Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 70 (2015) 821 – 828

Procedia Computer Science

4th International Conference on Eco-friendly Computing and Communication Systems, ICECCS 2015

Adaptive Checkpoint Interval Algorithm considering Task Deadline and Lifetime Reliability for Real-Time System

Mohamad Imran bin Bandan[a,*], Subhasis Bhattacharjee[a], Dhiraj K. Pradhan[a], Jimson Mathew[a]

[a] *University of Bristol, Merchant Venturers Building, Woodland Road, Bristol BS8 1UB, United Kingdom*

Abstract

Checkpointing mechanism is used to tolerate the impact of transient faults by rollback operation. Recently, it has also been used as a mechanism to enhance system's lifetime by identifying and tolerating permanent fault [5,19,10,12]. However, equidistant checkpoint interval may cause task deadline violation in the system. Here, we propose an adaptive checkpoint interval placement algorithm (ADeLiRACI) that meets all tasks deadline. The checkpoint intervals are adjusted to minimize the impact of stresses and permanent faults on the running hosts. This novel mechanism allows greater applicability in real time systems with hard deadline such as weather prediction, financial transactions etc. We compare the estimated completion time for increasing fault-rate in the system against five existing algorithms. For all applications, ADeLiRACI is able to meet the hard deadline along with enhancing lifetime reliability of the system.

Peer-review under responsibility of the Organizing Committee of ICECCS 2015

Keywords: Checkpointing; checkpoint placement; checkpoint interval; lifetime reliability; task deadline

1. Introduction

Transient and permanent fault manifested by technology scaling, operating environment, wear-out, stress effect, etc. has become a major reason of shorter lifetime reliability on integrated circuit. Usually checkpointing mechanisms [1,3,8,20,14] are proposed to combat transient faults. In [5,10,12], authors introduce checkpointing mechanisms that not only recovers system from transient faults but also from permanent faults. Checkpointing mechanisms mostly designed to take checkpoint in equally spaced time/cycle, referred as *interval* [18,1,5,19]. Authors in [18] extends rollforward scheme proposed by [1,3], through recording the state and performing self-detection. If it is confirmed to be in no fault state, the system continues to the next interval. Whenever self-detection is unable to establish any fault, the system performs comparison with its coupled host and compare their states. If the other host passed the self detection, its state is then copied to the faulty host. Authors in [5,19] proposed a checkpointing mechanism to not only tolerate transient fault but also permanent fault. The checkpointing algorithm works by using an extra storage to store the

* Tel.: +44-779-590-9208

E-mail address: mb12406@bristol.ac.uk

Peer-review under responsibility of the Organizing Committee of ICECCS 2015

doi:10.1016/j.procs.2015.10.124

last state it executed successfully; not only current and previous states. Authors in [10,12] further extends this idea by proposing a checkpointing mechanism with the objective of tolerating transient fault through rollback operation and permanent fault through Mean-Time-To-Failure (MTTF) estimation and migration. This mechanism smartly uses two checkpoints, Valid Checkpoint and Tentative Checkpoint located on the processor/host itself and three registers (for each host) in the controller to store checkpoints and comparison results. However, distance between two checkpoints or intervals, can be varied due to various objectives in real-time systems such as power, energy, deadline etc. Authors in [4] proposed a technique to determine checkpoint interval using a combination of interval formulation and Dynamic Voltage Scaling (DVS). In [14], authors extended this idea by identifying the appropriate number of checkpoints that minimizes the worst-case response time and optimizes energy consumption. Authors in [15] correlates parameters such as time to checkpoint, recovery time, fault-rate, deadline and Quality-of-Service (QoS) to determine checkpoint interval. Authors in [16,13] uses combination of Dynamic Voltage Scaling (DVS), Compare-and-Store Checkpoint (CSCP) and additional Store Checkpoint (SCP) to determine the checkpoint interval. In brief, the approach uses equidistant CSCP by adopting popular checkpoint interval formulation [4,7,6] and added more SCP in between those CSCP. Authors in [11,17] however uses multi speed processor model and energy left in the system as their determining parameters to obtain the next checkpoint interval. Thus, voltage threshold value is calculated as the break-off point in their approach. We define our contribution of this paper as follows:

1. Propose an adaptive checkpoint interval placement algorithm to complement Lifetime Reliability-Aware Checkpointing Algorithm (LRAC) [10,12]. The proposed algorithm is not to be mistaken as checkpointing algorithm as it's main objective is to determine when is the 'best' time to take checkpoints instead of how does the checkpointing algorithm works
2. Uses lifetime reliability (MTTF) of the system and task's deadline as determining parameter to determine checkpoint interval
3. Define conditions for task migration in LRAC mechanism [10,12]

The remainder of this paper is organized as follows. In Section 2, we outline the checkpoint interval and task deadline formulation. In Section 3, we explain the system model and underline assumptions. Section 4 explain our proposed algorithm. In Section 5, we compare it with existing algorithms. We conclude our paper in Section 6.

2. Checkpoint Interval Computation

We considers two important parameters; instantaneous Mean-Time-To-Failure (MTTF) and task deadline. MTTF is a measurement in years to represent the prediction of a failure to occur [21,22]. An instantaneous MTTF of a system is the expected time for a permanent fault to occur with regards to current operating condition [12]. Deadline however is a time constraint that is to be met by the system when running the task [9]. Our proposed algorithm's checkpoint interval computation takes hard deadline (D_{hard}) proposed in [12] as the deadline parameter.

$$D_{hard} = T_{total} + (3 \times (D_{min} - T_{total})) \tag{1}$$

T_{total} is expected total execution time (best-case) for allocated task in the allocated system when executed without any checkpointing mechanism.

D_{min} is a deadline parameter introduced in [8] to assess the deadline of the task (considering the current fault-rate of the system) when executed together with normal rollback checkpointing algorithm.

$$D_{min} \geq T_{total} + \left(\frac{T_{total}}{\sqrt{\frac{2 \times C}{\lambda}}} - 1 \right) \times C \tag{2}$$

C is the checkpointing cost/overhead

(Note: We use $\lambda = 10^{-3}$ as our threshold based on analysis done by [12].

Here, D_{min} is a time parameter with the consideration of the fault-rate. Using the same concept and formulation, substituting λ with $\frac{1}{MTTF_{current}}$, we could obtain T_{wCP}.

$MTTF_{current}$ is the instantaneous MTTF value previously inspected.

T_{wCP} is estimated total execution time when tasks are executed together with checkpointing algorithm (LRAC) using current/available $MTTF_{current}$.

$$T_{wCP} = T_{total} + \left(\frac{T_{total}}{\sqrt{\frac{2 \times C}{\frac{1}{MTTF_{current}}}}} - 1 \right) \times C \tag{3}$$

From here, checkpoint interval value is obtained using equation 4.

$$CP_I = \frac{T_{total} \times C}{T_{wCP} + C - T_{total}} \tag{4}$$

(*Note:Equation 4 is obtained from*[4,8] *which derived from*[6,7])

For proposed adaptive algorithm, re-assessment of $MTTF_{current}$ is required. After re-assessment, latest completion time of the task is estimated to compute a new checkpoint interval. All the formulations are the same as equation 3 and equation 4 with only difference of T_{left} and T_{latest} substituting T_{total} and T_{wCP} respectively.

T_{left} is expected execution time left for allocated task considering the amount of instruction counts that is already executed without any checkpointing/fault tolerant mechanism (best-case scenario without fault tolerant mechanism)

T_{latest} is newly estimated total execution time when tasks are executed together with checkpointing algorithm (LRAC) using T_{left} and current MTTF inspected ($MTTF_{current}$)

3. System Model and Assumptions

Figure 1 is the extended system model in [10,12] for our proposed ADeLiRACI. This system model was developed for Lifetime-Reliability-Aware Checkpointing (LRAC) where it consists of two components; Hosts and the controller. It assumes a quasi-synchronous distributed system where each running host runs its tasks independently and synchronization is perform at each checkpoint. Each host has the capability to perform self-test mechanism and have sufficient stable storage to store two checkpoints (Valid Checkpoint and Tentative Checkpoint). The controller however consists only three registers (for each running host) to store Current-Value Register (CR), Previous-Value Register (PR) and Tentative-Value Register (TR). The controller is mainly used for storing and comparing states and results. Detailed explanation on the system model can be obtained from [10,12].

We added two extra registers that holds the value of instantaneous MTTF for each running hosts and another register to hold $MTTF_{current}$. MTTF values are the result of MTTF estimation done by individual hosts which is a routine in Self-Test module. The details of the MTTF estimation technique is not explained here due to page limitation. $MTTF_{current}$ however is the lowest MTTF value among the two running hosts compared by the controller. We also added two extra communications between hosts and controller; sending of MTTF values from hosts to controller and $MTTF_{current}$ from controller to hosts. All details are presented in Section 4 using state machine diagrams.

We assume the same conditions in [10,12]:

a. The controller together with the associated comparators are assumed to be fault-free
b. A spare host is assumed to be always available in the system. This means that an (identified) permanently faulty/unfit host is being replaced by a good one by some external agent (outside the system)
c. We also assume that there is only one occurrence of fault (transient or permanent) between any two checkpoints. However, we do not assume any bound on the total number of faults/types of faults (permanent or transient) that can occur in the system throughout its lifetime
d. The host's speed/frequency are similar, including the spare and the speed remains the same throughout its lifetime. However, frequency scaling can be use with the existence of proper synchronization technique. This assumption is not a limitation, rather just to simplify the explanation in this paper

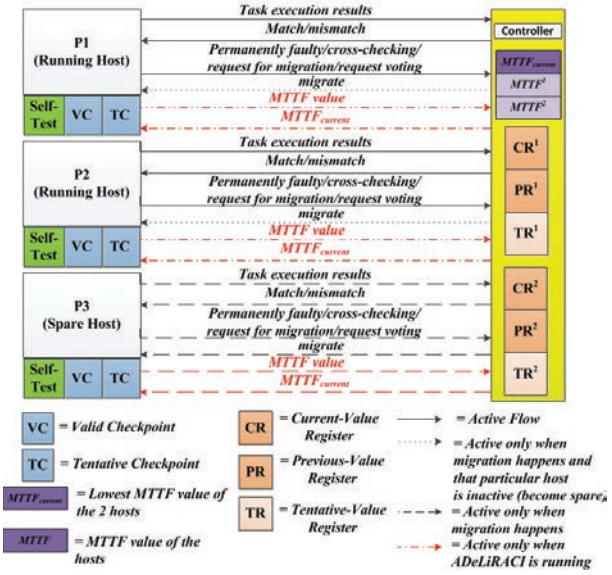

Fig. 1: System Model

4. Proposed Adaptive Deadline and Lifetime Reliability-Aware Checkpoint Interval Algorithm (ADeLiRACI)

We present our algorithm using Finite State Machine (FSM) representation for both controller and hosts. Similarly with system model presented above, state machine of host and controller are also designed based on [10,12] as shown in Figure 2(a) and 2(b) respectively. Our proposed algorithm shown by Algorithm 1 consists of three parts. The first part (Line 1 to 10) is executed at the beginning of the system's lifetime. The second part executed *every time a new task is being assigned to the hosts* and the third part is executed during run-time whenever re-evaluation of MTTF is required (Line 11 to 50). At the beginning of the system lifetime, MTTF estimation is invoke to obtain instantaneous MTTF

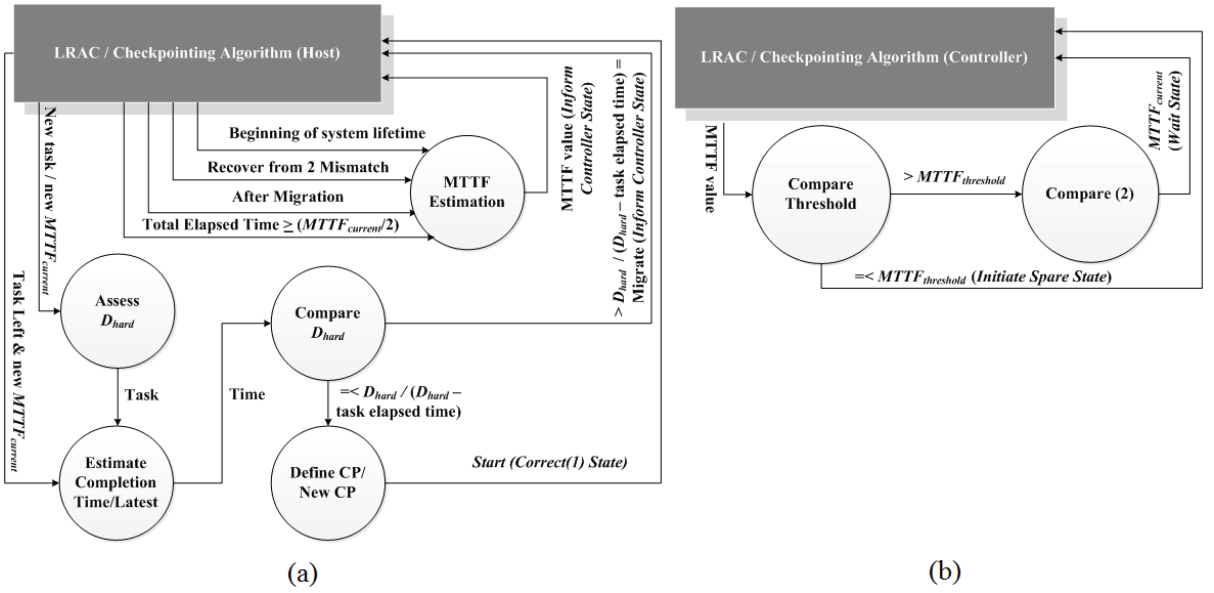

Fig. 2: State Machine Diagram for Host (a) and Controller (b)

for each host. These values are send to the controller for further processing.

Algorithm 1 ADeLiRACI

```
1: while (Beginning of system's lifetime) do
2:     Estimate MTTF
3:     if (MTTF ≤ MTTF_threshold) then
4:         Migrate task to another host (spare)
5:         Go to Line 2 onwards
6:     else
7:         Compare both host's MTTF
8:         Establish MTTF_current
9:     end if
10: end while
11: while (Task assigned) do
12:     Calculate D_hard
13:     Estimate completion time
14:     if (Estimated completion time > D_hard) then
15:         Migrate Task to another host (spare)
16:         Estimate MTTF
17:         if (MTTF ≤ MTTF_threshold) then
18:             Go to Line 15 onwards
19:         else
20:             Compare both host's MTTF
21:             Establish MTTF_current
22:             Go to Line 13 onwards
23:         end if
24:     else
25:         Define CP Interval
26:         Execute Task (Normal Operation using LRAC as CP algo)
27:         if (Total Elapsed Time ≥ MTTF_current / 2) then
28:             Estimate MTTF
29:             if (MTTF ≤ MTTF_threshold) then
30:                 Migrate task to another host
31:                 Go to Line 28 onwards
32:             else
33:                 Compare both host's MTTF
34:                 Establish MTTF_current
35:                 Estimate latest completion time
36:                 if (Latest comp. time> D_hard-task elapsed time) then
37:                     Go to line 30 onwards
38:                 else
39:                     Define new CP interval
40:                     Go to Line 26
41:                 end if
42:             end if
43:         else if (Recover from 2 MM) then
44:             Go to Line 28 onwards
45:         else if (Recover from permanent fault) then
46:             Go to Line 28 onwards
47:         else
48:             Go to Line 26
49:         end if
50:     end if
51: end while
```

Higher fault-rate translates to higher probability of having fault during the interval, thus more rollback operation may be required. This creates unnecessary overhead towards the task completion time. To avoid, shorter interval must be used. Also, if the completion time is estimated too high or the MTTF is inspected below the allowed threshold, the system performs task migration. $MTTF_{threshold}$ is an arbitrary predetermined value from the beginning of the system's lifetime. Whenever a new task is assigned, the system calculate D_{hard} for the task (instruction count) and $MTTF_{current}$. Then, estimated completion time is calculated and compare with D_{hard}. If it less than D_{hard}, the system proceed to

defining the checkpoint interval and execute the task accordingly. Otherwise, the system informs the controller and invokes task migration.

Apart from the early of the system's lifetime, there are three more conditions that require MTTF estimation; a) recovery from two consecutive mismatch (no permanent fault), b) after migration process is carried out (including recovery from permanent fault), and c) whenever total elapsed time is greater or equal to half of previously obtained $MTTF_{current}$. Whenever any of these conditions occurs, the system bypasses D_{hard} estimation state and moves directly to estimate latest completion time. The system compares it with ($D_{hard} - taskelapsedtime$). If latest completion time is less than ($D_{hard} - taskelapsedtime$), a new checkpoint interval is calculated. Subsequent tasks uses this new checkpoint interval until any of the three conditions re-appear again. Otherwise, the system informs the controller of this situation and starts migration process.

As for the controller, there are two states in the FSM diagram. Whenever the host is passing its MTTF value, the controller stores this value in its allocated registers. It also compares against $MTTF_{threshold}$. If the MTTF value is greater than the threshold value, the controller initiates *Compare(2)* state, where both MTTF values are compared to establish the lowest value and stores it as $MTTF_{current}$ and informs the running hosts. Both MTTF values are also stored respectively until a new value is obtained. If MTTF value is less or equal to the threshold value, controller performs initiation of the spare and a new host is added to the system to replace the faulty/unfit host. Similar steps are carried out to obtain MTTF value of the new host which results in a new checkpoint interval. This new checkpoint interval remains unaltered until it is again re-computed similarly. The complete details can be found in Algorithm 1.

5. Result and Discussion

We compare ADeLiRACI with LRAC to five existing approaches a) An Optimal Adaptive Checkpoint Strategy for DMR with Energy-Aware (CSCP with SCP) [16,13], b) Adaptive Checkpoint Placement in Energy Harvesting Real-Time System (ADPTCPEH) [11,17], c) An Efficient Forward Recovery Checkpoint Scheme in Dissimilar Redundancy Computer System (TDCS) [18], d) Checkpoint Management with DMR Based on the Probability of Task Completion (CPMDMR) [5,19], e) Equidistant Lifetime Reliability Aware Checkpointing (LRAC) [10,12]. This comparison is based on Table 1. Table 1 shows the characteristics of the host and programs inspected, based on SPEC95 benchmark programs where subsequent cycle per instruction (CPI) and instruction counts are obtained. The speed of the hosts are varied by two different speeds, to show how does it fared with different host's speed. We assume that both running host's frequency/speed are similar in this analysis. Although, variable speed (of the hosts) and usage of Dynamic Frequency Scaling (DVS) can also be considered with the existence of synchronisation technique. The instruction counts for all programs are multiplied by 10^{10} to make sure ample amount of checkpoints are inserted during the program's execution. We design a scenario where consecutive mismatch occur after 50% of the total task are executed. Initial

Table 1: Applications and host's characteristics

Set	Program/Application	Speed (Ghz)	Instruction Count (IC) $\times 10^{10}$	Cycle Per Instruction (CPI)
1	compress95.ss	1.2	3135520	0.5846
2	compress95.ss	1.5	3135520	0.5846
3	ijpeg.ss	1.2	22714	0.9716
4	ijpeg.ss	1.5	22714	0.9716
5	m88ksim.ss	1.2	99214	0.7995
6	m88ksim.ss	1.5	99214	0.7995
7	tomcatv.ss	1.2	14344	1.5049
8	tomcatv.ss	1.5	14344	1.5049
9	vortex.ss	1.2	41982	1.1566
10	vortex.ss	1.5	41982	1.1566

$\frac{1}{MTTF_{current}}$ is observed at 10^{-5}. Upon recovery, $\frac{1}{MTTF_{current}}$ is inspected at 10^{-3}. 10^{-3} [12] is the break-off point for migration used for this analysis. Once migration is performed, $\frac{1}{MTTF_{current}}$ is inspected to be 10^{-4}. Figure 3 shows the performance ratio of all mechanisms/algorithms (based on described scenario) compared to D_{hard} value. To meet the deadline, all algorithms must complete the task/programs assigned between the ratio of 0 to 1. With proposed ADeLiRACI, LRAC execution time is lower compared to equidistant LRAC. This can be seen by lower ratio value

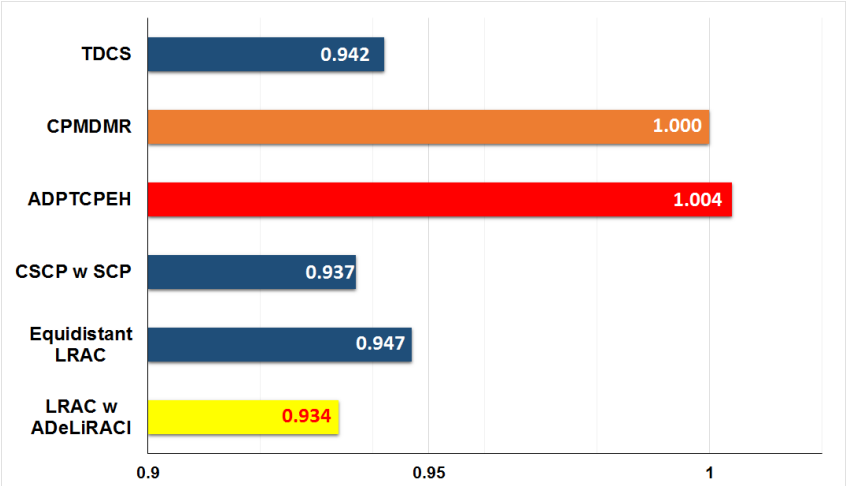

Fig. 3: Ratio to D_{hard} for all analysed algorithms

Table 2: Reliability metrics for interval before task executed and after recovery from consecutive mismatch

Set	Equidistant LRAC		LRAC with ADeLiRACI		CSCP with SCP		ADPTCPEH		CPMDMR		TDCS	
	Interval 1	*Interval 2*	*Interval 1*	*Interval 2*	*Interval 1*	*Interval 2*	*Interval 1*	*Interval 2*	*Interval 1*	*Interval 2*	*Interval 1*	*Interval 2*
1	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.8895	0.0000	0.9996	0.9995	0.9937	0.8888
2	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9008	0.0000	0.9996	0.9995	0.9937	0.8888
3	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9875	0.5292	0.9996	0.9995	0.9937	0.8888
4	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9888	0.6177	0.9996	0.9995	0.9937	0.8888
5	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9763	0.0429	0.9996	0.9995	0.9937	0.8888
6	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9788	0.0923	0.9996	0.9995	0.9937	0.8888
7	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9877	0.5385	0.9996	0.9995	0.9937	0.8888
8	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9890	0.6259	0.9996	0.9995	0.9937	0.8888
9	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9815	0.1818	0.9996	0.9995	0.9937	0.8888
10	0.9902	0.7077	0.9902	0.9892	0.9993	0.9989	0.9834	0.2754	0.9996	0.9995	0.9937	0.8888

in LRAC with ADeLiRACI (0.934) compared to equidistant LRAC (0.947). Also, it is the fastest running algorithm observed in this simulation. Secondly, CPMDMR's ratio is 1.000. CPMDMR uses deadline as one of its main parameter to determine the number of equidistant checkpoints. Since there is no deadline calculation mechanism explained, we use our D_{hard} formulation to do so thus results in ratio of 1. Lastly, ADPTCPEH is observed as violating the deadline imposed. ADPTCPEH's objective is to make sure the energy is enough to execute tasks. Under this approach, checkpoint interval is rather long thus the rollback overhead is also longer. For this simulation. we assume the energy left is not an issue hence high execution time.

$$R(t_{a2mm}) = e^{-(2(\lambda_P T))^{\beta}} \cdot e^{-(2\lambda_T T)} \tag{5}$$

where $R(t_{a2mm})$ is the reliability of the system for the interval after recovering from consecutive mismatch,
λ_P is the permanent fault-rate,
λ_T is the transient fault-rate,
β is the shape parameter,
T = interval + time taken to checkpoint

For reliability analysis, we assume λ_T is constant and λ_P is according to $\frac{1}{MTTF_{current}}$. The shape parameter is set to 2.5 and this is based on work in[2]. We analytically inspect the reliability metric using equation 5 at the first interval after recovery from the consecutive mismatch. For comparison, we inspect the reliability at the time when the task is assigned. The results are shown in Table 2. When compared to equidistant LRAC, LRAC with ADeLiRACI is able to maintain the reliability of 0.9XX compared to 0.7XX when executed using equidistant LRAC. Reduction of 0.28 is experienced when tasks are executed with equidistant LRAC compared to only 0.01 with LRAC with ADeLiRACI under described scenario. Secondly, although reliability of the system is maintained with LRAC with ADeLiRACI, it

is not the highest reliability inspected. LRAC with ADeLiRACI is observed to be below CSCP with SCP (0.9989) and CPMDMR (0.9995). It is because CSCP with SCP and CPMDMR uses very short interval (33.88 and 18.13 seconds respectively) compared to 154.92 seconds by LRAC with ADeLiRACI. Shorter intervals means greater reliability but at the expense of execution time.

6. Conclusions

Checkpointing has been widely used to tolerate fault and increase reliability of the system. However, the checkpoint intervals must be adjusted appropriately to avoid unnecessary overhead. Equidistant checkpoint interval is widely used because of its simplicity. However, adaptive checkpoint interval is needed to adapt the system to changes in operating environment. We proposed a new adaptive checkpoint interval algorithm that takes task deadline and the host's lifetime reliability as main parameters. We compare our algorithm with five existing algorithms [16,13,11,17,18,5,19,10,12]. For all applications, ADeLiRACI is able to meet the hard deadline along with enhancing lifetime reliability of the system.

Acknowledgment

This project is funded by Federal Government of Malaysia (Ministry of Higher Education) and Universiti Malaysia Sarawak (UNIMAS).

References

1. D. K. Pradhan and N. H. Vaidya. Roll-forward checkpointing scheme: A novel fault-tolerant architecture. *IEEE Trans. on Comp.* 1994 **43**:1163-1174
2. L. Huang and F. Yuan and Q. Xu. Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms. *Proc. of DATE* 2009
3. J. Xu and B. Randell. Roll-forward Error Recovery in Embedded Real-Time Systems. *Proc. of Int. Conf. on Par. and Dist. Sys.* 1996
4. Y. Zhang and K. Chakrabarty. Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems. *Proc. of DATE* 2003
5. J.M. Yang and S.W. Kwak. A Checkpoint Scheme with Task Duplication Considering Transient and Permanent Faults. *Proc. of the IEEM* 2010
6. J.W. Young. A first-order approximation to the optimum checkpoint interval. *Comm. ACM* 1974 **17**
7. A. Duda. The effects of checkpointing on program execution time. *Information Processing Letters* 1984 **16**:221-229
8. G. Aupy, A and Benoit, R Melhem, P. Renaud-Goud and Y Robert. Energy-Aware Chekcpointing of Divisble Tasks with Soft or Hard Deadlines. *Research Report Informatics Mathematic* 2013:1-30
9. M.Ben Ari. *Principles of Concurrent and Distributed Programming*. Prentice Hall;1990
10. Mohamad Imran Bandan, Subhasis Bhattacharjee, Rishad A. Shafik, Dhiraj K. Pradhan and Jimson Matthew. Lifetime Reliability-Aware Checkpointing Mechanism: Modelling and Analysis. *Proc. of International Symposium on Electronic System Design (ISED) 2013* 2013
11. Maryam Dehghan and Mehdi Kargahi. Adaptive Checkpoint Placement in Energy Harvesting Real-Time Systems. *Proc. of Iranian Conference on Electrical Engineering (ICEE2010)* 2010
12. Mohamad Imran Bandan, Subhasis Bhattacharjee, Dhiraj K. Pradhan and Jimson Matthew. Energy Efficient Lifetime Reliability-Aware Checkpointing for Real-Time System. *Journal of Low Power Electronics* 2014 **10**:1-16
13. Zhongwen Li, Hong Chen and Shui Yu. Performance Optimization for Energy-Aware Checkpointing in Embedded Real-Time Systems. *Proc. of DATE* 2006
14. Qiushi Han, Ming Fan and Gang Quan. Energy Minimization for Fault Tolerant Real-Time Applications on Multiprocessor Platforms Using Checkpointing. *Proc. of IEEE International Symposium on Low Power Electronics and Design (ISLPED)* 2013
15. Nianen Chen and Shangping Ren. Adaptive Optimal Checkpoint Interval and Its Impact on System's Overall Quality in Soft Real-time Applications. *Proc. of SAC'09* 2009
16. Zhongwen Li and Yefeng Jiang. An Adaptive Optimal Checkpoint Strategy for DMR with Energy-Aware. *Proc. of Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06)* 2006
17. Rami Melhem, Daniel Mosse and Elmootazbellah Elnozahy. The Interplay of Power Management and Fault Recovery in Real-Time Systems. *IEEE Transactions on Computers* 2004 **53**:217-231
18. Wang Guodong, Zhai Zhengjun, Huang Tao and Huang Kaichen. An Efficient Forward Recovery Checkpointing Scheme in Dissimilar Redundancy Computer System. *Proc. of Computational Intelligence and Software Engineering, 2009 (CiSE 2009)* 2009
19. Seong Woo Kwak, Kwan-Ho You and Jung-Min Yang. Checkpoint Management with Double Modular Redundancy Based on the Probability of Task Completion. *Journal on Computer Science and Technology* 2012 **27**:273-280
20. S. Mukherjee. *Architecture Design for Soft Errors*. Morgan Kaufmann Publisher; 2008
21. Global Standard for the Microelectronics Industry. *Methods for Calculating Failure Rates in Units of FITS (JESD85)*. JEDEC Publication; 2001
22. Global Standard for the Microelectronics Industry. *Failure Mechanisms and Model for Semiconductor Devices (JEP22C)*. JEDEC Publication; 2003