Analysis of Timing Based Mutual Exclusion with Random Times

Eli Gafni * 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 random distributions. In particular, we are concerned with how often such algorithms succeed in allowing a processor to obtain a critical section and how this success rate depends on the random variables involved. Correspondingly, we consider how to maximize system throughput. We explore this question in the case where operation times are governed by exponential and gamma distributions, using both theoretical analysis and simulations.

1 Introduction

A good design methodology for developing distributed algorithm, as advocated by Liskov [7], 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 [8]. The algorithm relies on Lamport's fast mutual exclusion algorithm [5] to guarantee that the safety condition of exclusion is never violated. It relies on Fischer's timed mutual exclusion algorithm [4] to provide Lamport's algorithm the environment it expects, namely a single contender. The algorithm is guaranteed to succeed if certain hard timing constraints are met; that is, all steps of a process must take time in a fixed range $[c_1, c_2]$. If Fischer's algorithm fails because the appropriate hard timing constraints are violated, then deadlock might occur, but two processors will never access a critical section at the same time.

In this paper, we consider how to design mutual exclusion algorithms under probabilistic assumptions on the timing of steps, rather than hard timing constraints. One motivation for this direction lies in new storage area network (SAN) architectures [1, 3, 6, 9]. The SAN architecture consists of servers sharing a "disk farm" over a network. Conceptually, with regard to the design of distributed algorithms, the physical model of this architecture is close to the abstract model of shared-memory, and consequently shared-memory solutions become practical. However, the timing behavior of disks are not well understood, and making hard-timing assumptions which are guaranteed to hold may entail prohibitively long time-outs or self-delays for practice.

Consequently, probabilistic analysis of mutual exclusion algorithms appear called for. In this paper, we focus on the case where step times have the exponential distribution. This distribution has properties which prove handy for analysis, and moreover algorithms that behave well under the

*UCLA Computer Science Department, E-mail: `eli@cs.ucla.edu` Part of this work was done while visiting Compaq Systems Research Center.

†Compaq Systems Research Center, Palo Alto, CA. E-mail: `michaelm@pa.dec.com`.

Contact: michaelm@pa.dec.com or eli@cs.ucla.edu; Track: Long presentation; Student paper : No.

exponential distribution are customarily assumed to behave well under "reasonable" distributions. We also examine the case where step 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.

We are interested in answers to questions such as:

1. How often do Fischer-style locks succeed, and how does this depend on the underlying distributions.

2. Are we better off with one Fischer-style lock with a long pause, or two consecutive Fischer-style locks with smaller pauses?

3. How should the Fischer-Lamport constructions be combined in this setting? (Nested? Sequential?)

4. In light of a recent novel contention-sensitive "collect" algorithm [2], we can follow a Fischer-style lock by a contention-sensitive algorithm to guarantee mutual exclusion. Could this approach work better than using Fischer-Lamport based construction?

We note that, in this preliminary version, we do not tackle all of these questions directly. Rather, 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 [8] for more details on Lamport's algorithm, Fischer's algorithm, and their combination, in order to put this work in context. For convenience we have included this paper as an appendix (with the authors' permission).

The paper is organized as follows. In Sections 2 and 3, we examine the behavior of locks under the exponential distribution and the gamma distribution, respectively. In Section 4, we present simulation results verifying our analysis and exploring the behavior of specific mutual exclusion algorithms.

2 Exponential Distribution

2.1 Properties of the Exponential Distribution

The exponential distribution proves convenient for theoretical study because of its unique special properties. We briefly note these properties here and make use of them without further reference throughout this paper.

- Memoryless property: Suppose 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 the times until each of k events are determined by independent exponential random variable with mean μ. Then the time until the first of these events occurs is exponential with mean $\frac{\mu}{k}$.
- Fairness property: Suppose the times until events A and B are determined by independent exponential random variable with means μ_1 and μ_2, respectively. Then event A occurs first with probability $\frac{\mu_2}{\mu_1+\mu_2}$.

2.2 How Many Pass Through?

Recall that the 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 passes successfully 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.

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 will be 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. In Fischer's timed mutual exclusion algorithm it is also possible for the algorithm to pause; in this section we do not consider the effects of pausing. (Pausing will be considered further in Sections 3 and 4.)

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 does just one processor successfully obtain 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 at least $\Omega(\sqrt{n/i})$ processors complete the inspection stage before the first write completes with constant probability.*

Remark: The assumption that the processors begin at the same time is for convenience; since all time 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 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

$$p_{z+1} = \prod_{1 \le j \le z} \frac{n-j}{ij+n-j} = \prod_{1 \le j \le z} \left(1 - \frac{ij}{ij+n-j}\right) \ge \prod_{1 \le j \le z} \left(1 - \frac{ij}{(1-\epsilon)n}\right),$$

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 constant probability, at least $\Omega\left(\sqrt{n/i}\right)$ 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 $\Theta(\sqrt{n/i})$. In fact, asymptotically exact formulae can be found with some work. We demonstrate here 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. But 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 $\Delta y = -xy$. 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

$$\int_0^1 n\mathrm{e}^{-nx^2/2} dx \approx \sqrt{\frac{\pi n}{2}}$$

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

Theorem 2 *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.*

Proof: The time between jth an $(j+1)$th 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, up to lower order terms, we have

$$\sum_{j=1}^{n} \frac{1}{r(n-j)+1} \approx \int_{x=0}^{n} \frac{1}{xr+1} dx = \frac{\ln rn + 1}{r}.$$

■

The argument of Theorem 2 can be easily modified into a result showing that in the setting of the theorem the number of processors that read their own value is $\Theta(\ln n)$ with high probability.

The key is that the event that a processor reads before the next write completes is independent of all other such events. Hence, to obtain a bound on the number of reads that complete, standard Chernoff-type bounds apply.

From Theorems 1 and 2 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, $\Theta(\ln\sqrt{n}) = \Theta(\ln n)$ then read their own values and hence pass through the lock. For a double lock, with high probability $\mathrm{O}(\ln n)$ get through the first lock, and hence on average 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) only changes 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 in regard to the number of processors that pass through (on average).

2.3 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, a single lock can quite poorly under high contention.

Theorem 3 *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 $O(\sqrt[r]{1/n})$.*

Proof: We begin with the case $r = 1$. Recall from Theorem 2 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 the last processor to do so is

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

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

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

where the approximations are correct up to lower order terms. ∎

The result of Theorem 3 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 under 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 only consider the problem of whether one processor writes to the second lock before any others read it. It is also reasonable, since generally a lock will be held long enough so that a critical section can be executed.

Theorem 4 *Consider a double lock where all actions take time exponentially distributed with mean 1 for each processor. Let n processors begin a write for a first lock of a double lock at the same time. Then with constant probability, 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 $c = r = w = 1$ for convenience.

The jth processor 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 such processor passes through the first lock is

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

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 right. 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, and then with probability 1/2 again, the second W operation completes before 1/2 the remaining at most $(n+j)/2$ writes to the first lock. But now, by the same argument as previously, the probability that no processors from the $n/2$'nd to the $(7n/8)$'th that finish its first write passes 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 constant probability a single processor successfully writes to the second lock as in the statement of the theorem.

When r and c 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 c.

■

The rather loose analysis of Theorem 4 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 4.

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. We feel that our intuitive proof, combined with simulation results, is a simpler and more natural approach to the problem.

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 that all times are deterministic, 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 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 k, the number of processors contending for the lock. Hence a single lock behaves more like a double lock under the exponential distribution in this case.

The intuition behind this performance in 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 of 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. 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 to finish 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 only happens 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. We note that, in practice, one must consider what the proper initial state more carefully. For example, if instead we begin by considering a set of processors in the check phase, then the state when the first processor writes will include processors half done with their writes. The argument can be easily modified to this case.

Theorem 5 *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 constant probability.*

Proof: We assume that n is sufficiently large throughout. We first show that, with constant probability, by the time the first few writes complete, with constant probability there are at least $6\sqrt{n}$ processors that are half done. We then show (again with constant probability) that there continue to be at least $4\sqrt{n}$ processors half done with their write until there are only $8\sqrt{n}$ processors that have not written. This fact will allow us to conclude that with constant probability, almost all processors fail to read their own value. We then argue that the probability that more than one of the last $8\sqrt{n}$ processors to write successfully reads its own value is at most a constant. Putting all the constants together will leave a constant probability of a single processor obtaining the lock.

Consider up until the time the first write completes. Let p_j be the probability that at least j processors are half done by this point. By the same argument as Theorem 1, $p_1 = 1$ and $p_{j+1} = p_j \frac{n-j}{n-j+1}$. Again, by the same argument as Theorem 1, one finds that at least $\sqrt{n}$ are half done when the first write completes with constant probability. One could use the same argument to show that at least $6\sqrt{n}$ writes are half done with constant probability; alternatively, one could show that in a similar manner that by the time some constant number of writes complete, at least $6\sqrt{n}$ writes are half done with (a higher) constant probability. In either case, notice that it is a very low probability event that any of these early writers successfully read their own value, since the probability they complete the final read before any of the processors that are half done with their write is at most $\left(\frac{1}{\sqrt{n}}\right)^2 = \frac{1}{n}$.

Once we reach the point where we have $6\sqrt{n}$ half done writes, we claim that there remain at least $4\sqrt{n}$ half done writes up until the point where there are at most $8\sqrt{n}$ processors left. This is most easily seen by noting that the number of half done writes is dominated by a simple random walk with boundaries at $4\sqrt{n}$ and $8\sqrt{n}$; at each step, the probability of gaining a half done write is at least as large as the probability of losing one. A simple calculation then shows that with at least constant probability, there are always at least $4\sqrt{n}$ half done writes. Hence, during the time when all these processors attempt to read their own value, the probability that each does so is at most $\left(\frac{1}{4\sqrt{n}}\right)^2 = \frac{1}{16n}$, and hence with constant probability none of them read their own value.

We need finally 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 with probability $\left(\frac{1}{n-j+1}\right)^2$. Hence the probability than any processor other than the last to write would read its own value would be at most $\sum_{j=1}^{n-1}\left(\frac{1}{n-j+1}\right)^2 < \frac{6}{\pi^2}$. (We elaborate on this below in Theorem 6.)

In the actual process, we have already seen that all behaves well up to the point when there are $8\sqrt{n}$ processors than are not even half way done with their write. After this point, the system behaves similarly to one where all remaining processors begin half done with their writes. Specifically, consider the point where there are k processors left that are not even half done with their write. It is easy to show that with high probability (on the order of $1/n^{2/3}$) that at his point there are $ck \log n$ processors half done with their write. Hence with high probability this holds at all points near the end, when at most $8\sqrt{n}$ processors are not even half way done with their write.

This implies that the probability of failure at the end, given that the system has behaved as predicted, is $\sum_{j=1}^{n-1}\left(\frac{1}{n-j+1}\right)^2 + o(1)$. The $o(1)$ term is partly due to the fact that we do not need to sum over values of j up to $n-1$, but only to $8c\sqrt{n}\log n$ Also, however, it is due to the fact the jth processor to complete its write would then successfully read its own value with probability slightly higher than $\left(\frac{1}{n-j+1}\right)^2$, because some processors are not yet half done with their writes. Note that the number of processor not yet half done is always small (a $\log n$ factor smaller) compared to the number that are. This leads to only an $o(1)$ difference.

To conclude, we find that in the 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 remain high, and hence no processors pass through the lock in the middle; and finally, at the end, conditioned on previous successful behavior, with constant probability only the last processor to write passes through the lock. This completes the argument. ■

Theorem 5 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, if the operation times are really exponential, it is natural to consider the case where only the read and not the write operation is delayed in this manner. This corresponds to using Fischer's algorithm [4], with an exponentially distributed pause before the final read.

Theorem 6 *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 $\left(\frac{j}{n-j+1}\right)^2$. Hence all but the last processor to write fail to pass through the lock with probability

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

■

4 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 lock-like structures. The goal of this section is to demonstrate that the theorems of Sections 2 and 3 accurately describe perceived performance, as well as gain more insight into the actual performance of mutual exclusion algorithms under these distributions. Because of space considerations, we present only some of our results; more complete results will appear in the final version.

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. 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 1. 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 lock outperforms a double lock.

Interestingly, the behavior as the number of processors increases is different for the three distributions! For the exponential, 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 non-monotonic 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 mutual exclusion algorithms in Figure 2. For these results, the distribution of the time for all operations is taken to be exponential with mean 1. Both algorithms FL1 and FL2 are combined Fischer-Lamport mutual exclusion algorithms. The algorithm FL1 is exactly the Algorithm 3 from [8]. We call this a nested Fischer-Lamport construction. The algorithm FL2 is a different variation that we call a sequential Fischer-Lamport construction. Details will appear in the full paper; here we note than the sequential construction takes more steps than the nested construction (and hence we would expect better performance).

Note the dramatic effect of the pause in the performance of FL1. This is not surprising, as the algorithm FL1 behaves similarly to a single lock, and as we have seen, a pause changes the behavior of single lock performance. Perhaps the most important feature to note, however, is that under the exponential distribution these mutual exclusion algorithms perform surprisingly well even in the face of very high contention.

5 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 lock 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 effects the design of mutual exclusion algorithms.

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 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.

In the full version of this paper, we will provide detailed analysis and simulation results focusing more directly on specific mutual exclusion algorithms, such as the combined exclusion algorithm of [8] and variations thereof. We will also provide some results for cases where read and write times for a processor are correlated.

References

[1] D. Attanasio, M. Butrico, J. Peterson, C. Polyzois, and S. Smith. Design and Implementation of a Recoverable Virtual Shared Disk, IBM Research Report, in preparation.

[2] H. Attiya and A. Fouren. In preparation.

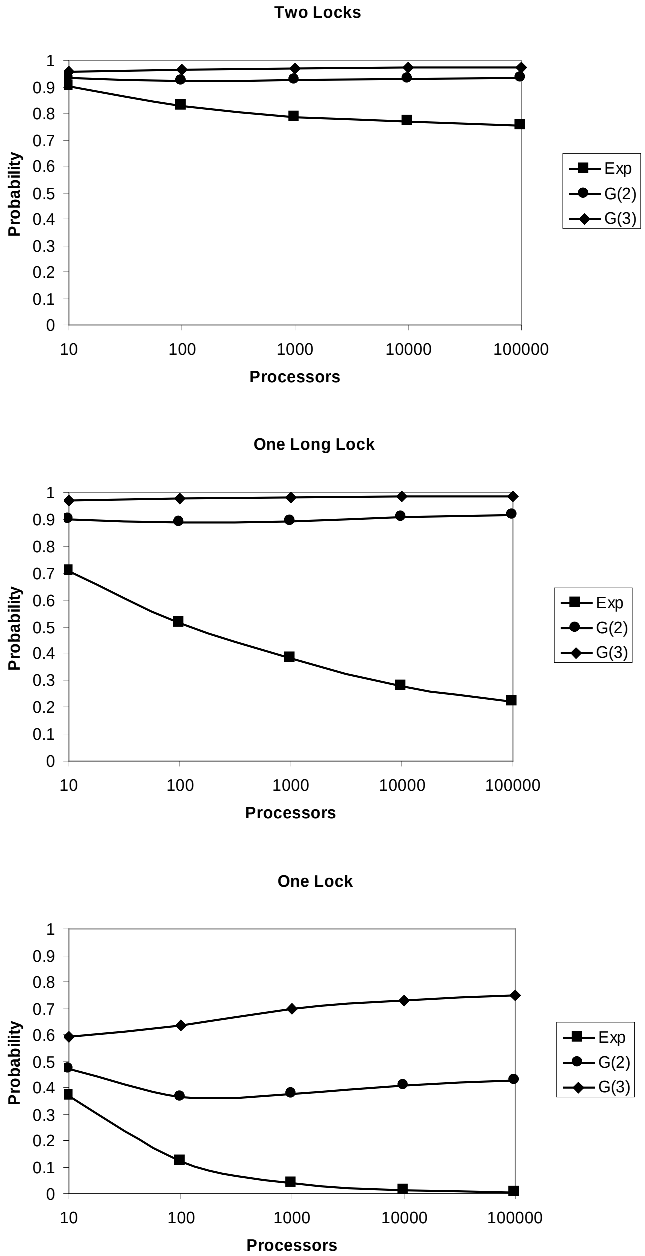

Figure 1: Comparing the behavior of a singe lock and a double lock.

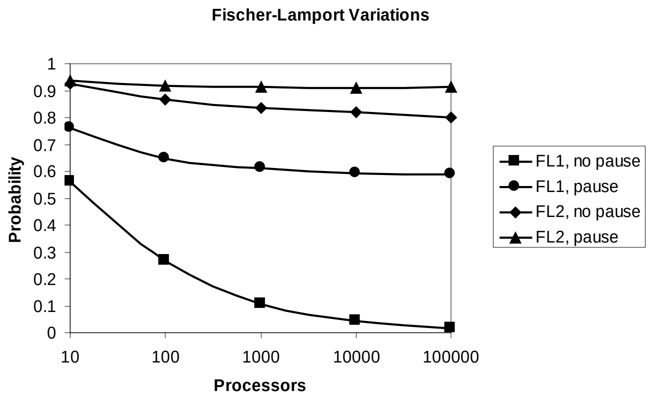

Figure 2: Comparing combined mutual exclusion algorithms.

[3] P. Cao, S. B. Lim, S. Venkatarman, and J. Wilkes. *ACM Trans. on Comp. Systems*, 12(3):236-267, Aug. 1994.

[4] M. Fischer. Personal communication from [8].

[5] L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. on Comp. Systems*, 5(1):1-11, Feb. 1987.

[6] E. K. Lee and C. A. Thekkath. Petal: Distributed Virtual Disks. *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84-92, Oct. 1996.

[7] B. Liskov. Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing*, 6: 211-219, 1993.

[8] N. Lynch and N. Shavit. Timing Based Mutual Exclusion. In *Proceedings of the Annual Real-Time Symposium (RTSS)*, Phoenix, pages 2-11 December 1992.

[9] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A Scalable Distributed File System. *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 224-237, Dec. 1997.