Data Warehousing on Oracle RAC Best Practices

An Oracle White Paper
October 2008

ORACLE

Data Warehousing on Oracle RAC Best Practices

Data Warehousing on Oracle RAC Best Practices

EXECUTIVE OVERVIEW

Modern Data Warehouse systems have grown and evolved to the point where they now demand an underlying database infrastructure that can adapt and support their great diversity, complexity and ever-changing needs. Oracle Real Application Clusters (RAC) is the database infrastructure that can address these needs. In order to make the most of what Oracle RAC has to offer to Data Warehousing systems, it is best to take a structured balanced approach. This paper delves into the technical data warehouse features that are relevant to Oracle RAC and provides a high-level review of the best practices for balancing technical, business and operational needs in the implementation of Data Warehouse systems on Oracle RAC.

INTRODUCTION

Data Warehousing on Oracle database systems is commonplace today but is still evolving in response to increasingly complex business demands and the improving technical capabilities of software and hardware. In parallel to these ongoing developments has been the growth and acceptance of Oracle RAC as a viable, highly-available, scalable architecture that can support diverse needs.

A marriage of the two, Data Warehousing and Oracle RAC, has become popular as Data Warehousing systems and their particular demands have increased in complexity. These requirements have found a home in the flexibility and power of Oracle RAC.

WAREHOUSING TODAY

Data Warehousing has evolved from the early days of a few monthly-updated star schemas satisfying one or two strictly predefined business needs. It now provides functional access to data and information in a variety of structures, inside and outside the database, via numerous access paths, with updates occurring on regular and irregular schedules. It provides 24*7 availability and real-time access to ever-growing business demands made by the users and business community in general.

Data Warehouse systems on Oracle RAC are proving to be a popular strategy for meeting these demands. According to a paper recently published by the Winter

Corporation, "Oracle's Top Ten Features for Large-Scale Data Warehousing"[1], the top three features listed are:

- Oracle Partitioning
- Oracle Real Application Clusters (RAC), and
- Parallel Operations

These three features are key to the successful implementation of a modern data warehouse. Implementing these features using best practices to support data warehouse system demands is of high importance. Therefore it is essential to understand the relevant aspects of these features in the context of a Data Warehouse in a Oracle RAC environment.

BEST PRACTICES FOR DATA WAREHOUSING ON ORACLE RAC

The key considerations when designing a Data Warehouse with Oracle RAC include

- Oracle Partitioning
- Parallel Execution
- Workload Management

The relevance of each of these features depends on the specific implementation. This section will present best practices and focus on the technical aspects of Data Warehouse features that are relevant to Oracle RAC. It will be followed by a discussion about considerations that should be taken into account when implementing.

Oracle Partitioning

Basics of Oracle Partitioning

Oracle partitioning allows a table, index or index-organized table to be subdivided into smaller pieces. Each piece of a database object is called a partition. Each partition has its own name, and may optionally have its own storage characteristics, such as having table compression enabled or being stored in different tablespaces. From the perspective of a database administrator, a partitioned object has multiple pieces that can be managed either collectively or individually. This gives the administrator considerable flexibility in managing these partitioned objects. However, from the perspective of the application, a partitioned table is identical to a non-partitioned table; no modifications are necessary when accessing a partitioned table using SQL commands.

---

[1] "Oracle's Top Ten Database Features for Large-Scale Data Warehousing" – Winter Corporation, Roger Dorin, March 2007.

Tables are partitioned using a 'partitioning key', a set of columns, which determine in which partition a given row will reside. Oracle provides the following techniques for partitioning:

- Range Partitioning
- Hash Partitioning
- List Partitioning
- Composite Partitioning
  - Composite Range-Range Partitioning
  - Composite Range-Hash Partitioning
  - Composite Range-List Partitioning
  - Composite List-Range Partitioning
  - Composite List-Hash Partitioning
  - Composite List-List Partitioning
- Interval partitioning (11g)
- REF partitioning (11g)
- Virtual column based partitioning (11g)

Oracle Partitioning Benefits

Partitioning for Manageability

With partitioning, maintenance operations can be focused on particular portions of tables. For maintenance operations across an entire database object, it is possible to perform these operations on a per-partition basis, thus dividing the maintenance process into more manageable chunks. A typical usage of partitioning for manageability is to support a 'rolling window' load process in a data warehouse.

Partitioning for Availability

Partitioned database objects provide partition independence. This characteristic of partition independence can be an important part of a high-availability strategy. For example, if one partition of a partitioned table is unavailable, all of the other partitions of the table remain online and available; the application can continue to execute queries and transactions against this partitioned table, and these database operations will run successfully if they do not need to access the unavailable partition. The database administrator can specify that each partition be stored in a separate tablespace; this would allow the administrator to do backup and recovery operations on each individual partition, independent of the other partitions in the table. Moreover, partitioning can reduce scheduled downtime.

Partitioning for Performance

By limiting the amount of data to be examined or operated on, and by enabling parallel execution, the Oracle Partitioning option provides a number of performance benefits. These features include:

Partitioning Pruning

Partitioning pruning is the simplest and also the most substantial means to improve performance from using partitioning. Given a query over partitioned table, the optimizer will access the minimal set of partitions that must be accessed to resolve the query. Partition pruning can often improve query performance by several orders of magnitude. Partition pruning works with all of Oracle's other performance features. Oracle will utilize partition pruning in conjunction with any indexing technique, join technique, or parallel access method.

Partition-wise Joins

Partitioning can also improve the performance of multi-table joins, by using a technique known as partition-wise join. Partition-wise joins can be full or partial. Oracle decides which type of join to use:

Full Partition-wise Join

A full partition-wise join divides a large join into smaller joins between a pair of partitions from the two joined tables. To use this feature, both objects (table or index or partition) must be equipartitioned on their join keys.

Example:

Consider the table sales partitioned by date using sale_date column and sub-partitioned by hash using customer_id column and the table customers partitioned by hash using customer_id column. Note that the number of hashed partitions is the same for customers and each ranged partition of sales.

Consider the following query:

*Select * from sales, customers*

where sale_date between to_date(1-jan-2008) and to_date(31-jan-2008)

and sales.customer_id = customers.customer_id

The sales partition and the customers table are equipartitioned. Therefore, a full partition-wise join will be used.

Partial Partition-wise Join

Unlike full partition-wise joins, partial partition-wise joins require to partition only one table on the join key, not both tables. The partitioned table is referred to as the reference table. The other table may or may not be partitioned. Partial partition-wise joins are more common than full partition-wise joins. To execute a partial partition-wise join, Oracle dynamically repartitions the other table based on the partitioning of the reference table. Once the other table is repartitioned, the execution is similar to a full partition-wise join. The performance advantage that partial partition-wise joins have over joins in non-partitioned tables is that the reference table is not moved during the join operation.

Going into the details of all partitioning strategies is beyond the scope of this paper. However, keep in mind that it is management and performance that will drive the decision of which strategy will be implemented. For example, if the goal of partitioning is to ease the loading and maintenance, then a typical strategy will be range partitioning by time with local indexes, so that partition exchange and dropping can be implemented. If the goal is performance, then the partition strategy will be decided based on the most critical queries, and will take into account the partition-wise joins (full and partial) and the use of global indexes, as will be seen in later sections.

Oracle's Parallel Execution Architecture

Oracle's Parallel Execution (PX) Architecture comprises of a query coordinator (QC) process and a set of processes (PQ slaves) which support intra-operation parallelism. Oracle supports parallel execution of all relational operations (e.g. various scans, joins, order-by, aggregation, set operations), DML's (insert, update, delete, merge and multi-table inserts), DDL's (e.g. create table, index, materialized views), data reorganization (e.g. partition maintenance operations like move, split, coalesce), Load/Unload via external tables and SQL-extensions for Analytics and Data-Mining.

How Parallel Execution Works

Parallel execution divides the task of executing a SQL statement into multiple small units, each of which is executed by a separate process. The user shadow process that wants to execute a query in parallel takes on the role as parallel execution coordinator or query coordinator(QC). The query coordinator does the following:

- Parses the query and determines the degree of parallelism

- Allocates one or two set of parallel execution servers - threads or processes (PQ slaves).
- Controls the query and sends instructions to the PQ slaves
- Determines which tables or indexes need to be scanned by the PQ slaves
- Produces the final output to the user

Parallel statements are executed on slave processes that can run within a node (intra-node parallelism) or across multiple nodes (inter-node parallelism) across the grid. Intra-node communication uses shared-memory and inter-node communication uses IPC protocol over high-speed interconnects. After the statement has been processed completely, the parallel execution servers return to the pool

Dynamic Determination of Work Units

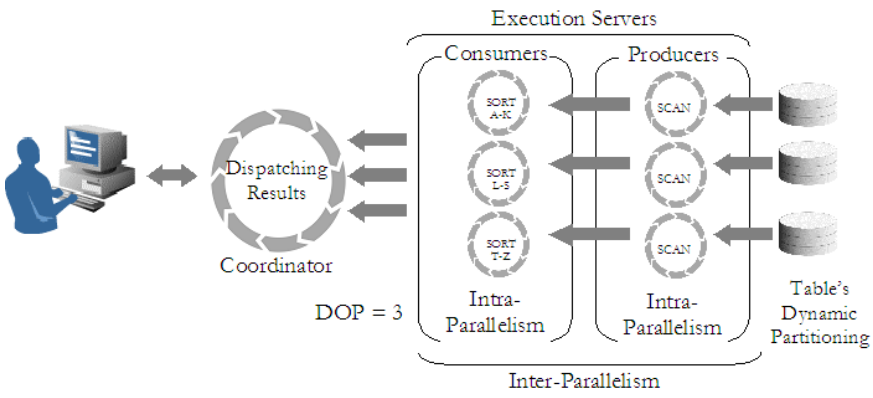
The basic unit of work in parallelism is called a granule. Oracle divides the objects used by parallel operations (table scan, table update, index creation,…) into granules. Oracle uses block range granules (range of physical blocks from a table or index) or partition granules. Parallel execution servers execute the operation one granule at a time. The number of granules and their size correlates with the degree of parallelism (DOP) and the object size. There is no way you can enforce a specific granule strategy as Oracle makes this decision internally. Depending on the parallel operation, Oracle either chooses the granule based on block ranges or a complete partition as the optimal granule strategy.

Parallel Operations and Degree of Parallelism (DOP)

After the optimizer determines the execution plan of a statement, the parallel execution coordinator determines the parallelization method for each operation in the execution plan. The coordinator must decide whether an operation can be performed in parallel and, if so, how many parallel execution servers to enlist. The number of parallel execution servers used for an operation is the degree of parallelism (DOP) and is determined at query execution time, based on the system's current workload and other criteria, such as the business priority of a user's query.

The following diagram illustrates the way parallel operations and DOP work:

```
SELECT cust_last_name,cust_first_name
FROM   customers  ORDER BY cust_last_name;
```



The execution plan implements a full scan of the CUSTOMERS table followed by a sort of the retrieved rows based on the value of the CUST_LAST_NAME column. For the sake of this example, assume this column is not indexed. Also assume that the degree of parallelism for the query is set to three, which means that three parallel execution servers can be active for any given operation. Each of the two operations (scan and sort) performed concurrently is given its own set of parallel execution servers. Therefore, both operations have parallelism. Parallelization of an individual operation where the same operation is performed on smaller sets of rows by parallel execution servers is called intra-operation parallelism. When two operations run concurrently on different sets of parallel execution servers with data flowing from one operation into the other, we achieve what is called inter-operation parallelism. Due to the producer/consumer nature of the Oracle server's operations, only two operations in a given tree need to be performed simultaneously to minimize execution time.

How Parallel Execution Servers Communicate

To execute a query in parallel, Oracle generally creates a producer queue server and a consumer server. The producer queue server retrieves rows from tables and the consumer server performs operations such as join, sort, DML, and DDL on these rows.

Each server in the producer execution process set has a connection to each server in the consumer set. This means that the number of virtual connections between parallel execution servers increases as the square of the DOP.

Each communication channel has at least one, and sometimes up to four memory buffers. Multiple memory buffers facilitate asynchronous communication among the parallel execution servers.

A single-instance environment uses at most three buffers for each communication channel. An Oracle Real Application Clusters environment uses at most four buffers for each channel. When a connection is between two processes on the same instance, the servers communicate by passing the buffers back and forth. When the connection is between processes in different instances, the messages are

sent using external high-speed network protocols (interconnect). Each parallel execution server actually has an additional connection to the parallel execution coordinator.

Parallel Query and Distribution Combinations

The knowledge about parallel everything operations is built into the optimizer; when the execution plan of a statement is determined, the decision about the DOP and the most optimal parallelization method for each operation is made. For example, the parallelization method might be to parallelize a full table scan by block range or parallelize an index range scan by partition. Furthermore, the redistribution requirements of each operation are determined. An operation's redistribution requirement is the way in which the rows operated on by the operation must be divided or redistributed among the parallel execution servers.

In the default case, parallel joins between two non-partitioned tables require both input tables to be redistributed on the join key into disjoint subsets of rows. These disjoint subsets of rows are then joined pair-wise by a single parallel execution server. This redistribution operation involves exchanging rows between parallel execution servers.

The possible methods used to distribute rows from producer query servers to consumer query servers are:

- `PARTITION`: Maps rows to query servers based on the partitioning of a table or index.
- `HASH`: Maps rows to query servers using a `HASH` function on the join key.
- `RANGE`: Maps rows to query servers using ranges of the sort key.
- `ROUND-ROBIN`: Randomly maps rows to query servers.
- `BROADCAST`: Broadcasts the rows of the entire table to each query server.
- `QC(ORDER)`: The execution coordinator consumes the input in order.
- `QC(RANDOM)`: The execution coordinator consumes the input randomly.

The execution plan of a parallel statement stores, in the `DISTRIBUTION` column of `PLAN_TABLE`, the method used to distribute rows from producer query servers to consumer query servers.

After determining the redistribution requirement for each operation in the execution plan, the optimizer determines the order in which the operations must be performed. With this information, the optimizer determines the final parallel plan and the data flow between parallel operations of the statement.

In Oracle Database 10g Release 2 the parallel execution model for queries changed from a slave SQL model to a parallel single cursor (PSC) model. Instead of having the query coordinator (QC) build a SQL statement for each DFO/slave on any instance, and each slave having to parse and execute his own cursor, Oracle now builds and compiles just one cursor per instance that contains all the information required for parallel execution. Each slave on the same instance is therefore able to share the same cursor. This model of a global parallel plan both yields performance gains and memory reduction.

Parallelism, Oracle Partitioning and Oracle RAC

By default, all available Oracle RAC instances are considered for parallel execution.

Parallel execution does not allocate slaves randomly across the available instances, but rather will start by allocating on the least loaded instance. The goal is to both minimize inter-node traffic and at the same time try to minimize any imbalance of work across the instances.

Parallel Query and Oracle Partitioning

As mentioned in the previous section, the redistribution operation involves exchanging rows between parallel execution servers. This is a CPU-intensive operation that can lead to excessive interconnect traffic in Oracle Real Application Clusters environments.

Use Partition-Wise Joins

When a full partition-wise join is executed in parallel, the granule of parallelism is a partition. As a result, the degree of parallelism is limited to the number of partitions. For example, you require at least 16 partitions to set the degree of parallelism of the query to 16. Various partitioning methods can then be used to equipartition both tables on the key column. Partition-wise joins reduce query response time by minimizing the amount of data exchanged among parallel execution servers when joins execute in parallel. This significantly reduces response time and improves the use of both CPU and memory resources. In Oracle Real Application Clusters environments, partition-wise joins also avoid or at least limit the data traffic over the interconnect, which is the key to achieving good scalability for massive join operations.

Most join operations in Oracle Real Application Clusters could experience high interconnect latencies without parallel execution of partition-wise joins. You should use this feature for large DSS configurations that use Oracle Real Application Clusters.

Use Parallel Instance Groups (in Oracle Database 10g)

In the vast majority of cases Oracle's grid architecture and parallel query optimization will deliver good performance. In a Oracle Real Application Clusters environment the optimizer takes into account the cost of sending a message across

the interconnect versus sending a message locally. It also takes into account the number of active instances. Though the optimizer will try to run the query on a single instance… Therefore, if the query is expected to return a very large number of rows from each node, it might be beneficial to limit the inter-node parallelism as described above and thus limit the amount of data delivered over the interconnect. Also if each node returns only a small number of rows then might be better to limit the number of nodes involved due to the overhead of remote process startup time.

A way to minimize inter-node traffic is to limit the parallel execution within one instance or a group of instances. You can establish group membership by setting the INSTANCE_GROUPS and the PARALLEL_INSTANCE_GROUP init.ora parameters. For example, consider the following assignments:

On instance A: INSTANCE_GROUPS=AMER

On instance B: INSTANCE_GROUPS=AMER, EMEA, APAC, JPN

Then a user can activate the nodes in the AMER group to spawn query server processes by using the following command:

ALTER SESSION SET PARALLEL_INSTANCE_GROUP = AMER;

In response, the parallel executions can be split across instances, i.e. can run on both instance A and instance B. On the other hand, by setting PARALLEL_INSTANCE_GROUP = APAC, only instance B can be used for parallel executions.

Be aware though that the init.ora parameter INSTANCE_GROUPS cannot be changed dynamically.

Note: In Oracle Database 11g parallel execution is now aware of the service definition and automatically takes on the appropriate PARALLEL_INSTANCE_GROUP setting, thus making the explicit setting of PARALLEL_INSTANCE_GROUP unnecessary. The use of services will be explained in the workload management section.

Parallel_adaptive_multi_user Parameter

PARALLEL_ADAPTIVE_MULTI_USER, when set to TRUE, enables an adaptive algorithm designed to improve performance in multi-user environments that use parallel execution. The algorithm automatically reduces the requested degree of parallelism based on the system load at query startup time. The effective degree of parallelism is based on the default degree of parallelism, or the degree of the table or hints, divided by a reduction factor.

Example: On a 17 CPU machine the default degree of parallelism could be set to 32. If one user issues a parallel query, that user gets a degree of 32, effectively using

all of the CPUs and memory in the system. When a second user enters the system, that user gets a degree of 16. As the number of users on the system increases, this algorithm will continue to reduce the degree until the users are running using degree 1, when there are 32 users on the system.

The default value of parallel_adaptive_multi_user is true.

Interconnect Considerations

The largest demand for interconnect traffic in data warehouse systems comes from inter-process communication (IPC). Parallel Servers communicate with each other using the Interconnect. The amount of interconnect traffic depends on the operation and the number of nodes participating in the operation. Join operations tend to utilize the interconnect traffic more than simple aggregations because of possible communication between Parallel Servers. The amount of interconnect traffic can vary significantly depending on the distribution method.

Which distribution method is used can be found in the DISTRIBUTION column of the query plan. Cases where one side of the join is broadcasted or both sides are hash-distributed result in the largest interconnect traffic. Partial Partition-wise Joins in which only one side of the join is redistributed result in less interconnect traffic, while Full Partition-wise Joins in which no side needs to be redistributed result in the least interconnect traffic.

Therefore, if inter-node parallel query is used, the interconnect network must be sized appropriately. Unless the application is very well structured with a mostly pre-defined set of queries that can take advantage of partition-wise joins, be sure to take into account that some of the data redistribution is going to happen over the interconnect for inter-node parallel queries.

The amount of interconnect traffic also depends on how many nodes participate in a join operation. The more nodes participate a join operation the more data needs to be distributed to remote nodes. For instance in a 4-node Oracle RAC cluster with 4 CPU each to maximize load performance with external tables one needs to set the DOP to 32 on both the external and internal tables. This will result in 8 Parallel Servers performing read operations from the external table on each node as well as 8 Parallel Servers performing table creation statements on each node.

The important considerations:

- Use multiple or dual-ported NIC's for redundancy and increase bandwidth with NIC bonding
- Use (multiple) 10 GigE or Infiniband interconnect (if it is available on your platform) if you plan to use heavily inter-node parallel query.
- Consider using CLUSTER_INTERCONNECTS initialization parameter when a single cluster interconnect cannot meet your bandwidth requirements and there is more than one interconnect. It enables you to specify multiple IP addresses, separated by colons.

Oracle RAC network traffic is distributed between the specified IP addresses.

CLUSTER_INTERCONNECTS = ip*1*:ip*2*:...:ip*n*

Oracle uses all of the interconnects that are specified. This provides load balancing as long as all of the listed interconnects remain operational. **Note** that if one of the interconnects in the CLUSTER_INTERCONNECTS parameter become unavailable, Oracle returns an error and the instance may fail.

Note: If you use CLUSTER_INTERONNECTS parameter for load balancing and performance, it will be at the price of High Availability

Consider increasing the PARALLEL_EXECUTION_MESSAGE_SIZE parameter. This parameter specifies the size of the buffer used for parallel execution messages. The default value is typically 2K. Increasing this value (to 4K or 8K for example) can improve performance. Consider increasing this value if you have adequate free memory in the shared pool.

Monitor the interconnect traffic using AWR reports. Monitor the Global cache traffic (blocks received/served, messages received/sent…) and the IPQ traffic (PX local and remote messages). Summing up the GES+GCS messages, Parallel Query messages and the Cache Fusion blocks, will give a good estimation of the bandwidth needed.

The following are 2 examples of AWR report highlighting the 2 primary interconnect traffic types:

Global Cache traffic (typical for OLTP operations)

Global Cache Load Profile	Per Sec	Per Trans
Global Cache blocks received:	2.70	2.23
Global Cache blocks served:	2.84	2.36
GCS/GES messages received:	164.07	136.03
GCS/GES messages sent:	136.96	113.56
DBWR Fusion writes:	0.22	0.18

IPQ traffic (for parallel operations)

Statistic	Total	per Sec	per Trans
PX local messages recv'd	104	0.1	0.1
PX local messages sent	104	0.1	0.1
PX remote messages recv'd	200271	200.2	151.1

```
PX remote messages      213267    213.2    156.1
sent
```

Loading Data

Parallel DML

Data Manipulation Language (DML) operations such as INSERT, UPDATE, and DELETE can be parallelized by Oracle. Parallel execution can speed up large DML operations and is particularly advantageous in data warehousing environments where it's necessary to maintain large summary or historical tables. Note that direct read is not used for parallel update/delete since the cache layer must update the block.

In the example below, 12 parallel execution servers were spawned on 3 instances for this UPDATE statement:

SQL> alter session enable parallel dml;

Session altered.

SQL> update /*+ parallel(ware,12) */ ware set w_ytd=w_ytd;

Partitioning enables the unlimited parallel execution of UPDATE, DELETE, and MERGE statements. Oracle will parallelize SELECT statements and INSERT statements when accessing both partitioned and non-partitioned database objects. UPDATE, DELETE and MERGE statements can be parallelized for both partitioned and non-partitioned database objects when no bitmap indexes are present; in order to parallelize those operations on objects having bitmap indexes, the target table must be partitioned. Parallel execution of these SQL operations can vastly improve the performance, particularly for UPDATE, DELETE, or MERGE operations involving large volumes data.

Parallel DML versus Manual Parallelism

DML operations can be parallelized manually by issuing multiple DML statements simultaneously against different sets of data. For example:

- Issue multiple INSERT statements to multiple instances of an Oracle Real Application Clusters to make use of free space from multiple free list blocks, or
- Issue multiple UPDATE and DELETE statements with different key value ranges or rowid ranges.

However, manual parallelism has the following disadvantages:

- It is difficult to use. You have to open multiple sessions (possibly on different instances) and issue multiple statements.
- There is a lack of transactional properties. The DML statements are issued at different times; and, as a result, the changes are done with

inconsistent snapshots of the database. To get atomicity, the commit or rollback of the various statements must be coordinated manually (maybe across instances).

The work division is complex. You may have to query the table in order to find out the rowid or key value ranges to correctly divide the work.

The calculation is complex. The calculation of the degree of parallelism can be complex.

There is a lack of affinity and resource information. You need to know affinity information to issue the right DML statement at the right instance when running an Oracle Real Application Clusters. You also have to find out about current resource usage to balance workload across instances.

Parallel DML removes these disadvantages by performing inserts, updates, and deletes in parallel automatically.

Index Block Contention

In a Data Warehouse, where the loading or batch processing of data is a dominant business function, there may be performance issues affecting response times because of the high volume of inserts into indexes. Depending on the access frequency and the number of processes concurrently inserting or updating data, indexes can become hot spots and contention can be exacerbated by:

Ordered, monotonically increasing key values in the index (right-growing trees),

Frequent leaf block splits,

Low tree depth - all leaf block accesses go through the root block.

A leaf or branch block split can become a serialization point if key values are being concurrently inserted in these particular leaf blocks or branches of the tree.

In a Oracle Real Application Clusters environment, the transaction splitting a leaf block and the transactions waiting for the split to complete are affected if data required for the operation are not locally cached.

The transactions encountering a split usually:

Re-scans the tree from the root if a leaf block is marked as having a split in progress,

Waits to read the branch block, and

Waits on the TX enqueue representing the service ITL for the split.

If during the split, new blocks or extents have to be allocated to the segment, the duration of the operation can be significantly increased.

As a general recommendation, to alleviate the performance impact of globally hot index blocks and leaf block splits, a more uniform, less skewed distribution of the concurrency in the index tree should be the primary objective. This can be achieved by:

Global index hash partitioning

In Oracle Database 10g Release 2, a global index can be hash partitioned. This will reduce contention on index blocks when inserting frequently into segments with indexes that are possibly right growing. In a data warehouse environment, you might also need a global index when your query cannot benefit from the way the table is partitioned. For example, some queries need to scan the entire index to retrieve information. If the index is LOCAL, Oracle will run and combine data from all the partitions of the index resulting in bad performance. If the index had been created GLOBAL, Oracle would performed less reads of the single (larger) index. This global index will perform better and will allow parallelism if it is hash-partitioned on the key index.

In the global index partition scheme, the index is harder to maintain since the index may span partitions in the base table. For example, when a table partition is dropped as part of a reorganization, the entire global index will become invalid and must be rebuilt using:

alter table <tablename> drop <partition name> update global indexes;

If the index is hash partitioned and the parallel clause is specified, Oracle will parallelize the index update.

Increasing the sequence cache, if the key value is derived from a sequence

Use natural keys as opposed to surrogate keys

Use reverse key indexes

A reverse key index works by reversing the order of the bytes in the key value. Reversing the keys of the index allows insertions to be distributed across all the leaf keys in the index. These indexes are designed to eliminate index hot spots on insert applications and are excellent for insert performance. However one of the major limitations of reverse key indexes is that they cannot be used in an index range scan, since reversing the index key value randomly distributes the blocks across the index leaf nodes. A reverse-key index can only use the fetch-by-key or full-index(table)scans methods of access.

Use a different block size

Larger oracle block sizes typically give fewer index levels and hence improved index access times to data. A single I/O will fetch many related rows and subsequent requests for the next rows will already be in the data buffer. This is one of the major benefits of a larger block size. Another benefit is that it will decrease the number of splits.

By using a smaller block size, the reduction in the number of rows in a block helps to reduce buffer busy contentions due to high concurrency. Generally, this would more likely be a problem for update or merge processes rather than for those that are insert-intensive. Thus, for a DW system in which inserts dominate the DML activities, smaller block sizes would not offer a benefit.

There are also some cases in which it may also prove beneficial to use a single instance for the load process. This approach will be discussed in the next section.

Automatic Workload Management

Automatic workload management facilitates and controls the distribution of work across the nodes in the cluster in order to achieve optimal performance for users and applications

Services

Oracle Database 10g introduced a powerful workload management facility, called services. They are the basis for workload management in Oracle RAC. Services divide the entire workload executing in the Oracle Database into mutually disjointed classes. Each service represents a workload/application with common attributes, service level thresholds, priorities, performance measures for real transactions, and alerts and actions when performance goals are violated.

When you create a service, you define which instances typically support that service. These are known as the **preferred** instances for that service. You can also define other instances to support a service if the service's preferred instance fails. These are known as **available** instances for a service.

When you specify a preferred instance for a service, the service runs on that instance during standard operation. Oracle Clusterware attempts to ensure that the service always runs on all the preferred instances that have been configured for a service. If the instance fails, the service is randomly relocated to one of the available instances. You can also manually relocate the service to an available instance. If you do not specify preferred or available instances when you create a service, then by default every instance in the Oracle RAC database is a preferred instance for that service.

A service can be defined using srvctl or Oracle Enterprise Manager-Grid Control.

Example:

srvctl add service –d rac_db –s dw_load –r inst1,inst2 –a inst3,inst4
srvctl add service –d rac_db –s dw_query –r inst3,inst4 –a inst1,inst2

In this example two services are created, DW_LOAD and DW_QUERY. The DW_LOAD service uses instances inst1 and inst2 as its preferred instances with inst3 and inst4 as its available instances. Service DW_QUERY uses the reverse of these assignments.

Middle tier applications and client-server applications use a service by specifying the service as part of the connection in the TNS connect data. This may be in the TNSnames file for thick Net drivers, in the URL specification for thin drivers, or may be maintained in the Oracle Internet Directory.

Example of tnsnames.ora:

```
LOAD_USERS =
 (DESCRIPTION =
  (ADDRESS_LIST = Service
   (ADDRESS = (PROTOCOL = TCP)(HOST = docrac1-vip)(PORT = 1521))
   (ADDRESS = (PROTOCOL = TCP)(HOST = docrac2-vip)(PORT = 1521))
  (LOAD_BALANCE = yes)
  )
 (CONNECT_DATA = (SERVICE_NAME = DW_LOAD))
  )
```

Server side work, such as the Job Scheduler, Parallel Query, and Oracle Streams Advanced Queuing set the service name as part of the workload definition.

For the Job Scheduler, the service that the job class uses is defined when the job class is created. During execution, jobs are assigned to job classes and job classes run within services. Using services with job classes ensures that the job scheduler work is identified for workload management and performance tuning. For high availability, DBMS_JOBS previously used the instance name to define where services should execute. This approach resulted in jobs not running when the instance was unavailable. With the new job scheduler, setting the service in the job class ensures that the job executes when the service is running anywhere in the cluster or grid.

Services and Parallel Query

For parallel query and parallel DML, the query coordinator connects to a service just like any other client. However, in Oracle RAC 10g, the parallel execution slaves are allocated on instances without regard for services thus greatly reducing the benefits of services. The workaround is to use parallel_instance_groups. With Oracle RAC 11g, parallel execution has been integrated with services and the slaves

will automatically be restricted to instances where the service is running making the use of the parallel_instance_group parameter unnecessary.

Limiting the Number of Resources for a User

The amount of parallelism available to a given user can be limited by establishing a resource consumer group for the user. Do this to limit the number of sessions, concurrent logons, and the number of parallel processes that any one user or group of users can have. Each query server process working on a parallel execution statement is logged on with a session ID. Each process counts against the user's limit of concurrent sessions. For example, to limit a user to 10 parallel execution processes, set the user's limit to 11. One process is for the parallel coordinator and the other 10 consist of two sets of query server servers. This would allow one session for the parallel coordinator and 10 sessions for the parallel execution processes.

Services Implementation

Use services to manage and partition the workload:

- Services provide a single system image for managing workload
- Service span one or more instances of the database. An instance can support multiple services. The number of instances offering the service is managed by the DBA and independent of the application
- Define one service for applications that have the same level agreement and similar requirements.
- Based on the SLA of each service, decide how many instances will offer a given service. For example, define a service that will run load processes, a second service for ad-hoc queries and a third service for batch operations. Based on your system and available resources you will decide which instances will offer each of the services,
- Check if there are services that should run on one instance for performance reason (contention on indexes during the load process for example).

Use services together with Oracle Database Resource Manager to provide effective control of system resources.

Use cluster managed services for load balancing.

Additional Considerations

Bloom Filters

In Oracle Database 11g, the performance of partition pruning has been enhanced by using bloom filtering instead of subquery pruning. While subquery pruning was

activated on a cost-based decision and consumed internal (recursive) resources, pruning based on bloom filtering is activated all the time without consuming additional resources.

A Bloom Filter is a probabilistic algorithm that quickly tests membership in a large set using multiple hash functions into a single array of bits. A join filter can be used when an execution plan fetches all of the rows of one table before fetching any rows from another table that joins to it:

The join filter builds an array of bits, and turns on bits on for each row of the first table that matches the search conditions

When scanning the second table, rows that could not possibly join with the first table are rejected.

As already mentioned, in a parallel hash join with inter-node parallelism the interconnect can become a bottleneck. Pre filtering using bloom filter can reduce communication overhead.

example of an explain plan with bloom filter:

select count(*)
from empp e, deptp d
where e.deptno = d.deptno

```
SELECT STATEMENT           |            |           |      |
     SORT AGGREGATE            |            |           |      |
      PX COORDINATOR           |            |           |      |
       PX SEND QC (RANDOM)     | :TQ10002   |        02 | P->S | QC
        SORT AGGREGATE         |            |        02 | PCWP |
         HASH JOIN             |            |        02 | PCWP |
          PX JOIN FILTER CREATE| :BF0000    |        02 | PCWP |
           PX RECEIVE          |            |        02 | PCWP |
            PX SEND HASH       | :TQ10000   |        00 | P->P | HASH
             PX BLOCK ITERATOR |            |        00 | PCWC |
              TABLE ACCESS FULL| EMPP       |        00 | PCWP |
          PX RECEIVE           |            |        02 | PCWP |
           PX SEND HASH        | :TQ10001   |        01 | P->P | HASH
            PX JOIN FILTER USE | :BF0000    |        01 | PCWP |
             PX BLOCK ITERATOR |            |        01 | PCWC |
              TABLE ACCESS FULL| DEPTP      |        01 | PCWP |
```

example of an explain plan with bloom filter pruning:

```
| Id  | Operation                      | Name     | Pstart| Pstop |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT               |          |       |       |
|   1 |  SORT AGGREGATE                |          |       |       |
|*  2 |   HASH JOIN                    |          |       |       |
|   3 |    PART JOIN FILTER CREATE     | :BF0000  |       |       |
|   4 |     PARTITION HASH ALL         |          |   1   |     2 |
|   5 |      TABLE ACCESS FULL         | TB1      |   1   |   2   |
|   6 |    PARTITION HASH JOIN-FILTER  |          |:BF0000|:BF0000|
|   7 |     TABLE ACCESS FULL          | TB2      |:BF0000|:BF0000
```

In Oracle Database 11g, the Bloom Filter is enabled by default.

Temporary Tablespace Considerations

The temporary tablespace is a global resource. Each instance in the Oracle RAC cluster, as it uses the temp space 'soft' reserves the space in their private SGA. The space, once allocated by an instance, is not visible to other instances and instances do not return temp space to the 'common pool'. So when a process runs out of any allocatable space in the temp tablespace, it requests the other instances in the cluster to free up any unused space. This is done through a Cross Instance Call (CI Call). The session requesting the space will get the 'SS enqueue' for the temporary tablespace and issue a cross instance call (using a CI enqueue) to the other instances (waiting for 'DFS lock handle'). All inter instance temp space requests will serialize on this 'CI' enqueue, and this can be very expensive if the application uses a lot of temp space for temporary tables, sort segments etc (lots of sessions waiting on 'SS enqueue' and 'DFS lock handle') and can cause, severe performance issues.

Best practices for allocating a temporary tablespace are:

- Make sure enough temp space is configured. Due to the way temp space is managed by instance in Oracle RAC, it might be useful to allocate additional space compared to similar single instance database.
- Isolate heavy or variable temp space users to separate temporary tablespaces. Separating reporting users from OLTP users might be one option.
- Monitor the temp space allocation to make sure each instance has enough temp space available and that the temp space is allocated evenly among the instances. The following SQL's are used:

  "select inst_id, tablespace_name, segment_file, total_blocks, used_blocks, free_blocks, max_used_blocks, max_sort_blocks from gv$sort_segment; select inst_id, tablespace_name, blocks_cached, blocks_used from gv$temp_extent_pool;"

  "select inst_id,tablespace_name, blocks_used, blocks_free from gv$temp_space_header;"

  "select inst_id,free_requests,freed_extents from gv$sort_segment;"

  If temp space allocation between instances has become imbalanced, it might be necessary to manually drop temporary segments from an instance. The following command is used for this: "*alter session set events 'immediate trace name drop_segments level <TS number + 1>;*"
- For each temporary tablespace, allocate at least as many temp files as there are instances in the cluster.

See Metalink notes 465840.1 and 469036.1

HOW SHOULD THESE BEST PRACTICES BE APPLIED?

The best practices presented above are applicable to all Data Warehouse environments on Oracle RAC. How should they be applied in your data warehouse? It depends…. Ultimately, the best practices for implementing a data warehouse on Oracle RAC will depend upon too many variables to be covered in a single definitive list that all must follow. You need to be aware of all of them so that you can implement only those that are suitable to your environment and workload, and meet your business needs. It is important to:

Understand and Plan the Interconnect Traffic

The difference between a Oracle RAC and a non-RAC data warehouse is the use of the interconnect in parallel operations. Understanding the interconnect traffic will help decide how much interconnect bandwidth is needed and how to manage it. In addition, high availability considerations need to be taken into account when planning the interconnect configuration.

Configure Parallel Query and Parallel Operations

Parallel operations within a Oracle RAC environment give the flexibility to utilize all the servers that are part of the cluster. In some cases it will be better to use all the available resources, whereas in other cases it will be better to restrict the parallel operation to only one node. The degree of parallelism can be defined to utilize all of the available resources across the cluster, or to restrict parallel query to one or a set of cluster nodes. Also by using instance groups and services, you can control the resource allocation based on your implementation, application requirements or service level. Thus nodes can be logically grouped for specific types of operations.

Manage Your Workload

You should manage and partition your workload using Services. You will have to decide how many services to define and how many instances will offer a given service at any point in time. For example, in some cases it will be better to run a load process on a single instance to avoid contention.

In addition, in some cases the resource manager should be used to optimally allocate resources to services.

Find the Optimal Strategy for Partitioning

Partitions are the foundation for achieving effective performance in a large data warehouse and other features depend on partitioning to achieve the benefit objective. The two important criteria to be considered when choosing partitioning are performance and the ease of administration. The partitioning strategy should be driven by performance considerations. This can mean that the user queries, reports, downloads, etc. will be optimized for performance and scalability, often at

the cost of operational performance and ease of use. As an example, sales data may most often be analyzed by geographic location first and then by date (i.e. the sales for this region over the past month), while the ETL processes will most likely load the data on a date basis (i.e. this week's sales data is loaded in a single batch from across geographic regions). The conflict in this example is that the data may be partitioned first by either geographic region or by date. One approach will favor the user accesses, while the other will optimize for the ETL loads. Ultimately, a balance must be achieved, but most often it will primarily support the users.

In an Oracle RAC implementation, this becomes even more important as efforts are made to reduce inter-node traffic contention. To this end, the workload would best be partitioned along the same lines as the data. This would reduce the likelihood of large amounts of data being passed back and forth between cluster nodes. An example, built on the sales data already discussed, would be that the user reports for particular geographic regions could be processed on specific nodes, while others could be managed on others – thus, the data from certain partitions would most often be accessed exclusively on particular nodes, greatly reducing interconnect traffic.

Measure and Monitor Continuously

The load on a Data Warehouse system changes, daily, hourly, even by the minute. It is imperative that the work being done, the resource usage, the disk throughput, network activity and general health of the Oracle RAC system be monitored and actively managed. Such a diligent approach to caring for the Data Warehouse system will enable the most optimal use of the supporting Oracle RAC system and its resources.

Design and Test to Meet Specific Business Needs

A data warehousing truism, as it might be for all computer systems. However, it has even higher importance in a warehousing system. This is because data warehousing systems have usually had limited, specific-use requirements. Now, as these types of systems evolve and take on more of a mixed-load profile, especially with business needs in flux, it is becoming harder and harder to design effectively.

This state of flux that typifies many of the newer data warehouse systems has become a key business 'need'. Thus, the design for the data warehouse must imbue the data warehouse system with the flexibility to meet ever changing requirements.

An Oracle RAC system, by its very nature, is adaptable, thus enabling it to better meet the developing needs of the data warehouse.

In order to confirm that you truly have designed your Data Warehouse system on Oracle RAC correctly, to meet those business and performance demands, you must test it – thoroughly and under an appropriate load, with clearly defined functional and performance goals.

CONCLUSION

As modern Data Warehouses and their technical requirements have become more common, and more complex, they have come to demand greater flexibility, higher availability and even better performance from their database infrastructure. These increasing demands can readily be met with the implementation of Oracle Real Application Clusters (RAC). This architecture, combined with the feature strengths of the Oracle RDBMS, can provide the stability, availability, scalability and overall functionality to make the Data Warehouse system a success.

In addition, the success of data warehouse system implemented on Oracle RAC depends upon balance: the balance in the design between optimizations for business needs versus operational needs, the balance in the implementation for the performance of a few coincident user accesses versus the scalable performance to satisfy any number of simultaneous processing demands.

Ultimately, the system must be designed and implemented to best meet the needs of the business while at the same time keeping the operational demands to a minimum.

REFERENCES

1. "20TB on a Linux Cluster Today : How To Build a Multiterabyte Data Warehouse, Using Linux and RAC".

http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_bi_dw_build_multi_tb_dw_using_rac_linux_0406.pdf

2. “Exploiting Parallel Operations with Real Application Clusters” by Kevin Conlon

http://www.oracle.com/technology/pub/articles/conlon_rac.html

3. “Parallel Execution with Oracle Database 10g Release 2”

http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_bi_parallel_execution_10gr2_0605.pdf

4. Data Warehousing Guide 10g Release 2 (10.2). Oracle Documentation. B14223-02

5. Data Warehousing Guide 11*g* Release 1 (11.1). Oracle Documentation. B28313-02

6. Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide 10*g* Release 2 (10.2). Oracle Documentation. B14197-03

7. Oracle Real Application Clusters Administration and Deployment Guide 11*g* Release 1 (11.1). Oracle Documentation. B28254-06

Data Warehousing on Oracle RAC Best Practices
October 2008
Authors: Annie Flint & Ian Cookson
Contributing Author: Srinivasan Subramaniam

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