May 6-10, 2007

San Jose Convention Center

San Jose, California, USA

Session: F08

IDUG® 2007 North America

Stored Procedures: If, When and How

Konstantin Tadenev
UPS

May 8, 2007 04:20 p.m. – 05:20 p.m.

Platform: z/OS

GoFurther

Agenda

- Stored procedures or inline SQL?
  - Network traffic reduction opportunities
  - Portability
  - Reuse
  - Services potential
  - Stored procedures overhead and overhead tradeoffs
  - Application scenarios
- Stored procedures design and implementation principles

Selecting right technology building blocks for a new project can be a daunting task. In the world of distributed DB2 applications the choice quite often is between inline SQL and stored procedures. This presentation is an attempt of rationalization of this decision-making process. Hence the title's "if and "when".

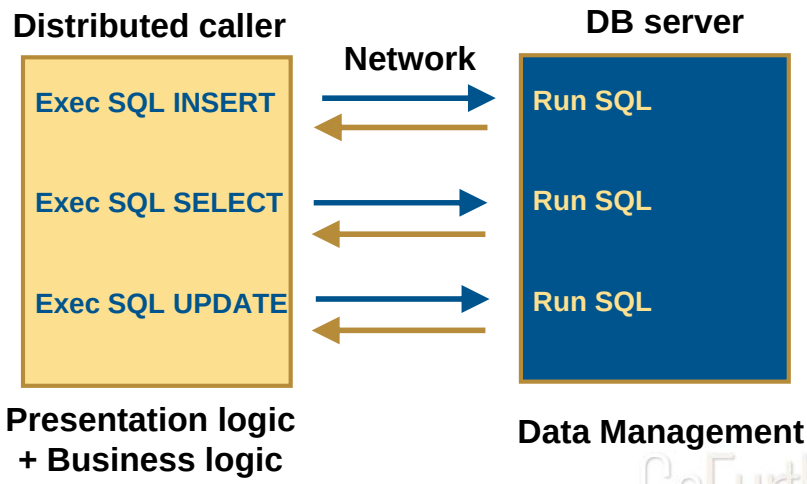
The "how" section of the presentation, also subtitled "Stored Procedures Design and Implementation Principles", is a map of some riffs and treacherous underwater currents that the author encountered while piloting many an application into the safe harbor of stability, performance and low cost. The "how" section, therefore, does not contain any step-by-step coding examples or similar routine items that can be found in manuals, but rather focuses on some hard-leaned lessons of a DB2 practitioner.

Network Traffic Reduction Opportunities

- Number of SQL calls per transaction
- Amount of data returned
- Server Topology

Network traffic reduction is the most frequently cited reason for using DB2 stored procedures. Going beyond this generic notion, we offer a more detailed look at various factors influencing effects of stored procedures on network capacity.

More than One SQL per Transaction:
Network Traffic Without Stored Procedures

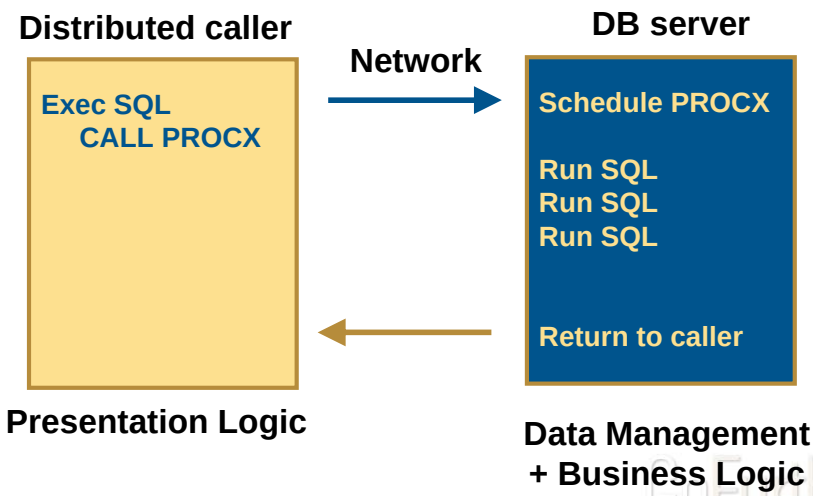


The most obvious benefit of a stored procedure comes in the case of a transaction that calls DB2 via network and contains more than one SQL statement.

The recommended solution consists of moving the SQL along with some business logic into a stored procedure.

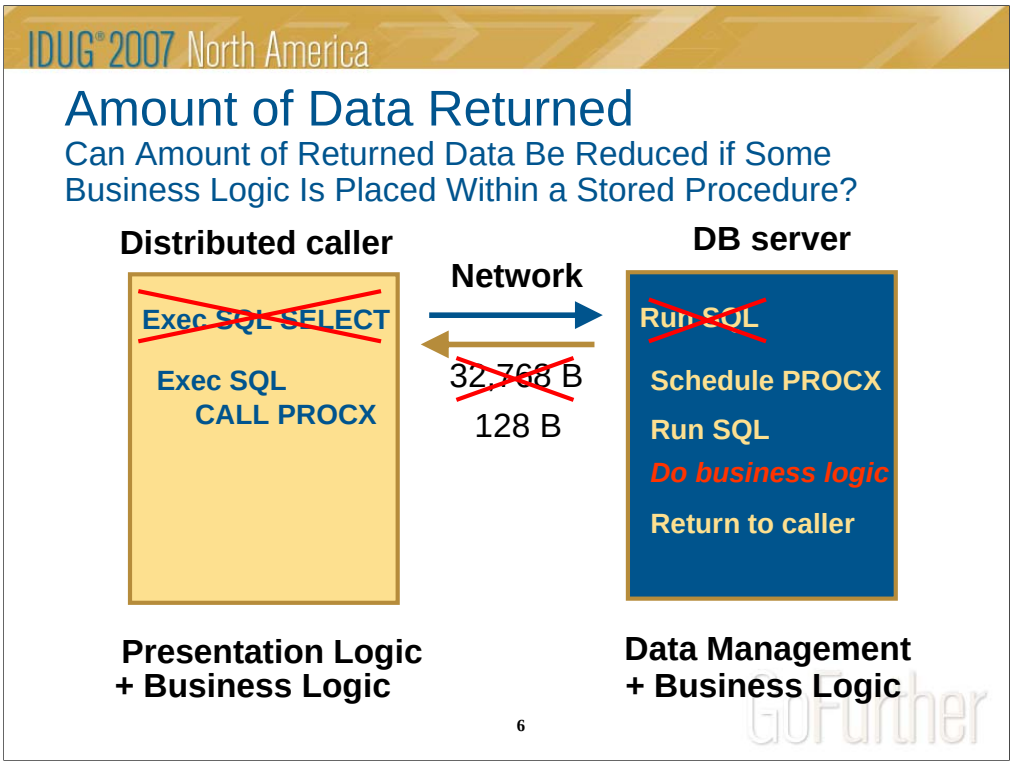
This and the next slides illustrate this idea graphically.

More than One SQL per Transaction:
Reduced Network Traffic With Stored Procedures



5

See the notes for the previous slide.



Can a stored procedure reduce network traffic if a transaction contains only one SQL?

Let's examine this example. The SQL statement in question is a SELECT returning a result set of approximately 32 K. Some business logic then transforms the result set and reduces it to 128 bytes (256 times reduction in size) before sending it downstream.

This transaction requires one and only one trip across the network regardless of whether we use inline SQL or a stored procedure, which sets the described situation apart from the previous example, where most savings are achieved by reducing the number of calls.

The opportunity lies elsewhere, i.e. in moving the data-transforming business logic into a stored procedure and running it in the database server. By doing so we reduce the amount of data sent across the network 256 times!

This happens to be a real-life case where high transaction volume made the savings truly astounding.

Network Traffic Reduction and Server Topology

- Is the application server co-located with the database server on the same z-series machine?
  - Linux on z-series
  - Various application server vendors are supported
  - HyperSockets
- Is more than one application server involved?
  - Load balancing?
  - High availability?
  - BCP?
- Is more than one database server involved?
  - Various apps?
  - DB2 Data Sharing?

These are some server topology factors that may influence the effects of network capacity savings by means of stored procedures.

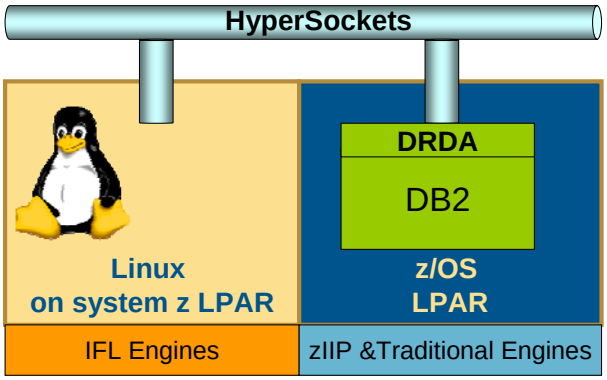
HyperSockets

- If the application servers are in zSeries LPARs, then up to 16 high speed TCP/IP networks can be created via the memory bus
- The benefits of HyperSockets:
  - Faster virtual server growth
  - Savings of adapter or cage slot costs, and simpler configuration
  - Enabling a “network in memory” at high speeds
  - Highly secure, available, and simple connection
  - Your applications are closer to DB2 and have a faster and safer connection to your DB2 data

8

As we will see below, presence or absence of HyperSockets in technical architecture may influence effects of stored procedures on savings in network capacity.

Is it Worth Calling a Stored Procedure over HyperSockets?

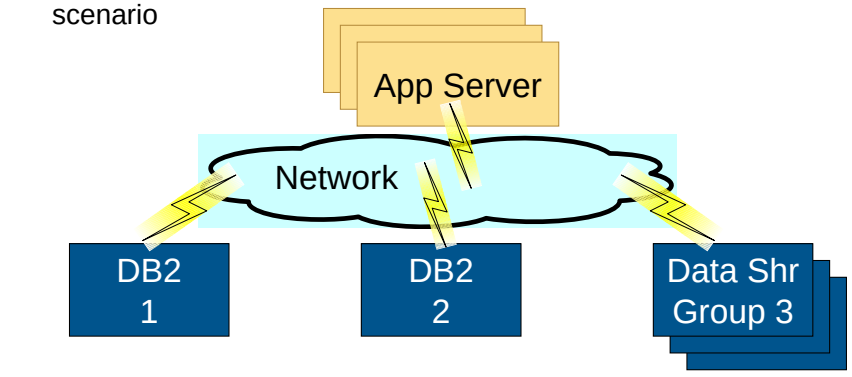


9

Fast in-memory network over HyperSockets challenges most concerns about network capacity that influence our decisions to use stored procedures. It appears that inline SQL comes out a winner over stored procedures in the topology involving one application server and one database server sharing the same z-box. But can we generalize this conclusion over more complex topologies?

Stored Procedures in Complex Topologies:
Am I Saving on Network Traffic?

- Application-level federation scenario



The following three slides examine an example of a federated application that involves multiple databases of distinct business areas running on different z-series servers.

This slide shows multiple application servers on non-z platform calling the DB2 databases via network. Obviously, this environment is opportune for stored procedures.

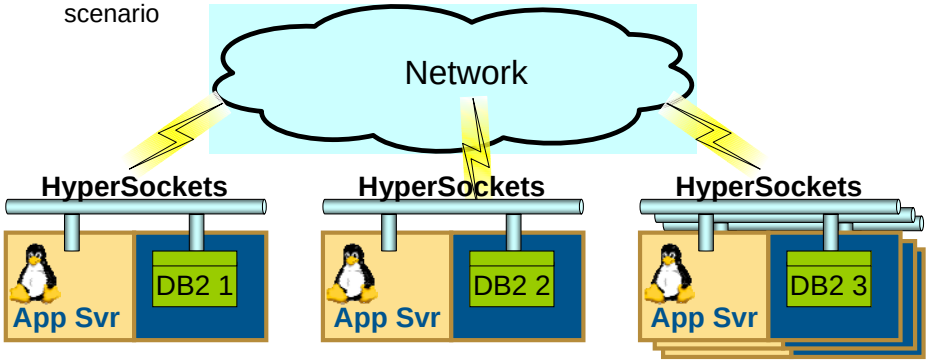
Application-Level Federation Scenario

- Each application server requires connectivity to more than one database server (a stand-alone DB2 or a data sharing group)
- The database servers reside on different z-series boxes
- All application servers support the same suite of applications, and must be configured identically
- Each database server represents a distinct business area

This slide offers some more details of the studied example.

Stored Procedures in Complex Topologies:
Am I Saving on Network Traffic?

- Application-level federation scenario

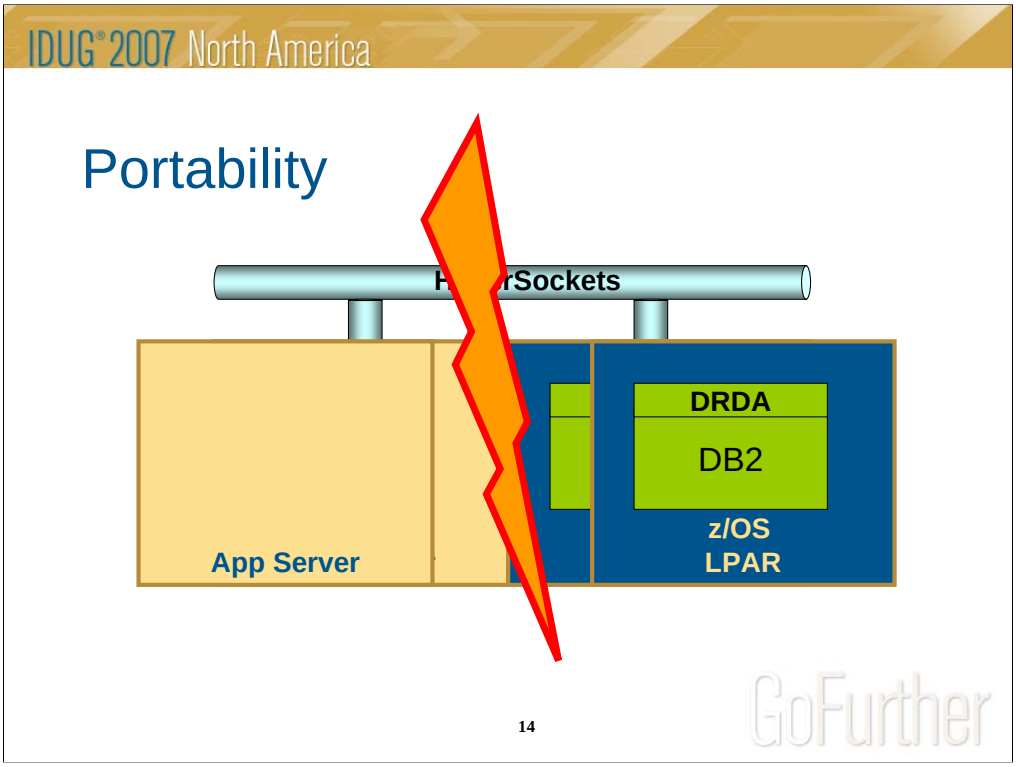


Introduction of application servers that run on z-series and can communicate to DB2 via HyperSockets changes the landscape. Here we find no obvious answer of whether or not stored procedures are preferred over inline SQL. This answer can be obtained only by means of analysis of the prospective application behavior with one key thought in mind: “Will the savings in network traffic outweigh overhead of stored procedures?”

Agenda

- Stored procedures or inline SQL?
  - Network traffic reduction opportunities
  - **Portability**
  - Reuse
  - Services potential
  - Stored procedures overhead and overhead tradeoffs
  - Application scenarios
- Stored procedures design and implementation principles

Portability as a consideration for using stored procedures.



Here we define “portability” in its narrow sense, i.e. an ability to move the caller to another platform and/or hardware without having to re-write or significantly modify the application.

Say, an application was originally designed based on the assumption that it will always be co-located with the respective database server, and will communicate to this server via HyperSockets. The designer made a conscious choice of using inline SQL for she didn’t see any benefit of using stored procedures.

If, at a later point, it is necessary to move the caller away from the database’s z-box, the cost of running the app is likely to grow rapidly, and the whole design may have to be altered in order to keep the operation cost in check.

Portability

- Portability in a narrow sense: portability of a caller
- Is this a requirement?
- Reliance on inline SQL via HyperSockets may limit platform and topology options

This slide offers further considerations on portability as a factor in deciding whether on not to use stored procedures.

Agenda

- Stored procedures or inline SQL?
  - Network traffic reduction opportunities
  - Portability
  - **Reuse**
  - Services potential
  - Stored procedures overhead and overhead tradeoffs
  - Application scenarios
- Stored procedures design and implementation principles

Reuse as a consideration for using stored procedures.

Reuse

- Incorporating data access and business rules within a stored procedure creates a potential for reuse
- Stored procedures offer a consistent way of reusing the same code base regardless of the caller's language, location, platform, etc.

Stored procedures may be used as means of reusing code including database access.

Agenda

- Stored procedures or inline SQL?
  - Network traffic reduction opportunities
  - Portability
  - Reuse
  - **Services potential**
  - Stored procedures overhead and overhead tradeoffs
  - Application scenarios
- Stored procedures design and implementation principles

Taking reuse of stored procedures one step further, we can create an entire abstraction layer between calling applications and the database. The calling applications do not need to be database-aware, database changes affect only the layer of stored procedures, but not the callers directly.

As an abstraction layer, stored procedures may be designed to play a role in SOA (Service-Oriented Architecture). The following are some benefits of SOA in high-level business terms

Business Effectiveness

– Agility, responsiveness to market and competitive dynamics

– Greater process efficiencies

– Deployment of resources based on business needs

Cost Efficiency

– Reduced maintenance costs

– Reduced skills and effort to support business change

– Price/performance optimization based on freedom to select platform, technology, and location independently

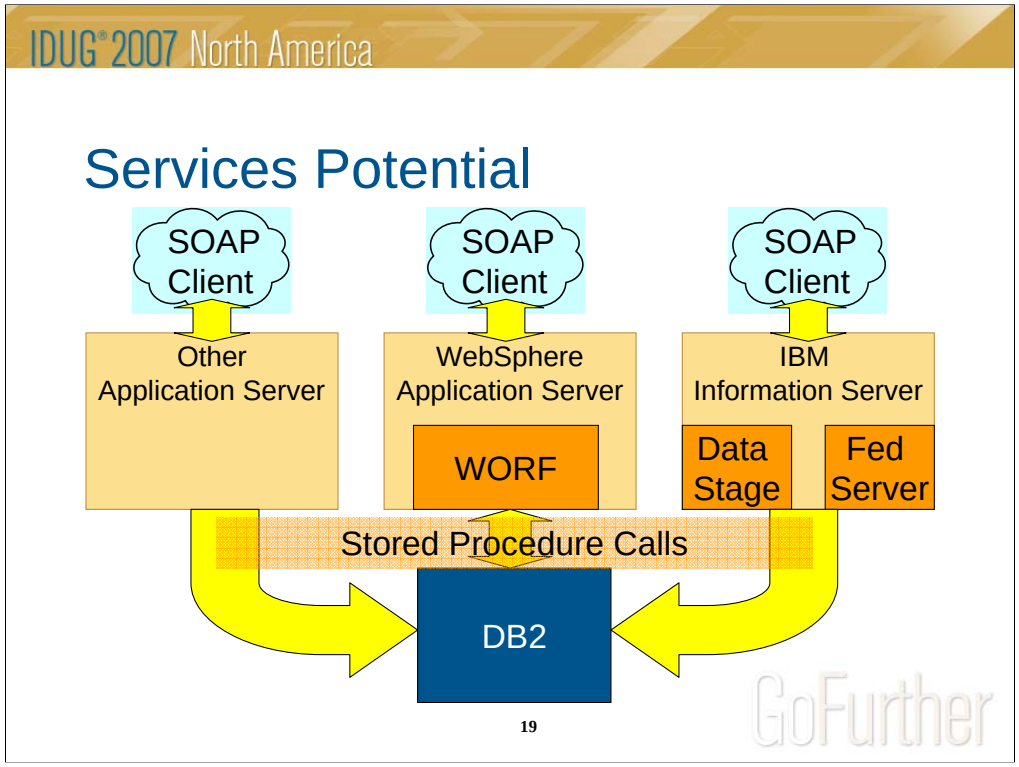
Reduced Risk

– Higher level of IT quality

– Incremental deployment

– Improved payback times*

•The Business Case for SOA: Rationalizing the Benefits of Service Oriented Architecture. Retrieved February 20, 2007, from http://www1.webmethods.com/PDF/The_Business_Case_for_SOA.pdf



Web Services provide powerful means for building SOA.

There are at least three methods of exposing a DB2 transaction as a Web Service:

1. WORF (DB2 WebSphere Object Runtime Framework)
2. Application Server facilities other than WORF
3. IBM Information Server's Data Stage and Federation Server Components

Services Potential

- Stored Procedures are required if DB2 information is exposed as Web Services via WORF (DB2 WebSphere Object Runtime Framework) under WAS (WebSphere Application Server)
- Other methods of exposing DB2 information as Web Services include:
  - Application server facilities other than WORF
  - IBM Information Server's Data Stage and Federation Server components
  - Etc.
- Application or Information Server facilities may call stored procedures or execute SQL directly against DB2
- Stored procedures, in this case, would provide a point of consistency across caller platforms

In a technologically-diverse environment stored procedures provide a point of consistency in exposing DB2 transactions as services.

Agenda

- Stored procedures or inline SQL?
  - Network traffic reduction opportunities
  - Portability
  - Reuse
  - Services potential
  - **Stored procedures overhead and overhead tradeoffs**
  - Application scenarios
- Stored procedures design and implementation principles

Considering network savings, portability, reuse or services potential, we need to keep in mind overhead of stored procedures and respective tradeoffs

Stored Procedure Overhead and Overhead Tradeoffs

- Your cost:
  - Additional 30K+ instructions per stored procedure call
  - Measure your cost in context of the overall application cost
- Your benefits:
  - Network traffic reduction
  - Portability
  - Reuse
  - Services
- Measure your benefits
  - Infrastructure cost savings (in terms of Overall Cost of Ownership)
  - Development labor savings over a specified period of time

Decision to use stored procedures should be both financially and technically sound. Forecasting and measuring the savings provides solid basis for discussions at the IT Governance level.

Overhead Tradeoffs: In Search of Compromises

- The same procedure is called over the network and locally. Is SP call overhead justified for local calls?
  - Is it really a problem? Does the peak time of local calls coincide with the peak utilization of CPU and other resources on that box? Is the overall cost of these calls significant?
  - If so, possible compromises:
    - Multiple linkedits for use as both SP and non-SP
      - Use DYNAM and entry point DSNHLI
    - Isolate shared functionality in a host-language program (COBOL, C, etc.) that can be called (linked) from an SP or non-SP module

Flexibility to employ more than one technology approach in a given application may open opportunities to lower-cost solutions.

Overhead Tradeoffs: In Search of Compromises

- You decided to use inline SQL in a high-cost Java environment (e.g. USS running under z/OS). This is a high-workload application, and you would like to address risks early in the project lifecycle
  - Measure in-line SQL cost aided by system z specialty engines
  - Measure the same SQL packaged in a stored procedure, which is coded in a less expensive host language, e.g. C, COBOL, etc.
  - Add other internal and external factors to the equation (company standards, available labor, etc.)
  - Whatever your final recommendation is, you have built a much stronger case for it, a case backed by empirical data

Another example of mixing technologies for the sake of overall technical and financial efficiency…

Agenda

- Stored procedures or inline SQL?
  - Network traffic reduction opportunities
  - Portability
  - Reuse
  - Services potential
  - Stored procedures overhead and overhead tradeoffs
  - **Application scenarios**
- Stored procedures design and implementation principles

The following slides present examples of applying network, portability, reuse, services and overhead considerations on decision to use inline SQL, stored procedures or a combination of both.

The cited scenarios illustrate thought process rather than providing “cookie cutter” solutions.

Scenario 1. Stored Procedures vs. Inline SQL in a Local Java App

- A Java app running in USS (Unix System Services) under z/OS collocated with the DB Server
- Most calls to DB2 are asynchronous
- Main concern: keeping operating cost in check
- Two types of workload:
  1. Predominantly read DB calls + heavy computation
  2. A mix of reads and writes against the DB
- ***Solution:***
  - ***A mix of SP and inline calls: COBOL SP for "1" and inline static SQL (SQLJ) for "2"***
  - ***An alternative solution would be to explore zAPP engine capabilities***

In this case, the decision was based on the measurements that proved lower operational cost of COBOL for a specific workload.

As it often happens, more than one right solution may exist. In this case, the research was focused on COBOL-JAVA comparisons. zAPP engines (IBM z-series specialty engines designated to Java cycles) became available in this installation after the application had been already built and deployed. Is new study warranted? It is a possibility, if there is a reason to believe that savings in future operational cost will outweigh cost of re-write.

Scenario 2. Portability Challenge

- The app runs in a J2EE app server in USS (Unix System Services) under z/OS collocated with the DB Server
- A back end for a browser-based app
- The database access was designed to be local
- Inline SQL was chosen over stored procedures
- New requirement: the application must be ported to a remote server
- ***Solution:***
  - ***Full replacement of the app with a vendor solution***

In this an example, the project team ruled out a possibility of a portability requirement coming at a later time, which proved to be a mistake…

Scenario 3. E-business Challenge

- An e-business order processing shared application component in J2EE framework
- Multiple application servers remote to a DB server
- Heavy transaction arrival rate (millions per day)
- A requirement of zero service interruptions in taking orders takes priority over visibility of placed orders
- Must be able to accept orders even if the DB connections are impaired or down
- ***Solution:***
  - ***SP for DB access***
  - ***Cache order validation data (pricing, etc.) at app server***
  - ***If DB connections slow or down, cache order data at app server, retry SP call later***

In this example DB2 stored procedures play a role in flexible architecture helping to solve an e-business challenge.

Scenario 4. “Fat” or “Skinny” Procedures

- A J2EE app running in multiple servers that are remote to the DB
- Most units of work are large and complex
- Options:
  1. One SP call per user event (e.g. “click”) whenever possible
  2. Smaller, more manageable SPs, typically more than one call per user event
- ***Solution:***
  - ***“1” is chosen; result – lighter network traffic, better response time and CPU time (MIPS) savings***

In this example the decision-maker applied the principal of minimizing number of SQL calls per transaction to arrive at the proposed solution. Note: multiple stored procedure calls in this case are rightfully viewed as multiple SQL calls.

Scenario 5. Why Would anyone Call Stored Procedures from a Batch Job?

- A J2EE app running in multiple servers that are remote to the DB
- Access to the DB via SPs
- A need to reuse the same code in batch
- Hint: the batch runs in a low-CPU utilization window
- ***Solution:***
  - ***Run SPs in both cases, reuse benefits outweigh additional cost***

In this example stored procedure overhead in batch processes did not affect overall operating cost, thus reuse benefits outweighed overhead considerations…

Agenda

- Stored procedures or inline SQL?
  - Network traffic reduction opportunities
  - Portability
  - Reuse
  - Services potential
  - Stored procedures overhead and overhead tradeoffs
  - Application scenarios
- **Stored procedures design and implementation principles**

This is the “how” section of the presentation. As we noted above, these are some “rules of thumb” that we developed while working on stability, performance and low cost of stored procedure applications. Thus this section does not contain any step-by-step coding examples or similar routine items that can be found in manuals, but rather focuses on some hard-leaned lessons of a DB2 practitioner.

Design and Implementation Principles

- More SQL and business logic within a procedure whenever possible
- Dynamic vs. static SQL
- JDBC Driver types
- Reduce locking
- Reduce overhead

Based on our experience, we focused on these design and implementation principals.

Dynamic vs. Static SQL

- Use static SQL whenever possible
  - Static SQL generally provides consistent and predictable performance, avoids all or most overhead of PREPARE at execution time
  - Statistics volatility, skewed data distribution or access path variance can no longer justify using dynamic SQL:
    - consider
      - REOPT (ALWAYS) or REOPT (VARS) with static SQL
      - VOLATILE on CREATE or ALTER TABLE for a table whose size may vary greatly
  - Static SQL offers
    - Better security controls
    - Validation at BIND time (hence fewer errors at run time)
    - More choices in performance monitoring

It appears that choice of static SQL for stored procedures is becoming even more attractive in DB2 v.8 and higher.

Dynamic vs. Static SQL

- Remaining good reasons to use dynamic SQL are few:
  - All or part of the SQL statement **must** be generated at execution time
  - Objects referenced by SQL are not known at BIND time
  - A vendor package contains dynamic SQL
- Java stored procedures or Java inline SQL do not necessarily imply dynamic SQL:
  - SQLJ and static SQL are preferred in these cases

Some more guidelines on static and dynamic SQL…

Type 2 vs. type 4 JDBC Drivers

- There are 4 types of JDBC drivers, but for practical purposes only type 2 and 4 should be considered
- Type 2
  - Java + native API
  - Requires DB2 client on the calling server
  - Offers performance advantages when the application and database share the same machine
- Type 4
  - Pure Java
  - Does not require a DB2 client on the calling server
  - Offers flexibility and portability advantages since it is written purely in Java
  - Is generally recommended when an application is remote to the database

High-level guidelines on choosing the right JDBC driver…

Type 2 vs. type 4 JDBC Drivers

- DB2 Universal JDBC Driver supports type 2 and type 4 implementations
  - SYSPLEX routing support:
    - Automatically balancing workload across members of a Parallel SYSPLEX based on feedback from WLM (Workload Manager)
- Other vendors offer JDBC drivers compatible with DB2

High-level guidelines on choosing the right JDBC driver…

Reduce Locking

- COMMIT ON RETURN when a stored procedure changes data
- Avoid cursors WITH RETURN against business tables. Alternative solutions:
  - A cursor WITH RETURN against a GTT (global temporary table)
  - Use of output parameters for passing data back to the caller
- Avoid writing procedures that return result sets and change data at the same time
- Use 2-phase commit in J2EE applications only when it is absolutely necessary

Most severe locking problems occur when a caller does not COMMIT or connectivity between the caller and the database server is impaired in such a way that the COMMIT cannot be processed in a timely manner. In this situation wide-spread timeouts may occur. The design principals outlined on this slide help minimizing this and many other locking-related risks.

Design Options or Environmental Variables Affecting Stored Procedure Overhead

- Examples of overhead variables:
  - Method of returning data to the caller
    - A result set via Global Temporary Table (GTT)
    - Output parameters
  - Metadata calls
    - E.g. SYSIBM.SQLPROCEDURECOLS procedure (DSNAPCOL package) executed by IBM JDBC driver
  - Application server connection pool settings

These variables may be easily overlooked while there potential in reducing application cost is quite significant.

Returning Data in an Output PARM vs. Result Set via GTT

- Workload tested:
  - 100,000 transactions ran in each test iteration
  - Average SQL counts per stored procedure call
    - 3.3 selects
    - 2 opens
    - 4.2 fetches
  - Average amount of data returned to the stored procedure caller
    - 89 bytes
  - Returning a result set via a 6-column GTT was compared to passing one VARCHAR PARM back to the caller

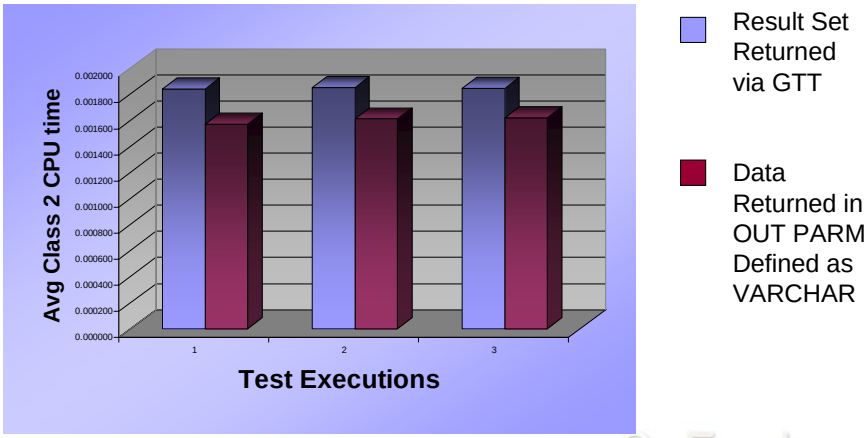
We conducted this analysis in the feasibility phase of a large-scale project in effort to come up with stored procedure design recommendations early in the project lifecycle.

This was a Web-based application with projected volume of over 2 million calls per hour. Each stored procedure call, as you can see on this slide, did relatively small amount of work. The result set passed back to the caller was also relatively small with average size of just 89 bytes.

The question was, what would be more efficient: passing the data back as a result set via a Global Temporary Table (GTT) or as a variable-length parameter?

Returning Data in an Output PARM vs. Result Set via GTT

In-DB2 CPU Time Comparison



40

This is a graphic representation of the test results.

Returning Data in an Output PARM vs. Result Set via GTT

- In our example, returning data via output parameters saved about 13% of in-DB2 CPU in comparison to the GTT-based solution
- A forecasted production volume of over 2 million calls per hour made 13% CPU savings into a financially attractive proposition

As you can see, using variable-length output parameters proved to be the best solution in this case.

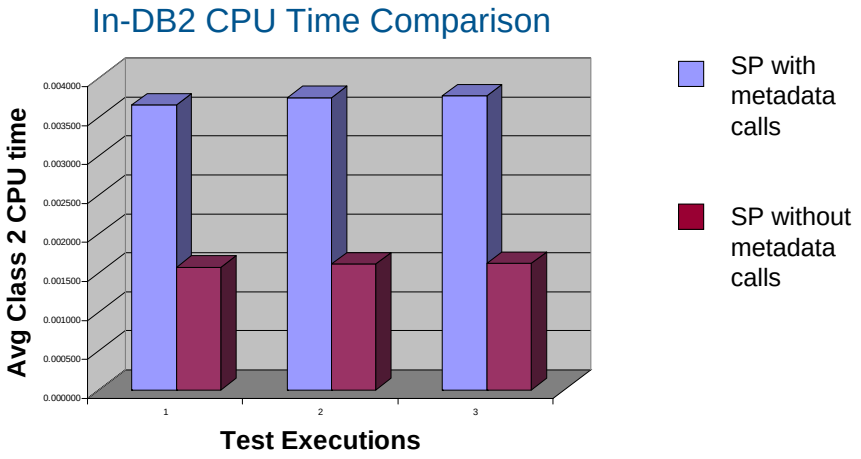
Will the result be the same with different workload and different size of returned data? The only way to find out is to model the process and take measurements as early in the project lifecycle as possible. Here we are offering a method as opposed to a one-size-fits-all solution.

Metadata Calls

- Some JDBC drivers make metadata calls on behalf of the caller
- E.g., IBM's DB2 "legacy" JDBC driver (the predecessor of the Universal JDBC driver) calls SYSIBM.SQLPROCEDURECOLS procedure (DSNAPCOL or DSNAPCO8 package)
- SYSIBM.SQLPROCEDURECOLS procedure does a join between SYSROUTINES and SYSPARMS, inserts the results into a GTT, and makes the result set available to the driver

We discovered this overhead component by scrutinizing performance reports of an application Proof-of-Concept test. A system package with high CPU consumption attracted our attention and triggered further investigation.

Overhead of Metadata Calls



The chart above illustrates importance of this discovery.

Overhead of Metadata Calls

- In this test metadata calls caused **132% of additional CPU overhead!**
- Remedy – eliminate metadata calls by
  - Upgrading to the DB2 Universal JDBC Driver
  - Or installing a third-party DB2-compatible JDBC driver that makes no metadata calls

Undoubtedly, metadata stored procedure calls should be avoided.

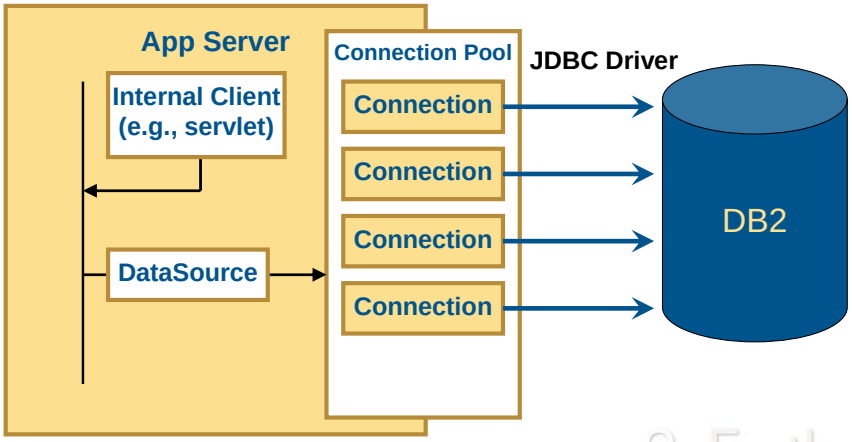
Connection Pooling

- Most Web apps execute large volumes of short transactions
- Establishing and tearing down a database connection for each of these short transactions is costly
- Connection pooling is a technique that allows reuse of an established connection infrastructure for subsequent connections
- Most application servers and some JDBC drivers offer database connection pooling

Inter-disciplinary explorations yield amazing discoveries at times. In this case, we found that some application server and/or JDBC driver settings may affect efficiency and cost of stored procedures in a profound way.

These settings pertain to database connection pooling, overview of which is offered on this and the following slides.

Connection Pooling:
An Example of an App Server Implementation



46

GoFurther

This diagram is an example of an application server connection pooling implementation[1]. We simplified the representation of the server architecture for clarity while making sure that the concept is not distorted.

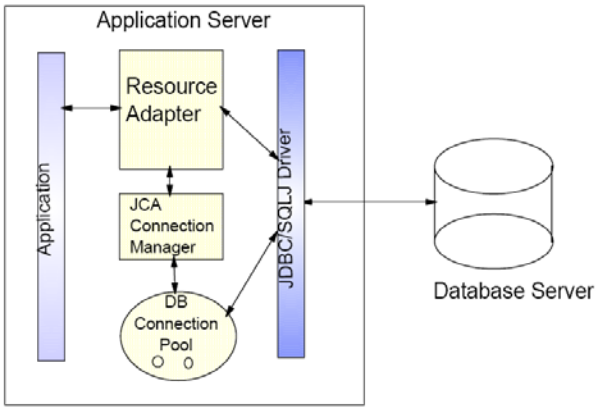
In a typical implementation, a browser communicates to a Web server, who in turn communicates to an application server. In most cases, business and presentation logic resides in the application server, thus the application server requires a database connection on behalf of an internal client such as JSP, servlet, session bean, etc. In order to obtain a connection the internal client invokes a common server component, which in this case is represented by a DataSource object. The DataSource object communicates to the Connection Pool, a server facility that allows internal clients to reuse existing database connections as opposed to building new connections on every call. The application server's connection pool calls DB2 via a JDBC driver. DB2 responds by creating a distributed thread for each active connection.

Please note that the application server's connection pool is not necessarily aware of inner workings of DB2 threads. E.g., if a DB2 distributed thread becomes inactive, reaches the idle thread timeout threshold (IDTHTOIN zParm value), and ends, the database connection would not be aware of the thread timeout, and may attempt to call DB2 over a thread that no longer exists causing a TCP/IP socket error.

---

[1]We used BEA WebLogic in this example

Connection Pooling:
Another Example of an App Server: WAS V.5

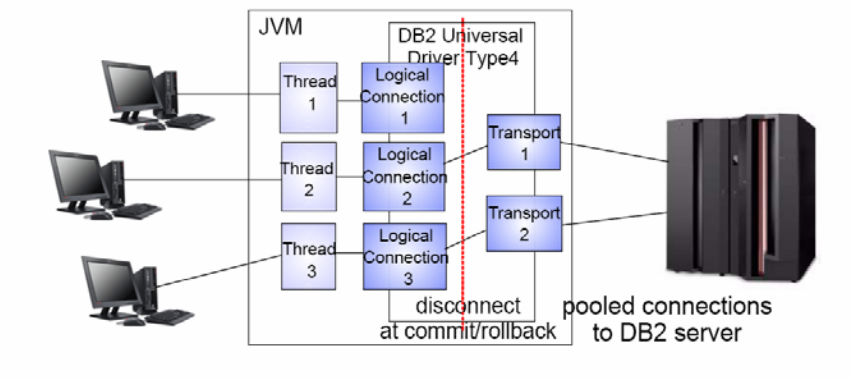


47

This slide illustrates similarity between various implementations of connection pooling within an application server. In this case the application server is WAS V.5.

Just like in the previous example, an internal client (“application” on this diagram) invokes a server component or components to connect to the database. These components may have different names and may interact with each other as well as with a JDBC driver differently, but the concept persists through various server architectures.

Connection Pooling:
DB2 Universal JDBC Driver Implementation with Connection Concentrator



48

In addition to the application servers, some JDBC drivers have their own facilities for connection pooling. This example represents DB2 Universal JDBC Driver's implementation via the Connection Concentrator facility.

Connection Pooling and "Test Connection"

- Application server database connections result in creating threads in DB2
- Transaction arrival rate may vary over time
- An application server may be set up to keep a certain number of connections live continuously
- Periodically, an SQL statement may be executed by the server as to avoid idle thread timeout in DB2
- This operation is referred to as "test connection"
- Note: "test connection" is generally not required when the DB2 Universal JDBC Driver is used for connection pooling

As we already mentioned, it is quite possible for a DB2 inactive distributed thread to time out, and an application server to call DB2 over a non-existing thread causing errors.

One of the ways to prevent this is to float a call (an SQL statement) over the established connection frequently enough to prevent idle thread timeout in DB2.

This operation is referred to as "test connection".

"Test connection" is just one of several ways to reconcile database connections with DB2 threads.

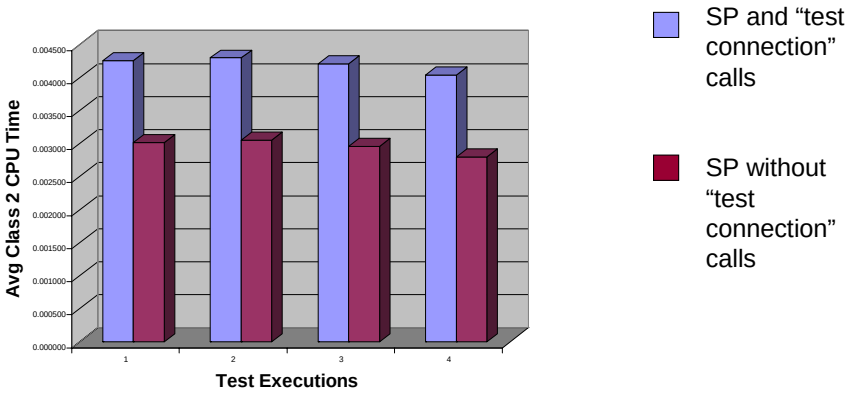
Overhead of "Test Connection"

- In some cases "test connection" may invoke any SQL statement – look out for expensive SQL
  - *Note: any SELECT without COMMIT invoked by "test connection" makes a thread ineligible to become inactive and thus not reusable!*
- If "test connection" calls are excessive, they may create substantial overhead
- In the following example "test connection" occurred for every stored procedure call

Some overhead consequences are possible if "test connection" is used.

Overhead of “Test Connection”

In-DB2 CPU Time Comparison



51

The smaller is the workload performed in each stored procedure, the greater would be the overhead of “test connection” if it is performed for each procedure call.

This chart refers to a similar workload to the ones we described earlier, i.e. just a few SQL statements per procedure call on behalf of an Internet-based application. This workload pattern is typical for many Web-based solutions.

Overhead of “Test Connection” Calls

- Over 3,500 stored procedure calls took place for each test iteration
- In this case “test connection” caused up to 42% of additional CPU overhead
- Remedies:
  - Tune application server “test connection” settings
    - E.g. “test frequency” in BEA WebLogic
  - Find another way of reconciling database connections and DB2 threads
    - Use DB2 Universal JDBC Driver capabilities
    - Tune application server idle timer setting in relation to the idle thread timeout in DB2 (make the former less than the latter)
      - A database connection should time out first before the respective DB2 thread times out

Overhead of “test connection” may vary depending on stored procedure workload, but the fact remains the same: substantial amount of database server capacity is burnt for no business reason.

Considering Overhead Variables

- Overhead variables should be studied and tested early in the project lifecycle as to enable the project team to make informed decisions on performance effects of various design choices

In your search for efficiency you may discover more overhead components that you may be able to tune. In our opinion, planning for such analysis is essential in any successful project involving stored procedures.

Session: F08

Stored Procedures: If, When and How

Konstantin Tadenev

UPS

KTadenev@ups.com

GoFurther