ORACLE DATABASE 12c

An Oracle White Paper
October 2014

Database Live Migration with Oracle Multitenant and the Oracle Universal Connection Pool (UCP) on Oracle Real Application Clusters (RAC)

ORACLE

Introduction

Consolidation in the data center is the driving factor in reducing capital and operational expense in IT today. This is particularly relevant as customers invest more in cloud infrastructure and associated service delivery like Database-as-a-Service (DBaaS). Hosting of many databases on the same shared infrastructure is a strategic component in this effort. In order to minimize capital and operational cost, an *efficient and elastic sharing of resources* is essential to increase the *density* (the number of supported databases or *tenants*) in such an environment. If the peak resource demand of all tenants exceeds the amount of physical resources, the deployment is referred to as *oversubscribed*. One of the key challenges for multitenant databases is to efficiently manage oversubscribed resources without violating a tenant's service level agreement (SLA) by dynamically allocating resources based on a tenant's current demand. The ability to quickly reassign resources from one tenant to another is therefore essential to ensure good Quality of Service (QoS) for all tenants especially for oversubscribed deployments.

Especially in DBaaS environments, the resource needs of tenants are typically unknown upfront, which makes initial resource allocations difficult and requires possibilities to easily reassign resources to correct imbalances. Within a server, multitenant databases can share resources such as CPU, memory, storage, and network between tenants and allocate them as needed when demand changes. However, at times the aggregate resource demand of all tenants may exceed the physical resources of a server. In such cases, some tenants may have to be migrated off an overloaded server to a lesser loaded server. Also maintenance operations such as installation of patches or hardware repair may require all tenants to be migrated off a server in order to take it down for maintenance. Such a migration must be completely transparent to the tenants with no service interruption or violation of SLA's, both for the tenants that are being migrated, as well as for all other tenants. **In other words, the migration of databases between servers must be performed online while the database is being accessed and updated, without any downtime or drop in throughput, and as little impact on response times as possible.**

This paper describes how to leverage Oracle Multitenant, Oracle Real Application Clusters (RAC), and the Oracle Universal Connection Pool (UCP) to implement seamless database live migration of Pluggable Databases (PDBs) between RAC nodes in a highly available and scalable database consolidation or DBaaS deployment. The described features result in a truly downtime-free live migration without any failed transactions and only minimal impact on the migrated database even for highly active OLTP workloads. Other tenants are not affected by the migration.

Technologies

Oracle 12*c* introduces **Oracle Multitenant** [1], a new database consolidation model in which multiple Pluggable Databases (PDBs) are consolidated within a Container Database (CDB). While keeping many of the isolation aspects of single databases, it allows PDBs to share the system global area (SGA) and background processes of a common CDB. Oracle Multitenant therefore allows PDBs to dynamically share CPU, memory, I/O, and networking resources within a server, resulting in significant efficiency gains and increased consolidation density compared to other deployment models such as instance consolidation and hardware virtualization technologies. A recent study shows that Oracle Multitenant on SuperCluster T5-8 gives 80% better performance and allows to consolidate 50% more active OLTP databases than instance-based consolidation. It reduces overall resource requirements to support 252 databases by 64 CPU cores, 368 GB of memory, and 225,000 storage IOPS [2].

Oracle Real Application Clusters (RAC) [3], introduced with the Oracle Database 9*i*, allows simultaneous access to a common database from multiple nodes of a cluster, each running one database instance. This enables scale-out of a database to many servers and ensures high availability (HA) of the database as the failure of a single database instance or server allows to maintain service on any of the remaining instances.

The **Oracle Universal Connection Pool (UCP)** [4] is a client-side connection pool for Java-based applications. It fully integrates with Oracle RAC and supports features such as Fast Connection Failover, run-time connection load-balancing. In version 12.1.0.2, UCP adds support for seamless PDB live migration through service relocation.

Combining Oracle Multitenant with Oracle RAC allows to build a highly available and scalable infrastructure for database consolidation or Database-as-a-Service (DBaaS). In such an environment, the Container Database (CDB) spans multiple servers with one CDB instance per node. Pluggable Databases (PDBs) can either be opened in all CDB instances, or selectively in only some of the instances for improved data locality. Accessing a PDB in only a single node largely eliminates the need for cross-instance communication and allows near-linear scale-out with the number of RAC nodes. Leveraging **UCP's service relocation capabilities**, a PDB can seamlessly be migrated from one RAC node to another to facilitate load-balancing between nodes or planned maintenance operations on one of the nodes. The migration frees up all CPU and memory resources on the original node, which immediately become fully available to other tenants. Since all nodes access the same database on a shared storage, only cache content (buffer cache) needs to be transferred between nodes, which makes the migration far more efficient than live migration of Virtual Machines (VMs), which have to also transfer process, operating system, file system cache and other memory pages. While the term *migration* is often associated with migration of VMs, the migration described here is only implemented at a different layer of the stack. It has all the attributes typically associated with live migration.

Service Configuration for PDB Live Migration

Services for Pluggable Databases on Oracle RAC

Databases, including PDBs, are accessed through *Oracle Net Services*, which allow clients to connect to a database using a *service name*. Services can dynamically be started on some or all nodes of a cluster. Each PDB has a default service with the same name as the PDB. In Oracle RAC, this default service should only be used for administrative purposes. Dynamic services should be created for each PDB to handle application payload. When a CDB instance is started on a node, PDBs are *not* automatically opened in the CDB instance and therefore not accessible on this instance until they are either explicitly opened through a command (i.e. `alter pluggable database` *`pdbname`* `open`) or implicitly opened when a non-default service for the PDB is started on this node. Services control in which nodes clients will access a PDB.

Data Locality

With many PDBs consolidated in a CDB, a single PDB typically requires only a small fraction of the physical resources of a node. If services for a PDB run on only a single or a subset of the nodes, clients will access a PDB only in those nodes, which avoids identical data blocks for this PDB to be cached in multiple nodes and reduces the need for inter-node traffic for cache block transfers.

Service Configuration for Admin-Managed Databases

For admin-managed databases, services can be configured to run on either a single node, some nodes, or all nodes of a cluster.

For example, in a 2-node cluster with a database *cdb* and instances *cdb1* and *cdb2*, a service *svc1* for PDB *pdb1* could be configured to only run on one instance at a time, but with the ability to run on either of the two nodes:

```
srvctl add service -d cdb -s svc1 -pdb pdb1 -preferred cdb1
-available cdb2
```

On a 4-node cluster with a database *cdb* and instances *cdb1*, *cdb2*, *cdb3*, and *cdb4*, a service *svc1* for PDB *pdb1* could be configured to run on any node, but only one at a time (preferrably *cdb1*):

```
srvctl add service -d cdb -s svc1 -pdb pdb1 -preferred cdb1
-available cdb2,cdb3,cdb4
```

For the same 4-node cluster, the following command would allow the service to run on two nodes simultaneously:

```
srvctl add service -d cdb -s svc1 -pdb pdb1 -preferred cdb1,cdb2
-available cdb3,cdb4
```

Service Configuration for Policy-Managed Databases

For policy-managed databases, services are either singleton or uniform services. Singleton services run only on a single node of a server pool at a time, while uniform services always run on all nodes of a server pool. The initial placement of singleton services is not controlled by the administrator, but instead automatically done by Oracle Clusterware based on load-balancing across all nodes in the server pool.

The following command creates a singleton service svc1 for PDB pdb1 in a server pool pool1 with a container database cdb:

```
srvctl add service -d cdb -s svc1 -pdb pdb1 -c singleton -g ora.pool1
```

Since uniform services always run on all nodes of a server pool and live migration of PDBs as described in this paper is based on the concept of service relocation, the use of uniform services is not supported in this context.

Configuring and Using PDB Live Migration

A PDB can be migrated between RAC nodes either using *service relocation* or *PDB relocation* commands. Service relocation requires the use of the Oracle Universal Connection Pool (UCP) and allows for a gradual migration of connections free of downtime and with minimal impact for the application. PDB relocation supports the use of any client as it implements migration control on the server-side, but currently only supports an immediate instead of a gradual migration of load.

PDB Live Migration using Service Relocation

The use of UCP in combination with service relocation allows to implement PDB live migration without any downtime or failure of transactions for the migrated tenant. Since load is gradually migrated from one node to another over a configurable period of time, the impact on the application's response times during migration can be kept small. This migration method is fully implemented in the UCP connection pool and completely transparent for the application.

A PDB is migrated from a source node (where it is currently being accessed) to a target node (where it shall be accessed) by

1. stopping the service on the source node (without disconnecting or affecting any existing sessions)
2. starting the service on the target node and opening the PDB on the target node (if it has not been opened there before)
3. gradually disconnecting client connections from the source node and reconnecting them to the service that is now running on the target node
4. optionally closing the PDB on the source node to prevent further access and force any remaining sessions to be disconnected.

Service Relocation

A service can be relocated from one node (where it is currently running) to another node (where it is allowed to run) with the *srvctl relocate service* command. This applies both to services for admin-managed databases as well as singleton services for policy-managed databases.

The following command will relocate service *svc1* from an admin-managed instance *cdb1* (where it is currently running) to instance *cdb2* (where it is currently not running):

```
srvctl relocate service -d cdb -s svc1 -oldinst cdb1 -newinst cdb2
```

For policy-managed databases, the service *svc1* can be relocated fron *node1* (where it is currently running) to *node2* (where it is currently not running):

```
srvctl relocate service -d cdb -s svc1 -currentnode node1 -targetnode
node2
```

The above commands will first stop service *svc1* on instance *cdb1*, and then start the service on instance *cdb2*. If the associated PDB is not yet opened in instance *cdb2*, it will implicitly be opened there. Note that these commands will *not* close the PDB in instance *cdb1*, and will *not* disconnect any clients from *cdb1*. In other words, the above commands have *no immediate effect* on any ongoing sessions, which all continue without any interruption on instance *cdb1*.

During the relocation, which typically takes only a few seconds, no new connections can be established to this service, unless the service is also running on another instance (for example *cdb3*). Once the PDB has been opened on *cdb2*, new incoming connections will automatically be established on *cdb2* (and other instances where the service may be running).

Oracle Notification Service (ONS)

Clients can subscribe to the Oracle Notification Service (ONS) to receive notification events when services are started or stopped. In the above example, a client that had registered itself with ONS for such events would have received a “service svc1 down on cdb1” event, followed by a “service svc1 up on cdb2” event. In order for UCP to detect that a service has been relocated, it must register itself as an ONS listener.

UCP Configuration

UCP version 12.1.0.2 adds support for gradual migration of client connections from one node to another in case of service relocation. The following configuration settings are required to enable UCP for seamless PDB live migration:

- register UCP to receive ONS events
- configure desired connection draining time

The following Java code illustrates how to enable UCP to receive ONS events from a cluster with the nodes node1 and node2:

```
PoolDataSource pds = ...;
pds.setONSConfiguration("nodes=node1:6200,node2:6200");
```

The following Java property was introduced with UCP version 12.1.0.2 to configure the desired connection draining time (the period of time over which established connection should be migrated from one node to another):

Parameter	Description
oracle.ucp.PlannedDrainingPeriod	The time period over which to migrate connection. Once the service has come up on the target node, UCP will migrate idle (unborrowed) connections at a steady rate (computed as *CurrentPoolSize / PlannedDrainingPeriod* connections per second) by disconnecting them from the source node and reconnecting them to the service (on the target node).

The following Java code illustrates how UCP can be configured to migrate all connections in the pool over a time of 60 seconds:

```
System.setProperty("oracle.ucp.PlannedDrainingPeriod", Integer.toString(60));
```

If set, the *PlannedDrainingPeriod* determines the migration duration from the time the service has come up on the target node. After a service goes down, UCP will not disconnect client sessions, but instead wait for the service to come up on another node. Once the service is up, UCP will start migrating connections over the time configured through this parameter. A small value will accelerate the migration, but might cause applications to experience higher response times as requests on the target node hit a cold buffer cache. A larger value migrates work more gently and gives the buffer cache on the target node more time to warm-up, which in consequence leads to reduced impact on the application, but a longer overall migration duration.

The following Java code illustrates all necessary steps to setup a connection pool with UCP:

```
import oracle.ucp.*;
import oracle.ons.*;

System.setProperty("oracle.ucp.PlannedDrainingPeriod", Integer.toString(60));

PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
UniversalConnectionPoolManager manager =
UniversalConnectionPoolManagerImpl.getUniversalConnectionPoolManager();
pds.setConnectionPoolName("MyPool");
pds.setONSConfiguration("nodes=node1:6200,node2:6200");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL(config.getParameterValue("jdbc:oracle:thin:@my-scan:1521/svc1"));
pds.setUser(config.getParameterValue("username"));
pds.setPassword(config.getParameterValue("password"));
pds.setInitialPoolSize(100);
pds.setMinPoolSize(100);
```

```
pds.setMaxPoolSize(100);
manager.createConnectionPool((UniversalConnectionPoolAdapter) pds);
manager.startConnectionPool("MyPool");
```

Closing of PDB on source node

In order to disconnect any potentially misbehaving clients and guarantee that *pdb1* can no longer be accessed on instance *cdb1* after all connections have been migrated off this node, it can be explicitly closed with the command `alter pluggable database pdb1 close immediate;`. A close of the PDB will implicitly also flush its buffer cache.

Clients should be given a reasonable amount of time to voluntarily migrate all their connections off a node where the service is stopped. A database service provider could for example close the PDB 5 minutes after initiating the service relocation, which guarantees that after 5 minutes, all pending sessions will be closed. This gives clients sufficient time to migrate all their connections in time, for example over a period of 60 seconds.

PDB Live Migration using PDB Relocation

Alternatively to service relocation, a PDB can also be migrated through a *PDB relocate* command. In this case, the migration of connections is controlled by the server, which consequently works in combination with any client and gives the database service provider full control over the migration. This method may be more disruptive to the client as it does not give the client any control as to when and at which rate to migrate connections.

PDB Relocation

A PDB can be relocated from one node (where it is currently running) to another node (where it's service is allowed to run) with the *alter pluggable database relocate* command. This applies both to services for admin-managed databases as well as singleton services for policy-managed databases.

The following command will relocate PDB *pdb1* from instance *cdb1* (where it is currently running) to instance *cdb2* (where it is currently not running):

```
alter pluggable database pdb1 close immediate relocate to 'cdb2';
```

The above command will open the PDB in instance *cdb2* and then immediately terminate all client sessions on the original instance *cdb1*. After that, it will close the PDB on *cdb1* and flush its buffer cache. Clients will then need to reconnect to the (now migrated) service themselves. Connection pools such as UCP will do this automatically and transparent for the client based on the configuration of pool sizes.

In a future version, the PDB relocate command may also support a gradual instead of immediate termination of client sessions.

PDB Live Migration Tests

This section provides test results that demonstrate how seamless a PDB can be migrated from one RAC node to another. In these tests, 4 PDBs have been deployed on a 2-node RAC CDB. Each PDB runs an OLTP workload that selects and updates rows at a rate of 1,500 - 6,000 transactions per second (tps) and is accessed from clients using a UCP connection pool.

PDB	Service	Preferred Nodes	Available Nodes	Database Size	Connection Pool Size	Transaction Rate
pdb1	svc1	node1	node2	14 GB	100	2,000 tps
pdb2	svc2	node2	node1	14 GB	100	1,500 tps
pdb3	svc3	node1	node2	14 GB	200	6,000 tps
pdb4	svc4	node2	node1	14 GB	100	2,500 tps

Initially, the service for each PDB runs on the preferred node (*node1* for *pdb1* and *pdb3*, and *node2* for *pdb2* and *pdb4*). Since *pdb3* is running high load, *node1* has to handle an aggregate transaction rate of 8,000 tps and is therefore much higher loaded than *node2*, which only needs to serve 4,000 tps. To correct the overload on *node1*, *pdb1* is then being migrated to *node2*, where it has not been opened before, and none of its blocks are cached in the buffer cache.

PDB Live Migration using Service Relocation and UCP

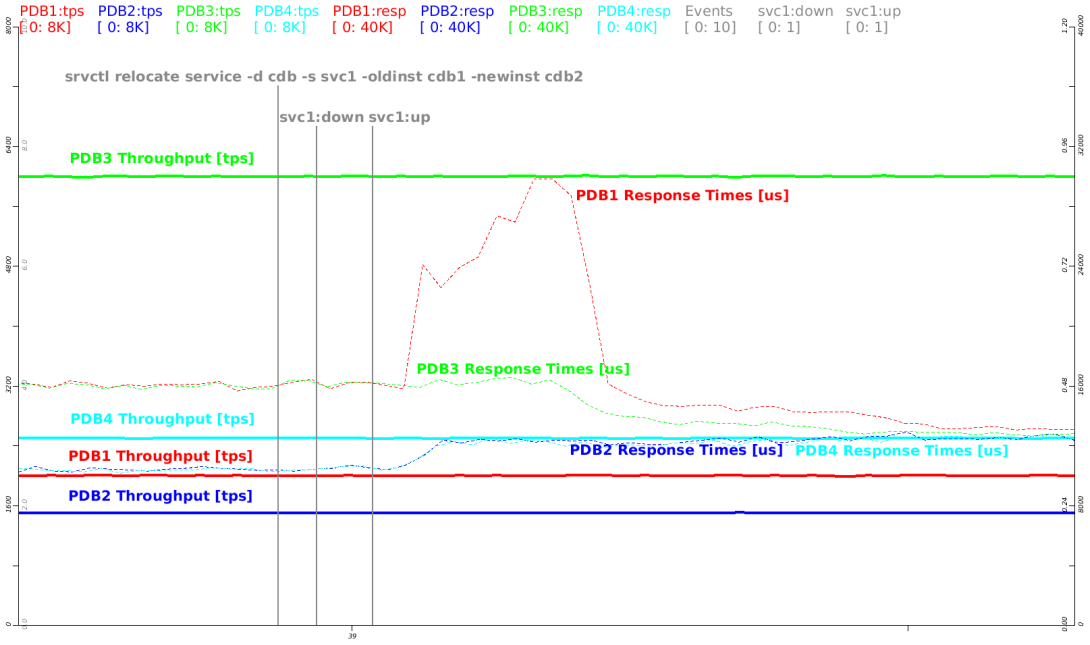
This test demonstrates live migration of a PDB using service relocation with the command:

```
srvctl relocate service -d cdb -s svc1 -oldinst cdb1 -newinst cdb2
```

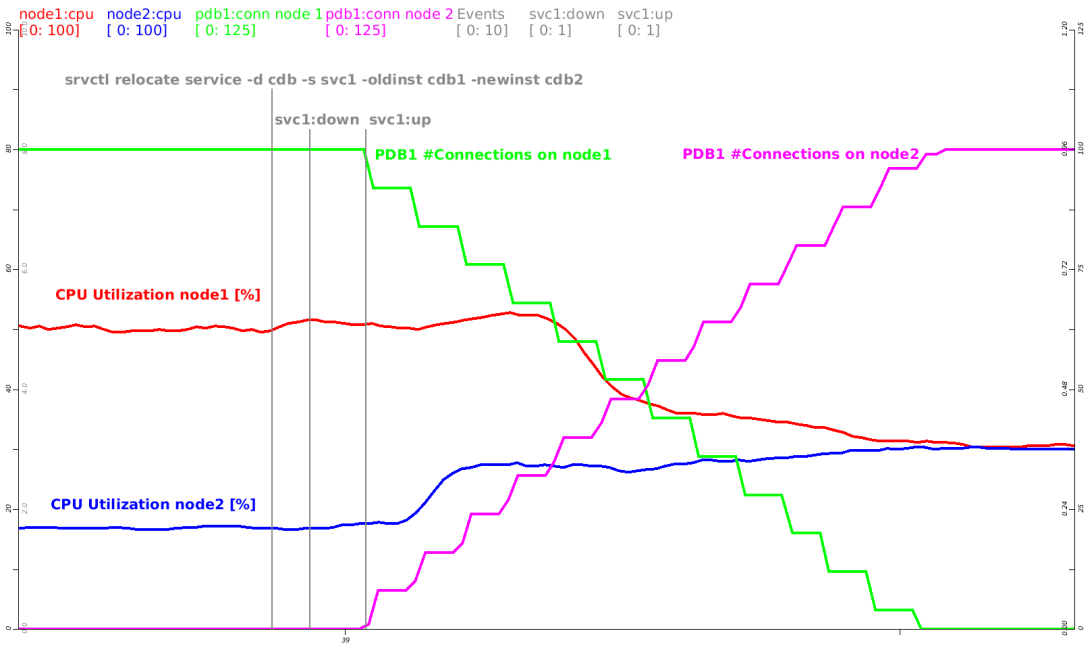
UCP for *pdb1* has been configured with a *PlannedDrainingPeriod* of 60 seconds.

Graph 1 shows the tenant's view with throughput and response times for all four tenants. The PDB that is being migrated (*pdb1*) is servicing 2,000 transactions per second. During the migration, throughput for this PDB (as well as for all others) is not affected at all: It continues to serve transactions at a rate of 2,000 tps without any drop in throughput or failed transactions. Response times during migration increase temporarily as first requests on the target node hit a cold buffer cache, but quickly stabilize again and reach an even lower level than before the migration as the PDB has been migrated to a less utilized node. Response times of other tenants are not affected by the migration at all and only change slightly as system utilization changes.

Graph 2 shows the provider's view: CPU utilization and the number of connections for *pdb1* per node. In the beginning of the test, CPU utilization is clearly unbalanced between both nodes. Shortly after the service relocation is initiated, the service goes down on *node1*, and a few seconds later comes back up on *node2*. Immediately after the service is up again, UCP begins to migrate connections by disconnecting them from *node1* and reconnecting them to *node2* over a period of 60 seconds (as configured). During the migration, CPU utilization on *node1* drops and slowly increases on *node2*, until both nodes serve a throughput of 6,000 tps and have a balanced CPU utilization.



Graph 1: PDB Live Migration through Service Relocation: Tenant's View



Graph 2: PDB Live Migration through Service Relocation: Provider's View

Summary

Live migration is an essential feature in consolidated or Cloud-based environments in order to balance load across servers or take systems down for maintenance without affecting any of the hosted tenants. This paper has described a technique for database live migration based on Oracle Multitenant, Real Application Clusters (RAC), and the Universal Connection Pool (UCP), that is capable of migrating highly active OLTP databases from one physical server to another without any downtime and only minimal impact on the migrated tenant.

References

[1] Oracle Multitenant. An Oracle White Paper, June 2013. http://www.oracle.com/technetwork/database/multitenant/overview/multitenant-wp-12c-2078248.pdf

[2] Oracle Multitenant on SuperCluster T5-8: Scalability Study. An Oracle White Paper, April 2014. http://www.oracle.com/technetwork/database/multitenant/learn-more/oraclemultitenantt5-8-final-2185108.pdf

[3] Oracle Real Application Clusters (RAC). An Oracle White Paper, June 2013. http://www.oracle.com/technetwork/database/options/clustering/rac-wp-12c-1896129.pdf

[4] Application Development: JDBC and UCP. Oracle Technology Network. http://www.oracle.com/technetwork/database/application-development/index-099369.html

ORACLE

Database Live Migration with Oracle Multitenant on Oracle Real Application Clusters (RAC)
October 2014
Authors: Nicolas Michael, Yixiao Shen

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