2009 11th IEEE International Conference on High Performance Computing and Communications

N-Level Diskless Checkpointing

Doug Hakkarinen and Zizhong Chen
Department of Mathematical and Computer Sciences
Colorado School of Mines,Golden, CO 80401, USA
{dhakkari, zchen}@mines.edu

Abstract

Diskless checkpointing is an efficient technique to tolerate a small number of processor failures in large parallel and distributed systems. In literature, a simultaneous failure of no more than N processors is often tolerated by using a one-level Reed-Solomon checkpointing scheme for N simultaneous processor failures, whose overhead often increases quickly as N increases. In this paper, we study an N-level diskless checkpointing scheme to reduce the overhead for tolerating a simultaneous failure of no more than N processors by layering the schemes for a simultaneous failure of i processors, where $i = 1, 2, \ldots, N$. Simulation results indicate the proposed N-level diskless checkpointing scheme achieves lower fault tolerance overhead than the one-level Reed-Solomon checkpointing scheme for N simultaneous processor failures.

Keywords: checkpoint; diskless checkpointing; fault tolerance; high performance computing; parallel and distributed systems

1 Introduction

Due to the recent decline in the rate processor speeds are advancing, manufacturers of computers and high performance computers are now using parallel processors and other components to continue improving computational speeds. Unfortunately, many popular multiprocessor library implementations require that all components are operational in order for the proper function of the overall computational platform. Using the well accepted heuristic that a computer has a constant failure rate during most its operation, failures will increase linearly with regards to the number of components. Unfortunately, this means that the strategy of using many components will lead to dramatically increasing failure rates as the number of processors and other components increase.

Additionally, the increased prevalence of computers in new fields has resulted in vastly greater amounts of data to analyze and perform computations on. This increased need for computation over larger data sets has resulted in ever longer computations to be performed. With increased time of computation, the penalty for a failure resulting in loss increases as there is more to be recomputed. Furthermore, as the computation will require more time with increased data, the likelihood of a failure during a specific execution of a program has also increased.

Due to this dramatically increasing rate and increased data loads, methods of avoiding and mitigating failures must be explored. Without such fault tolerant methods the field of computing would find itself limited in its ability to advance in processors and other components, as well as in its ability to keep pace with the computational need in the increasingly digital world.

One of the standard ways to add fault tolerance is through the use of checkpoints. In the traditional form, a checkpoint is a recording of processor state that is backed up on stable storage, or a disk. Checkpointing nearly always requires routine saving of state, which requires considerable overhead even without a failure. There are other methods of providing fault tolerance, such as using redundant execution in order to provide fault tolerance for distributed clusters [10], but checkpointing remains one of the prominent strategies. While the stable-storage-based checkpoint is able to tolerate the failure of the whole system, the simultaneous access of the stable-storage by a large number of processors becomes a bottleneck [6].

One improvement for such bottleneck is the use of

This research was supported in part by the Los Alamos National Laboratory under Contract No. 03891-001-99 49 and the Applied Mathematical Sciences Research Program of the Office of Mathematical, Information, and Computational Sciences, U.S. Department of Energy under contract DE-AC05-00OR22725 with UT-Battelle, LLC.

978-0-7695-3738-2/09 $25.00 © 2009 IEEE
DOI 10.1109/HPCC.2009.55

diskless checkpointing [6, 2], which provides a redundancy state by having all computation processors routinely calculate the XOR of different processor states and transmit the XORed state to a dedicated checkpoint processor. If a processor fails, then the original state of that processor at the last checkpoint can be reconstructed using the XORed state on the checkpoint processors and the states of all the other surviving processors. Thus, the original state of a processor can be reconstructed should it fail using the memory content of other processors. Specifically, the state that was last transmitted would be able to be recovered.

Previous research in diskless checkpointing includes where diskless checkpointing is implemented in order to assist with specific algorithms on clusters of workstations [7], application of diskless checkpointing to superscalar architectures [4], methodologies for increasing the performance of diskless checkpointing using incremental checkpoints [9], and the use of floating-point coding in conjunction with diskless checkpointing in order to improve performance and portability [3].

Diskless checkpointing has also been used to tolerate a simultaneous failure of no more than N processors by using a Reed-Solomon encoding scheme for N simultaneous processor failures. However, the time to recover after a failure and the amount of time to perform a checkpoint usually increase quickly as N increases.

In this paper, we propose an N-level diskless checkpointing scheme to reduce the overhead for tolerating a simultaneous failure of no more than N processors by layering the diskless checkpointing schemes for a simultaneous failure of i processors, where $i = 1, 2, \ldots, N$. While the N-level diskless checkpointing scheme is able to tolerate a simultaneous failure of N processors, it tolerates the more frequent case of a simultaneous failure of less than N processors with lower overhead than the one-level scheme for N simultaneous processor failures. Therefore, when the number of simultaneous processor failures is a random number from 1 to N (which is usually true in practice), the N-level diskless checkpointing scheme can reduce the total fault tolerance overhead significantly. We focus on developing a reasonable heuristic for determining the checkpoint interval for each layer of the checkpoint scheme given the time required to set the scheme up as well as the time to recover using that scheme. Additionally, we will provide a framework for simulation in order to help evaluate the model of expected time to complete a task produced by that heuristic. Simulation results indicate the proposed N-level diskless checkpointing scheme achieves lower fault tolerance overhead than the one-level N-simultaneous-failure scheme when tolerating a simultaneous failure of a random number processors.

2 Related Work

Techniques have been developed to handle multiple failures for specific matrix operations [5]. Multiple types of diskless checkpointing have also been compared [13]. Finally, Plank and Li present a system to use an additional processor to help perform diskless checkpointing and outlines how to expand the process to tolerate multiple failures [8].

The layering of diskless and standard checkpoints was studied in [15, 14, 12]. Layering checkpoints, as well with various types of checkpoints such as checkpoints that only detect an inconsistent state or those that only can recover from an inconsistent state has also been explored [16]. Layering of checkpoints has also been used in many systems to recover from multiple failures or multiple types of failure. Some examples of research in this field include creating a two layer recovery scheme for shared disk architectures [1] and application specific multi-level fault tolerance schemes [11].

However, to the best of our knowledge, layering different diskless checkpointing schemes to tolerate a simultaneous failure of a random number of processors has never been explored before.

3 N-Level Diskless Checkpointing

In order to perform N-Level Diskless Checkpointing, a system must schedule diskless checkpoints for each level of recovery supported. In other words, the system would schedule checkpoints that can be used to recover from one failure, two simultaneous failures, and up to N simultaneous failures. When a checkpoint occurs for a scheme that is designed to recover from N simultaneous failures, the processors will communicate their states to each other and then perform and store sufficient Reed-Solomon encoding calculations so as to collectively be able to recover N failures. Reed-Solomon encoding schemes ensure that no matter which processors fail, the remaining processors will be able to reconstruct the state of the failed processors. For any diskless checkpoint there will be both communication and calculation overhead. If two checkpoints of different levels are scheduled for the same time, there may be a reduced combined overhead, as the larger overhead of the two may be used for both checkpoints.

Upon a detected failure, a N-Level Diskless Checkpointing system would first attempt to use the most recent one-failure recovery checkpoint to recover the state of the failed processor. Should a failure occur during

this recovery period, it would then use the most recent two-failure recovery checkpoint to recover both failed processor states. We will consider any failures that occur before recovery has completed to be simultaneous. In general, after n failures have occurred without any successful recovery, the system will rollback to the most recent N-failure recovery checkpoint and use that checkpoint to attempt to recover the failed processors. If the number of processor failures exceeds the number that is supported, the system will need to restart the process.

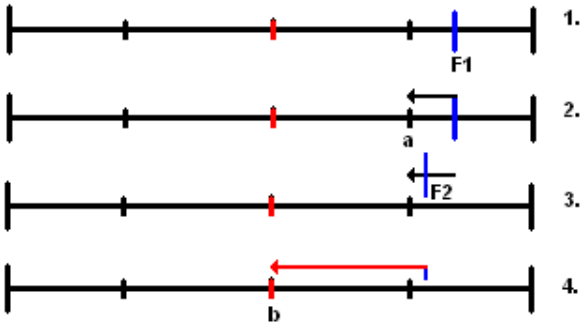

Figure 1. Flow of a two-failure recovery. 1) Failure occurs at time F1. 2) System attempts to recover to most recent one failure checkpoint (a). 3) Failure occurs while attempting recovery at F2. 4) System recovers to most recent two-failure checkpoint (b).

Multiple layers allow for additional flexibility in the use of diskless checkpointing. Previously, most diskless checkpointing schemes only allow for a single failure. The interval between diskless checkpoints in a single failure scheme is optimizable using similar techniques for stable storage checkpointing. The complexity that is added when using multiple layers of diskless checkpointing as opposed to stable storage checkpointing is that a failure during a recovery may result in needing to return to an even earlier checkpoint. In stable storage checkpointing, the system would simply restart from the same checkpoint. In diskless checkpointing, the system may need to restart from an earlier checkpoint. In this sense, multiple layer diskless checkpointing has a disadvantage against stable storage checkpointing, however the gains in not using slow stable storage make the development valuable. The possibility of falling through a checkpoint increases the importance of when checkpoints are scheduled.

In this paper we will present a method for analyzing multiple layers of diskless checkpointing, suggest a heuristic for checkpoint scheduling, and then compare the results of the analysis of an example of that scheduling heuristic against a simulation of the same schedule.

4 Performance Analysis

4.1 A Model for N-Level Diskless Checkpointing

In order to set up the problem we will assume that the system requires that all processors are functional in order to proceed with computation. This assumption is reasonable as it reflects the default mode among many implementations of Message Passing Interface (MPI). We will further assume that the system fails with a constant failure rate, leading to an exponential distribution of failures.

Let λ be the constant failure rate of the whole system, that is until at least one processor fails. Whenever a failure occurs, let it take t_{r1} to recover a single failure using a single recovery diskless checkpointing scheme. Checkpoints of the single failure recovery scheme each take a time of t_{c1} to set up, including communication costs between the processors. Whenever a single failure occurs, all processors stop executing and go into a recovery mode, which will recover to the last diskless checkpoint. Each processor may store more information than just a single checksum, so it may be possible to recover more than one failure at a time should an additional processor fail during recovery. We assume that for this to be used, an additional time of t_{c2} will be required to set up a checkpoint that can recover two errors.

Furthermore, we assume that anytime an additional processor fails, that it will take a longer time to recover than just a single processor. In fact, an additional t_{r2} will be required. In general, a scheme that can recover from i failures will require a t_{ci} to set up and an additional t_{ri} to recover.

We can treat the initial checkpointing interval question as if it were using stable storage checkpoints if we assume no more than one failure will occur at a time. The difference is that the λ will be for the system, as well as the time for setup and recovery. The approximate optimal number of checkpoints, assuming that no failures occur during recovery is shown in equation 1.

$$N = \lambda \cdot T \cdot \sqrt{\frac{1}{\lambda \cdot t_{c1} \cdot (2 + \lambda \cdot t_{c1})}} \tag{1}$$

This yields an approximate optimal expected time as shown in 2.

$$E(T_{total}) = (Nt_{c1}(1 + \frac{\lambda \cdot t_{c1}}{2}) + (\frac{\lambda \cdot T^2}{2N})) \cdot(1 + \lambda \cdot t_{r1}) + (T + Tt_{c1}\lambda) \cdot (1 + \lambda \cdot t_{r1}) \quad (2)$$

N is the value from the first equation and T is the time of one interval.

This scheme will be considered S_1, as it can recover from a single failure. Let S_i denote a scheme that is capable of recovering from i failures.

We will assume that with only $S_1, S_2, ...S_n$, only a max of n failures that are recoverable. It should be noted that n must be less than the total number of processors, otherwise the entire job must be restarted. It is possible to have some form of non-diskless checkpointing scheme set up rather than restart completely, but that is beyond the scope of this paper.

If a failure occurs during the recovery period of t_{ri-1}, then t_{ri} must be waited before the recovery period t_{ri-1} can complete. A second failure will be assumed to occur with a constant failure rate of λ. The justification for this is that any of the other machines could fail during this period. This is an overestimate, as if the original failing processor fails again, it should be a lesser amount of time than recovering for more than one failing processor.

We start by looking at what a single diskless checkpoint scheme might look like. The following assumes that we have N diskless checkpoints with an interval T between them.

$$E(T) = T \cdot (e^{-T \cdot \lambda}) + \int_0^T (t_1 + E(T) + t_{r1}) \cdot \lambda \cdot e^{-\lambda \cdot t_1} dt_1 \quad (3)$$

$$E(T) = (t_{r1} + \frac{1}{\lambda})(e^{T\lambda} - 1) \quad (4)$$

Reformulating the expected time to get through the task of length T_{total}, we first consider the case where we have only two levels of diskless recovery and assume that there is a maximum of two simultaneous errors. Let us consider E(T), or the expected time it takes to move forward a single one-level interval under these assumptions. We will assume $E(ckpt_1 - ckpt_2)$ is the expected time between the first and second levels of checkpoints.

$$E(T) = E(T|no - failures) \cdot P(no - failures) + E(T|Failure) \cdot P(Failure) \quad (5)$$

$$E(T) = T \cdot (e^{-T \cdot \lambda}) + \int_0^T (t_1 + E(T) + E(T_{r1})) \cdot \lambda \cdot e^{-\lambda \cdot t_1} dt_1 = (E(T_{r1}) + \frac{1}{\lambda})(e^{T\lambda} - 1) \quad (6)$$

$$E(T_{r1}) = (e^{-t_{r1}\lambda}) \cdot t_{r1} + \int_0^{t_{r1}} (\lambda \cdot e^{-\lambda \cdot t_2})(t_2 + t_{r2} + E(ckpt_1 - ckpt_2))dt_2 = (E(ckpt_1 - ckpt_2) + \frac{1}{\lambda} + t_{r2})(e^{t_{r1}\lambda} - 1) \quad (7)$$

This was solved for using similar mechanics to single diskless checkpoint technique. The difference is we need to factor in the expected time for recovery from the second failure before completing recovery of the first failure.

For the time being, for the calculation of $E(ckpt_1 - ckpt_2)$, we will assume that that two-failure checkpoints occur with certain one failure checkpoints in order to minimize this expectation. If the two always occur together, then this expected value will be zero. There are additional advantages to this as well. In many cases the cost of doing two types of checkpoints at the same time may only be marginally more than one of them due to decreased need for communication in the system.

An additional item to note is that the intermediary t_{r1} simply becomes a multiple of the success probability for that expectation and part of the bound for the failure rate for when the next level recovery scheme is used. If we had up to (n-1) level schemes, i.e., schemes $S_1, S_2, ..., S_{n-1}$, it would be possible to recover from n-1 failures, but the calculation of the expected value would be difficult.

Using the scheme described above, it can be noted that a recurrence relationship can be formed between the expectations of the different variables.

$$\begin{aligned} E(T_{ri}) &= (e^{-t_{ri}\lambda}) \cdot t_{ri} + \\ &\int_0^{t_{ri}} (e^{-t_{i+1}\lambda}\lambda)(t_{i+1})dt_{i+1} + \\ &\int_0^{t_{ri}} (e^{-t_{i+1}\lambda}\lambda)(E(T_{r(i+1)}))dt_{i+1} + \\ &\int_0^{t_{ri}} (e^{-t_{i+1}\lambda}\lambda)(E(ckpt_i - ckpt_{i+1}))dt_{i+1} \\ &= (E(T_{r(i+1)}) + \frac{1}{\lambda} + E(ckpt_i - ckpt_{i+1})) \cdot \\ &(e^{t_{ri}\lambda} - 1) \end{aligned} \quad (8)$$

We can note from this equation that the complexity of the calculation will be greatly simplified if the

checkpoints for the various levels occur when the checkpoint of the previous level occurs. There is the question of how to model it if the nth checkpoint fails. This is a small probability in real system, however we can model it as a n+1 checkpoint with a single checkpoint at time zero having setup and recovery time of zero. This would align with any of the checkpoint schemes, and thus could be used in conjunction with the upcoming analysis. However, this is not explored in this paper.

4.2 Calculating Expected Times Between Checkpoints

One difficulty with this setup of the analysis is that we do not necessarily know the expected time between checkpoints of two levels. To help relieve some apprehension of this approach, we now present the following bounds on the common condition that there are equal intervals within any level of checkpointing. In general, if all checkpoints occur at even spacings over the interval with N_i checkpoints overall, then this can be seen to be:

$$E(ckpt_i - ckpt_{i+1}) \leq \frac{N_{i+1} \cdot T_{Total}}{N_i^2} \cdot \sum_{k=1}^{\lfloor \frac{N_i}{N_{i+1}} \rfloor} k \qquad (9)$$

The reasoning behind this is that there is a max of $\frac{N_i}{N_{i+1}}$ checkpoints of scheme i in each interval of i+1. Each one would be an i interval more than the one before it within the i+1 interval. Each i interval is of size T/N_i, and there are N_{i+i} i+1 intervals. Since we are taking the expectation we divide by N_i to find the average. Equation 10 shows the case if all the $ckpt_{i+1}$ align with a $ckpt_i$.

$$E(ckpt_i - ckpt_{i+1}) \leq \frac{N_{i+1} \cdot T_{Total}}{N_i^2} \cdot \sum_{k=1}^{\frac{N_i}{N_{i+1}} - 1} k \qquad (10)$$

4.3 A Heuristic for Reasonable Checkpoint Times

If we consider the amount of time to set up checkpoints in addition to the length of the task, we can see that the total time spent performing the task is listed in equation 11. Equation 11 uses T as the interval between single failure recovery checkpoints, N_i is the number of i failure recovery checkpoints, and t_{ci} is the amount of time it takes to perform a checkpoint for scheme i.

$$\begin{aligned} T_{total} = \\ N_1 \cdot (T + t_{c1}) + t_{c2}N_2 ... + t_{cn}N_n \\ = N_1 \cdot (T + t_{c1}) + \sum_{i=2}^{n} t_{ci} \cdot N_i \end{aligned} \qquad (11)$$

Using this, we can calculate the actual time line for level one checkpoints, as shown in equation 12.

$$T_{real} = \frac{N_1 \cdot (T + t_{c1}) + \sum_{i=2}^{n} t_{ci} \cdot N_i}{N_1} \qquad (12)$$

Thus the total time spent is expressed in 13.

$$T_{total} = N_1 \cdot T_{real} \qquad (13)$$

Plugging this value into equation 6 yields 14.

$$E(T_{total}) = N_1 \cdot (E(T_{r1}) + \frac{1}{\lambda})(e^{T_{real}\lambda} - 1) \qquad (14)$$

Ideally at this point we would like to solve for optimal values of $N_1, N_2, ...N_n$. Unfortunately this is not easily solved. One heuristic is to perform N_i checkpoints at a regular subset of the N_{i-1} checkpoints, in order to take advantage of possible reduced overhead. A greedy heuristic may be to find the optimal value of N_1 with the assumption that there is only one failure, and then find the subsequent optimal values of the rest under the assumption that there is a max of that many failures.

The justification of this is that there will be more single failures than multiple failures. Thus, optimizing those with lower number of failures will provide a reasonably reduced overall time. The first iteration of this optimization will turn out exactly like equation 1 under the assumption that there are no additional failures and no more checkpoints, this will reduce to exactly these conditions.

5 Simulation

In order to show the advantage of layered diskless checkpointing and to validate the results of the above analysis, stochastic simulations were performed. Specifically, a simulator was written in Java designed to mimic the model of the process described in this paper. This allows comparison of the analysis for a given schedule or scheme. The simulation has parameters of the constant failure rate λ, the times for recoveries for the different schemes t_{ri}, the times for setting up checkpoints for different schemes t_{ci}, the length of

time the program would run without any failures or checkpoints, and the times in which checkpoints would occur for each scheme. This allows for great flexibility beyond what the analysis gives so that the analysis above can be compared with specific heuristics.

The simulation results are the average of one hundred thousand runs of a simulation. There were two types of simulations performed. The first compared how a two-level diskless scheme performed against a one-level, two-failure recovery scheme. The second set of simulations explores the impact of various parameters in the two-level scheme, and then compares the simulated time against the analytic bound evaluated with the same parameters.

The first simulation compared the two-level scheme against a similarly set up diskless scheme that was capable of recovering from two failures. This simulation was set for a program that would run for ten days if there were no checkpoints or failures. For the two-level checkpoint scheme, a checkpoint schedule that alternated between checkpoints that can recover one failure and two failures was used. The two-failure checkpoint scheme had a checkpoint at any point a checkpoint occurred in the two-level scheme. Checkpoints were simulated to occur every six hours, for a total of forty checkpoints in each scheme. The parameters tested were $t_{r1} = 10$ minutes, $t_{r2} = 20$ minutes, $t_{c1} = 10$ minutes, and $t_{c2} = 20$ minutes. The constant failure rate parameter, λ, was varied between 0.0 and 0.1 failures per hour, incrementing by .01.

We note that these checkpoint schedules are not optimal, as that is part of our future work. However, the parameters present a reasonable simulation to compare the two schemes.

The second type of simulation attempted to evaluate the accuracy of the analytic solution. For this reason, we performed the simulation with two different types of checkpoint schedules. In each, the task length was one time unit. In the first, we perform an analysis with two checkpoints of types single recovery failure and two recovery failure occurring at the same time. This was known as the synchronous scheme and will mean that $E(ckpt_1 - ckpt_2) = 0$. We independently varied t_{r1}, t_{r2}, t_{c2} between zero and .4 on increments of .1. λ was varied from 0 to .8 on increments of .2 on another independent trial. Whenever a variable is not the varied parameter, they have the following values: t_{r1} is .2,t_{r2} is .1, t_{c2} is .2, λ is .2. and t_{c1} is left constant at .2, as all other variables have been varied.

The second schedule we tested had four checkpoints that can recover from one failure and two that can recover from two failures, aligning at time zero and at halfway through the interval. This will be known as the 'Every Other' scheme. For this case, the $E(ckpt_1 - ckpt_2) = T/2 = .25$ We will use the same values of $t_{r1}, t_{r2}, t_{c1}, t_{c2}$ and λ as before.

The simulation results were compared against the analytic results.

6 Results

Figure 2 shows that the two-level scheme outperforms the two-failure scheme for all values of lambda tested, and it seems that it will continue to do so for other values of lambda as well.

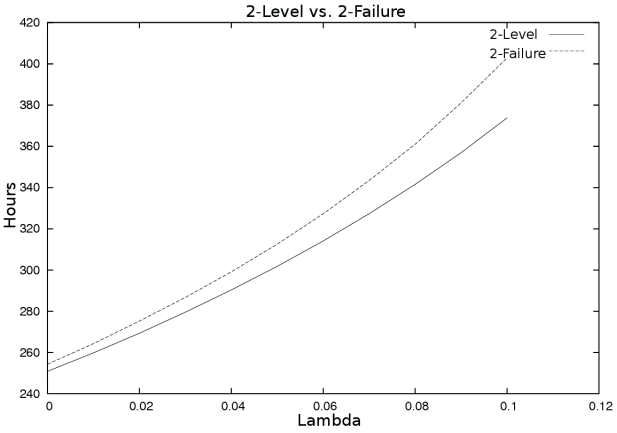

Figure 2. 2-level Scheme vs. 2-failure scheme over 10 day program

Figure 3 shows the improvement in overhead of the two-level scheme over the one-level scheme. The graph shows that there is a minimum improvement of around 14 percent for these parameters when λ is around .05 failures per hour. When λ is near zero, there is a much higher improvement as less overhead is required to set up checkpoints. When λ increases beyond this minimum, the reduced time to recover from one failure allows improved progress under the two-level scheme.

For the results comparing the simulation against the analytic bounds, in all four graphs the analysis bounds the simulation for the parameter ranges given. Further parameter ranges will be examined in future work.

It can be seen in the graph for changing λ (fig 4) that the shape appears to be held between the simulation results and the analytic results. Furthermore, we see the gain in both simulation and analytic results as the value of λ increases. While the bound is not as tight as might be hoped, it still shows that it captures the behavior of the simulation of the multi level checkpointing.

In the graphs (figure 5 and figure 6) that investigate the changing values of the time for recoveries,

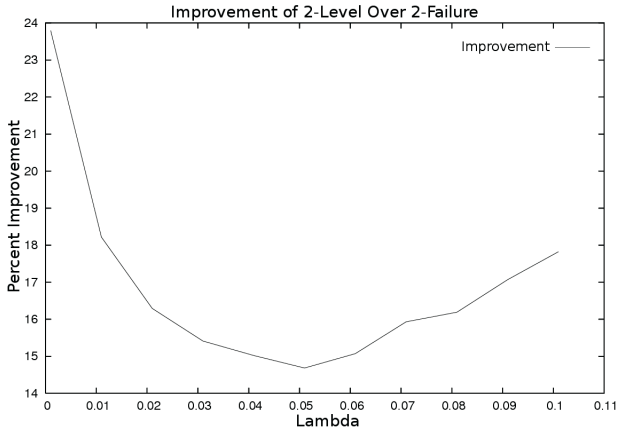

Figure 3. Percent improvement of 2-level scheme over 2-failure diskless scheme.

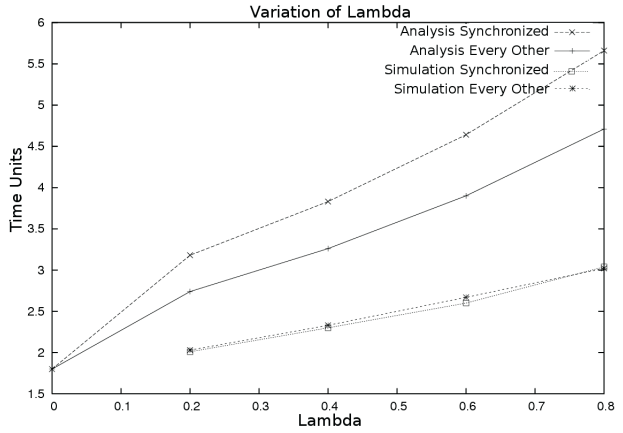

Figure 4. Total time required with variation of Lambda Comparison. Program running length is 1 unit.

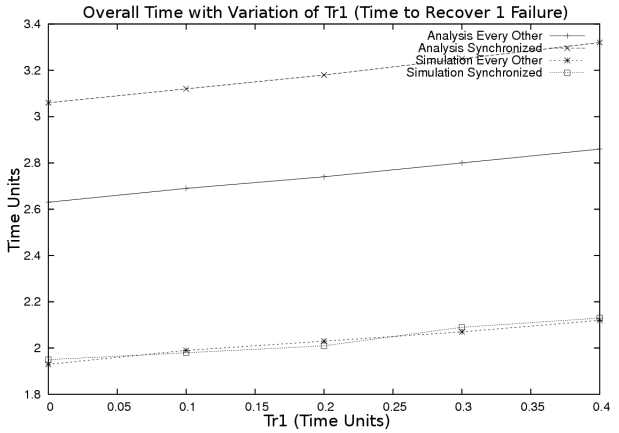

Figure 5. Total time required with variation of time to recovery for the single failure scheme. Program running length is 1 unit.

the curves are relatively flat for both analysis and for simulation. This is slightly surprising, however, as the number of bounds checked is limited, it may be that the parameters have the constant factors making these quantities subordinate to the effect of λ. With further research the true nature of the variables may be ascertained. It should also be noted the analysis is an upper bound for the simulation over these parameter ranges.

In the graph concerning varying the time to checkpoint for the two-failure scheme (figure 7) it can be seen that there is a slow increase on these parameter ranges. This again is surprising and finding better parameter ranges would greatly help decipher the nature of this variable. It is potentially troubling that the analysis for the synchronous scheme cannot be held for certain that it will continue to bound the simulation as the value of t_{c2} tends to grow. It would be more comforting if the slope of the analysis were greater than or equal to the slope of the simulation for this scheme. Continuing work will more closely examine this result. Regardless, for these parameter ranges we can see the analysis provides a reasonable bound for the simulation results.

Another interesting concept to explore in these graphs is the relationship between the two schedules. Over the interval studied it appears that the simulations perform quite similar while the analysis predicts that they should be changing. This certainly is supported in figure 4 where λ is being changed as the shape difference can be made out, however in the other three graphs there is no apparent growth.

7 Conclusion

In this paper, we explored a multiple level diskless checkpointing scheme and provided both a methodology to model the performance of the proposed scheme and a simulation engine to simulate and verify our theoretical analysis. Experimental results demonstrated that the use of the multiple level diskless checkpointing scheme is a viable way to decrease the expected execution time in a fault tolerant environment with simultaneous failures of a random number of processors.

In the future, we would like to first complete and verify the analysis experimentally with a wider focus of the full parameter ranges as well as with increased numbers of checkpoints and checkpoint layers. Additionally, it would be valuable to investigate into using

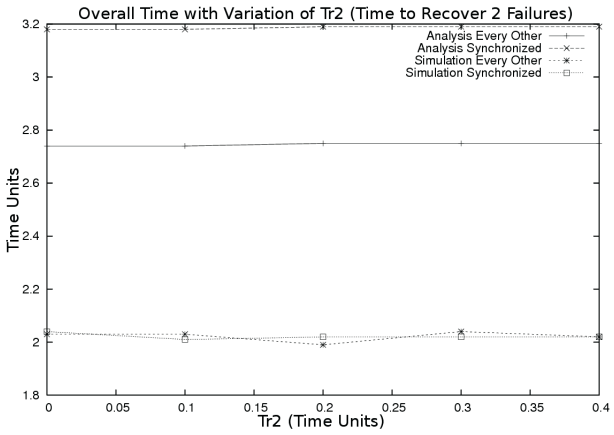

Figure 6. Total time required with variation of time to recovery for the two-failure scheme. Program running length is 1 unit.

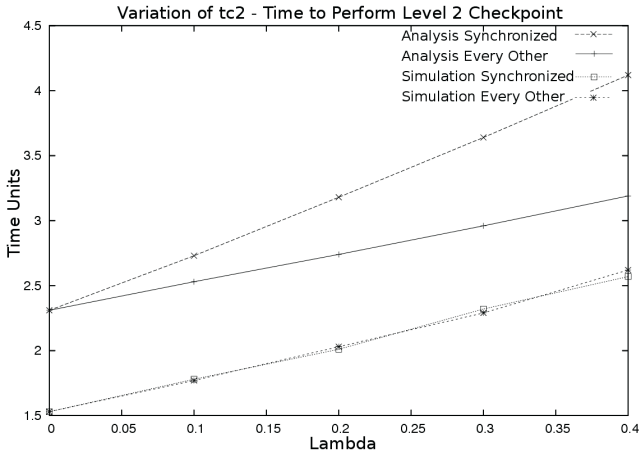

Figure 7. Total time required with variation of time to checkpoint for the two-failure scheme. Program running length is 1 unit.

multiple diskless checkpoints in conjunction with stable storage checkpoints in order to tolerate the failure of all processors.

References

[1] P. Bohannon, J. Parker, R. Rastogi, S. Seshadri, A. Silberschatz, and S. Sudarshan. Distributed multi-level recovery in main-memory databases. In *In Fourth International Conference on Parallel and Distributed Information Systems*, pages 44–55. IEEE Computer Society Press, 1996.

[2] Z. Chen. Draft:scalable self-healing algorithms for high performance scientific computing. Technical report, 2008.

[3] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra. Fault tolerant high performance computing by a coding approach. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 213–223, New York, NY, USA, 2005. ACM.

[4] C. Engelmann and A. Geist. A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform. In *CLADE '03: Proceedings of the 1st International Workshop on Challenges of Large Applications in Distributed Environments*, page 47, Washington, DC, USA, 2003. IEEE Computer Society.

[5] Y. Kim, J. S. Plank, and J. J. Dongarra. Fault tolerant matrix operations for networks of workstations using multiple checkpointing. *High Performance Computing and Grid in Asia Pacific Region, International Conference on*, 0:460, 1997.

[6] J. Plank, K. Li, and M. Puening. Diskless checkpointing. *Parallel and Distributed Systems, IEEE Transactions on*, 9(10):972–986, Oct 1998.

[7] J. S. Plank, Y. Kim, and J. Dongarra. Fault tolerant matrix operations for networks of workstations using diskless checkpointing. *Journal of Parallel and Distributed Computing*, 43(2):125–138, June 1997.

[8] J. S. Plank and K. Li. Faster checkpointing with n + 1 parity. Technical report, Knoxville, TN, USA, 1993.

[9] J. S. Plank, J. Xu, and R. H. B. Netzer. Compressed differences: An algorithm for fast incremental checkpointing. Technical Report CS-95-302, University of Tennessee, August 1995.

[10] Y. Shang, Y. Jin, and B. Wu. Fault-tolerant mechanism of the distributed cluster computers. *Tsinghua Science and Technology*, 12(Supplement 1):186 – 191, 2007.

[11] Y. Shang, B. Wu, T. Li, and S. Fang. Fault-tolerant technique in the cluster computation of the digital watershed model. *Tsinghua Science and Technology*, 12(Supplement 1):162 – 168, 2007.

[12] L. Silva and J. Silva. Using two-level stable storage for efficient checkpointing. In *Software, IEE Proceedings - Volume 145, Issue 6*, pages 198–202, 1998.

[13] L. M. Silva and J. G. Silva. An experimental study about diskless checkpointing. *EUROMICRO Conference*, 1:10395, 1998.

[14] N. H. Vaidya. Another two-level failure recovery scheme: Performance impact of checkpoint placement and checkpoint latency. Technical report, 1994.

[15] N. H. Vaidya. A case for two-level distributed recovery schemes. In *In ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 64–73, 1995.

[16] A. Ziv. *Analysis and performance optimization of checkpointing schemes with task duplication*. PhD thesis, Stanford, CA, USA, 1996.