Work-in-Progress: Optimal Checkpointing Strategy for Real-time Systems with Both Logical and Timing Correctness

Lin Zhang, Zifan Wang, Fanxin Kong
Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse NY
lzhan120@syr.edu, zwang345@syr.edu, fkong03@syr.edu

***Abstract*—This paper proposes an optimal checkpoint scheme for fault resilience in real-time systems, in which we consider both logical consistency and timing correctness. First, we partition message-passing processes into a directed acyclic graph (DAG) considering their dependencies, where the logical consistency of checkpoints is guaranteed. Then, we find the critical path of the DAG, which is the longest path performed in sequence. Next, we analyze the optimal checkpoint strategy on the critical path where the overall execution time (including checkpointing overhead) is minimized. When a fault is detected, the system rolls back to the nearest valid checkpoint for recovery. The optimal number of checkpoints and their intervals are derived by the algorithm.**

***Index Terms*—Real-time systems, fault resilience, checkpointing, logical consistency, timing correctness**

I. Introduction

As real-time systems such as automobiles become more complex and open architectures, they are vulnerable to many adversarial factors such as faults and attacks. With these adversaries, the controller may make dangerous decisions and cause serious consequences such as vehicle crashes and loss of human lives [1], [2]. Resilience to these adversarial factors is essential to the safety of such systems [3].

In this paper, we study the problem of tolerating transient faults for a controller running on computational nodes. In general, there are two popular research threads for fault resilience: redundancy and checkpointing. One thread relies on redundant components (e.g., standby processors [4] or task replica [5]), where if some components are faulty, other components can still process forward to finish the job. The other thread occasionally checkpoints system states, and the system rolls back to a consistent state (checkpointed in history) when detecting faults [6]. This work aligns with the second thread and studies checkpointing protocols for real-time parallel processes.

Existing checkpointing works can be divided into two groups. One group focuses on checkpointing computing tasks in general-purpose (non-real-time) systems. The goal is to guarantee the logical consistency of checkpoints (value correctness), which represents the cause-effect relation defined by messages sent and received among tasks [7]. The other targets real-time systems and carries out checkpointing under timing constraints (deadlines) [8]. There are three main protocols setting checkpoints: uncoordinated checkpointing, coordinated checkpointing, and communication-induced checkpointing(CIC). Uncoordinated checkpointing enables tasks to set checkpoints when convenient for a better schedule [9]. detecting [10]. However, these works are incapable of tackling checkpointing real-time parallel processes, where both logical and timing correctness need to be guaranteed.

To fill this gap, we propose a new three-step checkpointing protocol that considers both types of correctness. The first step is processes partition, which transfers real-time parallel processes into a directed acyclic graph (DAG) of tasks where the edge represents the message communicated between tasks. Compulsive checkpoints guaranteeing logical correctness are then placed for each task. The second step is to identify a critical path, which finds the longest execution path of the DAG. The last step is to ensure timing correctness, which minimizes overall execution time (task execution time plus checkpointing overhead). We propose an effective and efficient algorithm for each step.

The rest of this paper is organized as follows. Section II describes the overview of the optimal checkpointing strategy, the system model, and the threat model. Section III, Section IV, and Section V present task partition, finding the critical path, and placing optional checkpoints for timing correctness, respectively. Section VI evaluates our method, and Section VII concludes the paper and gives insights into our future work.

II. Preliminaries and design overview

In this section, we present the problem statement, our strategy overview, system model, and fault model.

A. Problem Statement

Consider a real-time system whose processes perform repetitive tasks, where each process aims at one specific function, for example, collecting sensor readings. Processes send messages to others during runtime, forming process dependencies. A checkpointing system is capable of resisting faults in real-time systems because of quick recoveries. When a fault is detected, the system rolls back to the nearest valid checkpoint that preserves the state of the process, avoiding redoing all the work from the beginning. The objective is to determine an optimal checkpointing strategy that achieves (i) logical consistency of checkpoints considering process dependencies and (ii) the shortest execution time considering checkpointing overhead and recovery time.

2022 IEEE Real-Time Systems Symposium (RTSS) | 978-1-6654-5346-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/RTSS55097.2022.00055

2576-3172/22/$31.00 ©2022 IEEE
DOI 10.1109/RTSS55097.2022.00055

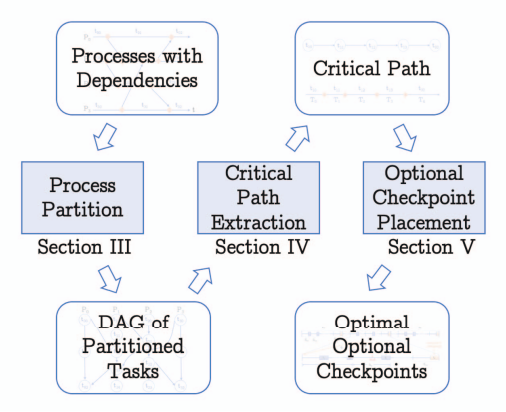


Fig. 1: The Overview of Optimal Checkpointing Strategy

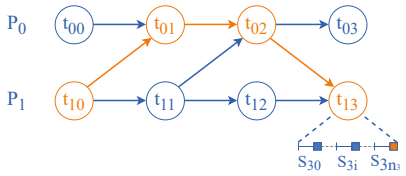


Fig. 2: Relationship of Process, Task, And Segment

B. Overview of the Checkpointing Strategy

We derive the optimal checkpointing strategy by three steps, as shown in Figure 1: (i) process partition, (ii) critical path extraction, and (iii) optional checkpoint placement. The following briefly describes these steps, and we will present their detailed design in Section III, IV, and V.

- *Process partition.* Processes are executed with dependencies, which are reflected by message passing. To tolerate faults and recover successfully, we need to place checkpoints considering these dependencies. This step is to partition dependent processes into a DAG graph and place compulsive checkpoints. These checkpoints meet the requirement of logical dependency.
- *Critical path extraction.* In the DAG, most tasks can be executed in parallel with the multiprocessor. However, tasks in a dependent path must be executed in sequence. This step is to identify a critical path, which is the longest dependent path. This critical path determines the performance the checkpointing and recovery. Note that the tasks in noncritical paths can do the same following step to set up checkpoints, but their execution time is less than that in the critical path. Therefore, in the proposed model, only the critical path should be considered.
- *Optional checkpoint placement.* Numerous works will be discarded in the critical path after a fault if we place too few checkpoints, and the checkpointing overhead will dominate if we place too many checkpoints. This step is to solve optimization problems to determine the optimal number and intervals of checkpoints.

TABLE I: Symbols and Notations Used in This Paper

Symbol	Description
P_i	i-th process in the system
T_i	i-th task in the critical path
S_{ij}	j-th segment in the Task i
s	the recovery overhead from the initial state
t_c	the overhead of checkpointing
q_i	the invalid rate of an optional checkpoints in the Task i
p_i	the valid rate of an optional checkpoints in the Task i
n_i	the number of optional checkpoints in the Task i
λ_i	the failure rate of the Task i
m_i	message i
w_{ij}	the total execution time before S_{ij} in the Task i
W_{ij}	the expectation of w_{ij}
d_{ij}	the completion time of S_{ij} before a fault
D_{ij}	the expectation of d_{ij}
F_{ij}	the probability that the fault occurs in S_{ij}
I_i	the fault-free computation time (excluding t_c) of Task i
τ_{ij}	the fault-free execution time (including t_c) of Segment S_{ij}

C. System Model

Symbols and notations used in this paper are listed in Table I. There are some **PROCESSes** (P_i is denoted as the i-th process) in real-time systems, and the passing messages between them form the dependencies of processes. We partition the processes into tasks (t_{ij} is denoted as the j-th task on the i-th process) according to the dependencies. In this way, a DAG is obtained through the partition, where tasks are nodes, and dependencies are edges. We place *compulsive checkpoints* on this DAG to guarantee logical consistency. Then, the critical path of the DAG is identified, and the **TASKs** in this path are renamed as T_i (i.e, the i-th task in critical path). Finally, we place *optional checkpoints* according to the strategy to achieve a shorter execution time. The optional checkpoints split each task into some **SEGMENTs** (S_{ij} is the j-th segment in T_i). The relationship among PROCESSes, TASKs, and SEGMENTs are shown in Figure 2. Two PROCESSes (P_0 and P_1) have four and three tasks separately. The critical path marked in orange has four tasks (t_{10}, t_{01}, t_{02}, and t_{12}) that are renamed to T_0, T_1, T_2, and T_3. The TASK T_3 can be split to n_3+1 SEGMENTs from S_{30} to S_{3n_3}. We assume that the system stores all compulsive checkpoints in the current critical path but only stores the latest optimal checkpoint because of limited resources.

D. Fault Model

Fault occurrence is usually regarded as random and independent, even though our method is not confined to any specific distribution, we assume, for brevity, that the arrival of faults is a Poisson Process with a failure rate of λ. For generality, we set a different failure rate for each task in the critical path, and λ_i denotes the failure rate of the i-th task. Then, its probability density function (PDF) is $\lambda_i e^{-\lambda_i t}, t \geq 0, \lambda \geq 0$.

When a fault arrives, there is a p_i chance for task T_i rollback to the latest checkpoint. However, there is also a $q_i = 1 - p_i$ chance for a task to roll back to the latest compulsive checkpoint because of the optional checkpoint availability.

III. Process Partition with Logical Consistency

Processes in real-time systems perform some recurrent tasks, between which sending and receiving messages form the dependencies. When a transient fault occurs, the system needs to recover back to normal. We backup some checkpoints so that the systems can re-execute from these states to tolerant the fault.

Definition 1 (Logical Consistency): Logical consistency is defined as a state, in which a sender process reflects sending a message once the corresponding receiver process indicates the message reception.

In this step, our checkpoints should meet the logical consistency requirements (Definition 1) to ensure that the recovery process goes smoothly.

We split processes into several tasks where they send or receive messages and add edges between neighbouring tasks within the same processes. Next, we add edges from a task sending a message to another task receiving the message. In this partition graph, the weight of vertices is the execution time of the tasks, and the edges represent the dependencies of these tasks. Then, we add a checkpoint overhead to the weight of each vertex. When a process sends a message, the process places a compulsive checkpoint following it, reflecting the message sent to guarantee logical consistency. When a process receives a message, the process places a compulsive checkpoint backup all work before the message.

IV. Critical Path Extraction

Definition 2 (Critical Path): In a DAG of tasks, the critical path is a path in which the total sum of the weight of vertices is no less than that of any other paths.

If there are several processors in a system, most of tasks can be executed in parallel. However, tasks connected by edge in the DAG must be performed in sequence. If we find a critical path (Definition 2) in this DAG, we get a maximum length of tasks that can only be executed in sequence. This critical path determines the total execution time of these processes.

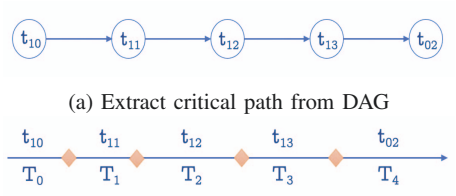
Definition 3 (Timing Correctness): Timing Correctness means all tasks in each process catch up with the deadline of this process in a real-time system.

First, we topologically sort the DAG and get an ordered vertex set V with a complexity of $O(|V|)$. Then, we use dynamic programming to calculate the maximum total weight $tw(v)$ ending with vertex v. Finally, we select the maximum value of $tw(v)$ as the largest total weight and reconstruct the critical path P using each optimal choice. We will expose the algorithm details in our future work.

The critical path is the worst case for the following analysis because other tasks can be executed in parallel. Figure 3a shows a critical path partitioned from processes, in which we already place some compulsive checkpoints (shown in Figure 3b) for logical consistency.

V. Checkpoint Placement for Timing Correctness

Placing checkpoints is non-trivial in the design due to the timing correctness. On the one hand, if we do not place



(a) Extract critical path from DAG

(b) Compulsive checkpoints

Fig. 3: Examples of The Critical Path

optional checkpoints, the conservative long-rollbacks would considerably waste previous work. On the other hand, placing too many checkpoints causes negligible overhead. Thus, we need to find a tradeoff to place an appropriate number of checkpoints with proper intervals. In each task, the problem is formulated into an optimization problem. We use an optimization solver to optimize and we will provide an efficient and robust optimization method in our future work.

For tasks after T_0, the task rolls back to a compulsive or optional checkpoint with the overhead r, if a fault is detected. There is no restart overhead because there is a compulsive checkpoint at the beginning of the task T_i. Therefore, the total execution time for S_{ij} is

$$w_{i(j+1)} = \begin{cases} h_{i0} & j = 0 \\ w_{ij} + h_{ij} & 1 \le j \le m_i \end{cases} \tag{1}$$

where

$$h_{ij} = \begin{cases} \tau_{ij} & P = 1 - F_{ij}(\tau_{ij}) \\ d_{ij} + r + h_{ij} & P = F_{ij}(\tau_{ij}) p_i \\ d_{ij} + r + w_{i(j+1)} & P = F_{ij}(\tau_{ij}) q_i \end{cases} \tag{2}$$

From Eq. (1), (2), and the fault model, we can derive the expectation of $w_{i(j+1)}$:

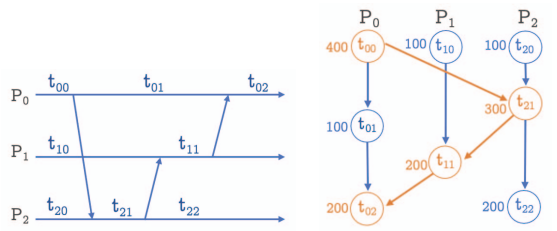
$$\begin{aligned} W_{i(j+1)} &= \frac{1 - p_i F_{ij}(\tau_{ij})}{1 - F_{ij}(\tau_{ij})} W_{ij} + \tau_{ij} + \frac{F_{ij}(\tau_{ij})}{1 - F_{ij}(\tau_{ij})}(D_{ij} + r) \\ &= (q_i e^{\lambda \tau_{ij}} + p_i) W_{ij} + (e^{\lambda \tau_{ij}} - 1)(\frac{1}{\lambda_i} + r) \\ &= v_{ij} W_{ij} + u_{ij} c_i \end{aligned} \tag{3}$$

where v, u, and c are substitutions. We can derive the expectation of the total execution time of the task T_i:

$$\begin{aligned} W_i &= W_{i(m_i+1)} \\ &= c_i u_{i0} \prod_{j=1}^{m_i} v_{ij} + c_i u_{i1} \prod_{j=2}^{m_i} v_{ij} + c_i u_{i2} \prod_{j=3}^{m_i} v_{ij} + \dots \\ &\quad + c_i u_{i(m_i-1)} v_{im_i} + c_i u_{im_i} \end{aligned} \tag{4}$$

The optimization problem is expressed as:

$$\begin{aligned} \arg\min_{m_i, \tau_{ij}} \quad & W_i \\ \text{s.t.} \quad & \sum_{j=0}^{m_i} \tau_{ij} = I_i + (m_i + 1) t_c \end{aligned} \tag{5}$$



(a) Simulation processes (b) DAG of tasks

Fig. 4: Simulation Setting

Note that, the first task T_0 in the critical path starts from the initial state, while other tasks $T_1, \ldots, T_n - 1$ starts from a compulsive checkpoint (as shown in Figure 3b). Thus, the first task needs a special consideration, and we will illustrate it in future work.

VI. Evaluation

In this section, we evaluate our optional checkpointing strategy on a randomly generated system with dependent processes as a proof of concept. In the future, we will evaluate the proposed method from more aspects.

The generated processes with dependencies are show in Figure 4a. According to Section III, we partition these processes into a DAG of tasks, shown in Figure 4b. The fault-free execution times of each task are marked beside the vertices. According to the Section IV, we extract the critical path of the DAG, and mark it in orange. The critical path contains four tasks with time-length 400, 300, 200, and 200. The deadline to complete all tasks is 3300, which is equal to 3 times the fault-free computation time of tasks on the critical path.

We compare our checkpoint placement strategy on the critical path with four different strategies, same on the number of checkpoints but differ on the checkpoint interval. They are: (a) optimal placement strategy obtained from Section V; (b) uniform distribution placement strategy: place checkpoints based on uniform distribution; (c) Gauss distribution placement strategy: place checkpoints based on Gauss distribution with $I/2$ as mean and $I/4$ as the standard deviation ; (d) narrowing placement strategy: gradually narrow the interval between two checkpoints; (e) widening placement strategy: gradually widen the interval between two checkpoints. The placement strategy (d) is based on the algorithm: the $i+1$-th checkpoint in a task is placed at the first third of the interval between the i-th checkpoint and the end. The placement strategy (e) is the reverse process of strategy (d). We simulate the critical path process 100000 times and list the result in Table II.

TABLE II: The Result of Simulation 2 - Performance Regarding Different Checkpoint Interval. Strategies: (a) Optimal, (b) Uniform, (c) Gauss, (d) Narrowing, (e) Widening.

Strategy	Avg Exec	Min Exec	Max Exec	%Deadline
Optimal	**2616.12**	1252.00	11995.20	**81.88**
Uniform	2744.79	1236.00	11732.31	77.80
Gauss	2732.77	1254.44	10993.74	78.19
Narrowing	2946.51	1250.99	14325.78	70.97
Widening	2941.82	1265.67	13142.09	71.02

The result shows that our model optimizes the interval between checkpoints. We notice that our optimal strategy has the shortest average execution time and the highest percentage of finishing the process on time. Note, the maximum execution time of strategy (c) being less than other strategies is due to randomness. Our model cannot guarantee to perform the best every time, but it promises a better average result when running time accumulates.

VII. Conclusion And Future Work

The main contribution of the paper is the consideration of both logical consistency and timing correctness during the checkpoint placement in real-time systems. We first partition processes with complex dependencies into a DAG, during which we place some compulsive checkpoints to guarantee the logical consistency and avoid much useful work waste. Then we extract the longest critical path to analyze timing correctness. Finally, we build a model to illustrate how to minimize each task's execution time in the critical path to achieve a minimum total execution time. Given that this work is still on-going, we will improve the algorithms used in the process partition and critical path extraction steps and optimize the optional checkpoint placement step in our future work. Furthermore, we will evaluate the proposed method from more aspects by extensive simulations and case studies.

Acknowledgement

This research was supported in part by NSF CNS-2143256. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

References

[1] M. Wolf and D. Serpanos, "Safety and security in cyber-physical systems and internet-of-things systems," *Proceedings of the IEEE*, 2017.

[2] H. He and J. Yan, "Cyber-physical attacks and defences in the smart grid: a survey," *IET Cyber-Physical Systems: Theory Applications*, 2016.

[3] F. Flammini, "Resilience of cyber-physical systems," *Springer*, 2019.

[4] Y. Guo, D. Zhu, and H. Aydin, "Generalized standby-sparing techniques for energy-efficient fault tolerance in multiprocessor real-time systems," in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2013, pp. 62–71.

[5] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 813–825, 2016.

[6] E. N. Elnozahy and J. S. Plank, "Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 97–108, 2004.

[7] J. C. Ho, C.-L. Wang, and F. C. Lau, "Scalable group-based checkpoint/restart for large-scale message-passing systems," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–12.

[8] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," in *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[9] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello, "Uncoordinated checkpointing without domino effect for send-deterministic mpi applications," in *2011 IEEE International Parallel Distributed Processing Symposium*, 2011, pp. 989–1000.

[10] D. M. Yi Luo, "Theoretical and experimental evaluation of communication-induced checkpointing protocols in fe and flazy-e families, performance evaluation," *Performance Evaluation*, 2013.