SIAM J. COMPUT.
Vol. 31, No. 3, pp. 816–837

ANALYSIS OF TIMING-BASED MUTUAL EXCLUSION WITH RANDOM TIMES*

ELI GAFNI† AND MICHAEL MITZENMACHER‡

Abstract. Various timing-based mutual exclusion algorithms have been proposed that guarantee mutual exclusion if certain timing assumptions hold. In this paper, we examine how these algorithms behave when the time for the basic operations is governed by probability distributions. In particular, we are concerned with how often such algorithms succeed in allowing a processor to obtain a critical region and how this success rate depends on the random variables involved. We explore this question in the case where operation times are governed by exponential and gamma distributions, using both theoretical analysis and simulations.

Key words. mutual exclusion, timed mutual exclusion, Markov chains, locks

AMS subject classifications. 68M14, 68W15, 68W20

PII. S0097539799364912

1. Introduction.

A good design methodology for developing distributed algorithms, as advocated by Liskov [10], is to assume the worst and hope for the best. In assuming the worst, one designs an algorithm which is safe regardless of the amount of time each operation takes. In hoping for the best, one designs the algorithm to optimize some utility function under certain timing assumptions.

A nice example of such a design is the mutual exclusion algorithm of Lynch and Shavit [12]. We describe the algorithm here at a high level; definitions of the relevant terms appear in section 2.1. The Lynch and Shavit algorithm for mutual exclusion is designed to cope with variations in timing of read and write operations. It combines previous mutual exclusion algorithms of Fischer [5] and Lamport [8] in a clever way in order to guarantee mutual exclusion and weak deadlock-freedom, as well as guarantee deadlock-freedom if certain timing constraints are met. Specifically, the algorithm is guaranteed to avoid deadlock if all steps of a process take time in a fixed range $[c_1, c_2]$. Given these timing constraints, specific pauses depending on the bounds c_1 and c_2 are added into the program for each process; these pauses ensure deadlock-freedom. Note that deadlock-freedom comes at a price, namely, the introduction of pauses that delay the completion of operations. In practice, detecting deadlock and breaking it are very costly in terms of time, and therefore a good design should ensure that deadlock never or rarely happens.

It is reasonable to assume that hard timing constraints will rarely or never be violated in the case of interprocess communication through a standard shared memory, such as when processes are running on machines in the same room. The gap between

*Received by the editors December 16, 1999; accepted for publication (in revised form) May 15, 2001; published electronically December 18, 2001. A preliminary version of this work appeared in *Proceedings of the* 18*th Annual Symposium on Principles of Distributed Computing*, Atlanta, GA, 1999, pp. 13–21.

http://www.siam.org/journals/sicomp/31-3/36491.html

†UCLA Computer Science Department, 3731 F Boelter Hall, Los Angeles, CA 90024-1596 (eli@cs.ucla.edu). Part of this author's work was done while visiting Compaq Systems Research Center. This author was supported by grant 4-592560-19914 from the UCLA Council on Research.

‡Maxwell Dworkin Laboratory 331, 33 Oxford Street, Harvard University, Computer Science Department, Cambridge, MA 02138 (michaelm@eecs.harvard.edu). Most of this author's work was done while employed at Compaq Systems Research Center. This author was supported in part by an Alfred P. Sloan Research Fellowship and NSF CAREER grant CCR-9983832.

the minimum and maximum memory reaction time is likely to be sufficiently small enough that pauses based on these timing constraints will generally yield only a small performance penalty. With the rise of fast networks and the Internet, however, there are alternative situations where processors may communicate through a much slower and more variable shared memory medium. For example, interprocess communication can be accomplished via servers reading and writing shared disk pages from a shared farm of disks accessible over a network. The mechanisms for using shared disks in this manner exist today and are described in several works on storage area networks [3, 4, 9, 14]. Indeed, storage area networks offer a shared memory that is cheap, reliable, and large; moreover, with regard to the design of distributed algorithms, the physical model of this architecture is close to the abstract model of shared memory. Operations on a disk-based shared memory might be slow and have large variance; moreover, its timing may not be well understood. Making hard timing assumptions that are guaranteed to hold may entail prohibitively long timeouts or self-delays for practice.

An alternative application that we envision involves multiple processes interacting via the Internet, such as in an auction on eBay. In such a scenario, the number of processes interacting may be extremely large. Also, while operations on shared memory may be instantaneous, users cannot expect response times on the order of shared memory systems, since Internet progagation delay will dominate.

Therefore, we are motivated to expand the analysis of the performance of mutual exclusion algorithms based on shared memory to systems that can potentially have long delays, so that the bounded timing model is not applicable. In many cases, even when timing bounds may prove problematic, knowledge of the distribution of operation times may be possible through systematic study. Consequently, we suggest introducing a probabilistic analysis of mutual exclusion algorithms under random delays.

Besides the above motivation, once we considered the idea of probabilistic analysis, it occurred to us that randomized algorithms for mutual exclusion may be more efficient than previous algorithms even in the context of fast shared memories. Instead of having algorithms introduce deterministic pauses designed for the worst case in order to guarantee mutual exclusion, using shorter pauses with random times may lead to better practical performance. The hope is that smaller random delays will avoid deadlock often enough that it will be more efficient to use small random delays and a mechanism for breaking deadlock than a slower deadlock-free algorithm. This approach may allow tradeoffs between correctness properties and efficiency.

A further motivation for introducing probabilistic models into this area is simply to gain more insight into the features of these algorithms. In particular, our analysis demonstrates that an appropriate pause (even one that lasts a random time) can dramatically change an algorithm's behavior.

We further note that the probabilistic framework we introduce is reminiscent of similar work on contention resolution in multiaccess channels. The contention resolution framework has proven highly successful. (See the notes in [6] or references from [7] or [13].) We suspect that this direction may therefore prove worthwhile in the context of mutual exclusion or other distributed algorithms as well. For example, since the publication of the original version of this paper, a similar probabilistic framework was used by Aspnes to study a deterministic consensus algorithm against an adversary who cannot control random timing noise introduced by the system [2].

In this paper, we focus on the case where operation times have the exponential

distribution. This distribution has properties which prove handy for analysis. Moreover, although the assumption of exponential distributions is not correct in practice, algorithms that behave well under the exponential distribution are generally assumed (whether correctly or not) to behave well under "reasonable" distributions. Thus they make an appropriate starting point for this analysis. We also examine the case where operation times have a gamma distribution, both to offer more insight and to avoid the problem of drawing conclusions specific to the exponential distribution.

We refer to the basic unit of much of our analysis as a *lock*. Loosely speaking, for our purposes a lock is a shared variable that can be inspected (or read, to see if it is clear), written (to attempt to take control), and read (to see if control has been obtained). A processor successfully passes through a lock if it finds it clear on inspection, writes its processor ID to it, and reads back its processor ID. Note that a processor may pause, or self-delay, between any of these steps. A lock is a basic unit in Fischer's mutual exclusion algorithm [5], which we describe in section 2.1. Studying locks provides us with the means and insight to study variations on the algorithm of Lynch and Shavit [12].

We are interested in answers to questions such as the following:

1. How often do locks succeed, and how does this depend on the underlying distributions?
2. Are we better off with one lock with a long pause or two consecutive locks with smaller pauses?
3. How should lock constructions be combined in this setting?

In this paper, we focus on the analysis of the basic lock construction and explore the behavior of these locks and some of our questions with simulations. As a by-product of our work, we explore the behavior of several simple but interesting Markov chains. We believe that further, more detailed analysis of these Markov chains would be interesting, not only because of their connection to timed mutual exclusion algorithms, but also in and of themselves.

Because we focus on the simple lock mechanism, the analysis in this version of the paper is essentially self-contained. However, we encourage the interested reader to peruse the work by Lynch and Shavit on timing-based mutual exclusion [12] for more details on Lamport's algorithm, Fischer's algorithm, and their combination, in order to put this work in context.

2. Background.

2.1. Definitions. For completeness, we describe the basic definitions associated with the mutual exclusion problem. Here we generally follow the definitions and notation of [12]. (See also [11] for extensive references and related work.)

A mutual exclusion algorithm arbitrates among n sequential threads of control, or processes. Processes communicate by reading and writing in some form of shared memory. Read and write operations on this memory are assumed to execute instantaneously; that is, they happen atomically on memory locations. The process itself, however, might not obtain the result of the read or write until some future time, depending on the architecture of the system. For our purposes, the program associated with each process has a form as given in Figure 1. In particular, a process has an associated *critical region*. A system is said to *satisfy mutual exclusion* if in any reachable system state at most one user is in its critical region. Note that the *trying region* and the *exit region* are used to coordinate entry to and exit from the critical region; the *remainder region* is where all other work is done.

```
Basic process
p: current process index

repeat forever:
  remainder region
  trying region
  critical region
  exit region
end repeat;
```

FIG. 1. *Basic process program.*

Two other properties are useful to consider. A system is said to be *weakly deadlock-free* if when any single process's trying region is concurrent only with the remainder regions of other processes, then its trying region terminates, and similarly if when any single process's exit region is concurrent only with the remainder region of other processes, then its exit region terminates. This property corresponds to the requirement that if a process runs alone, it accesses the critical region. The stronger property of being *deadlock-free*, which corresponds to the requirement that the system progress, requires that

- if some process is in the trying region and no process is in the critical region, then subsequently some process enters the critical region; and
- if some process is in the exit region, then subsequently some process enters the remainder region.

The algorithm of Lynch and Shavit relies on Lamport's fast mutual exclusion algorithm [8] to guarantee that mutual exclusion is never violated. (See Figure 2.) It also relies on Fischer's timed mutual exclusion algorithm [5] to provide Lamport's algorithm the environment it requires for deadlock-freedom, namely, a single contender. (See Figure 3.) We discuss the combined algorithms in section 4. Proofs of these properties appear in [12].

Note the appearance of a pause in Fischer's timed mutual exclusion algorithm. The point of the pause is as follows: suppose each *step* of a process, corresponding to a line of code, takes time bounded between $[c_1, c_2]$ for some positive finite values c_1 and c_2. Then if the pause time corresponds to at least $\lceil c_2/c_1 \rceil$ steps (using for example no-op operations), so that the pause takes time at least c_2, then Fischer's algorithm guarantees both mutual exclusion and deadlock-freedom.

2.2. Properties of the exponential distribution. Recall that a random variable that is exponentially distributed with mean μ is defined by its probability density function, $f(x) = (1/\mu)\mathrm{e}^{-x/\mu}$. The exponential distribution proves convenient for theoretical study because of its special properties. We briefly note these properties here and make use of them without further reference throughout this paper.

- *Memoryless property.* Suppose that the time until an event is determined by an exponential random variable with mean μ. Given that the event has not yet happened, the remaining time until the event happens is still an exponential random variable with mean μ.
- *Minimum property.* Suppose that the times until each of k events are determined by independent exponential random variables with mean μ. Then the time until the first of these events occurs is exponential with mean $\frac{\mu}{k}$.

Lamport
x, y: shared registers, initially 0
p: current process index

% Entering ME-lock
L:
$x := p$;
if $y \neq 0$ then goto L;
$y := 1$;
if $x \neq p$ then goto L;
enter critical region;
exit critical region;
$y := 0$;
% Exiting ME-lock

FIG. 2. *Lamport style mutual exclusion.*

Fischer
x: shared register, initially 0
p: current process index

% Entering ME-lock
L:
if $x \neq 0$ then goto L;
$x := p$;
pause
if $x \neq p$ then goto L;
enter critical region;
exit critical region;
$x := 0$;
% Exiting ME-lock

FIG. 3. *Fischer's timed mutual exclusion algorithm.*

- *Fairness property.* Suppose that the times until events A and B are determined by independent exponential random variables with means μ_1 and μ_2, respectively. Then event A occurs first with probability $\frac{\mu_2}{\mu_1+\mu_2}$.

2.3. How many pass through? We begin by considering a basic unit for mutual exclusion algorithms, namely, a *lock*. A lock access protocol consists of an *inspect* phase (which is an initial read of the shared variable that comprises the lock), a *write* phase, and a final *read* phase. A processor inspects the lock to see if it is clear; it attempts to write its processor ID to the lock; and then it passes through the lock successfully if it reads its own ID. A processor that successfully passes through the lock eventually clears the shared variable so that others may pass through; until this occurs, the processor is said to own the lock. Mutual exclusion is guaranteed as long as no two processors believe they own the lock at the same time. Recall that a lock is the mechanism behind Fischer's algorithm, as seen in Figure 3. Also, Fischer's mutual exclusion algorithm also allows for pauses. We begin our analyses without considering the effect of a pause; however, we return to consider the pause later in the paper.

We will often compare the behavior of a lock with a *double lock*, by which we mean two successive back-to-back locks, each with its own shared variable. When a processor passes through the first lock of a double lock, it then begins the inspection phase for the second lock of the double lock. A processor is said to own a double lock only after it has passed through the second lock, and mutual exclusion is guaranteed as long as no two processors believe they own the lock at the same time. A natural question we consider here is whether using two short locks in a double lock might be better than using a long single lock.

We denote the three phases by I, W, and R, respectively. In this section, unless otherwise stated, we assume that the times for each of these actions are exponentially distributed, with means i, w, and r respectively, where the values of i, w, and r are fixed constants (independent of the number of processors in the system). For convenience, we scale so that $w = 1$ unless otherwise noted.

We emphasize that an operation is meant to take place *atomically* (that is instantaneously, from the point of view of the processes) at the end of the time interval corresponding to the operation. That is, the fact that operations take time to complete is not to suggest that they do not take place atomically, but only that there is a delay between when an operation is initiated by a processor and when it completes. One way to view this model is that operations initiated by a processor are scheduled in some way, say, on a shared disk system. The scheduling causes a random delay between when an operation is initiated and when it is completed. A processor sees the results of an operation as soon as it is completed.

We begin by presenting some simple arguments regarding how many processors complete successive stages of a lock in the face of contention. These arguments do not answer our main question, which is how often just one processor successfully obtains a lock in the face of contention. They do, however, introduce the flavor of our arguments and provide some initial insight.

THEOREM 1. *Consider a situation where n processors begin inspecting a free lock at the same time. Then, with probability bounded below by some constant, at least $\Omega(\sqrt{n/i})$ processors complete the inspection stage before the first write completes.*

Remark. The assumption that the processors begin at the same time is for convenience; since all times are exponentially distributed, as long as a write has not occurred, we may take any instant when n processors are in the I stage as the beginning.

Proof. We derive a recursive function p_j describing the probability that at least j processors successfully inspect the lock before the first write. Suppose that jth inspection has just completed, and no writes have yet occurred. Then the time until the next inspection completes is exponentially distributed with mean $i/(n-j)$, as there are $n-j$ processors remaining. The time until the first write completes is exponentially distributed with mean $1/j$, as there are j processors attempting a write. Hence, the probability that another inspection completes before the first write is $\frac{n-j}{ij+n-j}$. Recursively, then, we have $p_1 = 1$ and $p_{j+1} = p_j \frac{n-j}{ij+n-j}$.

Let $z = \sqrt{n/i}$. Then

$$\begin{aligned} p_{z+1} &= \prod_{1 \le j \le z} \frac{n-j}{ij+n-j} \\ &= \prod_{1 \le j \le z} \left(1 - \frac{ij}{ij+n-j}\right) \end{aligned}$$

$$\geq \prod_{1 \leq j \leq z} \left(1 - \frac{ij}{(1-\epsilon)n}\right)$$

for an ϵ that goes to 0 as n gets large. Hence,

$$p_{z+1} \geq \prod_{1 \leq j \leq z} \left(1 - \frac{ij}{(1-\epsilon)n}\right) \geq \left(1 - \frac{1}{(1-\epsilon)z}\right)^z,$$

which is arbitrarily close to $\mathrm{e}^{-1/(1-\epsilon)}$ for sufficiently large n. This demonstrates that with at least some constant probability, at least $\Omega(\sqrt{n/i})$ processors complete the I stage. □

It is easy to extend the proof of Theorem 1 to show that the expected number of processors that complete their I stage before the first write is actually $\Theta(\sqrt{n/i})$.

THEOREM 2. *Consider the setting of Theorem* 1. *The expected number of processors that complete the inspection stage before the first write is* $\Theta(\sqrt{n/i})$.

Proof. The lower bound follows from Theorem 1. For the upper bound, note that the expected number of processors to complete the inspection stage before the first write is $\sum_{m \geq 1} p_m$. Let $z = \sqrt{n/i}$; then for any integer $k \geq 1$, for y such that $zk < y \leq z(k+1)$,

$$p_y = \prod_{1 \leq j \leq y-1} \frac{n-j}{ij+n-j} \leq \prod_{z \leq j \leq y-1} \frac{n}{ij+n} \leq \left(1 + \frac{1}{z}\right)^{-(k-1)z} < 2^{-k+1}.$$

It follows that $\sum_{m \geq 1} p_m < 3\sqrt{n/i}$. □

In fact, asymptotically exact formulae can be found with some work. We demonstrate this for the case $i = 1$, which yields an interesting result, although the same technique applies for other cases. When $i = 1$, we have $p_k = \prod_{1 \leq j \leq k-1} \frac{n-j}{n}$, and the expected number of processors that complete the I stage before the first write is $E_I = \sum_{k=1}^{n} p_k$. Consider plotting the points $((k-1)/n, np_k)$ in the first quadrant of the Euclidean plane for $k = 1, \ldots, n$. The area under the successive axes-parallel rectangles defined by these points equals the desired expectation E_I. Moreover, the area of these rectangles approximates the area under a curve passing through these points. Defining a curve that passes through these points is difficult, but we can find a curve that nearly passes through these points quite easily. Consider moving from $(x, y) = ((k-1)/n, np_k)$ to $(k/n, np_{k+1})$. Note that as we move $\Delta x = 1/n$ on the x-axis, the corresponding y-value drops by $\Delta y = -(x + \Delta x)y$. Hence, our points are well approximated by the curve defined by the differential equation $dy/dx = -nxy$ and the boundary condition $y(0) = n$. This curve is just $y = n\mathrm{e}^{-nx^2/2}$. The area under the curve is

$$\int_0^1 n\mathrm{e}^{-nx^2/2} dx = \sqrt{\frac{\pi n}{2}} + O(1).$$

Hence, if n processors begin an I stage, then (up to lower order terms) on average $\sqrt{\frac{\pi n}{2}}$ processors complete their inspection before the first write occurs.

This argument for $i = 1$ can be formalized by noting that

$$p_k = \prod_{1 \leq j \leq k-1} \frac{n-j}{n} \leq \prod_{1 \leq j \leq k-1} \mathrm{e}^{-j/n} = \mathrm{e}^{-k(k-1)/2n} \leq \mathrm{e}^{-(k-1)^2/2n}.$$

It easily follows that $\sum_{k=1}^{n} p_k$ is bounded above by

$$1 + \int_0^n \mathrm{e}^{-x^2/2n} dx = \int_0^1 n\mathrm{e}^{-nx^2/2} dx + O(1).$$

Similarly, using $1 - x \geq \mathrm{e}^{-x-x^2}$ for $0 \leq x \leq 1/2$, we have for $k \leq n/2$

$$p_k = \prod_{1 \leq j \leq k-1} \frac{n-j}{n} \geq \prod_{1 \leq j \leq k-1} \mathrm{e}^{-j/n - j^2/n^2} \geq \mathrm{e}^{-k(k-1)/2n - k(k-1)(2k-1)/6n^2}$$
$$\geq \mathrm{e}^{-k^2/2n - k^3/3n^2}.$$

It follows that $\sum_{k=1}^{n} p_k$ is bounded below by $\int_0^{1/2} n\mathrm{e}^{-nx^2/2 - nx^3/3} dx + O(1)$, and it can be checked that this is equal to $\int_0^1 n\mathrm{e}^{-nx^2/2} dx + O(1)$. (For example, split the integral into two parts, the first covering the range $[0, n^{-5/12}]$ and the second $[n^{-5/12}, 1/2]$. The cubic term is lower order in the exponent in the first range and can be absorbed in the $O(1)$. Similarly, the exponential term in the second range is small enough to be absorbed in the $O(1)$, which also explains why the difference between integrating to $1/2$ and integrating to 1 can be dismissed.)

THEOREM 3. *Consider a situation where n processors begin to write to a lock at the same time. Then on average $\Theta(\ln(rn)/r)$ read their own value, and in fact $\Theta(\ln(rn)/r)$ read their own value with probability $1 - o(1)$.*

Proof. The time between the jth and $(j+1)$st write is exponentially distributed with mean $1/(n-j)$. Hence, the probability that the processor that makes the jth write reads its own value is

$$\frac{\frac{1}{n-j}}{r + \frac{1}{n-j}} = \frac{1}{r(n-j)+1}.$$

The expected number of processors that read their own value is therefore

$$\sum_{j=1}^{n} \frac{1}{r(n-j)+1}.$$

When $r = 1$, this is simply $\sum_{j=1}^{n} 1/j = H(n) \approx \ln n$. Otherwise, bounding the sum by appropriate integrals we have

$$\int_{x=1}^{n} \frac{1}{xr+1} dx \leq \sum_{j=1}^{n} \frac{1}{r(n-j)+1} \leq 1 + \int_{x=0}^{n} \frac{1}{xr+1} dx,$$

and hence

$$\frac{\ln(rn+1)}{r} - \frac{\ln(r+1)}{r} \leq \sum_{j=1}^{n} \frac{1}{r(n-j)+1} \leq 1 + \frac{\ln(rn+1)}{r}.$$

The argument can be easily extended to show that the number of processors that read their own value is $\Theta(\ln n)$ with high probability. Let X_j be the event that the processor that makes the jth write reads its own value. Under the assumption of exponentially distributed read and write times, the X_j are independent. Letting

$X = \sum_{j=1}^{n} X_j$, we may use the standard Chernoff bound (see, for example, Corollary A.14 of [1])

$$\Pr(|X - E[X]| \geq \epsilon E[X]) \leq 2\mathrm{e}^{-\epsilon^2 E[X]/3}.$$

Hence, for any fixed r the probability of X deviating from the mean by more than $\epsilon E[X]$ falls inverse polynomially in n, proving the theorem. □

From Theorems 1 and 3 we immediately obtain as a corollary that two locks are significantly better than one, in terms of the number of processors that can get through (in the case of no pauses). Specifically, for a single lock with all times having the same mean, $\Theta(\sqrt{n})$ processors inspect the free lock before a write occurs with constant probability. Of these processors, with high probability $\Theta(\ln \sqrt{n}) = \Theta(\ln n)$ then read their own values and hence pass through the lock. For a double lock, from Theorem 3, with high probability $\mathrm{O}(\ln n)$ get through the first lock, and hence with high probability at most $\mathrm{O}(\ln \ln n)$ pass through the second. Note that changing the mean times for the I, W, or R operations (while keeping them constant) changes only these expressions by constant factors, and hence this remains true even if the average time to pass through the lock is the same in both scenarios. Hence, in the face of sufficiently large contention, double locks are much better with regard to the number of processors that pass through (on average, with no pauses).

2.4. How often does one pass through? Showing that on average fewer processors pass through a double lock than a long single lock does not really answer our question of which is better. The proper measure of performance is how often a lock successfully allows only one processor through. We now focus on this variable. First, we show that for a single lock with exponentially distributed read and write times (and no pause), a single lock can perform quite poorly under high contention.

THEOREM 4. *Consider a single lock with n processors beginning a write at the same time. The probability that just a single processor reads its own value is $O(\sqrt[r]{1/rn})$.*

Proof. We begin with the case $r = 1$. Recall from Theorem 3 that the jth processor to write reads its own value with probability $1/(r(n-j)+1)$ and that all such events can be treated as independent. Clearly, the last processor to write will read its own value. The probability that it is the only one to do so is

$$\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\cdots\left(1 - \frac{1}{n}\right) = \frac{1}{2}\frac{2}{3}\cdots\frac{n-1}{n} = \frac{1}{n}.$$

Thus, when $r = 1$, the probability that only one processor believes it obtains the lock is $1/n$. For a general r, this probability is

$$\begin{aligned}\prod_{j=1}^{n-1}\left(1 - \frac{1}{r(n-j)+1}\right) &\leq \prod_{i=1}^{n-1}\mathrm{e}^{-1/(r(n-i)+1)} \\ &= \mathrm{e}^{-\sum_{i=1}^{n-1} 1/(r(n-i)+1)} \\ &\leq \mathrm{e}^{-1-(\ln(rn)+1)/r},\end{aligned}$$

and the last term is $O(\sqrt[r]{1/rn})$. □

The result of Theorem 4 demonstrates how the probability of success increases with r and decreases with n. Although increasing r substantially increases the probability of just one processor successfully obtaining the lock, as n grows large, for any fixed r this probability falls to 0.

We now consider the probability of exactly one processor taking control of a double lock. Under a reasonable assumption, we find that in this case, the probability that a single processor obtains the lock is bounded below by a constant, *regardless of how n grows.* This result is somewhat surprising, given the previous result for a single lock.

In this setting, we adopt the following assumption: once a processor passes through the second lock, it will hold that lock for a reasonably long amount of time. Hence, if one processor writes to the second lock before any others read it, we assume that this processor does not clear the second lock until well after all others read that it has possession. This assumption simplifies the problem, as now we need to consider only the problem of whether one processor writes to the second lock before any others read it. It is also reasonable, since a lock is held long enough so that the critical region can be executed.

THEOREM 5. *Let n processors begin a write for a first lock of a double lock at the same time. Then with probability bounded below by some constant, one processor writes to obtain the second lock before any other processors successfully pass through the first lock.*

Proof. The intuition behind the theorem is relatively simple. With some constant probability, one lucky processor passes through the first lock quickly. It then writes to obtain the second lock before any other lucky processors can pass through the first lock. We now formalize this intuition. We first consider the case where $i = r = w = 1$ for convenience. Also, we assume all relevant quantities are integers and avoid floor and ceiling notation for convenience as well.

The jth processor to write passes through the first lock with probability $\frac{1}{n-j+1}$. Hence the probability that none of the first $n/2$ processors passes through the first lock is

$$\prod_{j=1}^{n/2}\left(1-\frac{1}{n-j+1}\right)=\frac{n-1}{n}\frac{n-2}{n-1}\cdots\frac{n/2}{n/2+1}=\frac{1}{2}.$$

Similarly, the probability that exactly one of the first $n/2$ processors passes through the first lock is

$$\sum_{j=1}^{n/2}\left[\frac{\frac{1}{n-j+1}}{1-\frac{1}{n-j+1}}\prod_{k=1}^{n/2}\left(1-\frac{1}{n-k+1}\right)\right]=\frac{1}{2}\sum_{j=1}^{n/2}\frac{1}{n-j}\geq\frac{\ln 2}{2}-o(1).$$

Now suppose exactly one processor from the first $n/2$ passes through the first lock; let it be the jth to write. We now lower bound the probability this processor writes to obtain the second lock before any other processor passes through the first lock. To do so, this processor must complete both an I and W operation. Since all operation times are exponential, with constant probability both these operations complete before the $(7n/8)$th processor completes its write to the first lock. This is clear since with probability 1/2, the I operation occurs before 1/2 of the remaining $n-j$ writes to the first lock. Assuming this happens, with probability 1/2 again, the second W operation completes before 1/2 the remaining writes to the first lock. Hence, with the probability 1/4, the jth processor finishes the I and W operation for the second lock by the time processor $j+(n-j)/2+(1/2)(n-(j+(n-j)/2))$ writes to the first lock. Since $j \leq n/2$, we have that with constant probability the jth processor finishes the I and W operation for the second lock by the time the $(7n/8)$th

processor writes to the first lock. Now, however, by the same argument as previously, the probability that no processors from the $(n/2)$nd to the $(7n/8)$th finish their first write and pass through to the second lock is

$$\prod_{i=n/2+1}^{7n/8}\left(1-\frac{1}{n-i+1}\right)=\frac{1}{4}.$$

Because of the memorylessness of the exponential distribution, all of these events can be treated as independent, and hence with probability bounded below by some constant a single processor successfully writes to the second lock as in the statement of the theorem.

When r and i are fixed constants other than 1, the same argument suffices; various constants in the argument must be changed to reflect the change in r and i. We sketch the required changes. The jth processor passes through the first lock with probability $\frac{1}{1+r(n-j)}$. Hence, the probability that none of the first $n/2$ processors passes through the first lock is

$$\prod_{j=1}^{n/2}\left(1-\frac{1}{r(n-j)+1}\right).$$

We may bound this by noting $1-x\geq \mathrm{e}^{-x-x^2}$ for $0\leq x\leq 1/2$. Hence,

$$\begin{aligned}\prod_{j=1}^{n/2}\left(1-\frac{1}{r(n-j)+1}\right)&\leq\prod_{j=1}^{n/2}\mathrm{e}^{-1/(r(n-j)+1)-1/(r(n-j)+1)^2}\\&=\mathrm{e}^{\sum_{j=1}^{n/2}-1/r(n-j)}(1-o(1))\\&=\mathrm{e}^{-(H(n-1)-H(n/2))/r}(1-o(1))\\&=2^{-1/r}(1-o(1)).\end{aligned}$$

Similarly, the probability that exactly one such processor passes through the first lock is

$$\begin{aligned}\sum_{j=1}^{n/2}\left[\frac{\frac{1}{r(n-j)+1}}{1-\frac{1}{r(n-j)+1}}\prod_{k=1}^{n/2}\left(1-\frac{1}{r(n-k)+1}\right)\right]&=2^{-1/r}\sum_{j=1}^{n/2}\frac{1}{r(n-j)}(1-o(1))\\&=\frac{2^{-1/r}\ln 2}{r}(1-o(1)).\end{aligned}$$

Now suppose exactly one processor passes through the first lock. For this processor to write to obtain the second lock before any other processor passes through the first lock, it must complete both an I and W operation. Since all operation times are exponential, with constant probability both these operations complete before the (αn)th processor completes its write for some constant α depending on i. However, the probability that no processors from the $(n/2)$nd to the (αn)th finish their first write and pass through to the second lock is

$$\prod_{j=n/2+1}^{\alpha n}\left(1-\frac{1}{r(n-j)+1}\right),$$

which can be bounded above and below by some constant independent of n. Hence, again with probability bounded below by some constant, a single processor successfully writes to the second lock as in the statement of the theorem. □

The rather loose analysis of Theorem 5 greatly underestimates the probability that a single processor successfully writes to the second lock before all others. The true probabilities are best determined by simulations, and hence we return to this question in section 5.

We also note that another way to gain better insight into the exact probability that a single processor successfully passes through the double lock is to consider the underlying Markov chain. For instance, this chain can easily be represented as a six-dimensional Markov chain, where each dimension tracks the number of processors in each state. Examining this Markov chain could lead to provable bounds on various probabilities associated with the lock's behavior. Of course, a complete analysis of this complex chain appears rather difficult. We therefore feel that our intuitive proof, combined with simulation results, is a natural approach to the problem.

Given that two locks have a different behavior than one, one might naturally ask whether three (or more) locks have a different behavior than two. Using induction and the above results one can show that using $2k$ consecutive locks, the probability of more than one processor successively passing through is at most γ^k for some constant γ. Could the behavior be even better than exponentially decreasing? We show that the answer to this question is negative by considering the limiting case where just two processors start together at the first lock.

THEOREM 6. *Consider two processors starting at a sequence of k locks. Then the probability that both processors pass through the final lock is at least β^k for some constant β depending on i, w, and r.*

Proof. We first show that the probability that two processors "follow each other" through the lock is a constant. That is, consider the following sequence of events:

1. Both processors inspect the lock before either writes.
2. The first processor to complete a write to the lock reads back its value before the other processor completes its write to the lock.
3. The second processor to complete a write to the lock reads back its value before the other processor inspects the subsequent lock.

If these events occur, because of the memorylessness property of the exponential distribution, the two processors are then in a similar state as though they had both just begun competing for the subsequent lock. It is clear that the intersection of these events hold with constant probability. In fact, by the fairness property and the memorylessness property, we can calculate that all of these events occur with probability β, where

$$\beta = \frac{w}{i+w}\frac{w}{r+w}\frac{i}{i+w}\frac{i}{i+r}.$$

By induction, the probability of both processors passing together through k consecutive locks is at least β^k. □

Thus, for any number of successive locks in this setting, the best one can hope for is a failure probability that decreases exponentially in the number of locks.

3. The gamma distribution. While the previous section, in which we considered exponential random variables, showed that a double lock is better than a single lock, the results must of course be taken in context. Since we know that in the case where all times are deterministic (and, for example, all operations require the same

time) that a single lock is sufficient, it becomes interesting to consider how strongly this behavior depends on the underlying distribution. We offer some insight into this problem by considering the *gamma distribution*. Recall that a gamma distribution is the sum of a number of independent exponential random variables (of the same mean). For example, a gamma(2) distributed random variable with mean 1 is the sum of two exponential random variables, each with mean 1/2.

We show that for a gamma(2) distribution, the probability that only a single processor obtains a single lock is bounded below by a constant independent of n, the number of processors contending for the lock. Hence, in this case, a single lock behaves more like a double lock under the exponential distribution.

The intuition behind this performance is as follows. Consider the case where n processors are initiating the write stage for the lock at the same time. We may think of the write phase for a processor as consisting of two subphases, each corresponding to an exponentially distributed amount of time. Let us say that a processor is *half-done* with the write stage if it has completed its first subphase, *done* or *completed* if its write is fully complete, and *unstarted* if it is not yet even half-done. Before the first processor to complete a write finishes the write, several processors will be half-done. The number of processors half-done with their write are very likely to prevent this first processor from reading its value, for it is very likely that one of these half-done processors will complete its write before this processor can finish its read. This situation, where half-done writes overwrite completed writes before the corresponding read finishes, is likely to occur until few processors remain to complete their writes. When there are few processors remaining, it is possible for a read to complete before the processor value is overwritten, but this happens only with constant probability.

We present the above argument more formally in the theorem below for the case where reads and writes execute with the same average time. For convenience, we take this mean to be 2.

THEOREM 7. *Consider n processors beginning the write for a single lock, where the times for writes and reads have independent gamma(2) distributions with mean 2. Then a single processor reads its own value with probability bounded below by some constant.*

Proof. We assume that n is sufficiently large throughout. We wish to show that only the last processor to write its value reads its own value with constant probability. The proof is divided into three parts, corresponding to the beginning, the middle, and the end of the process.

For the beginning, we wish to show that with constant probability, by the time the first write completes, with constant probability there are at least $6\sqrt{n}$ processors that are half-done. This will ensure that sufficiently many half-done processors are around to block the completion of any write for all but the end of the process. Consider the time until the first write completes. Let p_j be the probability that at least j processors are at least half-done by this point. By the same argument as Theorem 1, $p_1 = 1$ and $p_{j+1} = p_j \frac{n-j}{n}$. It is straightforward to use this recurrence in a manner similar to Theorem 1 to show that at least $6\sqrt{n}$ are half-done when the first write completes with constant probability. (Note that, if we wished to bound this probability, we might do better to consider explicitly the behavior of the process until the first few writes complete; however, for our purposes the above is sufficient.)

For the middle, we show that with constant probability, conditioned on the fact that at least $6\sqrt{n}$ half-done processors exist at the time the first write completes, there are always many half-done processors until the very end of the process. Explicitly,

we claim that with constant probability there are always at least $2\sqrt{n}$ processors half-done with their writes as long as there are at least $10\sqrt{n}$ unstarted processors. This is easily seen by making a stochastic comparison with the number of half-done processors and a simple random walk. When there are u unstarted processors and h half-done processors, the probability that h increases (and u decreases) is $\frac{u}{h+u}$, and the probability that h decreases (and u stays the same) is $\frac{h}{h+u}$. Moreover, because all distributions are exponential, each step is independent. In particular, when $u \geq 10\sqrt{n}$ and $h < 10\sqrt{n}$, the number of half-done processors h is biased upwards.

Now consider an unbiased random walk that starts at $6\sqrt{n}$ with boundaries at $2\sqrt{n}$ and $10\sqrt{n}$ that runs for $2n$ steps. We claim that it is more likely that h is at least $2\sqrt{n}$ until there are $10\sqrt{n}$ unstarted processors than that this unbiased random walk reaches the boundary $2\sqrt{n}$. This follows from a standard stochastic domination argument; the value h also begins at $6\sqrt{n}$, it changes less than $2n$ times, and it is always more likely to increase than the unbiased random walk. Standard results in probability theory now yield that the random walk (and hence h) stays above $2\sqrt{n}$ with constant probability.

This is most easily seen by noting that for the walk to reach $2\sqrt{n}$, it must fall $2\sqrt{n}$ in either the first n or the last n steps. Let $X_i = 1$ if a walk of n steps goes up on the ith step, and let $X_i = -1$ if it goes down on the ith step. Then from Theorem A.1 of [1], which is derived in a manner similar to Chernoff bounds, the probability that $\sum_{i=1}^{n} X_i \leq 2\sqrt{n}$ is at most e^{-2}. By a union bound, the probability that the walk falls $2\sqrt{n}$ in either the first n or the last n steps is at most $2\mathrm{e}^{-2}$.

Now, conditioned on all of the above, up to the point where there are $10\sqrt{n}$ unstarted processors, with constant probability no processor will read its own value. For to do so, any such processor must complete two read phases before any of the half-done writes complete. In each case the probability of doing so is $(\frac{1}{2\sqrt{n}})^2 = \frac{1}{4n}$, and hence by the union bound with probability at least $\frac{3}{4}$ no processor to this point reads its own value.

We clarify that this statement follows using conditional probabilities, not a union bound. That is, we define the following: let A be the event that $6\sqrt{n}$ processes are half-done when the first write completes. Let B be the event that at least $2\sqrt{n}$ processes are half-done as long as there are at least $10\sqrt{n}$ unstarted processes. Let C be the event that, up to the point where there are $10\sqrt{n}$ unstarted processes, no processor reads its own value. Then

$$\Pr(C) \geq \Pr(C|B) \cdot \Pr(B|A) \cdot \Pr(A),$$

and we have shown that all of the above on the right-hand side are constants.

We now need to consider the end of the process. To see what happens toward the end of the process, consider what would happen if the system began with all processors half-done with their writes. The jth processor to complete its write would then successfully read its own value if it completed two read phases before any of the half-done writes completed, which occurs with probability $(\frac{1}{n-j+1})^2$. Hence, the probability that any processor other than the last to write would read its own value would be at most $\sum_{j=1}^{n-1}(\frac{1}{n-j+1})^2 < \frac{6}{\pi^2}$. (We elaborate on this in Theorem 8.)

In the actual process, we have already seen that all behaves well up to the point when there are $10\sqrt{n}$ unstarted processors. After this point, we claim the system behaves similarly to one where all remaining processors begin half-done with their writes. Specifically, we show that at the last point in time when there are k processors left unstarted, there are at least $k \log n/2$ processors left with probability $1 - o(1)$ for

all k from 1 to $10\sqrt{n}$. This implies that at the end of the process, we always have that most processors are half-done, which will suffice.

Let us consider the specific case where $k = 1$. The probability that the $(n-j)$th processor to become half-done has not yet completed at the first time when there is one processor left unstarted is just $1/(j+1)$. Hence, the expected number of half-done processors at the point where there is just one unstarted processor is approximately $H(n) \approx \log n$. Moreover, the events (that the jth processor to become half-done has not yet completed) are independent, so we may apply Chernoff bounds. Hence, we find the probability that there are not at least $\log n/2$ half-done processors is at most $1/n^{1/16}$.

We may attack larger k similarly. The probability that the $(n-j)$th processor to become half-done has not yet completed while there are k processors left unstarted is just $k/(j+1)$. Hence, the expected number of half-done processors at the point where there are k unstarted processors is approximately $k(H(n) - H(k)) \approx k \log(n/k)$. For $k \leq \log n$, a Chernoff bounds yields that the probability that there are not at least $k \log(n/k)/2$ half-done processors is at most $1/n^{1/16}$. For $\log n \leq k \leq 10\sqrt{n}$, Chernoff bounds yield that the probability that there are not at least $k \log(n/k)/2$ half-done processors is at most $n^{-\log n/8}$. Using a union bound, we find that, for all k from 1 to $10\sqrt{n}$, at the last point in time when there are k process left unstarted, there are at least $k \log(n/k)/2$ half-done processors left with probability $1 - o(1)$.

Let us temporarily assume that this is the case. Let $u(j)$ be the number of unstarted processors when the jth processor to write completes its write. Then the probability that the jth processor to write reads its own value is at most

$$\left(\frac{1}{n-j-u(j)+1}\right)^2.$$

We are interested only in the situation when $u(j) \leq 10\sqrt{n}$. Hence, the probability that some processor reads its own value when $u(j) \leq 10\sqrt{n}$ is bounded above by

$$\sum_{j=1}^{n-1} \left(\frac{1}{n-j-\min(10\sqrt{n}, u(j))+1}\right)^2.$$

Note that summing to $n-1$ is clearly overcounting. Also, with high probability, for $j = n - \log n/2$ to $n-1$ the value of $u(j)$ is 0. It follows that with high probability the above sum is just

$$\sum_{j=1}^{n-1} \left(\frac{1}{n-j+1}\right)^2 + o(1) < \frac{6}{\pi^2} + o(1),$$

where the $o(1)$ term corrects for the $\min(10\sqrt{n}, u(j))$ term.

To show that this suffices, let D be the event that, from the point where there are $10\sqrt{n}$ unstarted processes, no processor except the last reads its own value. Let E be the event that for all k from 1 to $10\sqrt{n}$, when there are k processors left unstarted, there are at least $k \log(n/k)/2$ processors left. Finally, let S be the successful event that no processor except the last reads its own value.

Then

$$\Pr(S) = \Pr(D \wedge C) \geq \Pr(D \wedge C \wedge E) = \Pr(D|C \wedge E) \cdot \Pr(C \wedge E).$$

The argument regarding the end of the process shows that $\Pr(D|C \wedge E)$ is bounded below by a constant. Also, $\Pr(C \wedge E)$ is bounded below by a constant, since $\Pr(C)$ is and $\Pr(E)$ is $1 - o(1)$. Hence, $\Pr(S)$ is bounded below by a constant and the theorem is proven.

To summarize, we find that in the very beginning no processors pass through the lock with high probability, and several processors become half-done with their writes. Conditioned on this, with constant probability the number of processors half-done with their writes remains high, and hence no processors pass through the lock in the middle. Finally, at the end, with high probability we are always in a state where "almost all" of the processors are half-done. By combining all of the conditioning appropriately, we find that no processor except the last passes through the lock at the end with probability bounded below by a constant. □

The proof of Theorem 7 is somewhat limiting, in that the read and write times are taken to be equal, and in practice one may desire a different initial state, such as when all processors start at the inspect phase. It appears that the theorem above should hold for more general cases; however, writing an appropriate generalization appears difficult. Finding a more elementary proof therefore remains an interesting question.

Theorem 7 has an interesting implication. Because a gamma(2) distribution is just the sum of two exponential distributions, we could easily turn a setting with exponentially distributed read and write times into one with gamma(2) distributed read and write times. Each read and write operation would simply be preceded by a "dummy" read or write operation. If the operations are uncorrelated, this effectively changes the distributions from exponential to gamma(2). Although this doubles the average time to obtain a lock, it changes the probability that a single processor successfully accesses the lock from a diminishing function of the number of processors n to something bounded below by a constant.

In fact, the dummy read or write operations are equivalent to a pause operation, where a pause takes a random amount of time. In Fischer's algorithm, only the read and not the write operation is delayed in this manner. It is therefore natural to now consider the case of Fischer's algorithm, where all operation times are exponential and there is a pause before the final read.

THEOREM 8. *Consider n processors beginning the write for a single lock, where writes and reads have independent exponential distributions with mean* 1, *and there is a pause before each final read of time that is also independent and exponentially distributed with mean* 1. *Then a single processor reads its own value with probability* $\frac{n+1}{2n}$.

Proof. For the jth processor to complete its write to read its own value, the corresponding pause and read operation must occur before any other writes occur. This happens with probability $(\frac{1}{n-j+1})^2$. Hence, all but the last processor to write fail to pass through the lock with probability

$$\begin{aligned}\prod_{j=2}^{n}\left(1-\frac{1}{j^2}\right) &= \prod_{j=2}^{n}\frac{j^2-1}{j^2}\\ &= \frac{\prod_{j=2}^{n}(j-1)\prod_{j=2}^{n}(j+1)}{\prod_{j=2}^{n}j\prod_{j=2}^{n}j}\\ &= \frac{n+1}{2n}.\end{aligned}$$

□

Theorem 8 demonstrates the importance of the pause operation in the context of Fischer's algorithm in the case of exponentially distributed operation times. The pause leads to a completely different type of behavior, avoiding conflict in the critical region over half of the time.

It is worth noting also that the approach to lower bound the failure probability for multiple locks from Theorem 6 can be extended to the case where operation times have the gamma distribution as well. Again, we just imagine two processors following each other through the proper stages and use the properties of the exponential distributions. Hence, the best we could hope for is a failure probability that decreases exponentially with the number of sequential locks.

4. Two protocols. We now apply some of the previous results in considering the performance of two mutual exclusion algorithms first suggested by Lynch and Shavit [12]. Both provide mutual exclusion and weak deadlock-freedom.

The first protocol we consider, given in Figure 4, is the combined Fischer–Lamport algorithm presented as Algorithm 3 in [12]. It uses two registers. We also consider an algorithm using three registers also discussed in [12] that is obtained by directly replacing the critical region of Fischer's algorithm with a Lamport style algorithm for mutual exclusion, as shown in Figure 5.

The scheme using three registers (FL2) behaves similarly to a double lock. The first lock is represented by the x register, and the second "lock" consists of both the y and z registers. Hence, with exponential service times, even without a pause, we would expect a constant probability for some processor to successfully execute the critical region on each trial. The logic is the same as that of Theorem 5; one fortunate early processor passes through the lock represented by register x and then reaches the critical region before another processor can block it.

The scheme using two registers behaves essentially like a single lock on the register x with the additional register y to ensure that only a single processor enters the critical region. It follows immediately from Theorem 8 that if the operation times are independently and exponentially distributed (including the pause), then a single

FL1
x, y: shared registers, initially 0
p: current process index

% Entering ME-lock
L:
if $x \neq 0$ then goto L;
$x := p$;
pause
if $x \neq p$ then goto L;
if $y \neq 0$ then goto L;
$y := 1$;
if $x \neq p$ then goto L;
enter critical region;
exit critical region;
$y := 0$;
$x := 0$;
% Exiting ME-lock

FIG. 4. *A clever Fischer–Lamport combination.*

FL2
x, y, z: shared registers, initially 0
p: current process index

% Entering ME-lock
L:
if $x \neq 0$ then goto L;
$x := p$;
pause
if $x \neq i$ then goto L;
$y := p$;
if $z \neq 0$ then goto L;
$z := 1$;
if $y \neq p$ then goto L;
enter critical region;
exit critical region;
$z := 0$;
$x := 0$;
% Exiting ME-lock

FIG. 5. *A direct Fischer–Lamport combination.*

processor passes through the x lock and hence successfully executes the critical region with probability bounded below by some constant. Similarly, it is easy to show that the probability of a processor obtaining the critical region goes to 0 as the number of processors increases when the pause is removed. We formalize this explicitly.

THEOREM 9. *Consider n processors beginning at L in the algorithm FL*1 *of Figure* 4. *If writes and reads have independent exponential distributions with mean* 1, *and the pause takes time* 0 *(i.e., no pause), then the probability that any processor enters the critical region is $o(1)$.*

Proof. As usual, we assume that n is sufficiently large throughout. First, we note that with high probability $(1-o(1))$, at least $\Omega(\sqrt[3]{n})$ of the n processors starting at L reach the write step, as can be seen using the argument of Theorem 1 with $z = n^{1/3}$. We may therefore assume that we begin with $m = \Omega(\sqrt[3]{n})$ processors at the write stage.

We derive two bounds. The first shows that processors that complete the write to x early are unlikely to reach the critical region, and the second shows that processors that complete the write to x late are unlikely to reach the critical region.

The jth processor to write its own value in register x must read back its value, read register y, write register y, and read its own value again before any other processor writes to register x to obtain the critical region. By now familiar reasoning, the probability of all of these events occurring is $1/(m-j+1)^4$. By the union bound, the probability that any of first $m - m^{1/3}$ processors that write to register x read back its own value is

$$\sum_{j=1}^{m-m^{1/3}} \frac{1}{(m-j+1)^4} = o(1).$$

For the second bound, we consider the final $m^{1/3}$ processors that write their values into register x. Note that the jth processor to write its own value in register x can

reach the critical region only if no processor writes the value 1 on register y before this processor can read the register y. Consider the first $m - 5m^{1/3}$ processors. We claim that with probability $1 - o(1)$, one of these processors writes a 1 on register y before any of the final $m^{1/3}$ processors to write into register x reads register y.

By the same argument as Theorem 4, the probability that none of the first $m - 5m^{1/3}$ reads back its value from register x and proceeds to write to register y is

$$\left(1 - \frac{1}{(5m^{1/3}+1)}\right) \cdots \left(1 - \frac{1}{m}\right) = \frac{1}{5m^{1/3}} = o(1).$$

Hence, with probability $1-o(1)$ at least one processor attempts a write to y. Consider any such processor. For it to fail to write before the $(m - m^{1/3})$rd write to x, either y must have already been written over with a 1 (in which case we are done), or one the following events must occur:

- the read of y must occur after the $(m - 3m^{1/3})$rd write to x;
- the read of y occurs before the $(m - 3m^{1/3})$rd write to x and the write to y occurs after the $(m - m^{1/3})$rd write to x.

Since all operation times have the same mean, the probability of the first event is at most $1/2m^{1/3}$, and the probability of the second event is at most $1/2m^{1/3}$. By a union bound, the probability y still holds the value 0, for any of the last $(m - m^{1/3})$ writes is thus only $o(1)$.

Hence, considering both cases, a processor successfully enters the critical region with probability only $o(1)$. □

We note that we have not attempted to optimize the bounds of Theorem 9. A tight analysis would be interesting.

5. Simulations. In this section, we present the results of simulations of locks and double locks with varying service times, as well as examine the performance of some mutual exclusion algorithms that use locklike structures. The goal of this section is to demonstrate that our previous theorems accurately describe perceived performance, as well as gain more insight into the actual performance of mutual exclusion algorithms under these distributions.

We simulated single and double locks using operation times with an exponential distribution, a gamma(2) distribution, and a gamma(3) distribution. For the double lock, all operations have the same mean time, which we scale to be 1. For the single lock, we have simulated two cases: one where all operations have the same mean time, and one where the final read operation has mean 4, so that the total average time for a lock to try a processor is the same as that for a double lock. We call this a *long lock*. Each data point represents the fraction of 10,000 trials for which a single processor successfully passed through the lock.

The results are presented in Figure 6. We point out some features of interest. As expected, we find that a double lock dramatically outperforms a single lock in the case of the exponential distribution. Moreover, the poor performance of a single lock as the contention grows is clear. For the gamma distributions, however, the single lock performance does not deteriorate with contention, as expected. With a gamma(3) distribution, a single long lock outperforms a double lock.

Interestingly, the behavior as the number of processors increases is different for the three distributions. For the exponential distribution, the probability of success appears to decrease monotonically in the number of processors, while for the gamma(3) distribution the probability appears to increase monotonically in the number of processors. Meanwhile, for the gamma(2) distribution, the probability is nonmonotonic

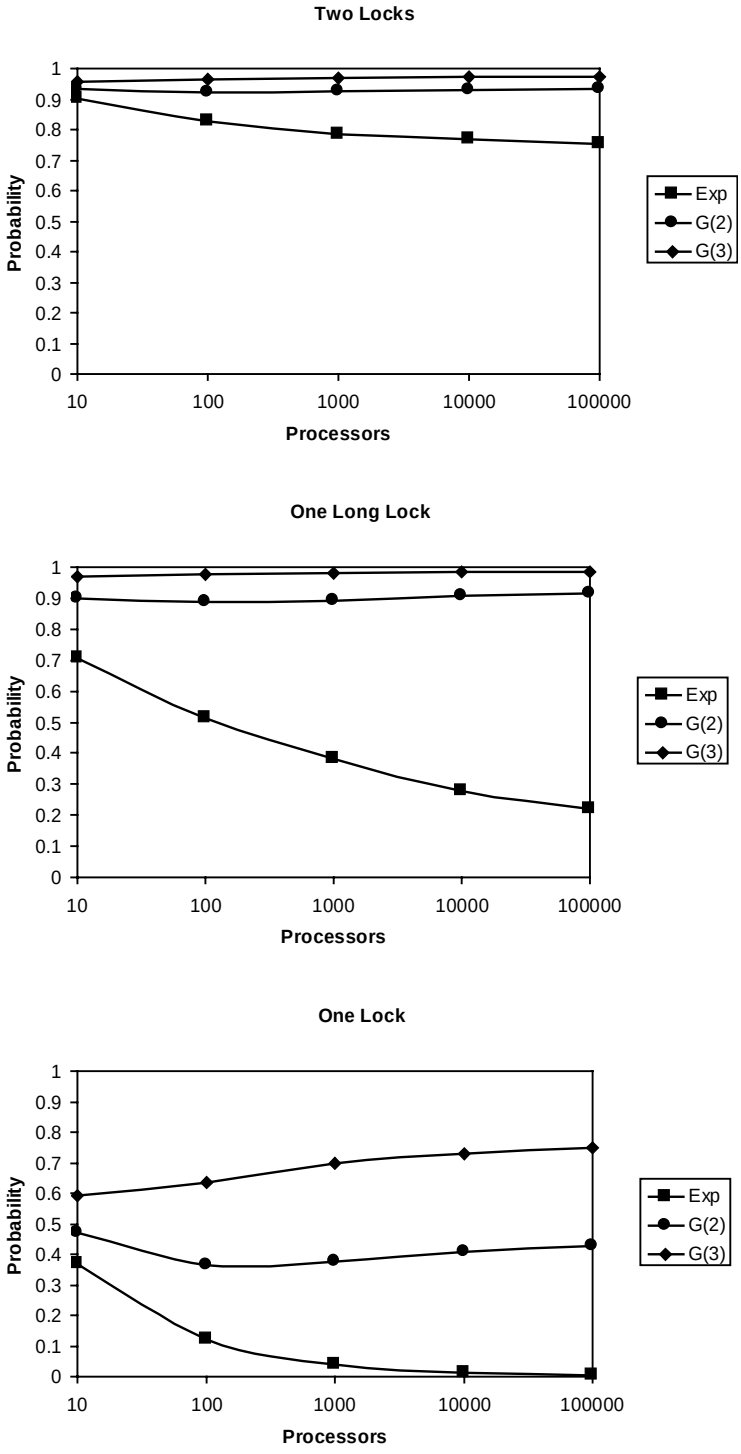

FIG. 6. *Comparing the behavior of a single lock and a double lock.*

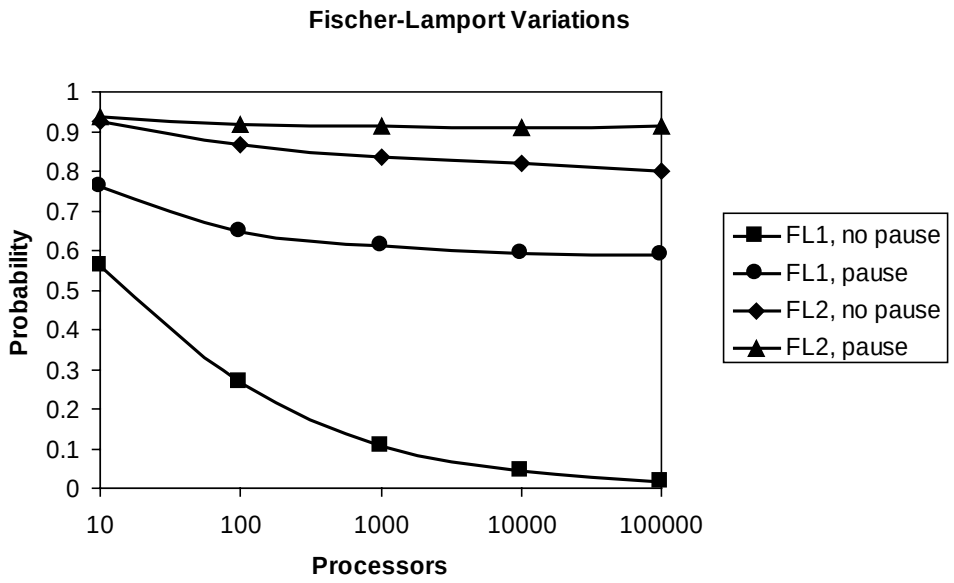

FIG. 7. *Comparing combined mutual exclusion algorithms.*

in the number of processors. This behavior may be worthy of future study, if only as a mathematical curiosity.

We also present some results for the mutual exclusion algorithms of section 4 in Figure 7. For these results, the distribution of the time for all operations is taken to be exponential with mean 1.

Note the dramatic effect of the pause in the performance of FL1. This is not surprising, given the analysis of section 4. Also, note that with the pause the FL1 algorithm succeeds a little more than 1/2 of the time. A rough approximation of this behavior is derivable from Theorem 8. Slightly over 1/2 of the time, a single processor will pass through the first lock. When multiple processors pass through the first lock, sometimes one will reach the critical region before any other processor can block it; this accounts for the additional probability of success. The mutual exclusion algorithm FL2 performs better, but of course it uses an extra register, and on average more time, since more reads and writes are performed by each processor. Tighter analyses or exhaustive simulations of the behavior of these algorithms might lead to a better comparison. It seems difficult to develop a more general statement as to which algorithm is preferable, as the decision may simply depend on the underlying timing distributions.

6. Conclusions and open questions. We have examined the behavior of timed locks under simple distributions, including exponential and gamma distributions, using both theoretical analysis and simulations. In particular, we have focused on the question of whether two locks are better than one and shown how it may depend on the distribution of the completion time of operations. We have also considered how this affects the design of mutual exclusion algorithms. Our work represents the first step toward designing a mutual exclusion algorithm based on random times that offers better performance in realistic situations than algorithms designed for the worst case.

We believe there are several ways to extend this work. A better understanding of the Markov chains underlying double or more extensive sequences of locks would

be interesting. For example, it would be appealing to determine with some accuracy the probability that only one processor passes through a double lock (even if only in the limiting case) by analyzing the underlying Markov chain in a more careful manner. Also, it would be worthwhile to understand the behavior of timed locks under more general distributions. In particular, truncated distributions where events occur within some bounded period of time may provide a more realistic description of actual behavior. Situations where the read and write times are somehow correlated may also be more realistic.

REFERENCES

[1] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[2] J. Aspnes, *Fast deterministic consensus in a noisy environment*, in Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, 2000, pp. 299–308.

[3] D. Attanasio, M. Butrico, J. Peterson, C. Polyzois, and S. Smith, *Design and Implementation of a Recoverable Virtual Shared Disk*, IBM Research Report, RC 19843, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1994.

[4] P. Cao, S. B. Lim, S. Venkataraman, and J. Wilkes, *The tickerTAIP parallel RAID architecture*, ACM Trans. Comput. Systems, 12 (1994), pp. 236–267.

[5] M. Fischer, *personal communication* from [12].

[6] L. Goldberg, *Contention resolution notes*, available at http://www.dcs.warwick.ac.uk/∼leslie/contention.html.

[7] L. Goldberg and P. MacKenzie, *Analysis of backoff protocols for contention resolution with multiple servers*, J. Comput. System Sci., 58 (1999), pp. 232–258.

[8] L. Lamport, *A fast mutual exclusion algorithm*, ACM Trans. Comput. Systems, 5 (1987), pp. 1–11.

[9] E. K. Lee and C. A. Thekkath, *Petal: Distributed virtual disks*, in Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, 1996, pp. 84–92.

[10] B. Liskov, *Practical uses of synchronized clocks in distributed systems*, Distrib. Comput., 6 (1993), pp. 211–219.

[11] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann, San Francisco, 1996.

[12] N. Lynch and N. Shavit, *Timing based mutual exclusion*, in Proceedings of the Annual IEEE Real-Time Symposium (RTSS), Phoenix, AZ, 1992, pp. 2–11.

[13] P. Raghavan and E. Upfal, *Stochastic contention resolution with short delays*, SIAM J. Comput., 28 (1998), pp. 709–719.

[14] C. A. Thekkath, T. Mann, and E. K. Lee, *Frangipani: A scalable distributed file system*, in Proceedings of the 16th ACM Symposium on Operating Systems Principles, 1997, pp. 224–237.