International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 3, Number 10 (2013), pp. 993-1000
© International Research Publications House
http://www. irphouse.com /ijict.htm

Fault Tolerance Approach with Energy Harvesting in Real Time System

Arvind Kumar and Bashir Alam

Department of Computer Engineering
Jamia Millia Islamia, New Delhi

Abstract

Real time systems are widely used systems in now days. Most of these systems work in fault prone environment. The job running in these systems should be finished on its deadline as timing constraints which requires some energy to finishing it. If there is any fault occurring in the system then it may not be possible that the job running will finish on its deadline and will give the desired result. Fault tolerance is the technique to give the required output in the presence of fault which can be achieved by different methods. Fault tolerance can be achieved by checkpointing to overcome the problem of total reexecution. In this paper we will propose the checkpointing scheme for energy harvesting real time system in which task running in such a way that it will finish on time with minimum energy consumption. The energy is harvesting from different resources which can be managed through dynamic voltage scaling (DVS).

Keywords: Real Time System, Fault tolerance system, DVS (Dynamic Voltage Scaling).

1. Introduction

All real-time systems share a vision of small, inexpensive and battery-operated networked processing devices. All the real time systems have timing as a constraint, beside that these systems usually has serious energy limitations. For example, the limited life time of a device battery is an energy constraint which decreases on use of it. Energy efficiency is crucial to many real-time systems due to their limited energy supply and severe thermal constraints of the operating environment. Among the power-management techniques proposed to tackle these challenges, Dynamic Voltage and Scaling (DVS) has emerged as the most popular and widely deployed scheme to manage stored energy as well as harvested energy [1][2]. Energy can be harvest from different resources and it is used when stored energy is not efficient for operating the resources. To increase the battery life-time of such devices, energy harvesting techniques are used. Such techniques can help to relax the energy constraints dictated by the initial limitations of the battery capacity. The energy is being harvest on the time when the battery is working and decreasing its capacity and it can be managed by DVS. Many real-time systems work in harsh and failure prone environments, they suffer from possible security attacks as well as some other kind of damages which can cause failure in the system. Such problems can have negative effect on the real-time service availability by causing the service deadline to be missed. Due to any attack or damage, missing the deadline causes a system failure. To avoid such inconveniences and due to the fact that total reexecution of the affected services is not applicable in many real-time services with relatively short deadlines, we can use of checkpointing methods. Using checkpoints, total reexecution is replaced with partial rollback and recovery [3]. DVS techniques used for power management without considering fault tolerance is explained in [4]-[7]. Checkpoint placement strategies for fault tolerance without power management were described [8], [9]. Recently, attempts have been made to combine fault tolerance with power management [10], [11].

In this paper, we use a task's initial slack to tolerate one transient fault that may occur during the task life-time in an energy harvesting real-time system. We use of time redundancy to tolerate such faults and more specifically checkpointing to avoid total task rollback for the sake of the task's time constraints. Checkpointing is basically used for rollback from previous checkpoint where acceptance test is applied. In this regards, according to some information, the optimal number of checkpoints and their best placing are determined. Number of checkpoints is optimal due to timing constraint as deadline. Such information includes the remaining slack, the current status of the energy storage, and the predicted harvesting energy before the task's deadline.

We have proposed offline and online method, in offline method the initial slack as well as the predicted harvesting energy is used to statically determine the optimal number of checkpoints and the respective processor speed to optimize the energy usage of the system. In this regards, the checkpoints are placed uniformly during the task's life-time. However, due to the unpredictability of energy harvesting methods, it may be required to adapt the number and places of checkpoints as the task advances. Therefore, we have also proposed an online method that dynamically adjusts the number of checkpoints, their placement, and the respective processor speed (using DVS). As the task progresses, this method recalculates the above parameters if the harvested energy deviates from the expected energy. Thus, the checkpoints are placed non-uniformly if this latter method is applied to the system.

2. Harvesting and Stored Energy

The system includes energy harvesting and storage modules. Let P_h (t) denote the power generated by the energy source module at time instant t, and E_h (t1, t2) denote the harvested energy during time interval [t1, t2], the harvested energy fed into energy storage module can be calculated as

$$E_h(t1, t2) = \int_{t1}^{t2} P_h(t)\, dt.$$

The power of the energy source is variable at different times. Therefore, it is needed to have curves that bound the energy source power. This has lower bound E^L (t2-t1) and upper bound E^U (t2-t1) respectively then we can say

$$E^L(t2-t1) \leq E_h(t1, t2) \leq E^U(t2-t1)$$

The system has energy storage with stored energy Ec(t) and maximum capacity C. A task can drain power P_d (t) and respective energy E_d (t1, t2) from the energy storage. We can say drain energy will be as

$$E_d(t1, t2) \leq Ec(t) + E_h(t1, t2)$$

The later inequality indicates that a processing element has to stop the execution of a task when the available energy is not enough to finish the task execution. The energy harvesting model can be explained as

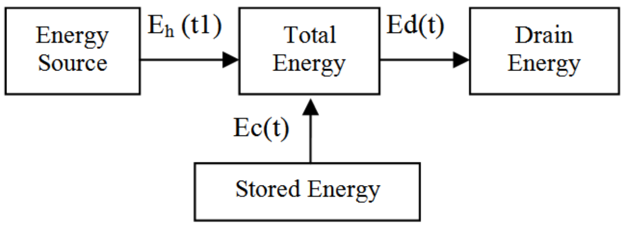

Figure 1: Harvesting and stored energy.

3. Task Model

In real-time system, a task can be characterized as a T(*D*, *C*), where *D* is deadline and *C* is worst case execution time at maximum CPU speed. Here we will only consider one task in the task set, and start time of the task can be considered at 0 and *D* can be assumed as the time interval that task can be executed. For the sake of simplicity, the maximum speed S_{max} of the CPU assumed to be 1. So, it is needed to normalize the units of C i.e. if the maximum CPU speed is *S*, then *C* can be expressed in units of S cycles.

4. Processor Model

There are two term in processing a task, busy time and ideal time. If a processing element executes tasks at a certain period of time, the period of time is the busy time of the processing element. During the period of time, the processing element is also called busy or being used. If a processing element is idle or does not execute tasks in a certain period of time, the period of time is called the idle time of the processing element.

Here we are considering only single task is running on uniprocessor.

5. Fault and Recovery Model

In any real time system fault can be of three types, transient, intermittent, and permanent. We assume that at most one specific transient fault, such as any security attacks or environmental noises, may occur during a task's life-time, between its release time and deadline. Such faults can be tolerated by total re-execution of the damaged portion of the task, by time-redundancy technique for fault tolerance. However, since the task is real-time, we should avoid of complete task rollback. Therefore, we use of checkpointing techniques [9]. Within a checkpoint, a consistent state of the system is saved onto a stable storage. Moreover, before saving the task state at the instant of checkpointing, an acceptance test is run to discover the possible occurrence of transient faults. Therefore, if a fault occurs, its detection is postponed until the next checkpointing interval. If a fault occurrence is detected, rollback to the last checkpoint is done and the task's remaining portion is executed by the maximum processor speed. In this regards, the target is to avoid the service unavailability by preventing the deadline of a task to be missed. It is necessary to choose the number of checkpoints such that checkpointing and energy management overhead together be less than or equal to the available slack.

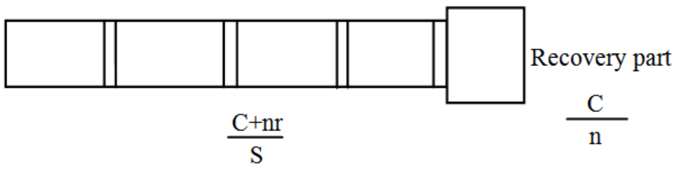

Figure 2: Execution of task with n check point with speed S and recovery part with speed 1.

6. Optimal Checkpointing

Now the main goal is to find optimal number of checkpoint placed to efficiently use the energy by scaling the processor speed up or down while tolerating at most one fault without missing deadline.

The work in [11] proposed a solution to finding the optimal number of checkpoints, n, to conserving energy and tolerating one fault without considering the likely energy that may be used in recovery part. That paper derived the value of n which minimizes energy by differentiating energy equation with respect to n and equating the result to zero. We calculate optimal checkpoint placement without considering available energy unpredictability. This unpredictability in future incoming energy may causes that in some time interval the system will be run out of energy and can't execute the task with optimal speed. If this condition results in a deadline miss, all the energy consumed so far to execute some part of task would be wasted. Therefore, our focus is managing the available energy under two checkpointing method so that least amount of energy would be wasted.

7. Off line Checkpointing

Under offline checkpointing Method the optimal number of checkpoints is calculated only once at the beginning of the task and it won't change until end of task execution. Therefore, if in each checkpointing interval system is run out of energy, task execution will be terminated and all the energy used so far will be wasted. This method checks the available energy at the beginning and if the system has enough energy, starts executing the task, and otherwise, ignores it. More ever, if a fault occurs during each checkpointing interval, we don't need to take checkpoint afterwards and optimal speed must be calculated again.

8. Proposed Model

Assuming that task should be executed in a device that harvests energy, in some time intervals, it may be out of energy to continue task with desired speed. Under online method, optimal number of checkpoints and execution speed are computed after each checkpointing interval according to remaining execution time and slack and available energy in the device. For example, in some intervals harvested energy is low and it is necessary to execute the task with reduced speed. Also it is possible that in some intervals energy is more than optimal energy and we can compensate the wasted time in previous case.

To illustrate online method, let introduce some parameter:

t_r : Recalculation time of n_{opt}

t_{int} :Time interval between two consecutive checkpoints when task should take n checkpoint.

$$t_{int} = \frac{D - t_r - \frac{C}{n}}{n}$$

S_{min} : Minimum supply voltage necessary to keep the system functional, so, below inequality must be hold:

$$0 \le S_{min} \le S_{max} = 1$$

S_e: Maximum speed that task can be executed with this speed in the current computed checkpointing interval, according to available energy in the device.

So, if we consider energy constraints, these two conditions for execution speed must be hold:

1. $S \leq S_e$
2. $S_{min} \leq S$

Considering these constraints, 6 cases can be considered.

Case 1: $S_{min} < S_e < S_{opt}$
In this case, S_e is less than S_{opt} and the required energy for continue the execution of task with speed S_{opt}, is not available. So, it is necessary to reduce the speed in this interval. The best case for number of checkpoints due to energy saving is n_{opt} and if we change it, more energy will be consumed. On the other hand, we must reduce the speed in this interval to S_e. Hence, we lose some time and the probability of missing the deadline will be increased.

Case 2: $S_{min} < S_{opt} < S_e$
In this case two mentioned conditions are satisfied and device energy storage has enough energy for execute the task with S_{opt} in this interval.

Case 3: $S_{opt} < S_{min} < S_e$
In this case we have enough energy to execute the task with optimal speed, but the problem is that S_{opt} is less than $S_{min.}$ The system can't work with the speed less than S_{min} and we must continue with S_{min} instead of S_{opt}. Because the speed is more that S_{opt}, the number of checkpoints must be calculated again to use the extra slack that is available due to reducing the speed.

Case 4: $S_e < S_{min} < S_{opt}$
In this case the available energy is not enough to work With S_{opt} or even with S_{min} We try to find a time interval that system can work with the speed S_c that is smaller than S_{opt} and greater than $S_{min.}$

$S_{min} < S_c < S_{opt}$

Case 5: $S_e < S_{opt} < S_{min}$
In this case the energy for task execution with speed S_{opt} and S_{min} is not available. But because S_{opt} is less than S_{min}, unlike case 4, we must find a time interval that system can work with S_{min}.

Case 6: $S_{opt} < S_e < S_{min}$
It is similar to case 4 with this difference that the below condition must be hold for computed speed:

$S_{min} < S_c < S_e$

9. Conclusion

In this paper we propose a model for energy harvesting real-time system that can be affected by transient faults due to kinds of security attacks. Our goal is to find the efficient number of checkpoints which are optimal to avoid of missing the task's deadline when the occurrence of one transient fault is probable, while efficiently using the harvested energy. Therefore, we proposed two checkpointing methods, namely online and offline methods. The offline method inserts uniformly distributed checkpoints in a manner that minimizes the energy consumption according to the predicted harvesting energy. On the other hand, the online method non-uniformly distributes the checkpoints according to the remaining slack, available energy in the energy storage and predicted incoming energy and also uses DVS to save as energy as possible.

References:

[1] V. Gutnik and A. Chandrakasan, "An efficient controller for variable supply-voltage low power processing", *Symposium on VLSI Circuits*, pp. 158-159, 1996.

[2] W. Namgoong, M. Yu, and T. Meng, "A high-efficiency variable-voltage cmos dynamic dc-dc switching regulator", *IEEE International Solid-State Circuits Conference,* pp. 380-381, 1997.

[3] Maryam Dehghan, Mehdi Kargahi "Adaptive Checkpoint Placement in Energy Harvesting Real-Time Systems" Proceedings of ICEE 2010, May 11-13, 2010.

[4] Y. Shin and K. Choi, ''Power conscious fixed priority scheduling for hard real-time systems,'' in *Proc. Design Automation Conf.*, New Orleans, LA, 1999, pp. 134--139.

[5] G. Qu ''What is the limit of energy saving by dynamic voltage scaling?'' in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, 2001, pp. 560--563.

[6] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, ''Energy efficient mapping and scheduling for DVS enabled distributed embedded systems," in *Proc. Design, Automation and Test Europe Conf.*, Paris, France, 2002, pp. 514–521.

[7] R. Jejurikar and R. Gupta, "Energy aware task scheduling with task synchronization for embedded real time systems," in *Proc. Int. Conf. Compilers, Architecture and Synthesis Embedded Systems,* renoble, France, 2002, pp. 164–169.

[8] K. Shin, T. Lin, and Y. Lee, "Optimal checkpointing of realtime tasks," *IEEE Trans. Comput.*, vol. 36, no. 11, pp. 1328–1341, Nov. 1987.

[9] A. Ziv and J. Bruck, "An on-line method for checkpoint placement," *IEEE Trans. Comput.*, vol. 46, no. 9, pp. 976–985, Sep. 1997.

[10] Y. Zhang and K. Chakrabarty, “Energy-aware adaptive checkpointing in embedded real-time systems,” in Proc. Design, Automation and Test Europe Conf., Munich, Germany, 2003, pp.918–923.

[11] R. Melhem, D. Mosse, and E. N. Elnozahy, “The interplay of power management and fault recovery in real-time systems,” IEEE Trans. Comput., vol. 53, no. 2, pp. 217–231, Feb. 2004.