From *Pacific Rim International Symposium on Fault-Tolerant Systems*, Taipei, Taiwan, December 15-16, 1997.

PERFORMANCE ANALYSIS OF TWO TIME-BASED COORDINATED CHECKPOINTING PROTOCOLS

Gerard P. Kavanaugh and William H. Sanders
Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
{gkavanau,whs}@crhc.uiuc.edu

Abstract

Time-based checkpointing protocols are a recently proposed way to improve a system's dependability. They claim to have the advantages of coordinated protocols without the normal costs of coordination. This paper investigates that claim, by analyzing and comparing two time-based checkpointing protocols. The analysis is performed by determining the forward progress of a system using each protocol, and it is described in such a way as to be easily modifiable for other time-based protocols. By carefully analyzing the behavior of each protocol between renewal points, we are able to obtain a closed-form expression for the forward progress of the two protocols considered. We also determine the checkpoint interval value that will maximize forward progress. A validation of the analytical model is then performed via a detailed simulation. The results obtained from the model show the advantages and disadvantages of each protocol.

1 Introduction

Checkpointing is a commonly used technique to improve the dependability of systems operating in faulty environments. Checkpoint protocols function by periodically saving the state of an application to stable storage. If a failure occurs, the application simply rolls back, or reloads, the most recently saved state, and resumes execution from that point.

There is an immense body of research on checkpointing. An excellent survey of the main principles of these protocols is [1]. There are two main categories of methods of checkpointing: coordinated and uncoordinated. In coordinated protocols, the individual processes save their state together. This creates a global state from which the entire system is recoverable. If a failure occurs in any processor, then all processors must restart execution from this global state. In uncoordinated protocols, processes save their checkpoints individually. In this type of protocol the occurrence of a failure causes the affected processes to roll back to a point of recovery.

In traditional coordinated protocols all processes must save their states at the same time. Normally, communication is needed to achieve this coordination, and can result in significant overhead in practical implementations of such protocols. Recently, however, a new class of coordinated checkpoint protocol called time-based checkpointing protocols [2, 3, 4, 5, 6], have been introduced which eliminate much of the communication overhead by allowing processes to checkpoint without explicit communication. The ability of time-based checkpointing protocols to generate consistent and recoverable global states is based on loosely synchronized timers within individual processes. Because they use synchronized timers rather than explicit communication to achieve coordination, they have the potential to avoid most overhead present in other coordinated protocols. There are numerous examples of analytical models of traditional checkpointing protocols, for example [7, 8, 9, 10]. To date, however, a detailed analysis of time-based checkpointing protocols has not been performed.

This paper analyzes two recently developed time-based protocols [2, 3]. The protocols differ in how they maintain consistency and recoverability – one uses blocking and the other uses additional message logging. Just which of the protocols performs best will depend, in a complicated way, on the characteristics of the application and system (e.g. state savings time, recovery time, time between checkpoints) which use the protocol. For each, we derive a closed-form expression for the forward progress rate of a parameterized system implementing the protocol. These expressions show how assumptions about how the system operations affect an application's execution time. In addition, we derive close-form expressions for the optimal checkpoint interval for each protocol, given a set of assumptions. The two protocols are then compared using the respective optimal checkpoint intervals, determining which one performs best for each set of system assumptions. The results show that there are regions in which each protocol is better than the other, and thus provide important information to implementers who must choose between the two approaches. A simulation model of each protocol is also developed to validate the derived expressions. The simulation result match extremely well with results from the analytical expressions, but as expected take significantly more time to execute.

2 Protocol Description Assumptions

2.1 Protocol Description

Both of the time-based checkpointing protocols analyzed in this paper were developed by Fuchs and

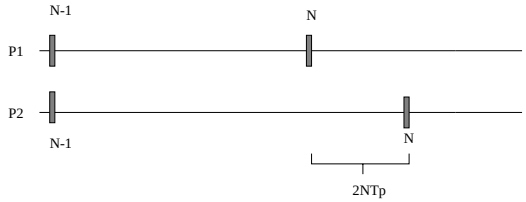

Figure 1: Maximum Inter-Process Drift After N Checkpoint Intervals

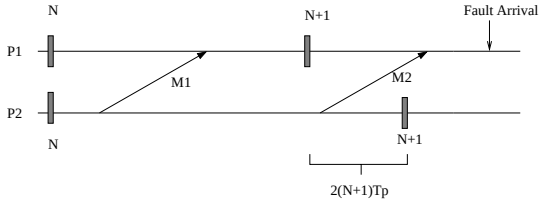

Figure 2: Unrecoverable Global State at Checkpoint N+1

Neeves [2, 3]. As time-based protocols, they both attempt to avoid much of the communication overhead associated with coordinated protocols by assuming that processes have loosely synchronized clocks. They avoid the overhead of synchronization found in previous protocols [4, 5, 6] by using timers and by assuming that all process timers are approximately synchronized with a deviation from real time of some value ρ, which is referred to as the clock drift.

In particular, if a timer is started on each of two processors at the same time, and each processor has clock drift rate equal to ρ, the timers will expire at most $2\rho T/(1-\rho^2)$ seconds apart, where T was the initial timer value [2]. Since typical ρ values are very small, on the order of 10^{-5} to 10^{-8}, this maximum inter-processor drift has been approximated as simply $2\rho T$. Assuming this, the clocks will exhibit a maximum drift of $2N\rho T$ after N checkpoint intervals, as shown in Figure 1. Notice that this value appears to assume fault-free execution. This is because the occurrence of a fault also causes all the participating processes to become resynchronized at the previous checkpoint.

The two protocols differ in their respective algorithms for ensuring that a process's saved checkpoint states are both consistent and recoverable. For a simple example of how checkpoint states may become unrecoverable if messages in transit are not considered, consider the case shown in Figure 2. Here, process P_1 has received message M_2 from process P_2 prior to the occurrence of a fault. Notice that checkpoint $N+1$ was saved in P_1 prior to receiving the message, while checkpoint $N+1$ in P_2 was saved after M_2 was sent. When the fault does occur, each process will recover by restarting execution at checkpoint $N+1$. Now P_1 has yet to receive M_2, but P_2 doesn't know. As a result, the global state at checkpoint $N+1$ is unrecoverable, and is not an acceptable recovery point. Inconsistent

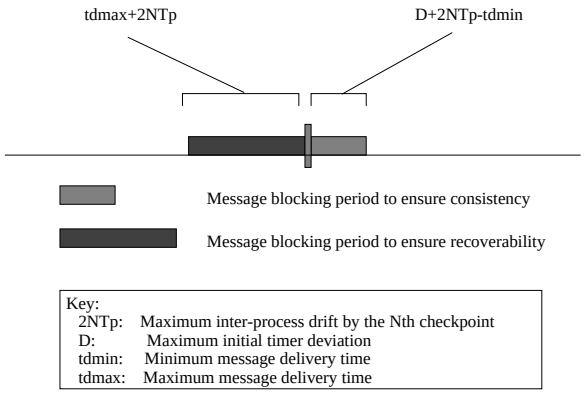

Figure 3: Means for Ensuring Consistency and Recoverability in the Blocking Protocol

checkpoints are created in much the same way, except that in order to create inconsistencies it is the sending process which loses track of the message.

The first protocol that will be discussed [2] will be referred to as the "blocking protocol." Figure 3 shows the important time periods around each checkpoint event. Note the blocking periods; hence our reference to this protocol as a "blocking protocol." This protocol ensures that there are no messages in transit that disturb the consistency or recoverability of the global states by preventing messages from being sent during an interval before and after the checkpoint is saved. The length of these intervals is calculated based on the maximum possible inter-process clock drift. The blocking period before the checkpoint is taken guarantees its recoverability, and the period after the checkpoint is saved ensures its consistency.

The second protocol, which we refer to as the "non-blocking protocol," uses the same message blocking period after the checkpoint to ensure consistency, but for recoverability it does not use blocking [3]. Instead of blocking messages, the non-blocking protocol saves, as part of the next checkpoint, all messages for which it has not received an acknowledge response. This way, if a fault does arrive, as shown in Figure 2, the protocol may resend all of the unacknowledged messages again. This will result in a greater time to save checkpoints, since certain messages must now be saved, but Neeves and Fuchs believe this cost is more than offset by the fact that the process no longer has to block execution.

In both protocols, the consistency blocking interval is a function of time and the current checkpoint number. Therefore, as the execution of the application progresses, this blocking interval will increase, and each process will spend a greater and greater portion of its current checkpoint interval blocking to ensure consistency. Eventually this will become prohibitive, so occasionally processes will be forced to resynchronize. This resynchronization is accomplished by the coordinating processor starting all the processes at the same time so that the maximum inter-process drift is only some initial deviation again.

Since the consistency blocking interval occurs while

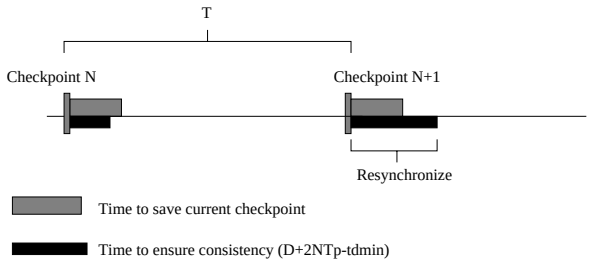

Figure 4: Time-Based Resynchronization

the process is blocking to save its checkpoint, these protocols do not waste time ensuring consistency until the blocking interval for consistency is longer than the time to save the current checkpoint. Thus, to avoid all blocking to ensure consistency, these protocols will resynchronize whenever any process's blocking interval becomes longer than the checkpoint creation interval in addition to resynchronizing when a fault occurs. An example of this is shown in Figure 4.

2.2 Model Assumptions

Several assumptions were made about the environment in which the protocols were applied to facilitate their analysis. These assumptions are realistic and do not in any way change the operation of the protocols. Whenever one of the following assumptions is referenced in the paper, a boldfaced number in parentheses will be used to refer to it. The assumptions made are as follows.

1. **Faults have an independent, exponentially distributed, time between occurrences**. This is a frequently made assumption, and reasonable for many fault classes. In this paper, the assumption is made in both the analytic model and the simulation model.

2. **Reads and writes to stable storage take constant time**. This assumption makes the discussion and notation used simpler. In reality, the analysis follows through for any read and write distributions, if the expected values of these distributions are used.

3. **Faults do not arrive while the system is recovering**. Although [2, 3] do not specify the protocol's reaction to a fault during recovery, private communication with the authors suggests that a means of recovering from faults that occur during recovery is possible. Since the protocol's activity is unspecified, though, the analysis in this paper assumes it will not happen. This assumption is valid since the probability of a second fault occurring during recovery is normally very small.

4. **Faults do not arrive while checkpoints are being saved**. Much like the previous assumption, this assumption is based on the fact that the checkpoint interval is much longer than the time to save a checkpoint or $T \gg S$. As with (**3**), the probability of a fault arriving during the saving of a checkpoint is nearly zero.

3 Analytic Model Development

With the few simplifying assumptions shown in the previous section, we can express the forward progress of the two protocols as a simple closed-form expression. The analysis obtains a system's or process's forward progress – defined as the ratio of useful work to total work during a given interval of time. The interval of time with which this analysis is concerned will be the period of time between resynchronizations. If there are no faults, this time will be equal to a fixed number of checkpoint intervals, N_M, followed by the resynchronization time, Y. The occurrence of a fault will also cause a resynchronization, so the expected number of intervals between resynchronizations, $E[N_R]$, will be determined by the probability of a fault occurring during any interval less than N_M and of no faults occurring before N_M intervals are successfully completed. The total time between resynchronizations will be determined by the number of intervals of length T until a resynchronization is expected to occur, and any additional work that is not useful.

Examples of work which is not useful include the checkpoint save time, the work that must be redone due to a fault occurrence, the recovery time, the blocking intervals to ensure consistency and recoverability, and the resynchronization time. Once a single process's forward progress (FP) is determined, we can determine the forward progress rate of a system of P communicating processes by simply averaging the individual process's forward progresses.

To begin the analysis, we determine the number of intervals between resynchronizations given that a fault does not occur. The time between resynchronizations is dependent on the time required to save a checkpoint, and the time during which messages may not be sent after a checkpoint. As seen in Figure 4, as soon as the message blocking time is greater than the checkpoint time a resynchronization is performed. We will refer to the time to save a checkpoint as S, and recall that the time during which messages are blocked can be calculated by the formula $D+2NT\rho-t_{dmin}$. Furthermore, we will assume that S is a constant (**2**). Thus, we may compute the number of checkpoints before a resynchronization is needed by solving the following equation for N_M.

$$S \leq D + 2N_M T\rho - t_{dmin}$$

$$\frac{S + t_{dmin} - D}{2T\rho} \leq N_M$$

$$N_M = \left\lceil \frac{S + t_{dmin} - D}{2T\rho} \right\rceil$$

Resynchronizations are also caused by the occurrence of faults. Therefore, in order to compute the expected number of intervals until a resynchronization

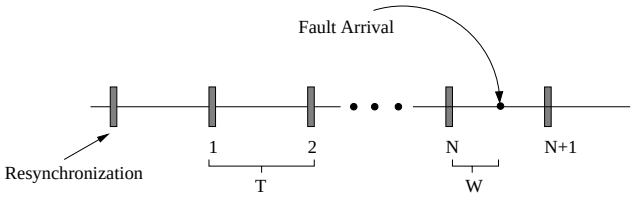

Figure 5: Number of Intervals Between Resynchronizations, Given That a Fault Arrives

occurs, we will need to determine the probability that a fault arrives during a given number of checkpoint intervals since the previous resynchronization. Since the time between fault occurrences is exponentially distributed **(1)**, the probability of a fault occurring N checkpoint intervals since the last resynchronization, as shown in Figure 5, is given by

$$\left[e^{-\lambda TN} - e^{-\lambda T(N+1)}\right].$$

Thus the expected number of intervals until a resynchronization, $E[N_R]$, is given by the weighted probability of different numbers of intervals occurring before a resynchronization, or

$$\begin{aligned} E\left[N_R\right] &= \left[\sum_{N=1}^{N_M-1} N\left[e^{-\lambda TN} - e^{-\lambda T(N+1)}\right]\right] \\ &+ N_M(e^{-\lambda TN_M}). \end{aligned} \quad (1)$$

Since

$$\sum_{i=0}^{n} ie^{ai} = \frac{e^a\left(1-e^{a(n+1)}\right)}{\left(e^a\right)^2 - 2e^a + 1} + \frac{(n+1)\,e^{a(n+1)}}{e^a - 1},$$

Equation 1 simplifies to

$$E\left[N_R\right] = \frac{\left(1-e^{-\lambda TN_M}\right)}{e^{\lambda T} - 1}.$$

In order to determine the FP of the system, we develop a ratio of useful work to total work. A major difference between the useful and total work between resynchronizations due to a fault occurrence will be the amount of wasted work that must be redone. To calculate this quantity, we will determine the expected time until a fault occurs given that at least one fault occurs during a checkpoint interval. Let J be the time, relative to the start of the checkpoint interval, of the first fault, as shown in Figure 6. Further, recall that in this analysis faults may not arrive while the current checkpoint is being saved (**4**), so our interval of interest will be $T_f = T - S$. Then, the cumulative distribution function of the time of the first fault, given that at least one fault occurs, is

$$F_{J|J<T_f}(t) = P\left[J \le t | J < T_f\right] = \frac{1-e^{-\lambda t}}{1-e^{-\lambda T_f}}.$$

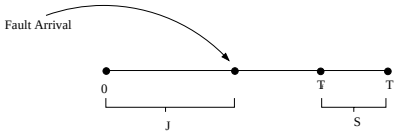

Figure 6: Wasted Time Given That a Fault Arrives

Notice that the expected time until a fault arrives given that it arrives during the current interval is also equal to the expected wasted time, $E[W]$, where W is shown in Figure 5. After taking the expectation, the expected wasted time or the expected time until a fault, given that one occurs, is

$$E\left[W\right] = E\left[J | J < T_f\right] = \frac{\left(e^{-\lambda T_f}\left(-\lambda T_f - 1\right) + 1\right)}{\lambda\left(1 - e^{-\lambda T_f}\right)}.$$

Since the determination of our expected total work factors is a result of cases of fault-free and fault-induced resynchronizations, another important quantity will be the probability of a fault occurring before a resynchronization is required. It is given by

$$P\left[J \le N_M T\right] = \left(1 - e^{-\lambda N_M T}\right),$$

and consequently the probability of a fault not occurring before mandatory resynchronization is

$$P\left[J > N_M T\right] = 1 - P\left[J < N_M T\right] = e^{-\lambda N_M T}.$$

With the above probabilities we may now determine the expected wasted time between resynchronizations by unconditioning the cause of the resynchronization. It is given by

$$E\left[W_T\right] = \left(1 - e^{-\lambda TN_M}\right)\left(E\left[W\right] + R\right) + e^{-\lambda TN_M} Y.$$

For both protocols, the expected total time between resynchronizations, $E[TW]$, will be the same. It will be given by the expected number of intervals before resynchronization times the interval length plus the expected wasted time, or

$$E\left[TW\right] = E\left[N_R\right] T + E\left[W_T\right].$$

Similarly, the expected time spent doing useful work will have a similar form for the two protocols. If we let T' represent the portion of T spent doing useful work, then the expected useful work time will be given by

$$E\left[UW\right] = E\left[N_R\right] T'.$$

Now we can determine a single process's forward progress, FP. In particular,

$$FP = \frac{E\left[UW\right]}{E\left[TW\right]} = \frac{E\left[N_R\right] T'}{E\left[N_R\right] T + E\left[W_T\right]},$$

or more specifically, $FP =$

$$\frac{\left(E\left[N_R\right]\right) T'}{\left(E\left[N_R\right]\right) T + \left(1 - e^{-\lambda TN_M}\right)\left(E\left[W\right] + R\right) + e^{-\lambda TN_M} Y}. \quad (2)$$

For the non-blocking time-based checkpointing protocol, the interval of useful work will be the interval time T minus the time to save the checkpoint S, or simply T_f, since execution is blocked while the checkpoint is being saved. Thus:

$$T'_{non-blocking} = T_f$$

The blocking protocol's useful interval will be T minus the blocking period, as shown in Figure 3. In this protocol, the blocking period will be equal to the sum of the consistency and recoverability blocking periods. Since the time to save a checkpoint overlaps the consistency blocking interval, we need only subtract the average recoverability blocking period over the expected number of intervals until resynchronization, $E[N_R]$, to determine T'. Thus:

$$T'_{blocking} = T_f - t_{dmax} - T\rho\left(E[N_R] + 1\right)$$

Now that we have determined the forward progress of a single process, we will use this knowledge to expand our analysis to a system of P communicating processes. In particular, for P processes let ρ be a vector where ρ_i specifies the clock drift of the ith process and $i \in \{1, 2, .., P\}$. Furthermore, note that all N_{M_i}'s are equal. This is because whichever process needs to resynchronize first will cause all the other processes to resynchronize also. Thus, all N_{M_i} will equal the minimum N_{M_i}. Since the maximum ρ_i will determine the minimum N_{M_i} we will simply define N_M as

$$N_M = \left\lceil \frac{S + t_{dmin} - D}{2Tmax(\rho_i)} \right\rceil. \tag{3}$$

The assumption that all other variables are equal among processes yields the following single process contribution to the overall forward progress:

$$FP_i = \frac{1}{P}\frac{E[UW_i]}{E[TW_i]}.$$

Summing over P processes yields a total system forward progress of

$$FP = \frac{1}{P}\sum_{i=1}^{P}\frac{E[UW_i]}{E[TW_i]}.$$

Assuming all processes have the same fault rate, λ, then the total system fault rate will equal $P\lambda$. Finally, if we are able to assume the same S and R among all processes (**2**) then the expression for the forward progress remains identical to equation 2, except

$$E[N_R] = \frac{\left(1 - e^{-P\lambda T N_M}\right)}{e^{P\lambda T} - 1}, \tag{4}$$

and

$$\begin{aligned} E[W_T] &= \left(1 - e^{-P\lambda T N_M}\right)\left(E[W] + R\right) \\ &+ e^{-P\lambda T N_M} Y. \end{aligned} \tag{5}$$

Equations 2, 3, 4, and 5 describe the forward progress of both of the time-based checkpointing protocols, as a function of measurable system characteristics. We now use these expressions to compute the checkpointing interval that gives the maximum forward progress rate, for particular values of λ, T, t_{dmin}, t_{dmax}, ρ, S, R, D, P, and Y. The expression also gives us a better understanding of possible bottlenecks and problems with these types of protocols. Also, as with any other type of checkpointing protocol, FP is a function of the checkpoint interval, T. This means that there exists a T for which the system is most productive. We will refer to this value of T as the optimal T.

4 Optimal Checkpoint Interval

At first glance, it may seem simple to determine the optimal checkpoint interval, by finding the derivative of the forward progress expression with respect to T. Due to the ceiling function that is found in our N_M variable, the forward progress expression has points of discontinuity at which it is not differentiable. Further, although the solution to the derivative will always be near the point of maximum forward progress, there is no guarantee that it is the absolute maximum. The complete derivation of the optimal forward progress is non-trivial, but it is straight forward. As a result of space considerations, the analysis could not be shown, but it may be found in [11], and it will be used in the validation and comparison sections to follow.

5 Model Validation

To confirm the correctness of our analysis, we compared results obtained from the derived forward progress expressions to a more detailed simulation of each protocol. The simulation of each model consists of N replicated processors, each of which has a fault rate λ. The clock drift for each processor is $max(\rho_i)$, since, as in the analytical model, smaller ρ_i's have no effect on the system. Having the same clock drift for all of the processor replications can therefore minimize the model's size without compromising the model's integrity. In the simulation models, the execution of an application may be interrupted by a fault at any time on any processor. The occurrence of this fault causes all processors to roll back to their previous saved state. This rollback also causes all processors to be resynchronized. If no faults arrive for a period of time equal to the length of the checkpoint interval then each processor independently saves its current state. If the time to save the state was less than $D + 2N_M T\rho - t_{dmin}$ at any processor, than all processors are resynchronized. Processes resume normal execution either after they are finished saving their states, or if a resynchronization was necessary then after it is complete. Blocking periods are simulated as periods of non-execution. Forward progress is determined in the protocol simulations by dividing the time spent

Comparison	Variable	Protocol	
		Blocking	Non-Blocking
	Y	.1	.1
	P	4	4
	t_{dmin}	.001	.001
	t_{dmax}	.01	.01
	D	.01	.01
Variable λ	T	3600	3600
	S, R	.6	.7
	ρ	1E-5	1E-5
Variable T	λ	1E-5	1E-5
	S, R	2	2.2
	ρ	1E-6	1E-6

Figure 7: System Values for Protocol Validation

in the execution state to the total time spent for the simulation. Thus, wasted times are never calculated, and the number of intervals until resynchronization is mandatory is never calculated which is just how the actual protocols are implemented.

Using this model, we show that our analytical model can correctly express the system's forward progress for varying fault rates and checkpoint interval lengths. The parameters for the following validations are shown in Figure 7. These parameters were selected because they are realistic. Although the analytical model should still be effective for unrealistic parameters, as the extreme range in checkpoint intervals and fault rates will show, it is not possible to present validation results for a range of every parameter.

Figure 8[t] contains the validation results for varying fault rates between the simulation and analytic models for the blocking and non-blocking protocols respectively. It is readily apparent from these tables that the analytic model is not only very accurate, but also robust enough to handle a wide range of fault rates.

It will also be important to validate the analytical model for varying checkpoint interval lengths, since that will ensure the validity of our optimal checkpoint interval calculation. In addition, it will reemphasize the accuracy of the analytical model for varying system parameters, because completely different systems are being used in the checkpoint interval validation from those used in the fault rate validation. Figure 8 also contains the results from the varying checkpoint interval validation. As can be seen from all of these results, the analytical model produces nearly identical forward progress calculations as the simulation for a wide range of checkpoint interval lengths and fault rates, and the analytical model will be accurate for system parameters representing a wide variety of application environments. Another reason for the usefulness of the analytic model is its dramatic speedup over simulation models. The time to complete a simulation for the validations in this section varied from several minutes to several hours. Through the use of the analytic model, a system's performance may be completely characterized almost instantaneously, but to perform the same task via simulation, it could take weeks.

Variable	Protocol	
	Blocking	Non-Blocking
λ	variable	variable
S, R	.6	.7
Y	.1	.1
P	4	4
ρ	1E-5	1E-5
t_{dmin}	.001	.001
t_{dmax}	1	1
D	.01	.01

Figure 9: System Values for Protocol Comparisons: Variable λ

	Forward Progress Protocol	
Fault Rate	Blocking	Non-Blocking
.0001	0.446826	0.446931
1E-5	0.929248	0.929532
1E-6	0.992279	0.99262
1E-7	0.998734	0.999083

Figure 10: Comparison of Protocols: Variable Fault Rate, Optimal T

6 Comparison of Protocols

The remainder of this section will use the developed analytic models to compare the two protocols considered. In order to make the comparison fair, almost all of the system variables will be the same. One consistent difference will be the time to save or restore the checkpoint, because the non-blocking protocol will have a slightly larger state to save. This is due to the fact that the non-blocking protocol will also be saving the messages that have yet to be acknowledged. Each comparison was performed between maximum forward progresses, which means that for every set of system parameters the optimal T was calculated for each protocol. The analysis is performed for several variables in order to show not only how the protocols operate, but also how flexible the analytical model is.

The first comparison is performed with a variable fault rate. The parameter values used are shown in Figure 9, and the results are shown in Figure 10. These figures show that for the given system parameters and realistic fault rates the non-blocking protocol will always have a maximum system forward progress greater than that of the blocking protocol. This result is expected for these parameter values, however. The only way that the blocking protocol could have superior performance is if the time to save or restore a checkpoint in the non-blocking protocol became large enough, due to the additional unacknowl-

Fault Rate (λ)	Model's Forward Progress			
	Blocking		Non-Blocking	
	Simulation (98% Confidence Interval)	Analytic Value	Simulation (98% Confidence Interval)	Analytic Value
1E-7	[0.9979 , 0.9984]	0.998983	[0.99815 , 0.99857]	0.999083
1E-6	[0.9910 , 0.9923]	0.992527	[0.9910 , 0.9924]	0.99262
1E-5	[0.927 , 0.931]	0.929481	[0.926 , 0.931]	0.929532
.0001	[0.441 , 0.452]	0.446938	[0.438 , 0.448]	0.446931
.001	[6.10E-6 , 1.02E-5]	8.00557E-6	[6.10E-6 , 1.02E-5]	8.00246E-6

Checkpoint Interval (T)	Model's Forward Progress			
	Blocking		Non-Blocking	
	Simulation (98% Confidence Interval)	Analytic Value	Simulation (98% Confidence Interval)	Analytic Value
100	[0.97644 , 0.97655]	0.976754	[0.97598 , 0.97609]	0.976002
10100	[0.803 , 0.814]	0.811356	[0.804 , 0.815]	0.811347
20100	[0.637 , 0.659]	0.651194	[0.640 , 0.660]	0.651189
30100	[0.502 , 0.522]	0.515915	[0.510 , 0.531]	0.515911
40100	[0.390 , 0.406]	0.403691	[0.389 , 0.405]	0.403688
50100	[0.301 , 0.313]	0.312181	[0.301 , 0.313]	0.312179

Figure 8: Protocol Validations

Variable	Protocol	
	Blocking	Non-Blocking
λ	1E-5	1E-5
S, R	.6	variable
Y	.1	.1
P	4	4
ρ	1E-6	1E-6
t_{dmin}	.001	.001
t_{dmax}	.07	.07
D	.4	.4

Figure 11: System Values for Protocol Comparisons: Variable S, R

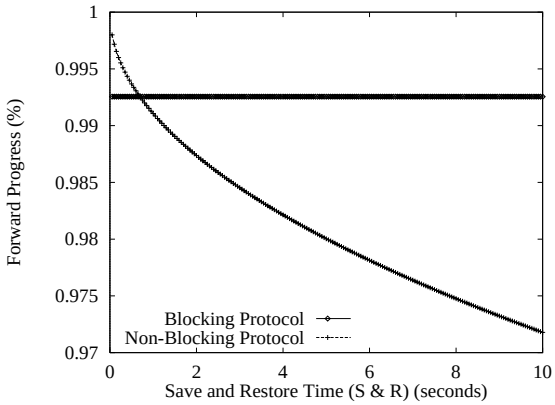

Figure 12: Comparison of Protocols: Varying Stable Storage Access Time, Optimal T

edged messages, that the non-blocking protocol was wasting time. The question now is whether a system with high enough inter-process communication is realistic. For this comparison we will use the parameter values specified in Figure 11.

In Figure 12 we see that the blocking protocol may outperform the non-blocking protocol if the checkpoint save time, S, and the checkpoint restore time, R, are high enough. For the parameters described in Figure 11, the crossover point is at approximately .7 seconds. Although this crossover point will vary for systems characterized with parameters other than those specified in Figure 11, it will still exist if the size or number of messages being passed between processes is large enough. For some systems this value may be unrealistic, but this is not the case in all systems. Through the use of the analytical model described in this paper, minimal effort is required to determine which checkpointing protocol is most appropriate.

Another application of this analysis is to see the effect of varying ρ, the clock drift rate, on a system's forward progress. Since different processors may have widely differing clock drifts these results could be very important in determining what protocol to implement. For the following analysis the values in Figure 13 were used.

Figure 14 shows the effects of varying clock drift rates on both protocols. The graph shows that the

Variable	Protocol	
	Blocking	Non-Blocking
λ	1E-5	1E-5
S, R	.6	.7
Y	.1	.1
P	4	4
ρ	variable	variable
t_{dmin}	.001	.001
t_{dmax}	.07	.07
D	.01	.01

Figure 13: System Values for Protocol Comparisons: Variable ρ

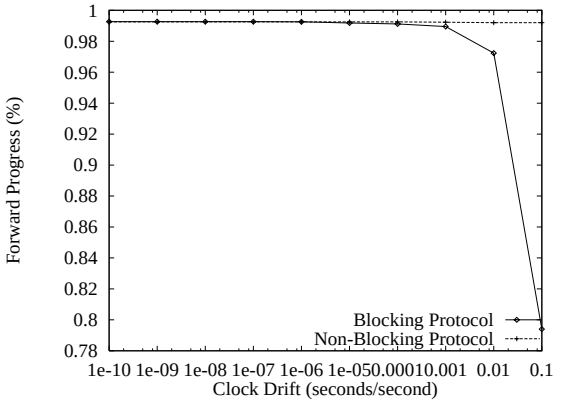

Figure 14: Comparison of Protocols: Varying ρ, Optimal T

blocking protocol's performance begins to degrade rapidly once a clock drift of .001 seconds per second is reached. The non-blocking protocol, however, is virtually immune to changes in ρ. This is because the recoverability blocking period in the former protocol is a function of the clock drift rate. Although the performance of the blocking protocol does not degrade until extremely high values are examined, these values do confirm our expectations of the protocol's performance. For realistic values of ρ, which is typically 10^{-7} to 10^{-8} for very precise clocks and 10^{-5} to 10^{-6} for normal clocks, both protocols appear to be ρ invariant. This is encouraging because it means that these time-based protocols will be effective for a wide variety of clock qualities. Interestingly, it may appear in Figure 14 as though the two protocols perform identically well for low clock drift rates, but the blocking protocol actually slightly outperforms the non-blocking protocol at low clock drift rates for the given parameters. This is since, for the specified parameters, the additional time to save a checkpoint in the non-blocking protocol is larger than the recoverability blocking period for low clock drift rates in the blocking protocol.

7 Conclusion

This paper considers two time-based checkpointing protocols. The first uses blocking to achieve consistency and recoverability, while the second uses additional message logging. For a given set of system parameter values it was not clear which protocol will perform best. To investigate this, we derived closed-form expressions for the rate for forward progress of an application, given a particular set of system parameters, and the checkpoint interval that optimizes the forward progress rate, in terms of the remaining system parameter values. These expressions were then used to analyze the behavior of the two protocols for a variety of system parameters. We saw that there were parameter value regions where each protocol performed best, thus yielding important information for system designers who may want to choose between the two protocols for for a given set of parameter values. In addition, they suggest ways in which the two protocols might be combined, in order to build a protocol that works well under widely varying system conditions.

References

[1] E. Elnozahy, D. Johnson, and Y. Wang, "A survey of rollback-recovery protocols in message-passing systems." This paper has been submitted for publication in ACM Computing Surveys.

[2] N. Neves and W. K. Fuchs, "Using time to improve the performance of coordinated checkpointing," in *IEEE International Computer Performance and Dependability Symposium*, September 1996.

[3] N. Neves and W. K. Fuchs, "Coordinated checkpointing without direct coordination." This paper has yet to be published.

[4] F. Cristian and F. Jahanian, "A timestamp-based checkpointing protocol for long-lived distributed computations," in *10th Symposium on Reliable Distributed Systems*, September 1991.

[5] P. Ramanathan and K. Shin, "Use of common time base for checkpointing and rollback recovery in a distributed system," *IEEE Transactions on Software Engineering*, vol. 2, June 1993.

[6] Z. Tong, R. Kain, and W. Tsai, "A low overhead checkpointing and rollback recovery scheme for distributed systems," in *8th Symposium on Reliable Distributed Systems*, October 1993.

[7] K. M. Chandy, J. Browne, C. Dissly, and W. Uhrig, "Analytic models for rollback and recovery strategies in data base systems," *IEEE Transactions on Software Engineering*, vol. SE-1, March 1975.

[8] V. Nicola and J. V. Spanje, "Comparative analysis of different models of checkpointing and recovery," *IEEE Transactions on Software Engineering*, vol. 16, August 1990.

[9] A. Tantawi and M. Ruschitzka, "Performance analysis of checkpointing strategies," *ACM Transactions on Computer Systems*, vol. 2, May 1984.

[10] J. Young, "A first order approximation to the optimal checkpoint interval," *Communications of the ACM*, vol. 17, September 1974.

[11] G. P. Kavanaugh and W. H. Sanders, "Performance analysis of two time-based coordinated checkpointing protocols," tech. rep., Center for Reliable and High Performance Computing, 1997.