-critical applications utilizing DDF
  - Prime example: customer-facing applications with a mobile front-end that always have to perform well and always have to be available
- High-volume, mission-critical applications with stringent availability requirements make DDF monitoring, tuning, management and application architecture more important than ever

The focus of the remainder of this presentation…

Monitoring and tuning DDF and DDF-using applications

My preferred primary sources of DDF information

- Db2 monitor-generated statistics long and accounting long reports
  - Depending on monitor, might be called statistics detail, accounting summary long
- For accounting report, I like data to be aggregated at the connection-type level
  - Depending on monitor, might mean “ordering” or ”grouping” by connection type
- Both reports should have same FROM and TO dates/times
  - My preference is to examine a one- or two-hour period during which DDF workload activity is high
- Look at data in these reports on a regular basis, to track trends and to measure the results of performance tuning actions

Statistics report: DBAT and connection limits

(DBAT = database access thread, the kind of thread used for DDF-related work)

```
GLOBAL DDF ACTIVITY                QUANTITY
--------------------------         --------
DBAT/CONN QUEUED-MAX ACTIVE           0.00
CONN REJECTED-MAX CONNECTED           0.00
```

- If value is non-zero, you have hit the limit on connections to Db2 subsystem specified via CONDBAT parameter in ZPARM
- If connection limit reached, new connection request from client application gets an error code – you probably don't want that to happen
- Typically, at any given time most connections to a Db2 subsystem will be in inactive state
- You can use Db2 profile tables to keep a given application (or a laptop PC) from consuming too many connections (more on this to come)

- If value is non-zero, you have hit limit on in-use DBATs specified via MAXDBAT in ZPARM
- If all available DBATs are tied up, an incoming DDF transaction will be queued until a DBAT is available to service the transaction
- Client doesn't get an error code when this limit is reached, but if wait is too long then transaction might be timed out on client side
- You generally want this value to be zero, but at some sites there is a desire to have only so many DDF transactions going at one time, and some queuing is tolerated if limit is reached
- Though field is labeled "MAX ACTIVE," note that ALL DBATs (even those in pool) are active – those in-use are in connected state, and those in pool are in disconnected state

Another way to see if MAXDBAT limit reached

Output of Db2 command -DISPLAY DDF DETAIL

```
DSNL080I  @ DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I xxxxxxxx           xxxxxxx.xxxxxxxx  -NONE
DSNL084I TCPPORT=4462  SECPORT=0     RESPORT=4463  IPNAME=-NONE
DSNL085I IPADDR=::1.2.3.4
DSNL086I SQL    DOMAIN=xxxxxxx.xxx.xxx.xxx.xxx
DSNL087I ALIAS              PORT  SECPORT STATUS
DSNL088I ALIAS1             4464  0       STOPD
DSNL090I DT=A  CONDBAT=    500 MDBAT=  200
DSNL092I ADBAT=     0 QUEDBAT=      0 INADBAT=      0 CONQUED=      0
DSNL093I DSCDBAT=      0 INACONN=      0
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

This field shows cumulative number of times that MAXDBAT limit has been reached since this DDF was last started (which was probably when Db2 subsystem was last started).

We'll return to the -DISPLAY DDF DETAIL command a little later

Statistics report: DBAT reuse

```
GLOBAL DDF ACTIVITY                    QUANTITY
--------------------------             --------
 ⋮
DBATS CREATED                             34.00
DISCON (POOL) DBATS REUSED               864.6K
```

- During the reporting interval (which in this case was 2 hours), Db2 needed a DBAT about 864,634 times to service a DDF transaction (34 + 864.6K)
- 34 of those times, Db2 needed to create a DBAT; the other approximately 864,600 times, Db2 reused an existing DBAT
- That's a thread reuse rate of 99.99% – great!
- Such super-high rates of thread reuse are not unusual for DDF workloads
- If you see a DBAT reuse rate that is less than 95%, consider increasing value of POOLINAC parameter in ZPARM
- POOLINAC is limit on how long a DBAT can sit in the DBAT pool without being used
- Default is 120 seconds – if DBAT in pool has not been used in that interval, it will be terminated

Statistics report: high-performance DBATs

```
GLOBAL DDF ACTIVITY                    QUANTITY
--------------------------             --------
 ⋮
CUR ACTIVE DBATS-BND DEALLC                0.00
HWM ACTIVE DBATS-BND DEALLC                0.00
```

- These fields relate to high-performance DBATs
- "Regular" DBAT becomes a high-performance DBAT when package bound with RELEASE(DEALLOCATE) is allocated to DBAT for execution
- A high-performance DBAT, once instantiated, will not go into DBAT pool when transaction using DBAT completes
- Instead, high-performance DBAT will stay dedicated to client connection through which it was instantiated, and can be reused 200 times (terminated after 200 uses)
- Result: better performance (reduced in-Db2 CPU time), because RELEASE(DEALLOCATE) packages (and associated *table space-level* locks) stay allocated to DBAT, instead of being released and reacquired at each commit

More on high-performance DBATs

- One way to get them: if DDF application uses Db2 stored procedures (or otherwise issues static SQL statements), bind those packages that are most frequently executed with RELEASE(DEALLOCATE)
- What if you want to use high-performance DBATs with a DDF application that issues only dynamic SQL statements?
  - You are still using packages: IBM Data Server Driver or Db2 Connect packages
  - Could you bind those packages with RELEASE(DEALLOCATE)?
    - Yes, but you should NOT do that for packages in default NULLID collection, because then all DBATs would be high-performance DBATs – sub-optimal for performance
    - Instead, bind IBM Data Server Driver / Db2 Connect packages into alternate collection with RELEASE(DEALLOCATE), and point selected DDF applications to that collection (easily done via Db2 profile tables – more on this to come)

The high-performance DBAT "on/off" switch

Remember the Db2 command -DISPLAY DDF DETAIL?

```
DSNL080I  @ DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION            LUNAME            GENERICLU
DSNL083I xxxxxxxx            xxxxxxx.xxxxxxxx  -NONE
DSNL084I TCPPORT=4462  SECPORT=0     RESPORT=4463  IPNAME=-NONE
DSNL085I IPADDR=::1.2.3.4
DSNL086I SQL    DOMAIN=xxxxxxx.xxx.xxx.xxx.xxx
DSNL087I ALIAS               PORT  SECPORT STATUS
DSNL088I ALIAS1              4464  0       STOPD
DSNL090I DT=A  CONDBAT=    500 MDBAT=   200
DSNL092I ADBAT=    0 QUEDBAT=     0 INADBAT=    0 CONQUED=    0
DSNL093I DSCDBAT=      0 INACONN=      0
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

To get high-performance DBATs, you need RELEASE(DEALLOCATE) packages and DDF has to be enabled for high-performance DBATs

To enable high-performance DBATs, DDF PKGREL setting needs to be BNDOPT

PKGREL = COMMIT means high-performance DBAT functionality is "off" for this Db2

Flip switch with Db2 command -MODIFY DDF

More on high-performance DBAT on/off switch

- -MODIFY DDF PKGREL(BNDOPT): high-perf DBAT functionality “on”
- -MODIFY DDF PKGREL(COMMIT) turns high-perf DBAT functionality “off”
- Why would you ever want to turn this functionality off?
  - Might need to to get certain database administration things done
    - Cannot rebind/replace/invalidate package when in-use, and RELEASE(DEALLOCATE) package allocated to persistent thread (persists through commits) in-use for life of thread
    - If you need to turn off high-performance DBAT functionality, issue -MODIFY DDF PKGREL(COMMIT), and wait a couple of minutes for existing high-performance DBATs to be terminated (DDF workload keeps going, using “regular” DBATs)
    - Do Db2 database administration work that might have been blocked by high-performance DBATs, then turn functionality back on via -MODIFY DDF PKGREL(BNDOPT)
    - **Note:** Db2 12 rebind phase-in functionality (M505 function level) should be helpful here

We’ll talk more about high-performance DBATs shortly…

Statistics report: dynamic statement cache

```
DYNAMIC SQL STMT                QUANTITY
---------------------------     --------
PREPARE REQUESTS                 5041.4K
  FULL PREPARES                77497.00
  SHORT PREPARES                 4963.8K
GLOBAL CACHE HIT RATIO (%)         98.46
```

- Relevance not exclusive to DDF applications, but particularly pertinent to such applications as they often involve issuance of mostly – sometimes entirely – dynamic SQL statetments
- "Short prepare" ("hit" in dynamic statement cache) can be two orders of magnitude cheaper than full prepare (CPU cost)
- Check global cache hit ratio – if < 90% and if z/OS LPAR's demand paging rate is really low (zero or < 1 per second), increase size of global statement cache to get hit ratio above 90%
- Global statement cache size specified via ZPARM parameter EDMSTMTC

Statistics report: Db2 address space CPU times

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	CP CPU TIME	PREEMPT IIP SRB
SYSTEM SVCS ADDRESS SPACE	1.598	3.060	0.226	4.885	0.374
DATABASE SVCS ADDRESS SPACE	4.362	1.857	0.918	7.137	9.364
IRLM	0.002	0.000	2.991	2.994	0.000
DDF ADDRESS SPACE	A 2.792	B 11:57.981	C 2.239	D 12:03.012	E 13:51.551

Some things to note:

- The fields labeled A, B, and C are the components of D (i.e., A + B + C = D): general-purpose CPU time
- Total DDF CPU is D + E (general-purpose CPU + zIIP CPU)
- DDF CPU time is by far the highest of any Db2 address space (not unusual) – is DDF a "CPU hog?"
  - NO – vast majority of DDF CPU reflects cost of executing SQL (think of DDF as allied address space, like CICS)
- DDF CPU reported as TCB time (A) and non-preemptable SRB time (C) is related to DDF "system" tasks
- DDF CPU reported as preemptable SRB time (B is general-purpose, E is zIIP) is related to "user" tasks (i.e., to tasks under which SQL statements issued by or invoked by DDF applications are executed)
- Note that majority of DDF preempotable SRB time is offloaded to zIIPs (i.e., E > D)

Accounting report: total cost of executing DDF SQL

```
SUBSYSTEM: DBPA                                ORDER: CONNTYPE          INTERVAL FROM: 10/05/18 09:00
                                                                                  TO: 10/05/18 10:00
CONNTYPE: DRDA
                                                CLASS 2 TIME DISTRIBUTION
                                                ---------------------------------------------------
                                                CPU     |=====> 10%
                                                SECPU   |======> 12%
                                                NOTACC  |
                                                SUSP    |=========================================> 79%

AVERAGE        DB2 (CL.2)    CLASS 3 SUSPENSIONS   AVERAGE TIME  HIGHLIGHTS
------------   ----------    --------------------  ------------  --------------------------
ELAPSED TIME     0.006914    LOCK/LATCH(DB2+IRLM)      0.000013  #OCCURRENCES     :C 863270
CP CPU TIME    A 0.000686
SE CPU TIME    B 0.000861
```

- Aggregate CPU cost of SQL statement execution for this DDF workload is (A + B) * C
  - Using numbers in this report snippet: (0.000686 + 0.000861) * 863,270 = 1335 CPU seconds
  - Explanation: A is average general-purpose CPU time, B is average zIIP CPU time, average is "per occurrence" so multiply by number of occurrences for total ("occurrence" = transaction)

Accounting report: DDF transaction rate

```
SUBSYSTEM: DBPA                                ORDER: CONNTYPE          INTERVAL FROM: 10/05/18 09:00
                                                                                   (B) TO: 10/05/18 10:00
CONNTYPE: DRDA
                                                CLASS 2 TIME DISTRIBUTION
                                                ---------------------------------------------------
                                                CPU     |=====> 10%
                                                SECPU   |======> 12%
                                                NOTACC  |
                                                SUSP    |=========================================> 79%

AVERAGE        DB2 (CL.2)    CLASS 3 SUSPENSIONS   AVERAGE TIME  HIGHLIGHTS
------------   ----------    --------------------  ------------  --------------------------
ELAPSED TIME     0.006914    LOCK/LATCH(DB2+IRLM)      0.000013   :
CP CPU TIME      0.000686                                         #COMMITS        :(A) 818417
SE CPU TIME      0.000861
```

- Commit count is good indication of the number of DDF transactions, so DDF transaction rate is number of commits (A) divided by number of seconds in report interval (B – one hour, or 3600 seconds, in this case)
  - Using numbers in report snippet: 818,417 transactions in 3600 seconds = 227 transactions/second
  - As noted previously, highest I've seen myself for one Db2 subsystem is a little over 4000 per second

Accounting report: zIIP offload for DDF workload

```
SUBSYSTEM: DBPA                                ORDER: CONNTYPE          INTERVAL FROM: 10/05/18 09:00
                                                                                  TO: 10/05/18 10:00
CONNTYPE: DRDA
                                                CLASS 2 TIME DISTRIBUTION
                                                ---------------------------------------------------
                                                CPU     |=====> 10%
                                                SECPU   |======> 12%
                                                NOTACC  |
                                                SUSP    |=========================================> 79%

AVERAGE        DB2 (CL.2)    CLASS 3 SUSPENSIONS   AVERAGE TIME  HIGHLIGHTS
------------   ----------    --------------------  ------------  --------------------------
ELAPSED TIME     0.006914    LOCK/LATCH(DB2+IRLM)      0.000013   :
CP CPU TIME    A 0.000686                                        #COMMITS        :   818417
SE CPU TIME    B 0.000861
```

- Percentage of in-Db2 CPU time (aka class 2 time – related to SQL statement execution) offloaded to zIIP engines is B / (A + B)
  - Using numbers in report snippet: 0.000861 / (0.000686 + 0.00861) = 56% zIIP offload
  - Highest that can be is 60%; anything in range of 55-60% is very good to see

Accounting report: when zIIP offload % is low

```
SUBSYSTEM: DBPP      ORDER: CONNTYPE

CONNTYPE: DRDA

AVERAGE          DB2 (CL.2)
-----------      ----------
 :
CP CPU TIME        0.004084
  STORED PRC       0.003600
 :
SE CPU TIME        0.000632
 STORED PROC       0.000067
```

- Suppose percentage of zIIP offload for a DDF workload is low (it is only 13% in the example at left)?
- In that case, look at the breakdown of average in-Db2 CPU time in the accounting long report (in the DRDA part of a report with data ordered by connection type)
- Is stored procedure in-Db2 general-purpose CPU time a large percentage of total average in-Db2 general-purpose CPU time (in this case, it is 88%)?
- If so, that indicates extensive use of external Db2 stored procedures – not zIIP-eligible because they always run under TCBs in stored procedure address spaces
- Native SQL procedures (written in SQL PL) always run under task of caller – if caller is DDF application, task is preemptable SRB in DDF address space, and SQL that runs under such a task is up to 60% zIIP-eligible
- When native SQL procedures extensively used for DDF applications, you see stored procedure zIIP time that is large percentage of total average zIIP time

Accounting report: zIIP spill-over

```
SUBSYSTEM: DBPP      ORDER: CONNTYPE

CONNTYPE: DRDA
 :
MEASURED/ELIG TIMES   APPL (CL1)
-------------------   ----------
 :
CP CPU TIME             0.000762
 ELIGIBLE FOR SECP   B  0.000001
 :
SE CPU TIME          A  0.000951
```

- "zIIP spill-over" refers to % of zIIP-eligible CPU time that ends up being consumed on general-purpose engines (happens when zIIP-eligible work is ready for dispatch but zIIP engines are busy)
- You can calculate zIIP spill-over for a DDF workload by using accounting long report fields shown at left
- Field A is average zIIP-eligible CPU time consumed on zIIP engines, and B is average zIIP-eligible CPU time consumed on general-purpose engines
- zIIP spill-over is B / (A + B)
- Using numbers in report snippet, zIIP spill-over is 0.1%
- Figure below 1% is great, 1-5% is OK, over 5% indicates undesirably high level of zIIP engine contention
- Besides external stored procedure usage (previous slide), higher zIIP spill-over % is another potential cause of reduced level of zIIP offload for a DDF workload

Accounting report: in-Db2 not-accounted-for time

```
SUBSYSTEM: DBPP                                ORDER: CONNTYPE          INTERVAL FROM: 10/02/18 09:00
                                                                                  TO: 10/02/18 10:00
CONNTYPE: DRDA
                                                CLASS 2 TIME DISTRIBUTION
                                                ---------------------------------------------------
                                                CPU     |================> 33%
                                                SECPU   |==> 5%
                                                NOTACC  |===> 7%  A
                                                SUSP    |============================> 55%

AVERAGE        DB2 (CL.2)    CLASS 3 SUSPENSIONS   AVERAGE TIME  HIGHLIGHTS
------------   ----------    --------------------  ------------  -------------------------
ELAPSED TIME     0.012242     .                                  #OCCURRENCES    :   283157
CP CPU TIME      0.004084     :
SE CPU TIME      0.000632    TOTAL CLASS 3             0.007183
```

- In-Db2 elapsed time that is not CPU time or known suspend time (often reflects wait-for-dispatch time)
- Generally speaking, for operational DDF workload you want to see < 10% (it is 7% in above example)
- > 10% may indicate CPU constraint or to lower priority for DDF and DDF applications in WLM policy
  - Compare to CICS/IMS, if applicable – if very different versus those workloads, should it be?

Priority of Db2 address spaces

- My recommendations:
  - IRLM – and only IRLM, among Db2 address spaces – should be assigned to SYSSTC (built-in z/OS service class with super-high priority)
    - IRLM uses very little CPU (as we have seen), but when it needs CPU it needs it NOW
  - Priority of DIST address space (DDF) should be same as Db2 MSTR and DBM1, and that priority should be below SYSSTC but above CICS application regions and/or IMS message regions
    - High priority of DIST pertains only to DDF system tasks, and they use very little CPU – priority of DDF-using applications determined by associated service class (next slide)
    - If MSTR and DBM1 priorities are below CICS application regions and/or IMS message regions, when system gets busy Db2 will be impeded from servicing CICS-Db2 and/or IMS-Db2 transactions, and throughput for those transactions will be negatively impacted

Priority of DDF applications, SP address spaces

- Priority of a DDF-using application is determined by service class to which application is mapped in z/OS WLM policy
  - If DDF-using applications are not mapped to a service class, they default to SYSOTHER service class – priority is DISCRETIONARY (really low)
- If you use external Db2 stored procedures (written in languages other than SQL PL): stored procedure address spaces should have same priority as Db2 MSTR, DBM1 and DIST address spaces (previous slide)
  - Priority applies to stored procedure address space's main task, not to stored procedures themselves (a stored procedure inherits the priority of its caller)
  - If a stored procedure address space has too-low priority and system is busy, there could be delays in scheduling called stored procedures for execution

Accounting report: impact of tuning actions

```
SUBSYSTEM: DBPA                          ORDER: CONNTYPE          INTERVAL FROM: 10/05/18 09:00
                                                                            TO: 10/05/18 10:00
CONNTYPE: DRDA

AVERAGE        DB2 (CL.2)    CLASS 3 SUSPENSIONS   AVERAGE TIME  HIGHLIGHTS
------------   ----------    --------------------  ------------  --------------------------
ELAPSED TIME     0.006914    LOCK/LATCH(DB2+IRLM)      0.000013  #OCCURRENCES    :   863270
CP CPU TIME    A 0.000686
SE CPU TIME    B 0.000861
```

- When you take a tuning action (e.g., use high-performance DBATs, enlarge buffer pools, tune SQL), how do you measure the resulting performance impact?
  - Get "before" and "after" Db2 monitor accounting long reports (ideally for same hour of same weekday), and compare average in-Db2 CPU time (A + B) in report snippet above
  - By what percentage did that figure improve? ⬅ This is your "done good" indicator
  - If tuning action was of "rising tide lifts all boats" variety (e.g., enlargement of several buffer pools), look at data aggregated at DRDA connection-type level
  - If a more narrow-scope action (e.g., tune SQL statement, add index to table), look at data aggregated in more-granular fashion (e.g., primary auth ID, workstation name, request location)

Accounting report: isolating REST-related activity

- Regarding identifier fields in Db2 accounting records, DRDA connection type can most accurately be thought of as “DDF-related activity”
  - Because DRDA path to Db2 from network-connected client applications and REST path both involve DDF, “connection type: DRDA” includes activity related to both
- Options for isolating REST activity in Db2 monitor accounting report:
  - If REST-using applications access Db2 using certain authorization ID, tell monitor to include information only for that ID(s) in report (command syntax varies by monitor)

```
INCLUDE(PRIMAUTH(APPXYZ))
```

  - Accounting records for REST clients have DB2_REST in correlation name identifier, so you can tell monitor to include only those records in report (again, syntax varies)

```
INCLUDE(CONNTYPE(DRDA)
  CORRNAME(DB2_REST))
```

Granular management of a DDF application workload

The key: leverage Db2 profile tables

- For a long time, you had only subsystem-wide controls for managing a DDF workload, via three ZPARM parameters:
  - CONNDBAT – limit on number of client connections to Db2 subsystem
  - MAXDBAT – limit on number of client connections that can be concurrently active
  - IDTHTOIN – limit on the time an in-use DBAT can be idle before being timed out
- As DDF workloads grew and became more diverse, organizations needed ability to manage those workloads in a more-granular way
- Db2 9 for z/OS introduced tables SYSIBM.DSN_PROFILE_TABLE and SYSIBM.DSN_PROFILE_ATTRIBUTES, and Db2 10 enabled use of those tables to manage a DDF workload

Using the Db2 profile tables

- A row inserted into DSN_PROFILE_TABLE indicates the scope of a given profile – i.e., to what part of the DDF workload will this profile apply?
  - Could be specified via..
    - …an IP address (or range of IP addresses) for an application server (or servers)
    - …an application's Db2 authorization ID (or IDs – can use wildcarding)
    - …(several other options)
- One or more rows inserted into DSN_PROFILE_ATTRIBUTES indicate what is to be done by Db2 for given profile (see next slide for examples)
- When more than one row in DSN_PROFILE_TABLE could apply to given DDF session, match based on rules concerning specificity and order of precedence – these rules are in *Managing Performance* manual:

https://www.ibm.com/support/knowledgecenter/SSEPEK_12.0.0/perf/src/tpc/db2z_systemprofileinteract.html

What can you do via profile tables?

- As noted previously, control connections, active connections and idle thread timeout for particular DDF applications or users
- Also, automatically set one or more special registers for DDF application
  - An increasingly popular action in this regard: SET CURRENT PACKAGE PATH, to point a particular DDF application to a collection – other than NULLID – in which the IBM Data Server Driver (or Db2 Connect) packages are bound a certain way
    - With RELEASE(DEALLOCATE) to get high-performance DBAT capability
    - With CONCENTRATESTMT(YES) to turn on statement concentration (Db2 12 bind option)
    - etc…
  - It will become increasingly common for Db2 sites to have 2, 3, 4 or more IBM Data Server Driver / Db2 Connect collections to enable desired application behaviors
    - More info: http://robertsdb2blog.blogspot.com/2018/07/db2-for-zos-using-profile-tables-to.html

A new (with Db2 12) use for profile tables

- Automatically set, for a DDF application, one or more of these built-in Db2 global variables:
  - SYSIBMADM.GET_ARCHIVE
  - SYSIBMADM.MOVE_TO_ARCHIVE

  Related to Db2-managed archiving
  - SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME
  - SYSIBM.TEMPORAL_LOGICAL_TRANSACTIONS

  Related to new (with Db2 12) temporal logical transaction functionality

Some DDF application architecture considerations

Two paths to DDF: SQL (i.e., DRDA) and REST

Note: these paths are not mutually exclusive – both can be used to access a given Db2 system

- Different circumstances could favor use of one path over the other

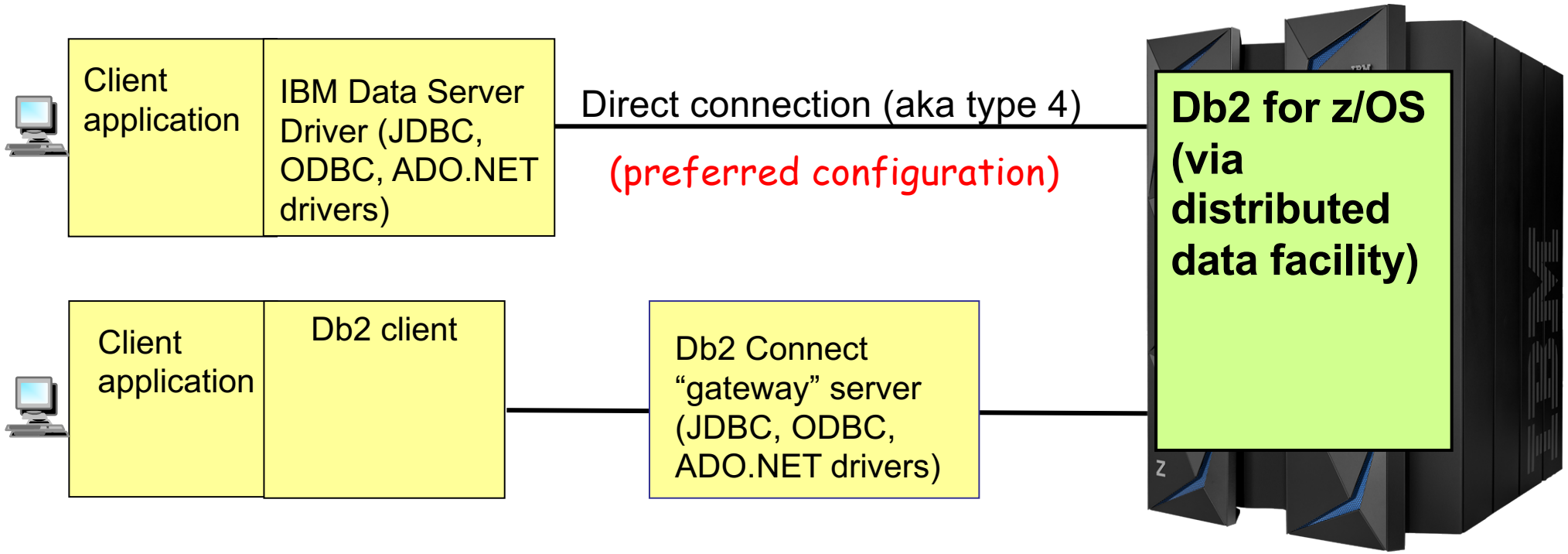
SQL path

- Db2 client software (IBM Data Server Driver or Db2 Connect) required on client side
- Leverages developers' SQL skills
- SQL statements can be dynamically constructed on client side, then issued to Db2
- Client control over transaction scope: client can issue multiple SQL statements, then commit
- Probably delivers best performance, scalability
  - Sysplex workload balancing, connection pooling, high-performance DBATs…

REST path

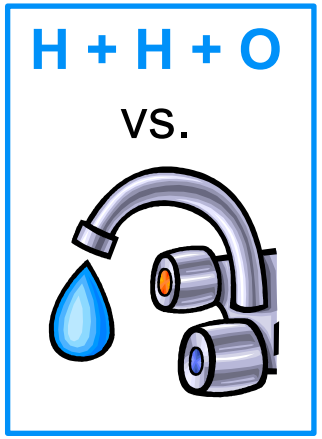
- No Db2 client-side software required for issuance of REST calls (i.e., no driver needed)
- No SQL statements issued from client side: REST calls invoke server-side static SQL statements
- No client control of transaction scope: each REST call is separate transaction from Db2 perspective
  - Stored procedures: server-side control of transaction scope
- Data-as-a-service programming model is attractive to many developers

SQL path to DDF: IBM Data Server Driver

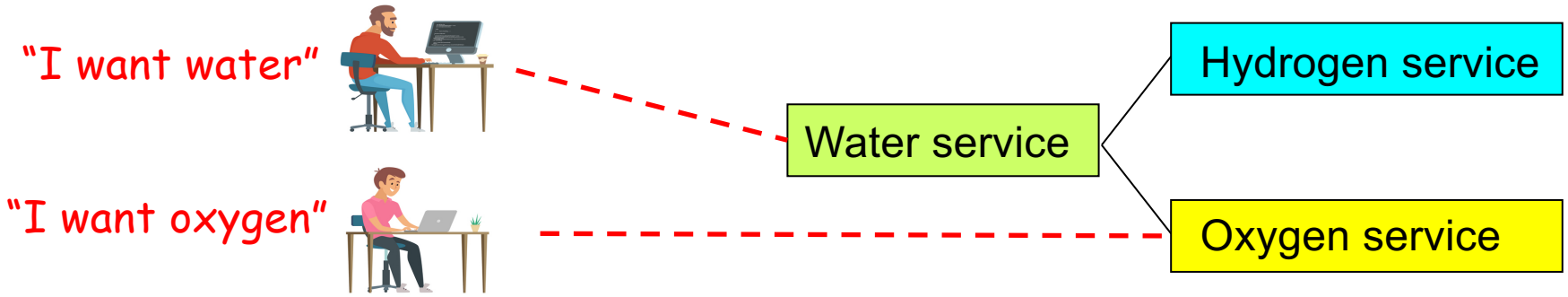


- Use IBM Data Server Driver vs. Db2 Connect "gateway" servers
  - Simplifies infrastructure, improves manageability, boosts performance
  - IBM Data Server Driver entitlement is via Db2 Connect license
    - Exception: "concurrent user" Db2 Connect license requires use of gateway server

Service granularity – a balancing act



- If granularity too fine, extra work for application developers
  - A software architect once complained that an application "makes me ask for two atoms of hydrogen and one atom of oxygen – I want water"
- If granularity is too coarse, might sacrifice flexibility with regard to combining services to create new applications
- An architecture that lets you have it both ways: coarse-grained, multi-function services that are comprised of more micro-level services, with latter directly call-able by programs wanting narrow-scope services

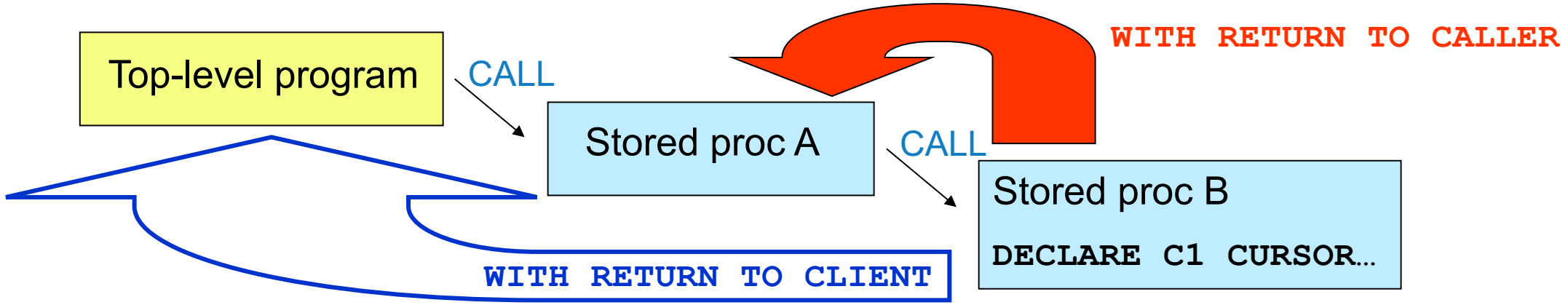


Service granularity and nested stored procedures

- Implementing a “coarse-grained-comprised-of-fine-grained” service architecture could involve use of nested stored procedures
- Db2 for z/OS supports stored procedure nesting up to 64 levels deep
- Stored procedure nesting more efficient when native SQL procedures used
  - Native SQL procedure always runs in Db2 database services address space and under task of caller, so multiple levels of nested native SQL procedures means no additional tasks – only a series of Db2 packages executed in processing a transaction
  - External stored procedure runs under its own task in stored procedure address space, so multiple levels of nesting mean more tasks that have to be scheduled for execution
    - Additionally, Db2 thread has to be switched from task of application process to task of external stored procedure – more overhead, more potential for delay

Result sets generated by nested stored procedures

- How can a query result set generated by a nested stored procedure be made available to the "top-level program" (i.e., the program that initiated the chain of nested stored procedure calls)?
  - Best option: declare cursor in nested stored procedure WITH RETURN TO CLIENT
  - That option make's cursor's result set directly FETCH-able by top-level program (default WITH RETURN TO CALLER makes result set available only "one level up")



Thanks for your time.

Robert Catterall
rfcatter@us.ibm.com