Database Resident Connection Pooling (DRCP)

Oracle Database 11*g*

Technical White paper

Introduction

Web tier and mid-tier applications typically have many threads of execution, which take turns using RDBMS resources. Currently, multi-threaded applications can share connections to the database efficiently, allowing great mid-tier scalability. Starting with Oracle 11g, application developers and administrators and DBAs can use Database Resident Connection Pooling to achieve such scalability by sharing connections among multi-process as well as multi-threaded applications that can span across mid-tier systems.

A connection resource for an Oracle Database is the database server process, the session and their associated memory. To minimize the database resources and to provide end-user concurrency at the same time, applications generally implement various resource pooling solutions. Oracle has provided connection-pooling solutions in all the data access drivers like OCI, OCCI, JDBC, ODP.NET etc. These take advantage of the ability of multiple threads in one application process to share resources.

However, there are many mid-tier deployments, which are process oriented or in which sharing of resources amongst threads is insufficient. These deployments fall into two major classes:

1. Single threaded applications such as most PHP applications
2. Multi threaded applications which often hold many idle connections, e.g. in redundant processes, due to spikes in usage, and could benefit from sharing across processes or even across systems.

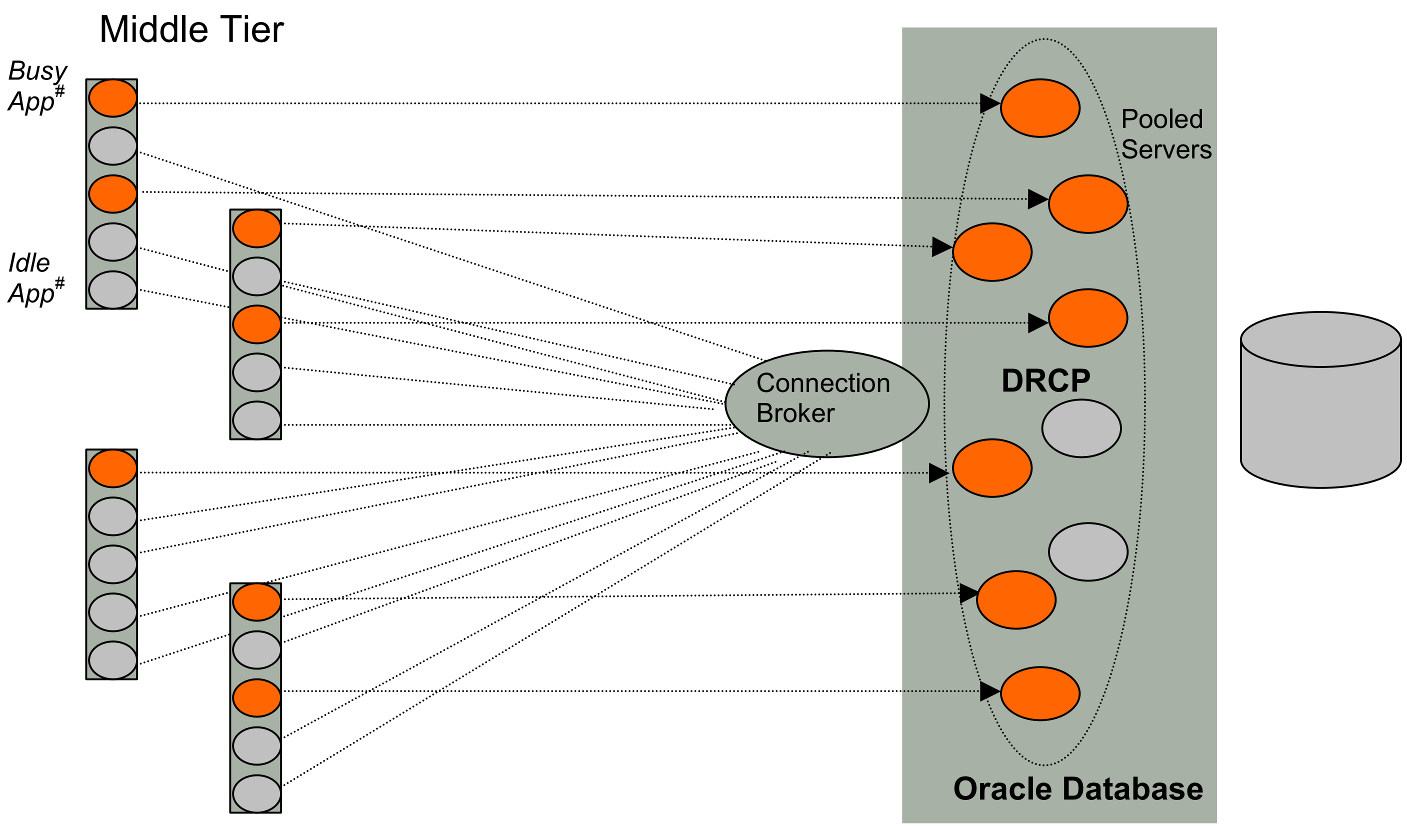
These applications tend to hold the database connections persistently while serving multiple user requests, to avoid connection creation and termination overheads when done on the fly. The mid-tier processes are typically configured to address the concurrency of end user requests, which is much higher than the concurrency of backend database accesses. If all the mid-tier processes use persistent connections to the database, the mid-tier scalability gets restricted due to database resource constraints.

Oracle Shared Server model partly addresses this issue, by pooling the Oracle server processes on the database side that can be shared by multiple connections. But the session resources are not shared. This model is useful where the clients need to hold the sessions for longer periods (for long running transactions etc.) with fewer clients actually doing the database activity at any point of time. However, in scenarios where the sessions are required for short database activity and the database activity across multiple requests does not depend on the session state, applications can achieve much higher scalability by using the Database Resident Connection Pooling.

Database Resident Connection Pooling (DRCP)

The Database Resident Connection Pooling, a new feature of Oracle Database 11g by design is targeted at addressing the scalability requirements in such environments that require support for tens of thousands of connections. DRCP pools database server processes and sessions (the combination is known as a *pooled server*), which are shared across connections from multiple application processes from the same host or from different hosts. A Connection Broker process manages the pooled servers in the database instance. Clients are persistently connected and authenticated to the Broker. Clients request the Broker to provide pooled servers when they need to perform some database activity, use them, and then release them back for reuse by other clients.

When the pooled servers are in use, they are equivalent to dedicated servers. Upon a request from the client on the persistent channel, the broker picks the appropriate pooled server and hands-off the client to that pooled server. The client directly communicates with the pooled server for all its database activity. The pooled server is handed back to the broker when the client releases it.



#: *Busy/Idle with respect to Database Activity*

Using Database Resident Connection Pool

Database Tier

Enabling and Configuring the pool: DRCP comes with an easy-to-use database administrative API. Every database instance of 11g has a default pool.
The pool can be configured and administered by the DBA using the PLSQL package DBMS_CONNECTION_POOL.

```
SQL>execute dbms_connection_pool.configure_pool(null, minsize=>10,
                                          maxsize=>100,
                                          inactivity_timeout=>300,
                                          max_think_time=>600, …);
```

The above is an optional step and can be used if the default settings have to be changed.

The pool needs to be started before the clients can request for connections. The command below brings up the Broker, which registers itself with the database listener.

```
SQL> execute dbms_connection_pool.start_pool;
```

Once enabled this way, the pool automatically restarts when the instance restarts, unless explicitly stopped by use of the `dbms_connection_pool.stop_pool` command.

Application Tier

Routing client connections to DRCP: Currently DRCP interface is available for OCI and OCCI clients using TCP/IP protocol and simple database authentication (userid/password based). The clients must specify the server type as POOLED via the connect string as shown below:

```
myhost.dom.com:1521/sales:POOLED

OR

(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=myhost.dom.com)
        (PORT=1521))(CONNECT_DATA=(SERVICE_NAME=sales)
        (SERVER=POOLED)))
```

Sharing the pooled servers: DRCP guarantees that the pooled servers are never shared across different users. However, DRCP allows for sharing the pooled servers across different instances of the same application. In addition, even for the same user, DRCP maintains a logical grouping of the pooled servers, based on the "connection classes" chosen by the applications. A connection class is a logical name supplied by a client while requesting for a pooled server. It indicates that the client is willing to reuse a pooled server, which was used by other clients using the same logical name.

For example, applications in AP suite may be willing to share the pooled servers among themselves but not among HR suite of applications. Differences in state that the applications leave in the sessions, or some session-specific attributes like language settings etc. may be the factors that influence this decision. Clients can request a brand new session if they cannot reuse a session from the pool.

OCI Clients using DRCP

1. Use `OCISessionPoolCreate()` to create an OCI Session Pool with the DRCP connect string. This maintains connection persistency with the Broker.
2. Use `OCISessionGet()` and `OCISessionRelease()` from that OCISessionPool which acquire and release pooled servers.
3. Specify a connection class value as an attribute on the auth handle passed to `OCISessionGet` call, before making the call.
4. If no connection class is specified, the OCISessionPool's name is implicitly used as the connection class by all the requests using that OCISessionPool, resulting in sharing only within a session pool, becoming equivalent to a plain client-side session pooling application.

OCCI Clients using DRCP

1. Use `Environment::createStatelessConnectionPool()` to create an OCCI Stateless Connection Pool with the DRCP connect string. This maintains connection persistency with the Broker.
2. Use `getConnection(…"APPCC"…)` and `releaseConnection()` from that StatelessConnectionPool which acquire and release pooled servers.

PHP Clients using DRCP

Oracle plans to work closely with the open source community on enhancing the PHP-OCI8 driver to leverage DRCP. This will enable PHP applications to maintain persistent connections with the database with minimal resource constraints.

Tuning DRCP

Depending on the available hardware resources and concurrency requirements, the pool can be configured for its size. The `minsize` ensures the pool comes up with and always keeps a certain minimum number of servers. The `maxsize` puts a cap on the total number of pooled servers. The client requests wait if all the pooled servers are busy and the pool is at its `maxsize`, till one becomes free.

The `inactivity_timeout` setting helps the pool to terminate idle pooled servers, optimizing the resources. To avoid pooled servers permanently being locked up by bad clients or dead clients, `max_think_time` can be set on the pool. The parameters `num_cbrok` and `maxconn_cbrok` can be used to distribute the persistent connections from the clients across multiple brokers. This may particularly be needed in cases where the Operating System limits the maximum number of connections per process.

DBA views are available to monitor the performance of DRCP. The DBA can check the number of busy and free servers, number of hits and misses in the pool against the total number of requests from clients etc. The DRCP configuration parameters can be altered dynamically any time, using `dbms_connection_pool.alter_param` method or `dbms_connection_pool.configure_pool` method.

DRCP Scalability

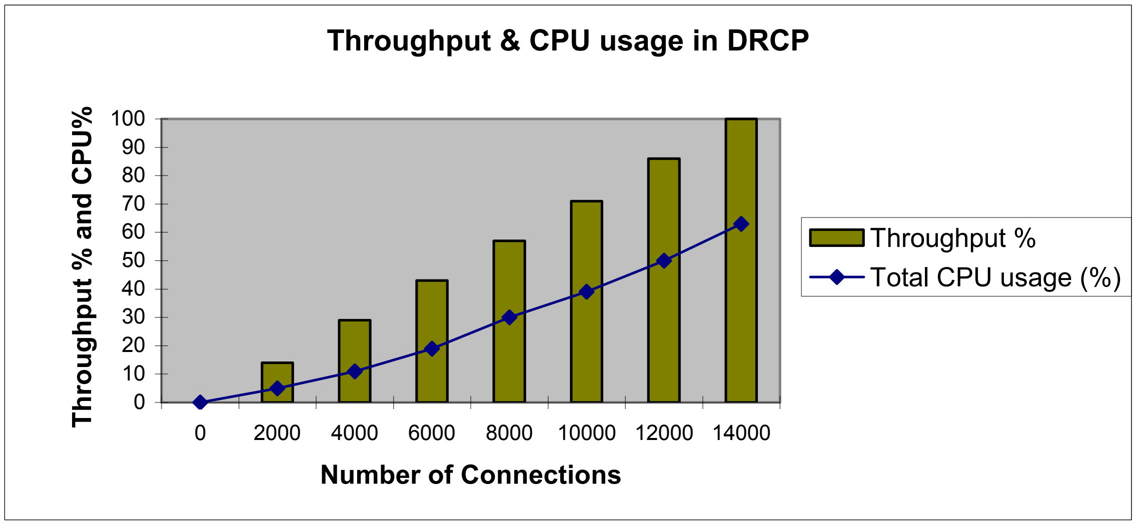
DRCP boosts the scalability of the database and the mid-tier, as the persistent connections to the database are held at minimal cost. The real database connection resources are only used by the active connections, and the scaling of that usage can be explicitly controlled by setting the pool size.

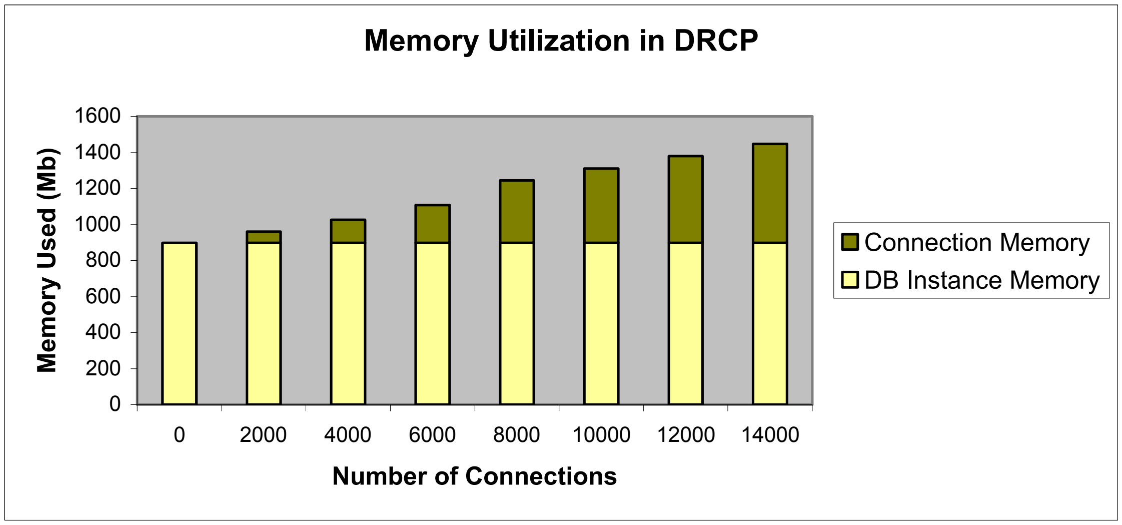
How much of an advantage can this provide?- As each inactive client connection has a very small memory requirement, DRCP can support many times the number of client connections (compared to dedicated and shared servers), when each client connection holds the database session for a short time. In one of our test scenarios on a 2GB system, DRCP was able to support 10 times the number of connections compared to shared servers, and 20 times the number of connections compared to dedicated servers.

Performance measurements were done with a sample OCI client against the Database Resident Connection Pool and the results are shown below. The clients were standalone OCI application processes concurrently requesting for pooled servers and performing query executions and DML operations. The workload chosen was a mix of a READONLY transaction and a READWRITE transaction in the ratio of 3:1. The READONLY transaction included a “get connection” operation, execution of five SELECT statements and a “release connection”. The READWRITE transaction, on the other hand, had the same operations but also included an UPDATE statement. The clients used 15sec think time between requests.

The database instance with DRCP was hosted on a 4 CPU Intel Xeon MP 2.80GHz machine with 2GB RAM, running 32bit Red Hat Enterprise Linux Version 4. The pool was up with one Connection Broker and 100 pooled servers. Oracle 11g Beta5 was used for the client and database software. The CPU and memory utilizations were captured using `vmstat` command and the throughput was captured using Oracle AWR reporting. The OS file descriptor limit was set at 20000.

The throughput had scaled linearly as the number of incoming connections grew and the memory required for the connections was minimal, again proportionate to the number of connections.





Summary

The Database Resident Connection Pool is an ideal solution for efficiently handling a large number (in the order of tens of thousands) of connections to Oracle database from clients distributed over multiple processes on multiple hosts.

For more information, please see the following Oracle guides:

- *Database Concepts*
- *Database Administrator's Guide*
- *Oracle Call Interface Programmer's Guide*
- *Oracle C++ Call Interface Programmer's Guide*
- *PL/SQL Packages and Types Reference*
- *Database Reference*

ORACLE

Database Resident Connection Pooling
September 2007

Author: Krishna Mohan Itikarlapalli
Contributors: Kevin Neel, Sreekumar Seshadri, Luxi Chidambaran, Amoghavarsha Ramappa, Srinath Krishnaswamy, Santanu Datta

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com