

# Multi-Tier Checkpointing for Peta-Scale Systems

Alan Wood, Swami Nathan, Timothy Tsai, Chris Vick, Lawrence Votta – Sun Microsystems;  
Anoop Vetteth – University of Illinois\*

## Abstract

*Checkpoints are an important component of availability for supercomputer applications, but current supercomputers require multiple minutes to save a checkpoint to disk, which significantly impacts system availability. The increased storage requirements for future supercomputers will make checkpoint storage time even more of a challenge. This paper describes a new multi-tier checkpoint architecture with a fast first tier of storage (e.g., RAM) for quickly saving a checkpoint and a slower second tier of storage (e.g., disk) for robustly saving a checkpoint. Markov models are used to calculate application availability for this architecture and demonstrate the significant availability benefits of the multi-tier checkpoint approach compared to the single-tier disk checkpoint approach.*

## 1. Introduction

The “grand challenge” problems, such as simulating the birth of the universe, require a supercomputer an order of magnitude more powerful than the current supercomputers: a machine that provides petaflop-scale performance. A current Defense Advanced Research Projects Agency (DARPA) program is focused on building such a supercomputer [1]. Specifically, DARPA wants a high productivity computing system (HPCS), where the emphasis is on overall productivity rather than just compute performance. Performance on a petaflop scale is obviously an important attribute of productivity, but there are also other important attributes. Availability is extremely important because the total amount of work that can be done on any computer is the product of the performance and the amount of time the machine is doing useful computing. Minimizing the amount of human effort required to program and operate the supercomputer is another important attribute.

Checkpointing is a useful feature for both availability and minimizing human programming effort. A checkpoint is a saved application or system state from which computation can be restarted. At certain times, the system saves its entire memory and operational state (system checkpointing) or appropriate

portion thereof (application checkpointing). If a failure occurs, the computation can be restarted from the most recent saved checkpoint. Application checkpointing requires the application programmer to insert checkpoint code or calls to checkpoint library functions at appropriate points in the computation. Application checkpoints are usually inserted at a point in the computation when the “live” state is small to minimize the amount of state that has to be stored. In contrast to application checkpoints, system checkpoints do not require any action by the application programmer. System checkpoints can be saved at times determined by the system, and the entire computational state of the machine is saved. System checkpoints require saving more state than application checkpoints, but allow programmers to be more productive.

A petascale computer requires petabytes of memory. Doing a checkpoint with a petabyte of memory is a significant challenge since the time required to store a petabyte of data on disk could be many minutes to hours. Checkpoints on current supercomputers consisting of several terabytes of data take minutes to tens of minutes to complete [2], which has led to research into technologies such as incremental checkpointing [3]. The current checkpoint bottleneck is the immense I/O bandwidth required to transmit terabytes of data [10], and this is not likely to improve as I/O channel speeds are not increasing as fast as other computer technologies.

Clearly, system checkpointing for a petascale computer is a daunting challenge. Nevertheless, the availability and productivity advantages of checkpoints make them highly desirable on a petascale machine. In this paper we propose a new checkpoint architecture that contains multiple tiers of checkpoint storage and makes petascale system checkpoints feasible. The first checkpoint storage tier consists of a fast storage device, e.g., RAM, to which a checkpoint can be stored much more quickly than it can be to disk. While the computation continues on the machine, the checkpoint stored on the first storage tier can be copied to a second tier of stable storage, e.g., disk, in case of failures in the first storage tier. Additional tiers of storage, such as tape, are also possible. The new architecture is described and analyzed in this paper, and a Markov model for determining the resulting availability and optimizing the checkpoint parameters is described. Related work includes [11], which describes a single-

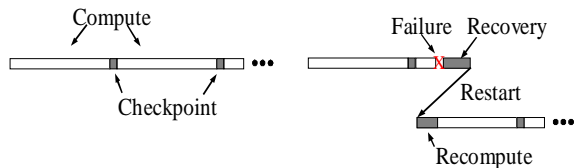
---

\* Anoop Vetteth was an intern at Sun Microsystems when this work was performed.

tier checkpoint in system memory rather than disk, and [12], which describes a multi-level checkpoint scheme for handling different types of failures. However, none of the literature we have surveyed discusses using multiple tiers of checkpoint storage to recover from failures of a checkpoint or failure of the checkpoint recovery mechanism.

## 2. Checkpointing Basics

Modern microprocessors, memory arrays, and I/O subsystems use information coding and redundancy schemes to minimize the number for faults that propagate to become observable component failures. For example, modern microprocessors have parity or ECC protection for most on-chip memory, including level 1 cache, level 2 cache, and buffers such as TLBs. They may also have instruction retry and even replicate part of the instruction and/or execution threads to mask faults [4]. Despite these advanced architectures, supercomputers suffer from the sheer quantity of hardware required to create a petascale machine and the increasing frequency of transient errors as hardware feature sizes decrease. Some current supercomputers have MTBFs on the order of a few hours, a time significantly smaller than the amount of time that may be needed to complete a “grand challenge” problem [5,6]. Under these conditions, an application would never complete unless it were able to do a useful portion of work and then continue where it left off after recovery from a failure. Checkpoints are the technology that provide the restart capability.



**Figure 1. Checkpoint phases**

As shown in Figure 1, checkpointing has two phases – (1) saving a checkpoint and (2) checkpoint recovery following a failure. To save a checkpoint, the memory and system state necessary to recovery from a failure is sent to storage. Checkpoint recovery involves restoring the system state and memory from the checkpoint and restarting the computation from the time the checkpoint was taken. The time lost to doing useful computation is the overhead time required to save a checkpoint, the time required to restore a checkpoint after a failure, and the recomputation time to replace the computation that had been performed after the checkpoint but before the failure. These times are shown in gray in Figure 1. This lost time contributes to the computer unavailability, but is a much better solution than restarting the job after every

failure. Note that this discussion applies both to system checkpoints and application checkpoints.

**Table 1. Checkpoint saving steps**

<i>Checkpoint Saving Steps</i>	<i>Discussion</i>
Decide to checkpoint	May be based on application code, a schedule, or other policy
Quiesce system	All processors must reach a safepoint, which includes appropriate handling of outstanding reads and writes. Detected errors should be handled before initiating a checkpoint.
Copy data to storage	Copy all memory for system checkpoints; the appropriate subset for application checkpoints
Commit checkpoint	Need to be certain there were no errors during data copy so that the checkpoint can be safely used for recovery
Resume computing	From the processor safepoints

**Table 2. Checkpoint recovery steps**

<i>Checkpoint Recovery Steps</i>	<i>Discussion</i>
Determine need for recovery	Normally to recover from a failure but could also be to rerun a computation from a certain point, e.g., for sensitivity analysis
Select checkpoint to use	Normally the most recent
Stop computation and reset system	Need to halt processors and reset them to a known state
Update system configuration	Determine resources to use for failure recovery, e.g., a spare processor to use in place of a failed processor
Copy data from storage	Copy all memory for system checkpoints; the appropriate subset for application checkpoints
Commit checkpoint recovery	Need to be certain there were no errors during data copy
Resume computing	From the recovered system state

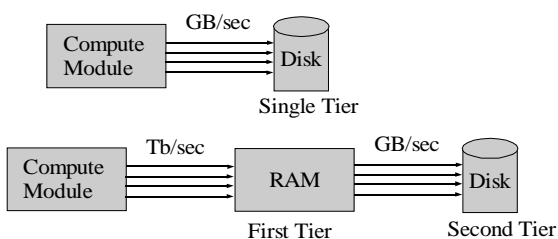
There are a number of actions involved in saving a checkpoint and doing a checkpoint recovery. The basic high-level actions are listed in Tables 1 and 2. Each

action may involve multiple messages and responses and context switches. The checkpoints described in this paper are coordinated (see [9] and [10] for a discussion of other types of checkpoints), and quiescing the system involves reaching an appropriate system or application breakpoint (e.g., barrier sync). The decision to commit a checkpoint or checkpoint recovery requires successful acknowledgment from all the system resources involved.

### 3. Checkpoint Architectures

The HPCS program envisions a petascale computer in 2010. Specifically, the DARPA goals are 2+ petaflops on the Linpack Top500 benchmark (Gaussian elimination), 6.5 petabytes per second data streams bandwidth (local vector operations), 3.2 petabytes per second bisection bandwidth (transpose a large matrix), and 64 tera-updates per second (random memory updates) [1]. An example of an architecture proposed to meet these requirements [7] includes multi-core/multi-threaded processors operating with a clock rate of several GHz each and interprocessor communication speeds of 2 terabits/second based on proximity communication [8]. Also, in this timeframe, we expect I/O channel speeds of 10 Gb/sec and 1 TB disk drives with 150 MB/sec bandwidth.

A petascale system would have about a petabyte of memory and perhaps 10-100 PB of disk storage. Considering the steps in Table 1, moving data to storage will dominate the time to save a checkpoint since even 10,000 instructions waiting for the processor to quiesce (reach a safepoint) is only a few microseconds with a Ghz clock rate. Therefore, an architecture that minimizes the time to move data to storage is likely to be optimal.



**Figure 2. Single and multi-tier checkpoint architecture**

Figure 2<sup>1</sup> shows both a standard single tier checkpoint architecture and the proposed multi-tier checkpoint architecture. A petascale system would consist of hundreds to thousands of compute modules,

<sup>1</sup> Although RAM and disk are the technologies illustrated in this paper for the first and second tier checkpoints, other storage technologies could be considered in the future. Non-volatile RAM might be used for first tier checkpoint storage to conserve power and to recover from a power failure.

each with terabits of memory. Attempting to send terabits to a disk farm using the standard single-tier checkpoint architecture would significantly impact system availability. Assume that there are 1000 compute modules, each with a TB of data (8 Tb, excluding coding overhead), and each attached to 10 I/O channels capable of 10 Gb/sec. At 100% efficiency, the I/O channels can stream the data to the disks in 80 seconds. Including overhead and inefficiencies, a checkpoint time on the order of a few minutes is reasonable based on I/O bandwidth. If there are 10 disks per I/O channel (100,000 disks total) at 150 MB/sec, the aggregate disk bandwidth per channel is 1.5 GB/sec (12 Gb/sec), which is a reasonable match for currently available 10 Gb/sec I/O bandwidth. Of course, changing the architecture or I/O resources would change the checkpoint duration, but a few minutes is a reasonable ballpark estimate.

A multi-tier checkpoint architecture streams data quickly to a first tier storage. This allows processing to resume with minimal overhead time spent doing a checkpoint. In the example architecture in [7], fiberoptic connections are used between the compute modules and I/O modules (that contain the RAM used for checkpoints) for speed, and to minimize the time spent getting the data from the compute module memory to the fiber connections, proximity interconnect [8] is used in the compute module. Using proximity interconnect and fiberoptic connections, memory can be dumped from the compute module to RAM at Tb/sec speeds. This allows the checkpoint to complete within a few seconds, a reduction of two orders of magnitude from the single-tier approach. The compute module resumes application processing much more quickly with the multi-tier approach than with the single tier approach.

After the data is stored in the first-tier RAM storage, it can then be sent to the disk farm at a much lower rate of speed as a background task while the storage system continues being used for application computation. From the standpoint of the storage system, the first-tier RAM storage can be thought of as a buffer that holds the entire checkpoint. Assuming 10 I/O channels capable of 10 GB/sec and using 10% of the I/O channel bandwidth allows the disk checkpoint to complete in approximately 20 minutes, which would not be good if it were interfering with computation but is fine as a background task.

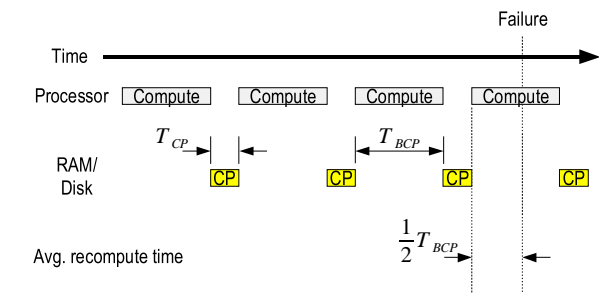
An interesting consideration for the multi-tier checkpoint architecture is the number of checkpoints that are stored in the first-tier RAM storage. If one checkpoint is stored in RAM, that checkpoint must be overwritten to store a new checkpoint. If a system failure were to occur while writing a checkpoint, the most recent disk checkpoint would have to be used for recovery since there is no valid checkpoint in RAM. Similarly, a failure in the first-tier RAM storage could

require using a disk checkpoint to recover from a system failure. In addition, a checkpoint cannot be written to RAM while the disk checkpoint is being written since the data stored in RAM must be used to write the disk checkpoint. This imposes a constraint on the frequency with which checkpoints can be stored in RAM.

If two checkpoints are stored in first-tier RAM storage, one of them remains valid while the other is being written, and a RAM checkpoint can be written at the same time a disk checkpoint is being written. This provides additional redundancy and flexibility at the cost of extra intermediate storage. Various redundancy schemes can also be considered with two or more checkpoints. Optimizing the number of RAM checkpoints and the use of those checkpoints is described in Section 5. It is also possible to consider the optimal number of disk checkpoints and to consider additional tiers of storage such as tape. Although we discuss modeling those architectures in Section 5, we assume that the disk checkpoints are 100% reliable for the purpose of reporting the availability results in Section 6.

#### 4. Single-Tier Checkpoint Model

There are many papers describing single-tier checkpoint models [9], including equations describing the optimal checkpoint parameters [5]. As a foundation for our multi-tier checkpoint models, we created a Markov model of the single-tier checkpoint



Parameter	Definition	Value
$\lambda$	System failure rate	4 times per day (6 hour MTBF[5,6])
$T_{BCP}$	Time between checkpoints	Parameter to optimize
$T_{CP}$	Time spent taking a checkpoint	2 minutes for disk model
$T_{RB}$	Time to do a rollback	3 minutes for disk model

Figure 3. Single tier checkpoint recompute timeline

architecture. The timeline for the single-tier checkpoint architecture and the model parameters is shown in Figure 3. Computing occurs for an interval  $T_{BCP}$ , and then a checkpoint is taken. Since a failure will occur halfway through the compute interval on average, the average recompute time following a checkpoint recovery is  $\frac{1}{2}T_{BCP}$ . However, if a failure occurs while taking a checkpoint, the recompute time is  $T_{BCP}$  since the previous checkpoint must be used for recovery.

The Markov model for the single-tier checkpoint architecture is shown in Figure 4. The states are described below. Note that this model assumes that the stored checkpoint is 100% reliable, which is the standard assumption in the literature, e.g., [5,10]. Stored checkpoint failures could be accommodated by additional states in the Markov model as described in the multi-tier models in Section 5.

- *Computing* state - time spent doing useful computation. Transitions to the *Checkpoint* state at a rate of  $1/T_{BCP}$ , where  $T_{BCP}$  is the time between checkpoints, and to the *Failure* state at a rate of  $\lambda$ , the system failure rate.
- *Checkpoint* state - time spent taking a checkpoint ( $T_{CP}$ ). Transitions to the *Computing* state at a rate of  $1/T_{CP}$  and to the *Checkpoint Failure* state at a rate of  $\lambda$ .
- *Failure* state - time spent doing a checkpoint recovery from a system failure. The time to recover is the time to do a rollback,  $T_{RB}$ , and the time spent recomputing the work that had been done after the checkpoint but before the failure, which is shown by the transition from the *Recompute1* state.
- *Checkpoint Failure* state - the same as the *Failure* state, except that the failure occurred while taking a

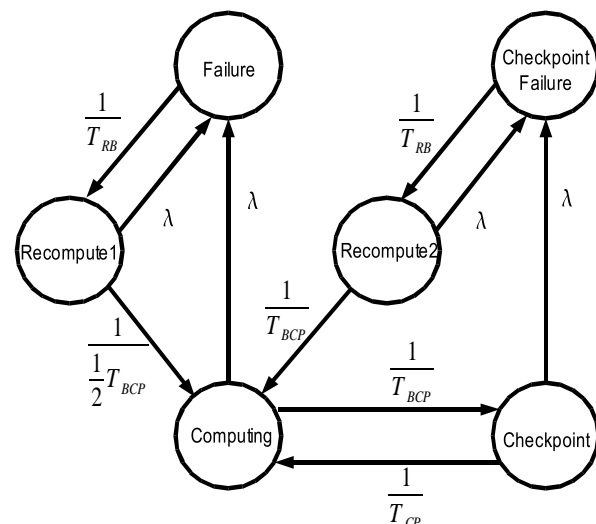
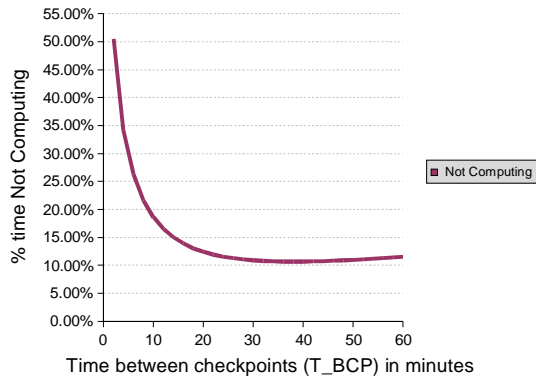


Figure 4. Single tier checkpoint markov model



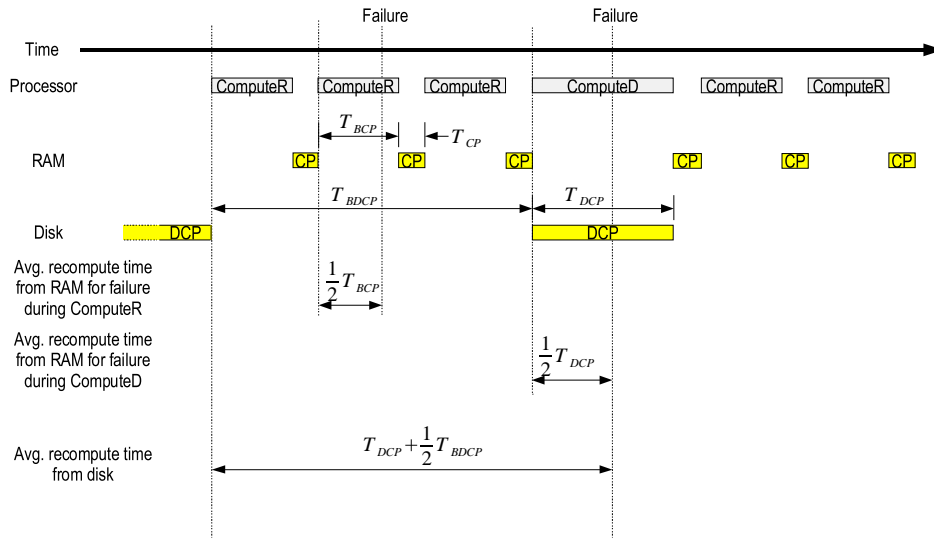
**Figure 5. Time between checkpoints ( $T_{BCP}$ ) in minutes**

checkpoint.

- *Recompute1* and *Recompute2* states - time spent recomputing lost work, shown in Figure 3. On average the recompute time is  $\frac{1}{2} T_{BCP}$  for checkpoint recovery from a failure that occurred during the

*Computing* state but is  $T_{BCP}$  for checkpoint recovery from a failure that occurred during the *Checkpoint* state since the recovery must be done from the previous checkpoint. Note that a system failure can occur during a checkpoint recovery

To help validate the model, we used it to compute the optimal time between checkpoints and compared it to the analytical results in the literature. For a time to checkpoint of 2 minutes, system MTBF of 6 hours, and rollback time of 3 minutes (2 minutes to copy the data and a minute for the other steps), the optimal time between checkpoints is 38.1 minutes. This is the same time calculated using the Equation 6 in [5]. For the optimal time between checkpoints, the percentage of time spent in the *Computing* state is 89.3%. Equation 12 in [5], which corrects for the fact that a checkpoint is not saved after the last compute interval in Figure 3 since the job is complete, calculates an optimal time between checkpoints as 36.1 minutes. For our parameters, we get the same availability result to two decimal places with a time between checkpoints between 35-40 minutes as shown in Figure 5. We also get the same availability result to two decimal places



Parameter	Definition	Value
$\lambda$	System failure rate	4 times per day (6 hour MTBF)
$T_{BCP}$	Time between RAM checkpoints	Parameter to optimize
$T_{CP}$	Time spent taking a RAM checkpoint	2 seconds
$T_{RB}$	Time to do a rollback from RAM	5 seconds
$T_{BDCP}$	Time between disk checkpoints	Parameter to optimize
$T_{DCP}$	Time spent taking a disk checkpoint	2 minutes
$T_{DRB}$	Time to do a rollback from disk	3 minutes
$P_F$	Probability that the RAM checkpoint recovery is unsuccessful	0.05

**Figure 6. Multi-tier checkpoint recompute timeline, single RAM checkpoint**

when we remove the failure transitions from the recomputing states, i.e., assuming no failures during checkpoint recovery, which is the assumption the analytical models make.

To motivate the multi-tier checkpoint architecture, consider the model shown in Figure 4 with a RAM checkpoint instead of a disk checkpoint. For a time to checkpoint of 2 seconds, system MTBF of 6 hours, and rollback time of 5 seconds (2 seconds to copy the data and 3 seconds for the other steps), the optimal time between checkpoints is 4.9 minutes. For the optimal time between checkpoints, time spent in the Computing state is 98.6%, a significant improvement over the disk checkpoint numbers. While this is not a fair comparison since the disk checkpoint is a more permanent checkpoint, it indicates the potential improvement that the multi-tier architecture can provide and is an upper bound for the multi-tier architecture availability. If there were multiple RAM checkpoints in the single-tier architecture, it would be more similar to the disk checkpoint. The multi-tier architectures in Section 5 illustrate that concept.

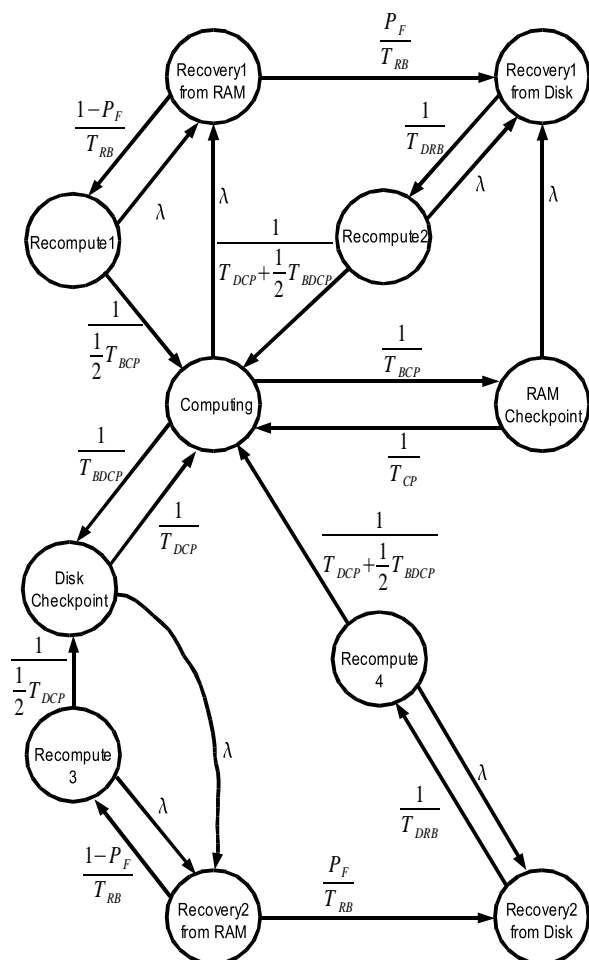


Figure 7. Multi-tier checkpoint markov model

## 5. Multi-Tier Checkpoint Model

The Markov model described in the previous section was extended to model the multi-tier checkpoint architecture described in Section 3. The timeline for the multi-tier checkpoint architecture with a single RAM checkpoint and the model parameters are shown in Figure 6. Computing occurs for an interval  $T_{BCP}$  (ComputeR in the figure), and then a RAM checkpoint is taken. When the first RAM checkpoint is complete, a disk checkpoint begins as a background task (other policies are possible) and completes after a time  $T_{DCP}$  (ComputeD in the figure). Computing and RAM checkpoints alternate for an interval  $T_{BDCP}$ , at which time another disk checkpoint begins. The average recompute time following a RAM checkpoint recovery is  $\frac{1}{2}T_{BCP}$ . However, if a failure occurs while taking a disk checkpoint, the recompute time is  $\frac{1}{2}T_{DCP}$  since the previous RAM checkpoint must be used for recovery. Other recompute times are discussed following the Markov diagram.

The Markov model for the multi-tier checkpoint architecture with a single RAM checkpoint is shown in Figure 7. The states are described following the figure. Note that this model assumes that the disk checkpoint recovery is 100% reliable, but RAM checkpoint recovery is allowed to fail.

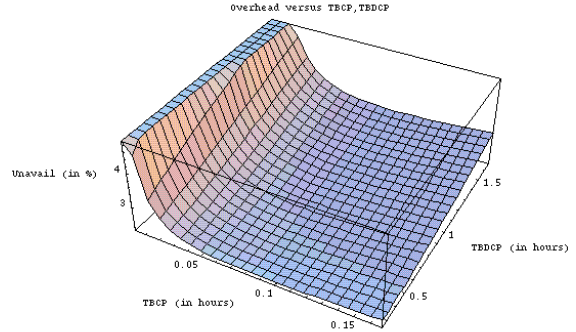
Note that the first 3 states described are similar to states in Figure 3.

- *Computing* state - time spent doing useful computation. Transitions are the same as in Figure 3, except that both RAM checkpoints and disk checkpoints are stored.
- *Disk Checkpoint* state - time spent taking a disk checkpoint ( $T_{DCP}$ ). Transitions are the same as in Figure 3, except that we attempt to rollback from RAM if a failure occurs.
- *Recovery1 from RAM* state - time spent doing a RAM checkpoint recovery from a system failure (Figure 3 *Failure* state). Transitions are the same as in Figure 3, except that we include the possibility of a RAM checkpoint recovery failure (probability  $P_F$ ), in which case we have to do a disk checkpoint recovery.
- *RAM Checkpoint* state - time spent taking a RAM checkpoint ( $T_{CP}$ ). Transitions to the *Computing* state at a rate of  $1/T_{CP}$  and to the *Recovery1 from Disk* state at a rate of  $\lambda$ .
- *Recovery1 from Disk* state - time spent recovering from a disk checkpoint when no RAM checkpoint is available, e.g., there was a failure during a RAM checkpoint or while attempting to rollback from a RAM checkpoint. Transitions to the *Recompute 2* state at a rate of  $1/T_{DRB}$ , which then transitions to the

computing state time at a rate of  $1/T_{BCDP}$ .

- *Recovery2 from RAM state* – essentially the same as the *Recovery1 from RAM state*, except that the failure occurred while taking a disk checkpoint, so the average recompute time is  $\frac{1}{2}$  the time to take a disk checkpoint rather than  $\frac{1}{2}$  the time between disk checkpoints.
- *Recovery2 from Disk state* – same as *Recovery1 from Disk state*, except the failure occurred while taking a disk checkpoint, so the average recompute time is longer by  $\frac{1}{2}$  the time to take a disk checkpoint.
- *Recompute 1, 2, 3, and 4* - time spent recomputing after a RAM or disk checkpoint recovery. See Figure 6 for an explanation of the recovery times.

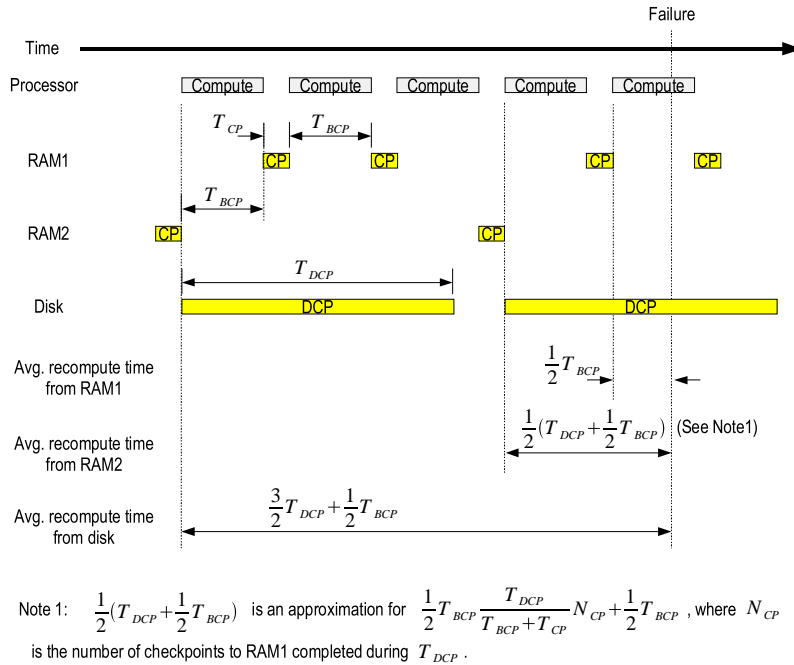
To determine the optimal times between checkpoints, we used a mathematics package to solve the Markov model equations. Since there are two parameters to optimize ( $T_{BCP}$  and  $T_{BCDP}$ ), we used the partial derivatives with respect to those quantities in the Markov model equations to find the optimal values. For the parameter values shown in Figure 6, the optimal time between RAM checkpoints is 5.2 minutes, and the optimal time between disk checkpoints is 23.1 minutes. For the optimal time between checkpoints, time spent in the Computing + Disk CP states is 97.6%, and time spent in the Computing + Disk CP + CP states (non-failure or failure recovery states) is 97.8%. Unavailability (time not spent in the Computing + Disk CP states) as a function of the time between checkpoints is shown in Figure 8. The results are fairly



**Figure 8. Unavailability vs. time between multi-tier checkpoints**

insensitive to variations in those parameters over a wide range of values.

There are many other multi-tier checkpoint architectures and model extensions that can be considered. A second RAM checkpoint could be included, which would add states to recover from this second checkpoint when the first failed, although the final recovery would still be from disk. Multiple disk checkpoints could be added to model disk checkpoint failures. Another tier of storage, e.g., tape could be added. We extended the model to cover the case in which there is a second RAM checkpoint, and the policy is to continually take disk checkpoints. This

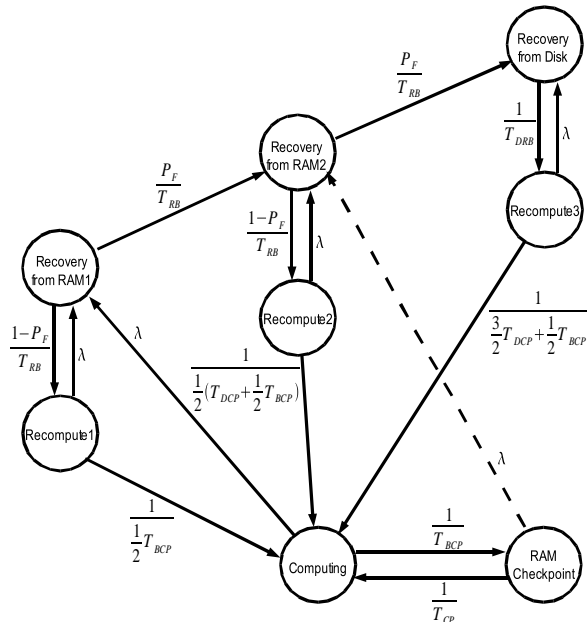


**Figure 9. Multi-tier checkpoint timeline**

policy has the benefit of keeping the disk checkpoint as current as possible and using a constant amount of I/O resources for the background disk checkpoints. It also turns out to be within 0.1% of the maximum possible availability.

With a second RAM checkpoint, the model becomes more complex due to all the possible recovery states. A timeline that demonstrates the policy of keeping the disk checkpoint as current as possible is shown in Figure 9. Two RAM checkpoints (RAM1 and RAM2) and a disk checkpoint are shown along with the average recompute times from each checkpoint. The average recompute times are used in Figure 10, which is a Markov model for a multi-tier checkpoint architecture with two RAM checkpoints. The states and parameters in Figure 10 are the same as in Figure 7 except that Figure 10 includes a *Recovery from RAM2* state, which represents the recovery from the second RAM checkpoint if the recovery from the first one fails. The probability of successful recovery from the second checkpoint is assumed to be 90% instead of 95% because there is the possibility of hitting the same failure mechanism, e.g., software.

For the parameter values shown in Figure 6, the optimal time between RAM checkpoints is 5.0 minutes. There is no optimal time between disk checkpoints since they are taken continuously. For the optimal time between checkpoints, time spent in the Computing + RAM Checkpoint states is 98.4%, and time spent in the non-failure or failure recovery states is 99.2%.



**Figure 10. Multi-tier checkpoint markov model with two RAM checkpoints**

## 5. Availability Results and Sensitivity Studies

There are at least two ways to look at availability in the context of checkpointing – machine availability and application availability. Table 3 contains the basic checkpoint states and their relationship to machine availability and application availability. Note that there are several states in which the machine is operating but the application is not making computational progress. The application does make progress when saving a second-tier checkpoint (saving a RAM checkpoint to disk) because it is done in the background. We believe it is important to report application availability when comparing various checkpoint architectures and schemes. For some organizations, it may also be interesting to look at the amount of time spent computing and saving a first-tier checkpoint because resource allocation for planned, short duration overhead like saving checkpoints is usually easily accomplished

**Table 3. Availability definitions**

System State	Machine Availability	Application Availability	Planned Operation
Computing	Yes	Yes	Yes
Saving First-Tier Checkpoint	Yes	No	Yes
Saving Second-Tier Checkpoint	Yes	Yes	Yes
Restoring from Checkpoint	Yes	No	No
Recomputing after Checkpoint Recovery	Yes	No	No
System Failure	No	No	No

**Table 4. Availability results**

Checkpoint Configuration	Application Availability	Planned Operation Percentage
Single-Tier Disk Checkpoint*	89.3%	94.0%
Single-Tier RAM Checkpoint*	98.6%	99.3%
Multi-Tier Checkpoint with 1 RAM checkpoint	97.6%	97.8%
Multi-Tier Checkpoint with 2 RAM checkpoints	98.4%	99.2%

\*Assumes the checkpoint is always available and checkpoint recovery never fails.

but long duration checkpoint recovery may cause problems. Therefore, the results shown in Table 4 are stated in terms of application availability and planned operation percentage. The difference between the two metrics indicates that nearly half the application unavailability is due to time spent saving the first-tier checkpoint. Note that the models in this paper and the standard literature do not include a system failure state since it is assumed that all failures can be recovered from a checkpoint, so machine availability is 100% for all models.

The results in Table 4 indicate that multi-tier checkpoints can significantly improve application availability for the parameter values and policies described in this paper. Since it is assumed that the single-tier RAM checkpoint is always available and checkpoint recovery never fails, it will always have better availability than the multi-tier models in which the RAM checkpoint can fail. Therefore, the multi-tier architecture with two RAM checkpoints is a nearly optimal multi-tier architecture per the results in Table 3. This indicates that other policies or additional RAM checkpoints will not significantly improve the multi-tier checkpoint application availability. Since these models do not consider a system failure state, the failure of the last available checkpoint is not allowed.

With one RAM checkpoint, the multi-tier checkpoint application availability is somewhat sensitive to the probability of a successful rollback recovery from the RAM checkpoint as shown in Figure 11. This indicates that using highly reliable RAM storage (e.g., chipkill, described in [4]) and redundant paths to access the RAM checkpoint is important. Adding a second disk checkpoint provides redundancy and significantly decreases this sensitivity as shown in Figure 11. In essence, the second checkpoint provides the highly reliable RAM storage, thus improving the application availability.

The RAM checkpoints are stored on disk as a background task, so the disk checkpoint state is considered as an application available state. However, the disk checkpoint is using a portion of the I/O

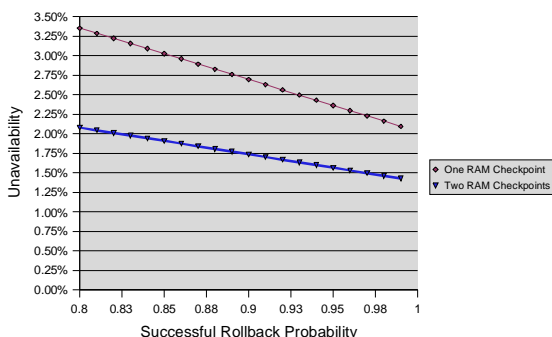


Figure 11. Unavailability vs. successful rollback probability

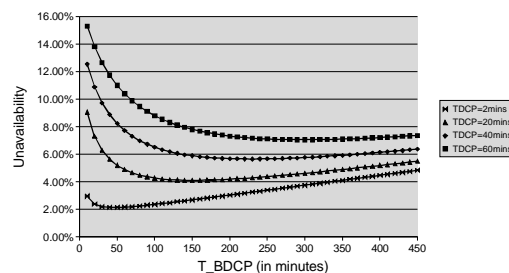


Figure 12. Unavailability vs. time between disk checkpoints for one RAM checkpoint

resources, so it is possible to consider the disk checkpoint state as a partially available state. Since we assumed that 10% of the I/O channels would be used for copying the RAM checkpoint to disk in the single RAM checkpoint model (20 minutes to copy to disk), we considered the disk checkpoint state as 90% available and evaluated the application availability as the time in the computing state plus 90% of the time in the disk checkpoint state. The results for various utilization of the I/O subsystem are shown in Figure 12. It can be seen that the optimal time between disk checkpoints is approximately 6x the time to take a disk checkpoint. The disk checkpoint time of 2 minutes shown in Figure 12 is the minimum time and assumes either 100% of the I/O channels used for copying the disk checkpoint or special dedicated I/O channels for copying the disk checkpoint.

**Cost tradeoffs.** Given that a multi-tier checkpoint architecture is feasible, the number of checkpoints to have in the first storage tier is a cost/availability tradeoff. A petabyte of RAM to provide a first-tier checkpoint may seem expensive but not when compared with an extra 8% application availability for a two-petaflop supercomputer. A second petabyte of RAM to store an extra checkpoint provides an additional 1% application availability and may provide a cost savings by allowing fewer I/O channels since the disk checkpoint can be done as a background task. In the future it is even possible to contemplate a single-tier architecture consisting of multiple checkpoints stored in highly reliable, non-volatile RAM or other storage media, and no disk checkpoints.

## 6. Summary

This paper has described a new multi-tier checkpoint architecture and demonstrated how Markov models can be used to calculate application availability for this architecture. The models demonstrate the significant availability benefits of the multi-tier checkpoint approach compared to the single-tier checkpoint approach. The availability achieved via the multi-tier approach makes productive petascale

computing more of a reality.

## Acknowledgments

This material is based on work supported by the US Defense Advanced Research Project Agency under contract No. NBCH3039002. The authors would like to thank Ravi Iyer and Zbigniew Kalbarczyk at the University of Illinois for many illuminating conversations regarding checkpoint recovery.

## References

1. The NITRD Program: FY2004 Interagency Coordination Report, Second Printing - October 2004, pp31-32, [http://www.hpcc.gov/pubs/20041020\\_icr.pdf](http://www.hpcc.gov/pubs/20041020_icr.pdf). See also <http://www.highproductivity.org>.
2. J. Urbanic, Large Scale Parallel I/O and Checkpointing, *New Methods for Developing Peta-scalable Codes Workshop*, [http://www.psc.edu/training/PPS\\_May04/](http://www.psc.edu/training/PPS_May04/), May 3-4, 2004.
3. S. Agarwal, R. Garg, M. Gupta, and J. Moreira, Adaptive Incremental Checkpointing for Massively Parallel Systems, In *International Conference on Supercomputing ICS'04*, Saint-Malo, France, June 26-July 1, 2004.
4. M. Mueller, L. Alves, W. Fischer, M. Fair, and I Modi, RAS Strategy for IBM S/390 G5 and G6, *IBM Journal for Research and Development*, Vol. 43, No. 5/6, September/November 1999, pp. 875-887.
5. J. Daly, A Model for Prediction the Optimum Checkpoint Interval for Restart Dumps. *ICCS 2003, LNCS 2660 Proceedings 4 (2003)*, pp. 3-12.
6. C. Lu and D Reed, Assessing Fault Sensitivity in MPI Applications, In *Supercomputing Conference SC2004*, Pittsburgh, PA, Novemer 6-12, 2004.
7. J. Mitchell, Sun High Productivity Computing Research Program, [http://research.sun.com/sunlabsday/docs/talks/1.01\\_Mitchell.pdf](http://research.sun.com/sunlabsday/docs/talks/1.01_Mitchell.pdf).
8. R. Drost, R Hopkins, R. Ho, and I. Sutherland, Proximity Communication, *IEEE Journal of Solid-State Circuits*, Vol. 39, No. 9, September 2004, pp. 1529-1535.
9. E. Elnozahy, D. Johnson, and Y. Wang, A Survey of Rollback-Recovery Protocols in Message Passing Systems, Technical Report CMU-CS-96-181, Carnegie Mellon Univ., October 1996.
10. E. Elnozahy and J. Plank, Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery, *IEEE Transactions on Dependable and Secure Computing*, Vol. 1, No 2, April-June 2004, pp 97-108.
11. J. Plank, K. Li, and M. Puening, Diskless Checkpointing, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 10, October 1998, pp 972-986.
12. N. Vaidya, A Case for Two-Level Distributed Recovery Schemes, *Proc. ACM SIGMETRICS Conf. Measuring and Modeling Computer Systems*, Ottawa, May 1995.