Oracle RAC Internals – The Cache Fusion Edition

Markus Michalewicz
Oracle Corporation
500 Oracle Parkway, Redwood City, CA 94065, USA

Keywords:

Oracle Real Application Clusters (RAC), Cache Fusion, scalability, performance, enhancements, mastering, Oracle Multitenant, Oracle In-Memory Database, best practices, handling contentions

Introduction

Oracle Cache Fusion is the technology that distinguishes an Oracle RAC Database from any other solution on the market. Understanding Oracle RAC internals with the intention to fully understand Oracle Cache Fusion really means understanding the principles or the "secret" to the scalability of any database system. Based on this understanding, any standard operation within a given system, including the Oracle RAC Database, can be understood in principle and only corner cases need to be evaluated. This is the intention of this paper. Based on a brief overview of scalable database management systems in general as well as an overview of Oracle Cache Fusion, it will discuss Oracle RAC-specific operations such as Dynamic Re-Mastering and how to handle contentions. Reviewing how Oracle Multitenant and the Oracle In-Memory Database can be used with and to improve Oracle RAC scalability, the paper will lay the foundation for an easy to scale management of an Oracle RAC Database system. It also provides the basis for future discussions regarding optimized use of memory in a database system.

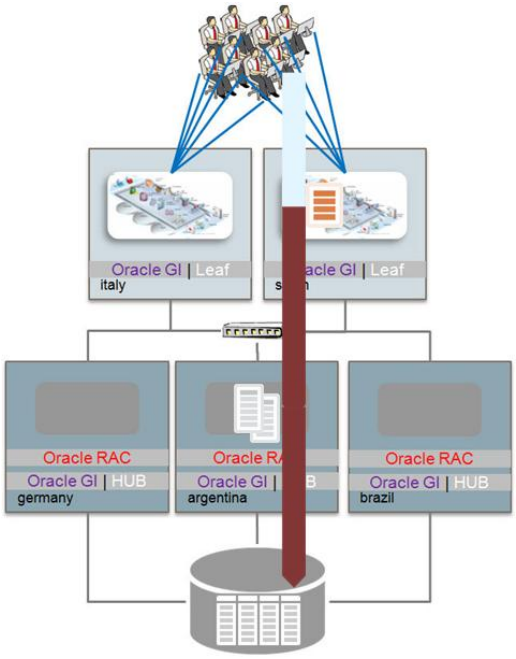

Illustration 1: The Secret to Scalability - A Straight Vertical Line

The Secret to Horizontal Scalability of Any System

If asked about the secret to scalability of any system, the simplest answer is "a straight vertical line". This answer assumes a tiered system such as shown in illustration 1, in which case it describes the shortest access path from the user to the data or more precisely from the place where the data is stored to the place where the data is actually processed. The latter is the basis for further simplifications:

1) The delivery of the result set to the user connected to the application layer (the application user) is typically not a concern. 2) Whether a user connects directly to the database management system (DBMS) or via a connection pool is merely a consideration to prevent contention rather than an access path concern; the per-connection access path to the data is typically the same from the perspective of the DBMS. In case of the Oracle Relational DBMS (from here on referred to as the "Oracle Database"), data is requested by a session connected to an instance regardless of whether the user connects directly to the database or uses a connection pool.

With these simplifications in mind, the actual access path needs to be considered. Most DBMSs use an indirect access path to data these days. "Indirect" means that the client (user) does not access the data files directly, but connects to a management layer, which is used to retrieve the data on behalf of the user, often providing additional capabilities such as caching functionality. In the concrete case of the Oracle Database, the client is connected to the Oracle Database Instance (the set of processes and memory allocated on a server), which will access the data files stored on disk (the database).
Using these definitions of "instance" and "database" for this paper going forward, a multi-instance database represents a database, in which more than one instance (on more than one server) connects to the database stored on concurrently accessed shared storage (shared disk approach). This is also the basic definition of an Oracle RAC database, which will be used later on.

In order to horizontally scale such systems, all that is required is to add instances (on additional servers) to the system. Whether this system uses a shared disk approach as described or uses a shared nothing approach, in which case each instance may actually only be responsible for a subset of data, would not matter in this case, as long as the DBMS as a whole guarantees (by whatever means) that the data that is requested by a user can transparently be delivered to the instance, to which the user is connected. The art, and hence the secret to scalability of such a system, would then be the way the DBMS routes the user request to the best possible instance to operate on the data.

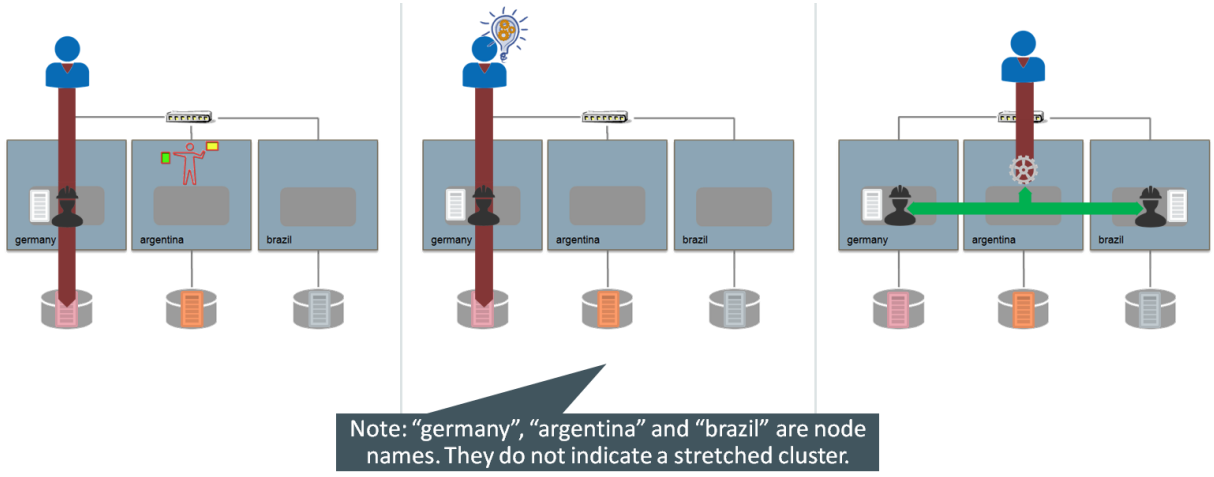

Illustration 2: Three Alternative Architectures - The Same Idea Applies

As illustration 2 shows, various systems may use different approaches to this problem. Some DBMSs may route the user to an instance based on some information that the user explicitly or implicitly provides while requesting access (such as a partition key, by means of which data is partitioned across the servers). Some other systems may expect the user (the application) to be aware of the fact that certain instances can only (optimally) serve a certain subset of data, in which case the application needs to choose the instance to which it should connect. Last but not least, the DBMS may decide to accept user connections on any instance and spawn a sub-process to retrieve data from any other instance, if required, as part of a sub-operation that will then return the partial data from multiple instances to the one that the user is actually connected to. This instance will then assemble the final result-set based on the sub-data retrieved from other instances.

Oracle RAC Combines It All and Adds Services

The secret to Oracle RAC scalability is that Oracle RAC can facilitate all approaches and techniques stated above in addition to some that are specific to this RDBMS solution. As this is the case, it is fair to assume that any DBMS based scalability problem can be solved based on an Oracle RAC system, assuming that changes are permitted on any layer of the stack, including the application layer. It has always been the expectation as well as Oracle's intention, however, to ensure Oracle RAC scalability without the need to make changes to the application. This article therefore only discusses techniques and components as part of the standard Oracle RAC offering, including services and connection pools.

In case of Oracle RAC, an incoming connection request is "guided" by the listener considering the availability of a cluster-managed service (a.k.a. Dynamic Database Services as per Oracle documentation). Starting with Oracle RAC 11g Release 2, Oracle RAC uses two layers of listeners, the SCAN listeners and the node listeners, to guide a connection request to the final instance in a cluster. For all practical purposes, this article will ignore the layers of listeners acknowledging that for establishing the "straight vertical line" paradigm, only the final connection with the instance matters. It will, however, consider the availability of a cluster-managed services, which by default, is assumed to be offered on all instances managing the database, as this would be the basis for horizontal scaling. Last but not least, parallel execution (PX) and parallel query (PQ) operations are explicitly excluded from the scalability considerations discussed as part of this paper. Parallel query is used by default, any time a GV$ table is queried in an Oracle RAC Database. However, the overall contribution of parallel operations to improve scalability of any application (as opposed to specific applications) is subject to a different discussion.

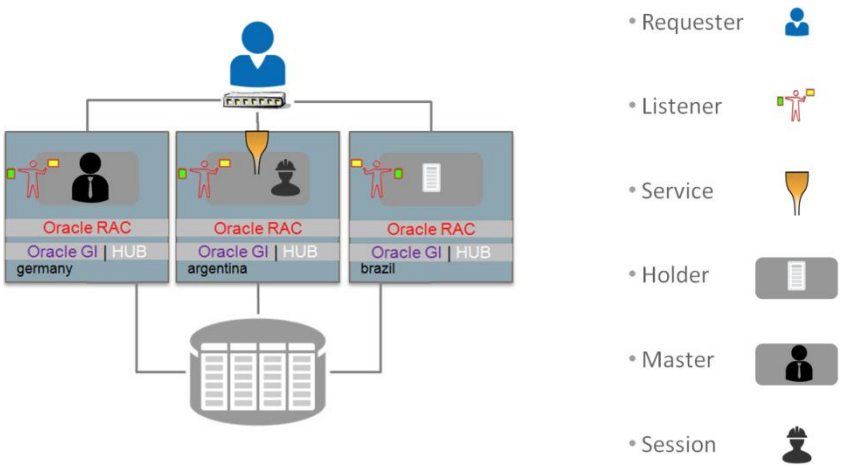

Illustration 3: Oracle RAC Combines It All

Oracle RAC Scaling Principles

One of the main principles to bear in mind is that there is no system that scales perfectly linearly. It is often assumed that Symmetric Multiprocessing (SMP) systems provide linear vertical scaling; or in other words that vertical scaling is loss-free. This is not the case, which is why both, SMP systems and clusters, which are commonly used to provide horizontal scaling, should be evaluated based on the needs at hand. For more information see: "In Search of Clusters: The Coming Battle in Lowly Parallel Computing" by Gregory F. Pfister who concludes that "Depending on the reader's needs, cluster-based systems may be more appropriate than a single large computer system or a SMP system."

For an Oracle RAC Database system, this general principle constitutes another one: an application will only scale horizontally on an Oracle RAC Database, if the same application scales on an SMP system. Vertical scaling in this context is often referred to as "scaling over CPU". This means that by adding more CPUs to the system, the system should either be able to serve more users, more data or decrease the response time with the number of users and amount of data unchanged. If this principle is fulfilled, there is a good chance that this application will scale on an Oracle RAC system, too.

In this context, it should be noted that vertical scalability (over CPU) is typically measured across all the CPUs in one system. In order to use a comparable approach to measuring horizontal scalability, any scalability related statement in this paper assumes scalability across all nodes in the cluster. Example: if a server has 16 CPUs and the full RAC system consists of three of those servers, then the scalability is expressed based on the power provided by the system as a whole (3*16 CPUs= 48 CPUs). A scalability of 80% would then mean that the Oracle RAC system should perform with an equivalence of 80% of the CPU (=38.4) power provided by the system (for CPU bound applications).

Accessing Data Across Instances

In any distributed system in which instances as previously defined also act as a data cache, it is not guaranteed that the instance will hold (cache) the data required to fulfil a user request at any given point in time. In such cases, the instance to which the user is connected ("session holding instance") needs to obtain the data either from disk or, if the DBMS allows for such an operation, from a "remote instance". A "remote access" is typically performed by a network based request in this case. It is therefore reasonable to assume that a local or remote cache access is faster than a spinning disk access.

In an Oracle RAC database, three cases can be distinguished; 1) local Cache hit, 2) remote cache hit and 3) disk access. The "local cache hit" is the most efficient access to data, as it assumes that the session holding instance is also the data holding instance and hence the user can access the data without any further operation. In case of a remote cache hit, the session holding instance needs to request the data from a "remote cache" via the network. Only if no instance in the cluster currently holds the data, the system would need to access the disk to obtain the data.

As the Oracle RAC database assumes a shared disk system, the session holding instance will obtain the data from disk. It needs to be noted in this context that once the data is cached in an instance it remains in the instance for subsequent local or remote access (subject to buffer cache aging and flushing rules). When an instance obtains data, the Global Resource Directory (GRD) is updated and synchronized across instances via lightweight messages, which ensures efficient subsequent access.

One of the main misconceptions when it come to accessing data in an Oracle RAC system is that the above described flow (local access first, remote access second, disk access as the last resort) is always maintained, which is not the case. For a single data access it may appear as such. However, if a request (e.g. a SQL statement) requires data from three different locations as shown in illustration 4, the Oracle Cache Fusion Algorithm allows for "Dynamic Data Retrieval".

Dynamic Data Retrieval describes a scenario, in which the Oracle RAC database may actually decide to avoid a remote cache access in favour of accessing the disk from the session holding instance directly. The decision to avoid the typically more efficient remote cache access is made dynamically and based on interconnect and disk access IO statistics that are refreshed periodically as part of the algorithm. Typical scenarios include, but are not limited to, cases in which the interconnect got saturated after serving some sub-data as part of the same or another request, in which case as (sequential) read from disk may be more efficient at this moment in time.

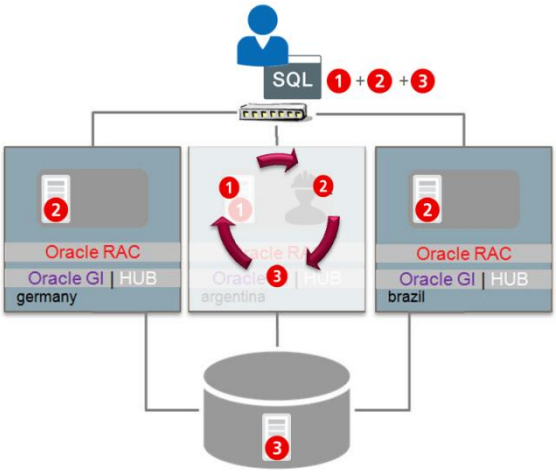

Illustration 4: Dynamic Data Retrieval

(Dynamic Re-) Mastering

Accessing data across instances needs to further distinguish whether the requested access is a read-only request or involves a write-request (updates or inserts are treated the same for this matter). If the request includes a write request or a potential for a write ("select for update"), lock grants to access the data need to be considered in addition to obtaining the data itself.

In an Oracle RAC system, the same consistency rules as in a Single Instance Oracle Database apply. This means, the Oracle RAC database observes read consistency for multi-user access just like in a single instance environment, ensuring read-consistency across the Oracle RAC database instances. An Oracle RAC database system also provides a row-level locking across instances, which means there is no escalation for maintaining a lock. *No* page or table locks or other, high granular access restrictions, which negatively impact scalability, are required within an Oracle RAC system.

In order to ensure the data consistency that the Oracle Database is famous for, the Oracle RAC system uses a mastering concept. In short, any object (e.g. a table, or, if the table is partitioning, the partition) in and Oracle RAC 11g Release 1 or later database is mastered by one instance of the database system at any point in time. (Note in this context that the mastering granularity has changed over versions and older versions had a higher granularity level.). Once a master instance is assigned, this instance will master and lock request to any object that it masters.

This means that a write-access to a given set of data (in simple terms, a "database block") has to request the lock grant from the mastering instance. The mastering instance can be the same or different from the session holding and / or data holding instance. This means that full access to data needs to consider the entities: 1) the session holding, 2) the mastering instance and 3) the data holding instance.

As illustration 5 shows, these entities can either be "all local", "local and remote" or "all distributed", in which case a three-point communication is performed to write to data: 1) the session holding instance asks the mastering instance for a lock grant to access data for write purposes. 2) The mastering instance will advise the data holding instance to ship the requested data along with a lock grant to the session holding instance. 3) The data holding instance will then ship the required data block(s) to the session holding instance.

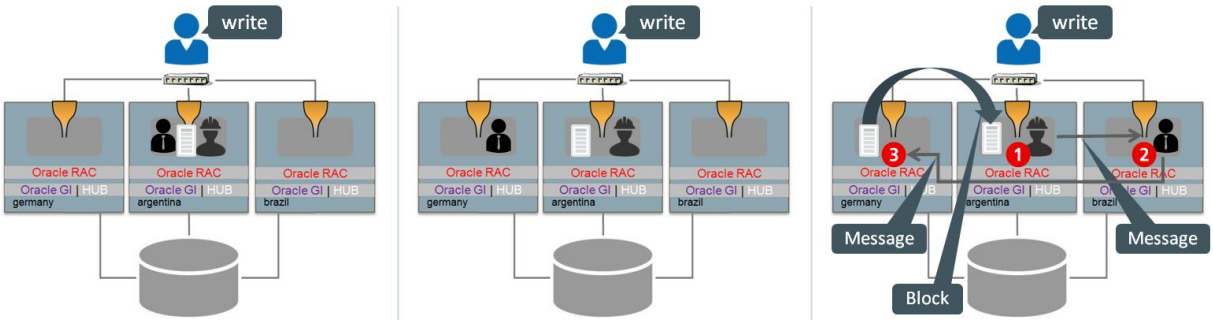

Illustration 5: Maximum Three Way Communication

In this context, it needs to be noted that the communication that is described here is performed over a dedicated network communication between the nodes in the cluster; the private interconnect. The interconnect is used for what is commonly referred to as "function" (message based) as well as "data shipping" (block transfer) and is therefore subject to more utilization than its equivalence in a failover cluster, failing over Single Instance Oracle Databases for example. In those clusters as well as in Oracle RAC One Node clusters (for more information on Oracle RAC One Node see here: http://www.oracle.com/goto/racone), no data shipping is performed. Oracle therefore recommends to not only use a redundant interconnect, but also to size the interconnect in anticipation of its utilization under normal operations. Further guidelines regarding interconnect sizing can be found on pages 8ff of this document "Oracle RAC 12c Practical Performance Management and Tuning OOW13"

The exact operations that are performed to obtain write access to a block may also differ between Oracle RAC versions, as the code area of the Oracle Cache Fusion technology, which deals with this type of access, is subject to constant improvements without notice. For example, changing from Oracle Database 10g Rel. 2 to later releases, significant improvements have been made in the area of reducing message in the so called "time relevant phase". This is the phase, during which a session is waiting to acquire the block for write purposes. Any message that can be reduced during this time helps to improve the performance for this operation, although messages are typically small and quick.

Another way of optimizing the communication in the cluster for the purpose for write access to data is to dynamically change the mastering instance, once it is determined that a majority of requests to a set of data come through an instance that is not the master of this set of data. This type of "shifting the master to another instance in the cluster" is called Dynamic Remastering (DRM) and has been a standard functionality of Oracle RAC since its release with Oracle Database 9*i*.

Similar to other code areas, DRM has been subject to quite some enhancements over releases. Unfortunately, there has been one release (10.2.0.3), in which the DRM default configuration caused more interruption when called than providing benefits thereafter. Hence, customers tended to disable DRM for this release. Starting with Oracle Database 11g Release 2, DRM is enabled by default (potentially overriding a previous disablement during upgrade). At the same time, the DRM default setup has been improved to allow for a more optimized execution of DRM based on access patterns.

Oracle recommends leaving DRM enabled under all circumstances, as it is used for internal as well as external operations. It is typically only external operations (user access patterns) that will trigger a DRM operation that leads to a remastering of objects. If the remastering is considered interruptive to user operations, consider optimizing _gc_policy_minimum, if necessary with help of Oracle Technical support. Also, note that the use of large SGAs (currently defined as > 300GB) can lead to longer DRM operations, for which reasons, "My Oracle Support" (MOS) note 1619155.1 should be reviewed, if such large SGAs are used. For more information contact Oracle Technical Support.

Handling Contentions

Contentions are common occurrences in a multi-user system in which more than one user can request write access to the same data at the same time regardless of whether a single (SMP) system or a multi-user / multi-instance system is used. In a multi-instance system, due to its distributed nature, contention can be more impactful than on a single server system. In either case however, user contention is best to be avoided. If an application causes contention in a Single Instance Oracle Database (typically identified by a "concurrency" wait class in an AWR report), it is fair to assume that the same contention will be more visible in an Oracle RAC environment, potentially impacting its performance. It is therefore best to fix any contention related issue whenever it occurs.

Unlike in other database management systems, only simultaneous writes can cause contention in an Oracle Database, as read operations will not be impacted by a write operation. The same is true in an Oracle RAC database, which can be operated similarly to a single instance database by routing all user connection requests to only one instance the RAC database. In this case, contention would mainly occur between sessions in the same instance rather than between instances, which will lead to an increased function and block shipping. It also shows that from a contention perspective, a 2-node or more than 2-node setup does not make a huge difference.

The reasons for contention are manifold, but typically caused by the application. The common principle behind any contention is frequent transactional changes to the same, typically rather small, set of data by multiple users at the same time. This leads to "hot write spots", which in an Oracle RAC database can lead to an increased block shipping between instances. The "concurrency" wait class thereby became a "cluster" wait class (in addition), as the cluster may have to wait for retrieving the block in a remote instance.

Block shipping as a result of a write request from a remote instance leads to shipping of the current block (as opposed to a consistent read image), which means that a log-flush (writing pending redo-data for the block(s) being shipped) needs to be performed on the data holding instance, which is also the one that will send the block to the requesting instance. This means that the block acquisition time depends on the time for the log-flush. This is an Oracle RAC-specific aspect and needs to be considered accordingly. Using Solid State Disks (SSDs) for redo log data mitigates the problem.

Contention can be caused directly or indirectly. Contention is caused by frequent transactional changes to the same set of data by multiple users at the same time. This means, contention can occur on dependent or metadata as well. Even if the user data is optimized so that users operate on distinct data sets for the most part, any common set of metadata or dependent data can still cause contention to some degree. The Oracle Database uses internal optimization to prevent such indirect contention as much as possible (e.g. Locally Managed Tablespaces were introduced to relieve data dictionary contention, while Automatic Segment Space Management introduces the idea of using an efficient bitmap rather than a linked-list approach to managing storage extents and free blocks). However, user-created indexes on user data can still be subject to contention, although the user data is not.

The solution to any kind of contention is therefore to either increase the size of the data set that a group of common users is actively working on or decrease the number of users that commonly access a given data set that is therefore subject to contention. In an Oracle Database, this typically means using partitioning and / or connections pools, ideally both. While the basic idea behind using connection pools is to reduce the number of sessions connected to the database and thereby reduce the number of concurrent access on an object, partitioning data needs to be considered carefully.

The Oracle Database, unlike other database solutions, supports logical partitioning. This means that the data is partitioned from a process and access point of view without the necessity (while possible) to reflect the partitioning scheme on disk. In general, this type of partitioning can be used to reduce the likeliness for concurrent access by multiple users via distributing the data over various buckets with the intention to spread the users randomly across those buckets (hash partitioning). It needs to be noted, however, that both techniques need to be used together, as otherwise this type of partitioning would only further reduce the size of a data set, which would mostly lead to a negative effect. Using a list or range partitioning scheme is useful, if the application using the database can be optimized to make use of this partitioning scheme, in which case the assumption is certain sets of users can be guided to mainly access a certain partition.

In an Oracle RAC database, the same principles apply. With respect to using partitioning for user data, the assumption is that cluster-managed or dynamic database services are used in addition so that certain groups of users would access a given set of partition(s) on a certain instance of the cluster most of the time (“straight vertical line” approach). This will lead to this instance to be the session and data holding instance for this data set and, over time, might lead to this instance being the mastering instance of those partitions, as DRM would re-master based on access patterns. The alignment of the application with the partitioning schema can be on various levels. Today’s applications often use distinguishable modules. If there is a way to assign a different service to each module and each module in turn majorly works on a distinguishable set of data, a perfect alignment can be established. Emphasis here is on “majorly works”. As the Oracle RAC database is shared disk / shared cache base, even frequent deviations from this alignment, or better, assignment are not as impactful as similar operations on database systems using a shared nothing or sharding approach. It explains, however, why any DBMS based scalability problem can be solved based on an Oracle RAC system, assuming that changes are permitted on any layer of the stack, including the application layer.

Partitioning can also be used on metadata, especially user-created indexes. As logical partitioning is used for the Oracle Database, indexes can be partitioned independently of the associated user data (and vice versa). If an AWR report (or Oracle Enterprise Manager) reveals an index related issue, various techniques can be used for mitigation. One of the most frequently recommended approaches is to use a reverse key index, which assumes that the contention was caused by a right-growing index typically. Other techniques may be suitable to mitigate such issues, while some of them may have side effects to consider. This discussion is outside of the scope of this paper. Using either global hash partitioned

indexes or locally partitioned indexes have shown to be effective if an index contention is reported, as both of these techniques generally lead to a better cache locality. Needless to say that removing unnecessary instances is the best approach to avoiding index contention.

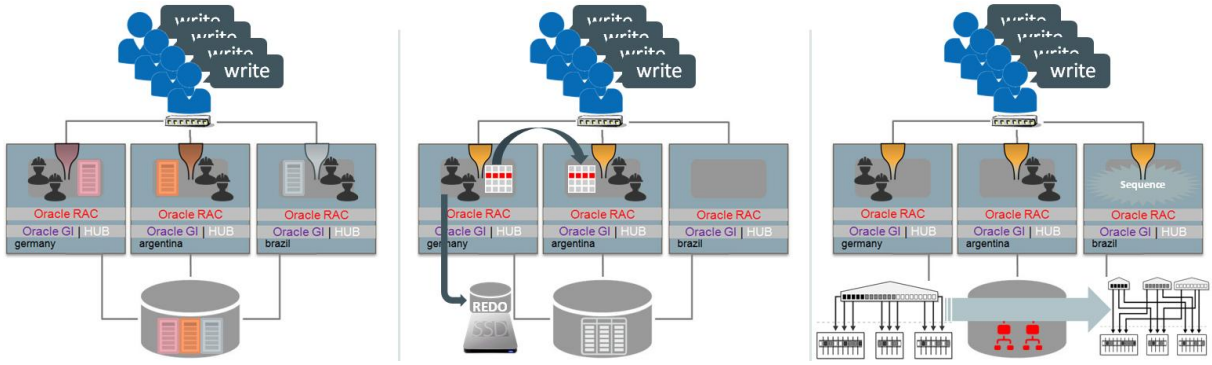

Illustration 6: General Solutions to Handling Contention

Changing the data set size is only half of the work. Contention by definition is typically caused by a high number of users concurrently working on a rather small data set. The other half of the solution is therefore to reduce the number of concurrent access to the data. In an Oracle Database users are reflected within the database as sessions, while in an Oracle RAC Database those sessions are assumed to be distributed across the instances in the cluster. Either way, reducing the number of sessions will help to prevent concurrent access to data.

The simplest way of reducing the number of sessions connected to the database as a whole without impacting the performance of the system is using a connection pool. Using an Oracle Fast Application Notification (FAN)-enabled connection pool, such as the Oracle Universal Connection Pool (UCP), can further improve the user experience. FAN-enabled connection pools can sign up for retrieving Workload Repository-based Load Balancing information on a per-service basis, which improves runtime load balancing behavior, leading to a better overall utilization of the system. The UCP also provides "Connection Affinity" (Transaction-Based or Web Session based), which can help to facilitate local access. For more information on services and optimized runtime load balancing in an Oracle RAC Database, see the Oracle documentation, especially the chapter on "Managing Workloads Using Dynamic Database Services"

Oracle RAC Meets Oracle Multitenant

Oracle Multitenant and Oracle RAC may have more in common than what meets the eye. Looking under the hood of both technologies reveals that they have a symbiotic relationship. Both technologies – in their own way – are used to consolidate databases. Using Oracle RAC, database consolidation on a cluster is not an uncommon use case, which some customers have further optimized by using schema consolidation on a RAC-enabled database. The latter use case is nowadays much better addressed using Oracle Multitenant, which not only solves quite some of the issues that schema consolidation entails, it also has a better integration into the Oracle RAC infrastructure. For more information on the integration and optimized use of Oracle Multitenant Databases with Oracle RAC, see: http://www.slideshare.net/MarkusMichalewicz/oracle-multitenant-meets-oracle-rac-ioug-2014-version

Oracle Multitenant Databases and Pluggable Databases (PDBs) simplify consolidation by removing the need to consider inter-instance interaction and competition for server resources on a per-server basis, assuming only one Container Database (CDB) or a very limited number of CDBs are used on a per server basis, as those could easily be managed using current technology, such as Instance Caging and common memory management approaches. If more than a limited number of database instances are used on a per server basis, consolidation considerations such as the ones discussed in the document linked would need to be considered, typically prolonging the deployment of a new database, which is particularly disadvantageous in a (self service) Database as a Service environment.

If PDBs are used to consolidate formerly independent databases in a CDB, resources for the PDBs can be managed within the scope of the container, which allows for more control over resource utilization using the Oracle Resource Manager in general. Using PDBs also reduces the amount of background processes as well as memory that need to be allocated on the server, increasing consolidation density.

However, increasing consolidation density within a container means increasing concurrent access on shared resources. One concern that is typically raised is that the redo-log data for each PDB would go into the same set of redo log files. This is why the Oracle Database 12c uses a multithreaded log-writer by default. Using SSDs for the redo log placement as previously recommended would further mitigate potential contention on the redo log files and thereby possibly even prevent it.

Oracle RAC Meets Oracle In-Memory

Oracle In-Memory addresses a variety of use cases facilitating memory in a more efficient way. For an Oracle RAC databases, it addresses at least two immediate use cases directly as shown in illustration 7 below. Oracle RAC databases have traditionally been used to perform Online Transaction Processing (OLTP) and Data Warehouse (DWH) operations on the same database, mostly by separating the different use cases on different nodes in the cluster. By Oracle In-Memory providing a "Dual Format" Database, allowing for OLTP and DWH operations being performed on the same database with lesser interference, these benefits directly translate to an Oracle RAC Database.

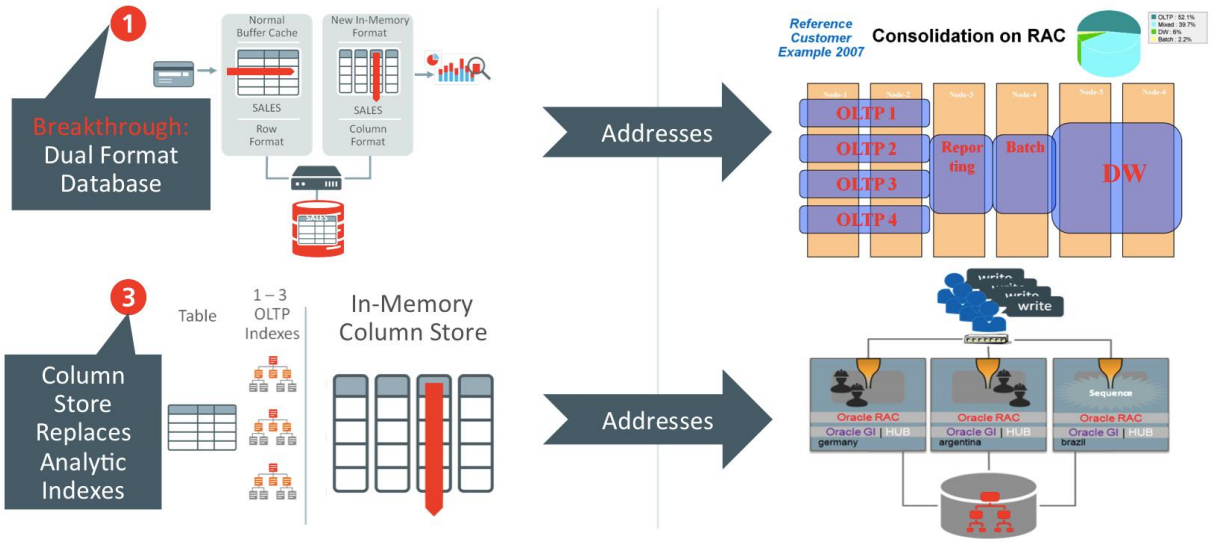

Illustration 7: Oracle In-Memory Address Oracle RAC Use Cases Directly

Illustration 7 also shows that Oracle In-Memory replaces analytic indexes using the Column Store. As previously discussed, indexes can be subject to contention in general as well in an Oracle RAC database, which is why one consideration could be to remove indexes that are not (frequently) used. With Oracle In-Memory, analytic indexes can be removed very easily, which has an immediate effect, if index contention based on such instances was identified for an Oracle RAC database.

Another benefit of using Oracle In-Memory for an Oracle RAC Database may not be as obvious and requires some general understanding of the Oracle In-Memory architecture. The key component in this architecture is the In-Memory Column Store, which is a separate area in the SGA of each instance. Objects in the column store are compressed and are subject to a simplified lock management. Which means that operations on the column store will not affect overall scalability for the data managed by the rest of the database and data in the regular buffer cache, which in turn will be smaller. As discussed as part of Dynamic Remastering, smaller database instance can lead to better DRM ad recovery performance. Of course, using Oracle In-Memory is only the first step of using memory in the stack in a more efficient way deserves another, more in-depth detailed discussion.

Summary

Managing an increasing amount of data using a database management system and scaling application workload over a database system to serve an increasing or fluctuating amount of users is one of the oldest and at the same time one of the most challenging problems IT departments face. There are, however, only two dimensions to scalability; vertical and horizontal, and, of course, a combination therefore. For horizontal scaling various database management systems use different approaches, which all have the same goal; bring the data as close to the place where it is processed and have the user connect to the very same place. A variety of techniques can be used when trying to achieve this goal, however, there are pros and cons for each one of them. Assuming that changes can be made in each layer between the user who connects to the application and the application connecting to the database management system, any scalability problem that can generally be solved by a database can be solved by an Oracle RAC Database. It is the goal of Oracle RAC, however, to solve any problem without requiring changes to the application, at least considering standard applications. Continuing its efforts, Oracle will strive to maintain this goal going forward.

Contact address:

Markus Michalewicz
Oracle America
500 Oracle Parkway M/S 4OP840
Redwood City, CA, 94065
United States of America

Phone: +16505065444
Fax: +16505065444
Email Markus.Michalewicz@oracle.com
Internet: www.oracle.com/goto/rac
Twitter: @OracleRACpm