Level of Confidence Study for Roll-back Recovery with Checkpointing

Dimitar Nikolov[†] dimitar.nikolov@liu.se
Urban Ingelsson[†] urban.ingelsson@liu.se
Virendra Singh[‡] viren@serc.iisc.ernet.in
and
Erik Larsson[†] erik.larsson@liu.se

[†]Department of Computer Science
Linköping University, Sweden

[‡] Supercomputer Education and Research Centre,
Indian Institute of Science, India

Abstract

Increasing soft error rates for semiconductor devices manufactured in later technologies enforces the use of fault tolerant techniques such as Roll-back Recovery with Checkpointing (RRC). However, RRC introduces time overhead that increases the completion (execution) time. For non-real-time systems, research have focused on optimizing RRC and shown that it is possible to find the optimal number of checkpoints such that the average execution time is minimal. While minimal average execution time is important, it is for real-time systems important to provide a high probability of meeting given deadlines. Hence, there is a need of probabilistic guarantees that jobs employing RRC complete before a given deadline. Therefore, in this paper we present a mathematical framework for the evaluation of level of confidence, the probability that a given deadline is met, when RRC is employed.

I. Introduction

Computer systems can be classified as real time systems and non-real-time systems depending on the requirement of meeting time constraints, *i.e.* deadlines. Real-time systems can be further classified into soft and hard depending on the consequences when not meeting the given deadlines. For a hard real-time system, it is a catastrophe not meeting the deadline, while for soft real-time systems, violating the deadlines usually degrades the quality of service and the consequences are not catastrophic.

As semiconductor technologies are increasingly susceptible to soft errors, it is for computer systems becoming important to employ fault-tolerant techniques to detect and recover from soft errors during operation. However, fault tolerance comes at a cost and usually degrades the performance of the system. To minimize the performance degradation it is important to analyze and optimize the usage of fault tolerance such that the performance degradation is minimized. In this paper we study Roll-back Recovery with Checkpointing (RRC).

Instead of executing the complete job and in case of errors, re-execute the complete job, RRC makes use of checkpoints such that if an error is detected, a job is rolled back from the most recently saved checkpoint. Saving checkpoints, introduces a time overhead.The time overhead depends on the number of checkpoints. A high number of checkpoints leads to early error detection, and thus the penalty of re-execution from the recently saved checkpoint becomes less expensive in time. However, a high number of checkpoints causes more time overhead due to checkpointing, which increases the total execution time for the job. It is a problem to find the optimal number of checkpoints.

RRC has been the subject of research for both non real-time [3], [6] [7], [2] and real-time systems [4], [10], [9], [8], [5]. While for non real-time systems, it is important to minimize the average execution time when RRC is applied, it is for real-time systems important to maximize the probability that a job completes before a given deadline, [1]. When using RRC in real-time systems, both hard and soft, it is important to provide a reliability metric that indicates the probability of meeting deadlines. However, to the extent of our knowledge no such reliability metrics have been presented. Therefore, we focus in this paper on the analysis of RRC for real-time systems and we derive an expression to evaluate the probability that a job completes before a given deadline, *i.e.* the level of confidence.

II. System model

In this section we detail the Roll-back Recovery with Checkpointing (RRC) scheme and we provide some basic assumptions regarding the occurrence of soft errors.

The RRC scheme that we adopt assumes that a job is duplicated and concurrently executed on two processors (illustrated in Figure 1). During the execution of a job, a number of checkpoints are taken and compared against each other. If the checkpoints match, they are saved as a safe point from which a job can be restarted. If the checkpoints mismatch, this indicates that errors have occurred and therefore the job is restarted in both processors from the most recently saved checkpoint. In the scheme, RRC provides fault tolerance at expense of hardware redundancy, *i.e* two processors execute the same job, and time redundancy, *i.e.* taking and comparing checkpoints introduces a time overhead. We define checkpointing overhead, τ (see Figure 1), as the time required to carry out checkpoint operations, *i.e.* to load/store a checkpoint and compare the checkpoints from the two processors. We assume that τ takes a constant amount of time for any checkpoint.

We define the portion of a job's execution between two successive checkpoints as an execution segment (see Figure 1). We refer to an execution segment as successful execution segment if no errors have occurred during the execution in both processors, or erroneous execution segment otherwise, *i.e.* if errors have occurred.

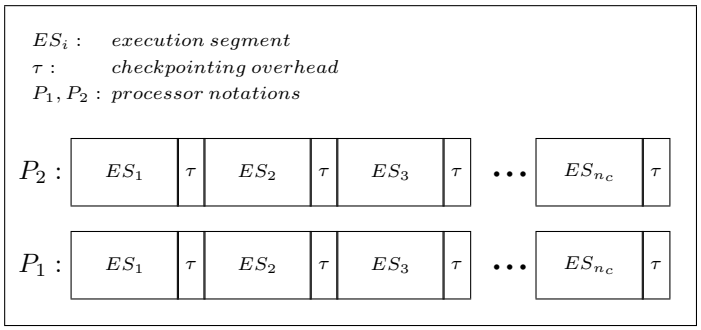


Figure 1: Graphical presentation of RRC scheme

For a job, we assume given is the processing time, T, which is the time required for a job to complete when RRC is not used and no errors have occurred during the execution of the job. When RRC is employed, a number of checkpoints are taken, n_c. Having n_c checkpoints, implies n_c execution segments and each segment is of length of $\frac{T}{n_c}$.

Next, we elaborate on the occurrence of soft errors. We assume that occurrence of soft errors is an independent event. In our work, given is the probability, P_T, that no errors occur in a processor within an interval equal to the processing time of the job, T. Due to the fact that the occurrence of soft errors is an independent event, we calculate P_ϵ, the probability of successful execution segment, with the following expression:

$$P_\epsilon = \sqrt[n_c]{P_T} \cdot \sqrt[n_c]{P_T} = \sqrt[n_c]{{P_T}^2} \tag{1}$$

Eq. 1 takes into account that no errors occur within an interval of length $\frac{T}{n_c}$ in both processors.

III. Evaluation of level of confidence

In this section we provide analysis and derive an expression to evaluate the level of confidence that a job that employs RRC will meet a given deadline. The level of confidence, with respect to a given deadline D, is the probability that a job completes before D. The level of confidence, is determined as the sum of intermediate terms that represent the probability that a job completes at a given discrete point in time. These terms are calculated according to a probability distribution function. Thus, to compute the level of confidence we need to derive an expression for the probability distribution function.

To derive the probability distribution function, we start by analyzing the expected completion time when RRC is employed. The expected completion time can be described by a discrete variable due to the fact that an integer number of execution segments (each followed by a checkpointing overhead) must be executed before a job completes. Assuming that n_c checkpoints are to be taken, a job can complete only when n_c successful execution segments have been executed. Thus, in the best case scenario, when no errors have occurred, a job completes after n_c executions segments. Each execution segment is of length $\frac{T}{n_c}$ plus the checkpointing overhead, τ. We denote the case when zero erroneous execution segments are executed with t_0 and it is defined as :

$$t_0 = n_c \cdot (\frac{T}{n_c} + \tau) = T + n_c \cdot \tau \tag{2}$$

If errors occur, and these errors only affect the execution of *one* execution segment, this segment will be re-executed. There will be $n_c + 1$ execution segments executed (one erroneous and n_c successful execution segments). We denote the case when one execution segment is re-executed with t_1 and it is defined as:

$$t_1 = (n_c + 1) \cdot (\frac{T}{n_c} + \tau) = T + n_c \cdot \tau + (\frac{T}{n_c} + \tau) \tag{3}$$

In the general case, when there are k erroneous execution segments, t_k denotes the expected completion time and t_k is defined as:

$$t_k = T + n_c \cdot \tau + k \cdot (\frac{T}{n_c} + \tau) \tag{4}$$

Next, we analyze the number of cases that a job completes exactly at time t_k. First, let us study the case that a job completes at time t_0. This can happen if and only if all the execution segments were successful, no errors have occurred. This is the only possible alternative for a job to complete at time t_0. Now, let us assume that a job completes at time t_1. If a job completes at time t_1, a single execution segment has been re-executed. This can be any of the n_c different execution segments. Thus, there are n_c possible cases that a job completes at time t_1. If a job completes at time t_2, two execution segments have been re-executed. It can be either two out of all n_c different execution segments were re-executed, or a single execution segment was re-executed twice (an error was again detected after the first re-execution). In general, if a job completes at time t_k, a total of $n_c + k$ execution segments have been executed, that is n_c successful execution segments and k erroneous execution segments. Note that the last execution segment among all $n_c + k$ execution segments must have been a successful execution segments otherwise it contradicts the assumption that the job has completed at t_k. Hence, the k erroneous execution segments are any of the $n_c + k - 1$ (any execution segment except for the last one). Therefore, the number of different cases that exists such that a job completes at time t_k is the number of all the combinations of k execution segments out of n_c+k-1 execution segments. $N(t_k)$ denotes the number of possible cases that a job completes at time t_k, and $N(t_k)$ is defined as:

$$N(t_k) = \binom{n_c + k - 1}{k} \tag{5}$$

In Figure 2(a) we illustrate $N(t_k)$ (Eq. 5) for $n_c = 3$, $P_T = 0.5$ and $t_k \in [t_0, t_5]$. For example, $N(t_1) = 3$ shows that there are three cases that a job completes at t_1, since any one of the three execution segments($n_c = 3$) could have been re-executed.

Next, to calculate the probability that a job completes at time t_k, we need a probability metric for each case (t_k). This probability metric is closely related to the probability that no errors will occur during execution of an execution segment, P_ϵ (Eq. 1). When a job completes at time t_k, $n_c + k$ execution segments were executed, n_c *successful* and k *erroneous* execution segments. Since P_ϵ represents the probability of successful execution segment, the probability of erroneous execution segment is evaluated as $(1 - P_\epsilon)$.

Since execution segments are independent, the probability of having n_c successful execution segments is $P_\epsilon^{n_c}$, and the probability of having k erroneous execution segments is $(1-P_\epsilon)^k$. Combining these two probabilities, probability of n_c successful and k erroneous execution segments, we get $P_\epsilon^{n_c} \cdot (1-P_\epsilon)^k$, which is the probability metric per possible case when a job completes at time t_k. In Figure 2(b), we illustrate the probability metric per possible case, $P_\epsilon^{n_c} \cdot (1-P_\epsilon)^k$, for $n_c = 3$, $P_T = 0.5$ and $t_k \in [t_0, t_5]$. From Figure 2(b) it can be observed that the probability metric, $P_\epsilon^{n_c} \cdot (1-P_\epsilon)^k$, has the highest value at t_0 and it is evaluated as $P_\epsilon^{n_c} = \sqrt[n_c]{{P_T}^2} = 0.25$. The probability metric per case, $P_\epsilon^{n_c} \cdot (1-P_\epsilon)^k$, drops rapidly by increasing t_k.

To calculate the probability that a job completes at time t_k, we need to multiply the number of possible cases, $N(t_k)$, with the probability metric per case. We denote the probability that a job completes at time t_k with $p(t_k)$, and it is defined as

$$\begin{aligned} p(t_k) &= N(t_k) \cdot P_\epsilon^{n_c} \cdot (1-P_\epsilon)^k \\ &= \binom{n_c+k-1}{k} \cdot P_\epsilon^{n_c} \cdot (1-P_\epsilon)^k \end{aligned} \tag{6}$$

Eq.6 defines the probability distribution function. In Figure 2(c) we illustrate the probability distribution function for $n_c = 3$, $P_T = 0.5$, and $t_k \in [t_0, t_5]$. To evaluate the level of confidence it is required to sum all terms from the probability distribution function $p(t_k)$ for which the discrete variable t_k has a value which is lower or equal to the given deadline, D. We denote the level of confidence of meeting the deadline, D, with $\Lambda(D)$, and we evaluate it by using the following expression:

$$\begin{aligned} \Lambda(D) &= \sum_{k=0}^{t_k \le D} p(t_k) \\ &= \sum_{k=0}^{t_k \le D} \binom{n_c+k-1}{k} \cdot P_\epsilon^{n_c} \cdot (1-P_\epsilon)^k \end{aligned} \tag{7}$$

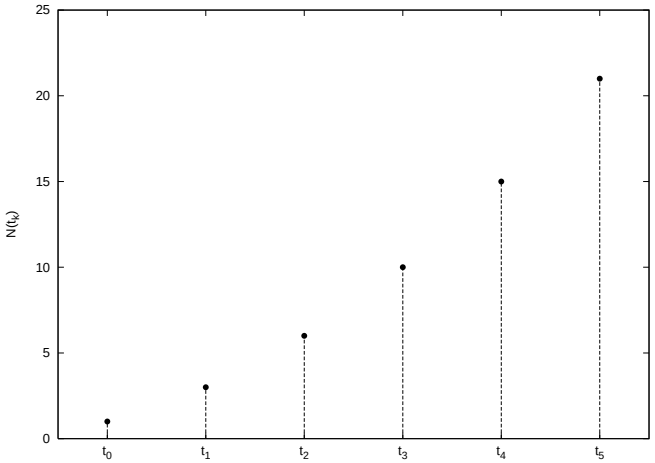
IV. Results

In this section we demonstrate the mathematical framework to evaluate the level of confidence that a job employing RRC completes before a given deadline. For the result set we used two input scenarios, Scenario A and Scenario B, presented in Table I. For each scenario, the following inputs are given: the processing time of a job (T), checkpointing overhead (τ), and the probability (P_T) that no errors occur in the processors within an interval equal to T.

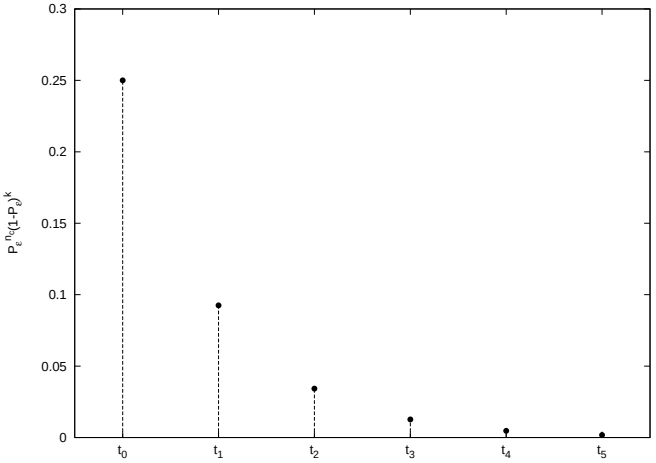
For the presented result set, we assume given is a deadline $D = 1500$t.u. (time units). The results represent the computed level of confidence that the job meets the deadline (D) when RRC is employed with different number of

Scenario A	Scenario B
$T = 1000$t.u.	$T = 1000$t.u.
$\tau = 20$t.u.	$\tau = 20$t.u.
$P_T = 0.99999$	$P_T = 0.9$

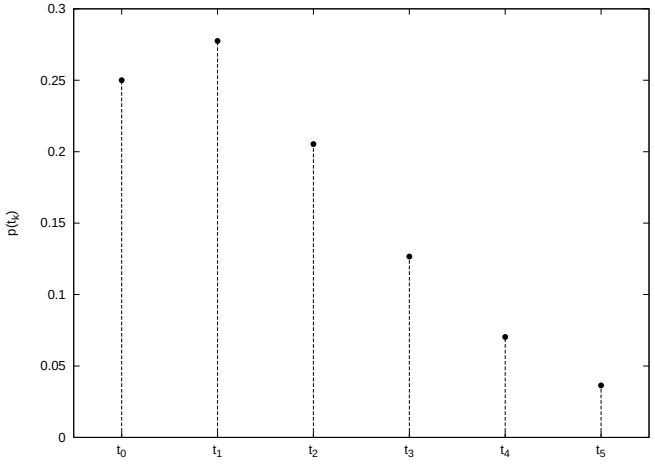
Table I: Input Scenarios



(a) Number of cases N(t_k)



(b) Probability metric per case $P_\epsilon^{n_c}(1-P_\epsilon)^k$



(c) Probability distribution function $p(t_k)$

Figure 2: Illustration of defined functions for $n_c = 3$ and $P_T = 0.5$.

$D = 1500$			
n_c	$\Lambda(D)$	n_c	$\Lambda(D)$
1	0.999980000100000000	14	0.999999999999998367
2	0.999980000100000000	15	0.999999999999998388
3	0.999999999733334814	16	0.999999999999998406
4	0.999999999750001250	**17**	**0.999999999999998422**
5	0.999999999760001120	18	0.999999999788889670
6	0.999999999999997925	19	0.999999999789474459
7	0.999999999999998040	20	0.999999999790000770
8	0.999999999999998125	21	0.999999999790476955
9	0.999999999999998189	22	0.999980000100000000
10	0.999999999999998240	23	0.999980000100000000
11	0.999999999999998280	24	0.999980000100000000
12	0.999999999999998314	25	0.999980000100000000
13	0.999999999999998343	26	0

Table II: Level of confidence for meeting deadline D using input Scenario A

$D = 1500$			
n_c	$\Lambda(D)$	n_c	$\Lambda(D)$
1	0.810000000000000000	14	0.998836333221060871
2	0.810000000000000000	15	0.998405709197021325
3	0.974827503159636872	16	0.998422589149847735
4	0.976266114316335439	**17**	**0.998437425722750770**
5	0.977137362167560214	18	0.979688847172390437
6	0.997980204415657095	19	0.979741032210778210
7	0.998085015474654920	20	0.979788017059326005
8	0.998162202793752259	21	0.979830542116846522
9	0.998221387037794418	22	0.810000000000000000
10	0.998268194669895683	23	0.810000000000000000
11	0.998306132813719019	24	0.810000000000000000
12	0.998337499909652013	25	0.810000000000000000
13	0.998363864473716882	26	0

Table III: Level of confidence for meeting deadline D using input Scenario B

checkpoints, n_c. The results are summarized in Table II and Table III. For each n_c, first we calculate K, the number of re-executions that can be accommodated within the interval $[t_0, D]$, and then we sum all terms from the probability distribution function (Eq. 6) for $t_k \in [t_0, t_K]$. As can be seen from Table II and Table III, the level of confidence, $\Lambda(D)$, for meeting a given deadline, D, depends on the number of checkpoints, n_c. When the number of checkpoints is low, the level of confidence is not sufficiently high. The level of confidence increases as the number of checkpoints increases. However, at a certain number of checkpoints, increasing the number of checkpoints further results in decreased level of confidence or even leads to a zero level of confidence. The reason is that when the number of checkpoints is low, the execution segments are longer, which means that it is difficult to accommodate many re-executions while meeting the deadline. This implies that only a small number of terms from the probability distribution function (Eq. 6) will be summed and therefore the level of confidence(Eq. 7) is low. Increasing the number of checkpoints, decreases the length of the execution segments and thus allows more re-executions to be accommodated before the deadline on one hand, but increases the total checkpointing overhead on the other hand. Having a high number of checkpoints may result in a zero level of confidence. As t_0, the case when zero erroneous execution segments are executed, depends on the number of checkpoints, n_c, (Eq. 2), having a high number of checkpoints may result in that t_0 violates the deadline D, *i.e.* $t_0 > D$. For example, for the given input scenarios when $n_c = 26$, $t_0 = 1000 + 26 \cdot 20 = 1520$ and the level of confidence $\Lambda(D) = 0$, (see Table II and Table III). With this result set we want to point out that it is useful to have a framework to calculate the level of confidence because it makes it possible to optimize the RRC scheme such that an optimal number of checkpoints that results in the highest level of confidence for meeting the given deadline can be obtained. From the presented results in Table II and Table III, we note the number of checkpoints that provides the highest level of confidence for the job to meet the deadline D is $n_c = 17$ for both Scenario A and Scenario B. However, the level of confidence for Scenario A is much higher than the level of confidence for Scenario B and it is due to the different values for P_T.

V. Conclusion

Due to the increasing soft error rates, fault tolerance becomes important in system design. One well studied fault-tolerant technique is Roll-back Recovery with Checkpointing (RRC) that copes with soft errors at the expense of introducing time overhead. This time overhead can be the reason of violating deadlines in real-time systems.

In this paper we have focused on analyzing RRC for real-time systems. We presented a mathematical framework to evaluate the level of confidence that a job employing RRC meets a given deadline. Our mathematical framework is important not only for computing the level of confidence and therefore getting a reliability metric, but also it is useful to acquire knowledge on how to adjust the RRC scheme, *i.e.* adjust the number of checkpoints to be taken, such that the level of confidence is maximized.

References

[1] I. Koren and C.M. Krishna, " Fault-Tolerant Systems", Morgan Kaufman, 1979

[2] M. Väyrynen, V. Singh, and E. Larsson, " Fault-Tolerant Average Execution Time Optimization for General-Purpose Multi-Processor System-on-Chips", *Design Automation and Test in Europe (DATE 2009)*, Nice, France, April, 2009.

[3] D. Nikolov, U. Ingelsson, V. Singh, and E. Larsson, "Estimating Error-probability and its Application for Optimizing Roll-back Recovery with Checkpointing", delta, pp.281-285, 2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications, 2010

[4] S. Punnekkat, A. Burns, and R. Davis "Analysis of Checkpointing for Real-Time Systems",*The International Journal of Time-Critical Computing Systems*,20, pp.83-102, 2001

[5] S. W. Kwak, B. J. Choi, and B. K. Kim Ling, "An Optimal Checkpointing-Strategy for Real-Time Control Systems Under Transient Faults", *IEEE Transactions on reliability*, vol. 50, no.3, September 2001.

[6] A. Ziv and J. Bruck, "Analysis of Checkpointing Schemes with Task Duplication", *IEEE Transactions on computers*, vol. 47, no.2, February 1998.

[7] A. Ziv and J. Bruck, "An On-Line Algorithm for Checkpoint Placement", *IEEE Transactions on computers*, vol. 46, no.9, September 1997.

[8] V. Grassi, L. Donatiello and S. Tucci, "On the Optimal Checkpointing of Critical Tasks and Transaction-Oriented Systems", *IEEE Transactions on software engineering*, vol. 18, no.1, January 1992.

[9] K. G. Shin, T. Lin, and Y. Lee, "Optimal Checkpointing of Real-Time Tasks", *IEEE Transactions on computers*, vol. C-36, no.11, November 1987.

[10] Y. Zhang and K. Chakrabarty, "Fault Recovery Based on Checkpointing for Hard Real-Time Embedded Systems", *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)*, 2003.