Analytical performance modelling of lock management in distributed systems

To cite this article: Eike Born 1996 Distrib. Syst. Engng. 3 68

View the article online for updates and enhancements.

Related content
- Capacity planning in client-server systems
  Tim Foxon, Mike Garth and Peter Harrison
- Performance evaluation of communication software systems for distributed computing
  R A Fatoohi
- High-performance asynchronous atomic broadcast
  Flavio Cristian, Shivakant Mishra and Guillermo Alvarez

Recent citations
- Performance Modeling of an EPC Information Service System
  So-Jung Kim et al
- Tariq Omari et al
- S. Narravula et al
Analytical performance modelling of lock management in distributed systems†

Eike Born
Siemens Nixdorf Informationssysteme AG, 81730 München, Germany

Abstract. From a project on upgrading an existing commercially used operating system for support of a distributed system architecture, we present results and experience from a comprehensive analytical performance evaluation of a large number of different implementation strategies for distributed lock management. The various lock management policies we consider differ in the management of the lock database, in optimization concepts applied, and in the corresponding communication protocol. The system architectures discussed include a centralized lock management, a lock management with a partitioned lock database both with immediate and direct communication and with a token ring based communication, and a lock management with a replicated lock database and token ring communication. We derive analytical formulae for the dependence of expected response time of lock requests and their throughput, and provide a practical example.

1. Introduction

Distributed systems consist of several independent computers with their own address spaces and operating systems, nevertheless appearing as a single system for a large number of offered services. To help applications in realizing such a single system image, the support of a distributed serialization function (distributed lock management, DLM) is a strong requirement for the corresponding operating systems. There are different concepts for a DLM, with different characteristics. In this paper, we present results and experiences from a project for identification of the best performing implementation strategies for a DLM, for advising BS2000‡ mainframe operating system development within realization of a distributed operating system.

There have been many investigations of distributed lock management. But most of them consider just a single concept of a DLM and thus do not support comparison of different protocols. Or they are directed towards particular applications of the corresponding locks (cf [7, 22]). Concurrency control mechanisms are considered for instance for two phase locking (cf [18]), and in connection with distributed shared memory architectures (cf [4, 17]), with distributed and shared data bases (cf [1, 3, 6, 15, 23]) and with transaction processing (cf [5, 10, 21]). Finally, an approach is lacking combining the impact of the lock protocol and the communication protocol on the overall performance. Thus the existing results are insufficient to give operating system developers a clear understanding of which DLM protocol fits best for the distributed management of locks.

In this paper we report on a performance analysis of implementation strategies for a DLM, differing in the management of the lock database, optimization concepts thereof, and the communication protocol used. The main concepts are a centralized lock management, a lock management with a partitioned lock database with different communication methods, and one with a replicated lock database and a token ring communication. We derive analytical formulae for the expected response time of a lock request and for the limiting throughput. In contrast to simulations, these models have been developed and can be evaluated in restricted time, and allow for a fast comparison of very different implementation strategies. They are thus of considerable help to operating system developers.

2. Hardware and software environment

We consider a system with a moderate number, \(n\), of processing nodes. The nodes are connected by a network allowing for direct node to node communication. On each node, a local operating system supports basic functions including communication. Every communication requires a message length independent execution of \(c_0\) instructions, and execution of \(c_m\) instructions per Byte transferred, including all communication controls on levels below the application. Network transfer induces a message length and processing power independent latency time \(t_0\), and a message length dependent latency \(t_m\), arising from the constraints of the network. As locally shared lock
management induces additional losses, we assume in what follows that at most one processor of each node may perform lock management processing. Communication processing is done either by this processor ($n_s = 1$) or by another one ($n_s = 2$).

3. Implementation concepts of distributed lock management

3.1. Distributed lock management

A lock is a mechanism for serializing access to data objects, offered for voluntary use to applications by a lock management system. The lock is realized as a piece of data with a unique identifier, and assumes a certain lock state from a predefined set of possible lock modes. Accessibility of data protected by a lock is managed via a compatibility matrix, giving for every current state of a lock the admissible access modes of the data. We say that a node has a lock in a mode $M$, if it has the access rights for the data corresponding to the lock in mode $M$. This node then is called an ($M$-mode) lock holder. Any node which may grant access rights for data protected by a lock, is called a (lock) master for this lock.

Simple lock modes are locked or unlocked, but applications often require more specific lock modes. We restrict our attention to the modes unlocked, as long as no request for the lock is granted, shared, allowing for concurrent access to the protected data, and exclusive, giving an exclusive access right to the data.

The master of a lock establishes a lock database to manage requests for the lock in a consistent way. Main parameters of lock processing are the average number $c_l$ of CPU-instructions for processing either a lock request and grant or a release, and the average size $s_m$ of one message for communication with the lock manager.

For distributed lock management, the lock database must be available to the lock management modules in all participating nodes. Thus a communication protocol must support DLM processing in a well tailored way.

3.2. Basic DLM implementation strategies under consideration

In order to cover a large variety of implementation strategies for a DLM, we consider different alternatives for the implementation of the lock database, different optimization strategies of the lock protocol and different communication protocols.

The following lock database implementation strategies are investigated:

- centralized lock database at one server node (abbreviation C),
- partitioned lock database, distributed among all nodes (abbreviation P),
- replicated lock database, existing virtually identically on all nodes (abbreviation R).

The realization of a centralized lock database is well understood. We include it as a reference implementation. For the two other approaches there are existing examples in commercially used systems. The DLM of DEC’s VAX Cluster uses a partitioned lock database (cf [9], chapter 13 and [14]), and the resource manager of IBM’s SYSPLEX uses a replicated lock database (cf [11] and [16]).

Optimizations of the lock management protocol concern the positioning of the lock masters in the case of a partitioned lock database, the granting of access rights, and the return of access rights (cf Table 1).

<table>
<thead>
<tr>
<th>Return of locks</th>
<th>Master location:</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediately</td>
<td>static</td>
</tr>
<tr>
<td>on demand</td>
<td>is</td>
</tr>
<tr>
<td>on demand with on demand</td>
<td>ds</td>
</tr>
<tr>
<td>on demand with grant optimization</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization concepts and their abbreviations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positioning of a lock master can be predefined and static; in which case directories in all nodes show the exact localization of the lock masters. Alternatively, the node first requesting a lock becomes the master and afterwards lock ownership moves in a dynamic way to that node which acquires the lock in an exclusive mode. In this case updating of the local lock masters directories is done by a probable master consistency protocol according to [13], which reduces communication overhead. The location of the initial master of a lock is communicated to all nodes. Afterwards, upon arrival of a lock request, a node either manages the request if it is the master, or it forwards the request and the identification of the requesting node to the node that is registered in its directory as the master, and in the case of an exclusive request records the requesting node as the new master in its own directory. Thus the directories only give information on a search path to the master.</td>
</tr>
<tr>
<td>Irrespective of optimization, a shared or exclusive lock request from an application on a node which already holds the lock in a compatible mode will be granted without any communication, even if the node is not the lock master (local caching). Within grant optimization, any node with a shared or exclusive lock mode, touched within the search for the master, may grant admissible access rights also to other nodes. Obviously this applies to dynamic masters location and return on demand strategies only (see below).</td>
</tr>
</tbody>
</table>
| Finally, granted locks may be returned to the lock master immediately after usage, or on demand only. In the second case, return requests and corresponding acknowledgments are necessary upon a lock request between the lock master and every remote lock holder before granting the lock to the requesting node. We assume in addition that in the case of only one remote lock holder,
it gives the lock to the requesting node directly, only informing the master.

For communication within the DLM, we consider three communication methods:

- immediate communication of a lock request to the (probable) master (abbreviation I),
- timer controlled remote node specific collection of communication requests and direct routing to the addressed node, $t_d$ being the delay between consecutive communications (abbreviation T),
- token ring communication with $n_t$ tokens and sojourn time $t_s$ at every node (abbreviation R).

Within the implemented performance models, all combinations of the above alternatives are available. Nevertheless some combinations are less important. For instance for a centralized lock database, only static location of lock masters and no token ring communication makes sense, while for the replicated lock database, the token ring communication is required (see section 3.4). Under consideration are primarily the strategies CI.xs, PL.x, PT.x, PR.x and RR.ds, where 'x' stands for the optimizations of table 1. In what follows, we give more specific descriptions for some basic strategies and selected combinations.

3.3. Partitioned lock database strategies (PL.x, PT.x, PR.x)

The lock protocol runs according to the chosen optimization (cf table 1). Location of lock masters is either fixed (PX.xs), or starts from the first requesting node and then dynamically moves to a node upon being granted an exclusive lock (PX.xd). A lock request from an application at a node is granted at once, if either the lock is granted to the node in a compatible mode, or if the node is the master and the lock has not been granted in incompatible mode to other nodes. If it has been granted to other nodes in an incompatible mode, the lock request can only be granted after all hand-outs have been returned, which in the case of a return on demand policy requires informations sending to all incompatible lock mode holders (observe further optimization in the case of only one remote lock holder). In all other cases the node forwards the request to the actual (PX.xs) or the probable (PX.xd) master from its masters directory, and introduces itself as the new master in its master directory, if it is an exclusive lock request and a probable master concept is applied. If the addressed node is the master, this node proceeds as above. If the addressed node is not the master but has the lock in a compatible mode and a grant optimization is applied, the request can as well be granted. Otherwise the request is again forwarded to the next probable master. In case of an exclusive request, the requesting node then is introduced as the new lock master in the masters directory and a corresponding notification is forwarded together with the lock request. All necessary communications are sent immediately (PL.x) or timer controlled (PT.x), which requires no further explanation, or via a token ring communication (PR.x).

For token ring based communication, each of the $n_t$ tokens can transport every lock request. Every token sojourns at every node in each of its cycles for a fixed time $t_s$ to avoid thrashing in the case of low load. Tokens are processed at a node one by one. If a token arrives at a node, all notifications concerning this node are taken from the token, all messages to be forwarded are appended to the token and after $t_s$ seconds the token is forwarded to the next node.

3.4. Replicated lock database with token ring communication (RR.ds)

For the strategy RR.ds, lock protocol and communication protocol interfere in a very specific way. Every node has a replica of the complete lock database. Consistency of the replicas is ensured by restricting local access to the database only if a lock specific rotating token is present. In contrast to strategy PR.x, the token now contains notifications of all lock database modifications of all nodes. If there are several tokens, they are responsible for disjunct subsets of locks. Parameters are again the number $n_t$ of tokens and the (absolute) sojourn time $t_s$ the token rests at every node. The latter now must be used for token construction and lock processing.

In the case of a lock request of a node, it waits until the lock specific token arrives. First all old messages of this node from the last token visit are removed. Then all notifications of other nodes are integrated into the local lock database. Now the node has the actual view of the database for the requested lock, and can decide whether to grant this lock at once, or to queue the request and to call back all hand-outs of the lock. For every such action, a notification is appended to the token. Since every node sees the requests queue in an actual state, there can be no overrun of a former lock request by a later one and the request under consideration will be granted eventually. Nevertheless, several rotations of the token may be necessary for completing a lock request, according to the limited sojourn time $t_s$ of the token.

4. Performance models

4.1. Assumptions on the load

Since the input stream of lock requests is determined by many, mostly independent causes, a Poisson process arrival of these lock requests is a well justified assumption. Nevertheless within a real system, the arrival rate of such Poisson input will depend on the node, on every specific lock and on the requested lock mode. In order to avoid modelling such a complex multi-compound Poisson arrival process, we apply additional assumptions.

Lock conflict rates depend primarily on the application design and do not give any information for an assessment of DLM strategies. Thus we assume that there are no lock conflicts. Accordingly we need not consider lock specific request rates but may assume a common arrival rate of lock requests. Next, one cannot assume a particular structure within the nodes. Thus we assume that lock requests arrive uniformly distributed among the nodes. To avoid complicated refinements of lock protocols, we finally
restrict our attention to the lock modes unlocked, shared and exclusive (cf section 3.1).

Thus, overall, lock requests arrive as a Poisson process with a node local lock request arrival rate \( \lambda \) and a probability \( p_x \) that a request will be for exclusive mode. A request is initiated at a particular node with probability \( 1/n \). For the strategies with a partitioned lock database for instance, we thus may neglect the start up period for initializing the lock master but assume that the initial lock masters are located uniformly distributed among the nodes.

### 4.2. The basic queueing network model

The most important parameter for assessing the performance of a DLM implementation is the expected response time \( E(R) \) of a lock request for a given lock request arrival rate \( \lambda \). It can be expressed as

\[
E(R) = E(R_{lock\, processing}) \\
+ E(C)[E(R_{local\, communication}) \\
+ E(R_{network\, latency}) \\
+ E(R_{remote\, communication})],
\]

with the obvious meaning of the particular response times, and \( E(C) \) denoting the expected number of communications induced by one lock request and contributing to its response time.

Because of the stochastic nature of the underlying dynamics, the response times from equation (1) are expected response times of particular queueing stations in a queueing network. This queueing network is shown in figure 1. In steady state, locally initiated remote processing equals the amount of remotely induced local processing. Hence the lock processing server in this network may serve for local and remote lock processing. The queueing network thus captures all lock management and communication strategies under consideration.

Lock and communication processing require fixed rather than exponentially distributed numbers of instructions. Thus both servers are assumed to be FCFS-servers with constant service times. Both for separate servers or a common server for lock processing and communication, the expected response time for these services and the limiting throughput can be approximated by Pollaczek–Khintchine’s formula and Little’s law using \( \lambda \) and \( E(C) \) (cf [19]). The derivation of \( E(C) \) is given in section 4.3.

For the non-token ring based strategies, message transfer time in the network depends not only on the size of the message, but on the overall network utilization \( U_C \). Hence we assume network transfer response time to consist of a constant part \( t_0 \) and response time in a processor sharing queue with arrival rate \( E(U_C) \) and constant service requests of size \( l_m s_m \), (respectively \( l_m E(S_m) \)) for strategies with varying message size \( S_m \). For the token ring

---

**Table 2.** Service requests arrival rate (upper value) and expected service time (lower value) for direct communication based strategies (’x’ stands for is, ds, dd, or gd; \( E(S_m) = \frac{1}{n} t_d E(C) s_m \)).

<table>
<thead>
<tr>
<th>Implementation strategies</th>
<th>Cl.x</th>
<th>Pl.x</th>
<th>PT.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local communication</td>
<td>( \lambda E(C) )</td>
<td>( \lambda E(C) )</td>
<td>( 2(n-1)/t_d )</td>
</tr>
<tr>
<td></td>
<td>( (\eta_0 + c_m S_m)/V ) at the client</td>
<td>( (\eta_0 + c_m S_m)/V )</td>
<td>+ constant delay ( t_d/2 )</td>
</tr>
<tr>
<td>Network transfer per link</td>
<td>( E(U_C) = \lambda E(C) s_m )</td>
<td>( E(U_C) = 2 \frac{1}{n-\eta} E(C) s_m )</td>
<td>( 2/t_d )</td>
</tr>
<tr>
<td></td>
<td>( t_m s_m ) (only links to the server)</td>
<td>( t_m s_m )</td>
<td>+ constant delay ( t_d/2 )</td>
</tr>
<tr>
<td>Remote communication</td>
<td>( (n-1) \lambda E(C) )</td>
<td>( \lambda_0 E(C) )</td>
<td>( 2(n-1)/t_d )</td>
</tr>
<tr>
<td></td>
<td>( (\eta_0 + c_m S_m)/V ) at the server</td>
<td>( (\eta_0 + c_m S_m)/V )</td>
<td>+ constant delay ( t_d/2 )</td>
</tr>
<tr>
<td>Lock processing</td>
<td>( 2 \lambda ) (server)</td>
<td>( 2 \lambda )</td>
<td>( 2 \lambda )</td>
</tr>
<tr>
<td></td>
<td>( c_i/V )</td>
<td>( c_i/V )</td>
<td>( c_i/V )</td>
</tr>
</tbody>
</table>
communication based strategies, a different approach is chosen for calculation of communication delays (cf sections 4.4 and 4.5). The overall arrival rates and expected service times for the non-token ring based implementation strategies can be found without sophisticated methods and are shown in table 2.

4.3. Expected number \(E(C)\) of communications induced by one lock request

The number \(C\) of communications induced by one lock request depends on the number \(C_{ml}\) of communications necessary to forward a lock request to a node which can grant the lock (master localization), and the number \(C_{gr}\) of communications necessary to handle the grant of the lock. Both depend on the actual number \(H\) of lock holders, which may be larger than 1 for the read optimization strategy.

4.3.1. Distribution of the number \(H\) of lock holders

Let \(H_t\) denote the number of holders of a fixed lock at time \(t \in \mathbb{R}_+\). We assume that requests for a lock arrive with a constant intensity. Thus \(H = (H_t)_{t \in \mathbb{R}_+}\) is a stationary Markov process. The embedded chain \(H^c := (H^c_t)_{t \in \mathbb{N}}\) of the number \(H^c_t\) of lock holders after the \(k\)th lock request is a stationary Markov chain (cf [20], p 204). \(H^c\) enjoys the same stationary distribution as \(H\) ([20], p 211). Hence we are left with finding the stationary distribution of \(H^c\). It can be obtained from a usual balance equation analysis for the transition diagram, given in figure 2. Since the first reference to a lock and lock destruction does not contribute to the stationary distribution, the initial state of zero lock holders is neglected. With straightforward calculations, one obtains the steady state distribution as

\[
P(H = h) = p_x(1 - p_x)^{n-1} \prod_{j=1}^{h} \frac{n - j}{n - j(1 - p_x)}
\]

\((1 \leq h \leq n)\).

4.3.2. Communications \(C_{ml}\) for master location

For all static master distribution concepts, we have \(E(C_{ml}) = \frac{(n-1)}{n}\), since a lock request arrives at a master with probability \(1/n\) requiring no communications, and arrives at a non-master node with probability \((n-1)/n\), requiring 1 communication.

It is shown in [12] that for dynamic master distribution and a single master (no read optimization), we have

\[
E(C_{ml}|H = 1) = \sum_{i=3}^{n} \frac{1}{i}
\]

while for a read optimization concept, we may approximate

\[
E(C_{ml}|H = h) \approx \sum_{i=h+1}^{n} \frac{1}{i} P(H = i)
\]

\[
= \sum_{i=h+1}^{n} \frac{1}{i} p_x(1 - p_x)^{n-1} \prod_{j=1}^{i} \frac{n - j + 1}{n - j(1 - p_x)}
\]

\((\text{for } h = 2, 3, \ldots)\) (4)

The first equation of (4) is shown in [12] to hold exactly for \(h \leq 16\) and by simulation to hold approximately for \(h \leq 256\), while the second equality is exact according to the results of section 4.3.1.

4.3.3. Communications \(C_{gr}\) for granting a lock

In the case of immediate return of a lock, we have \(E(C_{gr}) = 0 \cdot \frac{1}{n} + 1 \cdot \frac{1}{n} = (n-1)/n\) by the same reasoning as before. For return on demand concepts, lock return requests have to be sent and returned. Their number depends on whether the lock request is of shared or exclusive mode, is initiated from a lock holder or a lock master or none of these, and whether the lock master is also a holder. The probabilities of these situations and the corresponding number \(C_{gr}\) of communications for granting a lock can be calculated very easily and are shown in table 3.

4.3.4. Overall communications

Now the overall number of communications induced by a lock request can be expressed as

\[
E(C) = \sum_{h=1}^{n} E(C_{ml}|H = h) P(H = h)
\]

\[
+ \sum_{s} E(C_{gr}|S = s) P(S = s)
\]

(5)

where the first sum extends over the distribution of the number \(H\) of lock holders, while the second sum extends over the lock request situations from table 3. For all but the immediate return strategies, this is as well the expected number \(E(C_{load})\) of communications, defining the communication load. For immediate return of access grants and a static masters location, we have \(E(C_{load}) = E(C) + 1 \cdot \frac{1}{n}\). Immediate return of locks and dynamic master distribution is not a sensible combination.

4.4. Partitioned lock database with token ring communication (PR)

The expected cycle time \(E(T_{cycle})\) of every token consists of \(n\) sojourn times \(t_s\) and \(n\) passage times. Every passage time consists of a constant network latency \(t_0\), a message size
Table 3. Lock request situations, their probabilities and the number \( C_{gr} \) of communications to grant a lock for return on demand strategies. \( sh = \text{shared}, \ ex = \text{exclusive}, 'a \Rightarrow b' \text{ means 'a implies b'}; \ r = \text{requester}, \ h = \text{holder set}, \ m = \text{master}; \ '⋆' \text{ means 'irrelevant'}.

<table>
<thead>
<tr>
<th>Lock request situation s</th>
<th>Probability ( P(S = s) ) of situation s</th>
<th>Number of communications ( C_{gr} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( sh ) ( r = h ) ( r = m ) ( m = h )</td>
<td>( (1 - \lambda_n)E(H)/n )</td>
<td>0</td>
</tr>
<tr>
<td>( sh ) ( r \neq h ) ( r = m ) ( m \neq h )</td>
<td>( (1 - \lambda_n)(n - E(H))/n^2 )</td>
<td>2</td>
</tr>
<tr>
<td>( sh ) ( r \neq h ) ( r \neq m ) ( m = h )</td>
<td>( (1 - \lambda_n)(n - E(H))/n^2 )</td>
<td>2</td>
</tr>
<tr>
<td>( sh ) ( r \neq h ) ( r \neq m ) ( m \neq h ) ( m \neq h )</td>
<td>( (1 - \lambda_n)(n - E(H))/n^2 )</td>
<td>3</td>
</tr>
<tr>
<td>( ex ) ( r = h ) ( r = m ) ( m = h )</td>
<td>( p_{eh}(H)E(H)/n^2 )</td>
<td>( 2(E(H) - 1) )</td>
</tr>
<tr>
<td>( ex ) ( r = h ) ( r = m ) ( m = h ) ( m \neq h ) ( m \neq h )</td>
<td>( p_{eh}(H)(n - E(H))/n^2 )</td>
<td>( 2(E(H) - 2) + 2 )</td>
</tr>
<tr>
<td>( ex ) ( r = h ) ( r = m ) ( m \neq h )</td>
<td>( p_{eh}(H) = 1 ) ( E(H)(n - E(H))/n^2 )</td>
<td>0</td>
</tr>
<tr>
<td>( ex ) ( r \neq h ) ( r \neq m ) ( m \neq h )</td>
<td>( p_{eh}(H &gt; 1)E(H)(n - E(H))/n^2 )</td>
<td>( 2(E(H) - 1) + 2 )</td>
</tr>
<tr>
<td>( ex ) ( r \neq h ) ( r \neq m ) ( m \neq h )</td>
<td>( p_{eh}(n - E(H))(n - E(H))/n^2 )</td>
<td>( 2E(H) )</td>
</tr>
<tr>
<td>( ex ) ( r \neq h ) ( r \neq m ) ( m \neq h )</td>
<td>( p_{eh}(n - E(H))(n - E(H))/n^2 )</td>
<td>( 2E(H) - 1 + 2 )</td>
</tr>
</tbody>
</table>

The dependent latency \( t_m \cdot E(S_m) \), and the times for sending and receiving the token, as these are processed outside sojourn time \( t_c \). Since the latter happens at fixed times only, it is assumed to be without queuing. Thus

\[
E(T_{cycle}) = n \left[ t_c + \frac{2}{v}c_0 + t_0 \right] + \left( \frac{2}{v}c_m + t_m \right) E(S_m) \quad \text{seconds.} \tag{6}
\]

The calculation of token size \( E(S_m) \) is done via a balance equation. Straightforward considerations show that for the stationary system, every node sees \( \lambda_n \) \((n - 1)(n - 2) + (n - 1) \) messages in the token, and it removes messages from \( n - 1 \) nodes, that is, a fraction of \( 2/n \) of the messages. As each lock request requires \( E(C_{load}) \) communications according to the optimization applied, every node adds \( E(C_{load}) \cdot \lambda_n/n \) \( E(T_{cycle}) \) elementary messages of length \( s_m \) to the token, accumulated during absence of the token and destined to other nodes (we assume lock requests are accepted for the token just until its departure). Thus in steady state, we must have

\[
\frac{\lambda_n E(C_{load})s_m}{n} E(T_{cycle}) = E(S_m) \frac{2}{n}. \tag{7}
\]

Introducing for \( E(T_{cycle}) \) the explicit expression of equation (6) and using the abbreviations

\[
c := \frac{\lambda_n E(C_{load})s_m}{n}, \quad c_0 := \frac{2}{v}c_0 + t_0, \quad a := \frac{2}{v}c_m + t_m \quad \text{transfer times},
\]

we obtain

\[
E(S_m) = \frac{n^2c(t_c + a)}{2 - cn^2b} \quad \text{Bytes}, \tag{9}
\]

under the condition that \( cb < 2n^{-2} \) respectively

\[
\lambda < 2n_t/[n^2E(C_{load})s_m(\frac{2}{v}c_m + t_m)] \quad \text{seconds}^{-1} \tag{10}
\]

as limiting throughput. We finally obtain

\[
E(T_{cycle}) = n(t_c + a + \frac{bn^2c(t_c + a)}{2 - n^2cb}) \quad \text{seconds}. \tag{11}
\]

Response time \( E(R) \) of lock request now consists of the time \( c_t/v \) of lock processing and, for each of the \( E(C) \) communications, additional \( E(T_{cycle})/(2 \cdot n_t) \) seconds for waiting for any one of the \( n_t \) tokens and in the mean \( E(T_{cycle})/2 \) seconds to reach the destination node, including communication processing. As lock processing takes place all the time, we introduce queuing at this station with lock request arrival rate \( 2\lambda \) (request/grant and release). There is no queuing for communication processing which is already included by the stability condition of equation (10). Hence according to equation (1) we have

\[
E(R) = \frac{1 - \lambda c_t/v}{(v/c_t) - 2\lambda} + E(C) \left( \frac{E(T_{cycle})}{2n_t} + \frac{E(T_{cycle})}{2} \right) \quad \text{seconds}. \tag{12}
\]

4.5. Replicated lock database with token ring communication (RR.ds)

Expected cycle time again is given by equation (6). Now indeed every local lock request/grant and later on its release must be communicated to all other nodes (but without requiring a reply), and all messages are removed by their initiating nodes after one cycle only. Thus the balance equation now reads \( \frac{2\lambda}{n_t} s_m E(T_{cycle}) = E(S_m) \frac{1}{n} \). We obtain

\[
E(S_m) = \frac{2\lambda}{n_t} s_m E(T_{cycle}) = \frac{2\lambda}{n_t} s_m \left( t_c + \frac{2}{v}c_0 + t_0 \right) + \left( \frac{2}{v}c_m + t_m \right) E(S_m), \quad \text{seconds}. \tag{13}
\]
such that now
\[ E(S_m) = n^2 \frac{c(a + t_s)}{1 - n^2 cb}, \quad (14) \]
with
c := \frac{2\lambda}{n_t} s_m \text{ token specific arrival rate of bytes to be transferred,}
a := \frac{2}{v} c_0 + t_0 \text{ message size independent transfer times,}
b := \frac{2}{v} c_m + t_m \text{ transfer time per Byte message size,}

under the condition that \( cb \leq n^{-2} \) respectively
\[ \lambda < n_t \left[ 2 s_m \left( \frac{2}{v} c_m + t_m \right) n^2 \right] \text{ seconds}^{-1}. \quad (16) \]

With the above notations, expected token size \( E(S_m) \) and cycle time \( E(T_{cycle}) \) follow again from equations (9) and (11) respectively. One observes that limiting throughput is about two times lower than in strategy PR, provided that
\[ E(C_{load}) \approx 2. \]

The expected response time \( E(R) \) of a collision free lock request now cannot be obtained as straightforwardly as in equation (12). One may argue that all lock requests arrive in the mean just in the middle between two successive arrivals of the token, and then are processed in a M/G/1-FCFS queue, such that the expected response time is
\[ \frac{1 - \lambda c_1/v}{(v/c_1) - 2x} + \frac{1}{2}(E(T_{cycle}) - t_s) \text{ seconds.} \quad (17) \]

In fact, this expression underestimates the true \( E(R) \) for large \( \lambda \). Then a request will in general not be processed in the next sojourn time of the token, requiring an additional multiple of token cycle times. Approximating \( E(R) \) by the expected response time in a global M/G/1-queue approach with arrival rate \( \lambda \) and mean service request time \( t_s/E(T_{cycle}) \) underestimates true \( E(R) \) for small \( \lambda \). Then at least \( (E(T_{cycle}) - t_s)/2 \) seconds pass by until the token arrives. Hence we apply queueing with a server of walking type (cf [8], section 2.2). As this approach assumes that just one service request can be processed in the sojourn time of the token, we collect \( b = t_s \cdot v/c_l \) lock requests processable in one token sojourn time, to a new service request, such that \( \lambda' = \lambda/b \) is our new service requests arrival rate. Then \( E(R') = E(N')/\lambda' \), where \( N' \) denotes the number of new service requests in the system. According to [8], p 59,
\[ E(N') \]
\[ = \frac{\lambda'(E(T_{cycle}) - t_s)(\lambda' E(T_{cycle}) t_s - E(T_{cycle}) - t_s)}{2E(T_{cycle})(\lambda' E(T_{cycle}) - 1)}, \quad (18) \]

and every original lock request waits in the mean \( E(R) = (E(R')/b)((b + 1)/2 \) to be executed completely.

One should observe that for this formula it is assumed (different from our previous assumption) that lock requests are accepted for current service only just before the token arrives. Again we neglect queueing effects for communication processing, since these are already taken into account in \( E(S_m) \).

5. Evaluation of a particular HW/SW-configuration

The dependence of the expected response time \( E(R) \) of a lock request on actual throughput \( \lambda \) is by far the most important performance factor. For the HW- and SW-parameters from table 4 and 14 DLM implementation strategies, this dependence is shown in figure 3. The choice of the parameters stems from data from the BS2000 mainframe operating system, but may be fairly typical for a wider class of general purpose computer systems (cf e.g. [2]).
Performance evaluation of distributed lock management

Figure 3. Response time versus throughput for a DLM for the indicated implementation alternatives.

For the chosen HW- and SW-parameters, a partitioned lock database (PX.x) outperforms other approaches, by response time if combined with an immediate communication method (PL.x), and by throughput if a timer controlled communication is used (PT.x). This is not unexpected, since timer controlled or token ring based communication intend to increase tolerable throughput. The experiment unveils that this goal is better obtained by timer controlled communication. Additional simulations showed that in a token ring based communication, already for low lock request arrival rates several token cycles are necessary until completion of lock request processing. Token ring communication pays only if combined with a replicated lock database, which overall is not the best, but a fine and robust implementation strategy.

The different optimization concepts only gradually specify this result. In our experiments, static master location and immediate return of locks (X.is) proves to support shorter response times and highest throughput. And static master location cannot be recommended in combination with return on demand of locks (X.ds).

Theoretically, an optimal implementation must switch with increasing arrival rate $\lambda$ to the currently optimal implementation strategy. In general, neither will this be feasible for implementation, nor are these crossing points fixed but depend on further load characteristics.

6. Discussion

We have presented analytical models for the response time and for related measures, for several implementation strategies of distributed lock management. The models take into consideration a considerable number of HW-, SW- and load parameters, and can be used for a rapid assessment and comparison of strategies and optimizations, to assist in commercial operating system development.

Since our investigation aimed at tools for selecting different DLM implementation strategies and communication methods, they can be refined to obtain better accuracy and support higher flexibility within the model. We restrict this study to the derivation of average values. Though sufficient in most cases, such a procedure cannot cope with influences of variation, in particular, if unstable balance situations arise from actual (and not mean) values. Only a distributional analysis can unveil for instance the probability of overload situations by random variations of lock requests, even if the expected limiting throughput is not reached. We have adopted a rough load model of incoming
lock requests, distributed uniformly among the nodes and
the set of locks and lock modes, which does not capture the
influence load affinity and locality. The same holds for new
lock requests, which may impact on the performance in an
initializing state of the lock management system. Finally,
we neglect all secondary aspects of DLM implementations,
such as memory consumption, recovery provision and ex-
ecution, or deadlock provision, detection and resolution.
These aspects can be investigated separately, since their
costs simply add onto our results.

A final word is in order concerning the advantages
of an analytical approach in comparison to a simulation.
According to our experience, design, implementation and
in particular quality assurance of a reliable simulation
model will cost an order of magnitude more than that of
an analytical model. A simulation model in most cases
requires longer computing time so that a large set of
experiments, necessary for unveiling trends and functional
dependencies, often becomes impossible. Furthermore a
screening assessment of DLM implementations does not
require detailed simulations.

References

performance modeling: Alternatives and implications
ACM Trans. Database Syst. 12 609–54
Experiences in integrating distributed shared memory
with virtual memory management Operating Syst. Rev.
26 (3) 4–26
Control in Distributed Database Systems (Amsterdam:
North-Holland)
modelling of parallel access to shared data Commun.
ACM 26 196–202
concurrency–coherence control protocols for distributed
transactions processing systems with regional locality
IEEE Trans. Software Eng. 18 899–914
Environments (Cambridge, MA: MIT Press)
VAX/VMS Internals and Data Structures (Burlington,
MA: Digital Press/Digital Equipment Corp.)
[10] Hastings A B 1990 Distributed lock management in a
transaction processing environment Proc. 9th Symp.
Reliable Distributed Systems (Los Alamitos, CA: IEEE
Computer Society Press)
distributed lock manager 30 (7)
virtual memory Technical Report Universität Dortmund
[13] Li K and Hudak P 1989 Memory coherence in shared
virtual memory systems ACM Trans. Comput. Syst. 7
321–59
[14] Malamud C 1989 DEC Networks and Architectures (New
York: Intertext Publications, McGraw-Hill)
concurrency and coherence control protocols for data
sharing Research Report IBM Research Division, T J
Watson Research Center, New York
IMS/VS: an evolving system IBM Syst. J. 21 490–510
distributed shared memory Computer May 1990, 54–64
[18] Thomasian A and Ryu I K 1991 Performance analysis of
two-phase locking IEEE Trans. Software Eng. 17
386–402
(Englewood Cliffs, NJ: Prentice Hall)
Queues (Englewood Cliffs, NJ: Prentice Hall)
[21] Wolfson O 1990 A comparative analysis of
two-phase-commit protocols Proc. 3rd Int. Conf.
Database Theory (Paris) (Berlin: Springer) pp 291–304
[22] Wolman B L 1989 An analysis of server-based locking
[23] Yu P and Dias M 1993 Performance analysis of
concurrency control using locking with deferred