A Distributed Lock Manager on Fault Tolerant MPP

Mark Aldred, Ilya Gertner, Stephen McKeffer
Encore Computer Corporation
300 Nickerson Road, Marlborough MA, 01752
(Extended Summary)

Abstract

The Distributed Lock Manager (DLM) is the cornerstone of running the Oracle Parallel Server on a MPP UNIX system. The DLM grants resource locks in NULL, READ, WRITE, and other levels. The DLM is typically implemented either with distributed message passing or in global shared memory, depending on the architecture. Reflective memory on the Encore Infinity 90 architecture permits a shared memory style DLM. Each node runs a separate copy of UNIX so failure of one node is isolated and DLM recovery can clean up state. A node can later rejoin the Oracle session after node repair.

Introduction

The Oracle Parallel Server (OPS) is a unique parallel computing technology that supports both high-availability and multi-computer scalability. OPS maintains a per-node cache of buffers in shared memory, sharing the cache between all appropriate processes running on that node. Such a buffer cache minimizes I/O operations while maximizing performance. Multiple processes can access the information in the cache repeatedly with no internode processing required and no I/O, assuming the cache contains the buffers desired.

Since the buffer cache is directly visible to all processes which wish to reference the cache, arbitrating access to the information in the cache is relatively easy, with a local lock manager using local spinlocks and/or semaphores.

In a clustered environment, however, multiple nodes - each with its own local buffer cache - cannot arbitrate access to the information in the caches in such a painless manner. Oracle’s Parallel Cache Management ensures the consistency and integrity of the various buffer caches in such an environment, while still striving to minimize I/O and maximize performance. In order to provide this cache coherency, the Parallel Cache Management uses a Distributed Lock Manager (DLM) to track the location of blocks of data as they enter and exit the various buffer caches in the cluster. In effect, the DLM extends the semantics of local spinlocks/semaphores into a cluster-wide service available to all Oracle processes.

Features of a DLM

A DLM is a set of library routines that arbitrates locks on resources in a clustered (distributed) environment. When multiple processes on the same or different nodes wish to use the facilities of the DLM to coordinate access to a particular resource, they must each open a DLM lock, supplying the same resource name.

DLM locks that have been opened (initialized) exist in one of several levels (ie NULL, READ, WRITE) and are associated with exactly one resource. The actual resource being locked is not visible to the DLM; the DLM sees only a resource name provided by its caller, such as Oracle Parallel Cache Management.

The processes then attempt to convert the lock to the level desired. If the desired level is compatible with other locks on the resource (for example, any number of processes may simultaneously own locks on the same resource at the READ level), the DLM grants the lock (completes the convert operation).

However, if desired level is not compatible with other DLM locks on the resource, the DLM will notify the process(es) holding those locks on the resource in the blocking levels. Presumably, the blocking process will then lower the level of its lock (using a convert operation) such that the lock is no
longer blocking the pended conversion request. When the pended conversion request is no longer blocked, the DLM will notify the issuing process that the its lock request has been granted.\(^2\)

A DLM must also provide deadlock detection when appropriate. Since deadlock detection can be expensive, the caller can disable the function by specifying the proper flags when opening a lock and converting an open lock to another level.

Associated with each resource (actually part of the resource structure internal to the DLM) is a small buffer called the value block. The value block can be written to and read from by processes holding locks (in appropriate levels) on the resource. This facility allows processes to easily and efficiently coordinate access to protected resources other than blocks of information in a buffer cache.

A DLM must also guarantee consistency of its own data structures (and, by extension, the resources protected by those structures) in the event of a process failure or node failure.

**DLM Implementations**

A DLM implemented on an architecture which has no global shared memory must utilize a messaging paradigm in order to communicate information between the DLM processes on the various nodes in a cluster.

The overhead of an message passing implementation includes

- the processing required for the protocol used
- the latency inherent in the medium used
- the complexity of the deadlock detection algorithm required

In such an implementation, the DLM’s internal database (containing, among other items, DLM lock and resource structures) is either located at a single node or distributed among multiple nodes in the cluster. In the former approach, the node quickly becomes the bottleneck in the system, and is a single point of failure. In the latter approach, the added complexity of a partitioned DLM database - and the processing required to recover and repartition this database after a node crash - make it undesirable as well.

In a global shared memory implementation the DLM’s database exists in global shared memory, immediately and directly accessible by all processes on all nodes in the cluster. The integrity of the data structures residing in global shared memory is ensured using a combination of local and distributed spinlocks.

In addition, the advantages of a global shared memory over an message passing include

- no protocol for inter-node communication; each node sees updates made to the global shared memory from all other nodes
- a transmission medium several orders of magnitude faster than a traditional message passing medium
- a simpler, faster deadlock detection algorithm
- immediate value block reading/writing
- more easily managed recovery from process and node failures
- faster prototyping on new hardware, using local shared memory for initial testing

With global visibility of the DLM’s internal database, processes can directly manipulate DLM structures (including locks and resources), performing operations on locks with minimal inter-node interaction and therefore minimal overhead.

**Reflective Memory**

Encore Computer Corporation’s global shared memory clusters (the Infinity 90) are built on Encore’s Reflective Memory System (RMS) architecture, which provides replicated shared memory implemented in hardware.

Because the entire DLM database exists in RMS, all processes in the cluster can readily obtain the status of all objects in the DLM database, allowing the processes to quickly process lock operations using DLM library calls. This direct interaction among processes eliminates the need for a lock granting daemon (or set of daemons), thereby removing another potential bottleneck.

Maintaining the DLM database in RMS also allows efficient implementation of software to repair the effects of errant processes. Because of the persistent qualities of RMS, DLM daemons can also quickly repair the effects of failed nodes.

---

\(^2\) For more information, refer to "The Oracle Lock Manager", a publication written by Chuck Simmons and Patty Greenwald of Oracle Corporation.
Fault Tolerance

All DLM data structures which need to be referenced by processes on two or more nodes in the cluster are maintained in RMS. Maintaining these structures in RMS allows for easy and efficient implementation of fault tolerance.

Prior to accessing RMS, Oracle processes must register with the DLM. This involves acquiring an entry in the DLM process table, a data structure which resides in RMS and contains the process id and the process log.

```c
/* Process slot entry */
struct {
  uchar_t proc_log[LOGSIZE]; /* process log */
  pid_t pid; /* global pid */
  proc_slot;
}
```

Upon obtaining a process table entry, a process will write its global process id into the process table. This global process id is unique across the cluster. The process log contains records of execution of critical regions of DLM code. These critical regions include operations such as modifying a globally shared linked list or getting a RMS spinlock. For example the LOG macro enters information into the process log:

```c
LOG_AND_GET_SPINLOCK(list_slock);
LOG (MODIFYING_LIST);
add_to_linked_list (list_name, element);
LOG (DONE_MODIFYING_LIST);
LOG_AND_REL_SPINLOCK(list_slock);
```

The combination of spinlocks, logging, and fault recovery allow operations, such as the link list modification shown above, to appear atomic to all processes on the system.

The DLM daemons exists mainly to support fault tolerance. One daemon sends signals to every process registered in the process table. If the signal returns an error indicating that the process no longer exists, then the process log of the crashed processes is scanned. Information concerning the final activities of the process are easily accessible because the process log is maintained in RMS. Any DLM data structures that were being modified by the process at the time of the process crash will be checked for inconsistency and repaired if necessary. All RMS spinlocks held by the crashed process will then be released by the DLM daemons for use by other processes. Once recovery of spinlocks and damaged DLM data structures is complete the recovery of affected DLM lock and resource structures is undertaken.

If one or more nodes in the cluster crash, the DLM daemons will manage this as a series of single process crashes, using the algorithm described above. Due to the persistent qualities of RMS all surviving Oracle processes will maintain uninterrupted access to RMS even during a node crash.

At some point in the future, when the node has been repaired it will rejoin the cluster. The operating system will refresh the RMS on that node with the current copy in the cluster. Oracle processes which are then started on the repaired node will be able to participate in database access as before. If a new node is added to the cluster a similar refresh of RMS will occur. This "persistence of RMS" is one of the greatest assets to programming distributed fault tolerant applications.

An added benefit of storing data structures in RMS is that there is no need to replicate information in software across several nodes to support fault tolerance. The replication is transparently managed by the RMS hardware.

Summary

The library routines in our RMS based DLM contain about 7000 lines of commented C source code. Tools and daemons contain about 6000 lines of commented C source code.

We have used this DLM to run in configurations with >15 nodes (where each node is a 4CPU SMP machine), >200 GB of disk storage, and 64 total MB of RMS, 48 of which contained DLM data structures. Using just such a machine, we have set the world’s record for the tpcB benchmark running Oracle Parallel Server 7.0.15. Given the inherent capability and extensibility of the global shared memory architecture - and Encore’s RMS architecture in particular - we expect to achieve similar breakthroughs in other industry standard and customer benchmarks in the near future.