J Supercomput (2008) 46: 150–180
DOI 10.1007/s11227-007-0162-0

Optimization of checkpointing-related I/O for high-performance parallel and distributed computing

Rajagopal Subramaniyan · Eric Grobelny · Scott Studham · Alan D. George

Published online: 15 December 2007

Abstract Checkpointing, the process of saving program/application state, usually to a stable storage, has been the most common fault-tolerance methodology for high-performance applications. The rate of checkpointing (how often) is primarily driven by the failure rate of the system. If the checkpointing rate is low, fewer resources are consumed but the chance of high computational loss is increased and vice versa if the checkpointing rate is high. It is important to strike a balance, and an optimum rate of checkpointing is required. In this paper, we analytically model the process of checkpointing in terms of mean-time-between-failure of the system, amount of memory being checkpointed, sustainable I/O bandwidth to the stable storage, and frequency of checkpointing. We identify the optimum frequency of checkpointing to be used on systems with given specifications thereby making way for efficient use of available resources and maximum performance of the system without compromising on the fault-tolerance aspects. Further, we develop discrete-event models simulating the checkpointing process to verify the analytical model for optimum checkpointing. Using the analytical model, we also investigate the optimum rate of checkpointing

An earlier version of this paper appeared in Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications, June 2006.

R. Subramaniyan (✉) · E. Grobelny · A.D. George
High-performance Computing and Simulation (HCS) Research Laboratory, Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611-6200, USA
e-mail: subraman@hcs.ufl.edu

E. Grobelny
e-mail: grobelny@hcs.ufl.edu

A.D. George
e-mail: george@hcs.ufl.edu

S. Studham
National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6006, USA
e-mail: studham@ornl.gov

for systems of varying resource levels ranging from small embedded cluster systems to large supercomputers.

Keywords Checkpointing · Fault tolerance · Modeling · High-performance computing · Parallel computing · Distributed computing · Supercomputing · Technology growth

1 Introduction

The enormity and complexity of scientific and other large-scale applications have increased leaps and bounds in the past few decades, often requiring on the order of hundreds or thousands of computational hours for successful completion. With such long-running applications and usage of large volumes of resources, failures are bound to occur in the system, and fault tolerance and recovery procedures are hence required.

In general, computation of large-scale scientific applications can be divided into three phases: start-up, computation, and close-down, with I/O existing in all phases. A typical I/O pattern has the start-up phase dominated by reads, close-down by writes, and computation phase by both reads and writes. The primary questions of importance with relevance to I/O involved in the three phases are when, how much and how often? These questions can be addressed by segregating I/O into two portions: productive I/O and defensive I/O [2]. Productive I/O is the writing of data that the user needs for actual science such as visualization dumps, traces of key scientific variables over time, etc. Defensive I/O is employed to manage a large application executed over a period of time much larger than the platform's Mean-Time-Between-Failure (MTBF). Defensive I/O is only used for restarting a job in the event of application failure in order to retain the state of the computation, and hence the forward progress since the last application failure. Thus, one would like to minimize the amount of resources devoted to defensive I/O and computation lost due to platform failure. As the time spent on defensive I/O (backup mechanisms for fault tolerance) is reduced, the time spent on useful computations will increase. This philosophy applies to high-performance distributed computing in the majority of environments ranging from supercomputing platforms to small embedded cluster systems, although the impact varies depending on the system and other resource constraints.

The impact of productive I/O on I/O bandwidth requirements can be reduced by better storage techniques and to some extent through improved programming techniques. However, it has been observed that defensive I/O dominates productive I/O in large applications with about 75% of the overall I/O being used for checkpointing, storing restart files, and other such similar techniques for failure recovery [2]. Hence, by optimizing the rate of defensive I/O, we can reduce the overall I/O bandwidth requirement. Another advantage is that the optimizations used to control defensive I/O would be more generic and not specific to applications and platforms. However, reducing defensive I/O is a significant challenge.

Checkpointing of a system's memory to mitigate the impact of failures is a primary driver of the sustainable bandwidth to high-performance filesystems. Checkpointing

refers to the process of saving program/application state, usually to a stable storage (i.e., taking the snapshot of a running application for later use). Checkpointing forms the crux of rollback recovery and hence fault-tolerance, debugging, data replication and process migration for high-performance applications. The amount of time an application will tolerate suspending calculations to perform a checkpoint is directly related to the failure rate of the system. Hence, the rate of checkpointing (how often) is primarily driven by the failure rate of the system. If the checkpointing rate is low, fewer resources are consumed but the chance of high computational loss (both time and data) is increased. If the checkpointing rate is high, resource consumption is greater but the chance of computational loss is reduced. It is important to strike a balance and an optimum rate of checkpointing is required. Finding a balance is a difficult problem even in traditional ground-based high-performance computing (HPC) with fewer failures and more resources. The problem is aggravated for HPC with embedded cluster systems in harsh environments such as space with more failures and less resources.

In this paper, we provide a brief summary of growth trend of various technologies involved in HPC to highlight the importance of I/O usage. Then we analytically model the process of checkpointing in terms of MTBF of the system, amount of memory checkpointed, sustainable I/O bandwidth and frequency of checkpointing. We identify the optimum frequency of checkpointing to be used on systems with given specifications, thereby making way for efficient use of available resources, and gain maximum performance of the system without compromising on the fault tolerance aspects. Further, we develop discrete-event models simulating the checkpointing process to verify the analytical model for optimum checkpointing. The simulation models are developed using the Mission Level Designer (MLD) simulation tool from MLDesign Technologies Inc. [13]. In the simulation models, we use performance numbers that represent systems ranging from small cluster systems to large supercomputers. It must be emphasized here that this work is not an effort to develop a checkpointing library; rather, the contribution is to provide a model to checkpoint (using any already available method) at an optimum rate to reduce the usage of resources and improve effective computation time.

The remainder of the paper is organized as follows. Section 2 provides background on the growth trends of technologies to study the effectiveness of our research to improve I/O usage. Section 3 briefly highlights why checkpointing is the most common methodology of providing fault tolerance. In Sect. 4, the checkpointing process is analytically modeled to identify the optimum frequency of checkpointing to be used on systems with given specifications. Section 5 describes the simulation models that we develop to verify our analytical models, while the results derived from the analytical model are provided in Sect. 6. Section 7 provides conclusions and directions for future research.

2 Background on technology growth trends

In this section, we study the growth trend of technologies involved in high-performance computing (HPC), highlighting the poor growth of sustainable I/O band-

width. The poor growth of I/O bandwidth compared to other technologies substantiates our approach to reduce resource consumption and improve useful computation time by optimizing checkpointing-related defensive I/O.

Moore observed an exponential growth in the number of transistors per integrated circuit and predicted this trend would continue [4]. He made this famous observation in 1965, just 4 years after the first planar integrated circuit was discovered. This doubling of transistors every eighteen months, referred to as "Moore's law," has been maintained and still holds true. Similar to other exponentially growing systems, Moore's law can be expressed as a mathematical equation with a derived compound annual growth rate. Let x be a quantity growing exponentially, in this case the number of transistors per integrated circuit, with respect to time t as measured in years. Then, $x(t) = x_0 e^{kt}$ where k is the compound annual growth rate and the rate of change follows $\frac{dx}{dt} = kx$. For Moore's law, the compound annual growth rate can be established as $k_{moore} = 0.46$ (with $t = 1.5$, and $\frac{x}{x_0} = 2; k = \frac{\ln(2)}{1.5} = 0.46$). However, this compound annual growth rate is not the same for the other technologies involved in HPC. We briefly overview the growth of several technologies including CPU processing power, disk capacity, disk performance, and sustained bandwidth to/from disks in the remainder of this section.

2.1 CPU processing power

Figure 1 shows the LINPACK [5, 9] rating for the tenth most powerful computer in the world for several years as ranked by the Top500 [17] list. The compound annual growth rate can be established as $k_{HPC} = 0.58$ (values of k for the technologies are calculated as shown for Moore's law). We picked the tenth computer for no special reason except that in our opinion, the computers at the very top might have been custom tuned and hence may not provide the general trend.

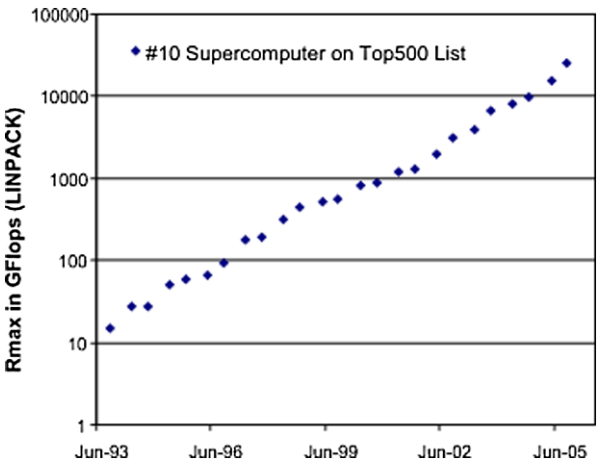


Fig. 1 Supercomputer performance over time

2.2 Disk capacity and performance

Figure 2 shows the capacity of a 7,200 RPM disk drive over time. Areal density of hard drives has grown at an impressive compound annual growth rate of $k_{IO\_cap} = 0.62$, and has accelerated to greater than a $k_{IO\_cap} = 0.75$ rate since 1999 as shown in the figure. While the compound annual growth rate of areal density of magnetic disk recording has increased at an average of over 60% annually, the maximum number of random I/Os per second that a drive can deliver is improving at a rate of less than $k_{IO\_perf\_io} = 0.20$.

As seen in Table 1, *access density* (the ratio of performance, generally measured in I/Os per second vs. capacity of the drive) has steadily declined while the capacity has increased substantially [1, 3, 7, 14]. Vendors are making small high-performance drives such as a 15,000 RPM 73.4 GB drive from Seagate with an advertised seek time of 3.6 ms leading to an access density of about 3.8 IO/s per GB. As of April 2006, these high performance drives cost $3.3 per GB [1], a factor of 11 higher than the $0.36 per GB for low-cost commodity drives [14]. The factor of 9 increase in performance is offset by the factor of 9 increase in cost, leaving most architects to select commodity drives for the additional capacity.

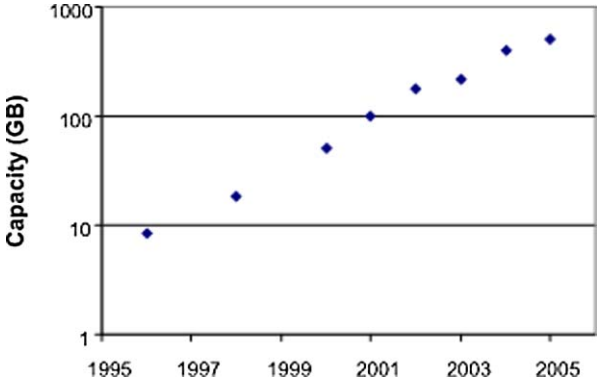


Fig. 2 Hard drive capacity over time

Table 1 Disk performance growth over time

Year	Drive	Seek time (ms)	I/Os per sec	Capacity (GB)	I/Os per sec per GB
1964	2314	112.5	8.9	0.029	306.50
1975	3350	36.7	27.2	0.317	86.00
1987	3380K	24.6	40.7	1.890	21.50
1996	3390-3	23.2	43.1	8.520	5.10
1998	Cheetah IS	18.0	55.6	18.200	3.10
2001	WD1000	8.9	112.4	100.000	1.10
2002	180GXP	8.5	117.6	180.000	0.70
2004	Deskstar 7K400	8.5	117.6	400.000	0.30
2005	Deskstar 7K500	8.5	117.6	500.000	0.24

2.3 Sustained bandwidth to/from disks

Sustained bandwidth from a disk relative to capacity of the disk has also continued to decline. Figure 3 shows the sustained transfer rate in MB/s for the 15,000 RPM, 37 GB disk drive from Seagate [3].

Although vendors provide the peak performance number in general, the average and minimum sustained performance can be significantly lower. Figure 4 highlights the minimum, average and maximum performance for typical disk drives introduced between 1995 and 2004. The average sustainable bandwidth from a hard disk drive has grown at an annual compounded growth rate of $k_{IO\_perf\_BW} = 0.26$ per year.

2.4 I/O performance

The technologies pertinent to HPC discussed in this section thus far are all growing exponentially although some are growing substantially slower than others. Table 2 summarizes the rate of change for these technologies.

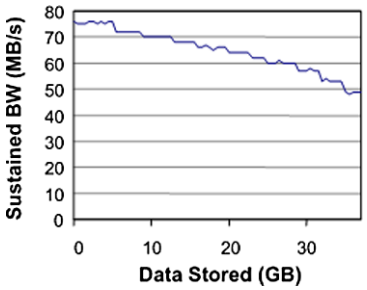


Fig. 3 Performance of a single disk with respect to amount of data stored

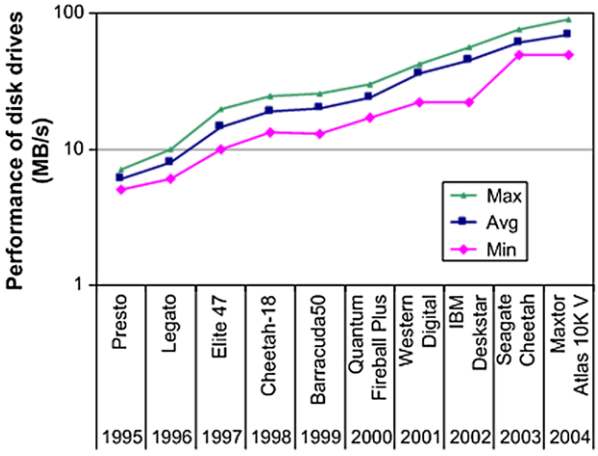


Fig. 4 Growth of disk drive bandwidth over time

Table 2 Growth rate of computing technologies

Technology		Growth rate	Time to double (months)
Transistors per integrated circuit	k_{moore}	0.46	18
LINPACK on #10 supercomputer	k_{HPC}	0.58	14
Capacity of hard drives	k_{IO_cap}	0.62	13
Cost per GB of storage	k_{IO_cost}	−0.83	10
Performance of hard drive in I/Os per second	$k_{IO\_perf\_io}$	0.20	42
Performance of hard drive in bandwidth	$k_{IO\_perf\_BW}$	0.26	32

3 Optimal usage of I/O

The performance of disk drives measured in both I/Os per second and sustained bandwidth is not keeping up with other technology trends. Assuming current systems meet I/O performance needs and for the balance of bandwidth per computational power to be maintained, we can use the formula $d(t) = e^{t(k_{HPC} - k_{IO\_perf})}$ to calculate number of more disk drives that will be needed in order to maintain that balance. In order to show the importance of efficient usage of I/O resources, in Fig. 5, we show how many disks will be required to work in parallel to maintain system balance in the coming years. Given these growth trends, we will need to use 46 times more disks in 10 years in order to maintain the same balance of I/Os per second on a classical supercomputer.

Based on the growth trend of the different technologies, it can be seen that I/O performance has fallen behind other technologies and the criticality of efficient usage of I/O resources can be realized. As mentioned earlier, checkpointing is a critical process that drives the need to improve the I/O bandwidth with frequent and at times lengthy accesses to the disk. Hence, optimizing the rate of checkpointing will optimize the usage of I/O resources. An alternative would be improving the checkpointing process itself with new techniques to consume fewer resources.

Several strategies are used as alternatives to traditional disk-based checkpointing. Many researchers are working on diskless checkpointing (checkpoints are encoded and stored in a distributed system instead of a central stable storage), for example [12], and suggest this strategy as an alternative to conventional disk-based checkpointing to reduce the I/O traffic due to checkpointing. The National Nuclear Security Administration (NNSA) issued a news release about the first full-system three-dimensional simulations of a nuclear weapon explosion (Crestone project), a significant achievement for Los Alamos National Laboratory and Science Applications International Corporation [10]. The simulation is essentially a 24-hour, seven-day-a-week job for more than 7 months. For a highly important seven-month computation such as Crestone, the notion of checkpointing and a rolling computation go hand-in-hand. The small success stories of using diskless checkpointing fail in such cases of large applications. True disk-based incarnations are required over heavy I/O phases of the code. Moreover, diskless checkpointing is often application-dependent and not generic.

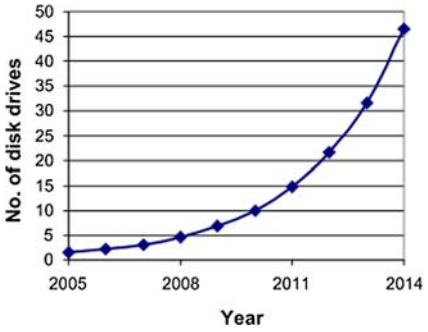


Fig. 5 Number of disks required in future systems to maintain present performance levels

Other checkpointing alternatives include incremental checkpointing (i.e., checkpointing only the information that has changed since previous checkpoint) and distributed checkpointing (i.e., individual nodes in the system checkpoint asynchronously thereby reducing the simultaneous load on the network). Although these emergent schemes may have their advantages, more maturity is required for their use in large-scale, mission-critical applications. In the current scenario, checkpoints are mostly synchronous with all the nodes writing checkpoints at the same time. Additionally, the checkpoints ideally involve the storage of the full system memory (at least 70% to 80% of the full memory in general) [10]. This scenario is quite common when schedulers such as Condor [16] are used or when applications checkpoint using libraries such as Libckpt [11]. The frequency of checkpointing can be decreased for productive I/O to surpass defensive I/O, but not without the risk of losing more computation due to system failures.

4 Optimizing checkpointing frequency

In this paper, we model the overhead in a system due to checkpointing with respect to the MTBF, memory capacity and I/O bandwidth of the system. In so doing, we identify the optimum frequency of checkpointing to be used on systems with given specifications. Optimal checkpointing helps to make efficient use of the available I/O and gain the maximum performance out of the system without compromising on its fault tolerance aspects. There have been similar efforts earlier to model and identify optimum checkpointing frequency for distributed systems [8, 18–20]. However, these efforts have not been simulatively or experimentally verified, and the approaches as yet are too theoretical to be practically implemented on systems.

It should be mentioned that optimizing the frequency of checkpointing is just one method of reducing the impact of defensive I/O on I/O bandwidth requirements. Other methods might include improvement of the storage system (high-performance storage system, high-performance interconnects, etc.), novel methods and algorithms for checkpointing, etc. There are claims that we can hide the impact of defensive I/O and work around this problem rather than tackling it. However, there are no recorded proofs to substantiate such claims.

4.1 Analytical modeling of optimum checkpointing frequency

Distribution of the total running time t of a job is determined by several parameters including:

- Failure rate, λ; $\lambda = \frac{1}{T_{MTBF}}$, where T_{MTBF} is the MTBF of the system.
- Execution time without failure or checkpointing, T_x.
- Execution time between checkpoints (checkpoint interval), T_i.
- Time to perform a checkpoint (checkpointing time), T_c.
- Operating system startup time, T_o.
- Time to perform a restart (recovery time) T_r.

Figures 6 and 7 show the various time parameters involved in the execution of a job without and with checkpointing, respectively. Without checkpointing, the system has to be failure free for a duration of T_x for the computation to complete successfully. When a failure occurs, computation is restarted from its initial state after system recovery. With checkpointing, the system state is checkpointed periodically. When the system is recovered from a failure, computation is resumed from the latest stable checkpointed state on system recovery. The execution process with checkpointing and failures can be modeled as a Markov process as shown in Fig. 8, where nodes $0, 1, 2, \ldots, n$ represent stable checkpointed states and $0', 1', 2', \ldots, n$ represent failed states. Let $t_1, t_2, t_3, \ldots, t_n$ be the random variables for the time spent in each cycle between two checkpointed states. These random variables are represented by τ in general.

The delays associated with each event in Fig. 8 are as follows: $a : T, b : \tau | \tau \leq T, c : T + T'$, and $d : \tau | \tau \leq T + T'$, where $T = T_i + T_c$ and $T' = T_o + T_r$.

The probabilities associated with each event in Fig. 8 are as follows: $p = e^{-\lambda T}$, $p' = e^{-\lambda(T+T')}$, $q = \text{prob}(\tau \leq T) = 1 - p$, $q' = \text{prob}(\tau \leq T + T') = 1 - p'$.

It should be noted that the total running time is a sum of individual random variables representing individual checkpointing cycles. However, the random variables

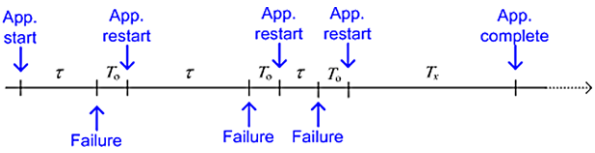


Fig. 6 Job execution without checkpointing

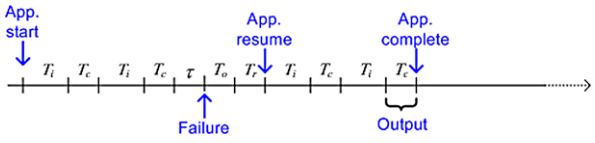


Fig. 7 Job execution with checkpointing

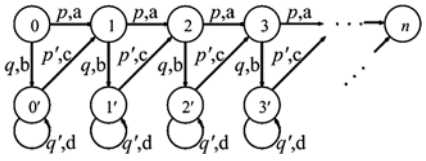


Fig. 8 Execution modeled as a Markov process

are similar, and hence the mean total running time ($\bar{t}$) can be given as the sum of the mean running time of each cycle.

$$\bar{t} = E(t) = E(t_1) + E(t_2) + E(t_3) + \cdots + E(t_n) = n \times E(t_1) \tag{4.1}$$

The mean running time of each cycle can be found by multiplying the probabilities associated with each path in the Markov chain and the corresponding time delay. There are several paths in the Markov chain that the process can actually take. The process can move from state 0 to state 1 with probability p and the delay associated with that transition is a. When there are failures in the system, the process does not directly move from state 0 to state 1. Instead, the process moves to state $0'$ with probability q and loops back in the same state with probability q'. The delays associated with these state transitions represented by b and d, respectively. b and d are exponential random variables with upper limit of T and $T + T'$, respectively. With probability p', the system moves from the failed state to the stable state 1 and delay associated with the transition represented by c is equal to $T + T'$, the upper limit of d.

$$\begin{aligned} E(t_1) &= p \times a + q \times p' \times (b + c) + q \times q' \times p' \times (b + c + d) \\ &\quad + q \times q'^2 \times p' \times (b + c + 2d) + \cdots \\ &= p \times a + q\left[(b + c) + \frac{q' \times d}{1 - q'}\right] \end{aligned} \tag{4.2}$$

where $a = T, b = e^{-\lambda T}(-T - \frac{1}{\lambda}) + \frac{1}{\lambda}, c = T + T'$, and $d = e^{-\lambda(T+T')}(-(T + T') - \frac{1}{\lambda}) + \frac{1}{\lambda}$.

Substituting the corresponding time delays into (4.2), we get (4.3).

$$\begin{aligned} E(t_1) = T e^{-\lambda T} + (1 - e^{-\lambda T})\Bigg[-e^{-\lambda T}\left(T + \frac{1}{\lambda}\right) + e^{-\lambda(T+T')}(T + T') \\ + \frac{1}{\lambda}\left(\frac{(1 - e^{-\lambda(T+T')})^2 + e^{-\lambda(T+T')}}{e^{-\lambda(T+T')}}\right)\Bigg] \end{aligned} \tag{4.3}$$

As can be seen from (4.1) and (4.3), the expression for the mean total running time is complicated. To avoid further complexity, we follow a different method as follows. The Laplace transform of a function $f(x)$ is given by $\phi_x(s) = \int f(x)e^{-sx}dx$. We can find the Laplace transform of the probability density function (pdf) of the total running time. The negative of the derivative of the Laplace transform at $s = 0$ gives the mean total running time.

$$\phi_t'(s) = \frac{d}{ds}\left(\int f(t)e^{-st}dt\right) = \int (-t) f(t)e^{-st}dt \tag{4.4}$$

$$\phi_t'(0) = \int (-t) f(t)dt = -E[t] \Rightarrow \bar{t} = -\phi_t'(0) \tag{4.5}$$

Laplace transform can be calculated by finding the Laplace transforms of individual transitions in the Markov process and then combining them together. For example,

the Laplace transform on the pdf of the time required for the transition from state 0 to state 1 without a failure is given by e^{-sT} and the transition happens with probability p. The transform on the pdf of the time required for transition from state 0 to state $0'$ that happens with probability q is given by

$$\varphi(s,T) = \frac{1}{1-e^{-\lambda T}} \int_0^T \lambda e^{-\lambda t} e^{-st} dt = \frac{\lambda}{\lambda + s} \left(\frac{1 - e^{-(\lambda+s)T}}{1 - e^{-\lambda T}} \right)$$

The Laplace transform on the pdf of the time required for the transition from state $0'$ to state 1 that happens with probability p' is given by $e^{-s(T+T')}$. If there is looping in state $0'$ due to repeated failures, the transform on the pdf of the time spent returning to state $0'$ (transition probability q') is given by

$$\varphi(s,T+T') = \frac{1}{1-e^{-\lambda(T+T')}} \int_0^{T+T'} \lambda e^{-\lambda t} e^{-st} dt = \frac{\lambda}{\lambda + s} \left(\frac{1 - e^{-(\lambda+s)(T+T')}}{1 - e^{-\lambda(T+T')}} \right)$$

The Laplace transform for the process with failures (state transition from 0 to 1 via $0'$) is found by doing a weighted sum of the Laplace transforms on the pdfs of the individual random variables. The transform is given by

$$q \times \varphi(s,T) \left[\sum_{i=0}^{\infty} p' e^{-s(T+T')} [q' \times \varphi(s,T)]^i \right]$$

where i denotes the number of number of loops in state $0'$.

Hence, the Laplace transform of the pdf for one cycle of the Markov process is be given by

$$\phi_{t_1}(s) = p e^{-sT} + q \times \varphi(s,T) \left[\sum_{i=0}^{\infty} p' e^{-s(T+T')} [q' \times \varphi(s,T)]^i \right]$$

We get (4.6) from the above and differentiating (4.6) with respect to s and substituting $s = 0$, we find mean total running time given by (4.7). We verified the validity of the expressions for the mean total running time given by (4.4) and (4.7) by running a Monte Carlo simulation with 10,000 iterations and cross-checking the results. We found that the time given by the expressions in the equations closely matched the simulation results. We will be using the expression in (4.7) for further development for simplicity reasons both in terms of representation and computation.

$$\begin{gathered}
\phi_t(s) = \phi_{t_1}(s) \times \phi_{t_2}(s) \times \phi_{t_3}(s) \times \cdots \times \phi_{t_n}(s) = \left(\phi_{t_1}(s)\right)^n \\
\phi_t(s) = \left(e^{-(s+\lambda)T} + \frac{\lambda(1 - e^{-(s+\lambda)T}) \times e^{-(s+\lambda)(T+T')}}{s + \lambda e^{-(s+\lambda)(T+T')}} \right)^n \\
\phi_t'(0) = -\frac{n}{\lambda} \left(\frac{1 - e^{-\lambda T}}{e^{-\lambda(T+T')}} \right) [\phi_t(0)]^{n-1}
\end{gathered} \tag{4.6}$$

$$\bar{t} = -\phi_t'(0) = \frac{n}{\lambda} \left(\frac{1 - e^{-\lambda T}}{e^{-\lambda(T+T')}} \right) = \frac{n}{\lambda} \left(e^{\lambda(T_i + T_c + T_o + T_r)} - e^{\lambda(T_o + T_r)} \right) \tag{4.7}$$

We find the optimum checkpointing interval T_{i_opt} that gives the minimum total running time by differentiating the mean total running time with respect to T_i and equating to zero as shown in (4.8). We set n to be equal to the ratio of T_x and T_i

$$\frac{\partial \bar{t}}{\partial T_i} = \frac{\partial}{\partial T_i}\left(\frac{T_x(e^{\lambda(T_i+T_c+T_o+T_r)} - e^{\lambda(T_o+T_r)})}{\lambda T_i}\right) = 0 \tag{4.8}$$

Solving (4.8), we get the following equation on optimum checkpointing interval:

$$T_{i\_opt} = \frac{1 - e^{-\lambda(T_{i\_opt}+T_c)}}{\lambda} \tag{4.9}$$

Equation (4.9) can be represented in the form $\alpha = 1 - \beta e^{-\alpha}$ where $\alpha = \lambda T_{i\_opt}$ and $\beta = e^{-\lambda T_c}$. From this form of representation, it can be seen that (4.9) is a transcendental equation and it is impossible to find a solution for α except by defining a new function. There is no analytic solution to this equation to obtain a closed form solution. However, it can be seen on expansion of $e^{-\alpha}$ that α is bound by the limit $\alpha < 1$, which implies that T_{i_opt} is bound by the limit $T_{i\_opt} < \lambda^{-1}$, i.e, $T_{i\_opt} <$ MTBF of the system. Also, since $e^{-\alpha} \leq 1$ we have $1 - \beta \leq \alpha$. Thus, we have a lower bound on optimum checkpointing interval as $T_{i\_opt} \geq \frac{1-e^{-\lambda T_c}}{\lambda}$. Hence, we can use numerical methods to solve the equation for optimum checkpointing interval.

4.2 Impact of checkpointing overhead on I/O bandwidth

We modeled the optimum checkpointing frequency that provides the minimum overall running time for applications. With the model developed, we study the impact of checkpointing overhead on the sustainable I/O bandwidth of systems in this section. The representative performance numbers used in this section are typical in HPC and supercomputing systems.

In our view, given a system with a specified memory capacity and MTBF, it would be useful to study the I/O bandwidth requirements of the system with respect to the overhead that is imposed by the checkpointing done in the system and the subsequent performance loss in terms of the total execution time of the application. Equation (4.10) obtains this performance loss as a function of λ, T_i and T_c.

$$F = \frac{T_x(e^{\lambda(T_{i\_opt}+T_c+T_o+T_r)} - e^{\lambda(T_o+T_r)})}{T_x \times \lambda T_{i\_opt}} = \frac{(e^{\lambda(T_{i\_opt}+T_c+T_o+T_r)} - e^{\lambda(T_o+T_r)})}{1 - e^{-\lambda(T_{i\_opt}+T_c)}} \tag{4.10}$$

Let F denote the factor of increase in the total running time of the application due to checkpointing overhead and failures while running with optimum checkpoint interval. F is given by the ratio of $\bar{t}$ to T_x as given by (4.10).

In (4.10), T_o can be considered negligible compared to the other times. Also, T_r can be considered equal to T_c as both represent the time to move the same amount of data through the same I/O channel. T_c can be given by the ratio of memory capacity to I/O bandwidth. Given a value of F, we can solve for T_{i_opt} by solving a quadratic equation on $e^{\lambda T_i}$.

$$T_{i\_opt} = \frac{1}{\lambda} \times \ln\left(\frac{K \pm \sqrt{K^2 - 4M}}{2}\right) \tag{4.11}$$

where $K = e^{-\lambda T_c} + Fe^{-2\lambda T_c}$ and $M = Fe^{-3\lambda T_c}$.

Figures 9a and 9b show T_{i_opt}, and hence the impact of checkpointing on the overall system execution time in systems with MTBF of 8 hours and 24 hours, respectively, for varying system memory capacities and I/O bandwidth. The MTBF values used in the figures are typical in recent high-performance systems [2]. The value of F is fixed at 1.2 in the figures. We pick 1.2 because F represents the impact of checkpointing on overall execution time and lower values of F (as close to 1 as possible) are certainly desirable. It can be seen from the figures that for systems with low I/O bandwidth, optimum checkpointing is not even possible, i.e., the allowable overhead (represented by F) is not achievable. As I/O bandwidth of the system is increased, the optimum checkpointing interval also increases. Since the total execution time has been fixed (1.2 times the actual execution time), an increase in checkpoint interval means a decrease in the number of checkpoints(n) during the course of the execution. Fewer checkpoints imply less overhead on the system's I/O.

It can be seen from Fig. 9 that sustainable I/O bandwidth is key to obtain optimum overall execution time. As memory capacity of the system increases, so does the requirement of higher I/O bandwidth. For systems with higher memory capacities, it is impossible to find a solution for optimum checkpointing interval to obtain the optimum overall execution time. For example, in a system with a MTBF of 8 hours and memory capacity of 75 TB, there is no solution for the optimum checkpointing interval until the I/O bandwidth is increased to 29 GB/s. The impact is less in systems with higher MTBF values. For a system with similar memory capacity but a MTBF of 24 hours, there exists a solution starting with an I/O bandwidth of 10 GB/s. However, an important factor to note is that although solutions exist for the optimum checkpointing interval that gives the minimum total running time, the system might be bogged down by too many checkpoints if the checkpoint interval is low. For example, in a system with a memory capacity of 75 TB and MTBF of 24 hours, the optimum checkpointing interval is around 10 minutes. Performing large checkpoints at such a high frequency will certainly cause a great load on the system and is not desirable.

In certain scenarios or systems, the utility of the system within each cycle can be critical. In a given system, let R_1 denote the utility in a cycle, i.e., the ratio of time spent doing useful calculations to the overall time spent in a cycle.

$$R_1 = \frac{T_i}{T_i + T_c} \quad \text{and when } T_i = T_{i\_opt},$$

$$R_1 = \frac{1 - e^{-\lambda(T_{i-opt}+T_c)}}{1 - e^{-\lambda(T_{i-opt}+T_c)} + \lambda T_c} \tag{4.12}$$

Equation (4.12) gives the utility in each cycle when checkpointing is performed at the optimum checkpointing interval. The checkpointing time T_c can be again given by the ratio of C_{MEM} (memory capacity) and IO_{BW} (sustainable I/O bandwidth). Figures 10a and 10b give the utility in a cycle for several memory capacities and I/O bandwidths for systems with MTBF of 8 hours and 24 hours, respectively. The value of F is fixed at 1.2. The trend of I/O bandwidth requirement is similar to that in Fig. 9.

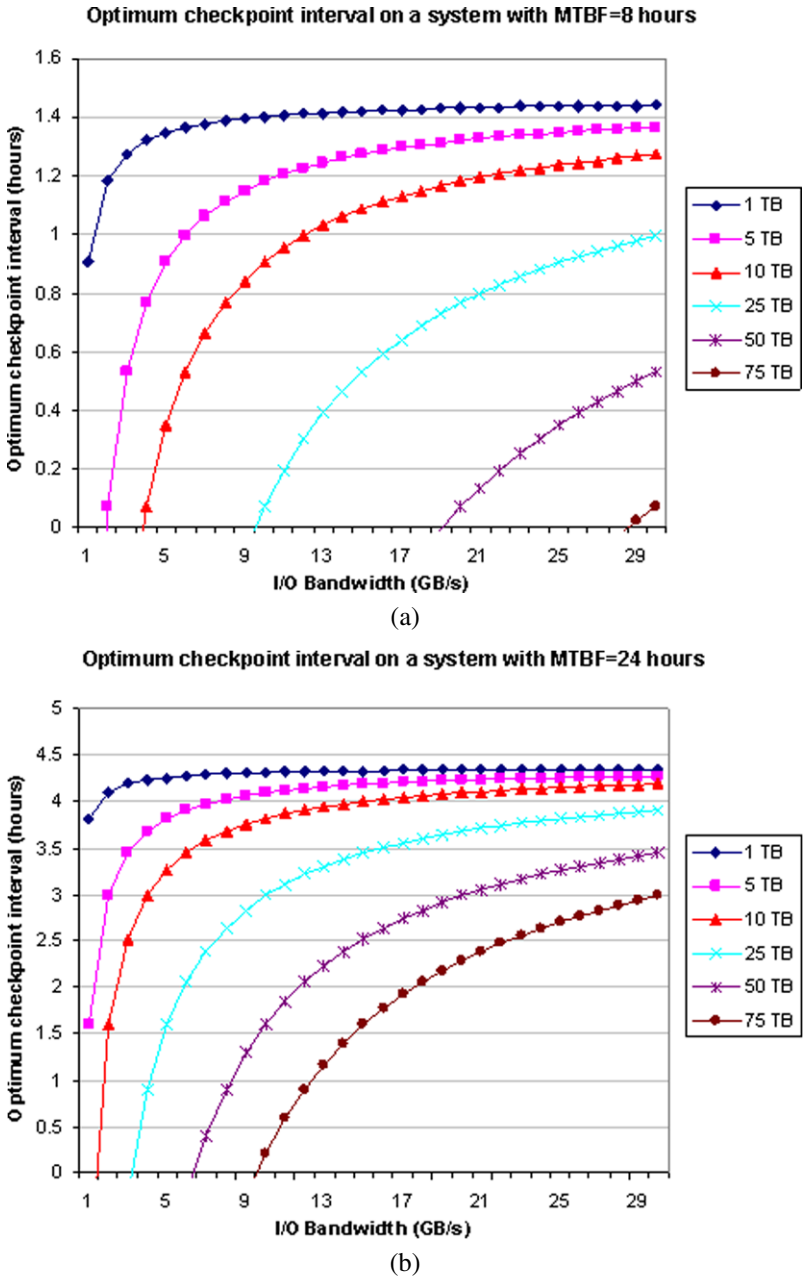
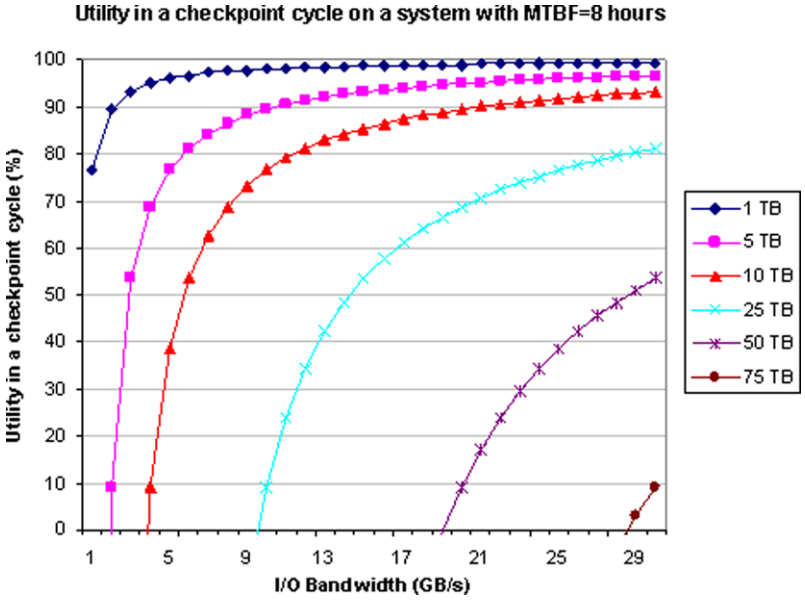
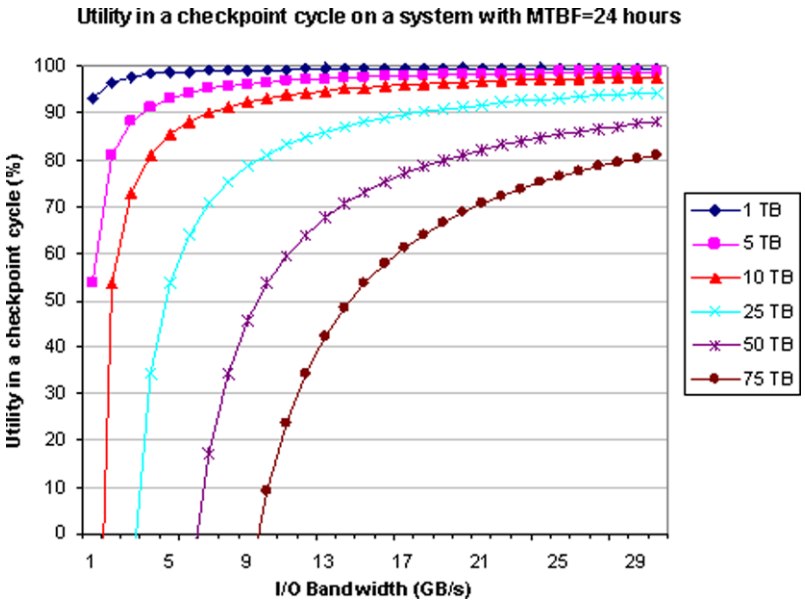


Fig. 9 Impact of I/O bandwidth on the system execution time and checkpointing overhead

It can be seen from the figures, in most of the cases, the utility in a checkpoint cycle is much less than 90% which implies that most of the time within a cycle is spent checkpointing and not doing useful calculations. Although not shown in the



(a)



(b)

Fig. 10 Impact of increasing memory capacity in systems on sustainable I/O bandwidth requirement

figure, it was found that for a system with MTBF of 8 hours and memory capacity of 75 TB, an I/O bandwidth of about 150 GB/s is required to utilize at least 90% of a checkpoint cycle on useful calculations. Such high values of I/O bandwidth are much

higher than what is available for present systems. The fact that the system cannot even provide useful computations in many cases shows the gravity of the situation.

We see from Fig. 10 that the utility within a cycle almost saturates beyond a certain I/O bandwidth. The I/O bandwidth at which the percentage utility begins to flatten is what is desirable for the system both present and future. For example, I/O bandwidth of around 10 GB/s would be desirable for a system with a memory capacity of 1 TB and MTBF of 8 hours. For a similar system with MTBF of 24 hours a lesser I/O bandwidth (around 5 GB/s) would suffice. But as the memory capacity of the system increases, the I/O bandwidth desirable is dramatically higher.

5 Simulative verification of checkpointing model

Simulation is a useful tool to observe the dynamic behavior of a system as its configuration and components change. As preliminary verification, Monte Carlo simulations were performed on the analytical model. The simulation and analytical results matched closely verifying the correctness of the model mathematically. However, for more accurate verification of the model, we develop simulation models to mimic typical HPC environments using Mission-Level Designer (MLD), a discrete-event simulation platform developed by MLDesign Technologies Inc. Section 5.1 presents the system model used to gather results, and Sects. 5.2–5.6 provide details about each major component model.

5.1 System model

Figure 11 displays the system model that we employed to conduct the simulation experiments. The system consists of four key component models, multiple nodes, network, central memory, and fault generator. Each component has associated parameters that are user-definable in order to model different system settings. The components and their parameters are discussed in detail in the subsequent sections.

5.2 Node model

A node is defined as a device that processes and checkpoints data, and is prone to failures. Figure 12 shows the MLDesigner node model. The behavior of the node is

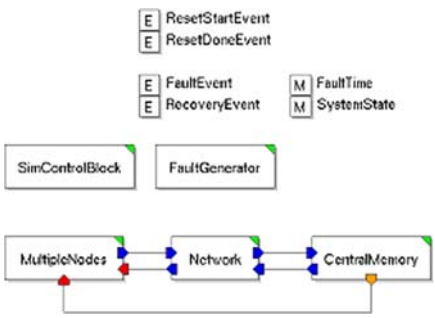


Fig. 11 System model

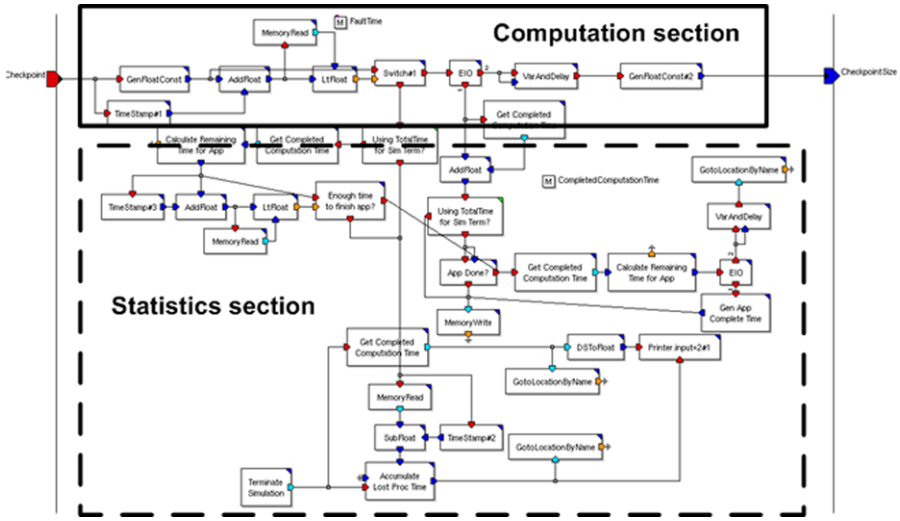


Fig. 12 Node model

modeled such that it checkpoints its entire main memory at a specified time period. The time spent between checkpoints represents useful computation, communication, and productive I/O time used to complete a specific task (see Computation section in Fig. 12). After a checkpoint has been successfully completed, the nodes can continue processing data.

The node model was designed in order to provide an abstract representation of a clustered processing element that runs a parallel job along with the other nodes in the system. If a single node in the system fails, all the nodes must recover from the last successful checkpoint to ensure data integrity. The statistics gathered at the node level (see Statistics section in Fig. 12) include completed computation time and lost computation time due to a failure. The user can define numerous parameters for the node model including checkpoint interval, main memory size, and application completion time.

5.3 Multiple nodes model

The multiple nodes model, illustrated in Fig. 13, uses a capability of the MLDesigner tool, dynamic instantiation that allows a single block to represent multiple instances of a model. The node model described in Sect. 5.2 is dynamically instantiated in the multiple nodes model. The technique is used to ease the design and configuration procedure used to model large homogenous systems. The main function of the multiple nodes model is to ensure global synchronous checkpoints and to collect statistics. The statistics gathered include completed checkpoint time and total checkpoint time lost. That is, it records the amount of time taken to successfully complete a checkpoint and also the amount of time lost when a failure occurs during a checkpoint.

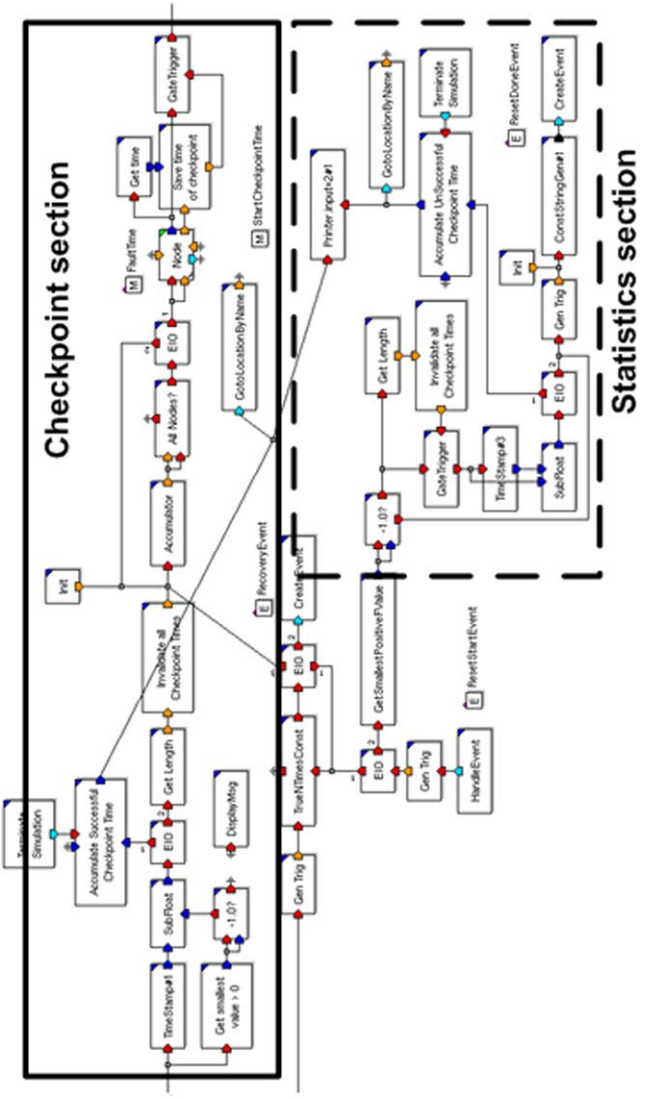


Fig. 13 Multiple nodes model

5.4 Network model

The network model is a coarse-grained representation of a generic switch- or bus-based network. The model uses user-defined latency and effective bandwidth parameters to calculate network delay based on the size of the transfer. Figure 14 illustrates the network model.

5.5 Central memory model

The central memory model is another coarse-grained model that is used as a mechanism to represent the storing and restoring of checkpointed data. The model accepts each checkpoint request and sends a reply when the checkpoint is completed. When a node fails, each node is sent its last successful checkpoint after some user-definable recovery time which represents the time needed for the system to respond to the failure. No actual data is stored in the central memory since the simulated system does not actually process real data. Also, the central memory is assumed to be large enough to hold all checkpoint data, therefore overflow is not considered. Figure 15 illustrates the central memory model.

5.6 Fault generator model

When a failure occurs in a component, the system must recover using a predefined procedure. This procedure starts by halting each working node followed by the transmission of the last checkpoint by the central memory model. When the failed node recovers and all nodes receive their last checkpoint, the system begins processing once again.

The fault generator model shown in Fig. 16 controls this process by creating a "failure event" at some random time based on an exponential distribution with a user-definable time (i.e., MTBF) and orchestrating the recovery process of the components in the system. For example, when a failure occurs, the fault generator will pass a notice to the node models to let them know that they must recover. It also tracks the status of each node regarding their recovery status (e.g., recovered or recovering). Figure 16 shows the fault generator model.

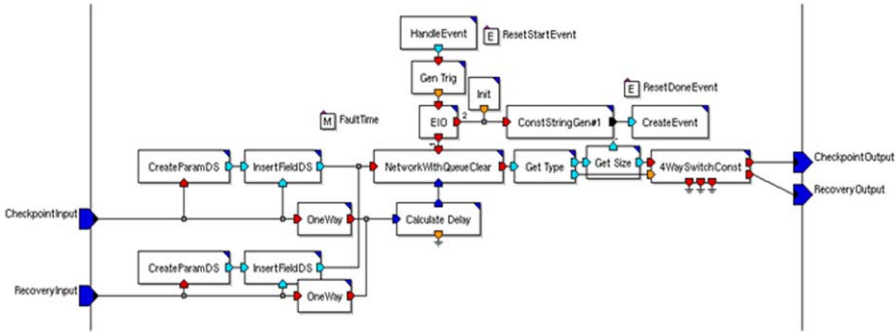


Fig. 14 Network model

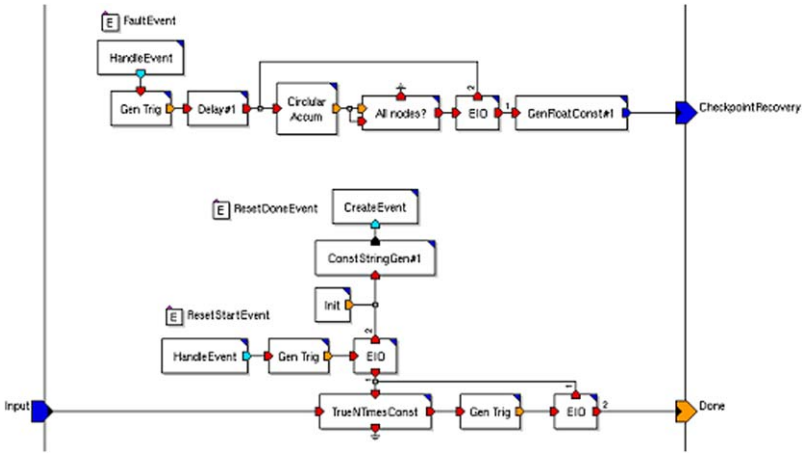


Fig. 15 Central memory model

5.7 Simulation process

Total execution time of the application, MTBF of the system, sustainable I/O bandwidth, amount of memory to be checkpointed, and frequency of checkpointing are input parameters to the simulation model and are variable. Faults are generated in the system based on the MTBF value input. After every checkpointing period, data (equal to the size of memory specified) is checkpointed to the central memory. The time to checkpoint is dependent on the amount of checkpoint data and the I/O bandwidth values. While recovering from a failure, the nodes in the system collect the data from the central memory and the transfer time is again dependent on the data size and the I/O bandwidth value. If a failure occurs in the system during the data recovery process, the transfer is reinitiated and the process is repeated until successful.

5.8 Verification of analytical model

In order to verify the correctness of our analytical model, we simulated the checkpointing process in several systems with memory ranging from 5 TB to 75 TB and I/O bandwidth ranging from 5 GB/s to 50 GB/s, running applications with execution times ranging from 15 days to 6 months (180 days). For each combination of values for the above parameters, the system was simulated for MTBF values of 8 hours, 16 hours and 24 hours. The numbers listed above are typical of high-performance computers found in several research labs and production centers. In this section, we pick a few systems and show the simulation results to verify the analytical model for optimum checkpointing frequency.

Figure 17 shows the results from simulations of the system running applications with fault-free execution time of 15 days ($15 \times 24 = 360$ hours). For a given value of checkpointing interval (T_i), the y-axes in Fig. 17 give the application completion time (i.e., total time for the application to complete with faults occurring in the system). Hence, the optimum checkpoint interval is the value for which the application

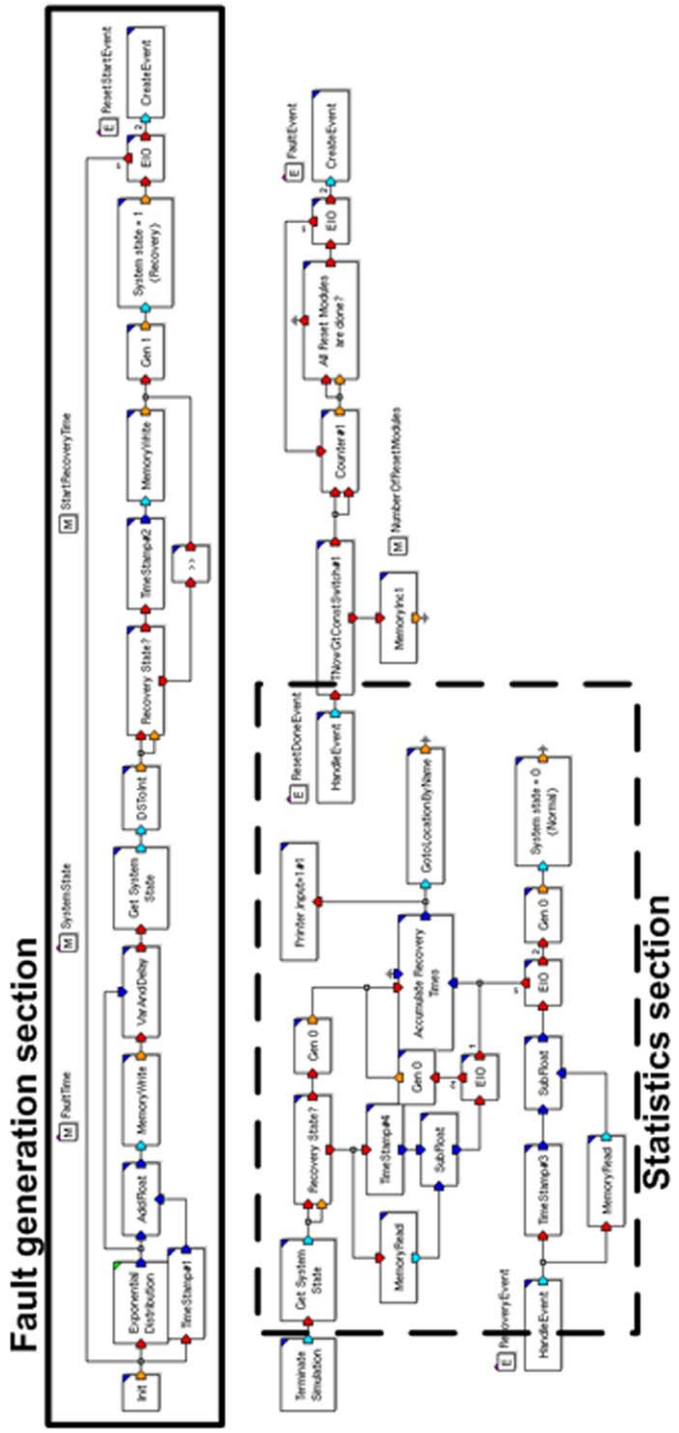


Fig. 16 Fault generator model

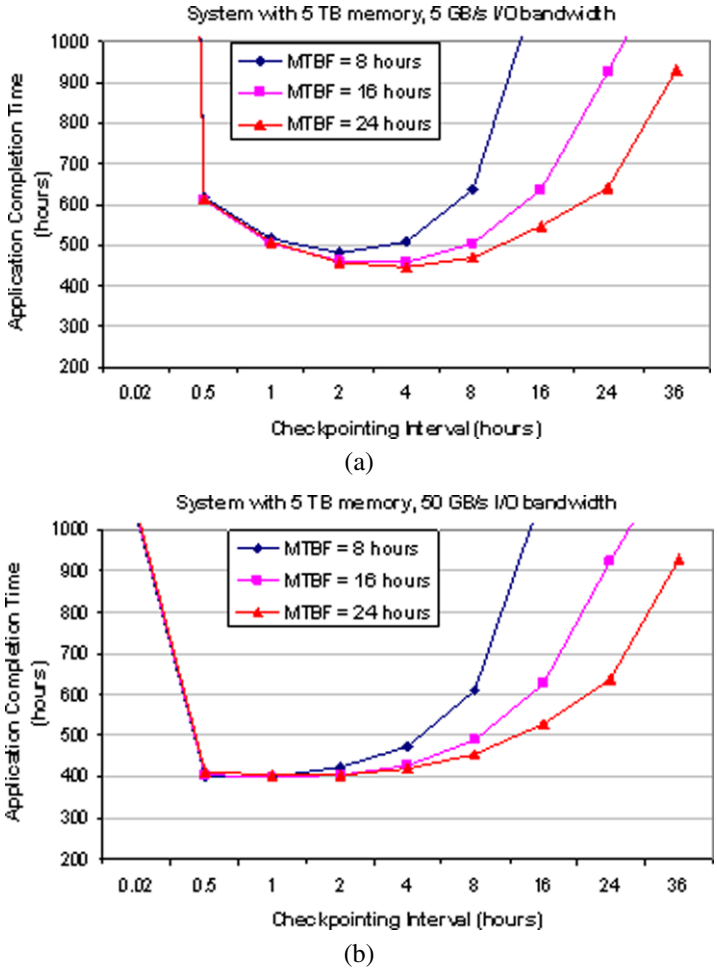


Fig. 17 Optimum checkpointing interval based on simulations of systems with 5 TB memory

completion time is the least with the given system resources. Based on the completion time given by Fig. 17a, for a system with 5 TB memory and 5 GB/s sustained I/O bandwidth, the optimum checkpoint intervals are around 2 hours, 3 hours and 4 hours, respectively, when the MTBF of the system is 8 hours, 16 hours, and 24 hours. Similarly, for a system with 5 TB memory and 50 GB/s sustained I/O bandwidth, the optimum checkpoint intervals as shown by Fig. 17b are 0.5 hours, 0.75 hours and 1 hour respectively when the MTBF values are 8 hours, 16 hours, and 24 hours.

In order to verify the correctness of the analytical model, (4.9) needs to be solved to find T_{i_opt} given the same system parameters used for simulation in Fig. 17. We use the fixed point iteration numerical method [6] to solve (4.9). Such strategies are quite common in solving equations as not all problems yield to direct analytical solutions [15]. Fixed point iteration is a method for finding roots of an equation $f(x) = 0$ if it can be cast into the form $x = g(x)$ as is (4.9). A fixed point of the function

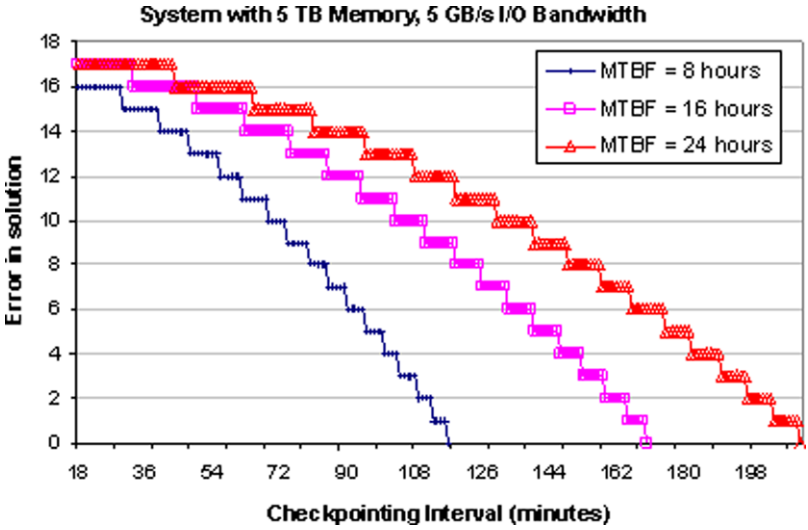


Fig. 18 Numerical method solution for the analytical model

$g(x)$ is the solution of the equation $x = g(x)$ and the solution is found using an iterative technique. A value is supplied for the unknown variable in the equation and the process is repeated until an answer is achieved.

In order to show the validity of using numerical method to solve our analytical model, we show the results from the fixed point iterative method that we used in Fig. 18. The system used for illustration in the figure is the one with 5 TB of memory and 5 GB/s of I/O bandwidth. For (4.9) to have a solution, i.e., for a given checkpoint interval to be the optimum, the left-hand side (LHS) is equal to the right-hand side (RHS) of the equation. For a given value of checkpointing interval (T_i), the y-axis in Fig. 18 referred to as the error in solution give the difference between the LHS and the RHS of (4.9). Hence, the optimum checkpoint interval is the one for which the error is zero. The checkpointing interval values used in our iterative technique are bound by the lower and upper limits discussed in Sect. 4. The optimum checkpoint intervals identified using the numerical method in Fig. 18 for the system are 2 hours, 3 hours, and 3.5 hours, respectively, when the MTBF is 8 hours, 16 hours, and 24 hours. The results of the analytical model are then cross compared with the solutions from the simulation model shown in Fig. 17. It can also be seen that the solutions are well below the upper bound (MTBF) derived in Sect. 4. Section 6 provides the results from the analytical model for a wide range of systems.

Solving our analytical model for a system with 5 TB of memory and 5 GB/s of I/O bandwidth, the optimum checkpointing intervals are around 2 hours, 3 hours, and 3.5 hours, respectively, when the MTBF of the system is 8 hours, 16 hours, and 24 hours. For a similar system with 5 TB of memory but 50 GB/s of I/O bandwidth, the optimum checkpointing intervals are around 0.5 hours, 0.75 hours, and 1 hour, respectively, when the MTBF of the system is 8 hours, 16 hours, and 24 hours.

Based on the simulation results for optimum checkpoint intervals in Fig. 17, it can be seen that the analytical results match that of simulation. According to both ana-

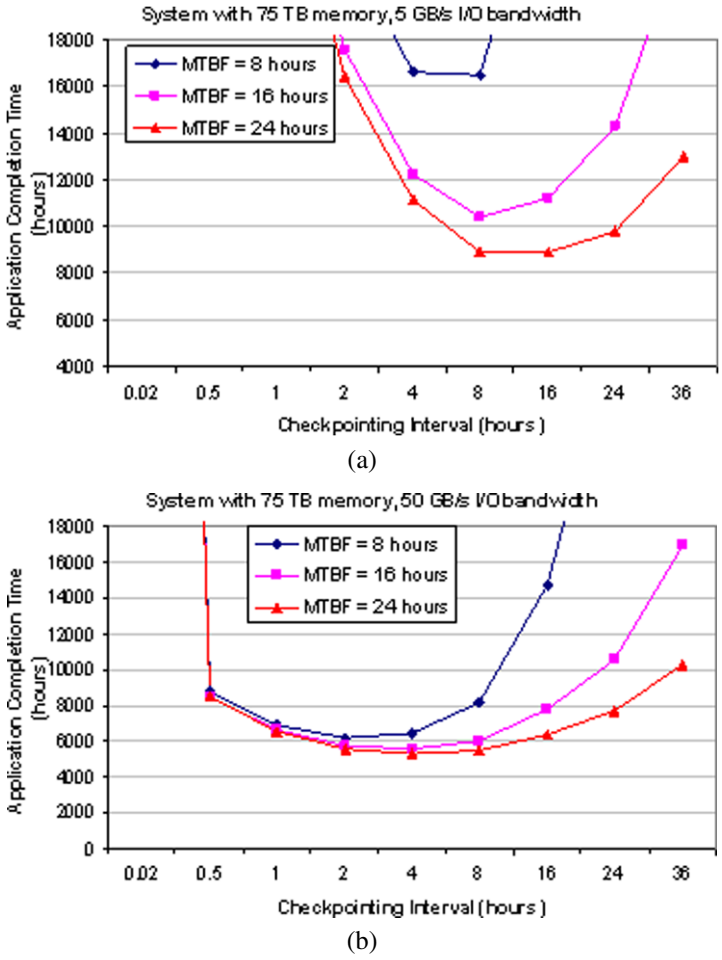


Fig. 19 Optimum checkpointing interval based on simulations of systems with 75 TB memory

lytical and simulation models, for a system with 5 TB of memory and 5 GB/s of I/O bandwidth, the optimum checkpointing intervals are around 2 hours, 3 hours, and 4 hours, respectively, when the MTBF of the system is 8 hours, 16 hours, and 24 hours. Likewise for a system with 5 TB of memory but 50 GB/s of I/O bandwidth, the optimum checkpointing intervals are around 0.5 hours, 1 hour, and 1 hour, respectively, when the MTBF of the system is 8 hours, 16 hours, and 24 hours. The analytical model is thus verified by the simulation model.

Equation (4.9) is independent of the total execution time of the application. In order to verify this behavior of the analytical model, we ran simulations for applications with larger execution times. Figure 19 shows the results from simulations of the system running applications with fault-free execution time of 180 days ($180 \times 24 = 4{,}320$ hours). We also wanted to verify that the analytical model holds true for systems with parameters different from those used. Hence, the system shown

in Fig. 19 has 75 TB of memory which is close to the high end of presently existing supercomputers.

Simulation results in Fig. 19a show that for a system with 75 TB of memory and 5 GB/s of I/O bandwidth, the optimum checkpointing intervals are around 6 hours, 9 hours, and 12 hours, respectively, when the MTBF of the system is 8 hours, 16 hours, and 24 hours. For the same system, the analytical model also provides the same optimum checkpointing intervals as can be seen from figures in Sect. 6.

Simulation results in Fig. 19b show that for a system with 75 TB of memory but 50 GB/s of I/O bandwidth, the optimum checkpointing intervals are around 2 hours, 3 hours, and 4 hours, respectively, when the MTBF of the system is 8 hours, 16 hours, and 24 hours. Analytical results for the same system matches the simulation results. Thus, it can be seen from Figs. 17–19 that the analytical results match closely with the simulation results, further verifying the correctness of the analytical model.

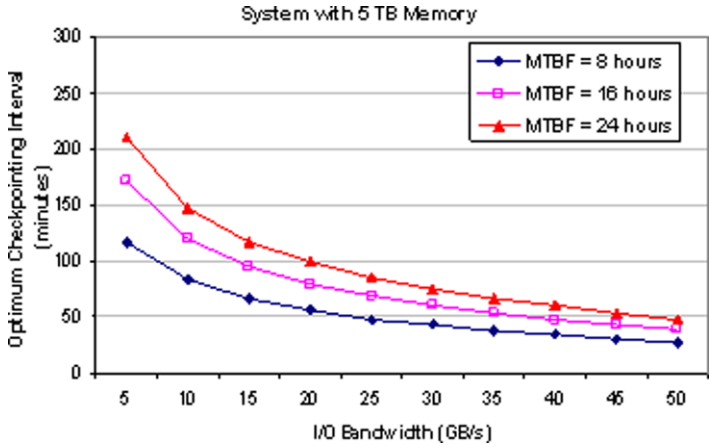
6 Optimal checkpointing frequencies in typical systems

The results used in Sect. 5 were primarily used to verify the analytical model. In this section, we show results derived from the analytical model for a wide range of systems with parameter values representing large supercomputers traditionally developed for HPC. The results provide insight about the checkpoint intervals that should be used in typical scenarios. In order to analyze the applicability of our model to a broader spectrum of systems, we also studied small systems intended for high-performance embedded computing (HPEC).

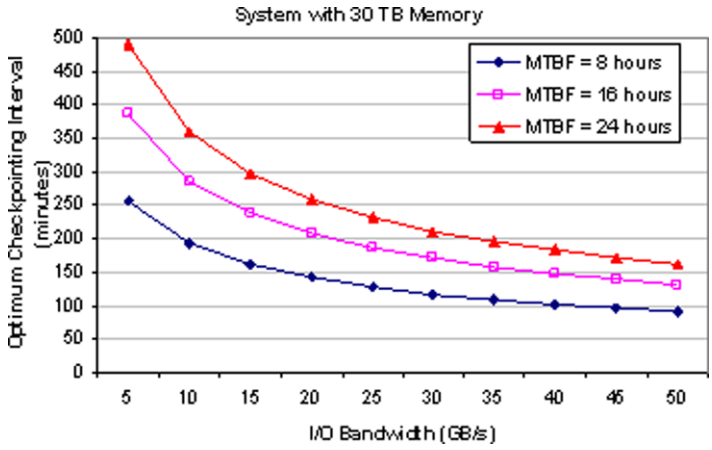
6.1 HPC systems

We show the optimum checkpointing frequency results for systems with parameter values in the range of traditional HPC systems in Fig. 20. The optimum checkpointing frequencies are calculated using the same method discussed in the previous section (i.e., fixed point iteration method). Figure 20a gives the results for a system at the lower end with 5 TB memory while Fig. 20b gives the results for a system at the other extreme with 75 TB memory. These higher-end resources are similar to what an application such as the Crestone project discussed earlier would solicit. The results for a midrange system with 30 TB memory are shown in Fig. 20c. The optimum checkpointing interval for typical HPC systems is in the order of hours.

Increase in MTBF implies lesser chances of faults and hence it is not required to checkpoint often. It can be seen from Fig. 20 that as the MTBF of the system increases the optimum checkpointing interval also increases. However, the difference is much pronounced only in systems with less I/O bandwidth. As the I/O bandwidth of the system is increased, the difference in optimum checkpointing intervals for varying MTBF decreases. It can also be observed from the figure that for a system with a given memory capacity and MTBF, the optimum checkpointing interval decreases as the I/O bandwidth of the system is increased. This trend implies that it is better to checkpoint at a higher frequency if the time to checkpoint lowers. This observation can also be ascertained by the fact that for the same I/O bandwidth the optimum



(a)



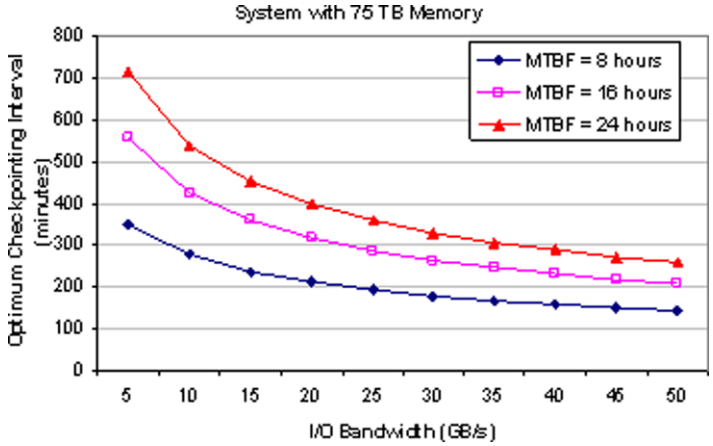
(b)

Fig. 20 Optimum checkpointing interval for HPC systems with varying system parameters

checkpointing interval increases as the memory capacity of the systems increases (implying that it is optimal to checkpoint at a lesser frequency if checkpointing time increases).

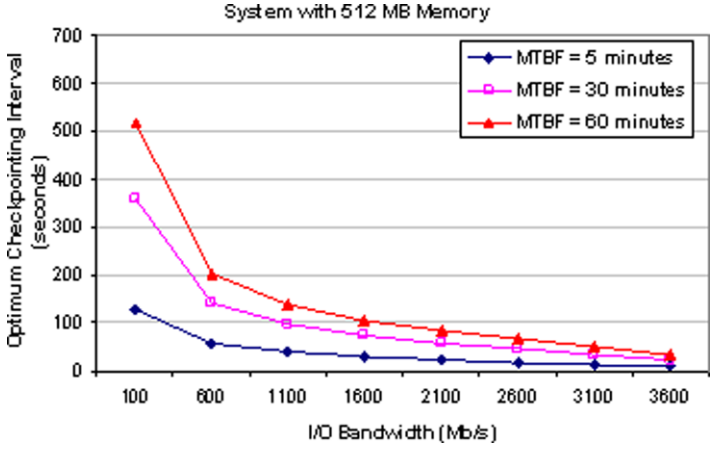
6.2 HPEC systems

Figure 21 shows the optimum checkpointing frequencies for systems with parameter values in the range of HPEC systems. Figure 21a gives the results for a system at lower end with 512 MB memory while Fig. 21b gives the results for a system at the other extreme with 8 GB memory. The results for a midrange system with 2 GB memory are shown in Fig. 21c. The optimum checkpointing interval for typical HPEC systems is in the order of minutes.



(c)

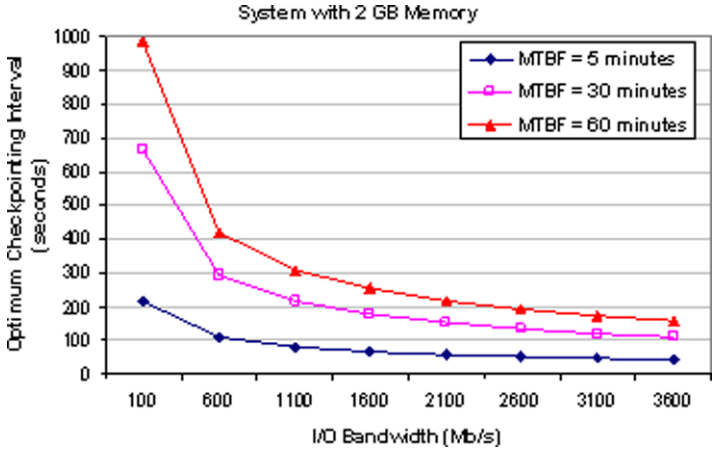
Fig. 20 (*Continued*)



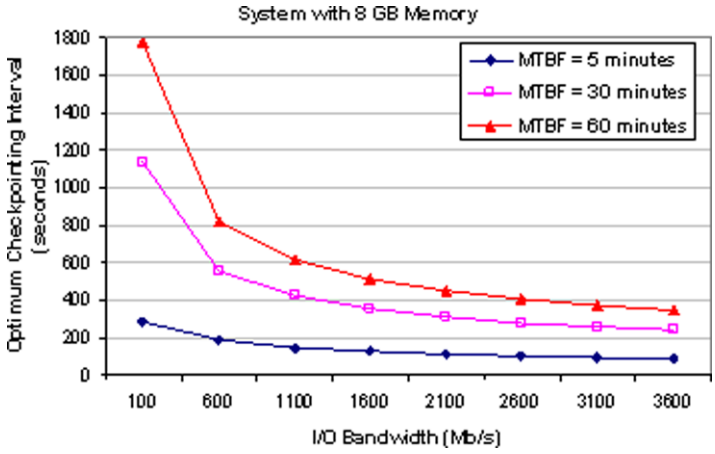
(a)

Fig. 21 Optimum checkpointing interval for HPEC systems with varying system parameters

It can be seen from Fig. 21 that the behavior of the HPEC systems is similar to the HPC systems shown in Fig. 20 although HPEC systems have much less resources and high failure rate. For a given system, the optimum checkpoint interval increases as the MTBF value is increased. Likewise for a system with a given MTBF and I/O bandwidth, the optimum checkpoint interval increases as the memory of the system increases. As I/O bandwidth of the system is increased with other system parameters kept constant the optimum checkpoint interval decreases.



(b)



(c)

Fig. 21 (*Continued*)

7 Conclusions

In this paper, the useful computation time of computing systems is improved by optimizing checkpointing-related defensive I/O. Presently, defensive I/O that is based on checkpointing is the primary driver of I/O bandwidth, rather than productive I/O that is based on processor performance. There are several research efforts to improve the process of checkpointing itself, but they are generally application-specific. To our knowledge, there have been no successful attempts to optimize the rate of checkpointing, which is our approach to reduce the fault-tolerance overhead. Such an approach is not application- or system-specific and is applicable to systems of any scale.

We developed a mathematical model to represent the checkpointing process in HPC systems and analytically modeled the optimum computation time within a checkpoint cycle in terms of the time spent on checkpointing, the MTBF of the sys-

tem, amount of memory checkpointed, and sustainable I/O bandwidth of the system. The model showed that the optimum checkpointing interval is independent of duration of application execution and is only dependent on system resources. In order to see the impact of checkpointing overheads and the overhead to recover from failures on the I/O bandwidth required in the systems, we analyzed the overall execution time of applications including these overhead tasks relative to the ideal execution time in a system without any checkpointing or failures. Results pertaining to the time spent on useful calculations between checkpoints suggest the need for a very high sustainable I/O bandwidth for future systems. A system with a projected MTBF of 1 day (24 hours) and a memory capacity of 75 TB, which may be typical for a system in the near future, would require a sustainable I/O bandwidth of about 53 GB/s to effectively use 90% of time on useful computations. For a similar system with a projected MTBF of 8 hours, the required I/O bandwidth would be 150 GB/s for the same performance.

In order to verify our analytical model, we developed discrete-event simulation models to simulate the checkpointing process. In the simulation models, we used performance numbers that represent systems ranging from small embedded cluster systems to large supercomputers. The simulation results matched closely with those from our analytical model. Finally, we also showed the optimum checkpointing interval for a wide range of HPC and HPEC systems. The results derived from the analytical model showed that irrespective of the duration of the application's fault-free execution, the optimum checkpointing interval is in the orders of hours for HPC systems, and in the order of minutes for HPEC systems. For a system with a given MTBF, the checkpointing time determined by the ratio of memory capacity and I/O bandwidth is the primary factor influencing the optimum checkpointing interval.

As future work, the model can be experimentally verified with real systems running scientific applications. When successfully verified, the model can be used to find the optimal checkpointing frequency for various HPC and HPEC systems based on their resource capabilities.

References

1. 73.4 GB 3.6MS/15000 (ULTRA 320 80PIN) 8192K 3.5″/HH, http://www.spartantech.com/product.asp?PID=ST373453LC&m1=pg (accessed: April 23, 2006)
2. ASCI purple statement of work, Lawrence Livermore National Laboratory, http://www.llnl.gov/asci/purple/Attachment_02_PurpleSOWV09.pdf (accessed: April 23, 2006)
3. Cheetah 15K.3-ST336753LC, http://www.seagate.com/cda/products/discsales/marketing/detail/0,1081,552,00.html (accessed: April 23, 2006)
4. Cramming more components onto integrated circuits. Electronics 37(8), April 19, 1965
5. Dongarra J, Luszczek P, Petitet A (2003) The LINPACK benchmark: past, present, and future. Concurr Comput Pract Experience 15:1–18
6. Fixed point iteration, http://pathfinder.scar.utoronto.ca/~dyer/csca57/book_P/node34.html (accessed July 3, 2006)
7. HITACHI eyes 1 TB desktop drives, http://www.pcworld.com/news/article/0,aid,120279,00.asp (accessed: April 23, 2006)
8. Kavanaugh GP, Sanders WH (1997) Performance analysis of two time-based coordinated checkpointing protocols. In: Pacific Rim international symposium on fault-tolerant systems, Taipei, Taiwan, December 15–16, 1997
9. LINPACK, http://www.netlib.org/linpack/ (accessed: April 23, 2006)
10. Los Alamos/Liv 3D simulations, Publication of Los Alamos National Laboratory, vol 3, No 6, April 4, 2002

11. Plank J, Beck M, Kingsley G, Li K (1995) Libckpt: transparent checkpointing under Unix. In: Usenix winter 1995 technical conference, New Orleans, LA, January, 1995
12. Plank JS, Kim Y, Dongarra J (1997) Fault tolerant matrix operations for networks of workstations using diskless checkpointing. J Parallel Distributed Comput 43(2):125–138
13. Schocht G, Troxel I, Farhangian K, Unger P, Zinn D, Mick C, George A, Salzwedel H (2003) System-level simulation modeling with MLDesigner. In: 11th IEEE/ACM international symposium on modeling, analysis, and simulation of computer and telecommunication systems (MASCOTS), Orlando, FL, October 2003
14. Seagate Barracuda 7200.8 400 GB 3.5″ IDE Ultra ATA100 Hard Drive–OEM, http://www.newegg.com/Product/Product.asp?Item=N82E16822148060 (accessed: April 23, 2006)
15. Stanat DF, Weiss SF (2006) Systematic programming. Online book resources, http://www.cs.unc.edu/~weiss/COMP114/BOOK/BookChapters.html (accessed: June 1, 2006)
16. Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the Condor experience. Concurr Comput Pract Experience 17(2–4):323–356
17. Top 500 supercomputer sites, http://www.top500.org/ (accessed: April 23, 2006)
18. Vaidya NH (1995) A case for two-level distributed recovery schemes. In: ACM SIGMETRICS conference on measurement and modeling of computer systems, Ottawa, May 1995
19. Vaidya NH (1997) Impact of checkpoint latency on overhead ratio of a checkpointing scheme. IEEE Trans Comput 46(8):942–947
20. Wong KF, Franklin M (1996) Checkpointing in distributed systems. J Parallel Distributed Syst 35(1):67–75

Rajagopal Subramaniyan is presently employed with Cisco Systems, Inc. as a software engineer. He received the B.E. degree in Electronics and Communication Engineering from Anna University, India and the M.S. and the Ph.D. degree in Electrical and Computer Engineering from the University of Florida. At University of Florida, he co-led the high-performance computing and communication group and was also a member of the advanced space computing group at the High-Performance Computing and Simulation Research Laboratory. His research interests include high-performance computing, systems and networks, and fault-tolerant computing.

Eric Grobelny is a Ph.D. student in Electrical and Computer Engineering at the University of Florida. He received the B.S. degree in Electrical and Computer Engineering from the University of Florida Presently, he works as a research assistant at the High-performance Computing & Simulation Research Laboratory. His main focus of research is in performance prediction of parallel scientific applications for clustered and embedded systems through modeling and simulation. He is also the team leader of the Mission Assurance group which focuses on disaster recovery and business continuity in high-performance computing environments.

Scott Studham is the Chief Information Officer for Oak Ridge National Laboratory (ORNL) and is responsible for planning and executing a coordinated information technology strategy that ensures cost-effective, state-of-the-art computing and networking capabilities for ORNL from the desktop to high-performance computing. He earned B.S. degree in chemistry and the M.S. degree in computer science from Washington State University. He also has earned many professional certifications in project management and information technology management. He received the 2003 Smithsonian Institution's Award for Innovative Technology in Information Technology and the 2004 IEEE Supercomputing Conference StorCloud Challenge Award for the most innovative use of storage.

Alan D. George is Professor of Electrical and Computer Engineering at the University of Florida, where he serves as Director and Founder of the HCS Research Laboratory. He received the B.S. degree in Computer Science and the M.S. in Electrical and Computer Engineering from the University of Central Florida, and the Ph.D. in Computer Science from the Florida State University. Dr. George's research interests focus on high-performance architectures, networks, services, and systems for parallel, reconfigurable, distributed, and fault-tolerant computing. He is a senior member of IEEE and SCS, and can be reached by e-mail at george@hcs.ufl.edu.