High Performance Distributed Lock Management Services using Network-based Remote Atomic Operations [*]

S. Narravula A. Mamidala A. Vishnu K. Vaidyanathan D. K. Panda

Department of Computer Science and Engineering
The Ohio State University
{narravul, mamidala, vishnu, vaidyana, panda}@cse.ohio-state.edu

Abstract

Recently there has been a massive increase in computing requirements for parallel applications. These parallel applications and supporting cluster services often need to share system-wide resources. The coordination of these applications is typically managed by a distributed lock manager. The performance of the lock manager is extremely critical for application performance. Researchers have shown that the use of two sided communication protocols, like TCP/IP (used by current generation lock managers), can have significant impact on the scalability of distributed lock managers. In addition, existing one-sided communication based locking designs support locking in exclusive access mode only and can pose significant scalability limitations on applications that need both shared and exclusive access modes like cooperative/file-system caching. Hence the utility of these existing designs in high performance scenarios can be limited. In this paper, we present a novel protocol, for distributed locking services, utilizing the advanced network-level one-sided atomic operations provided by InfiniBand. Our approach augments existing approaches by eliminating the need for two sided communication protocols in the critical locking path. Further, we also demonstrate that our approach provides significantly higher performance in scenarios needing both shared and exclusive mode access to resources. Our experimental results show 39% improvement in basic locking latencies over traditional send/receive based implementations. Further, we also observe a significant (upto 317% for 16 nodes) improvement over existing RDMA based distributed queuing schemes for shared mode locking scenarios.

[*]This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco systems, Linux Networx and Sun Microsystems; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, Dell, Microway, PathScale, IBM, SilverStorm and Sun Microsystems.

1 Introduction

Massive increase in computing requirements have necessitated the use of parallel applications in several fields. Applications in scientific computing, data-mining, web-hosting data-centers, etc. and services like load balancing, cooperative caching, cluster file-systems, etc. supporting the applications often involve multiple parallel coordinating processes accomplishing the required computational tasks. Cluster based architectures are becoming increasingly popular for the deployment of these parallel applications due to their high performance-to-cost ratios. In such architectures, the applications' processes are often distributed across different nodes and efficient coordination of these processes is extremely critical for achieving high performance.

Effective cooperation among the multiple processes distributed across the nodes is needed in a typical data-center environment where common pools of data and resources like files, memory, CPU, etc. are shared across multiple processes. This requirement is even more pronounced for clusters spanning several thousands of nodes. Highly efficient distributed locking services are imperative for such clustered environments.

While traditional locking approaches provide basic mechanisms for this cooperation, high performance, load resiliency and good distribution of lock management workload are key issues that need immediate addressing. Existing approaches [7, 1, 6, 11] handle these requirements either by distributing the per-lock workload (i.e. one server manages all operations for a predefined set of locks) and/or by distributing each individual lock's workload (i.e. a group of servers share the workload by distributing the queue management for the locks). While the former is popularly used to distribute load, it is limited to a high granularity of workload distribution. Further, some locks can have significantly higher workload as compared to others and thereby possibly causing an unbalanced overall load.

The second approach of distributed queue management has been proposed by researchers for load-sharing fairness and better distribution of workload. In such approaches, employing two-sided communication protocols in data-center environments is inefficient as shown by our earlier studies [13]. Devulapalli. et. al. [7], have proposed a distributed queue based locking protocol which avoids two-sided communication operations in the locking critical path. This approach only supports locking of resources in exclusive access mode. However, supporting all popular resource access patterns needs two modes of locking: (i) Exclusive mode locking and (ii) Shared mode locking. Lack of efficient support for shared mode locking precludes the use of these locking services in common high performance data-center scenarios like multiple concurrent readers for a file (in file system caching), or multiple concurrent readers for a data-base table, etc. Hence the distributed lock management needs to be designed taking into account all of these issues.

On the other hand, the emergence of modern generation interconnects have significantly changed the design scope for the cluster based services with the introduction of a range of novel network based features. InfiniBand Architecture (IBA) [3], based on open standards, defines a network that provides high performance (High bandwidth and low latencies). IBA also provides Remote Direct Memory Access (RDMA) which allows processes to access the memory of a process on a remote node without interrupting the remote node's processor. In addition, IBA also defines two network level atomic primitives, *fetch_and_add* and *compare_and_swap*, that allow atomic operations on a 64-bit field on the remote node's memory. Leveraging these novel network features, locking operations can be designed with very low latencies and with minimal CPU overhead on the target node.

In this paper, we propose and design a comprehensive high performance distributed locking service for data-center applications and services in clustered environments over InfiniBand. In particular, our contributions in designing a distributed lock manager are:

1. Providing efficient locking services by using network-based RDMA Atomic operations in the critical path for locking operations
2. Providing efficient support for locking services in both shared and exclusive access modes
3. Designing locking services that have CPU load resilient latencies
4. Designing locking services that try to fairly distribute the workload only among processes involved in locking operations

Our experimental results show 39% improvement in basic locking latencies over traditional send/receive based implementations. Further, we also observe a significant (upto 317% for 16 nodes) improvement over existing RDMA based distributed queuing schemes for shared mode locking scenarios.

Section 2 briefly describes InfiniBand Architecture and cluster based data-centers. Our proposed design is detailed in Section 3. We present our experimental results in Section 4. Related work in the field is summarized in Section 5. Finally, Section 6 presents the conclusions and future work.

2 Background

In this section, we briefly describe the required background in InfiniBand and Advisory Locking services.

2.1 InfiniBand

InfiniBand Architecture (IBA) [3] is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. A typical IBA cluster consists of switched serial links for interconnecting both the processing nodes and the I/O nodes. IBA supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and Memory Semantics (RDMA communication model). Remote Direct Memory Access (RDMA) [9] operations allow processes to access the memory of a remote node process without the remote node CPU intervention. These operations are transparent at the remote end since they do not involve the remote CPU's in the communication.

RDMA Atomic Operations: InfiniBand provides two network level remote atomic operations, namely, *fetch_and_add* and *compare_and_swap*. The network interface card (NIC) on the remote node guarantees the atomicity of these operations. These operations act on 64-bit values. In atomic *Fetch_and_add* operation, the issuing process specifies the value that needs to be added and the remote address of the 64-bit location to which this value is to be added. After the operation, the new value present at this remote address is the original value plus the supplied value. Further, the original value is returned to the issuing process. On the other hand in an atomic *Compare_and_swap* operation, the value at the remote location is atomically compared with the 'compare value' specified by the issuing process. If both the values are equal, the original remote value is swapped with the new value which is also provided by the issuing process. If these values are not the same, swapping does not take place. In both the cases, the original value is returned to the issuing process.

2.2 Advisory Locking Services

Concurrent applications need advisory locking services [6, 11, 10] to coordinate the access to shared resources. The lock manager often deals with only the abstract representations of the resources. The actual resources are usually disjoint from the manager. Each resource is uniquely mapped to a *key*. All locking services are performed using the *keys*. A given lock is usually in one of several possible states: (i) UNLOCKED, (ii) SHARED LOCK and (iii) EXCLUSIVE LOCK. There are several existing approaches providing these locking services. In the following subsections, we describe two relevant distributed lock management (DLM) approaches.

2.2.1 Send/Receive-based Server

The basic communication model used in this design is based on the OpenFabrics-Gen2 [8] two-sided *send-receive* primitives. For all locking operations, the local node sends the requests to the remote node responsible for the key. Based on the availability, the remote node responds. If the lock is unavailable, the remote server node queues the request and responds when possible. The basic advantage of this approach is that an inherent ordering of message is done for each request. And the number of messages required for each lock/unlock operation is fixed. Hence, the approach is free of live-locks and starvation.

2.2.2 Distributed Queue-based Locking

This approach has been proposed by researchers [7], as an attempt to use one-sided communication for distributed locking. In this approach, RDMA Compare-and-Swap is used to create a distributed queue. Each lock has a global 64-bit value representing the tail of the current queue. A new process requiring the lock performs an atomic RDMA CS operation on this 64-bit value assuming it is currently free (i.e. value = 0). If the RDMA CS succeeds then the lock is granted, otherwise the RDMA CS is repeated replacing the current 64-bit value with the new value representing the requesting node's rank. This rank forms the new tail of the distributed queue. It is to be noted that this approach does not support locking in true shared mode. Shared locks can only be granted in exclusive modes and hence are serialized.

3 The Proposed Design

In this section, we describe the various design aspects of our RDMA based complete DLM locking services. Section 3.1 describes the common implementation framework for our system. Section 3.2 describes the design details of our locking designs.

3.1 Basic Framework

The DLM works in a client-server model to provide locking services. In our design we have the DLM server daemons running on all the nodes in the cluster. These daemons coordinate over InfiniBand using OpenFabrics Gen2 interface [8] to provide the required functionality. Figure 1 shows the basic setup on each node. The applications (i.e. clients) contact their local daemons using IPC message queues to make lock/unlock requests. These requests are processed by the local daemons and the response is sent back to the application appropriately. Since typical data-center applications have multiple (often transient) threads and processes running on each node, this approach of having one DLM server daemon on each node provides optimal sharing of DLM resources while providing good performance. These DLM processes are assigned rank ids (starting from one) based on their order of joining the DLM group.

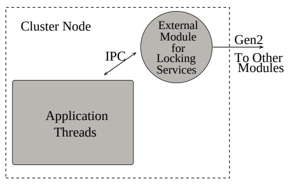


Figure 1. External Module-based Service

The DLM maintains the information on each lock with an associated key. These keys and related lock information is partitioned among the participating nodes; i.e. each key has a *homenode* that represents the default location of the locking state information for that lock (and the keys themselves are randomly distributed among all the nodes).

In order to support these operations, we have three threads in each of our design: (i) Inter-node communication thread, (ii) IPC thread and (ii) Heartbeat thread. The inter-node communication thread blocks on gen2-level receive calls. The IPC thread performs the majority of the work. It receives IPC messages from application processes (lock/unlock requests) and it also receives messages from the other threads as needed. The heartbeat thread is responsible for maintaining the work queues on each node. This thread can also be extended to facilitate deadlock detection and recovery. This issue is orthogonal to our current scope and is not dealt in the current paper.

In our design we use one-sided RDMA atomics in the critical locking path. Further, we distribute the locking workload among the nodes involved in the locking operations. Hence our design maintains basic fairness among the cluster nodes.

3.2 Network-based Combined Shared/Exclusive Distributed Lock Design (N-CoSED)

In this section, we describe the various aspects of our high performance design for providing shared and exclusive locking using network based atomic operations. In particular, we provide the details of the various protocols and data-structures we use in order to accomplish this. This section is organized as the following. First, we explain the organization of the data-structures used in protocols. We then explain the N-CoShED protocol proposed in this paper.

Global Shared Data-Structures: The primary data element used in our proposed DLM design is a 64-bit value. The required attributes of this value is that it should be globally visible and accessible (i.e. RDMA Atomics are enabled on this memory field) by all the participating processes. Each 64-bit value used for lock management is divided equally into two regions: Exclusive region and Shared region, each making up 32-bits. These fields are initalized to zero at the start and the details of the the usage are described in the following subsections.

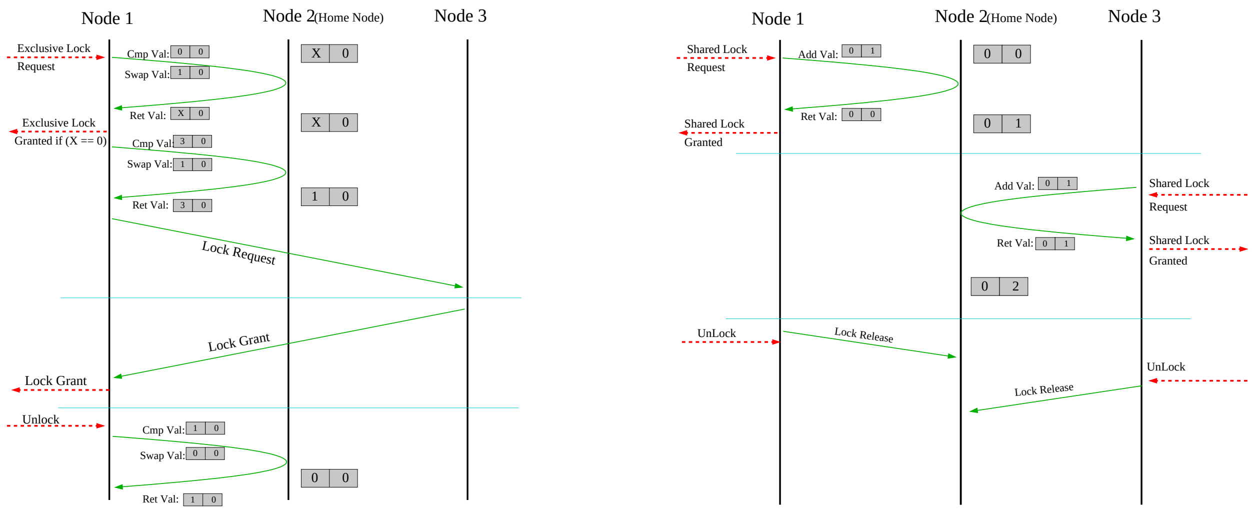


Figure 3. Locking protocols: (a) Exclusive only (b) Shared Only

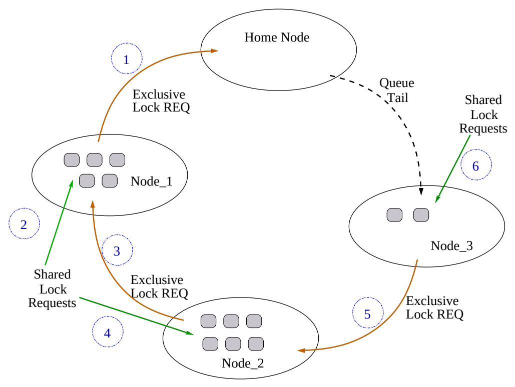


Figure 2. An Example Scenario of N-CoSED

We now explain the combined distributed locking protocol for shared and exclusive locks. To simplify understanding, we break this protocol into four broad cases:(i) Only Exclusive locks are issued, (ii) Only Shared locks are issued, (iii) Exclusive locks are issued following Shared locks and (iv) Shared locks are issued following Exclusive locks.

Figure 2 shows a sample snapshot of the state of the ditributed queue for locks in our design. The circled numbers label the lock request arrows to show the order in which the queue locks are granted. The three nodes shown have exclusive lock requests and each of them have a few shared lock requests queued that will be granted after they are done with the exclusive lock.

3.2.1 Exclusive Locking Protocol

In this section we outline the locking and unlocking procedures when only exclusive locks are issued. As explained above a 64-bit value (on the home node) is used for each lock in the protocol. For exclusive locking, only the first 32 bits of the 64-bit value are used. The following steps detail the exclusive lock/unlock operation. Figure 3(a) shows an example of this case.

Locking Protocol: *Step 1.* To acquire the lock the requesting client process issues an atomic compare-and-swap operation to the home node. In this operation, two values are provided by this process, the swap value and the compare value. The swap value is a 64-bit value whose first 32 bits correspond to the rank of the issuing process and the next 32 bits are zeros $[rank : 0]$. The compare value $[0 : 0]$ is passed for comparison with the value at the home node. If this value equals the value at the home node, the compare operation succeeds and the value at the home node is swapped with the supplied swap value. If the comparison fails then the swapping does not take place. The issuing process is returned with the original 64-bit value of the home node after the atomic operation completes.

Step 2. If the exclusive region of the returned value corresponds to zero, it indicates that no process is currently owning the lock. The process can safely acquire the lock in this circumstance.

Step 3. If the value is not zero, then the exclusive region of the returned value corresponds to the rank of the process at the end of the distributed queue waiting for the lock. In this case, the issued atomic comparison would have failed and the entire atomic operation has to retried. However, this time the exclusive region of the compare value $[current_tail : 0]$ is set to the rank of the last process waiting in the queue. Once the atomic operation succeeds, the local DLM process sends a separate lock request message (using Send/Recv) to the last process waiting for the lock. The rank of this process can be extracted from the 64-bit returned value of the atomic operation. This approach is largely adequate for performance reasons since this operation is not in critical path.

Unlocking Protocol: *Step 1.* After the process finishes up with the lock, it checks whether it has any pending requests received from other processes. If there is a pending lock request, it sends a message to this process indicating that it

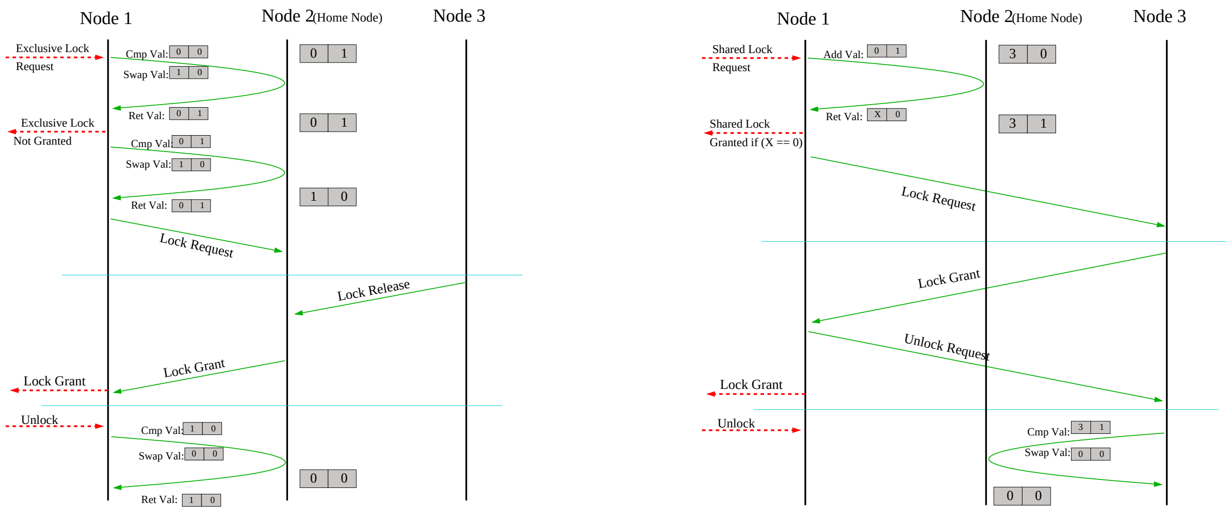


Figure 4. Locking protocols: (a) Shared followed by Exclusive (b) Exclusive followed by Shared

can go ahead and acquire the lock. This process is the next in the distributed queue waiting for the lock.

Step 2. If there are no pending lock requests, the given process is the last in the queue and it resets the 64-bit value at the home-node to zero for both the exclusive and shared regions.

3.2.2 Shared Locking Protocol

In this section we explain the protocol steps when only requests for the shared lock are issued. In this part of the protocol, the shared region portion of the 64-bit value is employed which makes up the last 32 bits. The basic principle employed is that the shared region is atomically incremented using Fetch-and-Add operation every time a shared lock request arrives at the home node. Thus, at any given time the count in the shared region represents the number of shared lock requests arrived at the home node. The following are the detailed steps involved.

Locking Protocol: *Step 1*. The process acquiring the shared lock initiates an atomic fetch-and-add increment operation on the 64-bit value at the home node. Please note that in effect, the operation is performed on the shared region of the value. The first 32 bits are not modified.

Step 2. If the exclusive portion of the returned value corresponds to zero then the shared lock can be safely acquired.

Step 3. If the exclusive portion of the returned value contains a non-zero value, it implies that some other process has issued an exclusive lock request prior to the shared lock request on the lines of the exclusive locking protocol described earlier. We explain this scenario in detail in the following sections.

Unlocking Protocol: *Step 1*. The process after acquiring the shared lock issues a lock release message to the home node.

Step 2. Once all the lock release messages from the shared lock owners have arrived, the shared region is re-set to zero atomically by the home node.

3.2.3 Shared Locking followed by Exclusive locking:

We now outline the steps when an exclusive lock request arrives after the shared locks have been issued. In this case, the value at the home node reads the following. The first 32 bits corresponding to the exclusive portion would be zero followed by the next 32 bits which contain the count of the shared locks issued so far. The process acquiring the exclusive lock issues an atomic compare-and-swap operation on the 64-bit value at the home node as described in the above exclusive protocol section. The following steps occur during the operation. Figure 4(a) shows the basic steps.

Step 1. Similar to the exclusive locking protocol, the issuing client process initiates an atomic compare-and-swap operation with the home node. Since shared locks have been issued, the atomic operation fails for this request. This is because the value in the home node does not match with the compare value supplied which is equal to zero. The atomic operation is retried with the new compare value set to the returned value of the previous operation.

Step 2. Once the retried atomic operation succeeds, the 64-bit value at the home node is swapped with a new value where the shared region is re-set to zero and the exclusive region contains rank of the current issuing process.

Step 3. The issuing process then gets the number of shared locks issued so far from the last 32 bits of the returned value. It also obtains the value of the first 32 bits which is the exclusive region. In our case, since we are assuming that only shared locks have been issued so far this value is zero. It then sends an exclusive lock acquire message to the home node. It also sends the count of the number of shared locks to this process. This count helps the home node keep track of the shared locks issued so far and hence needs to wait for all these unlock messages before forwarding the lock to the node requesting the exclusive lock.

Step 4. The exclusive lock is acquired only when the home node process receives the shared lock release messages from all the outstanding shared lock holders in which case it grants the exclusive lock request.

The case of subsequent exclusive lock requests is the same as described in the exclusive locking protocol section outlined above. Unlock procedures are similar to the earlier cases.

3.2.4 Exclusive Locking followed by Shared Locking:

The following are the sequence of operations when shared locks are issued after exclusive locks. Figure 4(b) shows an example scenario.

Step 1. The issuing client process initiates a fetch-and-add atomic operation in the same fashion described in the locking protocol for shared locks. However, the value of exclusive region in the returned value may not match with the rank of the home process. This is because the exclusive region contains the rank of the last process waiting for exclusive lock in the queue.

Step 2. The shared lock requests are sent to the last process waiting for the exclusive lock. This is obtained from the exclusive portion of the returned value of Fetch-and-Add operation.

Step 3. The shared lock is granted only after the last process waiting for the exclusive lock is finished with the lock.

The same procedure is followed for any shared lock issued after the exclusive locks.

4 Experimental Results

In this section, we present an in-depth experimental evaluation of our Network-based Combined Shared/Exclusive Distributed Lock Management (N-CoSED). We compare our results with existing algorithms (i) Send/Receive-based Centralized Server Locking (SRSL) (Section 2.2.1) and (ii) Distributed Queue-based Non-Shared Locking (DQNL) (Section 2.2.2).

All these designs are implemented over InfiniBand's OpenFabrics-Gen2 interface [8]. Message exchange was implemented over IBA's Send/Receive primitives. The one-sided RDMA operations were used (*compare-and-swap* and *fetch-and-add*) for all the one-sided operations for DQNL and N-CoSED.

Experimental Test Bed: For our experiments we used the a 32-node Intel Xeon cluster. Each node of our testbed has two 3.6 GHz Intel processor and 2 GB main memory. The CPUs support the EM64T technology and run in 64 bit mode. The nodes are equipped with MT25208 HCAs with PCI Express interfaces. A Flextronics 144-port DDR switch is used to connect all the nodes. OFED 1.1.1 software distribution was used.

Table 1. Communication Primitives: Latency

Primitive	Polling (us)	Notification (us)
Send/Recv	4.07	11.18
RDMA CS	5.78	12.97
RDMA FA	5.77	12.96

4.1 Microbenchmarks

The basic latencies observed for each of the InfiniBand's primitives used in our experiments are shown in Table 1 . The latencies of each of these is measured in polling and notification mode. The three primitives shown are *send/recv*, RDMA *compare-and-swap* (RDMA CS) and RDMA *fetch-and-add* (RDMA FA). For the *send/recv* operation we have used a message size of 128 bytes.

As clearly seen from the numbers, the polling approach leads to significantly better latencies. However, the polling-based techniques consume many CPU cycles and hence are not suitable in typical clustered data-center scenarios.

In addition to network based primitives, a DLM needs an intra-node messaging primitive as explained in Section 3.1. In our experiments we use System V IPC message queues. The choice is orthogonal to our current scope of research. We observe a latency of 2.9 microseconds for communicating with IPC message queues for a 64 byte message. The cost for initiating such a request is observed to be 1.1 microseconds. It is to be noted that while the network primitives operate in both polling and notification mode, the intra-node messaging is used only in notification mode. This is because multiple processes that require locking services usually exist on a single node and the situation of having all of these processes polling practically block the node from doing any useful computation and needs to be avoided.

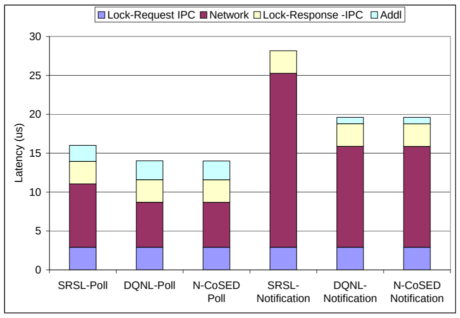
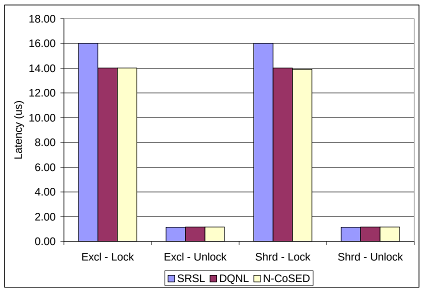


Figure 6. Timing breakup of lock operations

All the locking mechanisms dealing with the distributed locking service daemon, the total lock/unlock latency is divided into two parts: (i) the intra-node messaging latency and (ii) the lock wait + network messaging. While various distributed locking schemes differ significantly in the second component, the first component is usually common to all the different designs and hence can be eliminated for the sake of comparing the performance across different designs.



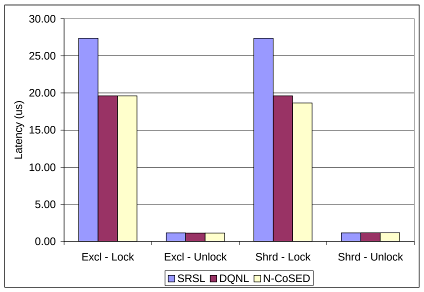


Figure 5. Basic Locking Operations' Latency: (a) Polling (b) Notification

Table 2. Cost Models

Scheme	Lock	Unlock
SRSL	$2 * T_{Send} + 2 * T_{IPC}$	$T_{IPC-Initiate}$
DQNL	$T_{RDMAAtomic} + 2 * T_{IPC}$	$T_{IPC-Initiate}$
N-CoSED	$T_{RDMAAtomic} + 2 * T_{IPC}$	$T_{IPC-Initiate}$

4.2 Detailed Performance Evaluation

In this section, we compare the locking performance of the various lock/unlock operations involved across the three schemes: SRSL, DQNL and N-CoSED.

Figure 5 shows the average latency of the basic locking operations as seen by the applications. In this experiment, we have only one outstanding lock/unlock request (serial locking) for a given resource at any given point of time. Both polling and notification modes are shown.

As seen in Figure 5(a) for polling based latencies, basic locking latency for SRSL is 16.0 microseconds and is identical for shared and exclusive mode locking. This basically includes work related to two Send/Recv messages of IBA, two IPC messages plus book keeping. The DQNL and N-CoSED schemes perform identically for serial locking and show a lower latency of 14.02 microseconds. As shown in Figure 6 the main benefit here is from the fact that two network *send/recv* operations are replaced by one RDMA atomic operation.

Figure 5(b) shows the same latencies in notification mode. The lock latencies for DQNL and N-CoSED show an average latency of 19.6 microseconds whereas SRSL shows a latency of 27.37 microseconds.

The more interesting aspect to note is that in case of polling based approach the SRSL lock latency is 14% more than the RDMA based DQNL and N-CoSED, while in the notification case the SRSL latency is 39.6% higher than the the RDMA based designs. As shown in Figure 6 this higher increase of latency for SRSL is in the network communication part which is due to the fact that it requires notifications for each of the two *send/recv* messages needed for it. On the other hand the RDMA based schemes incur only one notification. Hence the RDMA based schemes offer better basic latencies for locking over two sided schemes.

The basic unlocking latency seen by any process is just about 1.1 microseconds. This is because for unlock operations the applications just initiate the unlock operation by sending the command over messages queues. This actual unlock operation latency is hidden from the process issuing the unlock operation. Table 2 shows the cost models for each of the operations.

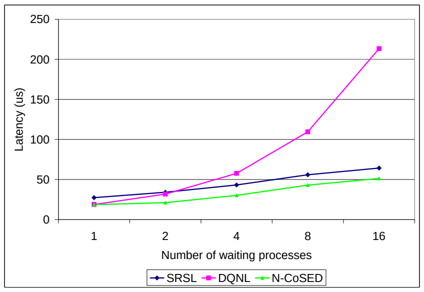
4.3 Cascading Unlock/Lock delay

While the basic unlock latency seen by any unlocking process is minimal, the actual cost of this operation is seen by processes next in line waiting for the lock to be issued. This aspect is equally important since this directly impacts the delay seen by all processes waiting for locks. The two extreme cases of locking scenarios are considered: (i) All processes waiting for exclusive locks and (ii) all waiting processes are waiting for shared locks. In this experiment, a number of processes wait in the queue for a lock currently held in exclusive mode, once the lock is released it is propagated and each of the processes in the queue unlock the resource as soon as they are granted the lock. This test intends to measure the propagation delay of these locks.

In Figure 7(a), the basic latency seen by DQNL increases at a significantly higher rate as compared to SRSL and N-CoSED. This is due to the fact that DQNL is as such incapable of supporting shared locks and grants these in a serialized manner only, whereas the other two schemes release all the shared locks in one go. It is to be noted that all the shared lock holders are immediately releasing the locks in this test. This effect is heightened when the lock holding time of each of the shared lock holders increases. As compared to N-CoSED, DQNL and SRSL incur 317% and 25% higher latencies, respectively. The difference in SRSL and N-CoSED is the extra message SRSL required from the last lock holder to the home-node server before the release can be made.

The increase in the latency for the N-CoSED scheme for longer wait queues is due to the contention at the local NIC for sending out multiple lock release messages using Send/Recv messages. This problem can be addressed by the use of multicast or other optimized collective communication operations [12]. These techniques are orthogonal to the scope of this paper.

Figure 7(b) captures this lock cascading effect by measuring the net exclusive lock latency seen by a set of processes waiting in a queue. The latency for propagation



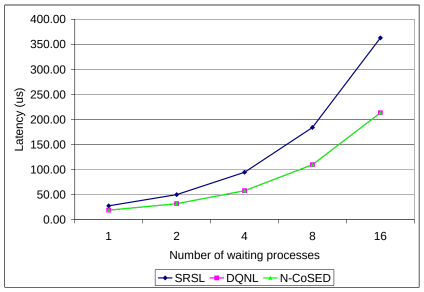


Figure 7. Lock Cascading Effect: (a) Shared Lock Cascade (b) Exclusive Lock Cascade

of exclusive locks is similar for both DQNL and N-CoSED, each of which incurs the cost of one IB Send/Recv message per process in the queue. On the other hand, the SRSL scheme incurs two Send/Recv messages per unlock/lock operation since all the operations have to go the server before they can be forwarded. In all these cases, N-CoSED performs the best.

5 Related Work

Several researchers [1, 6, 11, 5, 4] have designed and evaluated DLMs for providing better parallel application performance. Most of these rely on the traditional two sided communication protocols (like TCP/IP) for all network communication. On the other hand, prior research work [13, 7] have shown the performance benefits and load resiliency that one-sided RDMA based designs have over the traditional designs. In our current design we leverage the benefits of one-sided RDMA to provide efficient lock management services to the applications. DSM synchronizations [2] can be used to provide distributed locking, however, the DSM implemenation itself needs an underlying support of the locking primitives.

Devulapalli et. al. [7], have proposed distributed queue based DLM using RDMA operations. Though this work exploits the benefits of RDMA operations for locking services, their design can only support exclusive mode locking. In our work we provide both shared and exclusive mode locking using RDMA operations.

6 Conclusion and Future Work

The massive increase in cluster based computing requirements has necessitated the used of highly efficient DLMs. In this paper, we have presented a novel distributed locking protocol utilizing the advanced network level one-sided operations provided by InfiniBand. Our approach arguments the existing approaches by eliminating the need for two sided communication protocols in the critical locking path. Further, we have also demonstrated that our approach provides significantly higher performance in scenarios needing both shared and exclusive mode access to resources. Since our design distributes the lock management load largely on some of the nodes using the lock, basic fairness is maintained.

Our experimental results have shown that we can achieve 39% better locking latency as compared to basic *send/recv* based locking schemes. In addition, we have also demonstrated that our design provides excellent shared locking support using RDMA FA. This support is not provided by existing RDMA based approaches. In this regard we have demonstrated the performance of our design which can perform an order of magnitude better than the basic RDMA CS based locking proposed earlier [7].

We plan to extend our designs to include efficient support for starvation free one-sided locking approaches, provide these as a part of the necessary data-center service primitives and demonstrate the overall utility with typical complex applications.

References

[1] M. Aldred, I. Gertner, and S. McKellar. A distributed lock manager on fault tolerant mpp. *hicss*, 00:134, 1995.

[2] Gabriel Antoniu, Luc Bouge, and Lacour Lacour. Making a dsm consistency protocol hierarchy-aware: an efficient synchronization scheme. 2003.

[3] Infiniband Trade Association. http://www.infinibandta.org.

[4] E. Born. Analytical performance modelling of lock management in distributed systems. *Distributed Systems Engineering*, 3(1):68–76, 1996.

[5] Oracle8i Parallel Server Concepts and Administration. http://www.csee.umbc.edu/help/oracle8/server.815/a67778/toc.htm.

[6] Nirmit Desai and Frank Mueller. A log(n) multi-mode locking protocol for distributed systems.

[7] A. Devulapalli and P. Wyckoff. Distributed queue based locking using advanced network features. In *ICPP*, 2005.

[8] Open Fabrics Gen2. http://www.openfabrics.org.

[9] J. Hilland, P. Culley, J. Pinkerton, and R. Recio. RDMA Protocol Verbs Specification (Version 1.0). Technical report, RDMA Consortium, April 2003.

[10] Exclusive File Access in Mac OS X. http://developer.apple.com/technotes/tn/pdf/tn2037.pdf.

[11] H. Kishida and H. Yamazaki. Ssdlm: architecture of a distributed lock manager with high degree of locality for clustered file systems.

[12] Jiuxing Liu, Amith R.Mamidala, and Dhabaleswar K. Panda. Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support. In *Proceedings of IPDPS*, 2004.

[13] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting Strong Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. In *System Area Networks (SAN)*, 2004.