ORACLE

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

A Developer's Checklist

August, 2024, Version 1.0

ORACLE

TABLE OF CONTENTS

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 feature of Oracle Database that allows multiple clustered instances to simultaneously access a common, shared physical database. Such clustering allows for high availability and scalability of the database platform as the architecture can withstand node failures and additional server nodes can be added to increase capacity. Oracle RAC uses Oracle Clusterware software 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 cluster 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 interconnected. If an instance has data in its cache that is required by another instance then it is appropriately shipped across the high speed interconnect. The database data files, control files, and redo log files reside on cluster aware, highly available shared storage and are accessible from all cluster nodes. Figure 1 illustrates basic RAC implementation architecture for a two node cluster configuration.

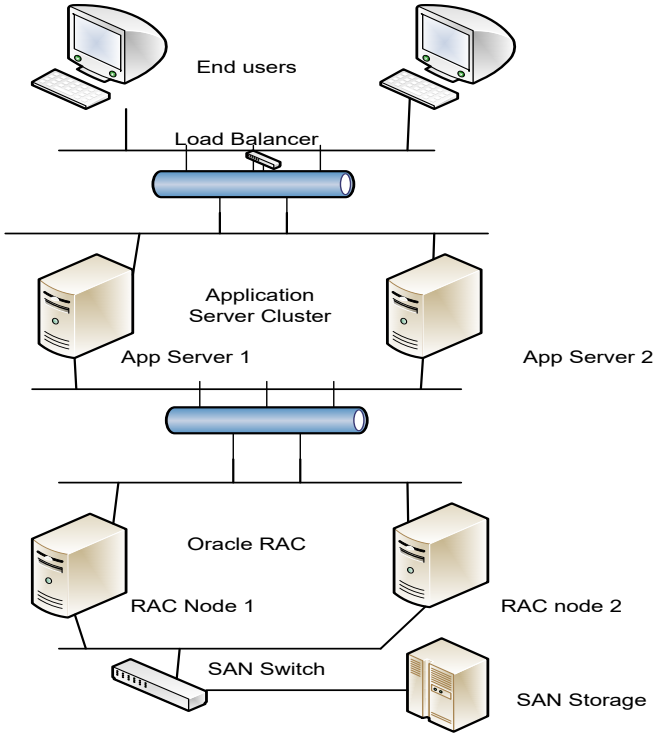

Figure 1 Oracle RAC architecture (a two node example)

As an application developer, it helps to understand the underlying database architecture and to use best practices to design and build an application can fully leverage the intended benefits provided by the database architecture. This may include writing applications that can minimize performance overhead, manage connections when failures occur, and in general being aware of the differences in certain nuances of clustering. This white paper outlines these aspects from an application designer and developer's perspective.

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.

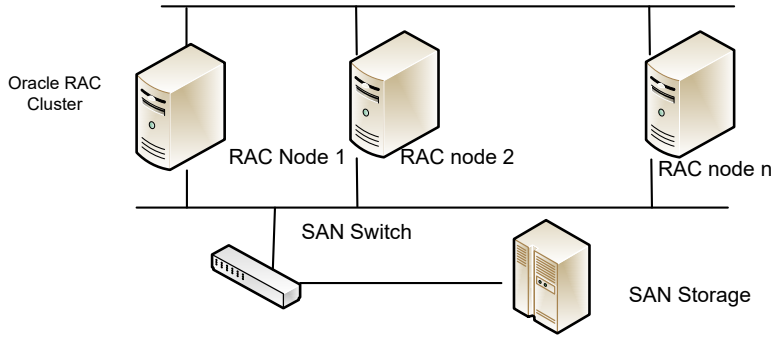

Figure 2 RAC Scalability

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 and performs well in both clustered and non-clustered environments. However, applications are not always written keeping scalability in mind. Many times, the focus of a developer is to produce functionality and not necessarily look at the performance and scalability aspects of it at the time of initial coding. In other words, performance and scalability are typically after thoughts for many application developers.

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. If the application did not scale in a single instance environment, chances are that you may want to resolve the old issues to fully exploit the new architecture and RAC scalability benefits.

Some of the common mistakes that developers make when writing applications that ultimately result in their applications lacking scalability are as follows:

- Not using bind variables and parsing code repetitively
- Using full table scans excessively and not leveraging indexing effectively
- Not using connections optimally, etc.

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

The following considerations apply for designing scalable applications.

Design schema to minimize or eliminate hot spots

To maximize scalability of applications with a 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 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 towards minimizing hot spots are as follows:

- ***Use appropriate row density*** - All database IO is in the units of database blocks. Larger database blocks may pack more rows per block. If these rows are concurrently accessed and locked from multiple instances, it can result in hot blocks.
- ***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 alleviate such hot spots.
- ***Avoiding hot spots*** - 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.
- ***Leverage HASH partitioned tables and indexes*** - Using hash partitioning may help distribute IO 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. Hash partitioned global indexes can be utilized even for non-partitioned tables.

  Refer to Oracle Support Article 220970.1 - RAC: Frequently Asked Questions
- ***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.
- ***Use full table scans optimally*** - Use full table scans (FTS) only when necessary and optimize index based access to data. Full table scans require active blocks to be made available to the querying instance(s) and can be expensive in a clustered database environment. For large tables with highly active application it may unnecessarily require extensive cache coordination and synchronization. Full table scans 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. 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 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. 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.

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 in a given tablespace when new instances are brought online. Refer to Oracle Support Article 180608.1 - Automatic Space Segment Management in RAC. Automatic Segment Space Management allows for more efficient space management inside database data blocks.

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. While this may result in some sequence numbers never getting used (for example, due to loss of cached entries in case of an instance crash) it may not be a concern to a majority of applications and can be a viable solution towards mitigating sequence contention. Caching may also result in use of un-ordered sequence numbers at different instances. In most cases, applications can be designed to be resilient to such scenarios in the interest of achieving higher scalability. Refer to Oracle Support Article 62002.1 - Caching Oracle Sequences.

Use advanced queuing optimally

Oracle RAC can be used to provide high availability and scalability to 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 an 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.

An important reason for using Oracle RAC database is application High Availability (HA). Availability of the database service offering from multiple instances can make applications resilient to planned and unplanned outages. In the event of an instance or node failure, the database service is expected to be available from the remaining instance(s). However, unless you use Transaction Guard, RAC does not protect against loss of in-flight transactions if an instance suddenly fails even though the service may continue to be available from the remaining (up) nodes. With awareness of the RAC database architecture applications can thus be architected to handle transient failures in a graceful and transparent manner.

For more information about Application Continuity refer to Oracle white paper Application Continuity with Oracle Database 12c . For more information about Transaction Guard refer to Oracle white paper Transaction Guard with Oracle Database 12c.

Application Continuity demonstration can be accessed at
http://www.oracle.com/technetwork/products/clustering/ac-overview-1967264.html

Use service feature of RAC database

Applications should connect to the database using service names. All advance features of Oracle RAC database are based on services.

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

b. Database connection string and / or URLs should always use service. Do not use SID 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. Oracle Client side connection pools are integrated with the FAN functionality 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

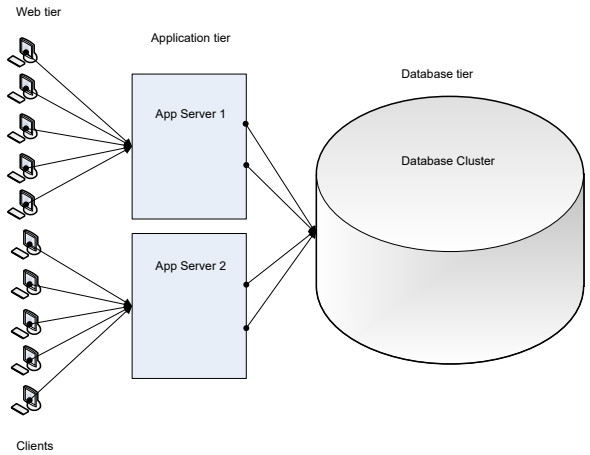

Figure 3 Illustration of basic connection pooling

As illustrated in figure 3, connection pools effectively consolidate large numbers of user requests, typically originating as connection from the client systems to web servers, into fewer shared connections between the application tier and database tier to serve the data requests. This results in 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 established a set of persistent, shared connections to the database that service transient application connection requests. Oracle connection pools provide the Runtime 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. Applications should be architected so as to use a connection only as long as it is required and to then release (close) the connection.

Build applications for transparent connection retry

Applications running with a RAC backend database should be architected to handle connection failure 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 received an exception. At the time of an exception, the application may be either be in the middle of a 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 1: Application pseudo logic for connection retry

Integrate applications using FAN API

Oracle published 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 for a RAC database. However, a multi-node architecture of a RAC DB introduces certain considerations that must be borne in mind when architecting and developing applications.

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

If an application uses the DBMS_PIPE feature of 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 in the single instance database architecture but is an important consideration 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_PIPLE. 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.

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 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 shared storage implementation in many situations.

Use shared storage for applications that use external tables feature

Just as with external files, for applications using the external tables feature the external table files need to be accessible from all server 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, please see 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$instance;`		`SQL> select instance_name, host_name from gv$instance;`	
INSTANCE_NAME	**HOST_NAME**	**INSTANCE_NAME**	**HOST_NAME**
rsdb1	rwsoda404c1n1	rsdb1	rwsoda404c1n1
		rsdb2	rwsoda404c1n2

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

Oracle RAC database implementation gives the user some flexibility in maintaining different configurations in different database instances.

Applications that use DBMS_JOB package

It is recommended that applications use the DBMS_SCHEDULER package instead of the DBMS_JOB package. 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). 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 1 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 service feature of RAC database
✓	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_PIPLE
✓	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 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 the appendix section and relevant Oracle documentation.

CONCLUSION

Oracle Real Application Clusters (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 the high availability features of RAC and make applications resilient to failures. Further, a multi-instance RAC database architecture requires awareness on part of application architects and developers to ensure continued access to resources outside of the database when failures occur to ensure transparent application operations. This white paper outlines some important considerations for architecting and developing applications that can help fully exploit the benefits of Oracle RAC database architecture.

REFERENCES

1. Oracle Tuning Pack for Oracle Databases
   http://www.oracle.com/technetwork/database/manageability/ds-tuning-pack-db12c-1964661.pdf
2. Oracle Load Testing
   http://www.oracle.com/technetwork/oem/pdf/511887.pdf
3. Oracle Real Application Clusters documentation

ORACLE

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle