Gerência de Energia no EPOS Utilizando Técnicas da Computação Imprecisa

Geovani Ricardo Wiedenhoft, Antônio Augusto Fröhlich

[1]Laboratório de Integração Software e Hardware
Universidade Federal de Santa Catarina
PO Box 476 – 88049-900 – Florianópolis, SC, Brasil

{grw,guto}@lisha.ufsc.br

Abstract. *This work explores energy as a parameter for QoS in embedded systems that are powered by batteries. The goal is to guarantee that the batteries used in this system can last at least the time required by the application and yet to preserve the deadlines of hard real-time tasks. We propose equations to check at project-time if a given set of tasks is schedulable. At execution-time, a preemptive scheduler for imprecise tasks based on the EDF algorithm prevents the optional subtasks execution if the mandatory subtasks deadlines or the battery lifetime will not be met. A prototype was developed in* EPOS *using power management mechanisms provided by the system.*

Resumo. *Este trabalho explora a energia como parâmetro para QoS em sistemas embarcados que são alimentados por baterias. O objetivo é garantir que as baterias usadas nesses sistemas possam durar no mínimo o tempo requerido pela aplicação e ainda preservar os* deadlines *das tarefas* hard *de tempo real. Nós propomos equações que verificam em tempo de projeto se um dado conjunto de tarefas é escalonável. Em tempo de execução, um escalonador preemptivo para tarefas imprecisas baseado no algoritmo* EDF *impede partes opcionais de executarem caso os* deadlines *das partes obrigatórias ou o tempo de duração da bateria não serão atendidos. Um protótipo foi desenvolvido no* EPOS *com a utilização de seus mecanismos de gerência de energia.*

1. Introdução

Sistemas embarcados são plataformas computacionais dedicados a executar um determinado conjunto de tarefas com objetivos específicos, como monitorar e/ou controlar os ambientes nos quais estão inseridos. Normalmente, esses sistemas apresentam rigorosas limitações em termos das capacidades de processamento e de memória. Além disso, muitos deles, devido à natureza móvel das suas aplicações, são alimentados por baterias com uma limitada carga de energia. Considerando todas essas limitações, é importante que eles sejam capazes de gerenciar seus consumos de energia sem comprometer a execução da aplicação.

O *hardware* dos sistemas embarcados disponibiliza diferentes mecanismos para realizar a gerência do consumo de energia. Dentre eles, são as técnicas de DVS (Dynamic Voltage Scaling) e hibernação de recursos. Alguns trabalhos na literatura exploram a integração dessas técnicas com abordagens que garantem qualidade de serviço (QoS).

A maioria dessas abordagens, entretanto, apenas buscam minimizar o consumo de energia com o foco principal nas métricas tradicionais de QoS, como para processamento, memória e comunicação. Como apresentado em um trabalho anterior [Wiedenhoft et al. 2007a], nós argumentamos que não é suficiente apenas garantir métricas tradicionais de QoS se a carga da bateria termina antes do término das tarefas.

Nós utilizamos a energia como um parâmetro de QoS para atender o tempo de duração do sistema especificado pelo usuário, e com isso, consideramos QoS em termos de energia. Neste trabalho, o objetivo não é apenas reduzir o consumo de energia, mas aumentar a utilidade da aplicação em um sistema com uma limitada carga de energia, garantindo o tempo de duração do sistema e os *deadlines* das tarefas *hard* de tempo real. A abordagem proposta espera que o desenvolvedor defina o período mínimo em que o sistema embarcado deve permanecer operacional. Através do monitoramento do tempo de duração da bateria, o escalonador é capaz de selecionar as tarefas que serão executadas ou pode diminuir os níveis de QoS com objetivo de reduzir o consumo de energia e aumentar o tempo de duração do sistema.

Para alcançar nosso objetivo, o controle de QoS foi inspirado na computação imprecisa [Liu et al. 1994]. A computação imprecisa divide as tarefas em duas partes: uma implementa o fluxo obrigatório e outra implementa o fluxo opcional. O fluxo obrigatório é a parte *hard* de tempo real da tarefa, e deve sempre ser executado com o seu *deadline* atendido. O fluxo opcional é a parte "melhor esforço" da tarefa, o qual é executado apenas se os requisitos temporais podem ser atendidos. O escalonador da computação imprecisa impede a execução das partes opcionais quando existe a possibilidade do *deadline* de alguma parte obrigatória ser perdido, e assim, reduz a demanda por processamento do sistema. Além disso, no nosso escalonador, propomos que as partes opcionais não sejam escalonadas quando o nível de energia não será suficiente para atender o tempo especificado pela aplicação. Esse controle cria períodos ociosos no sistema, que permite ao escalonador usar técnicas de gerência de energia para diminuir o consumo dos componentes durante os períodos ociosos criados.

O escalonador proposto é baseado no escalonador *Earliest Deadline First* [Liu and Layland 1973] (EDF), no qual as tarefas com menores *deadlines* possuem maiores prioridades. Um protótipo desta proposta foi implementado no EPOS [Marcondes et al. 2006], um sistema operacional embarcado baseado em componentes. EPOS disponibiliza um conjunto de mecanismos para a gerência do consumo de energia, desde uma infra-estrutura que permite as aplicações realizarem a gerência apropriada [Hoeller et al. 2006], até um gerente com diferentes modos de operação que realiza a gerência para a aplicação [Wiedenhoft et al. 2007b]. Além desses mecanismos, EPOS provê um sistema de monitoramento da carga da bateria, que informa a energia restante.

2. Trabalhos Relacionados

GRACE-OS [Yuan 2004] é um sistema operacional eficiente em termos de energia para aplicações móveis de multimídia. Esse sistema usa técnicas de adaptações multi-camadas para garantir QoS em sistemas com *software* e *hardware* adaptativos. GRACE-OS combina escalonamento de tempo real com mecanismos de DVS para dinamicamente gerenciar o consumo de energia. Ele foi implementado sobre o sistema operacional LINUX e suporta apenas tarefas *soft* de tempo real. GRUB-PA [Scordino and Lipari 2004] é, de

certa forma, semelhante ao GRACE-OS. A principal diferença é que GRUB-PA suporta tanto tarefas *soft* de tempo real quanto tarefas *hard* de tempo real.

Niu [Niu and Quan 2005] propôs minimizar a anergia consumida para sistemas *soft* de tempo real enquanto garante requisitos de QoS. Esse objetivo é alcançado através de um algoritmo de escalonamento híbrido (estático/dinâmico) que utiliza DVS e através de técnicas de particionamento do conjunto de tarefas em tarefas obrigatórias e em tarefas opcionais. Nesse trabalho, os requisitos de QoS são qualificados pela restrição *(m,k)*, a qual especifica que tarefas devem atender no mínimo *m deadlines* em qualquer *k* liberações de tarefas consecutivas. Em um trabalho semelhante, Harada [Harada et al. 2006] propôs resolver o compromisso entre a maximização dos níveis de QoS e a minimização do consumo de energia. Nesse trabalho, cada tarefa é dividida em parte obrigatória e em parte opcional, e é realizada a alocação de ciclos e freqüência do processador com garantias de QoS.

Outras pesquisas exploram um balanceamento entre QoS das aplicações e consumo de energia através de adaptações nas aplicações visando atender o tempo especificado pelo usuário. Um sistema que utiliza essa técnica é ODYSSEY [Flinn and Satyanarayanan 1999]. ODYSSEY realiza o monitoramento da energia fornecida e da energia necessária para executar as tarefas. Com essas informações o monitor é capaz de selecionar o estado correto entre economia de energia e qualidade da aplicação. Esse trabalho também demonstra como as aplicações podem dinamicamente alterar seus comportamentos (“fidelidade” dos dados) com o objetivo de economizar energia.

ECOSYSTEM [Zeng et al. 2002] é outro sistema operacional que suporta aplicações adaptativas. Esse sistema é baseado em uma “moeda” corrente que as aplicações utilizam para “pagar” (alocar) e utilizar recursos do sistema (CPU, disco, rede), chamada *currentcy*. O sistema distribui *currentcies* periodicamente para as tarefas de acordo com uma equação que define uma velocidade de descarga que a bateria pode assumir para forçar o sistema a durar um período de tempo definido. Isso faz com que as aplicações adaptem as execuções de acordo com seus *currentcies*. Esse modelo unifica o cálculo de energia sobre os diferentes dispositivos de *hardware* e proporciona uma alocação satisfatória de energia entre as aplicações.

3. Conceitos

Este trabalho objetiva garantir que a bateria usada em um sistema embarcado possa durar no mínimo o tempo desejado pela aplicação e ainda preservar os *deadlines* das tarefas essenciais, ou seja, os *deadlines* das tarefas *hard* de tempo real. Nosso escalonador realizar diminuições controladas dos níveis de QoS com objetivo de economizar energia quando é detectada que a carga da bateria não será suficiente para atender o tempo especificado pela aplicação. O controle da diminuição dos níveis de QoS da aplicação é inspirado nos mecanismos da computação imprecisa [Liu et al. 1994], que divide as tarefas em duas partes: uma parte obrigatória e outra parte opcional. O escalonador proposto é baseado no algoritmo de escalonamento EDF (*Earliest Deadline First*).

3.1. Computação Imprecisa

Computação imprecisa é uma técnica de escalonamento originalmente proposta para atender os requisitos temporais das tarefas de tempo real através de diminuições controladas

dos níveis de QoS. O controle dos níveis de QoS realizado pela computação imprecisa diminui a qualidade do resultado, não executando as partes opcionais, com objetivo de garantir que nenhum *deadline* de execução das partes obrigatórias seja perdido.

Com a divisão de cada tarefa em duas partes, a computação imprecisa une a computação de tempo real e as técnicas de "melhor esforço" para, respectivamente, a parte obrigatória e a parte opcional. A parte obrigatória das tarefas gera resultados imprecisos que refletem o mínimo de QoS para garantir que esses resultados sejam úteis. Os resultados imprecisos têm suas qualidades elevadas quando as partes opcionais são executadas, com a geração de resultados precisos.

Na literatura existem diversas possibilidades de aplicações da computação imprecisa, como, por exemplo, o processamento de imagens. Neste exemplo, as partes obrigatórias gerariam uma imagem com uma qualidade mínima aceitável, enquanto que as partes opcionais aumentariam a qualidade dessa imagem. Os algoritmos "a qualquer tempo" são outras possibilidades de aplicações para a computação imprecisa, que incluem: os métodos numéricos, os cálculos de raízes, os cálculos de polinômios, as aproximações numéricas, e entre outros. Esses algoritmos, normalmente, implementam métodos iterativos que refinam os resultados depois de cada iteração. Nesse caso, quanto mais tempo o algoritmo é executado, melhor é a qualidade do resultado. As aplicações de controle&conforto são outras possibilidades para a computação imprecisa. Um exemplo é o monitoramento da temperatura de uma caldeira que pode derreter ou até mesmo explodir caso ultrapasse uma certa temperatura. Nesse exemplo, o controle verifica em períodos específicos a temperatura da caldeira e aciona recursos para diminuir essa temperatura caso ultrapasse um valor. O conforto pode realizar cálculos adicionais nos dados das temperaturas obtidas, como a média das temperaturas analisadas em um determinado período, a contagem do número de vezes que a temperatura chegou a um certo nível, e entre outros.

A partir desse conceito de divisão de cada tarefa em parte obrigatória e parte opcional, a computação imprecisa mostra-se favorável para a utilização em nossa proposta em relação à energia. A figura 1 apresenta uma tarefa que consumiria X unidades de energia obrigatoriamente, e quando dividida em parte obrigatória (Y unidades de energia) e parte opcional (Z unidades de energia) permite a economia de Z unidades de energia caso a parte opcional não seja executada.

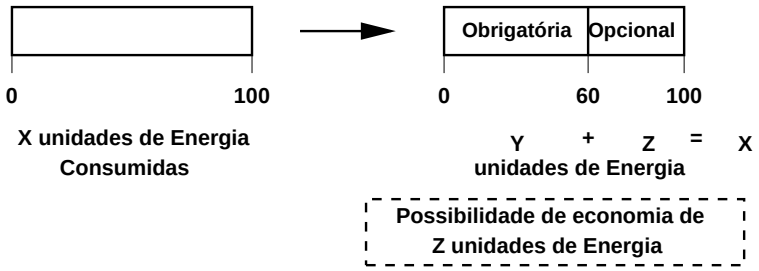

Figura 1. Computação imprecisa em relação à energia consumida.

3.2. EDF

O algoritmo EDF (*Earliest Deadline First*) [Liu and Layland 1973] é um mecanismo de escalonamento tempo real baseado em prioridades dinâmicas e muito utilizado na litera-

tura. EDF distribui maiores prioridades para as tarefas com *deadlines* mais curtos. Em tempo de projeto, um teste de escalonabilidade avalia a possibilidade de alguma tarefa perder o seu respectivo *deadline*. Em tempo de execução, um escalonador preemptivo escolhe a tarefa em estado *Pronto* de mais alta prioridade.

Um teste de escalonabilidade exato para o algoritmo EDF é apresentado a seguir. O sistema de tempo real considerado contém n tarefas periódicas e independentes, $\mathcal{T} = \{\tau_0, \tau_1, ..., \tau_{n-1}\}$. Cada τ_i é caracterizado por três parâmetros, (P_i, D_i, C_i), onde, P_i é o período em que a tarefa i é escalonada, D_i é o prazo (*deadline*) máximo de conclusão relativo ao instante da liberação da tarefa i e C_i é o tempo de execução da tarefa i no pior caso (incluído tempos de espera pela inversão de prioridades). Para este teste é suposto que $\forall\tau_i$, $D_i = P_i$. A utilização U_i de uma tarefa i em termos de processamento é representada pela equação $U_i = \frac{C_i}{D_i}$. A capacidade de um processador é definida como 1, ou seja, 100%. Um sistema com ω processadores possui capacidade ω. Dessa forma, para as tarefas serem escalonáveis no algoritmo EDF, o somatório das utilizações de todas as tarefas deve ser menor ou igual a capacidade dos processadores, ou seja,

$$\sum_{i=1}^{n} \left(\frac{C_i}{D_i}\right) \leq \omega \tag{1}$$

onde $\omega = 1$ para um sistema com mono-processador. Caso $\sum_{i=1}^{n} U_i > \omega$, o processador estará sobrecarregado e as tarefas não são escalonáveis nesse algoritmo.

4. Escalonador

O nosso escalonador, baseado no algoritmo EDF, garante a execução das partes obrigatórias com os seus respectivos *deadlines* atendidos, independentemente do nível de energia do sistema. Entretanto, a execução das partes opcionais não é garantida. Nesta proposta, as partes opcionais são executadas somente se os *deadlines* das partes obrigatórias e o tempo de duração da bateria desejado são sustentados. A figura 2 representa as tarefas que atendem ao parâmetro de energia (tempo de duração do sistema) e as tarefas que atendem ao parâmetro do tempo (*deadline* das partes obrigatórias). A intersecção dessas representações indica as tarefas que podem ser executadas e que serão atendidas em relação aos dois parâmetros desejados (energia e tempo). As tarefas fora dessa intersecção não são escalonáveis neste algoritmo.

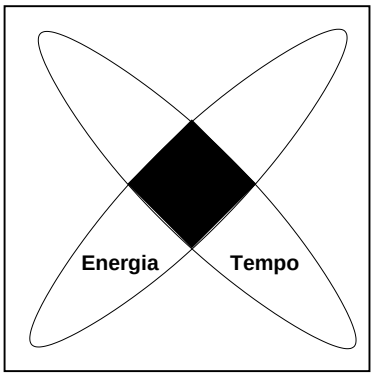

Figura 2. Intersecção entre a energia e o tempo.

O objetivo deste escalonador não é apenas economizar a energia consumida no sistema, pois, caso contrário, a técnica seria simplesmente nunca executar as partes opcionais. A partir disso, o objetivo é atender o tempo especificado pela aplicação com a execução dentro dos *deadlines* das partes obrigatórias e com a execução do máximo possível das partes opcionais, ou seja, otimizar a utilidade da aplicação.

O algoritmo do escalonador proposto neste trabalho é apresentado na figura 3, no qual as partes obrigatórias e opcionais são tratadas como tarefas em termos de escalonamento. Neste algoritmo, π é o intervalo entre medições da carga da bateria que pode ser especificado pelo programador da aplicação e que deve levar em consideração que cada medição também consome energia para ser realizada. Esse intervalo depende do estado de energia da bateria constatado na última medição. Caso a última medição constate que existe energia suficiente e que ultrapasse um determinado *threshold*, o valor do intervalo será maior, pois o sistema não necessita que sejam realizadas medições freqüentes. Entretanto, caso a última medição constate que a energia existente não é suficiente para atender o tempo de duração especificado, as medições devem ser mais freqüentes, pois tarefas opcionais estão sendo descartadas.

```
1: FOR cada tarefa que entra no estado de Pronto:
2:          Calcula o novo deadline absoluto de acordo com o tempo decorrido
3:          Calcula a prioridade baseada no deadline absoluto
4:          Adiciona na fila de acordo com a prioridade calculada
5:
6: FOR cada π unidades de tempo:                    /* π especificado pelo programador e depende do estado de energia */
7:          Afere a bateria
8:          Verifica se existe energia suficiente para atender o tempo desejado pela aplicação
9:
10: FOR cada reescalonamento:
11:          Seleciona na fila a tarefa com estado Pronto de mais alta prioridade
12:          IF, tarefa é hard de tempo real, THEN
13:               Executa a tarefa selecionada
14:          ELSE,                                                    /* tarefa é melhor esforço */
15:               IF, existe energia suficiente para atender o tempo de duração requerido, THEN
16:                    Executa a tarefa selecionada
17:               ELSE,                                               /* bateria não possui energia suficiente */
18:                    Executa a gerência de energia
19:
```

Figura 3. Algoritmo do escalonador proposto.

4.1. Testes de Escalonabilidade em Tempo de Projeto

Como o escalonador proposto é baseado no algoritmo EDF, é possível seguir a mesma lógica para o cálculo da escalonabilidade das tarefas em tempo de projeto com algumas adaptações. Supondo que o sistema de tempo real considerado possua n tarefas periódicas e independentes, $\mathcal{T} = \{\tau_0, \tau_1, ..., \tau_{n-1}\}$, sendo $\forall \tau_i$, $D_i = P_i$. No modelo da computação imprecisa, cada τ_i é dividida em parte obrigatória e parte opcional com tempos de execuções nos piores casos, respectivamente, de μ_i e θ_i. Com isso, o tempo total de execução de τ_i no pior caso é $C_i = \mu_i + \theta_i$. Para atender o nosso objetivo em relação aos *deadlines* das partes obrigatórias, a equação (2) deve ser respeitada

$$\sum_{i=1}^{n} \left(\frac{\mu_i}{D_i} \right) + \sigma \leq \omega \tag{2}$$

onde $\omega = 1$ para um sistema com mono-processador, e σ representa o pior caso de interferências, que inclui: tempo gasto no sistema operacional, nas trocas de contexto, no próprio algoritmo de escalonamento. A equação (2) deve ser atendida para as tarefas serem escalonáveis em relação aos *deadlines* das partes obrigatórias, caso contrário ($\sum_{i=1}^{n} \left(\frac{\mu_i}{D_i} \right) + \sigma > \omega$), o processador estará sobrecarregado.

Com a inclusão do tempo de execução da parte opcional na equação (2), podemos determinar se as tarefas como um todo serão executadas (parte obrigatória e parte opcional). Entretanto, é importante observar que a equação (3) não é um requisito fundamental no nosso algoritmo e será relevante, apenas, quando a equação (2) é válida, caso contrário, as tarefas já não seriam escalonáveis.

$$\sum_{i=1}^{n} \left(\frac{\mu_i + \theta_i}{D_i} \right) + \sigma \leq \omega \tag{3}$$

As partes obrigatórias e as partes opcionais são escalonáveis em relação aos seus *deadlines* quando a equação (3) for respeitada. Caso contrário, uma certa fração χ das partes opcionais é descartada. A equação (4) apresenta como encontrar a fração χ.

$$\chi = \frac{\sum_{i=1}^{n} \left(\frac{\mu_i + \theta_i}{D_i} \right) + \sigma - \omega}{\sum_{i=1}^{n} \left(\frac{\theta_i}{D_i} \right)} \tag{4}$$

O objetivo em relação à energia pode ser alcançado seguindo o mesmo tipo de raciocínio lógico que foi realizado até o presente momento, mas tendo em vista a o consumo de energia das tarefas. O consumo de energia de τ_i no pior caso, E_i, é dado pela soma dos consumos de energia da parte obrigatória e da parte opcional nos piores casos, respectivamente, $E_{\mu i}$ e $E_{\theta i}$, ($E_i = E_{\mu i} + E_{\theta i}$). Nós supomos que, semelhante aos tempos de execuções nos piores casos, os consumos de energia nos piores casos são previamente conhecidos pelo programador da aplicação. Esses valores podem ser obtidos traçando os perfis ou através de outras técnicas. O número máximo possível de execuções, η_i, de τ_i no tempo requerido pela aplicação, T_t, é dado pela divisão entre o tempo requerido e o intervalo de execução de τ_i, ou seja, $\eta_i = \frac{T_t}{P_i}$. T_t é dado pelo programador da aplicação baseado na capacidade da bateria. Com o intuito de atender, no mínimo, as partes obrigatórias das tarefas, temos a equação (5) que indica se o conjunto das tarefas será escalonável em relação à energia.

$$\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times \eta_i}{E_t} \right) + \epsilon \leq 1 \tag{5}$$

Onde E_t é a energia total do sistema (especificação da bateria), ou seja, a capacidade da bateria, ϵ representa o pior caso do consumo de energia de diferentes fatores, como, a energia consumida pelo sistema operacional, pelas trocas de contexto, pelo próprio algoritmo de escalonamento. A capacidade do sistema em relação à energia é

definida como 1, ou seja, 100%. Substituindo η_i de τ_i na equação (5) temos a equação (6).

$$\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times T_t}{P_i \times E_t} \right) + \epsilon \leq 1 \tag{6}$$

As tarefas são escalonáveis em relação à energia no nosso algoritmo se a equação (6) for atendida. Caso contrário ($\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times T_t}{P_i \times E_t} \right) + \epsilon > 1$), o sistema não atenderá ao tempo de duração requerido pela aplicação para esse conjunto de tarefas.

A inclusão da energia consumida pelas partes opcionais no pior caso na equação (6) possibilita que verifiquemos se as tarefas, como um todo (parte obrigatória e parte opcional), serão executadas. Como discutido anteriormente, isso não é um requisito obrigatório e a equação (7) só deve ser calculada se a equação (6) é respeitada, ou seja, partes obrigatórias atendidas.

$$\sum_{i=1}^{n} \left(\frac{(E_{\mu i} + E_{\theta i}) \times T_t}{P_i \times E_t} \right) + \epsilon \leq 1 \tag{7}$$

Caso a equação (7) seja respeitada, todas as partes obrigatórias e opcionais das tarefas são executadas em relação à energia do sistema. Caso contrário, uma determinada fração γ das partes opcionais não será executada, pois o sistema não atenderia ao tempo de duração desejado pela aplicação. A equação (8) fornece a fração de partes opcionais descartadas em relação à energia.

$$\gamma = \frac{\sum_{i=1}^{n} \left(\frac{(E_{\mu i} + E_{\theta i}) \times T_t}{P_i \times E_t} \right) + \epsilon - 1}{\sum_{i=1}^{n} \left(\frac{E_{\theta i} \times T_t}{P_i \times E_t} \right)} \tag{8}$$

Neste algoritmo, o objetivo é atender os dois parâmetros em relação ao tempo e à energia, respectivamente, os *deadlines* das partes obrigatórias e o tempo de duração da bateria especificado pela aplicação. Com isso, (9) é a equação completa do nosso escalonador que deve ser verdadeira para as tarefas serem escalonáveis.

$$\left[\sum_{i=1}^{n} \left(\frac{\mu_i}{D_i} \right) + \sigma \leq \omega \right] \wedge \left[\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times T_t}{P_i \times E_t} \right) + \epsilon \leq 1 \right] \tag{9}$$

As partes obrigatórias das tarefas tem as execuções garantidas no nosso escalonador em relação aos seus *deadlines* e ao parâmetro de energia caso a equação (9) seja respeitada. A fração máxima λ possível de tarefas opcionais perdidas em relação aos dois parâmetros pode ser obtida através da equação (10).

$$\lambda = \max(\chi, \gamma) \tag{10}$$

4.2. Teste de Escalonabilidade em Tempo de Execução

Com objetivo de prover QoS em termos de energia e aproveitar melhor os recursos com a execução das partes opcionais é necessário verificar periodicamente em tempo

de execução se o tempo de duração do sistema requerido pela aplicação, $T_{t\kappa}$, no instante κ pode ser alcançado. Para isso, $T_{t\kappa}$ é recalculado no instante κ de acordo com o tempo decorrido. A energia total do sistema (carga da bateria), $E_{t\kappa}$, também, deve ser recalculada no instante κ . As plataformas dos sistemas embarcados, normalmente, provêm mecanismos para obter a carga da bateria. Os novos valores podem realimentar a equação (11) com o intuito de verificar se $T_{t\kappa}$ pode ser atendido.

$$\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times T_{t\kappa}}{P_i \times E_{t\kappa}} \right) + \epsilon \leq 1 \tag{11}$$

As partes obrigatórias possuem garantia de execução e, assim, as partes opcionais podem ser escalonadas caso a equação (11) seja atendida, pois essa equação indica que existe energia suficiente para atender $T_{t\kappa}$. Caso contrário, as partes opcionais serão descartadas. O escalonador chama um gerente do consumo de energia no tempo em que as partes opcionais estariam em execução, aproveitando o tempo ocioso do sistema para economizar energia. Quando for constatado que a equação (11) volta a ser verdadeira, as partes opcionais das tarefas voltam a ser escalonadas.

Em princípio, nós supomos que a energia consumida pela próxima parte opcional a ser escalonada é irrelevante com relação a energia total do sistema. Entretanto, se essa constatação não for verdadeira necessitamos adicionar na equação (11) a energia no pior caso da próxima parte opcional a ser escalonada. Nesse caso, se a equação não for verdadeira é necessário descartar essa parte opcional e adicionar na equação a próxima a ser escalonada e, assim, sucessivamente até ser encontrada uma parte opcional que respeite a equação, caso nenhuma atenda, o gerente de energia é acionado.

5. Implementação

Um protótipo foi desenvolvido com o intuito de testar a abordagem de escalonamento proposta usando o EPOS (*Embedded Parallel Operating System*) [Marcondes et al. 2006]. EPOS é um *framework* com componentes hierarquicamente organizados para a geração de sistemas específicos a uma determinada aplicação embarcada. O EPOS analisa o conjunto das aplicações dedicadas que ele deve suportar para a geração do sistema, e então, configura o sistema de acordo. Além disso, através das abstrações, mediadores de *hardware* e aspectos, o EPOS permite o desenvolvimento de aplicações totalmente independentes de plataformas. Esse sistema suporta desde uma arquitetura de 32 bits (IA32, PowerPC) até uma arquitetura de 8 bits (AVR8).

No EPOS, todo componente do sistema implementa uma interface uniforme de gerência de energia [Hoeller et al. 2006]. Essa infra-estrutura permite às aplicações interagirem com o sistema para implementar uma apropriada gerência de energia para sistemas embarcados. Com a utilização dessa infra-estrutura, o EPOS disponibiliza um gerente de energia dinâmico com baixos sobrecustos para a aplicação [Wiedenhoft et al. 2007b]. Esse gerente de energia usa heurísticas “replugáveis” para a gerência de energia, permitindo configurabilidade e adaptabilidade para aplicações específicas. O gerente do EPOS possui diferentes modos de operação: a possibilidade de escolha se o gerente será habilitado ou não, a possibilidade de configurar somente os componentes desejados pela aplicação para a gerência, e se o gerente será ativo ou passivo na gerência de energia.

EPOS também disponibiliza um monitor da carga da bateria, o que contribui para alcançarmos os objetivos deste trabalho. O monitor do EPOS é baseado na observação da tensão da bateria para obter a carga da mesma, pois as baterias possuem a característica de ter suas tensões reduzidas conforme a utilização. Entretanto, existem alguns detalhes a serem observados, pois a tensão amostrada não é relacionada linearmente com a taxa de descarga da bateria, o sistema não tem a capacidade de converter toda a tensão fornecida em recurso utilizável e, também, existe uma tensão mínima em que o sistema opera. A figura 4 apresenta o gráfico Tensão X Tempo, no qual pode ser observado que a taxa de diminuição da tensão é variável no tempo. A partir disso, o monitor estabelece uma relação discreta entre a tensão obtida e a carga da bateria, através da divisão das tensões obtidas em 10 fatias de tempo, chamadas épocas, nas quais as tensões possuem diferentes variações, como apresentado no gráfico. Cada época corresponde a uma porcentagem da capacidade nominal da bateria utilizada.

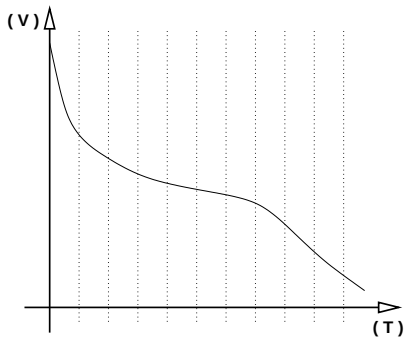

Figura 4. Gráfico da Tensão X Tempo.

O monitor do EPOS não realiza um acompanhamento constante da tensão real da bateria, apesar disso ser possível, pois cada leitura consome energia para ser realizada, além de um sobrecusto considerável para a aplicação. Para diminuir esses efeitos, o monitor utiliza uma estrutura com informações conhecidas previamente que permite acompanhar o consumo de energia de uma forma aproximada. As informações são a respeito das características específicas da bateria e dos consumos de energia pelos componentes de *hardware* do sistema a ser monitorado. No início da execução o monitor verifica a carga da bateria através da tensão, como mencionado anteriormente, e durante a execução atualiza esse valor com as energias consumidas pelos periféricos do sistema.

Nós estendemos EPOS para suportar o nosso algoritmo de escalonamento com tarefas imprecisas e execuções condicionais aos parâmetros de tempo e de energia. As tarefas imprecisas no EPOS foram modeladas com base nas funções monotônicas da computação imprecisa. Nessa modelagem, as tarefas melhoram a qualidade do resultado durante o tempo que permanecem executando e na pior das hipóteses não alteram o resultado. Com isso, a parte obrigatória retorna a solução com o mínimo de QoS necessária para a continuidade da aplicação e a parte opcional faz refinamentos sucessivos nessa solução. O término dessas tarefas pode ocorrer em qualquer momento da execução sem ocasionar problemas de integridade no resultado, assim sendo, o escalonador pode decidir em qualquer instante finalizar a execução da parte opcional. A aplicação fica responsável pela integridade dos resultados através de diferentes métodos, como o uso de bits de controle ou mesmo o uso de *timestamps* da última atualização dos dados.

A implementação das tarefas imprecisas no EPOS foi realizada através da criação de duas *threads*, uma contendo o fluxo de execução da parte obrigatória e outra com o fluxo de execução da parte opcional. O sistema cria essas *threads* de uma forma transparente para o programador da aplicação. Essa abordagem apenas espera que o programador especifique, no momento da criação de uma tarefa imprecisa, dois ponteiros para funções: um para a parte obrigatória e outro para a parte opcional, com os seus respectivos parâmetros.

Na execução, o escalonador sempre escolhe a parte de mais alta prioridade de acordo os *deadlines* como nosso algoritmo é baseado no EDF. As partes opcionais são escalonadas quando não existirem partes obrigatórias com o estado *Pronto* e se houver energia, ou seja, as partes opcionais possuem prioridades inferiores as partes obrigatórias. Caso uma parte obrigatória entre no estado *Pronto* e sua parte opcional ainda não tenha terminado de executar no período anterior, o escalonador imediatamente finaliza a execução da parte opcional. Essas características fazem com que não ocorram perdas dos *deadlines* das partes obrigatórias. O contexto da parte opcional é sempre reiniciado quando ela sai do estado *Pronto* e entra no estado *Esperando*.

O escalonador também atualiza em tempo de execução o $T_{t\kappa}$ com o tempo decorrido no sistema e o $E_{t\kappa}$ com a utilização do monitor de energia do EPOS. Com essas variáveis atualizadas, o escalonador realimenta a equação (11) em períodos π de tempos para determinar se o sistema é capaz de manter a carga de trabalho sem que a bateria termine antes de $T_{t\kappa}$ seja alcançado. π dependerá do resultado da última análise de energia. No melhor caso, a equação (11) é atendida e as partes opcionais podem ser escalonadas. Caso contrário, as partes opcionais serão descartadas e o escalonador executa (nesses intervalos que as partes opcionais estariam em execução) o gerente de energia do EPOS que encontra-se no modo passivo. Nesse caso, além da economia de energia por não executar as partes opcionais, o gerente reduz o consumo global do sistema através do uso de técnicas de gerência de energia. No instante $\kappa + \iota$ em que o escalonador identifique que $T_{t\kappa+\iota}$ pode ser alcançado novamente, o sistema volta a escalonar as partes opcionais.

6. Conclusão

Este trabalho propôs uma abordagem que explora a energia como um parâmetro de QoS em sistemas embarcados móveis para atender o tempo de duração do sistema e, ainda, preservar os *deadlines* das tarefas *hard* de tempo real. Nossa abordagem foi inspirada pelos conceitos de tarefas imprecisas, que são tarefas que podem ser divididas em parte obrigatória e parte opcional. Neste artigo, testes de escalonabilidade em tempo de projeto foram apresentados com o objetivo do programador da aplicação verificar se o conjunto de tarefas utilizado será escalonável no nosso algoritmo com relação aos dois parâmetros desejados, ou seja, tempo e energia. Em tempo de execução, nosso escalonador baseado no algoritmo EDF garante os *deadlines* das partes obrigatórias e com a realimentação da equação de energia verifica se o tempo de duração do sistema exigido será alcançado. Caso algum dos parâmetros desejados seja comprometido, as partes opcionais serão descartadas, ou seja, será realizada uma diminuição dos níveis da qualidade de serviço da aplicação. Um protótipo foi desenvolvido no EPOS, que permitiu a execução de um gerente de energia nos períodos ociosos em que as partes opcionais foram impedidas de executar, consumindo menos energia e aumentando o tempo de duração da bateria.

Referências

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.

Harada, F., Ushio, T., and Nakamoto, Y. (2006). Power-aware resource allocation with fair qos guarantee. In *IEEE RTCSA '06*, pages 287–293, Washington, DC, USA. IEEE Computer Society.

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, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61.

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.

Niu, L. and Quan, G. (2005). A hybrid static/dynamic dvs scheduling for real-time systems with (m, k)-guarantee. *rtss*, 0:356–365.

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

Wiedenhoft, G. R., Hoeller, A. S. J., and Fröhlich, A. A. (2007a). Quality-Of-Service: the Role of Energy. In *9th Workshop on Real-Time Systems*, pages 107–110, Belem, Brazil.

Wiedenhoft, G. R., Hoeller, A. S. J., and Fröhlich, A. A. (2007b). Um Gerente de Energia para Sistemas Profundamente Embarcados. In *IV Workshop de Sistemas Operacionais*, pages 796–804, Rio de Janeiro, RJ, Brazil.

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. ACM.