Power Saving of Real Time Embedded Sensor for Medical Remote Monitoring

Frédéric Fauberteau, Serge Midonnet,
Université Paris-Est,
Laboratoire d'Informatique Gaspard-Monge -
UMR CNRS 8049, France
{fauberte, midonnet}@univ-paris-est.fr

Dan Istrate
ESIGETEL - LRIT
1, Rue du Port de Valvins, Avon, France
dan.istrate@esigetel.fr

Abstract

The power saving is one of the important issue in the embedded systems. To reduce the consumption of the microprocessor of such a system, a way is to power down it when it is inactive. Theoretically, the time during which the microprocessor is inactive represents a pure gain of energy. But practically, we must consider that this time comprises a slot of time during which the clock must be synchronized. We propose to aggregate the idle times of the microprocessor to power down it the least possible but for the longest time possible. This aggregation can be perform by using a Slack Stealer algorithm.

1. Introduction and Application context

The context of the present study of power saving is the medical remote monitoring for elderly. The proportion of elderly is increasing in all societies throughout the world. As they are becoming older, they want to preserve their independence, autonomy and way of life. Several research teams have developed a number of systems for in-home health care monitoring and prevention towards day life risks. These systems are based on the deployment of several sensors in home in order to prevent and/or detect critical situations. They offer the comfort and independence of staying at home, the security of daily monitoring and proper medical attention. These sensors need to communicate with a central unit which take the decision to send the emergency in a distress case. In order to install the sensors a wireless solution is the only acceptable but which imply also a good autonomy.

To provide one answer to the medical remote monitoring, we assembled a group of researches from different backgrounds within a consortium (QuoVADis[1]) in order to develop a platform for several uses and to meet the needs identified above. QuoVADis is a French National project which aims to answering two of the problems arising from keeping elderly people at home: cognitive stimulation and the safety indoor.

[1]http://quovadis.ibisc.univ-evry.fr/

The first platform developed within this project [2] manages a system consisting of three modalities: a set of microphones disposed into the living rooms of the home of the elderly, a portable device that can measure heart rate, detect posture and possibly the fall of the person equipped and a set of infrared sensors that detect the presence of the person in a given part and also the standing posture of the person in question. The output of these three heterogeneous systems are collected, processed and fused through a multimodal platform (EMUTEM) [6].

The sound environment is used like an important source of information about the possible distress situations (screams, glass breaking, dishes, distress expressions). All microphones in the current implementation are connected to a central unit (embedded PC) in order to analyze the sound environment [3] in real time. We Analise currently the possibility to decentralize the treatments and to allow the use of embedded system based on DSP. This study concerning the task organization aims to evaluate the cost in terms of precessing time and power consumption in the case of decentralization of sound environment analyze.

The paper is organized as follows. In Section 2, we give a model representing a theoretical view of a sensor embedded system. In Section 3, we present several kind of class of *Slack Stealer* algorithm and we justify our choice. In Section 4, we show results of simulation. Finally, we conclude in Section 5 and we introduce future work in Section 6.

2. Preliminary Definitions

2.1. Sensor Model

Because we propose a general solution instead of a particular implementation for a specific system, we must introduce a model representing a sensor embedded system. This model is made of three modules : a set of sensors, communication and processing modules.

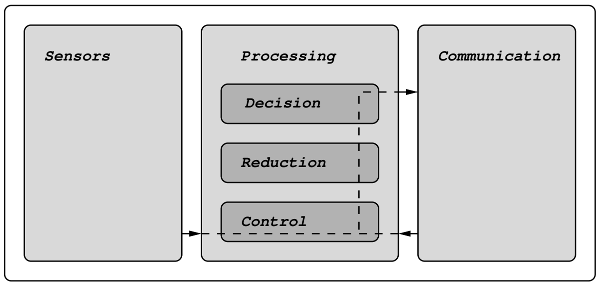

Figure 1. A model for a sensor embedded system

The sensors module corresponds to the sensor part of the sensor node. The sensor part collects data about its environment. This module generates a stream of messages which will be sent to the processing module.

The communication module enables a sensor node to communicate with other sensor nodes. Other sensor nodes collect data which can be received by this module and the suitable streams of messages are sent to the processing module. In the same way, locally processed data can be sent to other sensor nodes.

We will consider the processing module as a real-time system since it processes the streams of received messages which have time constraints. We divide this module in three functions.

- The control function is the most important one for our study. It provides controls over the system. In fact it can decide the admission of a new stream or the turning out of the processor,
- The decrease can apply operations over the stream to reduce the quantity of data. These operations can be logical : AND, OR, arithmetic : plus, average or more complex,
- The decision function can generate events according to the different dart's in the streams.

2.2. Real-Time Systems

Real-time systems are sets of tasks which have a priority, a cost and a deadline. The cost of a task is a worst case execution time. These tasks must not run over their deadline. The scheduler executes the available task with the highest priority. In our study, we consider preemptive scheduler with fixed priority. In such a scheduler, priority are not inheritable and a task can stop another task which has a lower priority. We can simply define the slack time in such a system as the period when the system is inactive.

Several kinds of task exists:

- periodic tasks which have a period. At each period, they are activated and an instance of these tasks must be executed,
- sporadic tasks which have a pseudo-period. They have the same properties that the periodic tasks but the pseudo-period is the minimum inter-arrival period. We can consider the worst pseudo-period for this type of task and it becomes easily assimilated with periodic tasks,
- aperiodic tasks which have no period. We cannot expect the arrival instant of these tasks.

In order to make our solution conceivable, we must consider aperiodic tasks as not the main type of task in the system. We can suffer aperiodic task in so much as their arrival is infrequent. Otherwise any speculation is possible.

2.3. Idle Times

A simple way to manage idle times is the scheduling of a task which has the priority the lowest. We can fix for this task an arbitrarily high cost. We just want this task to be executed when the system is inactive. This task could power down the processor. But without any scheduler modification, it can not predict its execution time. If the execution time of the sleeping task is lower than the necessary time to start the processor, some tasks can be postponed and don't meet their deadline. So we must add the time to power down and restart the processor in the worst case execution time to each periodic task. The advantage of this method is its simplicity. But we must increase the cost of the tasks because we can not know the length of the period of inactivity.

Because the over-cost of the first method is too high, we propose the use of a periodic task called "task server". We set its periodic cost called its capacity. So the time for starting the processor can be incorporated in the server cost. This method avoids to don't modify the worst case execution time of each periodic tasks. But the polling server can use not all the available idle times.

This method builds on slack stealer algorithms. These algorithms allow to compute the maximal value of slack times postponing periodic tasks with the constraint that they meet their deadline. We present these algorithms in the next section. The advantage of this method is that many scattered slack times can be aggregated. So the processor can be powered down less often but longer.

3. Slack Stealer Algorithms

The *Slack* is the maximum time that all tasks of the real-time system can be delayed without missing their deadline. We present in the following paragraphs several classes of algorithms called *Slack Stealer* which compute the *Slack* and we justify the choice of the last class, the approximate *Slack Stealer* algorithms.

The class of static algorithms is historically represented by the Lehoczky and Ramos-Thuel algorithm [7]. These algorithms compute the *Slack* value for each instance of each task before the starting up of the system and store it. During the execution of the system, the value of the *Slack* is known at t instant without more computation. These algorithms seem attractive because their don't require computation when the system is on-line. But they assume that the instants of the activation of the tasks are known. This assumption is restrictive in our case. Furthermore, a non negligible space is needed to stock the *Slack* values. This class of algorithms is not a good choice to implement our solution.

The class of dynamic algorithms is well represented by the Davis algorithm [1]. Contrary to static algorithms, they don't perform precomputation and so, they don't stock *Slack* values in memory. These algorithms compute the available *Slack* in the system on demand. They also seem attractive, but the complexity of the computation is in $O(n^2)$ where n is the number of tasks. Thus the implementation of such an algorithm is not possible for our solution.

The class of approximate algorithms is known to compute estimation of the *Slack* value. The algorithms of [7] and [1] give optimal values. Even if the algorithm presented in [5] give values which can be non maximal, it compute the *Slack* with linear complexity function of the number of tasks. This class of algorithms offer a good compromise between the computation time and the correctness of the result. We can implement such an algorithm for our solution.

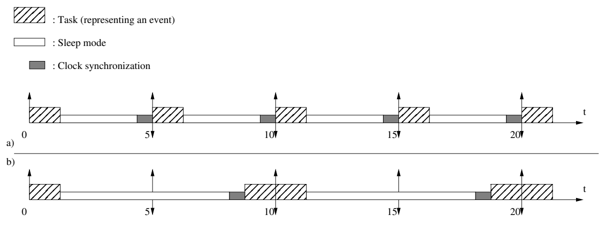

Figure 2. The sensor is powered down during idle times : a) without Slack management, b) with Slack management.

We show in Figure 2 the idea of our solution. We represented in a) a task representing the treatment of a event. This task is periodic and we power down the system when it is idle. In b), we show that the time during which the sensor is powered down can be increased by delaying the task. *Slack Stealer* algorithms allows us to know the length of the delay. In our example, we fix the cost of the task to 2 units of time and we fix the clock synchronization duration to 1 unit of time. During the clock synchronization, the system is restarted but it is not usable. Without the *Slack* management, the system can be powered down during $7 \times 4 = 28$ units of time. With the *Slack* management, the system is powered down 2 times instead of 4 and during $2 \times 15 = 30$ units of time.

4. Simulation Results

The results of simulation presented in the following section was obtained from a real-time simulator which was developed by Masson [4].

Firstly, we consider the instants when the processor can be switch off. Secondly, we compare various methods to exploit idle times.

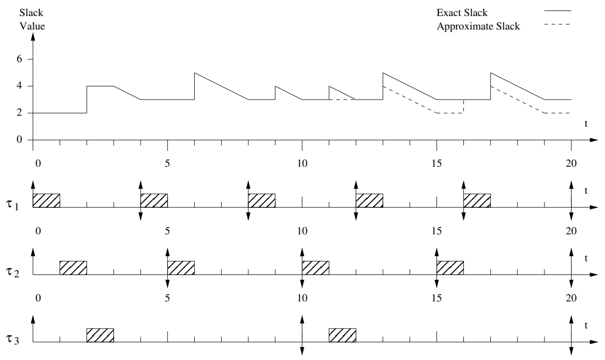

Figure 3. A system of 3 tasks with the associated slack at each instant.

We represent in Figure 3, a system of 3 periodic tasks. The task τ_1 is the most priority and τ_3 the less priority. When many tasks are active in the same time, the most priority are executed. We also represent the value of the *Slack* at each instant t. "Exact Slack" corresponds to the value computed from a dynamic algorithm [1] and "Approximate Slack" corresponds to the value computed from an approximate algorithm [5].When just one line is represented, the value estimated by the approximate algorithm is the same as the value computed by the exact algorithm. We remark that the approximation offers good results with regard to the exact value. To provide the better efficiency, we must power down the processor as few as possible because the restarting process represents a non-negligible cost. We remark that there is peak values for the *Slack* and they occur when a task is finished. To compute the *Slack*, the algorithm considers the current deadline of the active tasks. But when a task finished, the algorithm considers its next deadline. So we can say that a finished task reload the *Slack*. The approximate algorithm updates the *Slack* estimation at the beginning and at the end of each task. So we know at the end of each task if we have enough *Slack* to power down the processor and restart it without tasks miss their deadline.

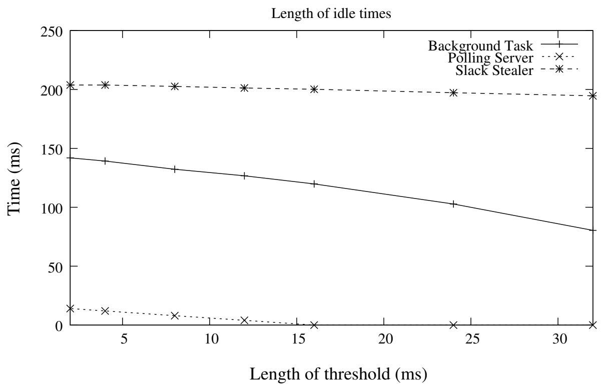

Figure 4. Average length of an idle times.

We compare in Figure 4 the average length of an idle times function of a power down threshold. 10 systems of 10 tasks has been generated; each system has a CPU load of 80%, so 20% of time is idle.

The simulations go on 800 seconds. We compare 3 methods to exploit idle times. The first method is a background task with a priority lesser than all other tasks. We show that the idle times have a very short length compared with other methods. The reason is that the idle times are scattered. The polling server is a better solution because a task is allowed and this task consume the maximum possible time. But it doesn't take advantage of all idle times. Therefore, the *Slack* appears as the better solution here. Indeed it aggregates idle times by delaying the task.

We compare in Figure 5 the total length of sleeping periods function of a power down threshold. The same parameters of simulation as previously has been used except that the CPU load was take the values 40%, 60% and 80%. We show that the total time during which the processor can be powered down is longer when we use *Slack* algorithm than when we use other methods. An exception occurs when the system is very loaded and the suspend threshold is small. But we can consider that this situation is critical and it is not acceptable in our application context. We show by these

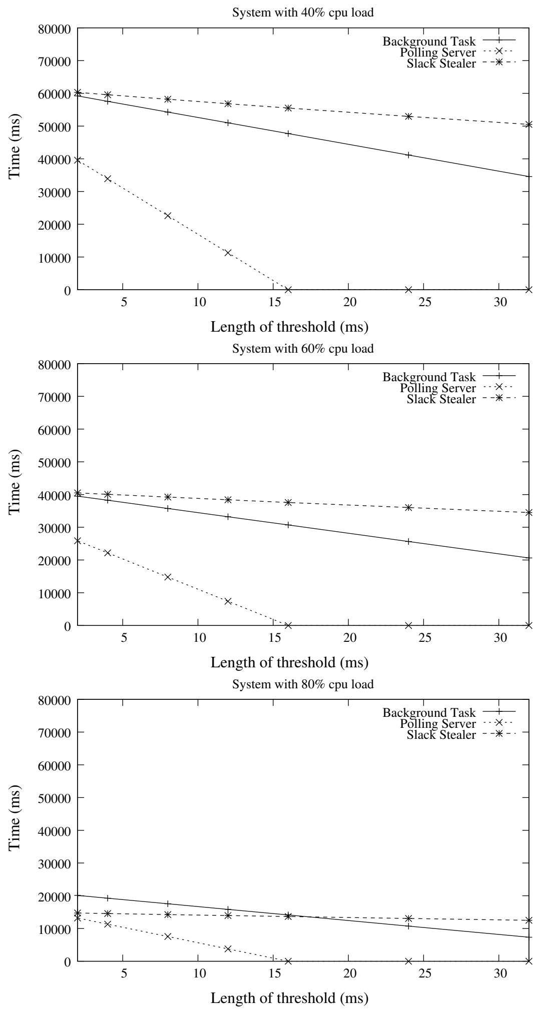

Figure 5. Total length of sleeping period.

results that we can optimize the power saving of a processor by aggregating idle times using an approximate *Slack* algorithm. If we consider that to start a idle processor represents a non-negligible cost, this solution can be a good alternative to passive power saving.

5. Conclusions

We presented in this paper a model to abstract a sensor node. We used this model to consider a sensor node as a real-time system. We proposed different methods to manage idle period in such systems and power down the processor during this period. We proposed to aggregate this idle period to increase the period of cutting off using slack stealer algorithms. After the presentation of different slack stealer algorithms, we proposed to use a approximate algorithm and we explained when power down the processor. Finally, we presented results of simulation which show the viability of our assumptions.

This first study of sensor model (in terms of acquisition and processing) allowed us to go to the decentralization of sensors signal processing in order to increase the system reliability and in the same have a good energy autonomy.

6. Future Work

We have to implement this solution on real time embedded sensors to perform measures on a real system. A first step will be to consider a single sensor and validate the solution. A second step could be to find a distributed solution for the wireless ad-hoc sensor networks. If the sensors communicate with a sink, they can be turned off without impact on the connexity of the network. But in an ad-hoc network, we can power down a node only if the graph stay connected.

Acknowledgments

The authors gratefully acknowledge the contribution of French National Research Association (ANR) in the framework of the QuoVADis Project.

References

[1] R. I. Davis. Scheduling slack time in fixed priority preemptive systems. Technical report, Dec. 08 1993.

[2] D. Istrate, E. Castelli, M. Vacher, L. Besacier, and J.-F. Serignat. Information extraction from sound for medical telemonitoring. *Information Technology in Biomedicine, IEEE Transactions on*, 10:264–274, April 2006.

[3] D. Istrate, M. Vacher, and J.-F. Serignat. Generic implementation of a distress sound extraction system for elder care. In *28th IEEE EMBS Annual International Conference*, pages 3309–3312, New York City, USA, Aug 30-Sept 3, 2006.

[4] D. Masson. Real-time systems simulator (rtss), 2006.

[5] D. Masson and S. Midonnet. Slack time evaluation with rtsj. In R. L. Wainwright and H. Haddad, editors, *SAC*, pages 322–323. ACM, 2008.

[6] H. Medjahed, D. Istrate, J. Boudy, J.-L. Baldinger, B. Dorizzi, I. Belfeki, V. Martins, F. Steenkeste, and R. Andreao. A multimodal platform for database recording and elderly people monitoring. In P. Encarnação and A. Veloso, editors, *BIOSIGNALS (2)*, pages 385–392. INSTICC - Institute for Systems and Technologies of Information, Control and Communication, 2008.

[7] S. R. Thuel and J. P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *IEEE Real-Time Systems Symposium*, pages 22–33. IEEE Computer Society, 1994.