Architectural Vulnerability Aware Checkpoint Placement in a Multicore Processor

Atieh Lotfi, Arash Bayat, Saeed Safari
School of Electrical and Computer Engineering
College of Engineering, University of Tehran, Iran
{a.lotfi, a.bayat}@ece.ut.ac.ir, saeed@ut.ac.ir

***Abstract*— As the system complexity increases, the failure probability increases substantially. Therefore, the system requires techniques for supporting fault tolerance. Checkpointing technique is widely used to reduce the execution time of long-running programs in presence of failures and enhancing the reliability of such systems. Several methods were studied thus far in order to determine the checkpointing interval which optimizes system performance. The crucial parameter in all of these solutions is system failure model which is primarily assumed as exponential or Weibull distributions. But, these models are not perfectly accurate since they fail to model the effect of soft errors. In this paper, we introduce a more realistic failure model based on the processors AVF. In addition, we propose three checkpoint placement methods with constant and variable intervals that determine suitable checkpoint places for the proposed failure model. Our experimental results show that our method, which is implementable on any multicore system, can find the suitable points in which checkpoints should be taken**.

I. INTRODUCTION

Recent changes in the high performance parallel computing make fault tolerant system design important. The growth in the complexity of parallel architectures and software design increases their failure probability. Furthermore, soft errors are becoming a serious concern in these state-of-the-art hardware architectures. Soft errors affect both hardware and software reliability. However, not all soft errors will affect the program output. Indeed, only a fraction of processor bits will influence the final program output in each cycle, while any fault in other bits will be masked. Architectural Vulnerability Factor (AVF) is used to quantize this effect [1].

One of the widely used fault-tolerant techniques to deal with failures in such systems executing long-running programs is checkpointing. However, checkpoinitng comes with non-negligible overhead during failure-free operation. Therefore, it is necessary to keep a balance between the overhead caused by taking checkpoints and the amount of lost work when the system fails. In this paper, we address the optimum checkpoint placement problem to find the suitable checkpoint intervals which minimize the total execution time including checkpoint overhead and wasted time in the event of failure.

II. SYSTEM FAILURE MODEL

In order to explain our proposed system failure model, we define the term effective soft error rate as the product of soft error rate and AVF. The variability of processor vulnerability during program execution results in a variable failure rate. Previous works on checkpointing neglected this fact and did not consider the effect of effective soft error rate in their failure models. In the initial hours or days of system life, software or hardware failure rates are more dominant than soft error rate. However, after a specific amount of time, their effect becomes less important compared with soft error rate. To put it another way, most of the design or manufacturing errors in software or hardware will be detected and corrected in the initial weeks of system lifetime. So, as the simulation time goes and system passes its lifetime, the system failures are mostly due to effective soft errors. Therefore, it's better to study both situations to have a more accurate system failure model.

In order to calculate the effective soft error rate, we need to estimate system's AVF. Our multitasking multicore system executes different applications simultaneously. This system has a task scheduler that gives each task a random-length time slot. We define the term *system scheduling slot (SSS)* as the time between two consecutive scheduling in the system. In fact, after each task scheduling, SSS is the maximum period of time in which no other scheduling is done. Figure 1 clarifies this definition for a dual-core processor. In the next step, to calculate system's AVF, the following approach is used. First, we extract the AVF of hardware components for each application using *Sim-SODA* [2]. In order to calculate total AVF of the processor, a weighted average on different components' AVF is calculated. Therefore, we have the AVF of each task in each time slot. The AVF of an SSS is equal to the average of cores' AVF in the corresponding duration. Suppose in Figure 1, the AVF in TS_{11}, TS_{21}, and TS_{22} are equal to 30%, 40%, and 25% respectively. Then, the AVF for SSS_1 and SSS_2 are 35% and 27.5% respectively. The AVF for other SSS_i is calculated in the same way.

Thus, we estimate system AVF as a function of time. In the following, we explain how the system failure rate is calculated. As discussed before, the system's failure can be studied in two cases. For the initial hours of system operation (say times less than T_b), failures are distributed exponentially with constant failure rate λ. However, for the times greater that T_b, a variable failure rate is considered to model the effect of soft errors. As a lot of studies have aimed to model the first situation, in this paper, we focus our study on system failures for times greater than T_b. For this interval, we define the system failure rate per minute (λ_m) for a multicore processor as equation (1). Here, we assume a multicore processor with *N* cores, *c_s* Mbit cache and *s_b* Mbit processor state bits per each core.

978-1-4673-2085-6/12/$31.00 ©2012 IEEE

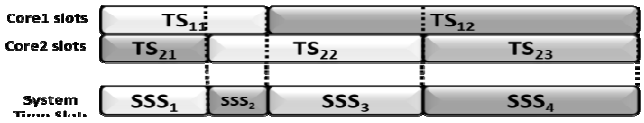

Figure 1. A dual-core task scheduling and its System Scheduling Slots

$$\lambda_m = \frac{FITperMbit \times N \times (c\_s + s\_b)}{10^9 \times 60}. \quad (1)$$

The system failure rate in any time greater than T_b is calculated as the product of λ_m and the computed AVF in that time (Equation (2)).

$$\lambda(t) = \lambda_m \times AVF(t). \quad (2)$$

The system failure is modeled by exponential distribution with variable failure rate $\lambda(t)$. Therefore, the failure density function is defined as equation (3).

$$f(t) = \lambda(t) \times e^{-\int_0^t \lambda(\tau)d\tau}. \quad (3)$$

Figure 2 shows the failure density function of our multicore system under study. The solid line illustrates f(t) for 150 days. The axes on the left and bottom indicate the values related to this curve. To show the variations of f(t) clearer, we magnify this curve for five days shown as the dashed line. The right and top axes hold the values for the zoomed area.

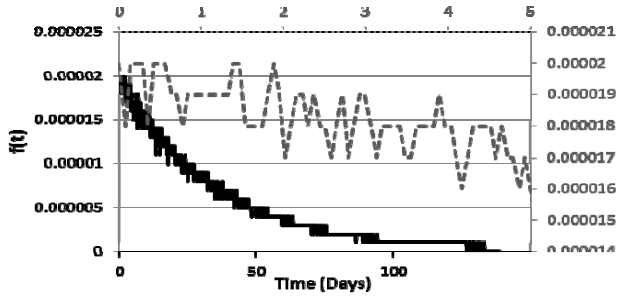

Figure 2. Failure probability density function against time for soft errors

III. CHECKPOINT PLACEMENT STRATEGY

Assume a multicore system that executes different tasks for T time units in a fault-free environment. The goal is to find suitable places for taking checkpoints which minimize the total wasted time in a faulty environment. In other words, we want to find n time intervals $t_1, t_2, \ldots, t_n$, such that $\sum_{i=1}^{n} t_i = T$. A checkpoint is taken at the end of each time interval. Figure 3 shows the checkpoint/restart model. Each checkpoint takes T_{ov} to be stored in a stable storage. T_r is the time needed to recover the task from failure and $T_{rollback}$ is the lost processing time due to the failure.

First, we define the lost time for a checkpointing slot, $L_{s_i}(t)$, if a failure occurs in time t during ith checkpoint interval. Let T_{B_i} and T_{E_i} be the beginning and the end of that interval, respectively. It should be considered that storing each checkpoint adds an extra overhead to the system execution time (T_{ov}). Therefore, as stated in equation (4), $L_{s_i}(t)$ is simply defined as sum of time overheads caused by taking checkpoints and rollback time (the distance between the failure time and T_{B_i}).

$$L_{s_i}(t) = t - T_{B_i} + (i-1)T_{ov}. \quad (4)$$

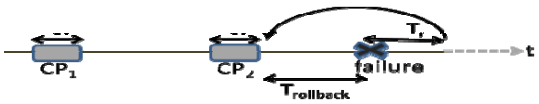

Figure 3. The Checkpoint/restart procedure

Next, we compute the total wasted computation time, T_w, in all checkpointing intervals which is the addition of times wasted due to the failure and storing checkpoints. This leads to the following equation:

$$T_w = T_r + \sum_{i=1}^{n} \int_{T_{B_i}}^{T_{E_i}} L_{s_i}(t) \times f(t)dt \quad (5)$$

$$= T_r + \sum_{i=1}^{n} \int_{T_{B_i}}^{T_{E_i}} (t - T_{B_i} + (i-1)T_{ov}) \times \lambda(t) \times e^{-\int_0^t \lambda(\tau)d\tau} \mathrm{dt}.$$

Since the failure rate is a function of system vulnerability, which is a measured parameter, we convert all integrals to summation:

$$T_w = T_r + \sum_{i=1}^{n} \sum_{t=T_{B_i}}^{T_{E_i}} (t - T_{B_i} + (i-1)T_{ov}) \times \lambda_i \times e^{-\sum_{j=1}^{i} \lambda_j}. \quad (6)$$

Three checkpoint placement strategies are considered in this paper with the objective of minimizing T_w. In all methods, we experimentally find the suitable checkpoint placements. The first strategy uses periodic checkpointing method. In this method the checkpoint interval is constant. We find the best checkpointing interval experimentally among different options. Our simulator tests different checkpointing intervals and chooses the one which minimizes T_w.

In the second strategy, namely threshold-based method, we use checkpoint placement strategy with variable checkpoint intervals. Our threshold-based checkpointing method uses a threshold value, Th_{Art}, in order to determine the suitable checkpoint intervals. Here, the objective is to keep the average rollback time in each checkpoint interval equal to the threshold value. In other words, the threshold value is an upper bound on the average rollback time in each interval. Using this idea, for an interval with the start time T_B, equation (7) determines the end of the interval, T_E.

$$\int_{T_B}^{T_E} (t - T_B)\, f(\mathrm{t})\, \mathrm{dt} = Th_{Art}. \quad (7)$$

The best threshold value is chosen experimentally among a wide range of different values with the goal of minimizing T_w. The reason behind defining threshold value is to change the checkpoint intervals according to the probability of failure. When the failure probability is high, the checkpoint intervals become smaller. As the system becomes safer, the distance between checkpoints becomes longer to reduce the overhead imposed by checkpointing. Equation (7) tries to keep the average rollback time per failure constant. This method decreases the number of checkpoints by omitting the unnecessary checkpoints when the system is in a safe state.

The third checkpointing method proposed in this paper is a combination of the two previous methods. We call it hybrid method. We combine these methods to take advantage of both methods and optimize both the total wasted time and number of checkpoints. In the hybrid method, first we use periodic checkpointing with the checkpointing interval extracted from our pure periodic checkpointing method. When the cumulative distribution of the failure reaches to 0.98, we change the strategy and determine the suitable interval by finding a

threshold value. We find the point in which the strategy should be changed experimentally as will be explained in the next section.

IV. Simulation Results

In this work, we model a multicore server system, with sixteen processor cores, which executes multiple applications simultaneously. In this system, coordinated checkpointing is used. We implemented the proposed checkpoint placement methods on our multicore simulator. We assume each multicore processor has 4MB (32Mb) cache and 1Mb processor state bits. As we investigate the soft error rate in processor memory bits (including state bits and cache), we use 10000/Mbit FIT as reported in [3]. In our experiments T_{ov} is set to 10 minutes [4]. We model the system operation for two years. First, we run SPEC2000 benchmarks on Sim-SODA and extract the components' AVF. Then, we estimate the system's AVF by calculating a weighted average on the extracted AVFs. Applications are scheduled by the system scheduler with time slots between 10 and 60 minutes. Also, we assumed each core may fail anytime.

We compute the average AVF of the system in each system scheduling slot. Since every core runs different tasks on different scheduling intervals, the AVF variation of the whole system is substantial. With these data available, we estimate system failure probability in any time and evaluate our checkpointing strategies. First, we evaluate our simulator with a constant failure rate and compare our periodic checkpoint placement method with an existing method presented in [5]. Since all of the existing studies use a constant failure rate, we evaluate our method with a constant failure rate $\lambda_m = 0.01$ without considering the effect of soft error rate and AVF. The checkpoint interval is changed until we find the minimum total wasted time. In this case, the best checkpoint interval is obtained equal to 48 minutes with total wasted time equal to 38.62 minutes over two years of system operation. Table 1 shows the comparison of our periodic checkpoint placement method with the model proposed in [5] for the constant failure rate ($\lambda_m = 0.01$). As can be seen in this table, our method results in less total wasted time even considering a constant failure rate.

Then, we focus on evaluating our proposed methods using variable failure rate model starting from our periodic checkpointing strategy. The checkpoint interval is changed over a wide range of options in our simulator to find the best interval which minimizes the total wasted time. The best interval in this case is 1038 minutes which has the total wasted time around 1004 minutes over two years.

TABLE 1. Comparison of Our Periodic Checkpointing with Method Proposed in [5] using Constant Failure Rate ($\lambda_m = 0.01$)

Method	*Checkpointing Interval (Minutes)*	*Number of Checkpoints*	*Total Wasted Time (Minutes)*
Our periodic method with constant λ_m	48	21900	38.62
Periodic method proposed in [5]	30	35040	43.12

Next, we experimentally find the optimum threshold in the threshold-based method. The minimum total wasted time is equal to 1157 minutes which can be achieved by Th_{Art} equal to 3.5. Although the total wasted time is greater than the achieved value using our periodic method, the number of checkpoints decreased 6 times. This happens because most of the checkpoints are taken where the failure probability is high, whereas the interval in other places becomes larger which increases the total wasted time.

To resolve the problem, we introduced the hybrid method. First, checkpoint interval is set to 1038 minutes which has been extracted from our first method. Next, we experimentally find the point in which the strategy should be changed. We set the value of cumulative distribution of failure (*CDF*) to 0.95 as the starting point of our trial and find the best Th_{Art} that minimizes T_w. This leads to Th_{Art}=0.25. Again, we fix Th_{Art} at 0.25 and find the best value of CDF to minimize T_w. This time we reach to CDF=0.98 that optimizes both the total wasted time and the number of checkpoints. Eventually, we find the best Th_{Art} for this CDF. To minimize T_w, the best Th_{Art} is equal to 0.07. However, if we increase T_w around 4 minutes, the number of checkpoints will be reduced around 70. It should be noted that the total wasted time in this method is very near to our periodic method with a significant decrease in number of checkpoints.

Table 2 summarizes the comparison of our proposed checkpoint placement methods with method presented in [5]. This time we compute the average failure rate in our system and use it as the constant failure rate for the method presented in [5]. Then, we use the extracted checkpoint interval, which is equal to 718 minutes, in our system with variable failure rate and report the number of checkpoints and total wasted time using their optimal interval. As shown in Table 2, in our threshold-based and hybrid methods we could reduce the number of checkpoints considerably. Moreover, the total wasted time is decreased in our periodic and hybrid methods. The hybrid method reduces number of checkpoints needed to be taken and keep total wasted time optimal.

TABLE 2. Comparison of Checkpoint Placement Methods

Method	*Number of Checkpoints*	*Total Wasted Time(Minutes)*
Periodic Method	1013	1004.6
Threshold-based Method	170	1157
Hybrid Method	368	1007
Optimal method proposed in [5]	1465	1066.69

References

[1] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," MICRO, 2003.

[2] X. Fu, T. Li, J. A. B. Fortes, "Sim-SODA: A Unified Framework for Architectural Level Software Reliability Analysis," Workshop on Modeling, Benchmarking and Simulation, 2006.

[3] Report, "Soft Errors in Electronic Memory-A White Paper," Technical report, Tezzaron Semiconductor, 2004.

[4] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, S. Scott, "A Reliability-aware Approach for an Optimal Checkpoint/Restart Model in HPC Environments," IEEE International Conference on Cluster Computing, 2007.

[5] Y. Liu, C. Leangsuksun, H. Song, S.L. Scott, "Reliability-aware Checkpoint /Restart Scheme: A Performability Trade-off," IEEE International Conference on Cluster Computing, 2005.