Quality-Of-Service: The Role of Energy

Geovani R. Wiedenhoft, Arliones Hoeller Jr., Antônio A. Fröhlich

[1]Laboratory for Software and Hardware Integration
Federal University of Santa Catarina
PO Box 476 – 88049-900 – Florianópolis, SC, Brazil

{grw,arliones,guto}@lisha.ufsc.br

Abstract. *This work explores energy as a parameter for Quality of Service (QoS) of embedded systems. We argue that it is not enough to guarantee other QoS metrics (e.g., processing, communication, memory) if doing so the system runs out of battery and is unable to complete its computations. In this context, we developed a scheduler for imprecise tasks that, knowing battery's life-time and tasks' energy consumption, prevents tasks from executing its optional parts to ensure that tasks finish. A prototype was developed for EPOS (Embedded Parallel Operating System) using the system's power management infra-structure.*

1. Introduction

Embedded systems are computational platforms dedicated to perform a known set of tasks. Many of these systems are, due to the mobile nature of their applications, powered by batteries. Considering this feature, it is important for the mobile embedded system to be able to manage energy consumption. Some projects [Yuan 2004, Scordino and Lipari 2004] explore the integration of mechanisms that manage energy consumption in works that assure quality of service (QoS) of applications while minimizing energy consumption. These works, however, focus their energy consumption management decisions on traditional QoS metrics for processing, memory and communication. In this work we argue that it is not enough to guarantee other QoS metrics if, by doing so, the system runs out of battery and is unable to complete its computations.

In this work we propose to use energy as a QoS parameter, where to guarantee energy QoS metrics is considered more important than to guarantee other metrics. The proposed approach expects the developer to define the period that the embedded system must be operational. By monitoring battery lifetime, a scheduler is able to decrease QoS level to reduce energy consumption and enhance system lifetime. In our prototype, QoS control of applications was inspired by imprecise computing [Liu et al. 1994], that divide tasks into two subtasks, one mandatory execution flow, and another optional flow. This prototype was implemented in EPOS [Marcondes et al. 2006], a component-based embedded operating system. EPOS provides a software infrastructure which allows applications to implement proper energy consumption management [Hoeller et al. 2006]. The scheduler prevents optional subtasks from executing when the energy budget is bellow a pre-defined limit, thus reducing system's processing demands. This reduction creates more idle periods in the system, and the scheduler can use the before mentioned infrastructure to reduce energy consumption by stopping or slowing down system components during these idle periods.

2. Related Work

GRACE-OS was presented by Yuan [Yuan 2004] as an energy-efficient operating system for mobile multimedia applications. This system uses a cross-layer adaptation technique to guarantee QOS on systems with adaptive software and hardware. It uses real-time scheduling, CPU clock scaling and kernel-based profiling to dynamically manage energy consumption. GRUB-PA [Scordino and Lipari 2004] is some similar to GRACE-OS, but what GRACE-OS provides support only for soft real-time tasks, GRUB-PA also supports hard real-time tasks.

Other projects explore trade-offs between QOS and energy consumption through adaptations in the applications. An operating system that performs this sort of management is ODYSSEY [Flinn and Satyanarayanan 1999]. The system monitors the applications' resource usage and alerts the applications which resources are becoming scarce, expecting the application to stay in a lower QOS level until the resources became available again. Another operating system that supports application adaptation is ECOSYSTEM [Zeng et al. 2002]. This system assign costs that applications must pay to use system resources (e.g., access to memory, network or disks) through a currency, called *currentcy*. The system distributes *currentcies* periodically to tasks accordingly to an equation that defines the discharge rate that the system battery can assume to force the system to last for a defined period of time. This allows applications to adapt their execution based on their *currentcy* balance.

3. QOS In Terms Of Energy

This work developed an approach to guarantee that the systems' batteries lifetime can last enough time to assure the execution of at least the essential tasks defined by the embedded system developer. When the system detects that batteries will not last long enough to satisfy a previously defined expected system lifetime, it starts to decrease QOS levels in order to save energy. The proposed system controls QOS using techniques inspired by imprecise computing mechanisms [Liu et al. 1994].

Imprecise computing is a scheduling technique originally proposed to satisfy timing requirements of real-time tasks through decreasing QOS levels. Imprecise tasks are divided into two subtasks: a mandatory one and an optional one. The mandatory portion is the core of the task, and must always be available for execution. The optional portion, however, performs non-critical actions, and can be prevented from executing if the system is lacking resources. Imprecise tasks worsen quality of results by not executing optional portions of work in order to guarantee that no execution deadline will be lost. With this division, imprecise computing unites real-time computing and best-effort techniques for, respectively, the mandatory and optional subtasks.

In order to use imprecise computing techniques to provide QOS in terms of energy, some adaptations had to be performed. In our prototype, the main goal of these adaptations is to switch the focus of the scheduling mechanisms from timing constraints to energy constraints. In this context, one of the adaptations is to change the scheduler to, instead of guaranteeing individual deadlines, guarantee the requested battery lifetime. After this adaptation, task's individual deadlines no longer constitute restrictions and timing constraints no longer exist in this approach.

The other adaptation is the substitution of the execution time estimate by the energy consumption estimate. The scheduler uses this estimate to verify if the required lifetime will be sustained and, eventually, act to prevent optional subtasks from executing. Specific tasks' energy consumption estimates allow the scheduler to select which optional subtasks to suspend.

4. Implementation

A prototype was developed in order to test the proposed approach using EPOS (Embedded Parallel Operating System) [Marcondes et al. 2006]. EPOS is a framework of hierarchically organized components that generates application-specific runtime support systems. We extended EPOS to support imprecise tasks and modified its scheduler to properly handle the conditional execution of the optional portion of each task. For the sake of this work EPOS also provides abstractions to access system batteries' information, such as charge level and lifetime estimates. In the current version, lifetime estimate is done solely by batteries discharge rate analysis.

We studied two ways to implement imprecise tasks support in EPOS. Both expect the programmer to specify, when creating a thread, two entry points: one for the mandatory subtask and another for the optional subtask. The difference is on the way the operating system will handle these subtasks. The first alternative we studied is the creation of two threads, i.e., one execution flow to handle the mandatory part and another flow to handle the optional part. The creation of these threads are performed by the system, thus being transparent to the application programmer. The drawbacks identified in this implementation are the overhead due to the increased number o threads in the system and the set of complex coordination problems that exist to assure a correct execution sequence for the mandatory and optional parts of the task.

The second one, that is adopted here, provides the application programmer with a way to specify in the mandatory subtask's implementation the point where the optional subtask should execute by calling the `optional` method of the task. In this approach, the programmer can position the optional part before, after or in the middle of the mandatory task. The application programmer also has to define how long the system is expected to last by passing a timestamp to the system through the `system_lifetime` method. This implementation also allows the application programmer for changing the pointer to the optional function, thus enabling the task to change its optional part if desired. It is done by the `optional(FunctionPointer)` method. Figure 1 presents an UML class diagram for the implementation of imprecise task in EPOS.

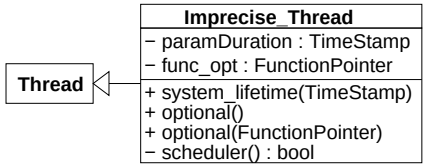

Figure 1. UML class diagram for imprecise task in EPOS

Every time the `optional` method is called, a test is performed to check if the system is able to sustain the current workload without running out of battery before the required lifetime is achieved. When the scheduler realizes that the requested lifetime cannot

be achieved with the current workload, it prevents optional subtasks from executing, thus reducing energy consumption. If later the scheduler realizes that the requested lifetime can be achieved again, the system restarts the execution of optional subtasks. Figure 2 presents an UML sequence diagram that shows how the system handles the execution of the optional subtask called by mandatory subtask.

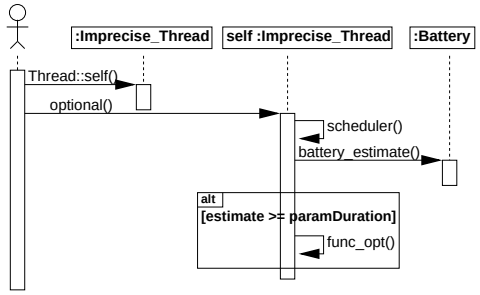

Figure 2. UML sequence diagram for the execution of an optional subtask.

5. Conclusion

This work proposed an approach to satisfy a defined battery lifetime in mobile embedded systems. This approach was *inspired* by the concept of imprecise tasks, i.e., tasks which are divided into mandatory and optional subtasks. A prototype was built in EPOS to test the approach. The proposed scheduler analyzes battery discharge to define whether a predefined battery lifetime can be achieved or not. If the scheduler realizes that the expected lifetime cannot be achieved, it prevents optional subtasks from executing, thus consuming less energy and enhancing battery lifetime. The implementation is functional and future work will walk towards an approach to guarantee both energy consumption and timing requirements.

References

Flinn, J. and Satyanarayanan, M. (1999). Energy-aware adaptation for mobile applications. In *ACM SOSP '99*, pages 48–63, New York, NY, USA. ACM Press.

Hoeller, A. S. J., Wanner, L. F., and Fröhlich, A. A. (2006). A Hierarchical Approach For Power Management on Mobile Embedded Systems. In *5th IFIP DIPES*, pages 265–274, Braga, Portugal.

Liu, J. W., Shih, W.-K., Lin, K.-J., Bettati, R., and Chung, J.-Y. (1994). Imprecise computations. *Proceedings of the IEEE*, 82(1):83–94.

Marcondes, H., Junior, A. S. H., Wanner, L. F., and Fröhlich, A. A. (2006). Operating Systems Portability: 8 bits and beyond. In *11th IEEE ETFA*, pages 124–130, Prague, Czech Republic.

Scordino, C. and Lipari, G. (2004). Using resource reservation techniques for power-aware scheduling. In *ACM EMSOFT '04*, pages 16–25, New York, USA. ACM Press.

Yuan, W. (2004). *Grace-OS: An energy-efficient mobile multimedia operating system*. PhD thesis, University of Illinois at Urbana-Champaign.

Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. (2002). Ecosystem: managing energy as a first class operating system resource. In *ACM ASPLOS-X*, pages 123–132, New York, NY, USA. ACM Press.