

# PHP Scalability and High Availability

Database Resident Connection Pooling and  
Fast Application Notification

*An Oracle White Paper*  
*April 2008*

# PHP Scalability and High Availability

Introduction.....	3
Connection Pooling with DRCP.....	4
What is Database Resident Connection Pooling?.....	4
How DRCP Works.....	4
PHP OCI8 Connections and DRCP.....	7
When to use DRCP.....	9
Sharing the Server Pool.....	10
Using DRCP in PHP.....	12
Configuring and Enabling the Pool.....	12
Configuring PHP for DRCP.....	15
Application Deployment for DRCP.....	16
Monitoring DRCP.....	19
V\$PROCESS and V\$SESSION Views.....	20
DBA_CPOOL_INFO View.....	20
V\$CPOOL_STATS View.....	20
V\$CPOOL_CC_STATS View.....	22
DRCP Scalability Benchmark.....	23
High Availability with FAN and RAC.....	24
Configuring FAN Events in the Database.....	25
Configuring PHP for FAN.....	25
Application Deployment for FAN.....	25
RAC Connection Load Balancing with PHP.....	26
Conclusion.....	27
More Information.....	28

## INTRODUCTION

PHP is a popular dynamic programming language for web applications. PHP comes with extensions that provide a wide range of capabilities including database access. The OCI8 extension\* in PHP allows applications to connect to Oracle Database. It has support for basic and advanced Oracle Database features, allowing easy and efficient use of SQL and PL/SQL.

This paper describes enhancements to PHP's OCI8 extension to use:

- Oracle Database 11g Database Resident Connection Pooling (DRCP)
- Oracle Database 10gR2 or 11g Fast Application Notification (FAN)

The Oracle features are usable separately or together.

Without pooling, frequent connection creation and destruction can be expensive and crippling to high scalability. Oracle's traditional middle-tier connection pools were not applicable to the multi-process PHP architecture. Caching connections within PHP removed connection creation and destruction costs but did not achieve optimal connection resource utilization, incurring unnecessary memory overhead in the database.

With DRCP, Oracle Database 11g introduces a pooling solution usable by PHP. Database web applications can now be highly scalable.

In addition, PHP previously did not fully exploit advanced Oracle RAC (Oracle's Real Application Clusters) features for high availability or scalability. After a database instance or machine failure, applications could hang until a network timeout occurred. There was no proactive cleanup of cached connections to failed instances.

The new FAN support allows PHP database applications to be resilient.

Whether these features are used together, separately, or even not used at all, PHP's OCI8 is an efficient and reliable extension for building database applications. The new extension can be built with PHP 4 and PHP 5, and will compile with Oracle 9iR2, 10g and 11g client libraries. Oracle's standard cross-version compatibility is

---

\* PHP's OCI8 extension gets its name from the C language API for Oracle. It gives PHP scripts an API with a higher abstraction of database functionality than Oracle's OCI8 C API; the two APIs should not be confused.

**Database Resident Connection Pooling (DRCP) is a new feature of Oracle Database 11g that addresses scalability requirements in environments requiring large numbers of connections with minimal database resource usage.**

**Clients that run in high availability configurations such as with Oracle RAC or Data Guard Physical Stand-By can take advantage of Fast Application Notification (FAN) events to allow applications to respond quickly to database node failures.**

applicable. The new extension is compatible with the previous release. Combined, this means existing applications can easily be migrated to take advantage of the latest database features available in Oracle Database 11g.

## **CONNECTION POOLING WITH DRCP**

### **What is Database Resident Connection Pooling?**

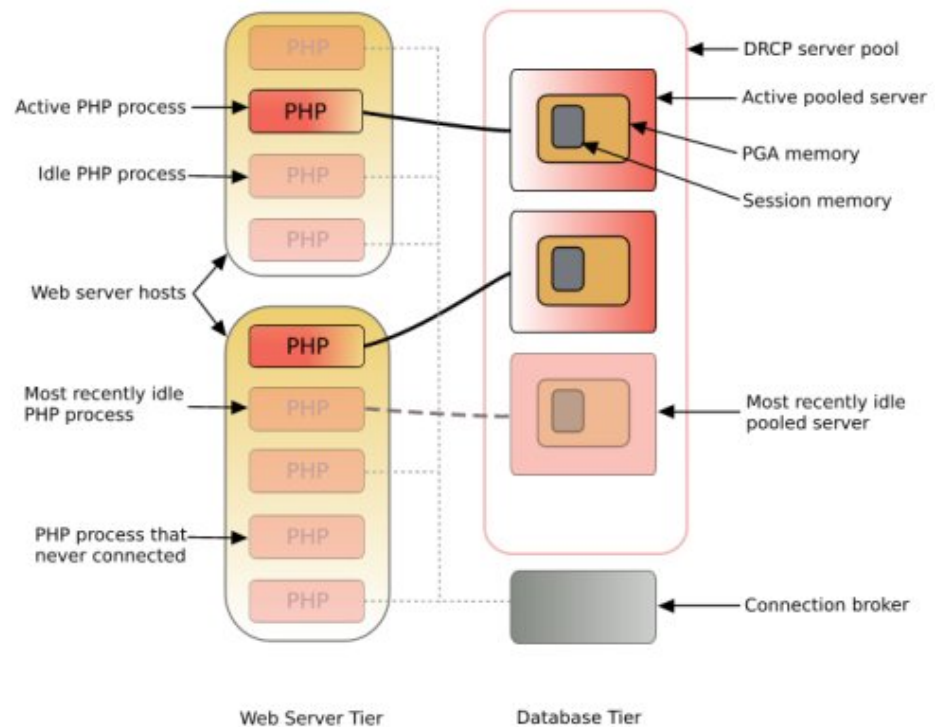
DRCP is a new feature of Oracle Database 11g that addresses scalability requirements in environments requiring large numbers of connections with minimal database resource usage. DRCP pools a set of dedicated database server processes (known as *pooled servers*), which can be shared across multiple applications running on the same or several hosts. A connection broker process manages the pooled servers at the database instance level.

### **How DRCP Works**

The architecture of DRCP is shown in Figure 1. A connection broker accepts incoming connection requests from PHP processes (e.g. web server processes handling PHP requests) and assigns each a free server in the pool. Each PHP process that is executing a PHP script communicates with this Oracle server until the connection is released. This happens automatically at the end of the script, or the connection can be explicitly released. When the connection is released, the server process is returned to the pool and the PHP process keeps a link only to the connection broker. Active pooled servers contain the Process Global Area (PGA) and the user session data. Idle servers in the pool retain the user session for reuse by subsequent persistent PHP connections.

When the number of persistent connections is less than the number of pooled servers, a “dedicated optimization” avoids unnecessarily returning servers to the pool when a PHP connection is closed. Instead, the dedicated association between the PHP process and the server is kept in anticipation that the PHP process will quickly become active again. If PHP scripts are executed by numerous web servers, the DRCP pool can grow to its maximum size (albeit typically a relatively small size), even if the rate of incoming user requests is low. Each PHP process, either busy or now idle, will be attached to its own pooled server. When the pool reaches its maximum size, a script that is handled by a PHP process without a pooled server will cause an idle server to be returned to the pool for immediate reuse.

Figure 1. DRCP Architecture.

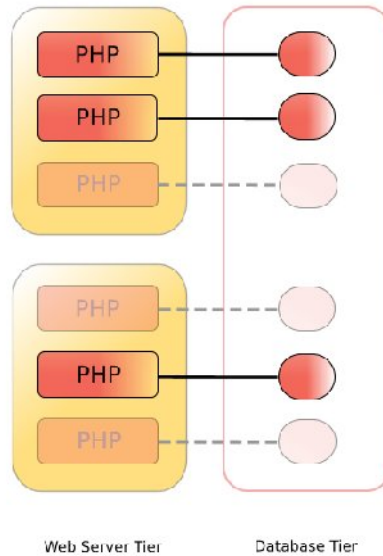


The pool size and number of connection brokers are configurable. There is always at least one connection broker per database instance when DRCP is enabled. Also, at any time, around 5% of the current pooled servers are reserved for authenticating new PHP connections. Authentication is performed when a PHP process establishes a connection to the connection broker.

DRCP boosts the scalability of the database and the web server tier because connections to the database are held at minimal cost. Database memory is only used by the pooled servers, and scaling can be explicitly controlled by DRCP tuning options.

Without DRCP, each PHP process creates and destroys database servers when connections are opened and closed. Or alternatively, each process keeps connections open (“persistent connections”) even when they are not processing any user scripts. This consumes database resources, as shown in Figure 2.

Figure 2. Without DRCP, idle persistent connections from PHP still consume database resources.



With the introduction of pooled servers used by DRCP, there are now three types of database server process models that Oracle applications can use: dedicated servers, shared servers and pooled servers.

Table 1. Differences between dedicated servers, shared servers, and pooled servers.

Dedicated Servers	Shared Servers	Pooled Servers
When the PHP connection is created, a network connection to a dedicated server process and associated session are created	When the PHP connection is created, a network connection to the dispatcher process is established. A session is created in the SGA	When the PHP connection is created, a network connection to the connection broker is established
Activity on a connection is handled by the dedicated server	Each action on a connection goes through the dispatcher, which hands the work to a shared server	Activity on a connection wakes the broker, which hands the network connection to a pooled server. The server then handles subsequent requests directly, just like a dedicated server

Dedicated Servers	Shared Servers	Pooled Servers
Idle PHP connections* hold a server process and session resources	Idle PHP connections hold session resources but not a server process	Idle PHP connections hold a server process and session resources
Closing a PHP connection* causes the session to be freed and the server process to be terminated	Closing a PHP connection causes the session to be freed	Closing a PHP connection causes the session to be destroyed and the pooled server to be released to the pool. A network connection to the connection broker is retained
Memory usage is proportional to the number of server processes and sessions. There is one server and one session for each PHP connection	Memory usage is proportional to the sum of the shared servers and sessions. There is one session for each PHP connection	Memory usage is proportional to the number of pooled servers and their sessions. There is one session for each pooled server

Pooled servers in use by PHP are similar in behavior to dedicated servers. After connection, PHP directly communicates with the pooled server for all database operations.

#### PHP OCI8 Connections and DRCP

The PHP OCI8 extension has three functions for connecting to a database: `oci_connect()`, `oci_new_connect()`, and `oci_pconnect()`. The implementation of these functions has been reworked and all benefit from using DRCP. Table 2 compares dedicated and pooled servers. Shared servers are similar to dedicated servers with the exception that only the session and not the server is destroyed when a connection is closed.

*Table 2. Behavior of OCI8 connection functions for Dedicated and Pooled Servers.*

OCI8 Function	With Dedicated Servers	With Pooled Servers
<code>oci_connect()</code>	Creates a PHP connection to the database using a dedicated server. The connection is cached in the PHP process for reuse by subsequent	Gets a pooled server from the DRCP pool and creates a brand new session. Subsequent <code>oci_connect()</code> calls in the same script use the same connection. Upon

\*These rows of Table 1 are for `oci_connect()`. See Table 2 for the differences between connection types.

OCI8 Function	With Dedicated Servers	With Pooled Servers
	<p><code>oci_connect()</code> calls in the same script. At the end of the script or with <code>oci_close()</code>, the connection is closed and the server process and session are destroyed</p>	<p>end of the script or with <code>oci_close()</code>, the session is destroyed and the pooled server is available for other PHP connections to use</p>
<code>oci_new_connect()</code>	<p>Similar to <code>oci_connect()</code> above, but an independent new PHP connection and server process is created every time this function is called, even within the same script. All PHP connections and the database servers are closed when the script ends or with <code>oci_close()</code>. Sessions are destroyed at that time</p>	<p>Similar to <code>oci_connect()</code> above, but an independent server in the pool is used and a new session is created each time this function is called in the same script. All sessions are destroyed at the end of the script or with <code>oci_close()</code>. The pooled servers are made available for other connections to use</p>
<code>oci_pconnect()</code>	<p>Creates a persistent PHP connection which is cached in the PHP process. The connection is not closed at the end of the script, and the server and session are available for reuse by any subsequent <code>oci_pconnect()</code> call passing the same credentials</p>	<p>Creates a persistent PHP connection. Calling <code>oci_close()</code> releases the connection and returns the server with its session intact to the pool for reuse by other PHP processes. If <code>oci_close()</code> is not called, then this connection release happens at the end of the script. Subsequent calls to <code>oci_pconnect()</code> reuse the existing network connection to quickly get a server and session from the pool</p>

With DRCP, all three connection functions save on the cost of authentication and benefit from the network connection to the connection broker being maintained,

even for connections that are “closed” from PHP’s point of view. They also benefit from having pre-spawned server processes in the DRCP pool.

The `oci_pconnect()` function reuses sessions, allowing even greater scalability. The non-persistent connection functions create and destroy new sessions each time they are used, allowing less sharing at the cost of reduced performance.

Overall, after a brief warm-up period for the pool, DRCP allows reduced connection times in addition to the reuse benefits of pooling.

#### **When to use DRCP**

DRCP is typically preferred for applications with a large number of connections. Shared servers are useful for a medium number of connections and dedicated sessions are preferred for small numbers of connections. The threshold sizes are relative to the amount of memory available on the database host.

DRCP can be useful when any of the following apply:

- A large number of connections need to be supported with minimum memory usage on the database host.
- PHP applications mostly use the same database credentials for all connections.
- The applications acquire a database connection, work on it for a relatively short duration, and then release it.
- There are multiple web servers and web server hosts.
- Connections look identical in terms of session settings, for example date format settings and PL/SQL package state.

DRCP provides the following advantages:

- It enables resource sharing among multiple middle-tier client applications.
- It improves scalability of databases and applications by reducing resource usage.

For persistent PHP connections, dedicated servers can be fastest. There is no broker or dispatcher overhead. The server is always connected and available whenever the PHP process needs it. But as the number of connections increases, the memory cost of keeping connections open quickly reduces efficiency of the database system.

For non-persistent PHP connections, DRCP can be fastest because the use of pooled server processes removes the need for PHP connections to create and destroy processes, and removes the need to re-authenticate for each connect call.

Consider an application in which the memory required for each session is 400 KB. On a 32 bit operating system the memory required for each server process could be 4 MB, and DRCP could use 35 KB per connection (mostly in the connection

broker). If the number of pooled servers is configured at 100, the number of shared servers is configured at 100, and the deployed application creates 5000 PHP connections, then the memory used by each type of server is estimated in Table 3.

*Table 3. Example database host memory use for dedicated, shared and pooled servers.*

	<b>Dedicated Servers</b>	<b>Shared Servers</b>	<b>Pooled Servers</b>
Database Server Memory	5000 * 4 MB	100 * 4 MB	100 * 4 MB
Session Memory	5000 * 400 KB	5000 * 400 KB Note: For Shared Servers, session memory is allocated from the SGA	100 * 400 KB
DRCP Connection Broker Overhead			5000 * 35 KB
<b>Total Memory</b>	<b>21 GB</b>	<b>2.3 GB</b>	<b>610 MB</b>

**There is a significant memory saving when using DRCP.**

There is a significant memory saving when using DRCP.

Even if sufficient memory is available to run in dedicated mode, DRCP can still be a viable option if the PHP application needs database connections for only short periods of time. In this case the memory saved by using DRCP can be used towards increasing the SGA, thereby improving overall performance.

Pooling is available when connecting over TCP/IP with userid/password based database authentication. It is not available using bequeath connections.

### **Sharing the Server Pool**

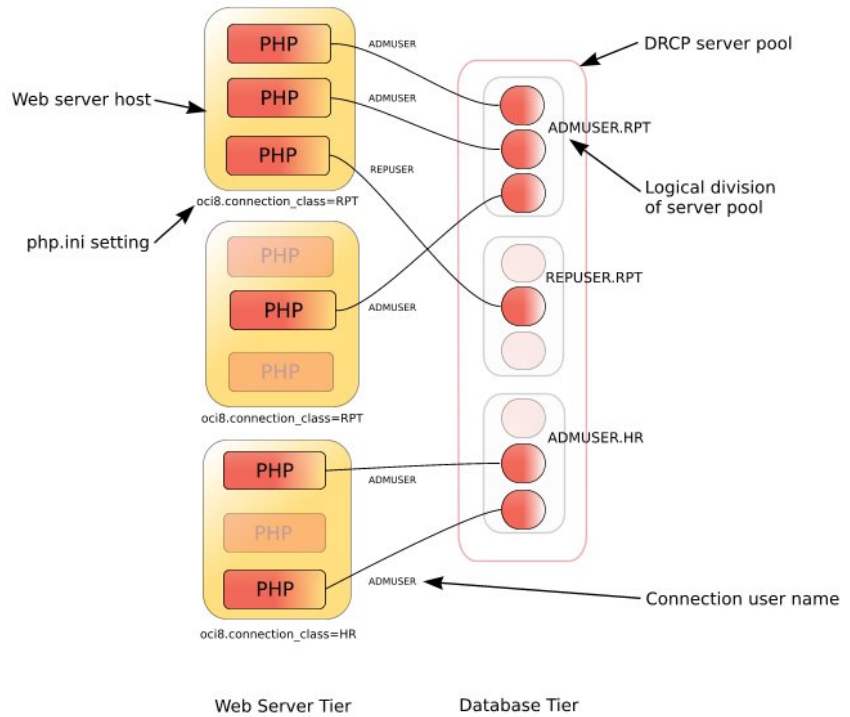
DRCP guarantees that pooled servers and sessions initially used by one database user are only ever reusable by connections with that same userid. DRCP also further partitions the pool into logical groups or “connection classes”. A connection class is a user chosen name set in the `php.ini` configuration file.

Session-specific attributes, like the date format or an explicit role, may be re-usable by any connection in a particular application. Subsequent persistent connections will reuse the session and inherit those settings if the username and connection class are the same as the previous connection.

Applications that need different state in the sessions should use different usernames and/or connection classes.

For example, applications in a suite called RPT may be willing to share pooled servers between themselves but not with an application suite called HR. The different connection classes and resulting logical partitioning of the DRCP server pool is shown in Figure 3. Connections with the same username and connection class from any host will share the same sub-pool of servers.

Figure 3. The DRCP pool is logically partitioned by username and connection class.



If there are no free pooled servers matching a request for a userid in the specified connection class, and if the pool is already at its maximum size, then an idle server in the pool will be used and a new session created for it. If the server originally belonged to a different connection class, the server will migrate to the new class. If there are no pooled servers available, the connection request waits for one to become available. This allows the database to continue without becoming overloaded.

The connection class should be set to the same value for each instance of PHP running the same application where sharing of pooled connections is desired. If no connection class is specified, each web server process will have a unique, system generated class name, limiting sharing of connections to each process.

If DRCP is used but session sharing is not desirable under any condition, use `oci_connect()` or `oci_new_connect()` which recreate the session each time.

Although session data may be reused by subsequent persistent connections, transactions do not span connections across scripts. Uncommitted data will be rolled back at the end of a PHP script.

### Using DRCP in PHP

Using DRCP with PHP applications involves the following steps:

1. Configuring and enabling the pool
2. Configuring PHP
3. Deploying the application

PHP applications deployed as Apache modules, FastCGI, CGI and standalone applications can benefit from DRCP. PHP applications deployed as Apache modules or with FastCGI gain most, since they remain connected to the connection broker over multiple script executions and can take advantage of other optimizations, such as statement caching.

### Configuring and Enabling the Pool

Every instance of Oracle Database 11g uses a single, default connection pool. User defined pools are currently not supported. The default pool can be configured and administered by a DBA using the `DBMS_CONNECTION_POOL` package:

```
SQL> execute dbms_connection_pool.configure_pool(
           pool_name => 'SYS_DEFAULT_CONNECTION_POOL',
           minsize      => 4,
           maxsize      => 40,
           incsize      => 2,
           session_cached_cursors => 20,
           inactivity_timeout => 300,
           max_think_time  => 600,
           max_use_session => 500000,
           max_lifetime_session => 86400);
```

Alternatively the method `dbms_connection_pool.alter_param()` can be used to set a single parameter:

```
SQL> execute dbms_connection_pool.alter_param(
           pool_name => 'SYS_DEFAULT_CONNECTION_POOL',
           param_name => 'MAX_THINK_TIME',
           param_value => '1200');
```

There is a `dbms_connection_pool.restore_defaults()` procedure to reset all values.

When DRCP is used with RAC, each database instance has its own connection broker and pool of servers. Each pool has the identical configuration. For example

all pools will have maxsize server processes. A single `dbms_connection_pool` command will alter the pool of each instance at the same time.

The pool needs to be started before connection requests begin. The command below does this by bringing 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 with the `dbms_connection_pool.stop_pool()` command:

```
SQL> execute dbms_connection_pool.stop_pool();
```

The DRCP configuration options are described in Table 4.

*Table 4. DRCP Configuration Options.*

	Description
<code>pool_name</code>	The pool to be configured. Currently the only supported name is the default value <code>SYS_DEFAULT_CONNECTION_POOL</code>
<code>minsize</code>	Minimum number of pooled servers in the pool. The default is 4
<code>maxsize</code>	Maximum number of pooled servers in the pool. If this limit is reached and all the pooled servers are busy, then connection requests wait until a server becomes free. The default is 40
<code>incrsz</code>	The number of pooled servers is increased by this value when servers are unavailable for PHP connections and if the pool is not yet at its maximum size. The default is 2
<code>session_cached_cursors</code>	Indicates to turn on <code>SESSION_CACHED_CURSORS</code> for all connections in the pool. This value is typically set to the size of the working set of frequently used statements. The cache uses cursor resources on the server. The default is 20. Note: there is also an <code>init.ora</code> parameter for setting the value for the whole database instance. The pool option allows a DRCP-based application to override the instance setting.

	Description
<code>inactivity_timeout</code>	Time to live for an idle server in the pool. If a server remains idle in the pool for this time, it is killed. This parameter helps to shrink the pool when it is not used to its maximum capacity. The default is 300 seconds
<code>max_think_time</code>	Maximum time of inactivity the PHP script is allowed after connecting. If the script does not issue a database call for this amount of time, the pooled server may be returned to the pool for reuse. The PHP script will get an ORA error if it later tries to use the connection. The default is 120 seconds
<code>max_use_session</code>	Maximum number of times a server can be taken and released to the pool before it is flagged for restarting. The default is 500000
<code>max_lifetime_session</code>	Time to live for a pooled server before it is restarted. The default is 86400 seconds
<code>num_cbrok</code>	The number of connection brokers that are created to handle connection requests. Note: this can only be set with <code>alter_param()</code> . The default is 1
<code>maxconn_cbrok</code>	The maximum number of connections that each connection broker can handle. Set the per-process file descriptor limit of the operating system sufficiently high so that it supports the number of connections specified. Note: this can only be set with <code>alter_param()</code> . The default is 40000

Note: the parameters have been described here relative to their use in PHP but it is worth remembering that the DRCP pool is usable concurrently by other programs.

In general, if pool parameters are changed, the pool should be restarted, otherwise server processes will continue to use old settings.

The `inactivity_timeout` setting terminates idle pooled servers, helping optimize database resources. To avoid pooled servers permanently being held onto by a dead web server process or a selfish PHP script, the `max_think_time` parameter can be set. The parameters `num_cbrok` and `maxconn_cbrok` can be used to distribute the persistent connections from the clients across multiple brokers. This may be needed in cases where the operating system per-process descriptor limit is small.

The `max_use_session` and `max_lifetime_session` parameters help protect against any unforeseen problems affecting server processes. The default values will be suitable for most users.

Users of Oracle 11.1.0.6 must apply the database patch for bug 6474441. This is necessary with OCI8 1.3 to avoid query errors. It also enables LOGON trigger support. The bug will be fixed in the planned 11.1.0.7 patch set.

### Configuring PHP for DRCP

PHP must be built with the DRCP-enabled OCI8 extension. Download OCI8 1.3 (currently still Beta) from PECL, extract it and use it to replace the existing `ext/oci8` directory in PHP. Configure, build and install PHP as normal.

Alternatively use the `pecl install` command to build and install the OCI8 as a shared module. PHP 5.3 and PHP 6, neither of which has been released, contain the new extension by default.

The new OCI8 extension can be used with Oracle client libraries version 9.2 and higher, however DRCP functionality is only available when PHP is linked with Oracle 11g client libraries and connects to Oracle Database 11g.

Once installed, use PHP's `phpinfo()` function to verify that OCI8 1.3 has been loaded.

Before using DRCP, the new `php.ini` parameter `oci8.connection_class` should be set to specify the connection class used by all the requests for pooled servers by the PHP application.

```
oci8.connection_class = MYPHPAPP
```

The parameter can be set in `php.ini`, `.htaccess` or `httpd.conf` files. It can also be set and retrieved programmatically using the PHP functions `ini_set()` and `ini_get()`.

The OCI8 extension has several legacy `php.ini` configuration parameters for tuning persistent connections. These were mainly used to limit idle resource usage. With DRCP, the parameters still have an effect but it may be easier to use the DRCP pool configuration options.

*Table 5. Existing `php.ini` parameters for persistent connections.*

Php.ini Parameter	Behavior with DRCP
<code>oci8.persistent_timeout</code>	At the timeout of an idle PHP connection, PHP will close the Oracle connection to the broker. The default is no timeout
<code>oci8.max_persistent</code>	The maximum number of unique persistent connections that PHP will maintain to the broker. When the limit is reached a new persistent connection behaves like <code>oci_connect()</code> and

	releases the connection at the end of the script. The default is no limit. Note: the DRCP "maxsize" setting will still be enforced by the database independently from <code>oci8.max_persistent</code>
<code>oci8.ping_interval</code>	The existing recommendation to set it to -1 to disable pinging and to use appropriate error checking still holds true with DRCP. Also, the use of FAN (see later) reduces the chance of idle connections becoming unusable. The default is 60 seconds

With OCI8 1.3, `oci8.ping_interval` is also used for non-persistent connections when pooled servers are used. This helps protect against the possibility of unusable connections being returned to PHP when there has been a database instance or node failure.

Web servers and the network should benefit from `oci8.statement_cache_size` being set. For best performance it should generally be larger than the size of the working set of SQL statements. To tune it, monitor general web server load and the AWR "bytes sent via SQL\*Net to client" values. The latter statistic should benefit from not shipping statement meta-data to PHP. Adjust the statement cache size to your satisfaction.

Once you are happy with the statement cache size, then tune the DRCP pool `session_cached_cursors` value. Monitor AWR reports with the goal to make the "session cursor cache hits" close to the number of soft parses. Soft parses can be calculated from "parse count (total)" minus "parse count (hard)".

#### Application Deployment for DRCP

PHP applications must specify the server type `POOLED` in the connect string. Using Oracle's Easy Connect syntax, the PHP call to connect to the `sales` database on `myhost` would look like:

```
$c = oci_pconnect('myuser', 'mypassword',
                 'myhost/sales:POOLED');
```

or if PHP uses an Oracle Network alias that looks like:

```
$c = oci_pconnect('myuser', 'mypassword', 'salespool');
```

then only the Oracle Network configuration file `tnsnames.ora` needs to be modified:

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

If these changes are made and the database is not actually configured for DRCP, connections will not succeed and an error will be returned to PHP.

### **Closing Connections**

PHP scripts that do not currently use `oci_close()` should be examined to see if they can use it to explicitly return connections to the pool, allowing maximum use of pooled servers:

```
// 1. Do some database operations
$conn = oci_pconnect('myuser','mypassword','salespool');
. . .
oci_commit($conn);

// Release the connection to the DRCP pool
oci_close($conn);

// 2. Do some non-database work
. . .

// 3. Do some more database operations
$conn = oci_pconnect('myuser','mypassword','salespool');
. . .
oci_commit($conn);
```

**With the new version of the OCI8 extension, closing a persistent connection now releases the server to the pool allowing optimal pool usage.**

Prior to OCI8 1.3, closing `oci_connect()` and `oci_new_connect()` connections had an effect but closing an `oci_pconnect()` connection was a no-op. Now, with the new version of the extension, `oci_close()` on a persistent connection rolls back any uncommitted transaction. Also the extension will do a roll back when all PHP variables referencing a persistent connection go out of scope, for example if the connection was opened in a function and the function has now finished. For DRCP, in addition to the rollback, the connection is also released; a subsequent `oci_pconnect()` may get a different connection. For DRCP, the benefit is that scripts taking advantage of persistent connections can explicitly return a server to the pool when non-database processing occurs, allowing other concurrent scripts to make use of the pooled server.

With pooled servers, the recommendation is to release the connection when the script does a significant amount of processing that is not database related. Explicitly control commits and rollbacks so there is no unexpectedly open

transaction when the close or end-of-scope occurs. Scripts coded like this can use `oci_close()` to take advantage of DRCP but still be portable to older versions of the OCI8 extension.

If behavior where `oci_close()` is a no-op for all connection types is preferred, set the existing `php.ini` parameter `oci8.old_oci_close_semantics` to `On`.

### **Transactions Across Re-connection**

Scripts should avoid re-opening connections if there are incomplete transactions:

```
// 1. Do some database operations
$conn = oci_pconnect('myuser','mypassword','salespool');

// Start a transaction
$s = oci_parse($conn, 'insert into mytab values (1)');
$r = oci_execute($s, OCI_DEFAULT); // no commit

. . .

// BAD: no commit or rollback done

// 2. Continue database operations on same credentials
$conn = oci_pconnect('myuser','mypassword','salespool');

$s = oci_parse($conn, 'insert into mytab values (2)');
$r = oci_execute($s, OCI_DEFAULT); // no commit

// Rollback or commit both 1 & 2
if (!$r)
    oci_rollback($conn);
else
    oci_commit($conn);
```

If there was a node or network failure just prior to point 2, the first transaction could be lost. The second connection command may return a new, valid connection if a ping (see `oci8.ping_interval`) occurs to validate the connection, and the script might not be aware of a problem.

The script should do an explicit commit or rollback before the second connect, or simply continue to use the original connection.

### **LOGON and LOGOFF Triggers with DRCP**

LOGON triggers are useful for setting session attributes needed by each PHP connection. For example a trigger could be used to execute an `ALTER SESSION` statement to set a date format. The LOGON trigger will execute when `oci_pconnect()` creates the session, and the session will be reused by subsequent persistent connections.

The suggested practice is to use LOGON triggers only for setting session attributes and not for executing per PHP-connection logic such as custom logon

auditing. This recommendation is also true for persistent connections with dedicated or shared servers, and is the existing recommendation for earlier releases of the OCI8 extension.

Database actions that must be performed exactly once per OCI8 connection call should be explicitly executed in the PHP script.

It is not possible to depend on LOGON triggers for tracking PHP OCI8 connect calls. The caching, pooling, timing out and recreation of sessions and connections with or without DRCP or the new extension can distort any record. With pooled servers, LOGON triggers can fire at authentication and when the session is created, in effect firing twice for the initial connection.

LOGOFF triggers do not fire for pooled servers. For non-pooled servers there is no change in OCI 1.3. LOGOFF triggers fire when connections are closed. For `oci_connect()` and `oci_new_connect()` connections this is with `oci_close()` or at the end of the script. For `oci_pconnect()` connections, it is when the web server process terminates or restarts.

### ***Changing Passwords with DRCP Connections***

In general, PHP applications that change passwords should avoid using persistent connections because connections stored persistently in PHP may allow connections to succeed using the old password. With DRCP, there is a further limitation - connections cannot be used to change passwords programmatically. PHP scripts that use `oci_password_change()` should continue to use dedicated or shared servers.

### **Monitoring DRCP**

Data dictionary views are available to monitor the performance of DRCP.

Database administrators can check statistics such as the number of busy and free servers, and the number of hits and misses in the pool against the total number of requests from clients. The views are:

```
V$PROCESS  
V$SESSION  
DBA_CPOOL_INFO  
V$CPOOL_STATS  
V$CPOOL_CC_STATS
```

The DRCP statistics are reset each time the pool is started.

For RAC, there are `GV$CPOOL_STATS` and `GV$CPOOL_CC_STATS` views corresponding to the instance-level views. These record DRCP statistics across clustered instances. If a database instance in a cluster is shut down, the statistics for that instance are purged from the `GV$` views.

**LOGON triggers are efficient for setting common session attributes such as date formats.**

### V\$PROCESS and V\$SESSION Views

The number of configured brokers per instance can be found from the V\$PROCESS view. For example, on Linux this query shows one broker has been enabled:

```
SQL> select program
      from v$process
      where program like 'oracle%(N%)';

PROGRAM
-----
oracle@localhost (N001)
```

The V\$SESSION view will show information about the currently active DRCP sessions. It can also be joined with V\$PROCESS via V\$SESSION.PADDR = V\$PROCESS.ADDR to correlate the views.

### DBA\_CPOOL\_INFO View

DBA\_CPOOL\_INFO displays configuration information about all DRCP pools in the database. The columns are equivalent to the dbms\_connection\_pool.configure\_pool() settings described in Table 4, with the addition of a STATUS column. The status is ACTIVE if the pool has been started and INACTIVE otherwise. Note the pool name column is called CONNECTION\_POOL.

For example, to check whether the pool has been started and what the maximum number of pooled servers is set to:

```
SQL> select connection_pool, status, maxsize
      from dba_cpool_info;

CONNECTION_POOL          STATUS          MAXSIZE
-----
SYS_DEFAULT_CONNECTION_POOL  ACTIVE          40
```

### V\$CPOOL\_STATS View

V\$CPOOL\_STATS displays information about the DRCP statistics for an instance.

Table 6. V\$CPOOL\_STATS View.

Column	Description
POOL_NAME	Name of the Database Resident Connection Pool
NUM_OPEN_SERVERS	Total number of busy and free servers in the pool (including the authentication servers)

Column	Description
NUM_BUSY_SERVERS	Total number of busy servers in the pool (not including the authentication servers)
NUM_AUTH_SERVERS	Number of authentication servers in the pool
NUM_REQUESTS	Number of client requests
NUM_HITS	Total number of times client requests found matching pooled servers and sessions in the pool
NUM_MISSES	Total number of times client requests could not find a matching pooled server and session in the pool
NUM_WAITS	Total number of client requests that had to wait due to non-availability of free pooled servers
WAIT_TIME	Reserved for future use
CLIENT_REQ_TIMEOUTS	Reserved for future use
NUM_AUTHENTICATIONS	Total number of authentications of clients done by the pool
NUM_PURGED	Total number of sessions purged by the pool
HISTORIC_MAX	Maximum size that the pool has ever reached

The V\$CPool\_STATS view can be used to assess how efficient the pool settings are. This example query shows an application using the pool effectively. The low number of misses indicates that servers and sessions were reused. The wait count shows just over 1% of requests had to wait for a pooled server to become available. Increasing the number of pooled servers would reduce the number of waits:

```
SQL> select num_requests,num_hits,num_misses,num_waits
       from v$cpool_stats
       where pool_name = 'SYS_DEFAULT_CONNECTION_POOL';
```

NUM_REQUESTS	NUM_HITS	NUM_MISSES	NUM_WAITS
100031	99993	38	1054

If `oci8.connection_class` is set (allowing pooled servers and sessions to be reused) then `NUM_MISSES` is low. If the pool `maxsize` is too small for the connection load then `NUM_WAITS` is high:

NUM_REQUESTS	NUM_HITS	NUM_MISSES	NUM_WAITS
50352	50348	4	50149

If the connection class is left unset, the sharing of pooled servers is restricted to within each web server process. Even if the pool size is large, session sharing is limited causing poor utilization of pooled servers and contention for them:

NUM_REQUESTS	NUM_HITS	NUM_MISSES	NUM_WAITS
64152	17941	46211	15118

### V\$CPOOL\_CC\_STATS View

V\$CPOOL\_CC\_STATS displays information about the connection class level statistics for the pool per instance. The columns are similar to those of V\$CPOOL\_STATS described in Table 6, with a CCLASS\_NAME column giving the name of the connection sub-pool the results are for:

```
SQL> select cclass_name,num_requests,num_hits,num_misses
         from v$cpool_cc_stats;
```

CCLASS_NAME	NUM_REQUESTS	NUM_HITS	NUM_MISSES
HR.MYPHPAPP	100031	99993	38
SCOTT.SHARED	10	0	10
HR.OCI:SP:SdjxIx1Ufz	1	0	1

For PHP, the CCLASS\_NAME value is composed of the value of the username and of the oci8.connection\_class value used by the connecting PHP processes. This view shows an application known as MYPHPAPP using the pool effectively.

The last line of the example output shows a system generated class name for an application that did not explicitly set oci8.connection\_class. Pooling is not effectively used in this case. Such an entry could be an indication that a php.ini file is mis-configured.

For programs like SQL\*Plus that were not built using Oracle's Session Pooling APIs, the class name will be SHARED. The example shows that ten such connections were made as the user SCOTT. Although these programs share the same connection class, new sessions are created for each connection, keeping each cleanly isolated from any unwanted session changes.

## DRCP SCALABILITY BENCHMARK

The throughput with DRCP 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.

DRCP performance was measured using the enhanced PHP OCI8 1.3.2 Beta extension. The impressive results are shown below.

The test consisted of Apache servers concurrently executing PHP scripts that connected to a single database instance and performed queries. The PHP application included a call to `oci_pconnect()` (i.e. requesting a pooled server), execution of a `SELECT` statement, and a call to `oci_close()`. After completing the database operations, each script slept for one second (simulating the “think time” of a user).

The workload was generated using Apache’s ‘ab’ benchmark tool to send requests to the Apache servers to execute the PHP scripts. At steady state, each Apache server had 20 persistent connections to the database. Three Intel P4/Xeon machines running 32bit Red Hat Enterprise Linux 4 were used to host the Apache servers and to run ‘ab’.

A single database instance was hosted on a dual CPU Intel P4/Xeon 3.00GHz computer with 2GB RAM. The operating system was 32bit Red Hat Enterprise Linux 4. The operating system file descriptor limit was set at 35000. Oracle 11g R1 (patched for bug 6474441) was used for the client and database software. The DRCP pool was configured with one connection broker and 100 pooled servers. DRCP’s `MAXCONN_CBROK` parameter was 35000. The CPU and memory utilizations were captured using the `vmstat` command. The throughput was captured using Oracle AWR reporting.

The results show throughput scaled linearly as the number of incoming connections grew and that the memory required for the connections was minimal, again proportionate to the number of connections. Very large numbers of connections can easily be supported by commodity hardware.

Figure 4. Throughput and CPU Usage with DRCP.

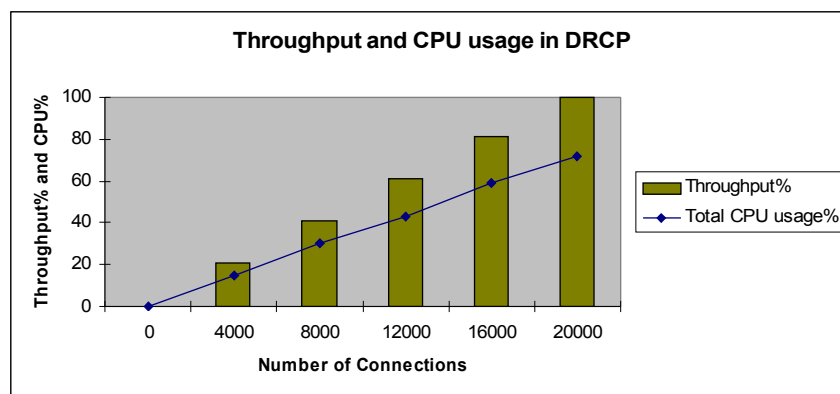
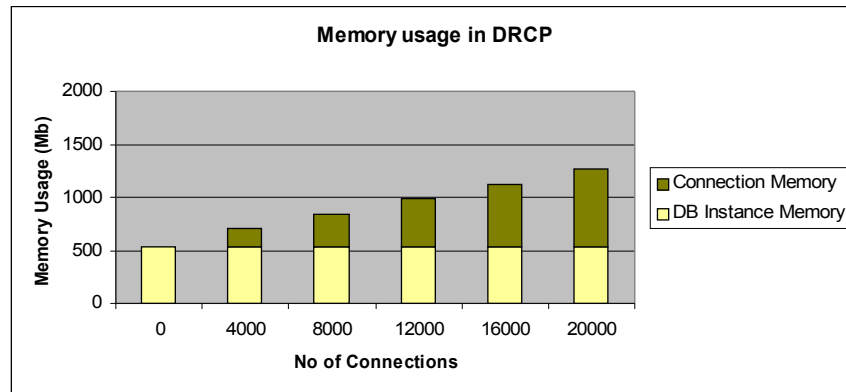


Figure 5. Memory Utilization with DRCP.



### HIGH AVAILABILITY WITH FAN AND RAC

Clients that run in high availability configurations such as with Oracle RAC or Data Guard Physical Stand-By can take advantage of Fast Application Notification (FAN) events to allow applications to respond quickly to database node failures. FAN support in PHP may be used with or without DRCP – the two features are independent.

Without FAN, when a database instance or machine node fails unexpectedly, PHP applications may be blocked waiting for a database response until a TCP timeout expires. Errors are therefore delayed, sometimes up to several minutes, by which time the application may have exceeded PHP's maximum allowed execution time.

By leveraging FAN events, PHP applications are quickly notified of failures that affect their established database connections. Connections to a failed database instance are proactively terminated without waiting for a potentially lengthy TCP timeout. This allows PHP scripts to recover quickly from a node or network failure. The application can reconnect and continue processing without the user being aware of a problem.

Also, all inactive network connections cached in PHP to the connection broker in case of DRCP, and persistent connections to the server processes or dispatcher in case of dedicated or shared server connections on the failed instances, are automatically cleaned up.

A subsequent PHP connection call will create a new connection to a surviving RAC node, activated stand-by database, or even the restarted single-instance database.

## Configuring FAN Events in the Database

To get the benefit of high availability, the database service to which the applications connect must be enabled to post FAN events:

```
SQL> execute dbms_service.modify_service(  
           service_name => 'SALES',  
           aq_ha_notifications => TRUE);
```

## Configuring PHP for FAN

With the enhanced OCI8 extension, a `php.ini` configuration parameter `oci8.events` allows PHP to be notified of FAN events:

```
oci8.events = On
```

FAN support is only available when PHP is linked with Oracle 10gR2 or 11g libraries and connecting to Oracle Database 10gR2 or 11g.

## Application Deployment for FAN

The error codes returned to PHP will generally be the same as without FAN enabled, so application error handling can remain unchanged.

Alternatively, applications can be enhanced to reconnect and retry actions, taking advantage of the higher level of service given by FAN.

As an example, the code below does some work (perhaps a series of update statements). If there is a connection failure, it reconnects, checks the transaction state and retries the work. The OCI8 extension will detect the connection failure and be able to reconnect on request, but the user script must also determine that work failed, why it failed, and be able to continue that work. The example code detects connection errors so it can identify if it needs to continue or retry work. It is generally important not to redo operations that already committed updated data.

Typical errors returned after an instance failure are “ORA-12153: TNS: not connected” or “ORA-03113: end-of-file on communication channel”. Other connection related errors are shown in the example, but errors including standard database errors may be returned, depending on timing.

```
function isConnectionError($err)  
{  
    switch($err) {  
        case 378: /* buffer pool param incorrect */  
        case 602: /* core dump */  
        case 603: /* fatal error */  
        case 609: /* attach failed */  
        case 1012: /* not logged in */
```

```

case 1033: /* init or shutdown in progress */
case 1043: /* Oracle not available */
case 1089: /* immediate shutdown in progress */
case 1090: /* shutdown in progress */
case 1092: /* instance terminated */
case 3113: /* disconnect */
case 3114: /* not connected */
case 3122: /* closing window */
case 3135: /* lost contact */
case 12153: /* TNS: not connected */
case 27146: /* fatal or instance terminated */
case 28511: /* Lost RPC */

    return true;
}
return false;
}

$conn = doConnect();
$error = doSomeWork($conn);
if (isConnectionError($error)) {
    // reconnect, find what was committed, and retry
    $conn = doConnect();
    $error = checkApplicationStateAndContinueWork($conn);
}
if ($error) {
    // end the application
    handleError($error);
}

```

### RAC Connection Load Balancing with PHP

PHP OCI8 1.3 will automatically balance connections across RAC instances with Oracle's Connection Load Balancing (CLB) to use resources efficiently.

It is recommended to use FAN and CLB together.

To enable CLB, the database service must be modified using `dbms_service` to send load events. Set the `clb_goal` parameter to `CLB_GOAL_SHORT` or `CLB_GOAL_LONG`.

For example:

```

SQL> execute dbms_service.modify_service(
           service_name => 'SALES',
           clb_goal => dbms_service.clb_goal_long);

```

Table 7. CLB *clb\_goal* parameter values.

Parameter Value	Parameter Description
CLB_GOAL_SHORT	Use for connection load balancing method for applications that have short-lived connections. This uses load balancing advisories if the <i>goal</i> parameter is not <i>GOAL_NONE</i> , otherwise a CPU utilization is used.
CLB_GOAL_LONG	Use for applications that have long-lived connections. This uses a simple metric to distribute load. This is the default.

No PHP script changes are needed. The connection balancing is handled transparently by the Oracle Net listener.

## CONCLUSION

The PHP OCI8 extension has been enhanced to take advantage of Oracle Database Resident Connection Pooling, and of Fast Application Notification. These features can be used together or separately. DRCP allows PHP applications to use a connection pool in the database that is shared across web servers. Applications can establish connections quickly and will use minimal database resources for large numbers of connections. FAN support allows PHP applications to quickly detect instance or hardware failures without blocking on a TCP timeout. Applications can use another RAC node or an activated standby database to continue database processing, minimizing user exposure to the failure.

## MORE INFORMATION

For information about PHP and the OCI8 extension see:

- PHP Developer Center on Oracle Technology Network <http://www.oracle.com/technology/tech/php/index.html>
- The Underground PHP and Oracle Manual <http://www.oracle.com/technology/tech/php/pdf/underground-php-oracle-manual.pdf>
- PHP OCI8 extension <http://pecl.php.net/package/oci8>. Refer to the README for up-to-date information.

For information on DRCP see:

- Oracle Database Concepts [http://download.oracle.com/docs/cd/B28359\\_01/server.111/b28318/toc.htm](http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/toc.htm)
- Oracle Database Administrator's Guide [http://download.oracle.com/docs/cd/B28359\\_01/server.111/b28310/toc.htm](http://download.oracle.com/docs/cd/B28359_01/server.111/b28310/toc.htm)
- Oracle Call Interface Programmer's Guide [http://download.oracle.com/docs/cd/B28359\\_01/appdev.111/b28395/toc.htm](http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28395/toc.htm)
- Oracle Database PL/SQL Packages and Types Reference [http://download.oracle.com/docs/cd/B28359\\_01/appdev.111/b28419/toc.htm](http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/toc.htm)
- Database Resident Connection Pooling (DRCP) Oracle Database 11g Technical White Paper <http://www.oracle.com/technology/tech/oci/pdf/oracledrcp11g.pdf>

For information on FAN and RAC see:

- Oracle Real Application Clusters Administration and Deployment Guide [http://download.oracle.com/docs/cd/B28359\\_01/rac.111/b28254/toc.htm](http://download.oracle.com/docs/cd/B28359_01/rac.111/b28254/toc.htm)



PHP Scalability and High Availability April 2008

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  
[oracle.com](http://oracle.com)

Copyright © 2008, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.