Failure tolerance for a multicore real-time system scheduled by PD2

Yves MOUAFO
LIAS-ENSMA
Teleport 2, 1 av. clément Ader
BP 40109,86961
Futuroscope-Chasseneuil
yves.mouafo@ensma.fr

Annie GENIET
LIAS-ENSMA
Teleport 2, 1 av. clément Ader
BP 40109, 86961
Futuroscope-Chasseneuil
annie.geniet@ensma.fr

Gaëlle LARGETEAU
SXlim-SIC, Univ. Poitiers
BP 30179, 86962
Futuroscope-Chasseneuil
glargeteau@sic.univ-poitiers.fr

ABSTRACT

This work addresses the problem of failure tolerance for real-time applications, running on a multicore hardware architecture. In fact, at any time during the scheduling process, a problem may occur on any of the processor cores, affecting the task that was running on it. We focus on systems composed of periodic independent tasks with simultaneous first release and implicit deadlines. The system is scheduled under the fair algorithm PD2.

Our approach is based on limited hardware redundancy: the system will run on a processor with one more core than required. Then we prove that, if the subtask running on the faulty core is not rescheduled, the application can keep running on the remaining cores without temporal or fairness faults.

General Terms

Real-time systems, Pfair scheduling, Fault tolerance, limited redundancy, PD2 algorithm

Keywords

Scheduling, failure, redundancy

1. INTRODUCTION

With the introduction of multicore system-on-a-chip architectures for embedded systems, tolerance to failure is bound to become a major aspect in application design. In fact, it is well-known [2] [4] that technology scaling sensitizes electronic devices to external disturbs. The overall effect is a probability that a core of the processor fails during the execution of the application. In this paper, we adopt the classical modeling of a real-time application [3] which consists of a set of n independent periodic tasks S=$\{\tau_1, \tau_2, ..., \tau_n\}$. Each task τ_i is submitted to hard temporal constraints and characterized by four temporal parameters: the first release date or offset r_i, the worst-case execution time C_i, the period T_i and the relative deadline D_i. Each task τ consists of an infinite set of instances (or jobs). An important characteristic of the task τ_i is its utilization $U_i = \frac{C_i}{T_i}$. For any system of tasks S, we denote $U = \sum_{i=1}^{n}(C_i/T_i)$ the load of the system. We assume that the temporal parameters are known and determinist, the tasks have simultaneous first releases ($r_i = 0$) and implicit deadlines ($D_i = T_i$). The system is scheduled under the Pfair algorithm PD2 [7]. Under these assumptions, a necessary and sufficient condition for feasibility is: $U \leq m$ [6] (m denotes the number of processor cores) and PD2 is optimal.

At any time during the scheduling process, a failure may occur on any of the cores, affecting or not a task. It becomes necessary to reorganize the system so that the system keeps on running on the remaining cores without temporal or fairness failures. The classical way to provide fault-tolerance on multicore platforms (which are generalized by multiprocessors) is to use time and/or space redundancy [1]. The idea is to introduce redundant copies of the elements to be protected (processor or other components), and exploit them in the case of a fault. In time redundancy, the same software is executed two or more times on the same CPU, and the produced results are compared. In space redundancy, on the contrary, the same software is executed at the same time on different CPUs. The number of cores used to implement space redundancy is determinant for the energy consumption of the processor and thus has an environnemental impact. We propose an approach based on limited hardware redundancy, where only one core is added to those requiered. Thus, we prove that, using a fair algorithm such as PD2, if the impacted subtask is not rescheduled, limited redundancy guarantees the validity and the fairness of scheduling despite the defection of one core.

The remainder of this paper is organized into four sections. In the next section (Section 2) PD2 algorithm is briefly introduced.Then the limited redundancy approach is presented (section 3). Finally, we present our experimental studies (Section 4) and our contribution which is scheduling with failure without re-execution of the impacted subtask (section 5).

2. PD2 ALGORITHM

PD2 [7] is a Pfair algorithm [6] which objective is to approach an ideal scheduling in which each task τ_i receives exactly $U_i \times t$ processor time units from the instant 0 to t. The construction of a PFair scheduling involves dividing each task τ_i into unitary subtasks. Each subtask $\tau_i^j (j \geq 0)$ has a pseudo-release date $r_i^j = \lfloor \frac{j}{U_i} \rfloor$ and a pseudo-deadline $d_i^j = \lceil \frac{j+1}{U_i} \rceil$. The interval $[r_i^j, d_i^j[$ represents the *feasability window* of the subtask τ_i^j. Subtasks are scheduled in increasing pseudo-deadline order and when the pseudo-deadlines are equal, PD2 uses two additional criterias to determine the priority order between subtasks: the bit succesor b_i^j and the group deadline D_i^j. These criterias are defined as fol-

lows:

$$b_i^j = \begin{cases} 1 & \text{if } r_i^{j+1} = d_i^j \\ 0 & \text{otherwise} \end{cases}$$

$$D(\tau_i^j) = \begin{cases} \lceil \frac{d(\tau_i^j) - j - 1}{1 - U_i} \rceil & \text{if } U_i \geq 0.5, j \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

According to PD2, a subtask τ_i^j has priority over the subtask τ_q^k if one of the following conditions is verified:

1. $d_i^j < d_q^k$
2. $d_i^j = d_q^k \wedge b_i^j > b_q^k$
3. $d_i^j = d_q^k \wedge b_i^j = b_q^k = 1 \wedge D_i^j > D_q^k$

2.1 Example of a scheduling with PD2

Let τ be the system of tasks defined by $\tau = (\tau_1 < 2, 3 >, \tau_2 < 2, 6 >, \tau_3 < 3, 8 >, \tau_4 < 6, 8 >, \tau_5 < 5, 12 >)$. Figure 1 shows the PD2 schedule of the system τ from the instant 0 to 12. Notice that $U(\tau) = 2.54 < 3$, thus τ is schedulable on a 3 cores processor.

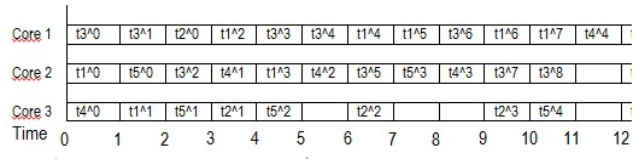

Figure 1: Example of PD2 schedule

3. LIMITED REDUNDANCY APPROACH

3.1 Failure model

We consider that while the application is running, one core of the processor failed. Moreover, the failure is permanent and we assume that there is an instantaneous mechanism for detection that locates the affected core [8] at the very moment when the failure occurs. In such a case, a fault can affect only one subtask, the one that is running on the core which suffers the failure. All or part of this subtask will have to be re-executed. In [5], S. MALO distinguishes two possible scenarios:

- The failure occurs immediately after the context switch. In this case, the application should simply continue execution on the remaining cores. We must just verify that the reorganization is possible and does not cause a task to miss it deadline;
- The failure occurs during the execution of the subtask. In this case, three policies are possible: (1) Re-execution of what has been executed since the last backup of the context; (2) The full task is re-executed; (3) The current subtask is simply abandoned and execution continues on the remaining cores.

The *figure 2* below gives an illustration of the scenarios and draws the outline of the context of our contribution.

In fact, we focus on the first scenario and the third case of the second scenario (*figure 2.b*). It concerns the continuation of the execution on the remaining cores, either when no subtask is affected or when the affected subtask is abandoned. We must therefore ensure that the application can be reconfigured without temporal errors.

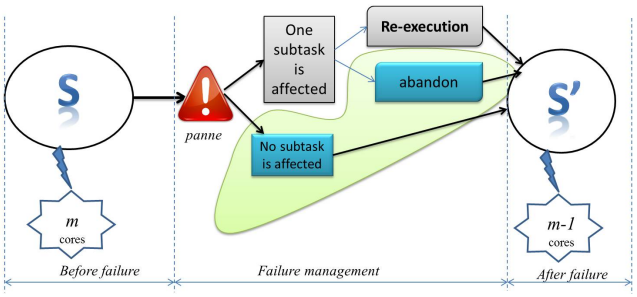

Figure 2: Failure model

3.2 Limited redundancy

Let S be a system of tasks. In the introduction it was established that S is schedulable if and only if $U \leq m$. To overcome a failure of a core, limited hardware redundancy is a technic which consists in providing a core more than necessary. So, instead of running S on m cores, it will run on $m + 1$ cores. So, when failure occurs, the system will remain schedulable on the m remaining cores.

The figure 3 illustrates this technic on the system τ of the previous example where a failure occurred on the CPU core 1 at time 6 affecting the subtask τ_1^4.

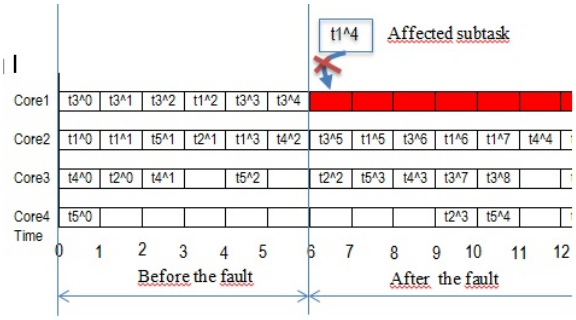

Figure 3: Example of scheduling with limited redundancy technic

4. EXPERIMENTAL STUDY

With the above example system τ, scheduling continues normally in case of failure with non re-execution of the impacted subtask. Can we therefore generalize this technic? To answer this question, we first conducted an experimental study. For this, a software prototype FTA (Fault Tolerance Analyser) was designed to simulate scheduling with fault by Pfair algorithms. To get concluding results about our approach, a total of 550 random systems has been generated and submitted to the simulation. The fault occurrence time and the failing core of the processor are randomly choosen and vary from one system to another. The experiment was repeated 50 times on the 550 generated systems. The obtained results show that, no matter the time the failure occurs or the affected core, no matter the system load or the number of heavy tasks ($U_i \geq 0.5$), scheduling with fault without reexecution guarantees the respect of validity and fairness constraints.

In the following section, we establishh the proof of that result.

5. SCHEDULING WITH FAULT WITHOUT REEXECUTION OF AFFECTED SUBTASK

5.1 The result

Our main result is given by theorem 1.

THEOREM 1. ***Any system*** $\tau = (\tau_1 < C_1, T_1 >, \tau_2 < C_2, T_2 >, ..., \tau_n < C_n, T_n >)$ ***which consists of n periodic and independent tasks, with simultaneous first releases and implicit deadlines feasible under the fair algorithm PD2 on a*** m ***cores processor and running on*** $m+1$ ***cores, remains feasible on*** m ***cores, after the failure of one of the cores without rescheduling the impacted subtask.***

In other words, the constraints of validity and fairness are met in the schedule even if a failure occurs on one core and there is no re-execution of the impacted subtask.

The proof of this theorem requires the demonstration of three lemmas. We adopt the following notations and hypothesis.

5.2 Notations and hypothesis

Notations:

O^m:Sequence obtained when the system is scheduled on m cores;

$O^{(m+1)->m}$:Sequence obtained when the system is scheduled on a processor with $m+1$ functional cores initially, and m cores after a failure;

$O^m(t_i...t_j)$**:**Sequence obtained when the system is scheduled on a m-core processor from the instant t_i to the instant t_j.

$C_m(t)$:Set of pending subtasks at time t in a schedule on a m cores processor;

$C^u(t)$:Set of subtasks released at time t;

$C^e_m(t)$:Set of elected subtasks at time t in a schedule on m cores;

$C^r_m(t)$**:**Set of non-elected subtasks at time t in a scheduling on a m cores;

$t(\tau_i^j, O)$:Date by which the subtask τ_i^j is executed in the sequence O;

t_p: Failure time;

$Rank(\tau_i^j, C)$:Position of τ_i^j in the set C which is assumed to be sorted according to PD2;

$<_{PD2}$**:** Priority order of PD2 algorithm (see section 2);

$S \rightsquigarrow <_{PD2}$:The subtask of the set S are sorted according to the PD2 priority order.

Hypothesis:

$U \leq m$ thus O^m and O^{m+1} are valid and fair.

5.3 Some lemmas

Lemma 1: At any time t, the set of the ready subtasks in a $m+1$ **cores schedule is included in the set of the ready subtasks in a** m **cores schedule.**

$$C_{m+1}(t) \subseteq C_m(t)$$

PROOF. (lemma 1) :by induction on t.

At $t = 0$, $C_{m+1}(0) = C_m(0) = C^u(0)$.

Assume that at $t-1$ we have $C_{m+1}(t-1) \subseteq C_m(t-1)$, we will show that at t we have $C_{m+1}(t) \subseteq C_m(t)$.

We know that the set of the ready subtasks at a time t is composed of the subtasks released at t and the subtasks non-elected at t-1. So,

$$\begin{cases} C_m(t) = C^u(t) \cup C^r_m(t-1) \\ C_{m+1}(t) = C^u(t) \cup C^r_{m+1}(t-1) \end{cases}$$

We will thus show that

$C^r_{m+1}(t-1) \subseteq C^r_m(t-1)$.

Consider any subtask $\tau_i^j \in C^r_{m+1}(t-1)$,

$\tau_i^j \in C^r_{m+1}(t-1)$ means that

$\tau_i^j \in C_{m+1}(t-1) = \{\tau_{i1}^{j1}\tau_{i2}^{j2}...\tau_{ir}^{jr}\}$

and τ_i^j is not scheduled at t-1.

We assume that $C_{m+1}(t-1)$ is sorted according to PD2.

So,$\exists h, m+1 < h < r$ such that $\tau_i^j = \tau_{ih}^{jh} (h = Rank(\tau_i^j, C_{m+1}(t-1)))$.

According to induction hypothesis,

$C_{m+1}(t-1) \subseteq C_m(t-1) = \{\tau_{a1}^{b1}\tau_{a2}^{b2}...\tau_{an}^{bn}\}$.

Therefore,

$Rank(\tau_i^j, C_m(t-1)) \geqslant Rank(\tau_i^j, C_{m+1}(t-1))$.

Hence,

$\tau_i^j \in C_m(t-1)$, $\exists v, m+1 < v \leq n$, such that $\tau_i^j = \tau_{av}^{bv}$.

Thus, τ_i^j is not scheduled on m+1 cores and consequently on m cores.

Conclusion:

$\tau_i^j \in C^r_m(t-1)$, thus $C^r_{m+1}(t-1) \subseteq C^r_m(t-1)$, and thus, $C_{m+1}(t) \subseteq C_m(t)$. □

Lemma 2: At any time t, the set of the pending subtasks in a $m+1$ **cores schedule is included in the set of the pending subtasks in a** $(m+1) -> m$ **cores schedule, which itself is included in the set of the pending subtasks in a** m **cores schedule.**

$$C_{m+1}(t) \subseteq C_{(m+1)->m}(t) \subseteq C_m(t)$$

PROOF. (lemma 2) : we reason in the same way as for Lemma 1

We first prove the second part of this inclusion, which is $C_{(m+1)->m}(t) \subseteq C_m(t)$.

At a time $t \leq t_p$, we have $m+1$ functionnal cores. So, $C_{(m+1)->m}(t) = C_{m+1}(t) \subseteq C_m(t)$ (Lemma1).

Let's suppose that at the time $t-1 \geq t_p$ we have $C_{(m+1)->m}(t-1) \subseteq C_m(t-1)$.

We must show that at t we have $C_{(m+1)->m}(t) \subseteq C_m(t)$.

We know that

$$\begin{cases} C_m(t) = C^u(t) \cup C^r_m(t-1) \\ C_{(m+1)->m}(t) = C^u(t) \cup C^r_{(m+1)->m}(t-1). \end{cases}$$

So, it remains to show that

$C^r_{(m+1)->m}(t-1) \subseteq C^r_m(t-1)$.

For that, consider any subtask $\tau_i^j \in C^r_{(m+1)->m}(t-1)$, $\tau_i^j \in C^r_{(m+1)->m}(t-1)$ means that $\tau_i^j \in C_{(m+1)->m}(t-1) = \{\tau_{i1}^{j1}\tau_{i2}^{j2}...\tau_{ir}^{jr}\}$ (sorted according to PD2) and τ_i^j is not scheduled at t-1.

As $C_{(m+1)->m}(t-1) \rightsquigarrow <_{PD2}$, and only m subtask was scheduled at time t-1. Thus, $\exists h, m < h \leq r$ such that $\tau_i^j = \tau_{ih}^{jh}$.

According to our induction hypothesis,

$C_{(m+1)->m}(t-1) \subseteq C_m(t-1) = \tau_{a1}^{b1}\tau_{a2}^{b2}...\tau_{an}^{bn}$.

Therefore,

$Rank(\tau_i^j, C_m(t-1)) \geqslant Rank(\tau_i^j, C_{(m+1)->m}(t-1))$.

Hence, $\tau_i^j \in C_m(t-1)$, $\exists v, m < v \leq n$, such that $\tau_i^j = \tau_{av}^{bv}$.

So, τ_i^j is not scheduled on m cores at t-1.

Conclusion:

$\tau_i^j \in C^r_m(t-1)$, thus $C^r_{(m+1)->m}(t-1) \subseteq C^r_m(t-1)$, and thus, $C_{(m+1)->m}(t) \subseteq C_m(t)$.

The first inclusion of our lemma:

$C_{m+1}(t) \subseteq C_{(m+1)->m}(t)$ can be proved in the same way.

□

Lemma 3:Any subtask is scheduled earlier in a configuration of $m+1$ cores without failure than in a configuration of $m+1$ cores at the beginning and m cores after a failure, and earlier in this last configuration than in a m cores configuration without failure.

$$\forall \tau_i^j, t(\tau_i^j, O^{m+1}) \leq t(\tau_i^j, O^{(m+1)->m}) \leq t(\tau_i^j, O^m)$$

PROOF. (lemma 3) As before, we first prove the second inequality. We thus prove that
$\forall \tau_i^j, t(\tau_i^j, O^{(m+1)->m}) \leq t(\tau_i^j, O^m)$.
Let τ_i^j be a subtask. At any instant t, if $\tau_i^j \in C^e_{(m+1)->m}(t)$ then $\tau_i^j \in C_{(m+1)->m}(t)$.
According the Lemma 2,
$C_{(m+1)->m}(t) \subseteq C_m(t)$, therefore, $\tau_i^j \in C_m(t)$.
Since $C_m(t) = C^e_m(t) \cup C^r_m(t)$
we have two cases to consider:
First case: $\tau_i^j \in C^e_{(m+1)->m}(t)$ and $\tau_i^j \in C^e_m(t)$.
In this case, τ_i^j is scheduled at time t in both configurations.
So, $t(\tau_i^j, O^{(m+1)->m}) = t(\tau_i^j, O^m) = t$
Second case: $\tau_i^j \in C^e_{(m+1)->m}(t)$ and $\tau_i^j \in C^r_m(t)$.
In this case, τ_i^j is scheduled at time t in $O^{(m+1)->m}$ and will be scheduled later in O^m.
So, $t(\tau_i^j, O^{(m+1)->m}) = t$ and $t(\tau_i^j, O^m) > t$
and then $t(\tau_i^j, O^{(m+1)->m}) \leq t(\tau_i^j, O^m$.
The first inequality can be proved in the same way. □

With these three lemmas we can now establish the proof of the theorem.

5.4 Theorem proof

Let us now show that even in the case of the failure of one core, the scheduling of the system remains valid and fair, if the affected subtask is not re-executed. In other word, we have to prove that apart from the affected subtask, all subtasks of systems still run in their respective feasability windows.i.e $\forall \tau_i^j, r_i^j \leq t(\tau_i^j, O^{(m+1)->m}) < d_i^j$

PROOF. (THEOREM)
Let τ_i^j be a sub-task.
If $t(\tau_i^j, O^{(m+1)->m}) < t_p$
then $t(\tau_i^j, O^{(m+1)->m}) = t(\tau_i^j, O^{m+1})$
because $O^{(m+1)->m}(1...t_p) = O^{m+1}(1...t_p)$.
Otherwise $t(\tau_i^j, O^{(m+1)->m}) \geq t_p$.
- The algorithm PD2 can schedule subtasks only if they are ready and so, after their pseudo-release date. Therefore,
$\forall \tau_i^j, r_i^j \leq t(\tau_i^j, O^{(m+1)->m})$.
- To prove the second inquality, let us reason by contradiction. Suppose that we have
$d_i^j \leq t(\tau_i^j, O^{(m+1)->m})$.
According to Lemma 3,
$t(\tau_i^j, O^{(m+1)->m}) \leqslant t(\tau_i^j, O^m)$.
We would then have $d_i^j \leq t(\tau_i^j, O^m)$ and thus, O^m would not be fair.
That is contrary to our initial hypothesis (section 5.2).
Conclusion: $\forall \tau_i^j, r_i^j \leq t(\tau_i^j, O^{(m+1)->m}) < d_i^j$
The sequence $O^{(m+1)->m}$ is valid and fair □

6. CONCLUSION

Because of the failure that may occur at any time on the processor, fault tolerance has become a major problem of real-time multicore systems. In this paper, we propose an approach of tolerance named limited hardware redundancy that consists in preventing a temporal fault due to the failure of one core by running the system on a processor with one core more than necessary. Then, we establish the proof that, if a failure affects one of the processor cores and the subtask that was running on it is not re-executed, the schedule can continue on the remaining cores while meeting the constraints of validity and fairness.

However, how to ensure compliance with these requirements in the case of re-execution of the affected subtask? In this regard, we have in mind the idea of adding to the limited hardware redundancy a policy of restriction and relaxation of subtasks feasability windows. This is to start the scheduling on a system of tasks with constrained deadlines, and to release the deadlines after failure. Thus the execution begins with smaller feasability windows that will expand after the failure. But one question remains: how to calculate the deadlines of the starting system from the system with implicit deadlines? Two approaches are possible. The first is to exploit the idle time of the scheduling and the second is to simulate the addition to each task of an abstract subtask which represents the re-execution of an eventual affected subtask. Both approaches will be explored in future works [9].

7. REFERENCES

[1] Michele Cirinei, Enrico Bini, Giuseppe Lipari, Alberto Ferrari, A Flexible Scheme for Scheduling Fault-Tolerant Real-Time Tasks on Multiprocessors. *1-4244-0910-1/07*,IEEE 2007 .

[2] M. Baleani, A. Ferrari, L. Mangeruca, A. SangiovanniVincentelli,M. Peri, and S. Pezzini, Fault-tolerant platforms for automotive safety-critical applications. *In Proceedings of the 2003 international conference on Compilers, Architecture and Synthesis for Embedded Systems* , pages 170 to 177, San Jose (CA), U.S.A., 2003.

[3] A. Choquet-Geniet, S. Malo, Scheduling an aperiodic flow within a real time system using Fairness properties *ARIMA Journal, vol. 18*, pp. 93 to 116, 2014.

[4] S. K. Baruah, Partitioning real time tasks among heterogeneous multiprocessors. *In Proceedings of the 33 Annual International Conference on Parallel Processing*, pages 467 to 474, Montreal, Canada, Aug. 2004.

[5] S. Malo Contribution à l'ordonnançabilité des applications temps-réel multiprocesseurs. *Thèse de doctorat de l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique*,décembre 2010.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15, p.600 to 625*,1996

[7] J. Anderson, A. Srinivasan A New Look at Pfair Priorities *Rap. tech.TR00 023, University of North Carolina at Chapel Hil* , sept. 1999

[8] E. Chanthery, Y. Pencole, Modélisation et intégration du diagnostic actif dans une architecture embarquée *Dans Journal Européen des Systèmes Automatisés*, MSR 2009, pages 789 to 803, 2009.

[9] Y. Mouafo, A. Geniet-Choquet, G. Largeteau-Skapin Robustesse des applications temps-réels multicoeurs *Actes de l'école d'été temps-réel 2015, pg 143-146.*