Performance analysis of three implementation strategies for distributed lock management

William J. Knottenbelt * Soraya Zertal † Peter G. Harrison ‡

July 23, 2001

Abstract

A Distributed Lock Manager (DLM) provides distributed applications with a convenient means of synchronising their accesses to shared resources. This paper presents a performance study of three different implementation strategies for a DLM considering both the layout of the lock database (centralised or distributed) and the strategy used to assign lock masters (static or dynamic). For each implementation strategy, we develop accurate analytical models of communication cost, resource utilisation and lock request response time. The models highlight bottlenecks in the system and show clearly for what mixes of incoming lock request types it is best to use static or dynamic lock master positioning. The analytical formulae are validated against a detailed event-driven simulation which uses realistic hardware parameters. This validation reveals a good agreement between analytical and simulation results, particularly with respect to communication cost, node and CPU utilisation, system capacity and the response time trend.

1 Introduction

A distributed application typically consists of several cooperating processes which are spread across a group of loosely coupled computers. Each computer in the group has its own address space, runs its own copy of an operating system and runs a number of the application processes.

Cooperating application processes on different nodes frequently need to share resources such as files, data structures, disk arrays and executable programs. However, accesses to these shared resources often have to be synchronised in order to ensure some form of distributed serialization. A convenient way to achieve this synchronisation is for applications to use the coordination facilities provided by a *distributed lock manager* (DLM). DLMs are integrated into popular commercial applications such as Oracle Parallel Server [Ora00] and are provided by several hardware vendors such as HP, IBM [BEIW97] and Compaq [KLS86, CGS96].

*Department of Computing, Imperial College, London SW7 2BZ, UK, email `wjk@doc.ic.ac.uk`
†PRiSM Lab, Université de Versailles, 78035 Versailles Cedex, France, email `zertal@prism.uvsq.fr`
‡Department of Computing, Imperial College, London SW7 2BZ, UK, email `pgh@doc.ic.ac.uk`

There have been many studies of the performance of distributed lock management and concurrency control schemes. However, these have mainly concerned themselves with one particular DLM architecture or one particular lock application, such as distributed shared memory or distributed and shared-disk databases [ACL87, TR91, Dan92, Rah93].

In [Bor96], Eike Born presented preliminary analytical models for expected lock request response times in a variety of DLM architectures, taking into account the underlying communication network used. A key simplifying assumption was that nodes issue shared and exclusive lock requests only, and hence do not release locks. In addition, no validation of the analytical models was done.

This paper uses Born's work as a starting point but allows nodes to explicitly release shared and exclusive locks. Although this complicates the analysis effort, it results in the development of more realistic performance models for the three most commonly-used DLM implementation schemes. In each case, we derive formulas for resource utilization and average lock request response time. We use Markov models to investigate lock state probabilities, perform a detailed breakdown of communication costs and apply a multiclass M/G/1 queueing model to determine lock response time taking resource contention into account. We also compare the accuracy of our analytical predictions against those of a detailed simulation model.

2 Distributed Lock Management

2.1 Concepts

The aim of a distributed lock manager is to provide facilities which enable cooperating distributed application processes to synchronise access to shared resources. To this end, a DLM is usually realised as a platform-specific software component consisting of two parts, namely a *library of lock management functions* for use by application processes, and a *server program* which executes on each node, either as a daemon process or as part of the operating system kernel.

Application processes use library calls to generate requests for access to shared resources; the requests are then handled by the DLM servers, which eventually grant the request and return an acknowledgement to the requesting process. The requesting process may then proceed to access the shared resource. Note that the presence of a DLM does not in itself ensure proper access to a shared resource. For correct access to be guaranteed, a process wishing to access a shared resource must always apply for access using DLM lock management functions before proceeding.

The DLM servers jointly maintain details of currently held locks in a *lock database*. Each shared resource is associated with a unique name and has an entry in the lock database detailing the set of processes that are currently permitted to access the resource, as well as the access rights held by these processes.

Processes permitted to access a resource are *lock holders* for that resource, and their access rights are described by a *lock mode*. DLMs typically support from two to six lock modes; here we will consider three representative modes: *null*, (no access rights), *shared* (permits concurrent access by several processes) and *exclusive* (gives a single process exclusive access).

A *lock conflict* occurs when a new request's lock mode is incompatible with the mode already held by other lock holders. In this case, existing lock holders are notified and asked to release or downgrade their locks; only when the incompatible locks have been released or converted can the new request be granted. This process is known as *invalidation*.

Any node whose DLM server can grant access permissions for a resource is known as a *lock master* for that resource and is said to be the *resource owner*. If a lock request for a resource arrives at a node which is not a lock master for that resource, the request has to be forwarded to the node that is a lock master before it can be processed. Each node therefore maintains a *lock master directory* in order to store which node is the lock master for each resource.

2.2 DLM implementation schemes

The architecture of a lock manager can be characterised by the implementation of the lock database and the strategy used to assign lock masters.

The lock database can be *centralised* at one server node which will act as lock master for all resources, or the database can be *distributed* by partitioning it among all nodes. Further, positioning of the lock masters can be *static*, in which case all nodes have identical lock master directories; alternatively the positioning of lock masters can be *dynamic*, in which case resource ownership migrates and the lock master is taken to be the last node which acquired the resource in exclusive mode.

These alternatives suggest that four DLM architectures can be considered. Note, however, that in the centralised case all lock master directory entries are fixed and refer to the server node. Resource ownership cannot migrate and consequently only static positioning of lock masters is considered in this case.

To achieve dynamic positioning of lock masters in the distributed case, a communication-efficient "probable master" protocol similar to that proposed in [LH89] for shared virtual memory is used. Here lock master directory entries do not necessarily refer directly to each resource's lock master, but instead describe a search path along which the master can be found. When an incoming lock request for a resource arrives at a node, the node either processes the request if it is the master for the resource, or the node forwards the request and the identity of the requesting node to the node currently registered in its lock directory as the lock master. In the case of an exclusive request, the node also notes the requesting node as the new lock master. In this way a request may be forwarded several times before it arrives at the lock master.

This paper thus considers the performance of three DLM implementation schemes: centralised static, distributed static and distributed dynamic.

3 Performance Model

3.1 Outline

Our aim is to determine the resource utilizations and lock request response times for each of the three different implementation schemes. We begin by considering the hardware and software parameters relevant to a performance model. We then derive some useful intermediate quantities, leading to the determination of $E(C)$, the expected number of communications induced by a lock request.

We use $E(C)$ and the intermediate quantities to compute the load on each resource (CPUs and links), and thus obtain formulae for resource utilization. Next we break up each lock request into its constituent subtasks, and use M/G/1 multiclass queueing models to determine the mean subtask waiting time at each node. Finally, we use the mean subtask waiting time to obtain the overall mean lock request response time.

3.2 Parameters and assumptions

Parameter	Description	Setting	Units
n	number of nodes	8	-
λ	lock request arrival rate	[10 .. 500]	s^{-1}
p_n	probability of a null lock request	[0 .. 1]	-
p_s	probability of a shared lock request	[0 .. 1]	-
p_x	probability of an exclusive lock request	[0 .. 1]	-
v	processor speed	25×10^6	instr/s
c_0	fixed CPU overhead per communication	8×10^3	instr
c_m	CPU overhead per byte of message	0.25	instr
c_l	CPU cost of initial lock processing	200	instr
c_u	CPU cost of updating lock database	300	instr
c_i	CPU cost of invalidations	500	instr
s_m	message length for one communication	10^2	bytes
t_0	fixed network overhead per communication	10^{-3}	s
t_m	network latency per byte of message	5×10^7	s

Table 1: Input parameters describing the hardware and software environment and the corresponding settings used for the case study presented in the results section

We consider a cluster of n ($n \geq 2$) nodes, each of which runs one application process. Each application is assumed to generate lock requests according to a Poisson process with parameter λ requests per second. Arriving lock requests have a random type, such that null lock requests occur with probability p_n, shared lock requests occur with probability p_s and exclusive lock requests occur with probability $p_x = 1 - p_n - p_s$. Initially, each node is a lock master for an equal number of resources.

The nodes each have one CPU which executes at a speed of v instructions per second. Messages have an average size of s_m bytes and incur a fixed CPU overhead per communication of c_0 instructions, as well as a length-dependent overhead of c_m instructions per byte. Processing of lock requests takes place in two phases: initial processing of lock requests requires c_l instructions and final updating of the lock database requires c_u instructions. Invalidations (i.e. the release or downgrading of a lock) require c_i instructions and processes comply with invalidation requests immediately (no blocking in the same way as hashed locks in Oracle Parallel Server).

The nodes are fully connected, with half-duplex network links between every pair of nodes (in a similar manner to a switched ethernet network). Each network link has a fixed latency of t_0 seconds per communication and a latency of t_m seconds per byte of message.

The basic input parameters are summarised in Table 1.

3.3 Calculation of the expected number of communications per lock request $E(C)$

The total number of communications C induced by a lock request can be found by considering the number of communications C_{lm} required to forward the request to the lock master, and the number of communications C_{gr} required for the master to grant the lock request.

The expected number of communications per lock request $E(C)$ can then be calculated as:

$$E(C) = E(C_{lm}) + E(C_{gr})$$

where $E(C_{lm})$ and $E(C_{gr})$ depend on some intermediate quantities that are summarized in Table 2 and which will be derived in the next section.

Quantity	Description
p_{mx}	probability that an exclusive lock holder is also a lock master
π_m	probability that a resource's master is a lock holder
π_x	probability that a resource is locked in exclusive mode
$P(X_m)$	probability that a resource is locked in exclusive mode by its master
$P(X_{\neq m})$	probability that a resource is locked in exclusive mode by a node other than its master
$E(H)$	expected number of lock holders on a resource

Table 2: Useful intermediate quantities to be derived

Table 2 summarises the quantities that need to be derived before $E(C)$ can be calculated.

3.3.1 Some useful intermediate quantities

We first consider p_{mx}, the probability that a node which is an exclusive lock holder for a resource is also a lock master for that resource. For a static lock master scheme, we have $p_{mx} = 1/n$ since lock ownership is fixed and the probability of a random node being the master for a given resource is $1/n$, while for a dynamic lock master scheme we have $p_{mx} = 1$ by definition.

To derive π_m, the probability that the master is a lock holder, we consider a two state discrete-time Markov chain at lock request arrival instants. The states of the chain denote respectively the case where the master is not the lock holder and the case where the master is the lock holder. The corresponding one-step transition probability matrix is:

$$\begin{pmatrix} 1-\alpha & \alpha \\ 1-\beta & \beta \end{pmatrix},$$

where

$$\alpha = p_x p_{mx} + p_s \frac{1}{n}$$

is the probability of moving from a state where the master is not a lock holder at a lock request arrival instant to a state where the master is a lock holder once this lock request has been processed. Effectively, in the case of a shared request, the master that is not a holder only becomes a holder if the shared request arrives at the master node (with probability $\frac{1}{n}$). The behaviour upon an exclusive request arrival depends on whether the DLM scheme is dynamic or static. In the dynamic scheme the master always becomes a lock holder. In the static scheme, the master only becomes a holder if the request arrives at the master node (with probability $\frac{1}{n}$) . The arrival of a null request cannot affect the state of the master if the master is not a lock holder.

In a similar way,

$$\beta \quad = \quad p_s + p_n \frac{n-1}{n} + p_x p_{mx}$$

is the probability of remaining in a state where the master is a lock holder after a lock request arrival instant.

Solving for the steady state distribution gives

$$\pi_m \quad = \quad \frac{p_s + n p_x p_{mx}}{1 + (n-1) p_x}. \tag{1}$$

It will also be useful to calculate π_x, the probability that some node holds an exclusive lock on a given resource. Again we consider a two-state Markov chain where the states denote respectively the case where there is no exclusive lock holder and the case where there is an exclusive lock holder. The transition matrix is given by:

$$\begin{pmatrix} 1 - p_x & p_x \\ \frac{1}{n} p_n + p_s & p_x + \frac{n-1}{n} p_n \end{pmatrix}.$$

Solving the steady state equations and normalising yields:

$$\pi_x \quad = \quad \frac{p_x}{1 - \frac{n-1}{n} p_n}.$$

By combining p_{mx} and π_x, we can also find the probabilities that an exclusive lock is held by the master or by a non-master node:

$$\begin{aligned} P(X_m) &= \pi_x p_{mx}, \\ P(X_{\neq m}) &= \pi_x (1 - p_{mx}). \end{aligned}$$

Since lock requests arrive at a constant intensity according to a Poisson process, the number of lock holders H that hold locks on a particular resource form a stationary Markov chain. The state transition diagram for this chain is presented in Fig. 1. Note that the number of lock holders does not depend on the implementation of the lock manager.

A closed-form analytical solution of the resulting steady state equations is difficult to obtain since it lacks a product-form or other appropriate structure. Fortunately, we can derive the expected number of lock holders $E(H)$, and other useful quantities without resorting to an automated solution of the chain.

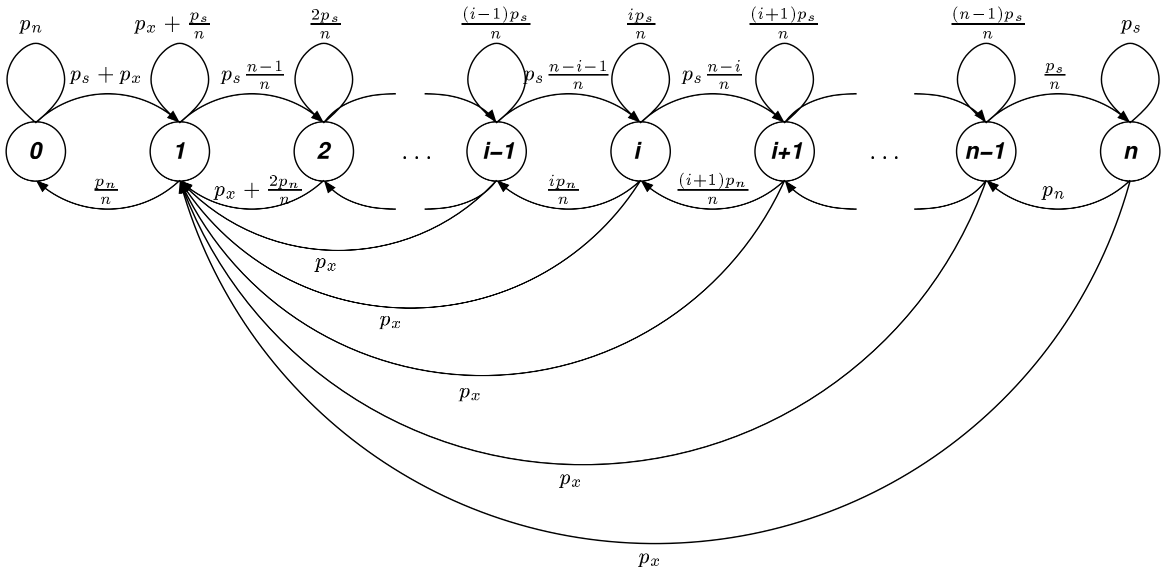

Figure 1: Markov chain of number of lock holders holding a lock on a particular resource

Indeed, in a static scheme, where $p_{mx} = 1/n$, it is clear that the probability for the master to be a lock holder is $\pi_m = \frac{E(H)}{n}$. Further, since the distribution of the number of lock holders is independent of the implementation of the lock manager, we have (from Eq. 1)

$$E(H) = n\pi_m = \left(\frac{p_s + p_x}{1 + (n-1)p_x}\right) n$$

for both static and dynamic schemes.

3.3.2 Calculation of $E(C_{lm})$

In static master schemes, the requesting node is the master with probability $1/n$; in this case no communication is required to forward the request to the master. The requesting node is not the master with probability $\frac{n-1}{n}$; in this case one communication is required to forward the request to the master. Thus the expected number of communications to locate the master in the centralised static and distributed static schemes for one lock request is:

$$E(C_{lm}) = \frac{n-1}{n}. \tag{2}$$

For the distributed dynamic scheme the situation is somewhat more complicated. It is shown in [Kla92] that the expected number of communications to locate the master can be written as:

$$E(C_{lm}) = \sum_{i=2}^{n} \frac{1}{i}. \tag{3}$$

3.3.3 Calculation of $E(C_{gr})$

The number of communications to grant a lock request once the request has been forwarded to the master depends on several factors, viz. the type of lock request, the number of lock holders, the identity of the requesting node, and whether or not the master holds a lock on the resource being requested. Such an analysis requires careful consideration of the set of all possible lock request situations, the probabilities that these situations occur, and their weighted cost.

Different lock request situations have to be considered depending on the type of request; consequently it is convenient to calculate $E(C_{gr})$ as:

$$E(C_{gr}) = p_s E(C_s) + p_x E(C_x) + p_n E(C_n)$$

where $E(C_s)$, $E(C_x)$ and $E(C_n)$ stand for the expected number of communications to grant shared, exclusive and null requests respectively.

State	Description
0	all the nodes own a null lock, there is no lock holder
X_m	the master holds an exclusive lock
$X_{\neq m}$	a node which is not the master holds an exclusive lock
S^i_m	i nodes, including the master, share a lock $(1 \leq i \leq n)$
$S^i_{\neq m}$	i nodes share a lock and the master is not one of those $(1 \leq i \leq n-1)$

Table 3: Notation used to describe individual system states

To calculate these probabilities, we will need to consider the probabilities of individual system states, as given by the number of lock holders and whether or not the master is a lock holder. Accordingly, the notation we will use to describe system states is presented in Table 3. This notation can be used to describe a Markov chain of system states, the transition matrix for which is given in Section A.1 in the appendix. Note that it turns out to be unnecessary to solve this chain for its equilibrium distribution when deriving $E(C_n)$, $E(C_s)$ and $E(C_x)$.

Section A.2 in the appendix describes the range of possible lock request scenarios with their corresponding costs and the probabilities of each scenario for both shared and exclusive lock requests. With straightforward calculations based on the law of total probability, the expected number of communications to grant a lock of each type are:

$$\begin{aligned} E(C_s) &= \left(\tfrac{n-1}{n}\right)\Big(1 + 2P(X_{\neq m})\Big), \\ E(C_x) &= \left(\tfrac{n-1}{n}\right)\Big(1 + 2(E[H] - \pi_m)\Big), \\ E(C_n) &= \tfrac{n-1}{n}. \end{aligned}$$

Using the quantities derived in Sec. 3.3.1 these formulae eventually simplify to:

$$\begin{aligned} E(C_s) &= \left(\tfrac{n-1}{n}\right)\Big[1 + 2\tfrac{p_x}{1-\frac{n-1}{n}p_n}\Big(1 - p_{mx}\Big)\Big], \\ E(C_x) &= \left(\tfrac{n-1}{n}\right)\Big[\tfrac{1+2p_s(n-1)+p_x(3n-1-2np_{mx})}{1+p_x(n-1)}\Big], \\ E(C_n) &= \tfrac{n-1}{n}. \end{aligned}$$

Since $p_{mx} = 1/n$ for static schemes and $p_{mx} = 1$ for dynamic schemes, the formulae above show that it is always cheaper to grant shared and exclusive locks using a dynamic scheme as compared to a static scheme. On the other hand, the cost of locating the master is higher for a dynamic scheme (c.f. Eq. 2 and Eq. 3).

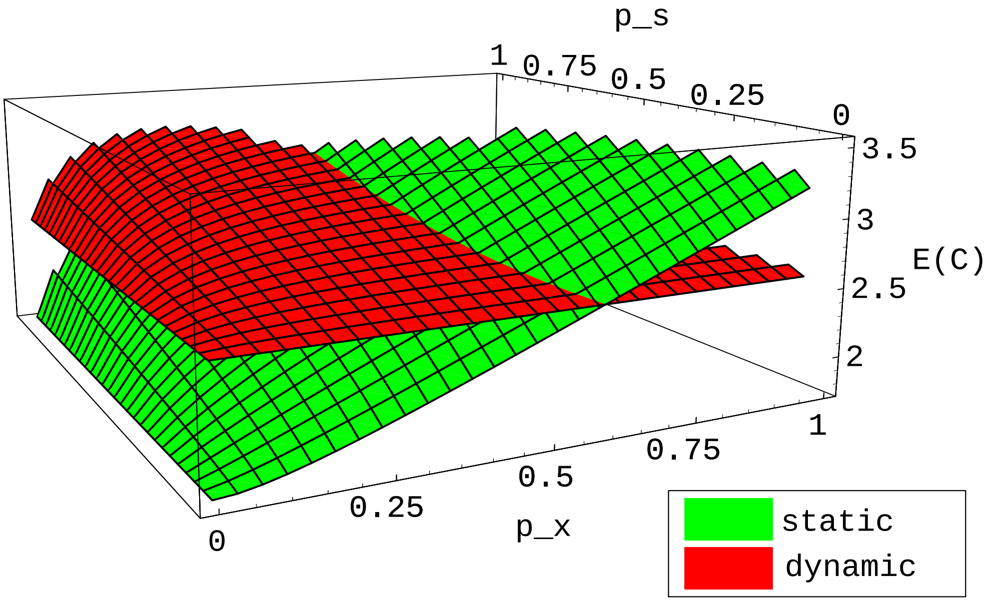

Figure 2: Expected number of communications to grant a lock request E(C) under static and dynamic schemes for various proportions of shared, exclusive and null lock requests.

Fig. 2 compares the overall cost of granting a lock $E(C)$ for an 8 node system with different proportions of exclusive, shared and null lock requests.

The graph consists of two surfaces, representing the cost of granting a lock under static and dynamic strategies respectively. The intersection of the two surfaces yields a threshold contour. For lock request profiles to the left of this contour (i.e. those profiles having a small proportion of exclusive lock requests), the static scheme outperforms the dynamic strategy. As the proportion of exclusive lock requests increases, the relative performance of the dynamic strategy improves, eventually surpassing the static strategy in the area to the right of the contour. These analytical results show which DLM strategy is most appropriate for a given mixture of lock request types and will be confirmed by the simulation results presented in Section 4.

From the expected number of communications to grant a lock one can derive the expected number of *remote invalidations* $E(I_r)$ that will be requested by the master at the clients. Here a remote invalidation is defined as a request from the master to another node to release or downgrade a lock. Since the communication in granting a lock request will involve performing remote invalidations and then possibly sending an acknowledgement to the requester (if it is not the master) with probability $(n-1)/n$, we have

$$E(C_{gr}) = 2E(I_r) + \frac{n-1}{n}.$$

Making $E(I_r)$ the subject of the formula yields

$$E(I_r) = \frac{1}{2}\Big(E(C_{gr}) - \frac{n-1}{n}\Big).$$

In a similar way we can calculate the expected number of *local invalidations* $E(I_l)$ that take place when granting a lock request. Here a local invalidation is defined as a request by the master to downgrade or release a lock that it currently holds. A local invalidation will take place if:

- there is an exclusive request for a resource and the master holds a shared or exclusive lock on that resource, or
- there is a shared request for a resource and the master holds an exclusive lock on that resource.

Thus,

$$E(I_l) = p_x \pi_m + p_s P(X_m)$$

3.4 Resource Utilisations and Lock Request Response Time

3.4.1 Approach

Our approach to determining resource utilisations and lock request response time is as follows:

Parameter	Description	Value
T_{LP}	CPU service time for a lock request	c_l/v
T_{INV}	CPU service time for an invalidation	c_i/v
T_{UP}	CPU service time for an update	c_u/v
T_{NET}	CPU service time for sending or receiving a message	$(c_0 + c_m s_m)/v$
T_{TRANS}	Link service time for message transfer	$t_0 + t_m s_m$

Table 4: Subtask execution cost (in seconds)

1. First, we consider the subtasks that need to be carried out by resources (i.e. CPUs and communication links) in order to execute a lock request. For each type of subtask, we calculate the total arrival rate at each resource and associate a cost of service (c.f. Table 4).
2. Next, we consider the CPUs and links as $M/G/1/\infty/FCFS$ queues where each subtask type corresponds to a class of customer. We calculate CPU and link utilisation by considering the arrival rate of subtasks and subtask execution cost at each resource. Resource utilization is given by
$$\rho = \sum_{i=1}^{S} \lambda_i \bar{x}_i \tag{4}$$
where S is the number of subtasks, λ_i is the arrival rate of subtasks of type i, and $\bar{x}_i$ is the mean service time for a subtask of type i.
3. We calculate the expected waiting time W of a subtask at a CPU or a link using the Pollaczek-Khinchin formula in a multiclass $M/G/1/\infty/FCFS$ queue without priority:
$$W = \frac{\rho \bar{x}}{2(1-\rho)}(1 + C_b^2) \tag{5}$$

where C_b is the coefficient of variation of the service time [Hoc96].

4. Finally, we calculate the average expected response time of each lock request by considering the expected number of subtasks that will be induced by the lock request and the total time subtasks will spend queueing for resources and being served (c.f. Eqs. 6, 7 and 8).

3.4.2 Centralised/Static

According to the previous assumptions, the CPU of each node and each network link can be modelled by a multiclass $M/G/1/\infty/FCFS$ queue. The load on the server and the load at a client are given in the Figs. 3 and 4 respectively.

Rate	Actions	Cost (seconds)
$n\lambda$	Process all lock requests	T_{LP}
$\lambda\Big((n-1)+E(C_{gr})n\Big)$	Accept clients' lock requests and process communications to grant requests	T_{NET}
$n\lambda E(I_l)$	Perform local invalidations	T_{INV}
$n\lambda$	Update lock database	T_{UP}

Figure 3: Load at the server CPU in the Centralised/Static scheme.

Rate	Actions	Cost (seconds)
λ	Process local lock requests	T_{LP}
2λ	Forward requests to the master and receive acknowledgements	T_{NET}
$2\lambda\frac{n}{n-1}E(I_r)$	Receive and acknowledge invalidation requests	T_{NET}
$\lambda\frac{n}{n-1}E(I_r)$	Process invalidation requests	T_{INV}

Figure 4: Load at a client CPU in the Centralised/Static scheme.

The total load on the network is the cost of transferring messages at a rate $n\lambda E(C)$. But as there are $(n-1)$ links between the server and the client, each link receives messages at a rate of $\frac{n}{n-1}\lambda E(C)$. Each message requires a service time of T_{TRANS}.

The corresponding resource utilisations are given by (c.f. Eq. 4):

$$\begin{aligned}
\rho_{SERVER} &= \lambda(nT_{LP} + (n-1+E(C_{gr})n)T_{NET} + nE(I_l)T_{INV} + nT_{UP}) \\
\rho_{CLIENT} &= \lambda(T_{LP} + 2(1+\frac{n}{n-1}E(I_r))T_{NET} + \frac{n}{n-1}E(I_r)T_{INV}) \\
\rho_{LINK} &= \lambda\frac{n}{n-1}E(C)T_{TRANS}
\end{aligned}$$

Using Eq. 5 to determine the mean waiting time at each queue, we can then derive the expected response time for a lock request as:

$$
\begin{aligned}
E(R) \;=\; & E(C)\Big(W_{LINK} + T_{TRANS}\Big) \\
& + E(C)\Big(W_{SERVER} + W_{CLIENT} + 2T_{NET}\Big) \\
& + E(I_r)\Big(W_{CLIENT} + T_{INV}\Big) \\
& + E(I_l)\Big(W_{SERVER} + T_{INV}\Big) \\
& + \Big(2W_{SERVER} + T_{LP} + T_{UP}\Big) \\
& + \Big(\tfrac{n-1}{n}\Big)\Big(W_{CLIENT} + T_{LP}\Big).
\end{aligned}
\tag{6}
$$

where W_{LINK}, W_{CLIENT} and W_{SERVER} represent subtask waiting time at the link, client nodes and the server node respectively. Note that our model assumes that all operations proceed sequentially.

3.4.3 Distributed/Static

To derive the average load at a node we consider what can happen to the node, at what rate and what cost is implied. The results are shown in Fig. 5.

Rate	Actions	Cost (seconds)
λ	Process local lock requests	T_{LP}
$2\lambda\frac{n-1}{n}$	Send lock requests to the master and recieve lock request acknowledgements	T_{NET}
$2\lambda(\frac{n-1}{n})$	Receive lock requests for locks of which the node is the master and send lock request acknowledgements	T_{NET}
$\lambda(\frac{n-1}{n})\lambda$	Process remote lock requests	T_{LP}
$2\lambda E(I_r)$	Send remote invalidations and receive invalidation acknowledgements	T_{NET}
$2\lambda E(I_r)$	Receive remote invalidation and send invalidation acknowledgements	T_{NET}
$2\lambda E(I_l)$	Perform invalidations for locks of which the node is the master	T_{INV}
$\lambda E(I_r)$	Perform invalidations for locks of which the node is not the master	T_{INV}
λ	Update lock database	T_{UP}

Figure 5: Load at a node in the Distributed/Static scheme.

The total load on the communication network is the same as for the Centralised/Static scheme but now it is evenly distributed among the $\frac{n(n-1)}{2}$ links. Therefore the arrival rate at a link is $\frac{2}{n-1}\lambda E(C)$. The service time on each link remains T_{TRANS}.

The corresponding resource utilisations for the nodes and links are:

$$\rho_{NODE} \;=\; \lambda((1 + \frac{n-1}{n})T_{LP} + T_{UP} + (E(I_r) + E(I_l))T_{INV} + 4(\frac{n-1}{n} + E(I_r))T_{NET})$$

$$\rho_{LINK} = \lambda\frac{2}{n-1}E(C)T_{TRANS}$$

In a similar way to the Centralised/Static scheme, we can now derive the expected response time for a lock request using the waiting times derived from the above model:

$$\begin{aligned}E(R) = \; & E(C)\Big(W_{LINK} + T_{TRANS}\Big) \\ & +2E(C)\Big(W_{NODE} + T_{NET}\Big) \\ & +\Big(E(I_r) + E(I_l)\Big)\Big(W_{NODE} + T_{INV}\Big) \\ & +\Big(W_{NODE} + T_{LP}\Big) \\ & +\Big(\tfrac{n-1}{n}\Big)\Big(2W_{NODE} + T_{LP} + T_{UP}\Big).\end{aligned} \tag{7}$$

where W_{NODE} is the time subtasks spend queueing at the CPU of a node.

3.5 Distributed/Dynamic

The Distributed/Dynamic case is different from the Distributed/Static case in two respects. Firstly, communication costs differ as described in Section 3.3 which affects $E(C)$ and hence the link utilisation. Secondly, node CPUs are subject to an additional load source because a node can now be required to forward requests for a lock of which it is not the master. The expected number of such additional forwarding operations for a lock request is given by:

$$E(C_F) = \Big(E(C_{lm}) - \frac{n-1}{n}\Big).$$

The extra load placed on each node's CPU in terms of the M/G/1 multiclass queueing model is given by the time taken to receive and pass on forwarded lock requests, as well as time taken to process the lock requests, i.e.

$$2\lambda E(C_F)T_{NET} + \lambda E(C_F)T_{LP}.$$

The corresponding resource utilisations for the nodes and links are:

$$\begin{aligned}\rho_{NODE} &= \lambda[(1 + \frac{n-1}{n} + E(C_F))T_{LP} + T_{UP} + (E(I_r) + E(I_l))T_{INV} \\ &+ 2(2\frac{n-1}{n} + 2E(I_r) + E(C_F))T_{NET}] \\ \rho_{LINK} &= \lambda\frac{2}{n-1}E(C)T_{TRANS}\end{aligned}$$

The response time for a lock request is given by:

$$\begin{aligned}E(R) = \; & E(C)\Big(W_{LINK} + T_{TRANS}\Big) \\ & +2E(C)\Big(W_{NODE} + T_{NET}\Big) \\ & +\Big(E(I_r) + E(I_l)\Big)\Big(W_{NODE} + T_{INV}\Big) \\ & +\Big(W_{NODE} + T_{LP}\Big)(1 + E(C_F)) \\ & +\Big(\tfrac{n-1}{n}\Big)\Big(2W_{NODE} + T_{LP} + T_{UP}\Big) \\ & +2E(C_F)\Big(W_{NODE} + T_{NET}\Big).\end{aligned} \tag{8}$$

4 Simulation results and discussion

In order to assess the accuracy of the analytical model, we have developed a detailed event-driven simulator. This simulator is written in C++ and is composed of three main parts. The first part is a domain-independent simulation library which contains general simulation routines such as event queue handling and random number generation. The second part is a domain-specific library of resource management functions. It includes control routines for all the simulated system components such as locks, lock database, request queues, links and nodes. The final part is the simulation engine which is responsible for configuring the initial system set up (including node and link topology, lock table initialisation etc.) and for launching the simulation execution. This subdivision enhances the flexibility of the simulator, allowing it to support the three DLM strategies using preprocessor directives which affect only a small proportion of the code. All the resource management functions make use of the realistic hardware parameters described in Table. 1.

Each node in the simulated system accepts a stream of synthetic lock requests which are generated according to a Poisson process. Three different mixes of lock request types are considered, namely a mix with high proportion of exclusive lock requests ($p_n = 0.1$, $p_s = 0.1$, $p_x = 0.8$), a mix with high proportion of shared lock requests ($p_n = 0.1$, $p_s = 0.8$, $p_x = 0.1$) and an intermediate mix ($p_n = 0.2$, $p_s = 0.4$, $p_x = 0.4$).

Simulations are run for a warm up period of 500 000 events to allow the system to reach a stable state. They then run for a further 1 000 000 events during which the statistics concerning CPU and link utilisation, the number of lock communications to grant a lock and the average lock request response time are gathered.

Figures 6 to 15 compare the predictions of the analytical model with statistics generated by the simulator. Figures 6, 8 and 10 compare the expected number of communications per lock request as predicted by the analytical model with the average number of communications observed in the simulation model for the three DLM strategies and for the three different mixtures of lock request types. For lock request arrival rates below maximum system capacity, the predictions of the analytical model are typically within 3% of the observed simulation results. For lock request arrival rates above maximum system capacity, the simulated average number of communications per lock request decreases sharply because the request queues at the nodes contain a large number of unprocessed lock requests at the end of the simulation.

Figures 7, 9 and 11 show the corresponding average response time per lock request as predicted by our analytical model and as observed by simulation. For each combination of DLM strategy and lock request mix, the predicted and the simulated average response times show the same trend and a similar maximum system capacity. For the centralised/static DLM strategy, the response time increases sharply since the central server quickly becomes a bottleneck. For distributed DLM strategies, dynamic lock master positioning yields a lower response time than static positioning when the proportion of exclusive lock requests is high (c.f. Fig. 7). When the proportion of shared requests is high, static lock master positioning gives a lower response time (c.f. Fig. 9). With a similar proportion of shared and exclusive requests, the two distributed DLM strategies yield similar response times (c.f. Fig. 11).

Figures 12, 14 and 16 show the predicted and observed CPU utilisations for the three DLM strategies. In the centralised static case, the utilisation shown is that of the central server's CPU. There is good agreement between the observed and the predicted utilisations. Figures 13, 15 and 17 show corresponding results for link utilisation. We notice again a good agreement between observed and predicted link utilisation for the distributed strategies. In the centralised static case, the utilisation is that of the link attached to the master. For this link, at lock request arrival rates above maximum system capacity, the predicted and observed link utilisation diverge. This can be attributed to the fact that the bottleneck is the central server's CPU and not the link (c.f. Fig. 12, 14 and 16).

It is worthwhile to conclude this section with a short comparison of the simulation and analytical models with respect to development effort and the time required to collect results. It was our experience that the simulation model required more development time, primarily because of the debugging effort necessary to ensure an accurate low-level simulation. As is typical, the analytical model also produced results in several orders of magnitude less CPU time than the simulation – while each simulation run (1 500 000 cycles) required a CPU time of the order of 10 seconds on a 500MHz Pentium PC, the analytical model produced results in around 0.01 seconds.

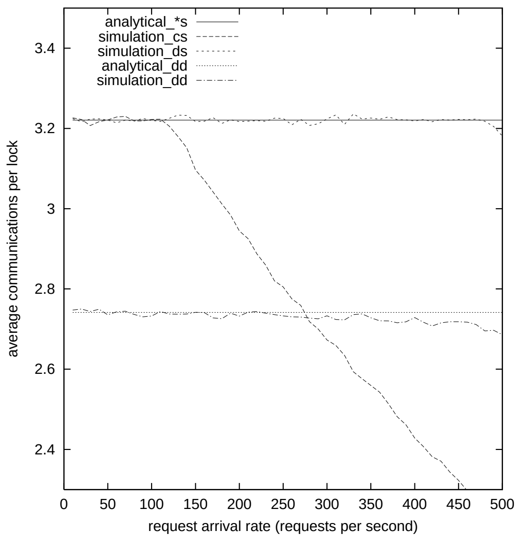

Figure 6: Average communications per lock request for the three DLM strategies with $p_n = 0.1$, $p_s = 0.1$ and $p_x = 0.8$

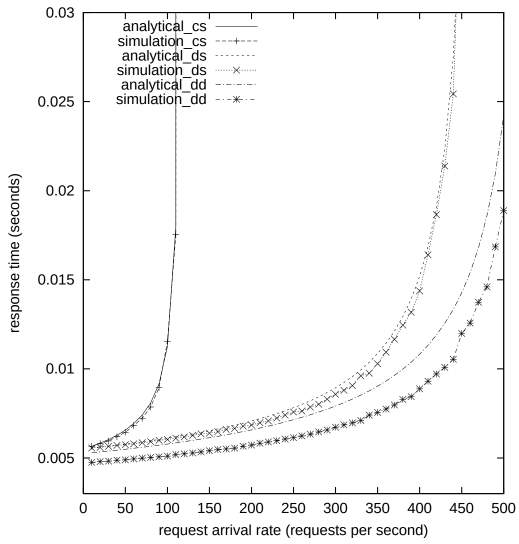

Figure 7: Average lock request response time for the three DLM strategies with $p_n = 0.1$, $p_s = 0.1$ and $p_x = 0.8$

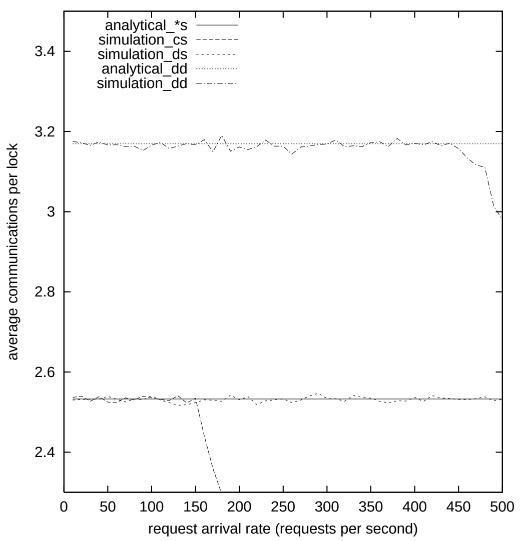

Figure 8: Average communications per lock request for the three DLM strategies with $p_n = 0.1$, $p_s = 0.8$ and $p_x = 0.1$

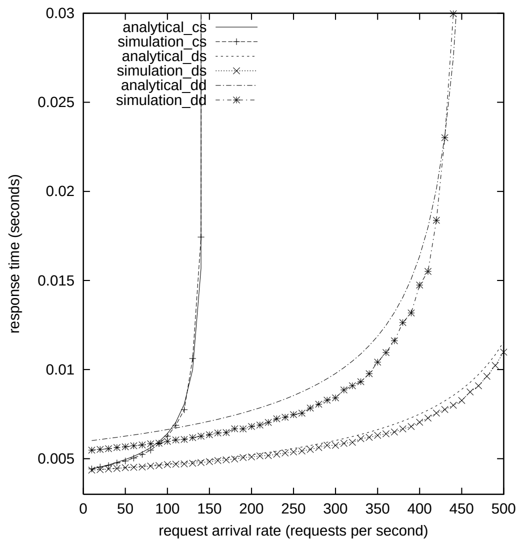

Figure 9: Average lock request response time for the three DLM strategies with $p_n = 0.1$, $p_s = 0.8$ and $p_x = 0.1$

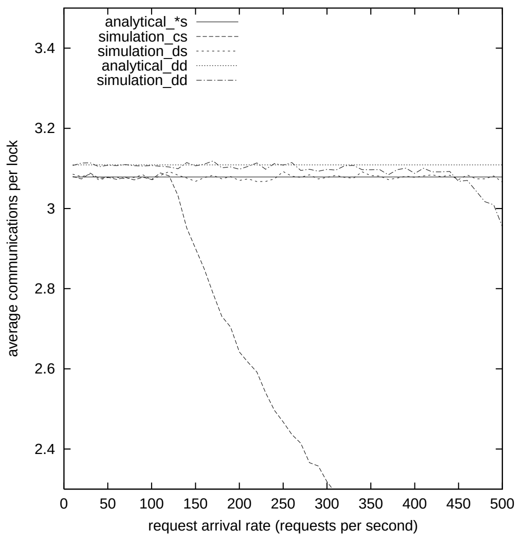

Figure 10: Average communications per lock request for the three DLM strategies with $p_n = 0.2$, $p_s = 0.4$ and $p_x = 0.4$

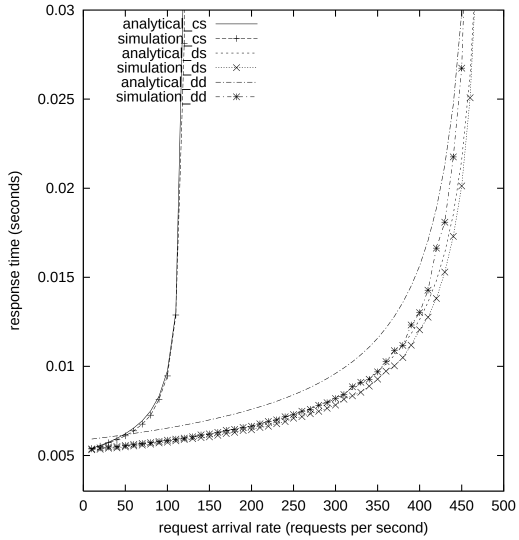

Figure 11: Average lock request response time for the three DLM strategies with $p_n = 0.2$, $p_s = 0.4$ and $p_x = 0.4$

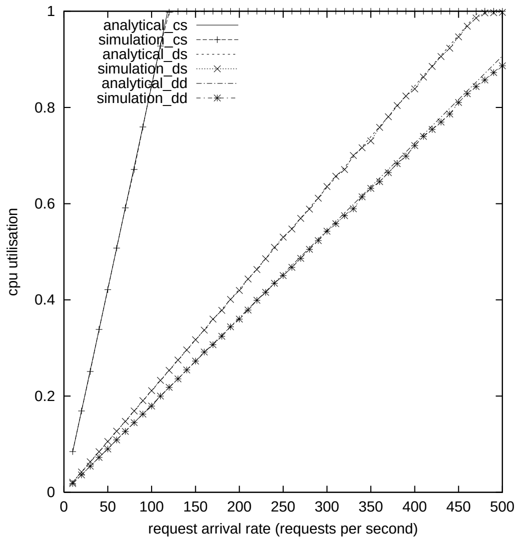

Figure 12: CPU utilisation of the three DLM strategies with $p_n = 0.1$, $p_s = 0.1$ and $p_x = 0.8$

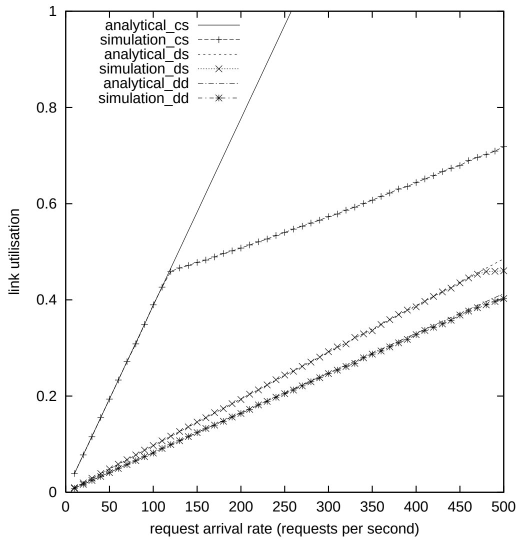

Figure 13: Link utilisation of the three DLM strategies with $p_n = 0.1$, $p_s = 0.1$ and $p_x = 0.8$

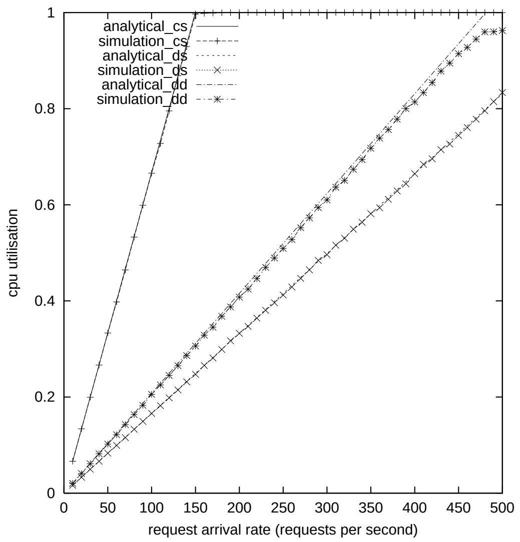

Figure 14: CPU utilisation of the three DLM strategies with $p_n = 0.1$, $p_s = 0.8$ and $p_x = 0.1$

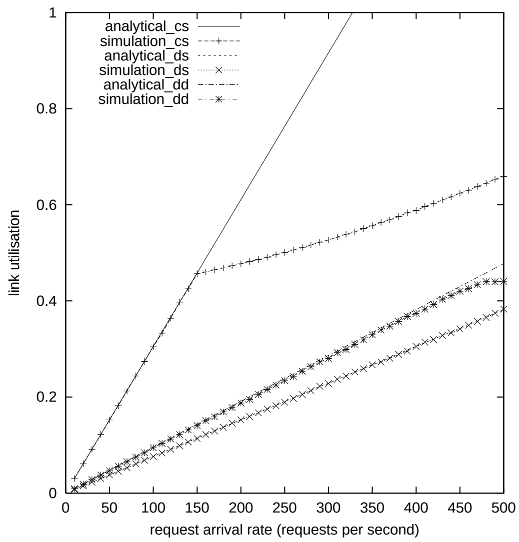

Figure 15: Link utilisation of the three DLM strategies with $p_n = 0.1$, $p_s = 0.8$ and $p_x = 0.1$

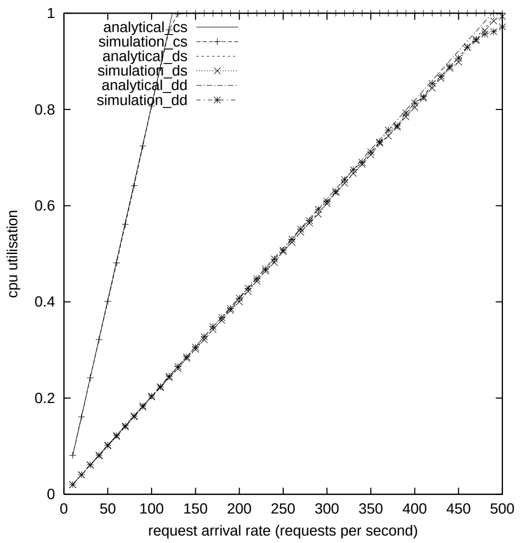

Figure 16: CPU utilisation of the three DLM strategies with $p_n = 0.2$, $p_s = 0.4$ and $p_x = 0.4$

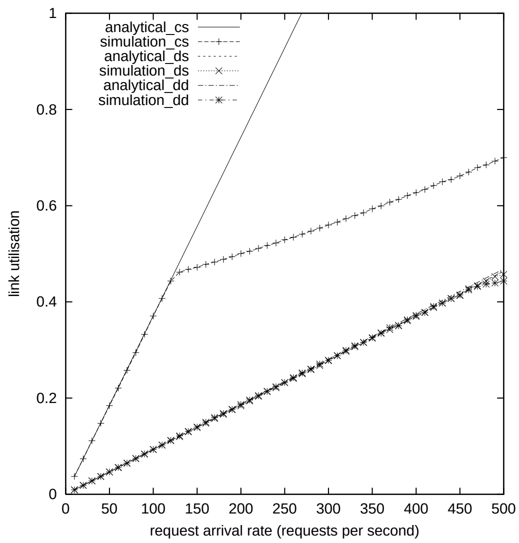

Figure 17: Link utilisation of the three DLM strategies with $p_n = 0.2$, $p_s = 0.4$ and $p_x = 0.4$

5 Conclusion

This paper has investigated the performance of three strategies for distributed lock management, focusing on accurate analytical models for communication load, CPU and link utilisation and average lock request response time. These results have been validated against a detailed simulation model.

The analytical models provide operating system developers and distributed application designers with a rapid way to assess DLM performance across a wide range of hardware and software parameters without the need to develop and debug complex simulators or expensive trial implementations.

In the future, the model could be extended to take into account the locality of access which can sometimes be found in real DLM request traces, to support various caching policies, to include secondary overheads such as deadlock detection recovery and to consider not only average values but also the variance of performance measures.

6 Acknowledgements

The research in this paper was supported by EPSRC grant GR/L06744. The authors would like to express their thanks to Rodolphe Ollivier of École Polytechnique, France for his help in developing an earlier version of the analytical model during his time as an occasional student in the Department of Computing at the Imperial College of Science, Technology and Medicine.

References

[ACL87] Rakesh Agrawal, Michael J Carey, and Miron Livny. Concurrency control performance modelling: Alternatives and implications. *ACM Transactions on Database Systems*, 12(4):609–654, December 1987.

[BEIW97] N.S. Bowen, D.A. Elko, J.F. Isenberg, and G.W. Wang. A locking facility for parallel systems. *IBM Systems Journal*, 36(2), 1997.

[Bor96] Eike Born. Analytical performance modelling of lock management in distributed systems. *Distributed Systems Engineering*, 3:68–76, March 1996.

[CGS96] Wayne M. Cardoza, Frederick S. Glover, and William E. Snaman. Design of the TruCluster multicomputer system for the Digital UNIX environment. *Digital Technical Journal*, 8(1):5–17, May 1996.

[Dan92] Asit Dan. *Performance Analysis of Data Sharing Environments*. MIT Press, Cambridge, Massachusetts, 1992.

[Hoc96] Ng Chee Hock. *Queueing Modelling Fundamentals*. John Wiley and Sons, 1996.

[Kla92] O. Klaassen. The performance of "simple" shared virtual memory. Technical report, Universität Dortmund, 1992.

[KLS86] Nancy P. Kronenberg, Henry M. Levy, and William D. Strecker. VAXclusters: A closely-coupled distributed system. *ACM Transactions on Computer Systems*, 4(2):130–146, May 1986.

[LH89] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, November 1989.

[Ora00] Oracle Corporation. *Oracle8 Parallel Server™ Concepts and Administration, Release 8.0.* Oracle Corporation, 2000. Part No. A58238–01.

[Rah93] Erhard Rahm. Empirical performance evaluation of concurrency and coherency control protocols for database sharing systems. *ACM Transactions on Database Systems*, 18(2):333–377, June 1993.

[TR91] Alexander Thomasian and In Kyung Ryu. Performance analysis of two-phase locking. *IEEE Transactions on Software Engineering*, 17(5):386–402, May 1991.

A Appendix

A.1 Transition matrix of system states

The system can be fully represented by the states listed in Table 3, which can be used to form a Markov chain. To derive the chain's transition matrix we introduce some general probabilities:

p_H is the probability that a node is a lock holder, which is $\frac{H}{n}$ if H is the number of holders.

p_{ms} is the probability for a node to be the master which is always $\frac{1}{n}$.

p_{mx} is the probability that an exclusive lock is held by the master given that there is an exclusive lock holder. In a dynamic scheme this probability is just 1 but in a static scheme it is $\frac{1}{n}$.

p_x, p_s, and p_n are the probabilities that a lock is requested in exclusive, shared or null mode respectively.

We then find the transition matrix shown in Fig. 18.

Current State	$\mathbf{0}$	$\mathbf{X_m}$	$\mathbf{X_{\neq m}}$	$\mathbf{S^1_m}$	$\mathbf{S^1_{\neq m}}$
Next State					
$\mathbf{0}$	p_n	$p_n p_H$	$p_n p_H$	0	0
$\mathbf{X_m}$	$p_x p_{mx}$	$p_n(1-p_H)$ $+$ $p_x p_{mx}$	$p_x p_{mx}$	$p_x p_{mx}$	$p_x p_{mx}$
$\mathbf{X_{\neq m}}$	$p_x(1-p_{mx})$	$p_x(1-p_{mx})$	$p_n(1-p_H)$ $+$ $p_x(1-p_{mx})$	$p_x(1-p_{mx})$	$p_x(1-p_{mx})$
$\mathbf{S^1_m}$	$p_s p_{ms}$	$p_s p_{ms}$	0	$p_n(1-p_H)$ $+$ $p_s p_{ms}$	0
$\mathbf{S^1_{\neq m}}$	$p_s(1-P_{ms})$	0	$p_s p_H$	0	$p_n(1-p_H)$ $+$ $p_s p_H$
$\mathbf{S^2_m}$	0	$p_s(1-p_{ms})$	$p_s p_{ms}$	$p_s(1-p_{ms})$	$p_s p_{ms}$
$\mathbf{S^2_{\neq m}}$	0	0	$p_s(1-p_H-p_{ms})$	0	$p_s(1-p_H-p_{ms})$

$$i \geq 2$$

Current State	$\mathbf{S^i_m}$	$\mathbf{S^i_{\neq m}}$
Next State		
$\mathbf{0}$	0	0
$\mathbf{X_m}$	$p_x p_{mx}$	$p_x p_{mx}$
$\mathbf{X_{\neq m}}$	$p_x(1-p_{mx})$	$p_x(1-p_{mx})$
$\mathbf{S^{(i-1)}_m}$	$p_n(p_H-p_{ms})$	0
$\mathbf{S^{(i-1)}_{\neq m}}$	$p_n p_{ms}$	$p_n p_H$
$\mathbf{S^i_m}$	$p_n(1-p_H)$ $+$ $p_s p_H$	0
$\mathbf{S^i_{\neq m}}$	0	$p_n(1-p_H)$ $+$ $p_s p_H$
$\mathbf{S^{(i+1)}_m}$	$p_s(1-p_H)$	$p_s p_{ms}$
$\mathbf{S^{(i+1)}_{\neq m}}$	0	$p_s(1-p_H-p_{ms})$

Figure 18: The transpose of the one-step transition probability matrix of system states.

A.2 Number of communications to grant a lock

Lock request scenario			C_{gr}	Scenario probability	
$r \in h$	$r = m$		$2(H-1)$	$\frac{1}{n}P(i_m)$	$(i > 0)$
$r \in h$	$r \neq m$	$m \in h$	$2(H-2)+1$	$P(S^i_m)\frac{i-1}{n}$	$(i \geq 2)$
$r \in h$	$r \neq m$	$m \notin h$	$2(H-1)+1$	$P(i_{\neq m})\frac{i}{n}$	$(i > 0)$
$r \notin h$	$r = m$		$2H$	$\frac{1}{n}P(i_{\neq m})$	$(\forall i)$
$r \notin h$	$r \neq m$	$m \in h$	$2(H-1)+1$	$P(i_m)\frac{n-i}{n}$	$(i > 0)$
$r \notin h$	$r \neq m$	$m \notin h$	$2H+1$	$P(i_{\neq m})\frac{n-i-1}{n}$	$(\forall i)$

Figure 19: Relevant lock request scenarios, their probabilities and the corresponding cost of granting an exclusive lock. r=requester; m=master; h=holder set. $P(i_m)$ $\Big($resp. $P(i_{\neq m})\Big)$ stands for the probability that there are i lock holders in exclusive or shared mode and the master is a lock holder (resp. and the master is not a lock holder).

Lock request scenario				C_{gr}	Scenario probability	
$r \in h$	$r = m$			0	$\frac{1}{n}P(i_m)$	$(i > 0)$
$r \in h$	$r \neq m$			1	$\frac{i-1}{n}P(i_m) + \frac{i}{n}P(i_{\neq m})$	$(i > 0)$
$r \notin h$	$r = m$	$Xh = 0$		0	$\frac{1}{n}P(0) + \frac{1}{n}P(i_{\neq m})$	$(i > 0)$
$r \notin h$	$r = m$	$Xh \neq 0$		2	$\frac{1}{n}P(X_{\neq m})$	
$r \notin h$	$r \neq m$	$Xh = 0$		1	$\frac{n-1}{n}P(0) + \frac{n-i}{n}P(S^i_{\neq m}) + \frac{n-i}{n}P(S^i_m)$	$(i > 0)$
$r \notin h$	$r \neq m$	$Xh \neq 0$	$m \in Xh$	1	$P(X_m)\frac{n-1}{n}$	
$r \notin h$	$r \neq m$	$Xh \neq 0$	$m \notin Xh$	3	$P(X_{\neq m})\frac{n-2}{n}$	

Figure 20: Relevant lock request scenarios, their probabilities and the corresponding cost of granting an shared lock. r=requester; m=master; h=holder set; Xh=exclusive holder set.