J. Hofmann
H. Schmutz

Performance Analysis of Suspend Locks in Operating Systems

Performance models of suspend locks in operating systems are developed and analyzed. Analytical expressions and algorithms for numerical results have been obtained for an arbitrary number of processors, an arbitrary number of tasks, and one suspend lock. The results are discussed and important dependencies among the major characteristic quantities such as queue length, processor speed, number of processors, dispatching overhead, and processor degradation are shown. Expressions are derived permitting the control program designer to estimate the system impact of locking during the early design phase.

1. Introduction

Locks are used in operating systems to synchronize concurrent logical processes, referred to as *tasks*. A task which intends to modify a global object (a control block or a set of control blocks) must ensure that no other task tries to access and/or modify this object at the same time. The part of a task which modifies the object is known as *the critical section*. The task acquires a lock at the beginning of the critical section via a LOCK operation and releases the lock immediately before it leaves the critical section via an UNLOCK operation. The implementation must ensure that at any time at most one task may hold the lock. This type of lock is known as an "exclusive" lock; other, less restrictive types of locks may be employed in real systems but are not investigated in this paper, since typical high frequency locks, which are most likely to become bottlenecks in a system, are, in general, of the type "exclusive." If a task issues a LOCK operation to a lock currently being held by another task, the first task must enter a "wait state" at least until the current holder of the lock issues the UNLOCK operation.

We distinguish between *spin locks* and *suspend locks*. If task A issues a lock request to a spin lock currently held by task B, task A executes the lock request in a loop until the lock is released. Task A spins during its wait time, *i.e.*, task A remains busy. Efficiency requires that tasks are running disabled for interrupts (and consequently may not cause a paging exception) while holding a spin lock, and, of course, that the lock hold time be very short.

The analysis of spin locks is contained in [1, 2]. Spin locks are, from a performance point of view, preferable to suspend locks; however, due to the restrictions mentioned above, suspend locks cannot generally be replaced by spin locks in control programs.

If task A attempts to acquire a suspend lock currently held by task B, the processor which interprets task A enqueues the task and searches the dispatcher queue for a third task, C, which is ready to continue execution. In this case task A is suspended. When task B releases the lock, two possibilities exist:

1. The processor which interprets task B puts task B into the dispatcher queue as a ready task and resumes the execution of task A. In this case the overhead for acquiring an occupied lock includes two dispatches.
2. The processor associated with task B dequeues task A and puts task A as a ready task into the dispatcher queue. In this case only one dispatch is caused by the lock request for an occupied lock.

If case (1) applies in a system, we talk of a "minimal queue length" strategy. If case (2) applies in a system, we talk of a "minimal dispatch" strategy. Mixed strategies are likely in practice, for example, according to the following rule: If task A has a priority greater than that of task B, case (1) applies, otherwise case (2) applies. In analyzing suspend

Copyright 1982 by International Business Machines Corporation. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to *republish* other excerpts should be obtained from the Editor.

locks we are primarily interested in the additional *lock-forced dispatches* and in the *queue length* of the queues in front of locks. Dispatches represent an overhead which may be significantly greater than the overhead of a LOCK/UNLOCK sequence to a free suspend lock. Queues in front of locks increase response time and are detrimental to the system objectives of making optimal use of resources, since at most one of the tasks in the queue may be dispatched. Alternatively, to achieve the same resource utilization with large queues in front of locks, the number of tasks in the system has to be increased correspondingly, with negative impact on performance and response time. In addition, suspend locks may cause processor degradation if no ready tasks are found in the dispatcher queue.

Spin locks are only necessary in operating systems supporting multiprocessors. If employed on a uniprocessor, no processor degradation results. Suspend locks, on the other hand, may show all three effects—queueing, forced dispatching, and processor degradation—even on a uniprocessor.

Locks as logical devices have gotten considerable attention in the literature. Frequently [3] spin locks are referred to as LOCKS and suspend locks as (binary) semaphores. The use of the term "semaphore" for a suspend lock is due to E. W. Dijkstra [4].

The danger of deadlocks resulting if a task which already holds a lock tries to acquire additional locks has been the subject of extensive research [5–7].

While the logical and semantical questions associated with locks have been thoroughly studied, little formal performance analysis of locking situations can be found. Gilbert [1] used queueing theory to study the impact of lock granularity on the interference of spin locks in multiprocessor systems. Reference [2] contains a generalization and extended analysis of his model.

Kumar [8] uses analytical techniques to investigate the cost of deadlock prevention in operating systems. Our model, described in the next section, does not model costs for deadlock prevention with the realistic assumption that deadlocks are avoided by a proper lock use discipline.

Our research has been strongly motivated by observations on suspend locks in MVS [9, 10] and in data base systems [11]. The convoy phenomenon is the name for observed stable queues in front of suspend locks. *Ad hoc* solutions (corresponding to the strategies described above) have been applied without formal analysis. Analysis in this paper is restricted to cases which are expected to avoid the convoy phenomenon.

Section 2 of this paper describes in detail the locking model underlying our analysis. Section 3 contains the mathematics to derive the formulas for calculating dispatch rate, queue length, and processor degradation. In Section 4 we illustrate and discuss the results. Section 5 summarizes the results and indicates areas for further research.

2. The suspend lock model

• *Global symbols*

The symbols used in this paper are described here for reference. This description is also a summary of model parameters and symbols for primary results.

Model parameters

M number of processors, *i.e.*, level of multiprocessing.

N number of tasks, *i.e.*, level of multiprogramming.

T_A CPU time consumed by a task between a lock release and the next lock request.

T_L CPU time consumed by a task between a lock request and the lock release.

T_W page wait time, *i.e.*, time between a paging exception and the availability of the page in main memory.

T_D dispatch time, *i.e.*, time between a lock release and the first normal dispatch of a task waiting for the lock while the lock is free.

D_0 Bernoulli variable (with values 0 or 1). If a task releases a lock and other tasks are waiting for the lock and if $D_0 = 1$, then one of the waiting tasks is immediately dispatched. If $D_0 = 0$, the processor remains with the lock-releasing task.

B_0 Bernoulli variable (with values 0 or 1). If a task obtains a lock and $B_0 = 0$, the task causes a paging exception immediately before the lock is released.

The above described random variables actually occur in sequences and are assumed to be within the sequence identically and independently distributed. Associated average transfer rates and means are denoted by

$\lambda = 1/E[T_A]$ lock request rate of an active task outside the critical region.

$\mu = 1/E[T_L]$ lock release rate of a lock-holding task.

$\tau = 1/E[T_W]$ page transfer rate.

$\delta = M/E[T_D]$ normal dispatch rate.

$d_0 = E[D_0]$ probability of an immediate dispatch.

$b_0 = E[B_0]$ probability of no paging exception in a critical region.

Further, the abbreviations

$\lambda_m = (M - m) \cdot \lambda,$

$\rho = \lambda/\mu,$

$\rho_m = \lambda_m/\mu,$

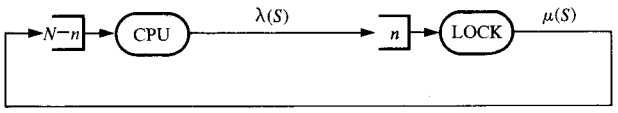

Figure 1 Basic structure of the suspend lock model. The service rates at the two queues are dependent on the system state S.

are used, where m is the number of processors outside a critical region.

Results

X_D number of delayed tasks, *i.e.,* number of tasks waiting for or holding lock.

q_L $= E[X_D]$, the mean number of delayed tasks.

u_L the lock utilization, *i.e.,* the expected fraction of time during which the lock is occupied.

r_D relative dispatch rate, *i.e.,* expected number of dispatches forced by locking per lock request. The value of r_D may be any value between 0 (the lock is always free) and 2 (the lock is always occupied and D_0 is always 1).

d_P processor degradation, *i.e.,* the expected fraction of time each processor is idle due to synchronization of locks.

• *Basic structure of the model*

Our main objective is the development of a model of queueing phenomena in front of high frequency suspend locks. Although the model should be as simple as possible, it has to reflect system events such as paging exceptions in the critical section and certain dispatching decisions.

The basic structure of the model is shown in Fig. 1. It is a cyclic queueing model consisting of a CPU queue and a lock queue. The processing rates at each of these queues are dependent on a system state S, which contains the length of the individual queues in addition to state information resulting from dispatching decisions and paging exceptions. The dynamic behavior of the model is discussed in detail subsequently.

The dispatcher is an important system process affecting suspend locks, although many dispatching decisions need be of no concern to us. Suspend locks are service stations rather than servers and depend on the dispatcher for actual processing. We assume that one of the strategies ("minimal queue length," "minimal dispatch," or a mixture thereof) is observed by the dispatcher. As known from [10, 11], a deviation from such a strategy for high frequency locks causes performance collapse even with low levels of multiprogramming. This implies, in particular, that the lock-holding task is, with ignorably low probability, preempted deliberately by the dispatcher.

The major dispatcher decisions remaining are related to the dispatching of tasks which have enqueued for the lock while the lock was held by some other task. Whenever a lock is returned and the lock queue is not empty, the dispatcher has the choice of either continuing with the task which returned the lock or dispatching one of the tasks waiting for the lock. This decision is modeled as a Bernoulli variable D_0 with mean $d_0 = E[D_0]$. With probability d_0 the dispatcher switches tasks at lock release time, and with probability $1 - d_0$ the processor remains with the task which releases the lock.

In general, if $d_0 < 1$, there are system states in which the lock is free and n tasks are waiting for the lock, *i.e.,* in a state immediately preceding the LOCK instructions. Consider, for example, the case $M = 3$ and $N = 5$ with tasks A, B, C, D, and E. Assume that task A is enqueued for the lock while task B was holding the lock. Now let task B leave the critical section and let its processor stay with task B, *i.e.,* $D_0 = 0$. All five tasks are now ready, and the three processors are assigned to task B and to, say, C and D. Task A's next instruction would be a LOCK request. We say task A is waiting for the lock, although it is ready as long as the lock remains free. In general, we have n ready tasks "waiting" for the lock while the lock is free. We assume that the dispatcher is aware of such tasks and selects one of them at the time of a normal dispatch. This normal dispatching is modeled to happen at time T_D after the last lock release, provided the lock is still free. Note that normal dispatching is modeled as an additional process running in parallel with CPU processing. In the example above, the lock may become busy as a consequence of a normal dispatching decision selecting task A (at a rate depending on dispatcher parameters such as time slice length) or as a consequence of a lock request generated by any of the tasks B, C, D, or E while dispatched.

A second important system process, which has to be reflected in the model, is forced preemption (above we have excluded only deliberate preemption). For suspend locks, preemption is, in principle, permitted. Nevertheless, the design objective is to preempt lock-holding tasks as seldom as possible. A typical example of a forced preemption, which cannot be excluded, is a paging exception. Other types of preemption can be mapped to the same mechanism. Forced preemption is modeled as a Bernoulli variable B_0 with mean b_0. For simplicity, we assume paging exceptions to occur only immediately before the UNLOCK instructions. A comparison of our model with a more general model has shown that this simplification has ignorable impact on the numeric

results, since we must assume b_0, the probability for no preemption in the critical section, to be very close to 1. A paging exception causes the lock-holding task to be delayed for a time T_W. During this time the processor of the lock holder is dispatched to some other task and thus increases the arrival rate of lock requests. Consider again the example system above. When task B entered the critical section, it had a processor assigned to it and only two processors were remaining to generate lock requests. However, after task B has caused a paging exception in the critical section, its processor is redispatched to some other task, say A. Now we have three processors generating lock requests. After the time T_W has elapsed, the lock holder returns the lock and a dispatching decision as described above takes place.

We assume the workload to consist of N tasks, each with statistically identical, independent internal behavior. Of course, tasks interact via the suspend lock. A task consists of a sequence of transactions. Each transaction consumes a CPU time T_A outside the critical section and a CPU time T_L within the critical section.

• *Extension of the model*

A significant simplification in our model results from the absence of queues for devices and, of course, queues for other locks than the one considered.

This is certainly justified for our objective of demonstrating the queueing phenomena at suspend locks. To use our model for complete real systems, one would have to consider a complex network which, with the current state of the art, would have to be solved approximately by replacing the subnetwork containing all queues except the queue in front of the lock of interest by one "effective" CPU queue with M "effective" processors. The effective M processors would reflect the degree of parallelism in the subnetwork. The effective processing rates λ_m at the CPU queue and the effective dispatching rates $\delta(n)$ would be obtained iteratively by solving the suspend lock model and the subnetwork model individually. This technique is standard [12, 13] in solving approximate queueing network problems which do not satisfy the criteria for product form solution.

• *Definition of the model*

Figure 2 illustrates the states of a task and the decisions affecting the states as the task performs transactions. Figure 3 illustrates the model as a cyclic queueing network of two queues and a stage for tasks waiting for the completion of a paging exception. Figure 3 also shows the flow of processors between the service stations for normal processing and for critical section processing. Note, however, that Fig. 3 is not a complete formal definition of the model, since not all dependencies between the different flows have been expressed.

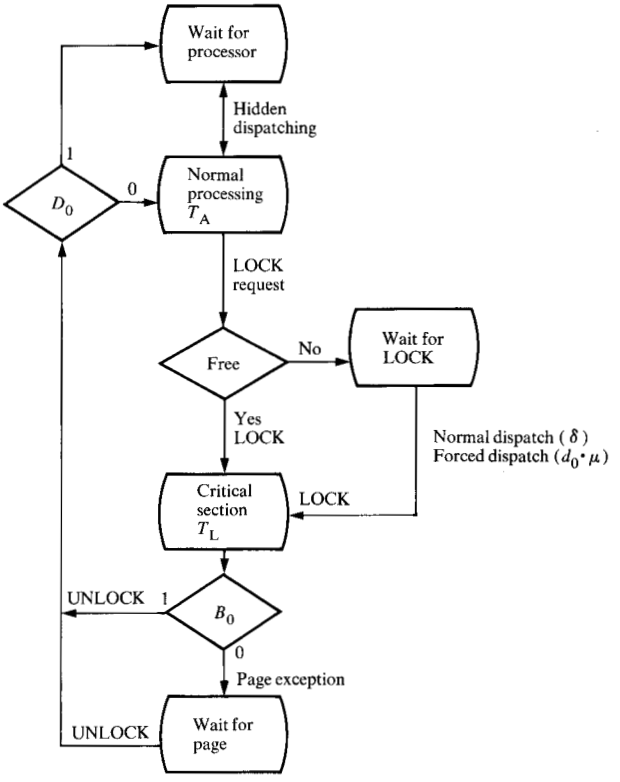

Figure 2 States and state transitions of a task in the suspend lock model.

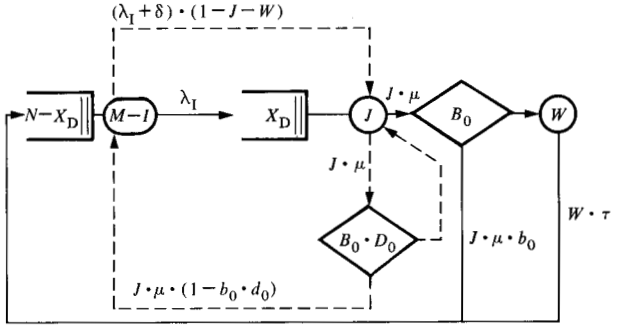

Figure 3 Flow of tasks (solid lines) and processors (dashed lines) in the suspend lock model, where $J \in \{0, 1\}$, $W \in \{0, 1\}$, $J + W \leq 1$; $X_D = 0$ implies $J + W = 0$; and $M - I = \min(M - J, N - X_D)$.

While Fig. 2 is self-explanatory, some discussion of Fig. 3 is in order. Consider first the CPU queue. It has $M - I$ processors actively working, where I depends on J and the number of tasks in the CPU queue. If the number of tasks in this queue is large enough, we can set $I = J$. The quantity J is the number of processors assigned to processing the lock-holding task (at most one, of course).

Assume initially that $X_D = 0$, *i.e.*, the lock is free and no task is waiting for the lock; then $J = I = 0$, M processors are generating lock requests at rate $\lambda_0 = M \cdot \lambda$, and $W = 0$, indicating that no lock-holding task waits for a page transfer. If a lock request is generated, one task moves along the solid line from the CPU queue to the lock queue and X_D becomes 1; simultaneously, one processor moves along the dashed line from the CPU service station to the lock service station. With this move J becomes 1 (but W remains 0). In this state, $M - 1$ processors continue to generate lock requests at a rate $\lambda_1 = (M - 1) \cdot \lambda$ and one processor interprets the task in its critical region.

Two possibilities exist for the next event:

1. A second lock request is generated before the first task has terminated CPU processing. Then the task issuing the second LOCK instruction moves to the lock queue, X_D becomes 2, but the processor which has interpreted the task remains at the CPU queue and assigns itself to some other ready task at the CPU queue. The rate of the processor moving along the dashed line contains a term $\lambda_I \cdot (1 - J - W)$, which is zero as long as J or W is different from zero.
2. The lock-holding task terminates CPU processing before a second lock request is issued. The task and its processor simultaneously enter a decision B_0, which determines whether a paging exception occurs. If $B_0 = 1$ (with probability b_0), no paging exception occurs, and both the task and the processor return to the CPU queue. X_D and J become zero. The decision $D_0 = 1$ is implied in this case (see below). If, instead, $B_0 = 0$, the task generates a paging exception, and W becomes 1. The processor, however, becomes free and returns to CPU queue processing [rate $J \cdot \mu \cdot (1 - B_0)$, since $D_0 = 1$]. J becomes zero again. The system is now in a state in which the lock is occupied and the lock-holding task is waiting for the completion of a paging exception. Lock requests are now generated again at a rate $\lambda_0 = M \cdot \lambda$. As soon as the page transfer is complete (rate $W \cdot \tau$), the task also returns to the CPU queue and W again becomes zero.

To illustrate the decision D_0, assume that the system is in the state after the first possibility above, *i. e.*, $X_D = 2$, $I = J = 1$, $W = 0$. Assume the lock-holding task to terminate the critical region without generating a page exception (*i. e.*, $B_0 = 1$). Then the task returns back to the CPU queue and X_D becomes 1, independent of the decision D_0. The latter decision correlates only to the dispatching strategy. The processor, which was assigned to the lock-holding task, stays at the lock queue if $D_0 = 1$ (with probability d_0) and continues processing with the one remaining task in the lock queue; alternatively, if $D_0 = 0$ (with probability $1 - d_0$), the processor accompanies the task which released the lock and enters the CPU queue [rate $J \cdot \mu \cdot (1 - D_0)$, since we assumed $B_0 = 1$].

Assume that the decision has been $D_0 = 0$ and that the system then moves into a state in which M processors are assigned to the CPU queue, $X_D = 1$, and $J = W = 0$. There exist now two possibilities for J to become 1. The first is that a lock request is generated (rate $\lambda_0 = M \cdot \lambda$) as discussed above. A second possibility is that in the course of normal dispatching a processor is assigned to the task in the lock queue. This happens with rate δ; however, this dispatching is only meaningful while the lock is free (*i. e.*, $J = W = 0$). Therefore, processors flow with a rate $\delta \cdot (1 - J - W)$ from the CPU queue to the lock queue, in addition to the flow along this path resulting from lock requests. Clearly, if a processor changes to the lock queue as a result of a dispatching decision, X_D remains unmodified until a new arrival to or departure from the lock queue.

Unfortunately, many interdependencies exist among the various states, the flow of tasks, and the flow of processors. These dependencies are not completely reflected in Fig. 3.

To illustrate the model more formally, we describe subsequently a complete semi-Markov model, which is a special case of the model solved in Section 3. While in Section 3 the times T_L, T_D, and T_W may be arbitrarily distributed, we assume here, for the purpose of illustration, that these times are exponentially distributed with rates μ, δ, and τ.

Let

- $(0, n)$ denote the states in which the lock is free and n tasks are waiting for the lock, and $N - n$ tasks are in the CPU queue, $0 \le n \le N - M$,
- $(1, n)$ denote the states in which one processor is assigned to a lock-holding task, and $n - 1$ tasks are waiting for the lock, $1 \le n \le N$, and
- $(2, n)$ denote the states in which the lock-holding task is waiting for completion of a page transfer, and $n - 1$ tasks are waiting for the lock, $1 \le n \le N$.

Further, let $d_{0,n}$ denote the queue-length-dependent probability for an immediate dispatch, and δ_n denote the queue-length-dependent dispatching rate.

Then the transition rates are

δ_n from $(0, n)$ to $(1, n)$, $1 \le n \le N - M$,
λ_0 from $(0, n)$ to $(1, n + 1)$, $0 \le n \le N - M$,
λ_1 from $(1, n)$ to $(1, n + 1)$, $1 \le n \le N - M$,
λ_m from $(1, N - M + m)$ to $(1, N - M + m + 1)$, $1 \le m < M$,

$\mu \cdot b_0$ from (1, 1) to (0, 0),
$\mu \cdot b_0 \cdot d_{0,n}$ from $(1, n+1)$ to $(1, n)$, $1 \le n \le N - M$,
$\mu \cdot b_0$ from $(1, N - M + m + 1)$ to $(1, N - M + m)$,
$1 \le m \le M - 1$,
$\mu \cdot b_0 \cdot (1 - d_{0,n})$ from $(1, n+1)$ to $(0, n)$,
$1 \le n \le N - M$,
$\mu \cdot (1 - b_0)$ from $(1, n)$ to $(2, n)$, $1 \le n \le N$,
τ from (2, 1) to (0, 0),
$\tau \cdot d_{0,n}$ from $(2, n+1)$ to $(1, n)$, $1 \le n \le N - M$,
τ from $(2, N - M + m + 1)$ to $(1, N - M + m)$,
$1 \le m \le M - 1$,
$\tau \cdot (1 - d_{0,n})$ from $(2, n+1)$ to $(0, n)$, $1 \le n \le N - M$,
λ_0 from $(2, n)$ to $(2, n+1)$, $1 \le n \le N - M$,
λ_m from $(2, N - M + m)$ to $(2, N - M + m + 1)$,
$1 \le m < M$.

The transition diagrams of Figs. 4 and 5 illustrate the model for the cases $M = 1$, $N = 2$ and $M = 3$, $N = 5$. Note that $\lambda_m = \lambda \cdot (M - m)$.

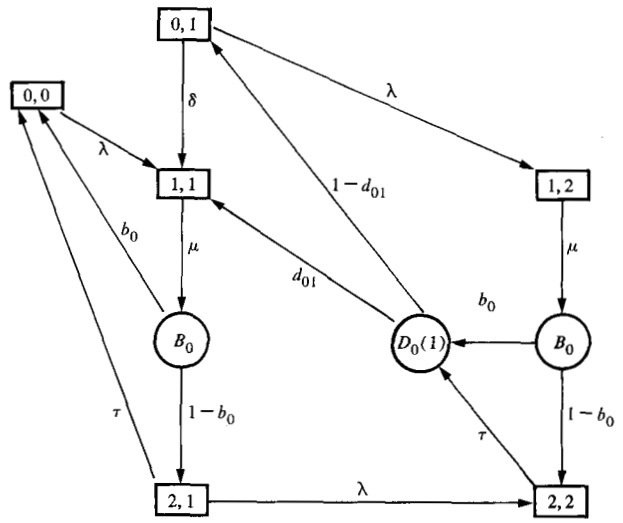

Figure 4 Complete state transition diagram for a suspend lock with $M = 1$, $N = 2$.

To analyze the transitions, consider the state (0, 2) in Fig. 5. In terms of Fig. 3 this means $X_D = 2$, $J = W = I = 0$. The lock is free; all three processors are engaged in CPU queue processing. Two tasks are "waiting" for the lock and ready for dispatching. Two types of events change the system state:

1. *A lock request.* The three processors generate lock requests at a rate $3 \cdot \lambda$. The consequence is a transition to $X_D = 3$ and $J = 1$, since the processor accompanies the task which generated the lock request. The resulting state is (1, 3).
2. *A dispatching process.* A normal dispatching decision may select one of the two ready tasks waiting for the lock. This happens with rate δ_2. The resulting state is (1, 2), *i. e.*, X_D remains 2, but J becomes 1.

Next consider the state (1, 2), where J is one and two processors are assigned to CPU queue processing. These two processors are responsible for the transition rate $2 \cdot \lambda$ from (1, 2) to (1, 3). One processor interprets the lock-holding task which finishes its CPU processing with rate μ and enters the Bernoulli stage B_0. For $B_0 = 0$, a paging exception is modeled and the transition is to (2, 2) [combined rate $\mu \cdot (1 - b_0)$]. For $B_0 = 1$, a second Bernoulli stage, D_{01}, is entered to model the dispatching strategy. For $D_{01} = 1$, an immediate dispatch decision is modeled, resulting in a transition to the state (1, 1); *i. e.*, the processor remains at the lock service station. The combined transition rate is $\mu \cdot b_0 \cdot d_{01}$. For $D_{01} = 0$, the processor remains with the task, thus returning to CPU queue processing. The transition is to the state (0, 1) with a combined rate $\mu \cdot b_0 \cdot (1 - d_{01})$.

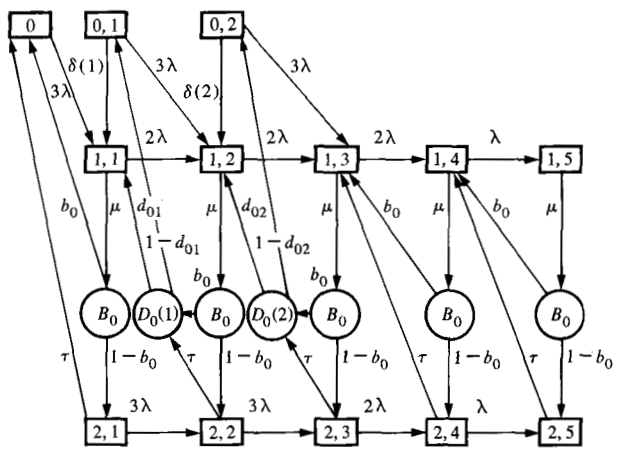

Figure 5 Complete state transition diagram for a suspend lock with $M = 3$, $N = 5$.

Finally, consider the state (2, 2), where $X_D = 2$, $J = 0$, and $W = 1$. The lock is occupied; the lock-holding task is waiting for the completion of a page transfer. All three processors are assigned to CPU processing, thus generating lock requests at a rate $3 \cdot \lambda$ and causing transitions to the state (2, 3). The paging device completes the page transfer at a rate τ, causing a Bernoulli decision D_{01}, as above. In fact, with the completion of the page transfer, the lock-holding task again obtains a processor; however, its next instruction is UNLOCK. The Bernoulli stage D_{01} decides whether the processor resides with this task [combined transition rate $\tau \cdot (1 - d_{01})$ to (0, 1)] or continues lock processing [combined transition rate $\tau \cdot d_{01}$ to (1, 1)].

The above discussion illustrates the transitions within the inner part of the chain of states. At the two ends of the chain slight modifications are necessary due to the unavailability of tasks waiting for the lock (if X_D becomes zero) or due to the unavailability of tasks for CPU processing (if X_D exceeds $N - M$). The verification of the transitions at the ends of the chain is left to the reader.

The Bernoulli stages B_0 and D_0 complicate the state transition diagram. This complication can be removed in the general model in Section 3, as we will see. Instead of modeling no paging exception within the critical section with probability b_0, we model the paging exceptions to always appear but to cause, with a probability b_0, a zero delay. Similarly, we can combine D_0 with the time T_D for normal dispatches. The alternative is mathematically equivalent to the model above, but easier to handle. However, the combination is only valid if we permit general distributions for T_W and T_D, since the combinations are, in general, not exponentially distributed, even if T_D and T_W were originally exponentially distributed. A complete formal definition of the general model is implicitly contained in the state transition diagrams and the derived balance equations of the next section.

• *Dependency of the dispatching strategy on the queue length*

The described model permits arbitrary dependency of the dispatch rate and the immediate dispatch decisions on the queue length. Only two types of functions have actually been used:

1. No dependence,
2. $\delta(n) = \delta(\infty) \cdot n/(n + 3)$,

   $d_{0n} = 1 - (1 - d_{01})^n$.

The rationale behind the function $\delta(n)$ is the observation that in a balanced system with good reponse time no queue becomes very large. An average CPU queue length of three is approximately adequate for the type of system considered with little dependence on the actual configuration.

For $d_{01} = 0.5$ the expression for d_{0n} approximately models the situation in which, at lock release time, a task may keep the CPU only if no tasks with a higher priority are waiting for the lock. For $d_{01} = 1$ we have the minimal queue length strategy and for $d_{01} = 0$ the minimal dispatch strategy.

3. Derivation of results

• *The method of phases and generalized Laplace transforms*

Usually, the method of imbedded Markov chains is applied to queueing systems with general service time distributions like the M/G/1 queueing system [14]. For lock queues, the arrival rate is dependent on the internal state of the lock. For example, if a task owning a suspend lock causes a paging exception, the lock-holding task is put into a wait state and its processor is redispatched to a task which may then execute a lock instruction involving the same lock. In other words, a paging exception in a critical region increases the arrival rate of tasks to the queue of the lock. Although the points of departure from the critical region still form a Markov chain, the probabilities of these events are not representative for the average behavior of the system. The method of imbedded Markov chains is therefore not directly applicable. In order to include general distributions of the lock hold time in the results we use a method which is equivalent to the technique of Coxian stages [14].

Let X be the random variable, $f(\cdot)$ its density function, and $\phi(\cdot)$ the Laplace transform of $f(\cdot)$. Then we assume without loss of generality that

$$\phi(s) = \sum_{i=0}^{m} g_i \cdot \omega_i/(\omega_i + s),$$

$$f(x) = \sum_{i=0}^{m} g_i \cdot \omega_i \cdot \exp(-\omega_i \cdot x),$$

$$\sum g_i = 1 \qquad (1)$$

for some m. g_i, and ω_i.

Note that the g_i are not restricted to positive values. Any real values are permitted under the constraint $f(x) \geq 0$ for all $x \geq 0$. The g_i have no interpretation as probabilities although the process can be thought of as a set of parallel stages (Fig. 6).

We refer to this special case of parallel stages as phases. The technique of phases is analytically equivalent to the technique of Coxian stages, but easier to handle [2]. In fact, the phase representation serves only as an intermediate step to derive results involving only the Laplace transform of $f(\cdot)$. The application of the method of phases to queueing systems yields expressions of the following form:

$$\phi^{(n)}(s_1, s_2, \cdots, s_n) = \sum_{i=0}^{m} g_i \cdot \omega_i / \prod_{j=1}^{n} (\omega_i + s_j).$$

Here $\phi^{(n)}(\cdot)$ is an abbreviation for the expression on the right-hand side of the equal sign.

It is easily seen from (1) that

$$\phi(s) = \phi^{(1)}(s)$$

and that the relation

$$\phi^{(n)}(s_1, \cdots, s_n) = \int_0^\infty dx_1 \cdots \int_0^\infty dx_n f(\Sigma x_j) \cdot \exp(-\Sigma x_j s_j) \qquad (2)$$

holds. Equation (2) shows $\phi^{(n)}(\cdot)$ to be a generalization of the Laplace transform. Fortunately, it is possible to calculate the $\phi^{(n)}(\cdot)$ from $\phi(\cdot)$ recursively. To show this we make use of the inverse Laplace transform

$$f(x) = (1/2\pi i) \cdot \int_{c-i\infty}^{c+i\infty} ds\phi(s) \exp(sx)$$

in (2) and integrate over the x_i to obtain

$$\phi^{(n)}(s_1, \cdots, s_n) = (1/2\pi i) \int_{c-i\infty}^{c+i\infty} ds \cdot \phi(s) / \prod_{j=1}^{n} (s_j - s). \quad (3)$$

Because $\phi(s)$ is the Laplace transform of a probability distribution, all its poles are on the negative real axis. Therefore (3) can easily be evaluated by summing over the poles s_j on the positive real axis. We apply the fraction expansion

$$\prod_j 1/(s_j - s) = \sum_j 1/(s_j - s) \cdot \prod_{k \neq j} (s_k - s_j)$$

to (3) and integrate over s in a loop closed within the positive real half plane by applying Cauchy's theorem to obtain

$$\phi^{(n)}(s_1, \cdots, s_n) = \sum_{j=1}^{n} \phi(s_j) / \prod_{k \neq j}^{n} (s_k - s_j).$$

The general form needed for solution of the suspend lock model contains k times the argument 0, $k \in \{0, 1, 2\}$, n times the argument s_0 $(0 \leq n \leq N)$, and m distinct arguments s_1 to s_m $(0 \leq m \leq M)$. We use

$$\Gamma(k, n, m) = \phi^{(k+n+m)}(0, \cdots, 0, s_0, \cdots s_1, s_2, \cdots s_m)$$

to denote this expression with implied parameters s_i and use the following notation to substitute arguments x_i for parameters s_i:

$$\Gamma(k, n, m \mid s_i = x_i).$$

For example $\Gamma(0, 0, 2 \mid s_1 = x, s_2 = y)$ denotes $\phi^{(2)}(x, y)$.

By applying partial fraction expansion and L'Hôpital's rule to (3) the following relations can easily be derived:

$$\Gamma(0, 0, m) = \sum_{j=1}^{m} \phi(s_j) \Big/ \prod_{k \neq j}^{m} (s_k - s_j), \qquad m \geq 0$$

$$\Gamma(1, 0, m) = \Gamma(0, 0, m + 1 \mid s_{m+1} = 0), \qquad m \geq 0$$

$$\Gamma(2, 0, m) = \sum_{j=1}^{m} \phi(s_j)/s_j^2 \cdot \prod_{k \neq j}^{m} (s_k - s_j) + \left(E[X] - \sum_{j=1}^{m} 1/s_j\right) \Big/ \prod_{j=1}^{m} s_j, \qquad m \geq 0$$

$$\Gamma(0, n, 0) = \frac{(-1)^{n-1}}{(n-1)!} \cdot \frac{d^{n-1}}{ds_0^{n-1}} \phi(s_0), \qquad n \geq 1$$

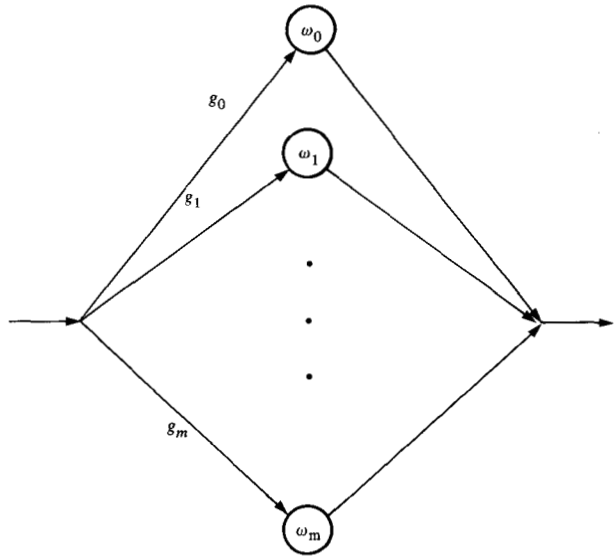

Figure 6 Phases as a set of parallel stages.

$$\Gamma(k, n, 0) = (\Gamma(k, n-1, 0) - \Gamma(k-1, n, 0))/s_0, \; n \geq 1$$

$$\Gamma(k, n, m) = (\Gamma(k, n, m-1) - \Gamma(k, n-1, m))/(s_m - s_0) \qquad m, n \geq 1. \quad (4)$$

Equations (4) permit the recursive calculation of each $\Gamma(k, n, m)$ for a given set of parameters. The computational effort (*i.e.*, the number of arithmetic operations) to obtain the expressions $\Gamma(k, n, m)$ is proportional to $M \cdot N$. For most situations, the calculations are computationally stable. If instabilities arise, we still may resort to the phase expansion as a linear combination of exponential distributions and calculate the $\Gamma(k, n, m)$ as

$$\Gamma(k, n, m) = \sum_i g_i \cdot \omega_i^{1-k}/(\omega_i + s_0)^n \cdot \prod_{j=1}^{m} (\omega_i + s_j). \quad (5)$$

Relation (5) is computationally stable for distributions without peaks (*e.g.*, for hyperexponential distributions where for all i, $g_i \geq 0$).

• *One suspend lock, $N = \infty$*

We assume the model as described in Section 2 with an infinitely long CPU queue. For this case we are able to derive closed form expressions similar to those for the M/G/1 queue. The times T_A are exponentially distributed with mean $1/\lambda$. The times T_L, T_D, and T_W are arbitrarily distributed and assumed to be given by their Laplace transforms $\phi_L(\cdot)$, $\phi_D(\cdot)$, and $\phi_W(\cdot)$ with respect means $1/\mu$, $1/\delta$, and $1/\tau$. The Bernoulli variables B_0 and D_0 are given by their

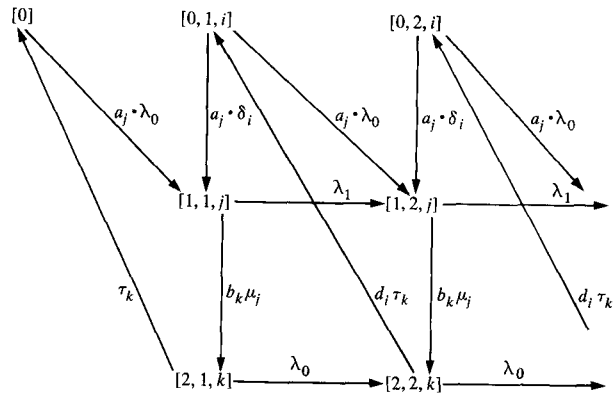

Figure 7 State transition diagram for the suspend lock model, initial part. Each node represents a set of states; each arc represents a set of transitions as indicated by the indices of the states and transition rates.

means b_0 and d_0. Mean d_0 and δ are assumed to be independent of the number n of delayed tasks. Further we use the abbreviations

$$\begin{aligned}
\rho_i &= \lambda_i/\mu = (M - i) \cdot \rho, \\
\alpha &= (1 - b_0) \cdot \mu/\tau, \\
A(z) &= \phi_L(\lambda_1 \cdot (1 - z)), \\
B(z) &= b_0 + (1 - b_0) \cdot \phi_W(\lambda_0 \cdot (1 - z)), \\
d &= d_0 + (1 - d_0) \cdot \phi_D(\lambda_0). \qquad (6)
\end{aligned}$$

Let p_n be the probability that n tasks are delayed (*i.e.*, holding or waiting for the lock), and let

$$G_{XD}(z) = \sum_{n=0}^{\infty} p_n \cdot z^n \qquad (7)$$

be the associated generating function. Further, we denote by C_L, C_D, and C_W the respective coefficients of variation and use the abbreviation

$$C_{IW}^2 = (1 + C_W^2)/(1 - b_0) - 1. \qquad (8)$$

We obtain for the equilibrium state

$$G_{XD}(z) = \frac{d - \rho_1 - \alpha\rho_0}{1 + \rho} \cdot \frac{(\lambda/\lambda_1) \cdot z \cdot (A(z) - 1) + A(z) \cdot B(z) \cdot (z - 1)}{z - A(z) \cdot B(z) \cdot (z + d(1 - z))}, \qquad (9)$$

$$q_L = \frac{\rho_0 \cdot (1 + \alpha)}{1 + \rho} + \frac{\rho \cdot \rho_1}{1 + \rho} \cdot \frac{1 + C_L^2}{2} + \frac{(1 - d) \cdot (\rho_1 + \alpha \cdot \rho_0)}{d - (\rho_1 + \alpha \cdot \rho_0)} + \frac{\rho_1^2 \cdot (1 + C_L^2)/2 + (\alpha \cdot \rho_0)^2 \cdot (1 + C_{IW}^2)/2}{d - (\rho_1 + \alpha \cdot \rho_0)}, \qquad (10)$$

$$u_L = \rho_0 \cdot (1 + \alpha)/(1 + \rho), \qquad (11)$$

$$r_D = (1 + d_0/d) \cdot (\rho_1 + \alpha \cdot \rho_0). \qquad (12)$$

Apparently, for the existence of an equilibrium state

$$d - (\rho_1 + \alpha \cdot \rho_0)$$

must be greater than zero. As this expression approaches zero, infinite queueing in front of the lock builds up. Since we assumed an infinite CPU queue, there is always work available for the M CPUs, and no processor degradation results.

The derivation of (9) to (12) makes use of the method of phases described earlier.

To simplify the derivation, we combine the variables B_0 and D_0 with the times T_D and T_W as described in Section 2 to

$$\begin{aligned}
T_{ID} &= (1 - D_0) \cdot T_D, \\
T_{IW} &= (1 - B_0) \cdot T_W. \qquad (13)
\end{aligned}$$

The corresponding Laplace transforms are

$$\begin{aligned}
\phi_{ID}(s) &= d_0 + (1 - d_0) \cdot \phi_D(s), \\
\phi_{IW}(s) &= b_0 + (1 - b_0) \cdot \phi_W(s). \qquad (14)
\end{aligned}$$

Note that d in (6) is equal to $\phi_{ID}(\lambda_0)$. We have

$$\begin{aligned}
E[T_{ID}] &= (1 - d_0) \cdot E[T_D], \\
E[T_{IW}] &= (1 - b_0) \cdot E[T_W],
\end{aligned}$$

and see that C_{IW} of (8) is the coefficient of variation of T_{IW}.

Let $\phi_L(\cdot)$, $\phi_{ID}(\cdot)$, and $\phi_{IW}(\cdot)$ be given in phase expansion:

$$\begin{aligned}
\phi_L(s) &= \sum_{j=1}^{q} a_j \cdot \mu_j/(\mu_j + s), \\
\phi_{ID}(s) &= \sum_{j=0}^{r} d_j \cdot \delta_j/(\delta_j + s), \\
\phi_{IW}(s) &= \sum_{j=0}^{t} b_j \cdot \tau_j/(\tau_j + s). \qquad (15)
\end{aligned}$$

Phases with index 0 in (15) are responsible for immediate dispatches and "no paging exception." They represent instantaneous transitions, *i.e.*, $\tau_0, \delta_0 = \infty$.

The following states span the state space:

0	lock is free, no task is delayed.
$(0, n, i)$	$n \geq 1$, $0 \leq i \leq r$, lock is free, n tasks are delayed, the dispatcher is in phase i with rate δ_i.
$(1, n, j)$	$n \geq 1$, $1 \leq j \leq q$, lock is occupied, lock holder is processing in phase j with rate μ_j, n tasks are delayed.

$(2, n, k)$ $n \geq 1, 1 \leq k \leq t$, lock is occupied, lock holder is in page wait in phase k with completion rate τ_k.

The initial part of the state transition diagram is shown in Fig. 7. The transition rates for $0 \leq i \leq r$, $1 \leq j \leq q$, and $0 \leq k \leq t$ are

$a_j \cdot \lambda_0$ from 0 to $(1, 1, j)$
$a_j \cdot \delta_i$ from $(0, n, i)$ to $(1, n, j)$ for $n \geq 1$
$a_j \cdot \lambda_0$ from $(0, n, i)$ to $(1, n+1, j)$ for $n \geq 1$
λ_1 from $(1, n, j)$ to $(1, n+1, j)$ for $n \geq 1$
$b_k \cdot \mu_j$ from $(1, n, j)$ to $(2, n, k)$ for $n \geq 1$
τ_k from $(2, 1, k)$ to 0
$d_i \cdot \tau_k$ from $(2, n, k)$ to $(0, n-1, i)$ for $n \geq 2$
λ_0 from $(2, n, k)$ to $(2, n+1, k)$ for $n \geq 1$.

We introduce some abbreviations for probabilities and average flows:

$$u_0 = p(0), \quad u_{ni} = p(0, n, i), \quad u_n = \sum_{i=0}^{r} u_{ni},$$
$$u = \sum_{n=0}^{\infty} u_n,$$
$$v_{n,j} = p(1, n, j), \quad v_n = \sum_{j=1}^{q} v_{n,i}, \quad v = \sum_{n=1}^{\infty} v_n,$$
$$w_{n,k} = p(2, n, k), \quad w_n = \sum_{k=0}^{t} w_{n,k}, \quad w = \sum_{n=1}^{\infty} w_n, \tag{16}$$

$$U_n = \sum_{i=0}^{r} \delta_i \cdot u_{n,i},$$
$$V_n = \sum_{j=1}^{q} \mu_j \cdot v_{n,j},$$
$$W_n = \sum_{k=0}^{t} \tau_k \cdot w_{n,k}. \tag{17}$$

With these symbols, the equilibrium equations may be written, for $n \geq 1$, $0 \leq i \leq r$, as

$$\lambda_0 \cdot u_0 = W_1,$$
$$(\delta_i + \lambda_0) \cdot u_{n,i} = d_i \cdot W_{n+1}; \tag{18}$$

for $0 \leq j \leq q$, $n \geq 2$, as

$$(\mu_j + \lambda_1) \cdot v_{1,j} = a_j \cdot \lambda_0 \cdot u_0 + a_j \cdot U_1,$$
$$(\mu_j + \lambda_1) \cdot v_{n,j} = a_j \cdot \lambda_0 \cdot u_{n-1} + a_j \cdot U_n + \lambda_1 \cdot v_{n-1,j}; \tag{19}$$

for $0 \leq k \leq t$, $n \geq 2$, as

$$(\tau_k + \lambda_0) \cdot w_{1,k} = b_k \cdot V_1,$$
$$(\tau_k + \lambda_0) \cdot w_{n,k} = b_k \cdot V_n + \lambda_0 \cdot w_{n-1,k}, \tag{20}$$

We recall the definitions of $A(z)$, $B(z)$, and d in (6); now with the phase expansions of (15),

$$A(z) = \sum_{j=1}^{q} a_j \cdot \mu_j/(\mu_j + \lambda_1 \cdot (1 - z)) = \phi_L(\lambda_1 \cdot (1 - z)),$$
$$B(z) = \sum_{k=0}^{t} b_k \cdot \tau_k/(\tau_k + \lambda_0 \cdot (1 - z)) = \phi_{IW}(\lambda_0 \cdot (1 - z)),$$
$$d = \sum_{i=0}^{r} d_i \cdot \delta_i/(\delta_i + \lambda_0) = \phi_{ID}(\lambda_0), \tag{21}$$

and we define

$$A^*(z) = \sum_{j=1}^{q} a_j/(\mu_j + \lambda_1 \cdot (1 - z)) = (1 - A(z))/\lambda_1 \cdot (1 - z),$$
$$B^*(z) = \sum_{k=0}^{t} b_k/(\tau_k + \lambda_0 \cdot (1 - z)) = (1 - B(z))/\lambda_0 \cdot (1 - z),$$
$$d^* = \sum_{i=0}^{r} d_i/(\delta_i + \lambda_0) = (1 - d)/\lambda_0. \tag{22}$$

The generating functions for the probabilities and flows of (16) and (17) are denoted by

$$G_\beta(z) = \sum_{n=1}^{\infty} \beta_n \cdot z^n, \qquad G_{\beta,j}(z) = \sum_{n=1}^{\infty} \beta_{n,j} \cdot z^n, \tag{23}$$

with $\beta \in \{u, v, w, U, V, W\}$.

With definitions (21) to (23) it is a straightforward matter to derive the following three pairs of equations from the original balance equations (18) to (20):

$$G_u(z) = d^* \cdot \{G_W(z) - z \cdot \lambda_0 \cdot u_0\}/z,$$
$$d \cdot G_u(z) = d^* \cdot G_U(z), \tag{24}$$

$$G_v(z) = A^*(z) \cdot \{z \cdot \lambda_0 \cdot G_u(z) + G_U(z) + z \cdot \lambda_0 \cdot u_0\},$$
$$A(z) \cdot G_v(z) = A^*(z) \cdot G_V(z), \tag{25}$$

$$G_w(z) = B^*(z) \cdot G_V(z),$$
$$B(z) \cdot G_w(z) = B^*(z) \cdot G_W(z). \tag{26}$$

We now show how to derive (25) from (19). Multiply each of the equations for $v_{n,j}(n \geq 1)$ by z^n and sum up over n to obtain

$$(\mu_j + \lambda_1) \cdot G_{v,j}(z) = a_j \cdot \lambda_0 \cdot z \cdot G_u(z) + a_j \cdot G_U(z) + \lambda_1 \cdot z \cdot G_{v,j}(z).$$

Resolving for $G_{v,j}(z)$ and summing over j leads to the first line of (25). The second, as all other relations in (24) to (26), is derived in a similar manner. Equations (24) to (26) form a system of simultaneous linear equations for the $G_\beta(\cdot)$, which is easy to solve by elimination.

G_{XD} of (9) can now be computed as

$$G_{XD}(z) = u_0 + G_u(z) + G_v(z) + G_w(z).$$

Since $G_{XD}(1) = 1$, we obtain for u_0 by applying l'Hôpital's rule

$$u_0 = (d - \rho_1 - \alpha \cdot \rho_0)/d \cdot (1 + \rho). \tag{27}$$

Note that

$$A'(1) = \lambda_1 \cdot E[T_L] = \rho_1,$$

$$B'(1) = \lambda_0 \cdot E[T_{IW}] = \lambda_0 \cdot (1 - b_0)/\tau = \alpha \cdot \rho_0. \tag{28}$$

Here the prime sign indicates derivation with respect to z. We have obtained (9). To derive (10) from (9) we make use of

$$q_L = G'_{XD}(1)$$

and (28).

To derive the lock utilization, we make use of

$$u + v + w = 1 \tag{29}$$

and two global balance equations. The first is

lock request rate = lock release rate, *i.e.*,

$$u \cdot \lambda_0 + v \cdot \lambda_1 + w \cdot \lambda_0 = v \cdot \mu, \tag{30}$$

and

lock release rate = page-in rate, *i.e.*,

$$v \cdot \mu = w \cdot \tau/(1 - b_0). \tag{31}$$

Note that a page transfer requires with a probability b_0 a zero time, while τ is the average transfer rate for true paging exceptions.

From (29) to (31) follows

$$u = 1 - \rho_0 \cdot (1 + \alpha)/(1 + \rho), v = \rho_0/(1 + \rho),$$

$$w = \alpha \cdot \rho_0/(1 + \rho). \tag{32}$$

From (32) we obtain the lock utilization $u_L = 1 - u$ as shown in (11). From the derivation of (11) it follows that (11) is valid for any distribution of T_A and an infinite level of multiprogramming.

The relative "lock-forced" dispatch rate is given by the lock request rate while the lock is occupied $(v \cdot \lambda_1 + w \cdot \lambda_0)$ plus the rate of immediate dispatches at lock release time $((W - W_1) \cdot d_0)$ divided by the total lock request rate

$$r_D = \frac{(v \cdot \lambda_1 + w \cdot \lambda_0 + d_0 \cdot (W - W_1))}{(\lambda_0 \cdot u + \lambda_1 \cdot v + \lambda_0 \cdot w)}.$$

With (26), (27), (32), and the first equation of (28) we obtain (12).

This completes the derivation of the results (9) to (12).

- *One suspend lock, finite N*

In this section we develop an algorithm to compute the queue length distributions for given levels N and M of multiprogramming and multiprocessing. Again, we assume the variable T_A to be exponentially distributed with mean $1/\lambda$. The lock hold time T_L and the page wait time T_W are generally distributed. Now we permit the dispatch time T_D to be dependent on the queue length n, and we indicate this dependence with the notation T_{Dn}. Similarly, we permit the Bernoulli variable D_0, which determines immediate dispatches, to be a function D_{0n} of the queue length. To ease the subsequent computation we combine the variables B_0 and D_{0n} with T_W and T_{Dn} to

$$T_{IW} = (1 - B_0) \cdot T_w,$$

$$T_{IDn} = (1 - D_{0n}) \cdot T_{Dn}. \tag{13a}$$

For the Laplace transforms of T_L and T_{IW} we assume the phase expansion of (15). In analogy, T_{IDn} is assumed to be given by

$$\phi_{IDn}(s) = \sum_{j=0}^{r} d_{j,n} \cdot \delta_{j,n}/(\delta_{j,n} + s) \tag{15a}$$

States are characterized as above. The initial part of the state transition diagram is shown in Fig. 7. Due to the finite level of multiprogramming we obtain now a finite transition diagram with a tail, as shown in Figs. 8 and 9.

The complete set of transitions can easily be inferred from the transition rules given in Section 2 for exponentially distributed times and from the transition rules for infinite N specified above. The only difference for the case of infinite N is in the tail at the time when fewer tasks are in the CPU queue than there are processors available for CPU queue processing. Within the tail no immediate dispatch is necessary since an idle processor immediately picks up one of the tasks waiting for the lock.

The transitions are implicitly contained in the balance equations for the $u_{n,i}$ (18a), $v_{n,j}$ (19a), and $w_{n,k}$ (20a) below. Probabilities and flows are defined as in (16) and (17).

$$u_0 \cdot \lambda_0 = W_1,$$

$$u_{n,i} \cdot (\lambda_0 + \delta_{i,n}) = d_{i,n} \cdot W_{n+1},$$

$$v_{1,j} \cdot (\mu_j + \lambda_1) = a_j \cdot \lambda_0 \cdot u_0 + a_j \cdot U_1,$$

$$v_{n,j} \cdot (\mu_j + \lambda_1) = \lambda_1 \cdot v_{n-1,j} + a_j \cdot \lambda_0 \cdot u_{n-1} + a_j \cdot U_n,$$
$$2 \le n \le N - M \qquad (18a)$$

$$v_{N-M+1,j} \cdot (\mu_j + \lambda_1) = \lambda_1 \cdot v_{N-M,j} + a_j \cdot \lambda_0 \cdot u_{N-M} + a_j \cdot W_{N-M+2},$$

$$v_{N-M+m,j} \cdot (\mu_j + \lambda_m) = \lambda_{m-1} \cdot v_{N-M+m-1,j} + a_j \cdot W_{N-M+m+1},$$
$$2 \le m \le M \qquad (19a)$$

$$w_{1,k} \cdot (\lambda_0 + \tau_k) = b_k \cdot V_1,$$

$$w_{n,k} \cdot (\lambda_0 + \tau_k) = \lambda_0 \cdot w_{n-1,k} + b_k \cdot V_n,$$
$$2 \le n \le N - M. \qquad (20a)$$

In the preceding form the balance equations are dependent on the phase expansion. We subsequently replace the references to phases by introducing generalized Laplace transforms.

From (18a) and (1) to (5) we derive

$$u_0 \cdot \lambda_0 = w_1. \qquad (33)$$

For $1 \le n \le N - M$,

$$u_n = d_n^* \cdot W_{n+1}, \qquad (34)$$

$$U_n = d_n \cdot W_{n+1}, \qquad (35)$$

with $d_n = \phi_{\mathrm{ID}n}(\lambda_0)$, $d_n^* = (1 - d_n)/\lambda_0$.

To eliminate the phase expansion from (19a) we introduce the abbreviations

$$A(k, i, m) = \lambda_1^i \cdot \left(\prod_{i=1}^{m} \lambda_{i+1}\right) \cdot \Gamma_{\mathrm{L}}(k, i, m \,|\, s_i = \lambda_{i+1}), \qquad (36)$$

$$g(k, r, m) = \left(\prod_{j=r}^{m} \lambda_j\right) \cdot \Gamma_{\mathrm{L}}(k, 0, m - r + 1 \,|\, s_i = \lambda_{i+r-1}), \qquad (37)$$

$$E(k, r, n) = \sum_{i=r}^{n} A(k, i, 0) \cdot (\lambda_0 \cdot u_{n-1} + U_{n+1-i}), \qquad (38)$$

$$Q(k, m, n) = \sum_{i=1}^{N-M} A(k, i, m) \cdot (\lambda_0 \cdot u_{N-M+1} + U_{N-M+1-i}) + g(k, 1, m) \cdot \lambda_0 \cdot u_{N-M} + \sum_{j=1}^{n} g(k, j, m) \cdot W_{N-M+1+j}. \qquad (39)$$

It is straightforward to eliminate recursively references to $v_{n,j}$ and $V_{n,j}$ on the right-hand side of the equations (19a). Making use of (36) to (39) we obtain, for $1 \le n \le N - M$,

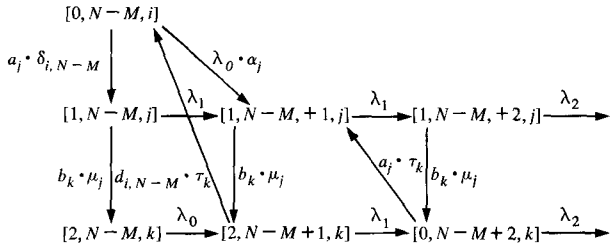

Figure 8 State transition diagram for $M - N \le n < N$.

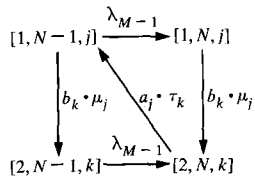

Figure 9 State transition diagram for $N - 1 \le n \le N$.

$$v_n \cdot \lambda_1 = E(1, 1, n), \qquad (40)$$

$$V_n \cdot \lambda_1 = A(0, 1, 0) \cdot (\lambda_0 \cdot u_{n-1} + U_n) + E(0, 2, n); \qquad (41)$$

for $1 \le m < M$,

$$v_{N-M+m} \cdot \lambda_m = Q(1, m, m), \qquad (42)$$

$$V_{N-M+m} \cdot \lambda_m = g(0, m, m) \cdot W_{N-M+1+m} + Q(0, m, m - 1); \qquad (43)$$

for $m = M$,

$$v_N = Q(2, M - 1, M - 1), \qquad (44)$$

$$V_N = Q(1, M - 1, M - 1). \qquad (45)$$

Equations (40) to (45) represent the balance equations (19a) in a form which is independent of the phase expansion. With the abbreviations

$$B(k, i, m) = \lambda_0^i \cdot \left(\prod_{j=1}^{m} \lambda_j\right) \cdot \Gamma_{\mathrm{IW}}(k, i, m \,|\, s_i = \lambda_j), \qquad (46)$$

$$h(k, r, m) = \left(\prod_{j=r}^{m} \lambda_j\right) \cdot \Gamma_{\mathrm{IW}}(k, 0, m - r + 1 \,|\, s_i = \lambda_{i+r-1}), \qquad (47)$$

$$F(k, r, n) = \sum_{i=r}^{n} B(k, i, 0) \cdot V_{n+1-i}, \qquad (48)$$

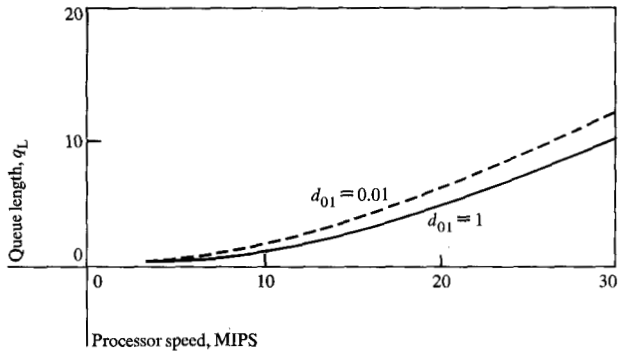

Figure 10 Queue length of a suspend lock with minimal queue length strategy and minimal dispatch strategy (dotted line) as a function of the processor speed in a uniprocessor; $M = 1$, $N = 3$ times processor speed in MIPS, $T_L = 50$, $T_A = 1000$, $T_D = 10\,000$ instructions, $T_W = 10$ milliseconds, and $b_0 = 0.9995$.

$$R(k, m, n) = \sum_{i=1}^{N-M} B(k, i, m) \cdot V_{N-M+1-i} + \sum_{j=1}^{n} h(k, j, m) \cdot V_{N-M+j}, \quad (49)$$

we obtain in a very similar way the new balance equations corresponding to (20a): for $1 \le n \le N - M$,

$$w_n \cdot \lambda_0 = F(1, 1, n), \quad (50)$$

$$W_n \cdot \lambda_0 = B(0, 1, 0) \cdot V_n + F(0, 2, n); \quad (51)$$

for $1 \le m < M$,

$$w_{N-M+m} \cdot \lambda_m = R(1, m, m), \quad (52)$$

$$W_{N-M+m} \cdot \lambda_m = V_{N-M+m} \cdot h(0, m, m) + R(0, m, m-1); \quad (53)$$

for $m = M$,

$$w_N = R(2, M-1, M-1) + E[T_{IW}] \cdot V_N, \quad (54)$$

$$W_N = R(1, M-1, M-1) + V_N. \quad (55)$$

Equations (33) to (35), (40) to (45), and (50) to (55) together with the normalization condition

$$\Sigma(u_n + v_n + w_n) = 1$$

represent a system of homogeneous linear equations which may be solved numerically as indicated in the following scheme:

1. Compute arrays $\mathbf{A}(k, i, m)$, $\mathbf{g}(k, r, m)$, $\mathbf{B}(k, i, m)$, and $\mathbf{h}(k, r, m)$ making use of Eqs. (4), (36), (37), (46), and (47). Note: There is a danger of computational instability, in which case one may have to resort to an actual phase expansion. Then Eq. (5) has to be applied instead of Eq. (4).
2. Define functions $E(k, r, n)$, $Q(k, m, n)$, $F(k, r, n)$, and $R(k, m, n)$ according to Eqs. (38), (39), (48), and (49).
3. Apply the following algorithm:

$u_0 := 1$,

$W_1 := u_0 \cdot \lambda_0$. (33)

For $n = 1$ to $N - M$ compute

V_n using (51)
w_n using (50)
U_n using (41)
v_n using (40)
W_{n+1} using (35)
u_n using (34)
end n

For $m = 1$ to $M - 1$ compute

V_{N-M+m} using (53)
w_{N-M+m} using (52)
$W_{N-M+m+1}$ using (43)
v_{N-M+m} using (42)
end m.

Compute

V_N using (45)
w_N using (54)
v_N using (44).

Notes: 1. Equation (55) may be used to cross check the computations.
2. For $M = 1$ we have $\lambda_1 = 0$. The above algorithm needs some minor modifications to remain applicable to this special case.

4. Multiply all obtained probabilities and flows with a factor to satisfy $\Sigma u_n + v_n + w_n = 1$.
5. The results of interest are obtained as

$$u_L = \sum_{n=1}^{N} (v_n + w_n),$$

$$q_L = \sum_{n=1}^{N-M} n \cdot u_n + \sum_{n=1}^{N} n \cdot (v_n + w_n),$$

$$d_p = \sum_{m=1}^{M} (v_{N-M+m} \cdot (m-1) + w_{N-M+m} \cdot m)/M,$$

$$r_D = D1/D2,$$

where

$$D1 = \sum_{n=1}^{N-M} (\lambda_0 \cdot w_n + \lambda_1 \cdot v_n + d_{0,n} \cdot W_{n+1}),$$

$$D2 = u_0 \cdot \lambda_0 + \sum_{n=1}^{N-M} (\lambda_0 \cdot u_n + \lambda_1 \cdot v_n + \lambda_0 \cdot w_n) + \sum_{m=1}^{M-1} \lambda_m \cdot (v_{N-M+m} + w_{N-M+m}).$$

4. Discussion of results

From a performance point of view, high frequency suspend locks are dangerous. This is already apparent from the

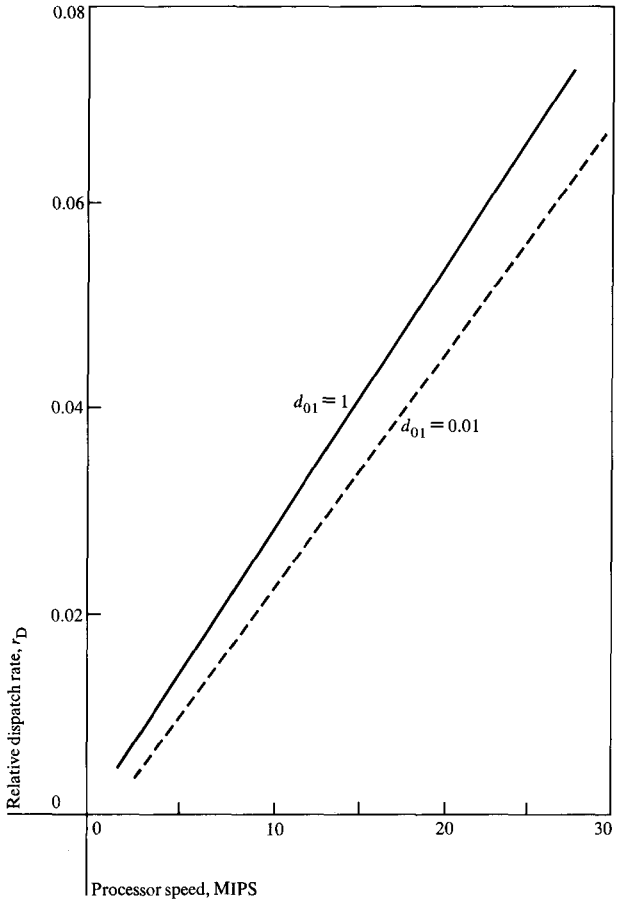

Figure 11 Relative dispatch rate as a function of the processor speed. The parameters are identical to those of Fig. 10.

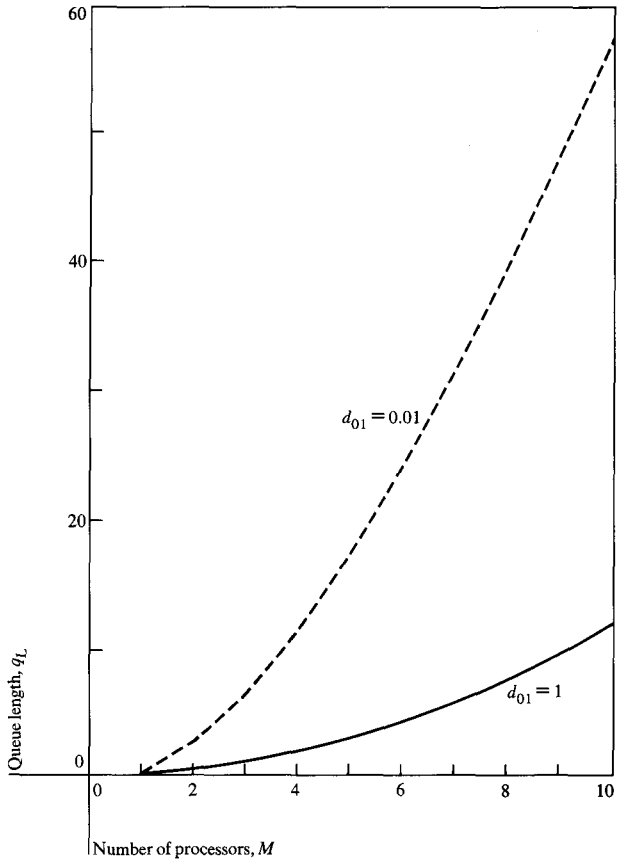

Figure 12 Queue length of a suspend lock as a function of the level of multiprocessing. The parameters correspond to those of Fig. 10, except that processing power is increased by increasing the number of processors; each processor has a power of 3 MIPS.

phenomenon of convoys in front of such locks. Nevertheless, there are situations in which suspend locks cannot be avoided. The subsequent discussion points to conditions under which suspend locks contribute to the degradation of throughput and/or to the increase of response time.

Unlike spin locks, suspend locks do not directly cause processor degradation. However, the additional dispatching overhead caused by suspend locks may already outweigh this advantage. In addition, tasks which are queueing in front of a suspend lock are not available to utilize the real resources of the system (*e.g.*, devices, processors). Queueing in front of suspend locks either requires an increase in the level of multiprogramming, with the consequence of increased response time, or it causes reduced utilization of the system resources. The amount of queueing (*i.e.*, the queue length) in front of suspend locks is therefore as important with respect to performance as the relative dispatch rate (*i.e.*, the number of dispatches per lock request).

A brief discussion of the parameters selected for the illustrations is in order. A high frequency lock is assumed with 1000 instructions outside the critical section and 50 instructions within the critical section if no paging exception occurs. These values are in the range of values observed in real systems [11]. We assume that no paging exception occurs in a critical section with probability 0.9995. This corresponds to one paging exception per 100 000 instructions. For the page transfer time we assume 10 milliseconds. Further, we assume a normal dispatching process to occur every 10 000 instructions and, unless explicitly stated, the minimal queue length strategy. A multiprogramming level of 3 per MIPS is used.

- *Large system effects*

Figures 10 to 13 illustrate the impact of processor performance increases. In the case of spin locks, the increase in the performance of a processor has no impact since only the ratios of processor times are effective. In contrast, suspend

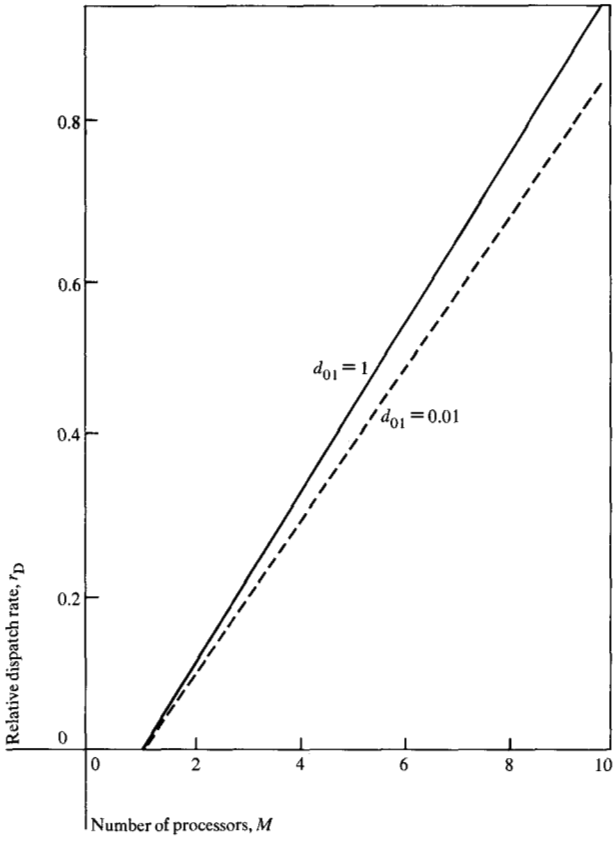

Figure 13 Relative dispatch rate of a suspend lock as a function of the number of processors. The parameters are identical to those of Fig. 12.

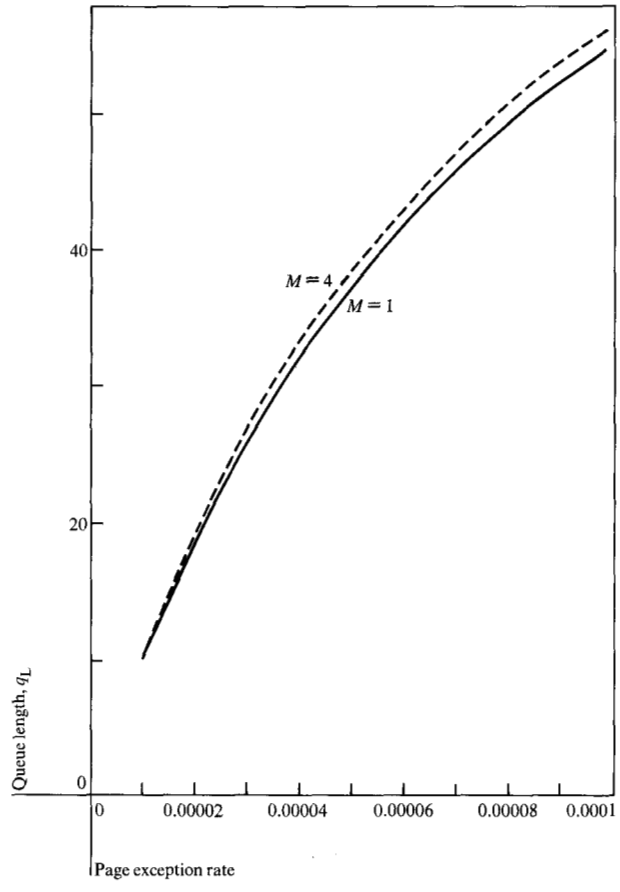

Figure 14 Queue length of a suspend lock as a function of the paging rate for a uniprocessor and for a four-way multiprocessor (dotted line); $N = 90$, processing power is 30 MIPS, $T_A = 1000$, $T_L = 50$, $T_D = 10\,000$ instructions, mixed dispatching strategy ($d_{01} = 0.5$), $T_W = 10$ milliseconds.

locks show an increase of queueing (Fig. 10) and dispatching overhead (Fig. 11) even for a uniprocessor. While the performance of the processor increases from 3 to 30 MIPS, the suspend lock queue contains a growing fraction of the available system tasks. Figure 10 shows clearly that the minimal queue length strategy remains superior to the minimal dispatch strategy, even though, as shown in Fig. 11, the minimal dispatch strategy causes less dispatching overhead.

We refer to effects illustrated in Figs. 10 and 11 as "large system effects" since within the 3-MIPS system the suspend lock is no problem at all, while it becomes more and more of a performance bottleneck with increased processor speed.

Figures 12 and 13 illustrate the large system effects if performance is increased by increasing the number of processors. The situation is very similar to that of Figs. 10 and 11. Each processor has 3 MIPS power, and the number of processors grows from 1 to 10.

A comparison of the multiprocessor case with the uniprocessor case shows that the right kind of dispatching strategy is more important in the multiprocessor situation. The phenomenon is not easy to explain. The expression for the queue length q_L (16) for infinite N contains as third term a factor ρ_1 proportional to $(1 - d)/(d - (\rho_1 + \alpha \cdot \rho_0))$. If $d = 1$ or if $M = 1$ this part of the term is zero. Here ρ_1 is proportional to $M - 1$ and independent of the speed of a single processor. For low values of d, the whole term becomes quickly dominating and increasing with the level of multiprocessing.

- *High paging activity*

Figure 14 shows the dependence of the queue length on the paging activity. The processor performance is 30 MIPS and the multiprogramming level is 90. The left end of the graph corresponds to a paging exception per 100 000 instructions, the right end to a paging exception per 10 000 instructions. While the paging rate has a strong influence on the queue length, the level of multiprocessing has only a small impact.

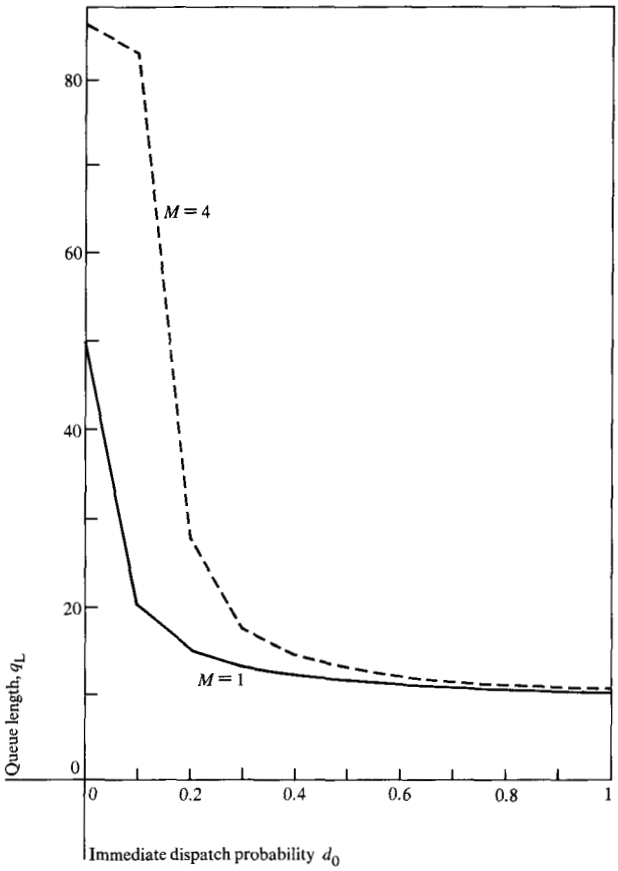

Figure 15 Queue length of a suspend lock as a function of the probability of an "immediate dispatch" for a uniprocessor and a four-way multiprocessor (dotted line); $N = 90$, processing power is 30 MIPS, $T_A = 1000$, $T_L = 50$, $T_D = 10\,000$ instructions, $b_0 = 0.9995$, and $T_W = 10$ milliseconds.

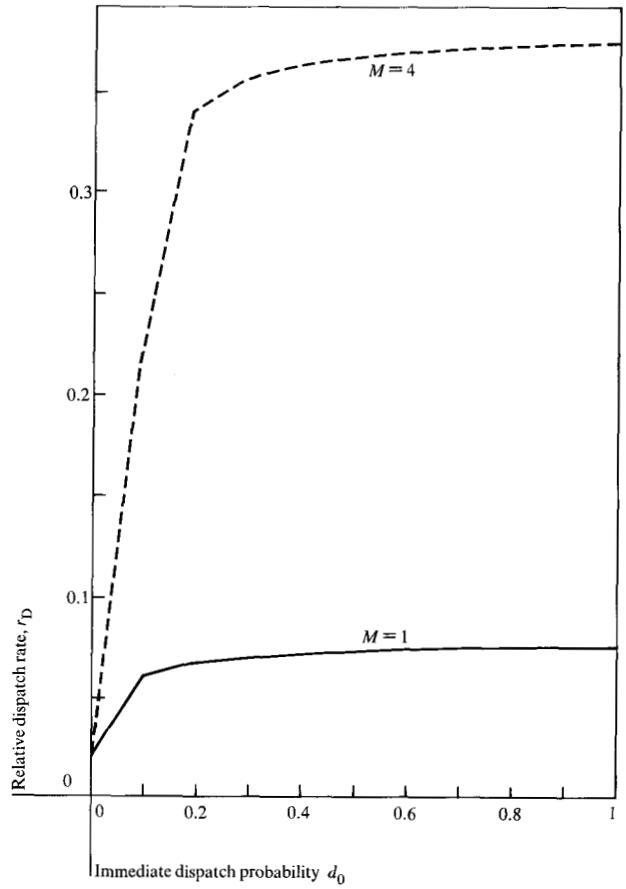

Figure 16 Relative dispatch rate of a suspend lock as a function of the probability of an "immediate dispatch." The parameters are identical to those of Fig. 15.

An increase of the page transfer time has an effect similar to an increase of the paging rate. Both effects show up in thrashing situations.

• *Dispatching strategy*

Figures 15 and 16 show the impact of the dispatching strategy on queueing and relative dispatch rate. Unlike previous figures, d_0 has here been taken to be constant, *i.e.*, independent of the queue size; $d_0 = 0$ corresponds to the exact minimal dispatch strategy, $d_0 = 1$ to the exact minimal queue length strategy. Figure 16 shows that the minimal dispatch strategy causes considerably less dispatching, however, at the expense of excessive queueing. We draw the conclusion that a mixed or pure minimal queue length strategy is preferable.

• *Distributions*

Figure 17 shows the impact of the distribution of T_L on the queue length. The coefficient of variation has little impact. Even in situations with four processors, each capable of 7.5 MIPS, and a multiprogramming level of 90, the queue length only doubles from 1.5 to 3 if the coefficient of variation of the lock hold time increases from 0 to the extreme value of 100. In situations with more queueing, the impact of the distribution is reduced further.

Similarly, the coefficient of variation of the page wait time T_W has very little impact. However, as the expressions for $N \rightarrow \infty$ show, the value of d in (6) depends strongly on the distribution of T_D. Interestingly, a larger coefficient of variation of T_D decreases the queue length in the expression (10) for q_L. This is also confirmed by numerical evaluations for the finite case.

Our results for the suspend lock are in contradiction to the sometimes expressed opinion that large coefficients of variation cause large increases of queue lengths and response times. That is, in general, not true for the suspend lock in a closed queueing network.

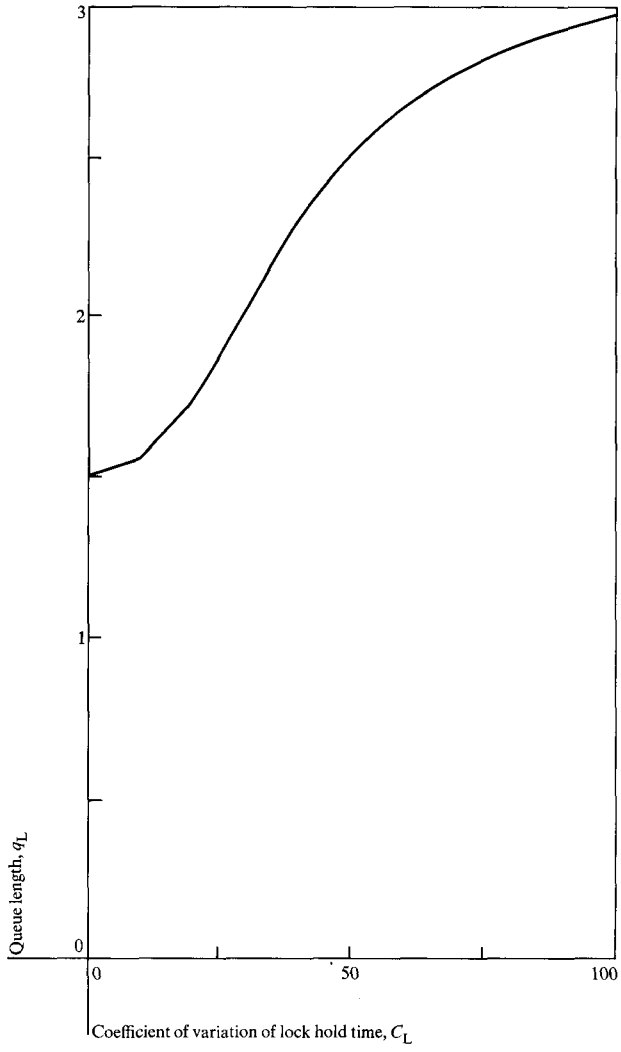

Figure 17 Queue length of a suspend lock as a function of the coefficient of variation of the lock hold time; $M = 4$, $N = 90$, processing power is 30 MIPS, $T_A = 5000$, $T_L = 50$, $T_D = 1000$, mixed dispatch strategy ($d_{01} = 0.5$), and $T_W = 10$ milliseconds.

• *Approximations*

A theory which permits the analysis of systems with many suspend locks is the subject of current research. We are therefore currently unable to derive an approximation for large systems similar to the formula for spin locks in [2]. The only analytical result in closed form is for systems with one lock and an infinite level of multiprogramming (Section 3). The expression for the queue length (10) shows a queue explosion as the expression

$$d - (\rho_1 + \alpha \cdot \rho_0) \tag{56}$$

approaches 0. The value of d is defined in (6) and is determined by the dispatching strategy and the rate of normal dispatching. If we assume minimal queue length strategy or a mixed strategy, then d has a value close to 1. A sufficient condition for the queue length to be small is

$$\rho_1 + \alpha \cdot \rho_0 \ll d. \tag{57}$$

If both quantities ρ_1 and $\alpha \cdot \rho_0$ are small, it is easily seen from (10) and (12) that both queue length and relative dispatch rate remain small compared to 1. The meaning of (57) for multiprocessors is apparent: The time a task spends on average in a critical section should be small compared to the time the task spends in non-critical sections divided by the number of processors.

It should, however, be noted that (57) is a strong condition. A violation of (57) by no means implies excessive queueing. For example, with minimal queue length strategy in the configuration of Fig. 15 with four processors we have $\rho_1 + \alpha \cdot \rho_0 = 0.3$ and approximately 12% queueing in front of the suspend lock. The average contention of 12% of the available tasks for logical resources is undesirable, but certainly not catastrophic.

5. Summary

Probabilistic models of suspend locks have been developed and solved analytically or by algorithms to obtain numerical results. The applied methods are essentially those of queueing theory; however, the method of embedded Markov chains failed for locks, because the probabilities of states of the recurrence points are not representative for the average system behavior. An equally powerful and convenient method, referred to as the "method of phases," has been developed and successfully applied.

The analysis confirmed the experience that high frequency suspend locks are dangerous from a performance point of view. It is known that such locks require a specific dispatching strategy to avoid the disastrous convoy phenomenon. The models show that even with such strategies any of the following events may cause a drastic increase of queues in front of suspend locks:

1. Increase of processor speed,
2. Increase in the number of processors,
3. Increase of the paging rate,
4. Increase of the page transfer time.

Conditions (3) and (4) arise typically in thrashing situations. Conditions (1) and (2) lead to what we call "large systems effects." The slope of increases with any of the above events has a strong dependence on the dispatching strategy. The general conclusion to be drawn from our model with respect to this strategy is as follows: Whenever a lock is released for which another task is waiting, the waiting task should immediately be dispatched.

The lock models developed and applied in this paper suffer a drawback. Locks are considered in isolation with the environment of locks being represented by a single queue. It would be desirable to obtain solutions for models of the complete system, *i.e.*, models representing device queues, processor queues, and lock queues as a single queueing network. We have little hope of being able to solve such models exactly for finite networks. However, we see a chance that the models for infinite systems [15] may be accessible at least for numerical solutions. Currently, one would have to resort to the iterative approximations described in [12, 13].

The models developed in this paper show that queues formed internally by control programs are important objects for performance analysis. The problems which may be caused by such queues grow with the processing power or, equivalently, the level of multiprogramming. Current and anticipated advances in hardware technology increase the importance of a theory of computer system models which encompasses the internal queues caused by software as well as the queues in front of real resources.

References

1. D. C. Gilbert, private communication, IBM Corporation, Poughkeepsie, NY.
2. J. Hofmann and H. Schmutz, "Performance Analysis of Locking in Operating Systems," *Technical Report TR 81.03.003,* IBM Scientific Center, Heidelberg, W. Germany, March 1981.
3. A. N. Habermann, *Introduction to Operating System Design,* Science Research Associates, Chicago, 1976.
4. E. W. Dijkstra, "Cooperating Sequential Processes," *Programming Languages,* F. Genuys, Ed., Academic Press, London, 1968.
5. E. G. Coffmann, M. J. Elphick, and H. Shoshani, "System Deadlocks," *Computing Surv.* **3,** 67–78 (1971).
6. A. N. Habermann, "Prevention of System Deadlocks," *Commun. ACM* **12,** 373–377 (1969).
7. J. W. Havender, "Avoiding Deadlocks in Multi-Tasking Systems," *IBM Syst. J.* **7,** 74–84 (1968).
8. B. Kumar, "Modelling and Analysis of Distributed Software Systems," *Proceedings of the 7th Symposium on Operating System Principles,* Pacific Grove, CA, December 1979, pp. 2–8, ACM Order No. 534790.
9. *OS/VS2 System Logic Library, Vol. 4,* Order No. SY28-0716, available through IBM branch offices.
10. D. C. Gilbert, private communication. IBM Corporation, Poughkeepsie, NY.
11. M. Blasgen, J. Gray, M. Mitoma, and T. Price, "The Convoy Phenomenon," *ACM Oper. Syst. Rev.* **13,** 20–25 (1979).
12. K. M. Chandy, U. Herzog, and L. Woo, "Approximate Analysis of General Queuing Networks," *IBM J. Res. Develop.* **19,** 43–49 (1975).
13. C. H. Sauer and K. M. Chandy, "Approximate Solution of Queuing Models," *IEEE Computer* **13,** 25–32 (1980).
14. H. Kobayashi, *Modelling and Analysis,* Addison-Wesley Publishing Co., Inc., Reading, MA, 1978.
15. H. Stenzel and H. Schmutz, "Approximate Performance Models of Large Computer Systems," *Technical Report TR 81.04.004,* IBM Scientific Center, Heidelberg, W. Germany, April 1981.

Received January 28, 1981; revised June 23, 1981

J. Hofmann is located at Fachhochschule Heilbronn, West Germany, and H. Schmutz is located at the IBM Scientific Center, Heidelberg, West Germany.