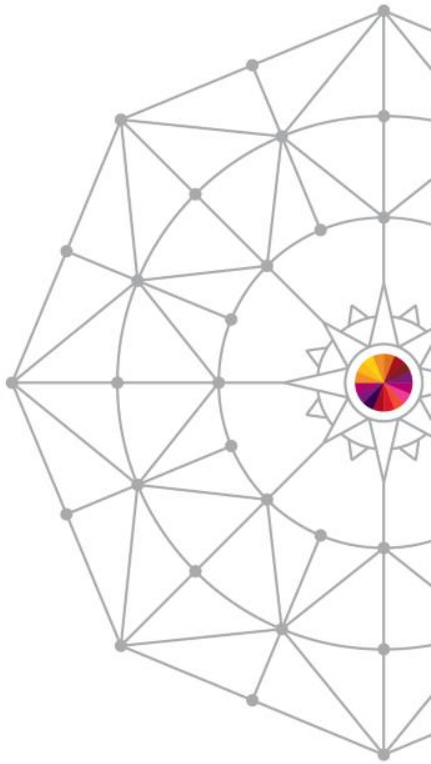
At Your Service: Optimizing DB2 for z/OS for Client-Server Applications

Robert Catterall
IBM

August 5, 2014
Session 15865



www.SHARE.org

#SHAREorg

Copyright (c) 2014 by SHARE Inc. Except where otherwise noted, this work is licensed under http://creativecommons.org/licenses/by-nc-sa/3.0/

SHARE in Pittsburgh 2014

Agenda

- The current DB2 for z/OS client-server landscape
- Optimizing DB2 for z/OS for client-server performance
- Optimizing DB2 for z/OS for client-server scalability
- Optimizing DB2 for z/OS for client-server availability
- Optimizing DB2 for z/OS for client-server security

The current DB2 for z/OS client-server landscape

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

What I mean by “DB2 for z/OS client-server”

- In the context of this presentation: an application that accesses DB2 for z/OS data via the DB2 distributed data facility (DDF)
  - There are, of course, other client-server configurations that involve DB2 for z/OS and don’t involve DDF – for example:
    - Client invokes a CICS transaction through the CICS Transaction Gateway, and CICS transaction program accesses DB2 data
    - Client places a message on a WebSphere MQ queue, and a server-side process (perhaps a CICS transaction or a DB2 stored procedure) gets that message and interacts with the DB2 database accordingly
    - Client programs run in WebSphere Application Server for z/OS, and access data in a “local” DB2 subsystem, using the type 2 JDBC driver and the RRS attach
  - These other DB2-related client-server configurations are outside the scope of this presentation

Big, and getting bigger

- My observation: client-server computing (the DDF kind) is far and away the #1 driver of DB2 for z/OS workload growth
  - DB2 accounting trace data can be aggregated by connection type
    - At many sites the DRDA connection type (DDF workload) accounts for the largest share of class 2 CPU time (SQL statement execution) – more than call attach, TSO attach, CICS, RRS attach, etc.
  - For some DB2 subsystems, over 90% of the total SQL workload is driven by DRDA requesters
  - Plenty of organizations have a DDF transaction rate that is in the hundreds per second for a single DB2 subsystem, or over 1000 per second on a DB2 data sharing group

If client-server is driving DB2 for z/OS growth…

- …what's driving client-server growth?
- There are multiple contributors:
  - Packaged applications (e.g., ERP, HR, and financial management apps)
  - Analytics applications (e.g., COGNOS, SPSS, BusinessObjects)
  - In-house-written applications
- For these applications, DB2 for z/OS is an attractive data server because:
  - It houses a lot of "source" data (often called "data of record")
  - *That data can be accessed via standard, non-DBMS-specific interfaces* (two popular examples being JDBC and ODBC)

The importance of standards

- “Modern” application developers are fine with writing DB2 for z/OS-accessing programs, as long as this doesn’t require them to do things differently versus coding for other DBMSs
- Similarly, application vendors are willing to support DB2 for z/OS as a data server if this doesn’t require a lot of change versus other data servers
- What that means for mainframe DB2 folks: if you want to grow the DB2 for z/OS workload at your site, put out the welcome mat for application developers (and application vendors)

Seeing things from a developer's perspective

- Scenario: an application written by your organization's developers has pretty severe lock contention problems with the standard DB2 for z/OS page-level locking in effect
  - You could switch to row-level locking (the standard for other DBMSs, including DB2 for LUW), or you could insist that the developers change their code to work well with page-level locking
  - If you do the former, have you "sold out?"
    - NO! Row-level locking is a feature of DB2 for z/OS, *and it's **good** that the application in question is accessing a mainframe DB2 database*
- The key, then, is to optimize DB2 for z/OS (for performance and in other ways) without making DB2 for z/OS an uninviting data-serving platform for client-server applications
  - *That's what the rest of this presentation is about*

Optimizing DB2 for z/OS for client-server performance

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Use high-performance DBATs

- Introduced with DB2 10 for z/OS (in conversion mode)
- How they're activated: a DBAT used to execute a package bound with RELEASE(DEALLOCATE) becomes a high-performance DBAT
  - DBAT will stay dedicated to connection through which it was instantiated and can be reused by 200 transactions (then it will be terminated to free up resources allocated to the thread)
- Benefit: high-performance DBATs enable you to get for DB2 client-server workloads the CPU-efficiency benefits of RELEASE(DEALLOCATE) packages + persistent threads
  - Conceptually similar to executing RELEASE(DEALLOCATE) packages with CICS-DB2 protected entry threads
  - For simple transactions, could reduce in-DB2 CPU time by 10% or more

More on high-performance DBATs

- One way to activate high-performance DBATs: bind the IBM Data Server Driver packages (or the DB2 Connect packages) with RELEASE(DEALLOCATE)
  - If you do that, consider binding these packages into the standard NULLID collection with RELEASE(COMMIT), and into another collection with RELEASE(DEALLOCATE)
    - Then, by way of client-side data source property, point applications to one collection or the other – this allows you to use high-performance DBATs in a more granular fashion, vs. for all client-server applications
- Another way to activate high-performance DBATs: bind packages of DB2 stored procedures called by DRDA requesters with RELEASE(DEALLOCATE)

Use native SQL procedures

- These can boost performance in two ways
  - They provide a means of *dynamically invoking* static SQL
    - Meaning: a native SQL procedure (like an external stored procedure) can be invoked by way of (for example) a JDBC or ODBC call, and a lot of client-side developers like to use JDBC and/or ODBC
    - Static SQL generally delivers optimum performance (though CPU savings vs. client-issued dynamic SQL DML may be insignificant for very simple transactions involving only 2 or 3 SQL statements)
  - If you're already using external stored procedures for DB2 client-server applications, shifting to native SQL procedures can reduce general-purpose CPU utilization by increasing zIIP offload

Use “native” SQL user-defined functions, too

- Introduced with DB2 10 in new-function mode
- “Native SQL UDFs” is my term – officially these are called compiled SQL scalar functions (or sometimes non-inline SQL scalar functions) in the DB2 documentation
- Written in SQL PL, like native SQL procedures
- Also like native SQL procedures, these UDFs run under the task of the invoking application process
  - If that process is a DDF-connected application, the z/OS task is a preemptable SRB in the DDF address space, and that makes the “native” SQL UDF zIIP-eligible
  - If a UDF appears in a correlated subquery, it could be invoked LOTS of times when the fullselect is executed
    - “Native” SQL UDF: no need to switch DB2 thread from application’s task to UDF’s task (and back) every time UDF is invoked

Have big DB2 buffer pools!

- An organization conducted an application performance test, with DB2 for z/OS versus DB2 for LUW as the data server
- Initial tests showed that throughput with DB2 for z/OS was only 25% of that seen with DB2 for LUW
  - 94% of in-DB2 time (on z/OS) was wait-for-read-I/O time
- Turned out that the DB2 for LUW server had **96 GB** of memory for buffer pools, versus **2 GB** for DB2 for z/OS
  - In that light, DB2 for z/OS performance was remarkably good
- The message: distributed systems data servers often have loads of memory – your System z server can, too, and a great use of Big Memory is big DB2 for z/OS buffer pools
  - Mainframe memory is way less expensive than it used to be, does not affect software costs, saves CPU, boosts throughput

Leverage dynamic statement caching

- Unless you've packaged "table-touching" SQL in DB2 stored procedures, likely that most of your client-server application SQL will be dynamic, due to use of JDBC and/or ODBC
  - Yes, SQLJ (static SQL in Java programs) is an option, but in my experience JDBC is much more widely used than SQLJ
- That being the case, caching of prepared dynamic SQL statements is very important for DB2 client-server application performance
  - No new news there – my message to you is:
    - If your z/OS LPAR has a lot of memory (it should – see previous slide), make your statement cache large enough to deliver a high hit ratio (I regularly see hit ratios > 90%, and I'd tell you to shoot for that)

Automatic parameterization of dynamic SQL

- A lot of client-side developers (and application vendors) know that parameterizing dynamic SQL statements is key to maximizing performance benefit of dynamic statement caching
  - Referring to ? parameter marker vs. literal values in predicates
- Sometimes you have to deal with unparameterized SQL
  - In such cases, use the CONCENTRATE STATEMENTS WITH LITERALS functionality introduced with DB2 10 for z/OS
    - Can be specified as attribute of a PREPARE statement, or in a client-side data source or connection property (JDBC or ODBC)
    - When used, DB2 checks cache for incoming statement containing a literal value, and if there's no hit then literal value is replaced by & and DB2 checks again (and prepares and caches statement if no hit)
    - Sometimes you WANT optimizer to see literal values in predicates

Get DDF-related dispatching priorities right

- Assign DDF itself to a high-priority service class
  - Same priority as DBM1, MSTR, and stored procedure address spaces (IRLM should be in higher-priority SYSSTC service class)
  - Only DDF "system" tasks will have this priority, and they typically account for very small % of total DDF workload CPU consumption
    - Though these tasks use little CPU capacity, they must be dispatched readily when they have work to do, or DDF throughput will suffer
- Assign DDF-using applications to service classes as appropriate; otherwise, they will get "discretionary" priority
  - There are a dozen or so identifiers that you can use to map particular DDF-using applications to service classes
    - DB2 auth ID, collection name, client-provided accounting information…
  - Good write-up on WLM policy set-up for a DDF workload: section 3.3.2 of IBM "redbook" titled *DB2 9 for z/OS: Distributed Functions*
    - http://www.redbooks.ibm.com/abstracts/sg246952.html?Open

Don’t over-utilize zIIP engines

- As z/OS people, we’re accustomed to driving general-purpose engines to very high levels of utilization (90% or more) while still getting excellent throughput and response times
- DO NOT run zIIP engines at such high levels of utilization
  - If zIIP engines are too busy, zIIP-eligible work can run on general-purpose CPs, but that can introduce processing delays
  - In a DB2 10 (or 11) environment, these delays can negatively impact prefetch performance, because asynchronous read-related processing became zIIP-eligible starting with DB2 10
    - That could slow prefetch-intensive applications (e.g., batch, business intelligence), with slowdown showing up as an increase in DB2 monitor-reported “wait for other read” time (accounting class 3)
- I’ve seen zIIP-eligible workloads run well with zIIP utilization at about 60%, but I wouldn’t want to go much over that level

Optimizing DB2 for z/OS for client-server scalability

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Have plenty of connections

- CONDBAT in ZPARM (aka MAX REMOTE CONNECTED)
  - Value can be up to 150,000
  - **Important:** CONDBAT > MAXDBAT, CMTSTAT = INACTIVE (so connections can go inactive when DDF transactions complete, and DBATs can be disassociated from connections and pooled)
    - Referring here to “regular” DBATs – high-performance DBATs remain associated with their instantiating connections
  - Probably want CONDBAT value to be at least 2X the MAXDBAT value (small % of connections likely to be active at one time)
  - How you can tell if your CONDBAT value is too small (reaching limit causes subsequent connection requests to be rejected):
    - Check DB2 monitor-generated statistics long report (or online display), and find the DDF ACTIVITY section
    - Look for field with a label like CONN REJECTED-MAX CONNECTED – if the value is non-zero, you should probably increase your CONDBAT value

Have plenty of DBATs

- MAXDBAT in ZPARM (also known as MAX REMOTE ACTIVE)
  - Starting with DB2 10, can be set as high as 19,999 (though practical limit is probably below that number)
    - My recommendation: MAXDBAT + CTHREAD <= 10,000
    - This assumes that packages have been rebound in DB2 10 (or 11) environment – in that case almost all thread-related storage will be above the 2 GB bar in DBM1
    - If you increase your MAXDBAT value, increase CONDBAT, too
  - If you want to use high-performance DBATs, may want to substantially increase MAXDBAT value first (like maybe double it)
    - A high-performance DBAT, once instantiated, does not go back into the DBAT pool; thus, more high-performance DBATs means fewer DBATs in the pool if you don't increase MAXDBAT
    - In DB2 monitor-generated statistics long report, compare a field with a label like HWM ACTIVE DBATS-BND DEALLC to your MAXDBAT value (if former is near latter, boost MAXDBAT value)

Have plenty of zIIP MIPs

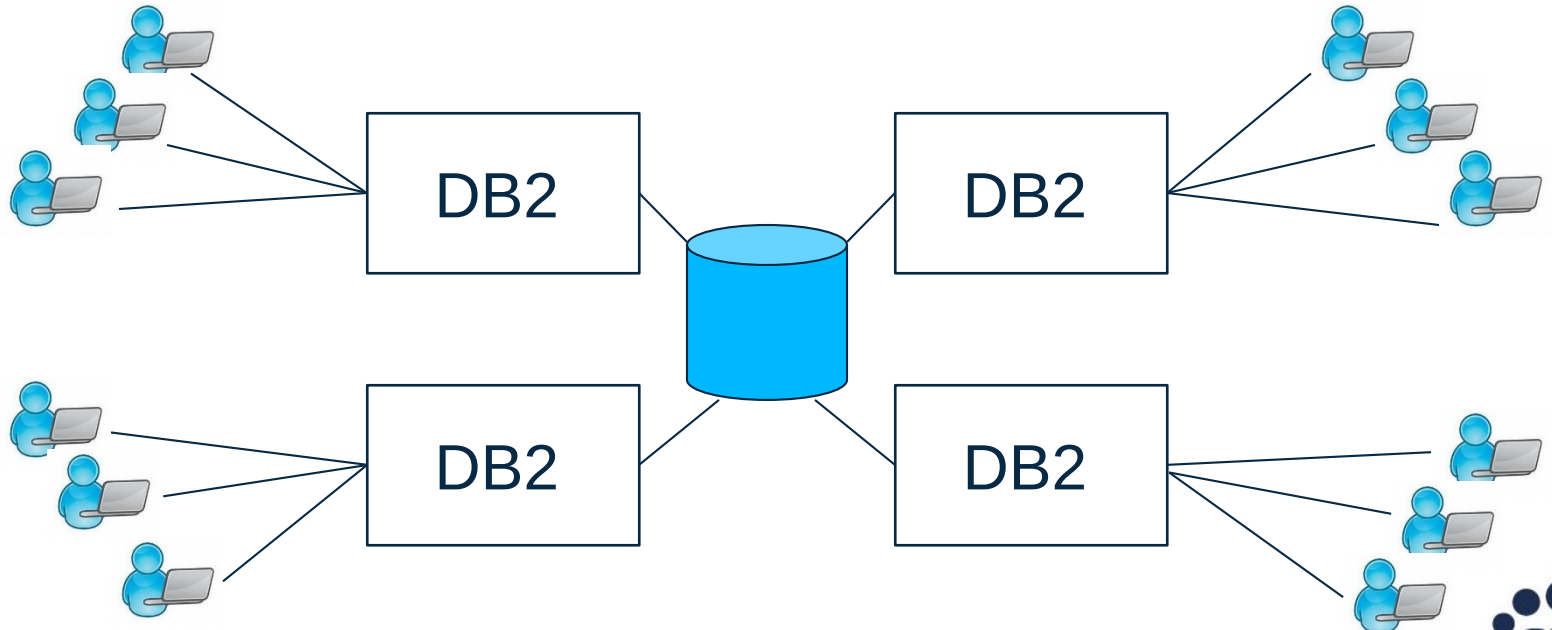
- This is a performance thing, but also a scalability thing
- At one time, you could only get 1 zIIP engine for every general-purpose engine on a System z server – now you can get 2
  - On zEC12 or zBC12, since July 23, 2013
- The more heavily weighted your workload is towards DB2 DDF client-server, the more zIIP vs. GCP capacity you want
- Remember, it's not just DDF-related activity driving zIIP utilization – other use is increasing
  - DB2 10: RUNSTATS, prefetch reads, database writes, XML schema validation
  - DB2 11: log I/Os, pseudo-deleted IX entry clean-up, utilities
  - And, I'm seeing more WebSphere on z/OS – think zAAP-on-zIIP
  - Ensure that zIIP capacity stays ahead of zIIP demand

Use type 4 JDBC/ODBC drivers

- Straight from application server to DB2 – not via gateway
  - Simpler, more robust, more scalable infrastructure
  - Better performance (eliminates “hop” to DB2 Connect gateway)
- Another recommendation: use IBM Data Server Driver vs. DB2 Connect (if you’re licensed for latter you can use former)
  - Lighter-weight client
  - Easier upgrade to new maintenance levels and new releases
  - Not just JDBC, ODBC: also SQLJ, .NET, CLI, OLE DB, PHP, Ruby, Python, Perl
- WAS for z/OS to local DB2: use type 2 or type 4 JDBC driver?
  - Type 2: more CPU-efficient access to DB2 (could be good choice for transactions that issue quick-running SQL statements)
  - Type 4: more zIIP utilization for SQL statement execution (good for longer-running SQL statements, native SQL procedures)

Leverage DB2 data sharing technology

- Scalability on steroids
- Up to 19,999 DBATs and 150,000 connections per member of a DB2 data sharing group



Optimizing DB2 for z/OS for client-server availability

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Beyond the availability basics (1)

- "The basics" are the "motherhood and apple pie" things like:
  - Partition your larger tables
  - Regularly back up your data
  - Use online REORG
  - Yada yada yada
- Beyond these basic practices, there are other actions that can boost DB2 data availability (for all apps, not just client-server):
  - Convert non-universal table spaces to universal (for single-table table spaces, this can be done – starting with DB2 10 NFM – via ALTER and online REORG)
  - Availability angle: support for pending DDL (DB2 10 NFM), which enables non-disruptive database design changes via ALTER + online REORG (e.g., page size, SEGSIZE, DSSIZE)
    - DB2 11 adds ALTER TABLE with DROP COLUMN

Beyond the availability basics (2)

- Image copy indexes on large tables
  - Do-able since DB2 V6, but not being done at a lot of sites
  - Availability angle: for a large table, there's a good chance that RECOVER of index will be faster than REBUILD of same index
    - Also: if you lose table space AND its indexes, you can recover indexes WHILE table space is being recovered if indexes were COPY'ed
- Leverage FlashCopy support for the COPY utility (and for inline copies generated by other utilities)
  - At some sites, there is a requirement to run SHRLEVEL REFERENCE image copies of some large objects prior to the start of a batch update process
    - Run COPY with FLASHCOPY YES, and the read-only period will be much briefer
  - Another plus: can create a *time-consistent* (i.e., "non-fuzzy") image copy, even when SHRLEVEL CHANGE is specified

DB2 data sharing is great for high availability

- Drastically reduces the scope of a DB2 subsystem failure
  - If one DB2 member in the group fails, only data rows or pages that the failing member had X-locked at the time of the failure will be unavailable
    - Those retained locks will be freed when failed member is restarted
    - Restart is typically initiated automatically (on another z/OS LPAR in the Parallel Sysplex, if necessary) and will likely complete *faster* than would be the case in a standalone DB2 environment
- Virtually eliminates the need for planned DB2 outages
  - Example: apply DB2 maintenance without ever stopping the application workload

More DB2 data sharing: DVIPAs and DDVIPAs

- Define a dynamic virtual IP address (DVIPA) for each member of the DB2 data sharing group
  - If a member fails and is restarted on another z/OS LPAR in the Parallel Sysplex, requesters utilizing DRDA 2-phase commit protocol will be able to find it (to resolve in-doubt DBATs)
- Define a distributed DVIPA (DDVIPA) for Sysplex Distributor (a z/OS component), make that data sharing group's IP address
  - That way, an initial client request to connect to the data sharing group will succeed, as long as at least 1 DB2 member is active
- Put IP address for each DB2 member (and group's IP address) in the member's BSDS, not in a PORT reservation statement
  - Lets DB2 member accept a request to its port on any IP address
  - Also, required for DRDA SSL support (more on this to come)

Data sharing and "granular" high availability

- Scenario: you have a DB2 data sharing group, and you want to restrict an application to a subset of the group's members
  - Example: separate SQL execution for BI and OLTP applications
- Solution: define two *location aliases* (DB2 V8 feature) by updating members' BSDS data sets (do-able via command with DB2 10)
  - One alias could map to members 1 and 2 of a 6-way group (for example), and the other could map to members 3, 4, 5, and 6
  - Have the BI application connect to the 2-member alias, and the OLTP application to the 4-member alias
- The payoff: workload separation achieved, both applications get high-availability benefits of multi-member DB2 service:
  - Failure protection, maintenance without maintenance window, connection success if any alias member is active

Optimizing DB2 for z/OS for client-server security

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Tighten security via roles and trusted contexts

- Scenario: application issues dynamic SQL DML statements, so application's DB2 authorization ID needs table access privileges (e.g., SELECT, UPDATE, DELETE)
  - Problem: it's likely that a number of people know the application's DB2 auth ID and password – could someone use those credentials to gain access to data outside the application?
  - Solution: grant the privileges needed for successful execution of the application's dynamic SQL statements to a role, not to the application's ID
    - Then create a trusted context that limits use of role's privileges to an application that connects to DB2 using a particular auth ID and runs on a particular application sever or servers (identified by IP address)

Encryption in transmission: SSL

- Use DSNJU003 utility to add secure port to the communication record of a DB2 subsystem in BSDS (or use -MODIFY DDF command if on DB2 10 or 11 and change is for a location alias)

```
DDF LOCATION=XYZ,SECPORT=448
```

- A request directed to this port will have to use SSL encryption, or the request will be rejected
- You'll want to work with your z/OS TCP/IP and security people to get this set up (there are AT-TLS, certificate considerations)
  - Good write-up in the DB2 for z/OS *Managing Security* manual (see "Encrypting your data with Secure Socket Layer support")
- DB2 trusted context can require use of SSL encryption (specify ENCRYPTION 'HIGH' in ATTRIBUTES part of trusted context)

Stored procedures: static SQL and security

- Many client-side developers like to use data access interfaces (e.g., JDBC, ODBC) that mean dynamic SQL on DB2 server
- As previously noted, successful execution of dynamic SQL DML statements requires granting table access privileges to an application process's DB2 authorization ID
- An alternative: package "table-touching" SQL statements in DB2 stored procedures – application IDs then only need EXECUTE privilege on called stored procedures
  - Stored procedures allow for dynamic invocation of static SQL
    - Client-side programmers get to use preferred data access interface
    - Server-side security people get enhanced security provided by static SQL (and they like that stored procedures reduce need to expose detailed database schema information)

Restricting access to rows and columns of a table

- Row permissions and column masks, introduced with DB2 10, can be a better choice versus “security views” for granular and selective restriction of data access
  - Row permissions: preferable to security views in a general sense
    - A key advantage: users (and programmers) continue to refer to the base table by name – no need to refer to a view name
  - Column masks also have “no need for a view name” advantage, but in some situations, security views could be preferable:
    - When access to a lot of a table’s columns is to be restricted (a single view can of course leave a lot of columns out of the SELECT list, whereas one mask restricts access to data in only one column)
    - When columns are to be “hidden,” versus merely masked (with view, a “left out” column appears to be “not there,” whereas a masked column would appear in a query’s result set, but with masked data values)

And finally, enterprise identity mapping

- Consider this situation:
  - End users have IDs that they use to authenticate to your network and to client-server applications
    - What if RACF IDs are different from users' "network" user IDs, and you don't want users to have to deal with both?
    - What if you'd like to map a set of end users to a single RACF ID?
- Enterprise identity mapping addresses these needs
  - Possible with DB2 9, became easier to implement with DB2 10
  - Leverages a RACF capability called distributed identity filters – one of these can associate a RACF user ID with one or more end-user "network" IDs
    - "Or more" association done via wild-card character in "name" part of the USERDIDFILTER portion of the RACF RACMAP command that defines the filter – can map all end-user IDs defined in a given registry to one RACF ID

A little more on enterprise identity mapping

- Besides the RACF piece, there are DB2 and application pieces of the enterprise identity mapping whole
- The DB2 piece:
  - Enterprise identity mapping works for DB2 only in the context of a trusted connection
    - Grant DB2 privileges needed by a client-server application to a DB2 role, specify RACF ID to which one or more distributed identities map in WITH USE FOR part of CREATE TRUSTED CONTEXT statement
- The application piece:
  - Application 1) gets trusted connection to DB2, and 2) requests authorization ID switch for connection (so application will pass to DB2 the distributed user identity and name of associated registry)
    - WebSphere Application Server can do this for you, and it can also be accomplished by way of APIs for Java, CLI/ODBC, and NET

Robert Catterall
rfcatter@us.ibm.com

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval