Implementing Connection Pools for Data-Centric Applications

Michael Rosenblum

Dulcian, Inc.

www.dulcian.com

NYOUG

September 16, 2009

Background

- Problem (discovered in mid-1990’s):
  - Keeping persistent database sessions for every client connection is technically impossible.
  - This is especially true when building scalable web-based IT solutions.
- Solution:
  - Separating logical and physical database sessions.

Physical session

- Set of activities in the context of one server connection.
- Two different approaches:
  - Full cycle:
    - Request→processing→response as a complete set
    - Starts from the moment that the request is initiated
    - Ends when the last part of the response is interpreted.
  - One-way:
    - Two completely different queues (request and response), where both events can occur independently.
    - Requests are sent without waiting.
    - A special listener retrieves responses as soon as they are ready.

Logical Session

- Set of activities between user logon and logoff that consists of a number of physical sessions.
- Each physical session is completely independent of the next/previous one.
- Developers are responsible for capturing enough information to simulate the persistence of a logical session.
- This architecture is called *stateLESS* to differentiate it from the old *stateFUL* architecture where one physical session was always equal to one logical session.

StateFUL Systems

Advantages

- Predictable and reasonable number of connections.
- Predictable resources required to keep system running.
- Possibility of using session-level features to optimize performance:
  - Temporary tables, packaged variables, etc.
  - No need to reload packages/execution plans to memory

Disadvantages

- Stateful systems do not scale well.

StateLESS Systems

Advantages

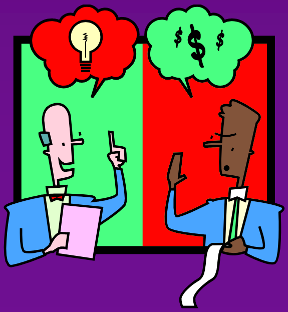
- The system can be scaled much easier.
  - At any point in time, there are only a small number of sessions connected to the database.
  - Workload typically follows a statistical trend.

Disadvantages

- Keeping a persistent layer is difficult.
  - Different schools of thought about where to place it (database/middle tier/client)
- Each physical session must be opened and closed.
  - Very expensive if done thousands of times, especially if code is PL/SQL-intensive
  - Each package must be reloaded and reinitialized.
- Difficult to manage possible unpredicted activity spikes

Cost of Building StateLESS

- Able to solve the core scalability problem
  - Possible to build systems that scale up to thousands (if not hundreds of thousands) of simultaneous users.
- High costs because:
  - Managing the persistent layer is time-intensive.
  - Significant performance impacts of the activities required to manage a huge number of separate physical requests
  - Only a low level of control over how many sessions are executed at any point in time.



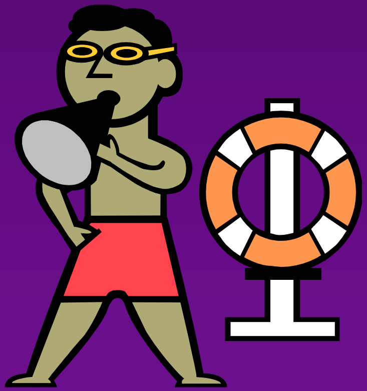
Solution: Connection Pools

- Middle tier creates a small set of physical connections to the database.
- Incoming request serves the next free session from the pool to the request (instead of opening a new session for each request).
- If all sessions are busy, the middle tier adds extra ones to the connection pool.

But…

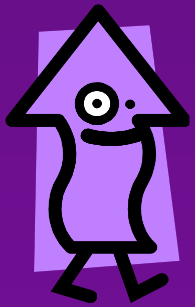
- Implementation of connection pools is challenging
  - Pool management
  - Training issues
  - Session resource management
  - Database resource management

Pool Management



Connections Upper Bound

- Delay option is recommended:
  - Request could wait for some time until a free session from the pool is found.
  - Users of web applications are accustomed to network glitches.
    - Will not be surprised by an extra few seconds of wait time
- Reason:
  - Cost of a failed request could be too high.
  - Recovery process may require a lot of manual effort
    - Each failed request should be logged.
    - If system hits an upper bound, it is either set incorrectly, or something is very wrong.



Randomization of Connection Assignments

- No randomization (done in a majority of implementations)
  - the number of sessions at any point of time is very small,
  - Workload of these sessions is very high.
  - Slightest problem either with Oracle (memory leaks still happen especially in more OO-oriented modules, like XML) or your code, and session could consume a huge amount of resources.
- Randomization:
  - Some protection from having a single very resource-intensive session
  - Makes managing total size of connection pool much more difficult.

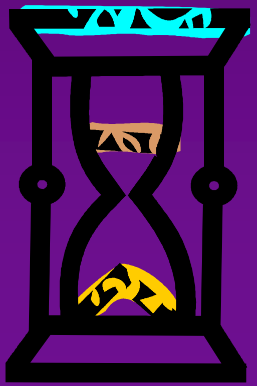
Expiration Mechanism (1)

- Applicable only for non-randomized connections
- Problem to solve:
  - Size of connection pool will reach high watermark and stay there.
- Reason:
  - Keeping sessions opened for unnecessarily long periods of time is very expensive, because of locking many database resources.
  - PGA/UNDO/temporary segments, etc. are released only at the end of the session.

- Thing to consider:
  - Faster sessions are closed - less resources used at any one point in time
  - Normal rule of thumb: 30-60 minutes of inactivity
  - Less time than that should be avoided or it negates the whole reason for connection pools

Expiration Mechanism (2)

- Expiration of “heavy” connections
  - “Heavy” can mean anything – PGA, opened cursors, allocated temporary tablespace, etc.
  - Nice option for long-term projects where you go through a number of different Oracle versions/patches/bugfixes
  - Nice back-door (if implemented using some kind of rule repository)



Full Refresh

- Feature:
  - More civilized way of completely resetting all database connections instead of bouncing the application server
- Solution:
  - Special type of request to the middle tier to stop it from serving an existing set of sessions (and eventually retire them) and get completely new ones.
- Reasons to use:
  - Handy if you need to modify some PL/SQL in a production system.
  - Stateless implementations make people less scared of encountering an "existing-state-of-packages" error
  - Connection pools reintroduce this issue in most real environments.

Resource Management

Session Resource Management (1)

- StateLESS implementation + session-level tricks for a single request:
  - Convenient to use temporary tables of package variables as buffers while processing.
  - Built-in feature (because middle tier would immediately release these when the session is closed).
- #1 cause of problems with connection pool:
  - Sessions are not closed anymore unless you do something about them.
  - High probability that one request could get data from the other one leading to data cross-contamination.

Session Resource Management (2)

- Cannot trust ANYTHING defined at the session level.
- Everything should be handled manually
  - Built-in in the connection pool mechanism executes a special cleanup module before serving any request in the session.

Handling Package Variables

- A few lines of code (both procedures take very little time to fire):
  - Reset all variables to the initial state
  - Release all memory freed by previous state

```
begin
   dbms_session.reset_package;
  dbms_session.free_unused_user_memory
end;
```

Temporary Tables

- More difficult to resolve
- No simple way to identify which tables have data, or to clean that data

```
procedure p_truncate is
    v_exist_yn varchar2(1);
begin
    select 'Y' into v_exist_yn
    from v$session s, v$tempseg_usage u
    where s.audsid = SYS_CONTEXT('USERENV','SESSIONID')
    and   s.saddr = u.session_addr
    and   u.segtype = 'DATA'
    and rownum = 1;
    for c in (select table_name from user_tables
              where temporary = 'Y'
              and duration = 'SYS$SESSION')
    loop
        execute immediate 'truncate table '||c.table_name;
    end loop;
end;
```

Caution!

- Since using *V$TEMPSEG_USAGE* makes it possible to detect whether or not the current session has temporary segments allocated, the cycle of cleanups can be avoided in most cases.
- Oracle DBMS does not release the TEMP tablespace allocated to temporary CLOBs (all CLOB variables) until the end of a session.
- Metalink ID 5723140 in 10.2.0.4 and 11.1.0.6, Oracle introduced event 60025 to get around the described behavior, but caution is strongly recommended.

Use a join?

- Join between V$SESSION and V$TEMPSEG_USAGE
  - Known to cause very strange errors in some cases (including even ORA-600).
  - Solution is simple - Just split the query in two as shown here:

```
select saddr
into v_saddr
from v$session s
where s.audsid = SYS_CONTEXT('USERENV','SESSIONID');

select 'Y'
into v_exist_yn
from v$tempseg_usage u
where u.session_addr = v_saddr
    and   u.segtype = 'DATA'
    and   rownum = 1;
```

Database Resource Utilization (1)

- Core assumption underlying any implementation of connection pools:
  - Single request to the database takes a very small amount of time.
  - Total number of active requests at any point in time is small compared to the total number of logical users in the system.
  - Slightest slow-down in the processing of requests could very quickly kill the whole system.

Database Resource Utilization (2)

- Problem:
  - System could work fine 99% of the time, but once some kind of a threshold is reached, the degradation spiral starts to unwind.
- Reason:
  - The more time needed to process an individual request, the more often it is necessary to add a new session to the pool.
- Mechanism:
  - No free sessions→ more simultaneous sessions
  - More sessions → more resources to be used
  - More resources used → less resources available per session
  - Less resources available → each request is slower
  - …
  - After a few cycles, the system has no resources left at all and collapses

Avoiding Problems

- Difficult to resolve in a production environment
- Should therefore be prevented using the following strategies during development:
  - Most often executed requests should be very carefully tuned because these requests define the average workload
  - Most expensive requests should not enter the system via the connection pool at all.
  - Avoid pooled sessions for any special kinds of requests
  - Connection pool should notify administrators when reaching a defined workload level (e.g. allocated PGA per session or total allocated PGA) or number of sessions in the pool

Training Issues

Working with Connection Pools (1)

- Source of problems:
  - Developers hear about session-reusability in connection pools and start using old tips and tricks for client-server solutions.
- Nightmare:
  - Everything works with a single user.
  - Adding a second user creates complete havoc.
- Reasons:
  - With only one user in the system, code will always use the first connection (unless the pool is randomized) ~ stateFULL!
  - Adding a second user means that requests from both logical sessions will be served by the same physical one.
  - Previously perfectly working “client-server-ish” code will cause very serious data cross-contamination.

Working with Connection Pools (2)

- Do not tell developers about connection pools at all?
  - An architectural way of solving resource workload problems on the system should not have anything to do with development solutions.
- Only applicable in some cases (unfortunately)
  - Sometimes, developers should know about alternative options for handling sessions.

Working with Connection Pools: Real-world Example

- Actual development environment:
  - PL/SQL wrappers on Java classes, loaded into an Oracle database
  - Java code establishes a connection with the external geocoding server, passes data, and returns results.
  - These requests are one of the most critical parts of the system and executed regularly by all users.
  - The cost of the initial request is very high (~ 10 sec) because of the whole initialization process (both Java and geocoding APIs)
  - Additional requests in the same session < 0.3 sec.
- Solution: Use non-randomized connection pools
  - Most costly request is the first request per session
  - Goal is to keep the smallest number of sessions

Conclusions

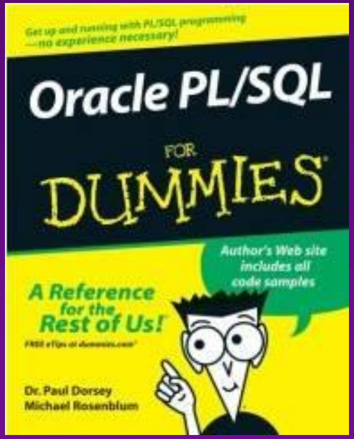
- There is no way to build any reasonable web-based solution without going stateless, but there are different ways of doing that.
- Using or not using connection pools is not a matter of preference, but a matter of understanding exactly what you are trying to build.
- Every feature solves some problems and introduces other ones. It is your responsibility to balance the pros and cons of using connection pools.

Contact Information

- ◆ Michael Rosenblum – mrosenblum@dulcian.com
- ◆ Dulcian website - www.dulcian.com

Latest book:

Oracle PL/SQL for Dummies


Get up and running with PL/SQL programming
—no experience necessary!
Oracle PL/SQL
FOR
DUMMIES
Author's Web site includes all code samples
A Reference for the Rest of Us!
Dr. Paul Dorsey
Michael Rosenblum