Application Development Best Practices for Oracle Real Application Clusters (RAC)

A Developer's Checklist

ORACLE WHITE PAPER | JULY 2016

Table of Contents

Table of Contents 1

Introduction 1

Terminology 2

RAC Overview 3

Scalability Considerations 4

High Availability Considerations 8

Functionality Considerations 11

RAC Database Application Developer’s Checklist 13

Tools for tuning applications 13

Conclusion 14

References 15

Introduction

This white paper can serve as a checklist for application developers to develop scalable, highly available, and high performance applications for Oracle Real Application Clusters (RAC) database environments. This white paper assumes that the Oracle RAC database environment has been setup using relevant general best practices. The intended audience for this white paper is application developer community and database administrators.

Terminology

Abbreviation	Description
AC	Application Continuity
API	Application Programming Interface
AQ	Advanced Queuing
Cache Fusion	Ability to ‘fuse’ multiple local caches on servers into a global, coordinated, and synchronized cache
FAN	Fast Application Notification
FCF	Fast Connection Failover
FTS	Full Table Scan
GCS	Global Cache Service
HA	High Availability
JDBC	Java Database Connectivity
OCI	Oracle Call Interface
Oracle XA	Oracle’s implementation of X/Open Distributed Transaction Processing interfaces
RAC	Real Application Clusters
SGA	System Global Area
TAF	Transparent Application Failover
TG	Transaction Guard
UCP	Universal Connection Pooling

RAC Overview

Oracle Real Application Clusters (RAC) is an optional but a very popular feature of Oracle Database that allows multiple databases instances to be clustered to simultaneously access a common, shared, physical database. Such clustering allows for database high availability and scalability as the database can tolerate N-1 (N= number of nodes in the cluster) node failures while additional servers can be easily be added to increase capacity. Oracle RAC uses Oracle Clusterware as the underlying glue to bind the servers together in a cluster so they appear as a single system to applications and end users. A dedicated, high speed, low latency private network is used to keep the servers of the cluster in synchronization.

Each instance in an Oracle RAC Database runs on a separate cluster member (node) and has its own REDO thread and UNDO tablespace. Each instance has its own SGA that it manages. However, all instances share their cache with other instances of the cluster over the dedicated, private interconnect. If an instance has data in its cache that is required by another instance then it is shipped across the high speed interconnect while maintaining overall data consistency and coherency. The database data files, control files, and redo log files reside on highly available shared storage and are accessible from all cluster nodes with Oracle Clusterware managing the communication. Figure 1 illustrates basic application implementation architecture for a two node RAC cluster.

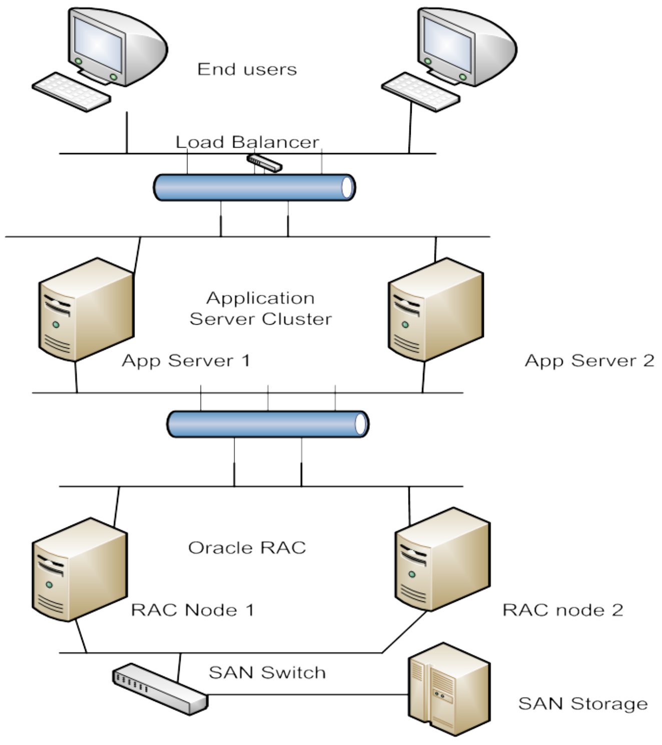

Figure 1 Application deployment architecture using Oracle RAC (a two node cluster example)

For an application developer, it helps to understand the underlying database architecture in order to apply best practices when designing and building an application that can fully leverage the benefits provided by Oracle RAC. Familiarity with the architecture also helps to avoid common but preventable problems. This includes writing efficient applications that leverage cluster capabilities. This white paper outlines these aspects from an application designer and developer's perspective.

Scalability Considerations

Scalability refers to the architecture's ability to increase its capacity by utilizing additional resources as workload requirements increase. In case of an Oracle RAC database environment this can mean being able to accommodate additional workload by simply adding one or more database instances.

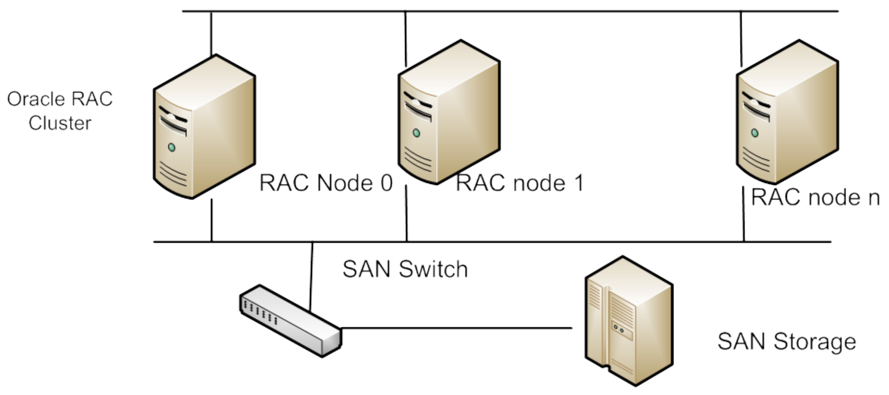

Figure 2 RAC Scalability (adding nodes to cluster)

As the cluster architecture is scaled, the availability of additional (incremental) CPU and memory (cache) resources at the instance level and a scalable storage solution act as key enablers.

Ensure application is designed with scalability in mind

A well written application is scalable by nature, as it is written such that it does not produce any hotspots or points of contention or serialization during execution. If an application is written that way, it will perform well in both clustered and non-clustered environments. However, applications are not always written with scalability in mind. Many times, the focus of a developer is to produce functionality. Performance and scalability are usually after thoughts. The pre-existing application performance, become more pronounced in a cluster environment and thus application tuning may be required before porting an application to a clustered environment. Oracle Corporation has always maintained that an application should scale well in a non-RAC environment for it to scale well in a RAC environment.

When an applications workload processing increases in direct proportion to the level of resources, it is said to scale linearly. When moving existing applications from a single instance database environment to a RAC database environment it is important to first test the scalability of the application and perform any tuning necessary. Oracle Real Application Testing is designed and optimized for testing database tier infrastructure changes using real application production workloads to validate database performance in your test environment.

There are best practices that developers sometimes overlook when writing applications. The below habits will result in a scalability lacking application:

- Not reusing code appropriately (e.g., not using bind variables and parsing code repetitively)
- Using full table scans excessively and not leveraging indexing effectively
- Not using connections and connection pools efficiently and optimally, etc.

Details of the above can be found in Oracle documentation. Additionally, in order for an application to scale the database should be created with best practices in mind. Oracle RAC best practices for database implementations are well documented and available to database administrators and architects.

The following considerations apply for designing scalable applications.

Design objects and schema to minimize or eliminate hot spots

To maximize scalability of applications with RAC backend database architecture, it is important to minimize hot spots. Hot spots occur when multiple active sessions, specifically from separate instances, access a set of common storage blocks frequently. Tables or parts of table with frequent updates and reads and a high row density per database block can become “globally hot” because of serialization. Some considerations in this regard for minimizing hot spots are as follows:

- **Use appropriate row density**. All database IO is in the units of database blocks. Larger database blocks may typically pack more rows per block. If these rows are concurrently accessed and locked from multiple instances, it can result in hot blocks. Using smaller block size or making storage blocks less dense can be useful in such situations. Oracle database allows creation of tablespaces with different block sizes within the same database.
- **Consider using smaller block sizes for index tablespaces**. Due to fewer data elements in an index entry, indexes may have an even higher row density for the same block size as the table. Using a smaller block size for index tablespaces may sometimes alleviate such hot spots.
- **Avoiding hot spots using localized access**. In many cases, applications may find it useful and practical to localize access to globally hot blocks from one instance instead of multiple instances. This means while the overall application may continue normal access to a RAC database, specific functionality may only take place from a local instance and failover when needed. This may also be called workload partitioning.
- **Leverage HASH partitioned tables and indexes**. Using hash partitioning may help distribute IO somewhat randomly and minimize or eliminate hot spots. Use of hash partitioned global indexes can spread out the contention over multiple partitions and alleviate hot spots as index entries are hashed to different partitions based on the partitioning key. Note that hash partitioned global indexes can be utilized even for non-partitioned tables. Refer to Oracle Support note 220970.1 - RAC: Frequently Asked Questions and this OTN note for more information.
- **Leverage REVERSE KEY indexing**. Indexes that correspond to sequences (sequentially increasing numbers) may always have a hot spot on the right edge as multiple values are inserted into the same blocks by concurrent multiple sessions. In many situations such sequentially generated numbers may not be necessary. In other situations, a reverse key index (instead of a regular index) may eliminate the hot spot.
- **Optimize Index maintenance using local indexes**. Use of local indexes at partition level can reduce overhead during index maintenance. A locally partitioned index has a one to one mapping with the associated table partitions. Local index partitioning facilitates easier and more efficient index maintenance as each index partition is independent of the other. Accordingly, use of local indexes reduces overall overhead during index maintenance operations. For more information see the Oracle documentation.

Use full table scans optimally. Use full table scans (FTS) only when necessary and optimize index based access to data when possible and appropriate. RAC does check whether a fully sequential read may be more beneficial than collecting data from remote instances. If the result is to require active blocks from the other instances this causes extensive cache coordination and synchronization. Like with single instance databases, FTS should thus be used prudently in a clustered database environment.

Test application for scalability

Testing for scalability before production cutover can expose hot spots and provide you an opportunity to tune your application in advance. It is best to do scalability testing in a production like environment with production like representative workload. Oracle Load Testing 12c can be a useful tool to conduct scalability testing of your applications. For more details please review the Data Sheet for Oracle Load Testing. Oracle Real Application Testing (RAT) can be used to record workload from a standalone database and replay it in a RAC database. For more information on RAT see the Oracle 12c RAT Overview white paper. Consider as well the opportunity to setup a RAC database in the Oracle public cloud. This provides a fast and cost effective way to create a RAC database environment for a limited time. On Oracle Engineered Systems, such as Oracle Database Appliance, you can dynamically configure CPU and Memory resources and use single instance or RAC instance configurations to test the application for vertical and horizontal scalability.

Use Distributed Transaction Processing (DTP) services for XA applications

Use Distributed Transaction Processing (DTP) for applications that use XA transactions where multiple branches of a global transaction accessing the database could benefit by utilizing the same RAC database instance to service the transaction. Connection pools that provide RAC XA Affinity do not require the use of DTP services. For example, Oracle WebLogic's Multi-pool Data Source and Grid Link provide XA Affinity optimizations. For more information review the white paper XA and Oracle controlled Distributed Transactions .

Use Automatic Segment Space Management (ASSM) for RAC databases

Automatic Segment Space Management greatly simplifies segment space management and eliminates the need to manually manage storage parameters such as PCTUSED, FREELISTS, and FREELIST GROUPS when new instances are brought online. Refer to Oracle Support note 180608.1 - Automatic Space Segment Management in RAC. Automatic Segment Space Management allows for more efficient space management inside database data blocks and should be used in clustered database environments.

Use database sequences optimally

Sequence is a schema object that belongs to a database and is shared among all instances belonging to the database. One approach to scale the sequence object is to use caching of sequence numbers locally at the instance level in a clustered environment. While this can lead to gaps in the sequence number generation, it may not be a concern for a majority of applications and can be a viable solution for mitigating sequence contention. Note that caching may also result in use of un-ordered sequence numbers at different instances but in most cases, applications can be designed to be resilient to such scenarios in the interest of achieving higher scalability. Refer to My Oracle Support (MOS) note 62002.1 - Caching Oracle Sequences. Sequence number caching can produce significant performance gains for an application running in a clustered database environment. For example, see the elapsed time (performance) of the following two scenarios.

```
Create sequence mycached_seq start with 1 increment by 1 cache 1000000;

set timing on;

declare

    xyz number;

    i number;

    begin
    for i in 1..100000
    loop
    select mycached_seq.nextval into xyz from dual;
    end loop;
    end;
/
PL/SQL procedure successfully completed.
Elapsed: 00:00:10.38
```

Now try the same anonymous code with a NOCACHE sequence:

```
Create sequence mynocached_seq start with 1 increment by 1 NOCACHE;
Set timing on;
declare
    xyz number;
    i number;
    begin
    for i in 1..100000
    loop
    select mynocached_seq.nextval into xyz from dual;
    end loop;
    end;
/
PL/SQL procedure successfully completed.
Elapsed: 00:03:57.44
```

Listing 1: Sequence number caching

It is clear from the above example that caching sequences locally at instance level reduces the serialization bottleneck in a clustered environment.

Use advanced queuing optimally

Oracle RAC can be used to provide high availability and scalability to Oracle Advanced Queuing (AQ). For queues being accessed exclusively via the JMS drivers, JMS Sharded Queues provide optimized performance on Oracle RAC with improved manageability. For queues being accessed through the AQ PL/SQL APIs, the performance of AQ can be improved by allowing different queues to be managed by different RAC instances. Different instance affinities or preferences can be specified for the queue tables that allows for parallelization of queue operations on different queues. Oracle recommends setting instance affinities for the queue tables. Setting instance affinities allows distribution of the background processing for queue-monitor scheduling and propagation. If instance affinity is not set, queue table affinity is allocated arbitrarily amongst the available instances, which can cause pinging between the application accessing the queue tables and the queue-monitor process monitoring the queue under high loads. Please see as well the recommendation "Use Advanced Queuing (AQ) or Java Messaging Service (JMS) instead of DBMS_PIPE" in the later chapter ‘Functionality Considerations’.

High Availability Considerations

An important reason for application and database architects to use Oracle RAC is application High Availability (HA). Availability of the database service provided via multiple instances can make applications resilient to outages and provide transparent protection against planned and unplanned downtime. In the event of an instance or node failure in a clustered environment, the database service is expected to be available from the remaining instance(s).

Transaction Guard and Application Continuity (AC)

From Oracle Database 12.2 on the Transaction Guard and the Application Continuity features are available.

Transaction Guard provides a generic tool for applications to use for at-most-once execution in case of planned and unplanned outages. Applications use the logical transaction ID to determine the outcome of the last transaction open in a database session following an outage. Without Transaction Guard, applications that attempt to replay operations following outages can cause logical corruption by committing duplicate transactions.

Application Continuity is a feature that enables the replay, in a non-disruptive and rapid manner, of a request against the database after a recoverable error that makes the database session unavailable. The request can contain transactional and non-transactional work. After a successful replay, the application can continue where that database session left off, instead of having users left in doubt not knowing what happened to their funds transfers, flight bookings, and so on, and avoiding the need to reboot mid-tier machines to recover from logon storms. With Application Continuity, the end user experience is improved by masking many outages, planned and unplanned, without the application developer needing to attempt to recover the request. For more information about Application Continuity refer to Oracle white paper Application Continuity with Oracle Database 12c . Application Continuity demonstration can be accessed at http://www.oracle.com/technetwork/products/clustering/ac-overview-1967264.html

Transaction Guard is used by Application Continuity and automatically enabled by it, but it can also be enabled independently. Transaction Guard prevents the transaction being applied by Application Continuity from being applied more than once. If the application has implemented an application-level replay, then it requires the application to be integrated with transaction guard. For more information about Transaction Guard refer to Oracle white paper Transaction Guard with Oracle Database 12c.

Note: Unless you use Transaction Guard, RAC does not protect against the loss of in-flight transactions in case of an instance failure, although the database service as such persists on the remaining nodes in the cluster.

With awareness of the RAC database architecture applications can be architected to handle transient failures in a graceful and transparent manner.

Use Dynamic Database Services

In order to maintain high availability transparently, applications should connect to the database using service names. Note that all advanced features of Oracle RAC databases are based on services.

a. Database administrators should create a specific service for a given application.

b. Database connection strings and / or URLs should always use service and not instance ID (SID), for example, in JDBC connection string for Java applications.

Use standard Oracle connection pooling

The Fast Application Notification (FAN) functionality of Oracle RAC allows applications to receive notifications on database instance outage events. This allows applications to react appropriately during backend outages and for application developers to exercise better control over application behavior during such events. The FAN functionality is integrated in most of Oracle's Connection Pools and allow applications to take advantage of the HA capabilities of Oracle RAC. Application architects should avoid using home grown connection pools and instead use Oracle provided connection pools. For example,

- Oracle Universal Connection Pool (UCP) for Java (J2SE based applications)
- GridLink data sources for WebLogic based applications
- FAN/OCI pools for OCI connections integrated with TAF

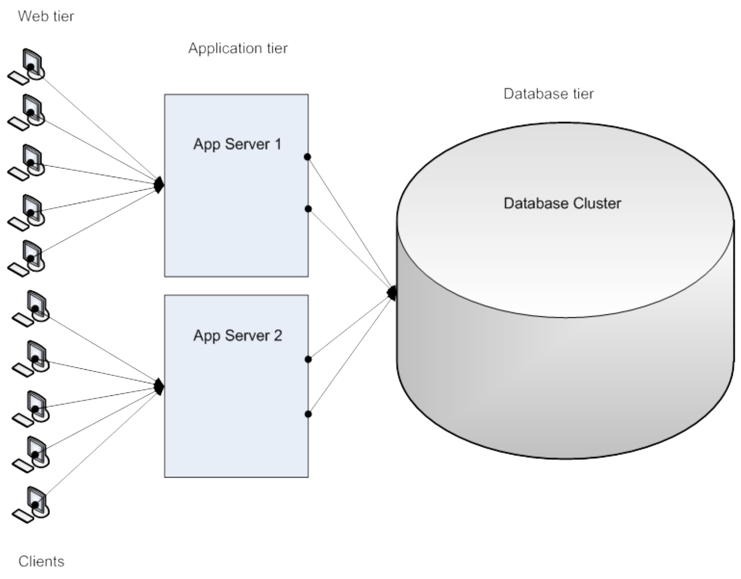

Figure 3 Illustration of basic connection pooling

As illustrated in figure 3, connection pools effectively consolidate large numbers of user requests, typically originating as connections from the client systems to web servers, into fewer shared connections between the application tier and database tier to serve the data requests. This typically results in higher efficiency and reduced overhead of user connection administration and management on part of the database tier.

Connection-time and run-time load balancing

Typically applications connecting to a RAC back end database establish connections to all instances that offer the requested service. Oracle connect-time load balancing ensures that a new client connection to the database is established with the least loaded database instance as of that time.

Oracle connection pools (UCP and GridLink) typically establish a set of persistent, shared connections to the database that service transient application connection requests. Oracle connection pools provide the run-time load balancing feature whereby they ensure that backend instances are evenly loaded as application processing continues. When an application requests a connection to the database, it is normally handed over a pre-established connection from the pool. Oracle connection pools enabled with runtime load balancing feature use load metrics information sent as periodic notifications from RAC instances to determine the suitability of a pre-established connection to accept a new application connection request. This ensures real-time load balancing in distributing database workload across RAC instances and an efficient clustered database environment.

In addition, applications should be architected so that they keep a connection open only as long as required and release (close) it as soon as possible (or the work completes). This eliminates unnecessary overhead related to connection operations and management.

Build applications for transparent connection retry

As explained above Connection Pools, AC and the other features aid with a transparent connection retry. The approach described in this section is mainly required if the formerly named and explained features cannot be used. Applications running with a RAC backend database should be architected to handle connection failures transparently. This requires retrying to establish a connection and continuing with the application transaction. For example, when a JDBC connection is used, an application may catch the SQL_RECOVERABLE_EXCEPTION exception and retry the ongoing transaction once a new connection is established with a surviving database instance.

Once an application has established a connection, it may use it for one or more transactions before returning it to the connection pool. If an instance outage occurs, the application receives an exception. At the time of an exception, the application may be briefly idle between transactions, in the middle of an active transaction. If the application is not in the middle of a transaction, then a retry is simple. When an exception is received in the middle of a transaction, retrying is more involved and the application code needs to handle this situation appropriately. For example, see below the typical application logic for a typical application using the retry functionality to maintain transparent connection:

```
Do pre-transaction-processing (if any)
WHILE number of specified retries > 0
TRY
  Decrement number of specified retries
  Do main transaction processing (perhaps batching out to a separate method)
      IF transaction completed successfully THEN
            DO post transaction processing
      ENDIF
CATCH SQLException
  IF connection is still valid and usable THEN
      As exception is not due to a db outage, process as appropriate or return exception back to caller
  ELSE
    Close the connection
    Get a new good connection
            IF good connection is received THEN
                Retry back to while loop to begin transaction again
            ELSE
                Return exception back to caller
  ENDIF
ENDTRY
ENDWHILE
```

Listing 2: Application pseudo logic for connection retry

Integrate applications using FAN API

Oracle publishes APIs to consume FAN notifications in applications using these APIs applications can react to outages in an Oracle RAC database environment and manage application connections and incorporate more advanced behavior in applications automatically. For more information about integrating FAN APIs in applications refer to Oracle Real Application Clusters documentation.

Functionality Considerations

From a functional perspective, most applications work seamlessly when deployed on a RAC database. There should be no changes required to application functionality whether the application operates in a single instance or an Oracle RAC environment. Multi-node database architectures such as Oracle RAC nonetheless require certain considerations when architecting and developing applications.

Use Advanced Queuing (AQ) or Java Messaging Service (JMS) instead of DBMS_PIPE

If an application uses the DBMS_PIPE feature of the Oracle database, then it should ensure that both the producer and consumer of messages are connected to the same instance. This may not be a concern for single instance database architecture but it is important in a RAC database environment. Better still, it is recommended that applications use a messaging mechanism such as Advanced Queuing (AQ) or Java Messaging Service (JMS) instead of DBMS_PIPE. In Oracle Database 12c JMS Sharded Queues have been introduced and provide optimal database queuing implementation. A sharded queue is a single logical queue that is transparently partitioned into multiple physical queues and automatically maintained by the system. For more information about JMS Sharded Queues, please refer to Oracle Advanced Queuing User's Guide 12c and the Oracle white paper http://www.oracle.com/technetwork/database/jms-wp-2533807.pdf .

Use shared storage for accessing external files from within the database

If applications access external files from within the database using database stored procedures (which may run from any of the RAC database instances) then it is imperative that the files must be accessible from all RAC instances where the database service can run. For example, such external files may be hosted on shared storage accessible from all nodes of the RAC database cluster. The shared storage should provide the right level of concurrency control for file access as needed by the application. Oracle ASM Clustered File System (ACFS) can be an optimal choice for such a shared storage implementation in many situations.

Use shared storage for applications that use external tables

Just as with external files, for applications using external tables the external table files need to be accessible from all nodes of the RAC database cluster where the application can run from.

Switch to GV$ views for applications and scripts that use system views

Applications and database management scripts that use v$ database views need to be aware that the corresponding views in a RAC database environment are the GV$ global views that provide a complete view of the information from all instances. For example, see below the result of a query in a single-instance environment and corresponding query and results from a RAC database environment.

Single Instance	Real Application Clusters
`SQL> select instance_name, host_name from v$instance;` **INSTANCE_NAME HOST_NAME** rsdb1 rwsoda404c1n1	`SQL> select instance_name, host_name from gv$instance;` **INSTANCE_NAME HOST_NAME** rsdb1 rwsoda404c1n1 rsdb2 rwsoda404c1n2

Table 1 Queries in single-instance and Oracle RAC database environments

Applications that use DBMS_JOB package

It is recommended that applications use the DBMS_SCHEDULER package instead of the DBMS_JOB package because it ensures that the job is allocated on an instance that is available at the scheduled time, For applications that need to continue to use the DBMS_JOB package, ensure that the resources required by the job are available and accessible from all instances of the database (including the ones where the service may not run). You may use instance affinity feature to bind a job to a specific instance if a job needs to run from a specific instance.

RAC Database Application Developer’s Checklist

Table 2 summarizes the information provided in this white paper in a checklist format which users may find useful.

	Oracle RAC Database Application Developer’s Checklist
✓	Ensure application is designed with scalability in mind
✓	Design schema to minimize or eliminate hot spots
✓	Test application for scalability
✓	Use Distributed Transaction Processing (DTP) services for XA applications
✓	Use Automatic Segment Space Management (ASSM) for RAC databases
✓	Use database sequences optimally
✓	Use advanced queuing optimally
✓	Use Dynamic Database Services
✓	Use standard Oracle connection pooling
✓	Connection-time and run-time load balancing
✓	Build applications for transparent connection retry
✓	Integrate applications using FAN API
✓	Use Advanced Queuing (AQ) or Java Messaging Service (JMS) instead of DBMS_PIPFE
✓	Use shared storage for accessing external files from within the database
✓	Use shared storage for applications that use external tables feature
✓	Switch to GV$ views for applications and scripts that use system views
✓	Use the DBMS_SCHEDULER package instead of the DBMS_JOB package

Table 2 RAC Database Application Developer’s Checklist

Tools for tuning applications

The primary tools to quickly identify scalability problems and to tune applications for scalability are Oracle Enterprise Manager Cloud Control along with the Tuning Pack and Oracle Load Testing, a component of Oracle Application Testing Suite (ATS). For more information about these tools please refer to relevant Oracle documentation.

Conclusion

RAC provides a highly available and scalable database foundation for applications. While an application that scales well in a non-RAC database environment will generally scale in a RAC environment, application architecture needs to be RAC-aware to fully utilize and benefit from the high availability features of RAC and make applications resilient to failures. Furthermore, the multi-instance RAC database architecture requires awareness on part of application architects, system administrators, and application developers. To ensure transparent application operations when failures occur, continued access to resources outside of the database has to be available. This white paper has outlined some important considerations for architecting and developing applications that can help fully exploit the benefits of Oracle RAC database architecture.

References

1. Oracle Enterprise Manager 13c Cloud Control Data Sheet
http://www.oracle.com/technetwork/database/manageability/ds-tuning-pack-db12c-1964661.pdf

2. ORACLE REAL APPLICATION TESTING Data Sheet
http://www.oracle.com/technetwork/database/manageability/real-application-testing-ds-12c-1964674.pdf

3. MOS note: "RAC and Oracle Clusterware Best Practices and Starter Kit (Platform Independent) (Doc ID 810394.1)"

4. MOS note: "RAC: Frequently Asked Questions (Doc ID 220970.1)"

5. OTN: "Using Basic Database Functionality for Data Warehousing"
http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/10g/r2/prod/bidw/bdf/bdf_otn.htm#t5a

6. Oracle Database VLDB and Partitioning Guide 12c Release 1 (12.1) E41057–11
https://docs.oracle.com/database/121/VLDBG/E41057-11.pdf

7. ORACLE LOAD TESTING Data Sheet
http://www.oracle.com/technetwork/oem/pdf/511887.pdf

8. Oracle Database 12c Real Application Testing Overview White Paper
http://www.oracle.com/technetwork/database/manageability/real-application-testing-wp-12c-1896131.pdf

9. Using Oracle Database Cloud - Database as a Service
https://docs.oracle.com/cloud/latest/dbcs_dbaas/CSDBI/GUID-EB0BB703-B7D0-4C3C-B40B-B77D3F08127A.htm#CSDBI-GUID-EB0BB703-B7D0-4C3C-B40B-B77D3F08127A

10. XA and Oracle controlled Distributed Transactions White Paper
http://www.oracle.com/technetwork/products/clustering/overview/distributed-transactions-and-xa-163941.pdf

11. Application Continuity with Oracle Database 12c White Paper
http://www.oracle.com/technetwork/database/options/clustering/application-continuity-wp-12c-1966213.pdf

12. Transaction Guard with Oracle Database 12c Hiding Unplanned Outages White Paper
http://www.oracle.com/technetwork/database/database-cloud/private/transaction-guard-wp-12c-1966209.pdf

13. Database JDBC Developer's Guide, Oracle RAC Fast Application Notification
https://docs.oracle.com/database/121/JJDBC/apxracfan.htm#JJDBC28934

14. Oracle Database 12c: JMS Sharded Queues White Paper
http://www.oracle.com/technetwork/database/jms-wp-2533807.pdf

15. Oracle Enterprise Manager Cloud Control Documentation 12c
http://docs.oracle.com/cd/E24628_01/index.htm

16. Oracle Application Testing Suite OTN page

http://www.oracle.com/technetwork/oem/app-test/etest-101273.html

17. MOS note: Automatic Space Segment Management in RAC Environments (Doc ID 180608.1)

18. MOS note: Caching Oracle Sequences (Doc ID 62002.1)

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Hardware and Software, **Engineered to Work Together**

Application Development Best Practices for Oracle Real Application Clusters (RAC), A Developer's Checklist
July 2016
Authors: Hagen Herbst (RAC Pack) and Ravi Sharma (RAC Pack) [Adapted from an earlier version written by Pradeep Bhat]
Contributors: Sanjay Singh (RAC Pack), Markus Michalewicz (RAC Product Management)