EE5903 Real-Time System

Assignment 2

Fault-tolerant strategy for RT scheduling

Liu Weihao (A0232935A)

Mobile: 80320267

1. Introduction

In this report, I simulated the Real-Time scheduling algorithms on a quad-core processor. I select DFTS (Dynamic fault-tolerant scheduling) and CH (checkpointing with roll back recovery) form the attached paper to compared. I assume that there is an extra scheduler monitor the status of processor and decide to allocate the task to which specific processor core.

There are some transient faults occurring during the task execution. After processor completed a specific task, it will detect whether the fault has occurred. If the faults are detected, the task should be re-executed or rescheduling which is dependent on the algorithm. For the CH methods, the task will roll back and recover from the last saved check point. Besides, CH provides an optimal checkpoints selection algorithm to minimize the checkpointing overhead so that the exception of the execution time is minimized.

DFTS algorithms integrate CH algorithm and RP (Hardware replication) algorithm. According to the parameters of each task, the tasks is classified into two categories, critical and noncritical. For those noncritical tasks, DFTS apply RP to deal with them. Other tasks are executed by CH algorithm.

The main different between the two algorithm is why we need to classify the tasks and how to do that. For CH method, no matter when the tasks' deadlines are coming, it always uses only one core to execute this task and creates many checkpoints. But saving checkpoint will cost some time which may cause the task missing its deadline. In DFTS, the use the utilization of the task to determine its emergency. For those critical tasks, if we apply CH, the deadline maybe missed. So, we will give those tasks two cores to execute the same task without any checkpoint. If one of them complete the task without any fault, we think the task is completed. For those noncritical tasks, we think even if the worst-case happened, the CH can complete this task before the deadline.

2. System Model

In order to evaluate the two algorithms mentioned in section 1, I generate a set of tasks which consists of some independent tasks. These tasks cannot be interrupted during their execution. Every task has four parameters, their module is given as:

$$\tau_i = (C_i, D_i, T_i, R_i)$$

Where:

C_i: Worst-case execution time in a fault-free condition
D_i: Relative deadline
T_i: Inter-arrival time

R_i Release time of the task

The definition of the utilization of the task in DFTS is the division of C_i by D_i,

$$U_i = \frac{C_i}{D_i} \tag{2.1}$$

When U_i close to "1", there's nearly not enough time to execute this task. So, these tasks are marked as "critical task", scheduler has to give these tasks high priority and more computing resources. To decide on which task is critical or not, a threshold θ is used.

$$Class(\tau_i) = \begin{cases} Noncritical, & U_i < \theta \\ Critical, & U_i \geq \theta \end{cases} \tag{2.2}$$

θ will be computed when scheduler want to schedule the task.

2.1 Fault model

The fault occurrence probability is a function of time. In the paper, they use Poisson Process to estimate the fault. The fault arrival rate is λ, which is in the range of 10^{-2} to 10^{2} per hour [1]. Besides, every fault is independent to other faults. Assume that the interarrival time between the $(i-1)^{th}$ and i^{th} fault is X_i. Its PMF (Probability Mass Function) is given by:

$$f_{X_i}(x) = \lambda e^{-\lambda x} \tag{2.3}$$

Where t is the total task execution time. Only when there's no faults occurred during the processor deal with the task, the task is executed successfully. Only when $X_1 > t$, the task is successful. Therefore, $\Pr\{X_1 > t\}$ is the successful probability.

$$\Pr\{X_1 > t\} = \int_t^{\infty} \lambda e^{-\lambda x} dx = -e^{-\lambda x}\Big|_t^{\infty} = e^{-\lambda t} \tag{2.4}$$

In this report, I use k-fault-tolerant model [2] which represents the maximum number of faults could be tolerated for each task. According to the paper, the value of k should be a small multiple of λt, i.e., $2\lambda t \leq k \leq 3\lambda t$.

Besides, we assume that all the faults are independent and only impact a specific task, i.e., four core execute the task at the same time, a transient fault occur on a specific core, it won't impact other cores. Besides, if we apply checkpoint algorithms, the task could recover from the last check point.

2.2 Fault detection

There are many faults detection methods mentioned in [1], such as CRC codes, control-flow

error detection, etc. Anyway, I didn't simulate the fault detection algorithm. In this report, I assume that all faults can be detected at the end of the task execution. Moreover, the latency of the fault detection process is assumed to be zero. If the fault occurred, the processor will re-execute this task from last checkpoint.

3. Task scheduling algorithm

As mentioned previously, there are three scheduling algorithms in this report: 1) CH, 2) RP, 3) DFTS. Where DFTS is the combination of CH and RP. In this part, I will introduce these algorithms separately.

3.1 Checkpointing optimization (CH)

There are trade-offs between frequency of the checkpoints and the overhead of the task. As equation (2.4) is showed, frequent checkpointing will decrees the length of the time slot which could increase the successful probability for every sub task, but will increase the overhead. On the other hand, infrequent checkpointing has low successful probability, but could decrease the time overhead. Assume that the time of saving/getting a checkpoint (Cs) and recovery from a checkpoint (Cr) are the execution time C_i multiple a constant parameter. We have:

$$\begin{aligned} Cs &= \varphi \times C_i \\ Cr &= \mu \times C_i \end{aligned} \tag{3.1}$$

Assume that the optimal number of checkpoints is m, the time interval can be expressed as: $\Delta = C_i/(m+1)$. Considering there are worst-case k faults occur during the task execution. m is given by:

$$m = \left\| \sqrt{\frac{k \times C_i}{Cs}} - 1 \right\| = \left\| \sqrt{\frac{k}{\varphi}} - 1 \right\| \tag{3.2}$$

Where $2\lambda C_i \le k \le 3\lambda C_i$, in this report, I assume $k = 3\lambda C_i$. Considering the are n faults occur during the task execution, there will be n sub task recovery from checkpoint and re-executed. The time Cc is given by:

$$Cc = (C + m \times Cs) + n \times (Cs + Cr) + \frac{n \times C}{m+1} \tag{3.3}$$

where the term $(C + m \times Cs)$ is the total execution time using the CH without any fault. $(Cs + Cr)$ is the cost of single fault rollback to last check point. $C/(m+1)$ is the cost of re-execute a sub task.

3.2 Hardware replication (RP)

Although CH method could reduce the total overhead by re-execution only on part of the

task in the presence of faults, it cannot use other free processing cores. Besides, saving checkpoint will cost some time which may cause the task missing its deadline.

In RP algorithm, we use two idle cores to execute the same task without saving any checkpoints. As long as one of them is executed without any faults, the task will be completed. Since in dynamic scheduling policy, there may be not available to schedule the task on two idle cores. Hence, the second replica will be executed until there is another processing core becomes idle

3.3 Dynamic fault-tolerant scheduling (DFTS)

This method is a hybrid method combine the CH and RP. The scheduler category the task into two set based on their utilization (U_i) and the threshold (θ). For those non-critical tasks, there are enough time to execute them. So, scheduler will apply CH algorithm to minimize the total overhead and increase the hardware utilization.

For those critical tasks, there is not enough time to re-execute the tasks or even if rollback to the last checkpoint. CH method may miss the deadline. In this case, scheduler will apply RP method to allocate more hardware resources to these tasks, so that they can completed before the deadline.

U_i is defined in Eq. (2.1). The threshold θ should be computed before we schedule this task. According to [1], the threshold is given by:

$$\theta = \frac{1 - \frac{Delay_i}{D_i}}{1 + m \times \varphi + k(\varphi + \mu) + \frac{k}{m+1}} \tag{3.4}$$

where $Delay_i$ is the scheduling delay which indicates the wasted time between the time a task joins the ReadyList, and it applies our fault-tolerance policies

4. Implementation of the algorithm

Although [1] already provide its algorithm flows, when I try to follow their algorithm flows, I cannot get the similar results. I guess that's because it doesn't conclude the fault model. For the CH methods, they always use Cc as the execution time without any extra fault.

Besides, if the task satisfies the schedule policy and the probability of its execution success is positive, the task will be marker as "Feasible ".

In my implementation code, the processor core will compute $\Pr\{X_1 > t\}$ once it received the offloading command based on Eq. (2.4). After it complete this task, it will generate a random number rv within $[0,1]$. If $rv > \Pr\{X_1 > t\}$, we think there is at least one fault

occurred during the task execution. Otherwise, the task will be marked as "executed successfully ".

The dataset is same as [1] mentioned. I generate 50 different tasks, all of the parameters are random and uniform distribution. The specific range is in table 1.

Table 1. Range of the parameters

	Definition	Range
C_i	Execution time	10~100 ms
D_i	Deadline	$2\boldsymbol{C_i}$~$3C_i$
R_i	Release time	0~5500 ms

Besides, the task interval is equal to its deadline, $T_i = D_i$. The 50 tasks will release 10 times after the first one is released.

For those critical tasks, my DFTS scheduler will allocate two idle cores to them

5. Results and comparisons

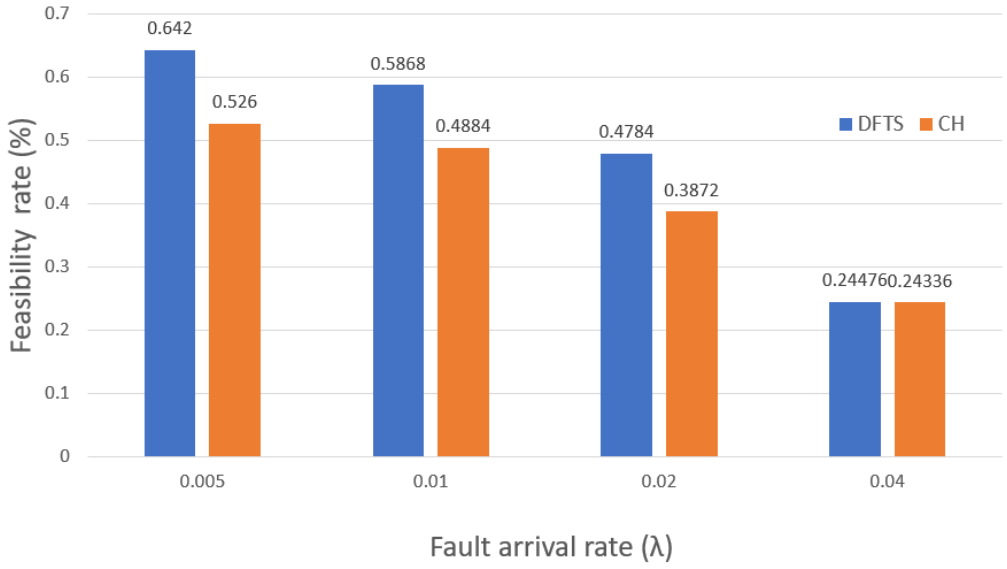

Fig 1. Completion ratio vs fault arrival rate

When I fix the parameters $\mu = 0.05$ and $\varphi = 0.05$, only change the fault arrival rate, the result is as Fig 1. Shown. When λ is smaller than 0.02, DFTS could get a higher completion ratio, about 12% higher than CH. But when $\lambda = 0.04$, their completion ratio is nearly same. That's because the parameter k in Eq. (3.4) is $3\lambda t$. When λ is very high, the threshold will very small which will result nearly every task become critical tasks and apply RP method. But because $\Pr\{X_1 > 0\}$ also close to 0. Use DFTS to complete the whole task without any checkpoint is impossible. Besides, DFTS need to use much more hardware resources which

will impact other tasks cannot be executed before their deadline.

We can also change φ and μ to see the impact of checkpoint configuration.

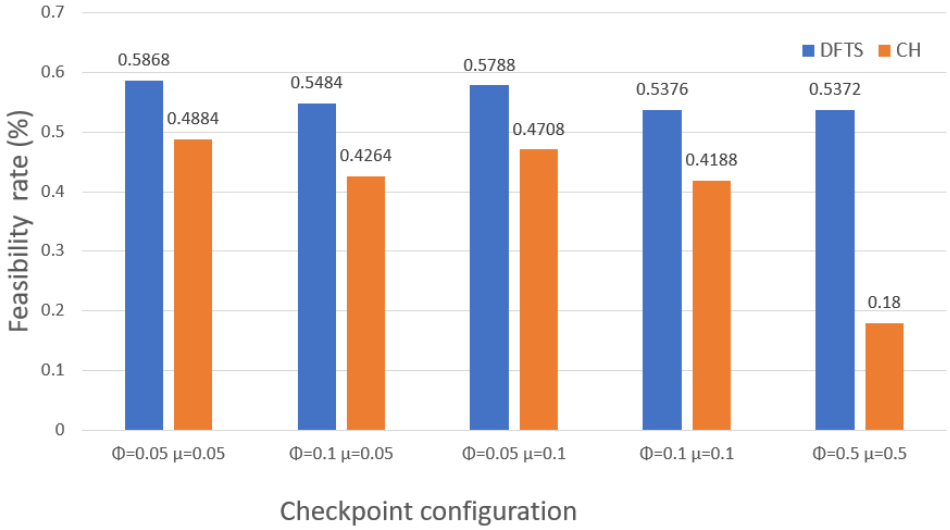

Fig 2. Completion ratio vs checkpoint configuration

In Fig 2. the fault arrival rate is a constant $\lambda = 0.01$. Because DFTS use CH and RP at the same time, changing φ and μ don't affect DFTS significantly. But if we doubling the checkpoint saving cost and the recovering cost the completion ratio will decrease significantly.

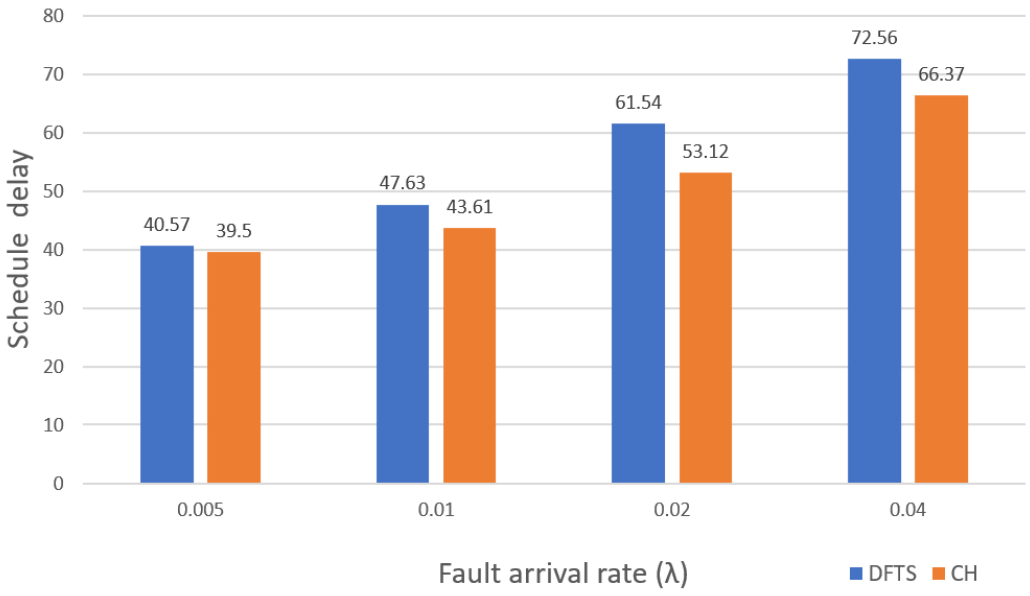

Fig 3. Scheduling delay vs fault arrival rate

Fig 3. shows the average task scheduling delay. The delay of CH method is always less than DFTS. That's because CH only allocate one processor core to a task. But DFTS would like to allocate 2 idle cores to the critical tasks which will result in longer delay

6. Conclusion

In this project, I simulated two real-time scheduling algorithm DFTS and CH. And compared their performance.

In terms of task completion ratio, DFTS is always better than CH because it applies CH and RP methods at the same time. DFTS could solve some problems that CH can not deal with. When the fault arrival rate and the checkpoint cost increase, the completion ratio will decrease significantly.

In terms of task scheduling delay, CH's performance is always better, because DFTS uses too many hardware resources on the critical tasks.

Anyway, time delay is always better than task execution failure. So, DFTS is the better choice.

Reference

1. Mottaghi, Mohammad H., and Hamid R. Zarandi. "DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors." *Microprocessors and Microsystems* 38.1 (2014): 88-97.

2. Zhang, Ying, and Krishnendu Chakrabarty. "A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.1 (2005): 111-125.

Coding Effort

I completed the 100% coding work without any reference codes.

The entire program is based on the Fig 4. in the paper [1] and some of its algorithm description.

I generate the dataset using the program *task_generate.py* which is attached. The parameters value is described is the report table 1. The paper mentioned that $10 < C_i < 100$. And I selected other parameters range randomly.

All the programs are written in python. The packets used in the program is as follows:

simpy
numpy
pandas

Just use *pip install* to install these packets