Performance Optimization for Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems

Zhongwen Li,          Hong Chen,          Shui Yu
Information Science and Technology College, Xiamen University, China
lizw@xmu.edu.cn

Abstract

Using additional store-checkpoints (SCPs) and compare-checkpoints (CCPs), we present an adaptive checkpointing for double modular redundancy (DMR) in this paper. The proposed approach can dynamically adjust the checkpoint intervals. We also design methods to calculate the optimal numbers of checkpoints, which can minimize the average execution time of tasks. Further, the adaptive checkpointing is combined with the DVS (dynamic voltage scaling) scheme to achieve energy reduction. Simulation results show that, compared with the previous methods, the proposed approach significantly increases the likelihood of timely task completion and reduces energy consumption in the presence of faults.

1. Introduction

Checkpointing is an important method for fault-tolerance in real-time systems in the condition of harsh environment. The following three types of checkpoints are well known: CSCP, SCP, and CCP [1-3]. CCPs are used to compare the states of the processors without storing them, while, the processors store their states without comparison in SCPs. If the two operations are used together in the same checkpoint, we call it CSCP. Using CCP and SCP, Ziv and Bruck have shown numerically that the task execution time is significantly reduced [1,4]. Using additional CCPs and SCPs, Nakagawa and Fukumoto have used a triple modular redundancy and double modular redundancy to analyze the optimal checkpoint intervals that can minimize a task execution time, respectively [5].

In addition, many real-time systems are often energy-constrained since system lifetime is determined to a large extent by the battery lifetime [2]. For example, autonomous airborne and sea-borne systems working on limited battery supply, space systems working on a limited combination of solar and battery power supply, time-sensitive systems deployed in remote locations where a steady power supply is not available [3,6]. DVS has emerged as a popular solution to the problem of reducing power consumption during system operations. The DVS becomes possible on the availability of embedded processors that can dynamically scale the frequency by adjusting the operation voltage [2,3]. Many embedded processors have the ability to dynamically scale the operation voltage currently. Such as, the mobile processors from Intel with its SpeedStep [7] technology. In the realm of real-time systems, the DVS techniques focus on minimizing energy consumption of the system under the condition of meeting the deadlines. The DVS and fault tolerance for real-time systems have been studied as separate problems. It is only recently that an attempt has been made to combine fault tolerance with the DVS [3].

The combination of DVS, CSCPs (CCPs or SCPs) can be used to satisfy system’s DVS requirement and improve the performance of real-time systems. However, none of the mentioned papers addressed these issues in terms of conjunction. Using additional SCPs and CCPs, we modify the methods of [3] in the double modular redundancy (DMR) in this paper. Different from the existing methods, our approach is to tune the scheme to the specific system which it is implemented on, and use both the comparison and storage operations efficiently, the performance of checkpoint schemes is improved.

Some notations used in our paper are listed below:

- $t_s$: the time to store the states of processors.
- $t_{cp}$: the time to compare processors’ states.
- $t_r$: the time to roll back the processors to a consistent state.
- $R_r$: remaining execution time.
- $R_d$: time left before the deadline.

*$^*$ This work is supported in part of Fujian natural science grant (A0410004), Fujian young science & technology innovation grant (2003J020), NCETXMU 2004 program, Program of 985 Innovation on Information in Xiamen Univ.(2004-2007) and Xiamen Univ. research grant (0630-E23011).
Assume task \( \tau \) has a period \( T \), a deadline \( D \), a worst-case computation time \( N \) when there are no fault in the system. An upper boundary \( k \) represents the number of fault occurrences that have to be tolerated. \( C \) is the overhead of a checkpoint. Faults arrive as a Poisson process with parameter \( \lambda \), the average execution time for the task is minimum, if a constant checkpoint interval of \( \sqrt{2C/\lambda} \) is used\(^{[6]}\). We refer to this as the Poisson-arrival approach. If the Poisson-arrival scheme is used, the effective task execution time is minimum, if the constant checkpoint interval is set to \( \sqrt{NC/k} \) \(^{[9]}\). This is the \( k \)-fault-tolerant approach.

In addition, we assume that task \( \tau \) is divided equally into \( n \) intervals of length \( T = \frac{N}{n} \), and at the end of each interval, CSCP is always placed.

### 2.1 Additional SCPs

Each CSCP interval is divided equally into \( m \) intervals of length \( T = \frac{T}{m} \) (figure 1). The SCPs are placed between the CSPCs, the states of two processors are stored at \( iT \) and \( jT \) \((i=1,2,\ldots,m-1)\). If two states do not get an agreement at time \( jT \), then, we need to find the most recent SCP with identical states and roll back to it. As shown in figure 1, two processors are rolled back to \((i-1)T_i \) because some errors have occurred during \((i-1)T_i \), \( iT_i \), and repeat the execution from \((i-1)T_i \). The average execution time \( R_{\tau}(m) \) for a CSCP interval \((i-1)T, jT \) is given by a renewal-equation\(^{[6,10]}\):

\[
R_{\tau}(m) = \left( mT_i + mt_s + t_{cp} \right) e^{-2\lambda m T_i} + \sum_{i=1}^{m} \left( mT_i + mt_s + t_{cp} + T + R_i(m-1) \right) d(1-e^{-2\lambda T})
\]

\[
= mT_i + mt_s + t_{cp} + \left[ \frac{1}{2} m(m+1)(T_i + t_s) + m(t_{cp} + t_s) \right] e^{2\lambda T_i} - 1
\]

Therefore, the average execution time of a task \( R_{SCP}(n) = nR_{\tau}(m) \).

Replace \( m = T / T_i \), we have

\[
R_i(T_i) = \frac{T}{T_i} + t_{cp} + [(1 + \frac{T}{T_i}) - \frac{T}{2T_i} + \frac{T}{T_i} e^{2\lambda T_i} - 1] \ldots (1)
\]

If \( T_i \to 0 \), then \( R_i(T_i) \to +\infty \). Let \( T_i = T \), we have \( R_i(T_i) = (T + t_c + t_{cp}) e^{2\lambda T} \). Thus, there exists a finite \( \hat{T}_i \in ((j-1)T, jT) \) which minimizes \( R_i(T_i) \).

The adaptive checkpointing with SCPs, adapchp-SCP \((D, E, C, k, \lambda)\), is described in Figure 2. A check is performed to see if fault has been detected at each CSCP.

![Fig. 2 Procedure for calculating the \( m \)](image)

Procedure adapchp-SCP \((D, N, C, k, \lambda)\)\{\}

1. Find \( \hat{T}_i \) which minimizes \( R_i(m); \)
2. if \( (\hat{T}_i < T) \) \}
3. \( m = \frac{T}{\hat{T}_i}; \)
4. if \( (R_i(m) < R_i(m+1)) \) then
5. \( \hat{m} = m; \)
6. else \( \hat{m} = m+1; \)
7. \} else \( \hat{m} = 1; \)
8. return \( \hat{m}; \)

The adaptive checkpointing with SCPs, adapchp-SCP \((D, E, C, k, \lambda)\), is described in Figure 3. A check is performed to see if the task has been completed in line 4, and line 5 checks for the deadline constraint. The length of SCP and CSCP interval is set in line 6 and line 7, respectively. In line 9, a check is performed to see if fault is detected. If there is no fault, then continue to run task, otherwise, roll back to previous SCP with identical states and continue execution, which are described from line 12 to
In line 2 and 14, we use procedure interval \((R_\lambda, R_n, C, R_f, \lambda)\) \(^3\) (figure 4) to calculate the checkpoint interval. In figure 4, \(I(C, \lambda) = \sqrt{2C/\lambda}\) is the checkpoint interval of the Poisson-arrival approach.

Procedure interval\((R_\lambda, R_n, C, R_f, \lambda)\)\):

1. \(\exp_{\text{error}}=\lambda^T\)
2. if \((\exp_{\text{error}}< R_f)\) \{
3. if \((R > Th_1 (R_\lambda, \lambda, C))\) then
4. \(\text{chk\_interval}=I(R_\lambda, R_n, C);\)
5. else if \((R > Th(R_n, R_f, C))\) then
6. \(\text{chk\_interval}=I(R_\lambda, R_n, \exp_{\text{error}}, C);\)
7. else \{ if \((R > Th(R_\lambda, R_f, C))\) then
8. \(\text{chk\_interval}=I(R_\lambda, R_n, C);\)
9. else \{ if \((R > Th(R_\lambda, \lambda, C))\) then
10. \(\text{chk\_interval}=I(C, \lambda, C);\)
11. else return \(\text{chk\_interval}\);\}

**Fig. 4 Calculating checkpointing interval**

\[I_2(N,k,C)=\sqrt{NC/k}\] is the checkpoint interval of the k-fault-tolerant approach. In addition, \(^3\) defined some equations:

\[
Th_1 (R_\lambda, \lambda, C) = (R_\lambda + C)/(1 + \sqrt{2C/\lambda})
\]

\[
Th(R_\lambda, R_n, R_f, C) = R_n + C + 2R_f - C - \sqrt{R_n(C + R_f + C)}
\]

\[
I_3(N,D,C) = 2NC/(D + C - N)
\]

Line 1 of figure 4 calculates the number of faults \(\text{Exp-fault}\) that are expected to occur in the remaining time \(R_\). If \(\text{Exp-fault}\) is less than or equal to \(R_f\), the k-fault-tolerant requirement is deemed to be more stringent than the Poisson-arrival criterion. In line 3, a check is performed to see if \(R\) exceeds the threshold \(Th_1 (R_\lambda, \lambda, C)\). If this condition is satisfied, the checkpoint interval is set to \(I(R_\lambda, R_n, C)\). In line 5, a check is performed to see if \(R\) exceeds threshold \(Th(R_\lambda, R_n, C)\) but is below \(Th_1 (R_\lambda, \lambda, C)\). If this condition is satisfied, the checkpointing interval is set to \(I(R_\lambda, \exp_{\text{fault}}, C)\). If the k-fault-tolerant threshold is met, the checkpoint interval is set to \(I(R_\lambda, R_n, C)\) in line 7. Line 8-10 handle the case when the k-fault-tolerant requirement is deemed to be less stringent than the Poisson-arrival criterion.

2.2 Additional CCPs

Each CCP interval is divided equally into \(m\) intervals of length \(T_i = \frac{T}{m}\). The CCPs are placed between CCPs, and the states of the two processors are compared at \(iT_i\) and \(jT\) \((i=1,2,\ldots, m-1)\). If two states do not reach to an agreement at \(iT_i\) and \(jT\), that means some errors have occurred during this interval, the two processors will be rolled back to \((i-1)T\) (figure 5).

The average execution time \(R_2(m)\) for an interval

\[
\text{Fig. 5 Task execution with ICCPs}
\]

\((j-1)T, jT\) is given by a renewal-equation:

\[
R_2(m) = (mT_j + mT_j + T_j)e^{-2jT_j}
\]

\[
+ \sum_{i=0}^{m} \left[ t_i^2 + T_j + T_j + R_2(m) \right] d \left( 1 - e^{-2jT_j} \right)
\]

\[
= T_je^{-2jT_j} + \left( e^{2jT_j} - 1 \right) T_j + T_j
\]

Therefore, the average execution time \(R_{CCP}(n)=nR_2(m)\).

Replacing \(m = T / T_j\), we have:

\[
R_2(T_j) = \left( t_i e^{2jT_j} + (e^{2jT_j} - 1) T_j + T_j \right) / (1 - e^{-2jT_j})
\]

If \(T_j \rightarrow 0^+\), then \(R_2(T_j) \rightarrow +\infty\). If \(T_j = T\), then \(R_2(T_j) = (T + T_j + T_j)e^{2jT_j}\). Therefore, there exists a finite \(T_j \in ((j-1)T, jT\)], which minimizes \(R_2(T_j)\). Differentiating equation (2) with respect to \(T_j\) and setting it to zero, we can get \(T_j\). We can use the similar approach described in figure 2 to calculate \(m\) which minimize \(R_2(m)\).

3 Adaptive checkpointing with DVS

With additional SCPs and CCPs, we show how adaptive checkpointing scheme can be combined with the DVS to obtain fault tolerance and power savings in real-time systems. In the one hand, our approach is to maximize the probability that the task meets its deadline in the presence of faults. In another hand, our approach is to reduce energy consumption through the DVS.

Assume that task \(\tau\) has a fixed quantity of computation cycles \(N\) in the fault-free condition. Because the variable voltage CPUs are available, the time to execute task \(\tau\) depends on the processor speed. We therefore characterize \(\tau\) by a fixed quantity \(N\), namely, its worst-case number of CPU cycles, needed to execute the task at the minimum processor speed. For the rest of this paper, we normalize the units of \(N\) such that the minimum processor speed is 1. That is, if the minimum processor speed is \(S\) cycles per second, then we express the number of cycles in units of \(S\) cycles and thus normalize the minimum processor speed to \(S_{min}=1\). Of course, period \(T\) and deadline \(D\) are expressed in terms of the number of CPU cycles at the
To simplify the analysis and to allow for the derivation of analytical formulas, we would like to assume that a single processor with two speeds $f_1$ and $f_2$, and $f_1$ is the minimum processor speed, namely, $f_1 = S_{\text{min}} = 1$. Moreover, the processor can switch its speed in a negligible amount of time.

Additional notations we use is below:

- $R_c$ : the number of instructions of the task that remain to be executed at the time of the voltage scaling decision.
- $c$ : the number of clock cycles that a single checkpoint takes.

The checkpointing cost $C$ at frequency $f$ is given by $C = c/f$.

To ensure $\lambda t_{\text{est}}$ fault tolerance during task execution, the checkpointing interval must be set to $\sqrt{\lambda t_{\text{est}}}/(\sqrt{C}/2) = \sqrt{C}/(2 \sqrt{\lambda f})$. In addition, we have $t_{\text{est}} = R_c (1 + \sqrt{\lambda c/(f)}$ [3].

We consider the voltage scaling to be feasible if $t_{\text{est}} \leq R_f$. This forms the basis of the energy-aware adaptive checkpointing that are described in procedure adapchp_dvs_SCPs and adapchp_dvs_CCPs (Figure 6 and Figure 7).

Procedure adapchp_dvs_SCP(D, N, c, k, $\lambda$)

1. $R_r = E, R_f = D, R_j = k;$
2. if $(t_m(R_f, f_f) \leq R_f) f = f_f$; else $f = f_f;$
3. $t_v = \text{interval}(R_f, R_j / f, c / f, R_j, \lambda);$ 
4. $m = \text{num}_\text{SCP}(t_v);$ $t_v = t_v / m;$
5. while ($R_f \neq R_f > 0$) do{
6. if ($R_f > R_f$) break with task failure;
7. Insert SCP with interval length $t_v$;
8. Update $R_f$, $R_j$ according to speed $f$;
9. if(no error has been detected at CCP/CSCP)
10. Resume execution;
11. else{
12. Roll back to the last CCP;
13. $R_f = R_f - 1;$
14. if $(t_m(R_f, f_f) \leq R_f) f = f_f$; else $f = f_f;$
15. $t_v = \text{interval}(R_f, R_j / f, c / f, R_j, \lambda);$ 
16. $m = \text{num}_\text{SCP}(t_v);$ $t_v = t_v / m;$
17. Resume execution;}
}

Fig. 7 adapchp_dvs_CCPs

4 Simulation results

We carried out a set of simulation experiments to evaluate our adaptive checkpointing schemes adapchp_dvs_CCPs and adapchp_dvs_SCPs (referred to as A_D_C and A_D_S) and to compare it with the Poisson-arrival (referred to as Poisson), the $k$-fault-tolerant (referred to as $k$-f-t) checkpointing schemes and ADT_DVS[3] (referred to as A_D). Faults are injected into system using a Poisson process with various values for the arrival rate $\lambda$. Due to the stochastic nature of the fault arrival process, the experiment is repeated 10,000 times for the same task and the results are averaged over these runs. We are interested here in the probability that the task completes on time, and the energy consumption. Energy consumption is measured by summing the product of the square of the voltage and the number of computation cycles over all the segments of the task [3]. As in [3], we use the term task utilization $U$ to refer to the ratio $N/D$. In order to compare with results of ADT_DVS scheme, we let $t_{\text{cp}} = 0$ and $f_2 = 2f_1$. Moreover, let $P$ and $E$ represent the probability of timely completion of tasks and energy consumption, respectively.

4.1 Additional SCPs

As mentioned previously, additional SCPs scheme fits systems, in which time overhead is determined mainly by the time to compare processor’s states. Therefore, the parameters are as following: $D = 10000$, $t_{\text{cp}} = 20$, $c = 22$.

First, we let the Poisson-arrival and the $k$-fault-tolerant schemes use the lower speed $f_1$. The task
utilization $U$ in this case is $N(f_tD)$. Our experimental results are shown in Table 1. In Table 1(a), for $f_t > 0.001$ and $0.7 < U < 0.9$ (high fault arrival rate and relatively high task utilization), the experimental results show that adachp-dvs-SCPs scheme clearly outperforms the ADT_DVS scheme. Although Poisson-arrival and the $k$-fault-tolerant schemes clearly outperform the task utilization, the experimental results show that $0.2$. In Table 1(b), for $f_t < 0.001$ and $0.9 < U \leq 1$ (low fault arrival rate and high task utilization), we draw the similar conclusions described above.

We assume that both Poisson-arrival and the $k$-fault-tolerant schemes use the higher speed $f_t$. Then the task utilization $U$ in this case is $N(f_tD)$. Our experimental results are shown in Table 2. We also can draw a conclusion that our scheme outperforms the other three schemes.

### Tab. 1 The comparison between adachp-dvs-SCPs and other algorithms, both the Poisson-arrival and the $k$-fault-tolerant schemes use the lower speed $f_t$.

<table>
<thead>
<tr>
<th>$U$</th>
<th>$\lambda$ (10^{-2})</th>
<th>$P$</th>
<th>$E$</th>
<th>$A_D$</th>
<th>$A_D_S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.76</td>
<td>0.14</td>
<td>0.1185</td>
<td>0.1115</td>
<td>0.9991</td>
<td>0.9999</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>0.0489</td>
<td>0.0466</td>
<td>0.9992</td>
<td>0.9999</td>
</tr>
<tr>
<td></td>
<td>0.78</td>
<td>0.0504</td>
<td>0.0486</td>
<td>0.9990</td>
<td>0.9999</td>
</tr>
<tr>
<td></td>
<td>0.14</td>
<td>0.0181</td>
<td>0.0182</td>
<td>0.9993</td>
<td>0.9999</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3943</td>
<td>0.39396</td>
<td>62687</td>
<td>56814</td>
</tr>
<tr>
<td>0.80</td>
<td>0.14</td>
<td>0.0091</td>
<td>0.0204</td>
<td>0.9993</td>
<td>0.9999</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>0.0021</td>
<td>0.0062</td>
<td>0.9990</td>
<td>0.9998</td>
</tr>
<tr>
<td></td>
<td>0.82</td>
<td>0.0013</td>
<td>0.0018</td>
<td>0.9995</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3916</td>
<td>0.39122</td>
<td>65778</td>
<td>60731</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3920</td>
<td>0.39200</td>
<td>67987</td>
<td>62091</td>
</tr>
</tbody>
</table>

4.2 Additional CCPs

Additional CCPs scheme fits systems which overhead time is determined mainly by the time to store processor’ states. Therefore, the parameters is as following: $D=10000$, $t_r=20$, $t_p=2$, $c=22$.

### Tab. 2 The comparison between adachp-dvs-SCPs and other algorithms, both the Poisson-arrival and the $k$-fault-tolerant schemes use the higher speed $f_t$.

<table>
<thead>
<tr>
<th>$U$</th>
<th>$\lambda$ (10^{-2})</th>
<th>$P$</th>
<th>$E$</th>
<th>$A_D$</th>
<th>$A_D_S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.92</td>
<td>1.0</td>
<td>0.3914</td>
<td>0.3965</td>
<td>0.9229</td>
<td>0.9549</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>0.1650</td>
<td>0.1628</td>
<td>0.9793</td>
<td>0.9985</td>
</tr>
<tr>
<td>0.95</td>
<td>1.0</td>
<td>0.3851</td>
<td>0.3852</td>
<td>0.9188</td>
<td>0.9516</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>0.1520</td>
<td>0.1510</td>
<td>0.9462</td>
<td>0.9944</td>
</tr>
<tr>
<td>1.00</td>
<td>1.0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.9146</td>
<td>0.9557</td>
</tr>
</tbody>
</table>

4.3 Additional CCPs scheme fits systems which overhead time is determined mainly by the time to store processor’ states. Therefore, the parameters is as following: $D=10000$, $t_r=20$, $t_p=2$, $c=22$.

### Tab. 3 The comparison between adachp-dvs-CCPs and other algorithms, both the Poisson-arrival and the $k$-fault-tolerant schemes use the lower speed $f_t$.

<table>
<thead>
<tr>
<th>$U$</th>
<th>$\lambda$ (10^{-2})</th>
<th>$P$</th>
<th>$E$</th>
<th>$A_D$</th>
<th>$A_D_S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.76</td>
<td>0.14</td>
<td>0.1104</td>
<td>0.1070</td>
<td>0.9990</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>0.0505</td>
<td>0.0479</td>
<td>0.9989</td>
<td>0.9999</td>
</tr>
<tr>
<td>0.78</td>
<td>0.14</td>
<td>0.0530</td>
<td>0.0534</td>
<td>0.9989</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>0.0190</td>
<td>0.0210</td>
<td>0.9989</td>
<td>0.9998</td>
</tr>
</tbody>
</table>
Our experimental results are shown in table 3 and table 4. Similar to section 4.1, simulation results show that compared to ADT_DVS scheme, the proposed scheme significantly increases the likelihood of timely task completion and reduces power consumption in the present of faults.

5. Conclusion

In this paper, we presented an adaptive checkpointing, using a DMR with two processors, and tuning the scheme to the specific system which it is implemented on. The proposed scheme is done by inserting two types of checkpoints (CCP and SCP) between CSCP. Separating the comparison and store operations enables choosing the optimal interval for each operation, without concerning about the other. We also discussed the optimal numbers of checkpoints that minimize the average times. Based on that, we combined the adaptive checkpointing with the DVS schemes to achieve energy reduction. We presented simulation results which showed the advantages of our scheme. We will extend the proposed scheme to other task duplication systems with security needs as a future work.

References