Cache Fusion: Extending Shared-Disk Clusters with Shared Caches

Tirthankar Lahiri, Vinay Srihari, Wilson Chan, Neil Macnaughton, Sashikanth Chandrasekaran

Oracle Corporation

{tirthankar.lahiri, vinay.srihari, wilson.chan, neil.macnaughton, sashikanth.chandrasekaran}@oracle.com

Abstract

Cache Fusion ^{TM} is a fundamental component of Oracle's Real Application Cluster configuration, a shared-cache clustered-database architecture that transparently extends database applications from single systems to multi-node shared-disk clusters. In classic shared-disk implementations, the disk is the medium for data sharing and data blocks are shipped between nodes through disk writes and reads under the arbitration of a distributed lock manager. Cache Fusion extends this capability of a shared-disk architecture by allowing nodes to share the contents of their volatile buffer caches through the cluster interconnect. Using Cache Fusion, data blocks are shipped directly from one node to another using interconnect messaging, eliminating the need for extra disk I/Os to facilitate data sharing. Cache Fusion thus greatly improves the performance and scalability characteristics of shared-disk clusters while continuing to preserve the availability benefits of shared-disk architectures.

1. Introduction

A cluster is a group of independent servers that cooperate as a single system. The key components of a cluster are the constituent server nodes, the interconnect, and the disk subsystem. The Oracle *Real Application Cluster* (RAC) architecture is a clustered database architecture running on a shared-disk cluster, a form of cluster in which all nodes have direct access to all disks. RAC is so called since it transparently allows any database application to run on a cluster without requiring any application changes. RAC allows for improvements in application performance since the application is executed in parallel across multiple systems, as well as improvements in availability, since the application is available as long as at least one of the cluster nodes is alive.

Proceedings of the 27th VLDB Conference,
Roma, Italy 2001

In a classic shared-disk clustered database, the disk is the medium of data coherency across the cluster nodes. For instance, if a node requires a copy of a block that is presently dirty in another node's buffer cache, the second node must first write the block to disk before the first node can read the block.

However, recent advances in cluster hardware technology merit a fresh approach to building clustered databases. Storage Area Networks (SAN) now provide sophisticated mechanisms for disk connectivity, circumventing the limitations of directly attached disks by allowing each node to be connected to a much larger number of disks. For example, *Infiniband* ^{SM}[2] is an emerging standard for high-performance clusters, and by using the same protocol for I/O and inter-node messaging allows cluster networks that carry both data and cluster interconnect messages. Other high-performance commodity interconnect standards such as *Virtual Interface Architecture* (VIA)[4] now allow vendors to build high-performance clusters from standard components. These advances mean that clusters are now becoming mainstream, capable of high data volumes and high data-transfer bandwidths.

Cache Fusion exploits these advances in clustering technology by using the network rather than the disk as the medium for data sharing between nodes. With the Cache Fusion protocol, blocks can be shipped directly between Oracle Instances through fast inter-node messaging, without requiring expensive disk I/O. Oracle instances therefore directly share the contents of their volatile buffer caches, resulting in a *shared-cache* clustered database architecture.

The rest of this paper is organized as follows: Section 2 contains a brief overview of the Real Application Cluster architecture. Section 3 describes Cache Fusion protocols, highlighting techniques for read-sharing, write-sharing, as well as efficient inter-node messaging. Section 4 briefly discusses RAC mechanisms for Decision Support workloads. Section 5 describes recovery mechanisms with Cache Fusion. Finally, Section 6 concludes.

2. Overview of Real Application Clusters

An Oracle Instance is a collection of processes and memory accessing a shared set of data files (see Figure 1 below). Each Oracle instance inside RAC has its own private set of log files referred to as a *Redo Thread*. Each instance also has its own buffer cache of disk buffers, and taken together, these local caches form a global buffer cache. In order to maintain cache coherency in this global cache, global resource control is needed. We call this resource control mechanism the *Global Cache Service* (GCS). For additional details on RAC and GCS, see [3].

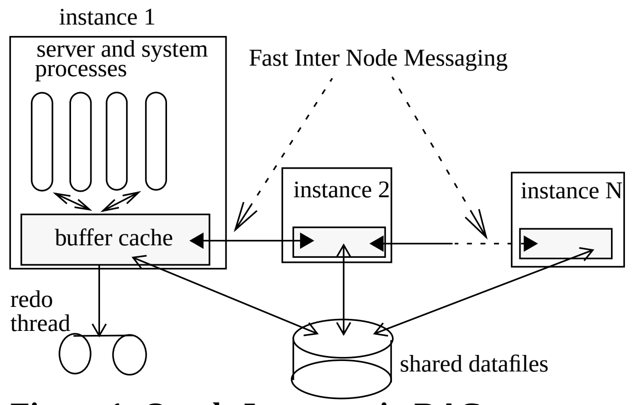

Figure 1: Oracle Instances in RAC

The GCS tracks and maintains the locations and access modes of all cache resources (data blocks) in the global cache. It synchronizes global cache accesses, allowing only one instance at a time to modify a cache resource.

The GCS adopts a distributed architecture. Each instance shares the responsibility of managing a subset of the global cache. There are several advantages to this approach. First, the work of handling cache resource requests can be evenly divided among all existing database instances. Second, in case of hardware or software failure in a node, only the instance running on the failed node is affected. Access to cache resources managed by this instance may be temporarily unavailable. However, all other resources will continue to be accessible.

The assignment of global resources to a particular instance takes into account the access pattern of cache resources. Resources accessed most frequently by an instance will be likely to be managed by the same instance.

By knowing the global view of all data blocks, GCS can direct a read or write request to the instance that can best serve it. For example, suppose an instance issues an updatee request for a particular block to the GCS. The GCS will then forward the request to the instance which has the current cached buffer for that block. This current holder will transfer the cache buffer to the requester instance directly, and the GCS will then update the holder information to reflect the fact that the requesting instance is now the holder.

3. Cache Fusion

Cache fusion refers to the protocol for sharing of instance buffer cache contents through fast inter-node messaging, resulting in a cluster-wide global buffer cache. There are two types of sharing involved: *Read-Sharing,* which refers to the mechanism used by a query to access the contents of another instance's buffer cache, and *Write-Sharing* which refers to the mechanism by which an update operation accesses data in another instance's cache. In the following subsections, we describe both kinds of sharing, followed by a brief description of inter-node messaging.

3.1 Cache Fusion Read-Sharing

The mechanism for read-sharing in Cache Fusion exploits Oracle's *Consistent Read* (CR) mechanism [1]. CR is a version-based concurrency control protocol which allows transactions to perform reads without acquiring any locks. Each transaction in Oracle is associated with a snapshot time, known as the *System Change Number* (SCN), and the CR mechanism guarantees that any data read by a transaction is transactionally consistent as of that SCN. When a transaction performs a change to a block, it stores the information required to undo that change in a rollback segment. When a transaction reads a block, the CR mechanism uses the stored undo information to create an earlier version of the block (a clone) which is consistent as of the reading transaction's SCN. Clones are created in-memory and are never written to disk. A read operation therefore never needs to wait for another transaction to commit or abort since the CR mechanism automatically reconstructs the version of the block required by the operation. This mechanism therefore allows high concurrency for read operations.

In RAC, when Instance A requires read access to a block that is present in the buffer cache in Instance B, it requests a copy of the block from Instance B without requiring any change of resource ownership. Instance B creates a transactionally consistent CR clone of the block and ships it back to Instance A across the interconnect. Doing so has no impact on processes on Instance B since ownership of the block by Instance B is not affected.

Only when the requested block is not present in any instance's cache is a disk I/O performed for the block. However, the Read-Sharing protocol guarantees that once a block is read from disk by any instance in RAC, subsequent read accesses to that block from any other instance do not require disk I/O or inter-node ownership changes.

3.2 Cache Fusion Write-Sharing

Write-sharing is handled by the GCS. When Instance A wishes to update a block it invokes the GCS to perform the necessary cache-coherency messaging to obtain a copy of the block.

If the GCS determines that the block is already in another instance (B's) buffer cache, it notifies instance B that it must release its ownership of the block. Instance B then saves a copy of the block in its cache for future read access and releases ownership. Along with the message acknowledging the release, Instance B also ships its cached copy of the block to the requesting instance, even if that copy is dirty (i.e. contains changes that have not been written to disk). Thus, sharing dirty buffers between instances does not require any extra disk writes and reads.

Only if the block is not already present in any instance's buffer cache must the requesting instance issue a disk read for the block.

An important benefit of the write-sharing fusion protocol is that after a write request is performed for a block, the instance that had the current copy can continue to perform read accesses on the block. Thus, in the above example, Instance B can continue to perform read accesses on its cached image of the block even after it has relinquished ownership of the block and sent a copy over to Instance A. This is in contrast with typical shared-disk protocols in which a write request by a node invalidates all cached copies and prevents any other nodes from accessing that block for the duration of the write.

Cache Fusion Read-Sharing and Write-sharing therefore ensure that in RAC, the total number of disk I/Os that need to be performed is comparable to the number of I/Os that would be performed by the same workload running on a single instance of Oracle with a buffer cache equal to the sum of the buffer caches of all the constituent RAC instances.

3.3 Efficient inter-node messaging

The protocols described so far reduce the number of I/Os required for inter-node data-sharing. Also critical to the scalability and efficiency of a clustered database is the efficiency of inter-node messaging. There are three factors that contribute to the efficiency of inter-node messaging, these are discussed below:

- **Latency of inter-node messaging:** Cache Fusion is essentially a large state machine, and uses fixed-length, fixed-format messages which can be generated and interpreted very efficiently, as opposed to higher-level SQL messages which are an order of magnitude more expensive to generate and interpret. Furthermore, exploiting high-performance communication substrates like VIA implies that on-the-wire message transmission times are minimal.

- **Number of nodes involved in servicing a request:** With Cache Fusion, at most three nodes are involved in any block request: the requesting node, the owner of the directory information for the block requested, and the holder node. This means that the number of nodes and messages required to service a request is constant and does not grow with the number of nodes in the cluster, allowing a RAC cluster to scale to large numbers of nodes.

- **Frequency of inter-node synchronization events:** The GCS has been designed to minimize the number of such events through an adaptive and dynamic directory migration mechanism: the instance that most frequently accesses a particular set of blocks will eventually end up owning the directory information associated with those blocks. Thus, over time, any instance-locality patterns are observed to affine directory information to the instances that most frequently access the corresponding resources. Local ownership of directory information greatly reduces the number of inter-node events in the cluster.

4. RAC support for DSS workloads

The Cache Fusion sharing protocols described earlier facilitate fast fine-grained data sharing for OLTP applications. RAC also benefits high-bandwidth Decision Support workloads through parallel execution across cluster nodes. Oracle's cluster-aware cost-based optimizer takes into account cluster topology, such as any affinity of disks to nodes, the storage parallelism for a table, the number of nodes and the number of cpus on each node, etc. when computing an appropriate parallel execution plan for a query. The parallel execution engine also takes into account the relative cost of remote vs local execution when assigning parallel slaves to a query, preferentially allocating local query slaves for a query whenever possible.

A powerful mechanism for reducing the cost of remote parallel execution is *Function Shipping*, a mechanism traditionally employed only in Shared-Nothing systems. With Function Shipping, parallel execution servers are sent modified SQL queries to indicate the work that needs to be performed. Function Shipping requires far fewer messages to be interchanged than the more typical *Data-Shipping* approach for shared-disk systems.

5. Recovery in a RAC environment

While Cache Fusion provides a high-performance cluster-database architecture for both OLTP and DSS operations, it also facilitates high-performance for recovery in the event of instance failure. The RAC architecture guarantees availability of the database as long as at least one instance is alive. Recovery cost in a RAC environment is also designed to be proportional to the number of failures, not the total number of nodes in the cluster since only redo logs from failed nodes are read and applied. The key advantage provided by Cache Fusion is that disk reads for recovery are eliminated for blocks that are present in a surviving node’s cache, thus speeding up recovery.

Global Cache Resources can be made available as soon as the set of blocks needing recovery has been identified following a scan of the redo log. Cluster availability can therefore occur well before application of redo to the recovery set begins.

Parallel recovery mechanisms exist to make use of all surviving nodes, particularly in the case of large numbers of instance failures. This provides the benefit of parallelizing disk I/Os for log reads and data block reads across multiple recovering instances and further reduces recovery time.

6. Conclusions

In this paper, we have briefly described Oracle’s Cache Fusion architecture, a shared-cache approach to building clustered databases for shared-disk clusters. Cache Fusion allows off-the-shelf applications to run on shared-disk clusters with no changes. This architecture exploits advances in hardware cluster technology by using the network rather than the disk as the medium for data sharing. Multi-versioning read-sharing protocols based on Consistent-Read allow queries to access cached data in the cluster without requiring inter-node coherency operations. Cache Fusion write-sharing protocols allow update operations to operate on buffers shipped directly across the cluster interconnect from other instances, even if those buffers contain unwritten changes. Efficient inter-node messaging allows a Cache Fusion cluster to scale to large numbers of nodes. A cluster-aware cost-based optimizer and Function Shipping allow large DSS workloads to fully exploit the processing capabilities of the cluster, again without any user-level knowledge of cluster topology or application-level optimizations. Cache Fusion also improves the performance of recovery in the event of any failures in the cluster by allowing recovery to utilize cached copies of blocks requiring recovery.

References

[1] W. Bridge, A. Joshi, M. Keihl, T. Lahiri, J. Loaiza, and N. Macnaughton. The Oracle Universal Server Buffer Manager. In *Proceedings of the Twenty-Third International Conference on Very Large Databases*, Athens, Greece, September 1997.

[2] Infiniband Trade Association, *http://www. infinibandta.org*.

[3] Oracle Corporation. *Oracle9i Real Application Clusters Concepts Release 1 (9.0.1)*, Part Number A89867-01.

[4] Virtual Inteface Architecture, *http://www.viaarch.org.*