社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－EMB－2 (1)
2006／9／27

組込システム資産移行を支援する仮想周期実行(VPE)システムの提案

菅谷 みどり† 追川修一†† 中島達夫†

近年，情報処理機器の高機能化・高性能化は著しく，それに伴いソフトウェアも高度なサービス要求を満たす事が求められている．こうした中，開発効率の向上を目的としたLinuxに代表されるUNIX系のOSへの資産移行の動きがある．しかし資産移行では，特にスケジューリングに対する要求事項を満たす工学的な手法が存在しない．そのため資産移行に伴う調整やソフトウェア変更に起因するソフトウェアの品質低下が問題となっている．本研究はVPE（仮想周期実行）システムを提案する事で，資産移行時に適切なリアルタイム性能と応答性の保証を行える事を示す．本文では，仮想周期実行システム内容と提案と設計について述べ，有効性を論じる．

VPE: Virtual Periodic Execution for Embedded System

MIDORI SUGAYA SHUICHI OIKAWA†,†† and TATSUO NAKAJIMA†

Recently, by the rapid growth of embedded area, software recyclicity is recognized as an important issue especially when they are moved to the other operating systems with keeping performance requirements. We proposed VPE (Virtual Periodic Execution) which provide a procedure to support the scheduling with keeping the real-time and worst case responsiveness. In this paper we introduce the system and discuss its availability.

1. はじめに

近年，情報家電機器，電子制御機器の高機能化・高性能化により，組込みシステムは，従来の制御機器から情報処理機器へ適用分野が拡大している．また，ネットワーク接続やGUI操作などのサービスが求められており，ソフトウェアは急激な規模の拡大・複雑化傾向にある．また，開発期間の短縮化により，効率的な開発が求められている．これらの背景から最近ではLinuxに代表されるUNIX系の汎用的なオペレーティングシステム(以降汎用OS)の利用が拡大し，従来の制御系のリアルタイムOS (以降RTOS)からのアプリケーションの資産移行が進行している．

資産移行においては，資産の有効利用が重要な課題となる．既存資産にできるだけ手を入れず，移行対象のアプリケーションの動作仕様を満たし，さらに移行先アプリケーションの要求仕様も考慮する事が求められている．こうした課題は重要ではありながら，工学的に適切な方法・手法が確立していないため，開発者のコード修正による対処になりやすく，この事がソフトウェアの品質低下を招いている．

本研究の目的は，ソフトウェア資産移行の中でも特にスケジューリングに関するリアルタイム性，応答性を満たすという課題に対し，アプリケーション及びスケジューラを変更せずに要求を満たす手法を提案する．これを仮想周期実行(Virtual Periodic Execution，以降VPE)と呼ぶ．

本論文では，第2章にて，組込みシステムの課題について述べ，第3章にて資産移行に関する検討について述べ，第4章で提案手法であるVPEについて述べる．第5章にて設計，第6章で今後の課題，第7章で結論を述べる．

2. 課題

2.1 スケジューリング課題

本研究では，従来のRTOS上で動作していたアプリケーションを，汎用OS上で動作させる事を資産移行と呼ぶ．資産移行時には，主に次の問題に対処する

† 早稲田大学理工学術院
Waseda University, Department of Computer Science
†† 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Deaprtment of Computer Science, Graduate School of Systems

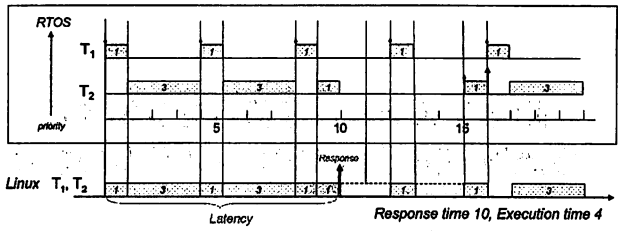

図 1 RTOS から汎用 OS へのスケジューリング対応付けの例
Fig. 1 Example scheduling with real-time and time-sharing tasks

必要がある．1. タスクモデルの違い，2. API の違い，3. スケジューリングの違いである．1 では，RTOS タスクをプロセス，またはスレッドにマップする事で，2 は互換ライブラリの利用などで対処可能である．しかし，3 のスケジューリングの違いについては，工学的な裏付けに基づく対応が不十分である．

ここでスケジューリングとは，システムの動作目的に応じ最適な CPU リソース配分の順序及び時間を決定するものとする．スケジューリングの違いとは，従来の RTOS 上で行われてきたスケジューリングと，情報機器上で求められるスケジューリングの差異を指す．

情報機器と呼ばれる組込みシステムは，携帯電話，PDA, カーナビなど複雑な処理や複数の資源要求を満たすことが期待されている．特に，音声や動画再生など，周期的な時間制約を持つアプリケーションには，ソフトリアルタイム保証が必要である．また，ユーザからの入力を受け付けるための応答性も，重要なスケジューリング要求である．さらに，リアルタイム性と応答性を同時に実現させる事がスケジューリングの課題となる．

2.2 スケジューリングの問題点

課題への対処にあたり，移行前の RTOS と移行後の汎用 OS のスケジューリングモデルの差異を示す．RTOS ではスケジューリングの対象となる実行単位はタスクである．汎用 OS ではプロセスである．RTOS 上のタスクのスケジューリングはほぼ固定優先度であるのに対し，汎用 OS では固定優先度と動的優先度が用いられる．汎用 OS 上の固定優先度のプロセスはリアルタイム (以降 RT) とされ，高い優先度を割り振られる．これに対して，動的優先度のプロセスは，タイムシェアリング (以降 TS) とされ，RT プロセスよりも低い優先度が動的に割り振られる．汎用 OS で RT は，常に TS プロセスよりも高い優先度となる[7]．

汎用 OS 上で動作するアプリケーションの多くは優先度が低い TS ポリシーでスケジューリングされる．その中には GUI を扱う Xserver など応答性を要するアプリケーションも含まれる．こうした TS でスケジューリングされる低優先度アプリケーションは，RT 優先度を持つアプリケーションと同一システムで動作させた場合，応答性が悪化する問題がある．

図 1 に，RTOS 上のスケジューリング単位であるタスクを，Linux の RT プロセスにマップし，その他の TS プログラムを同時に動作させた例を示した．RTOS 上で動作する 2 つのタスクを T_1, T_2 について，周期，実行時間，デッドラインを {T, C, D} とし，値をそれぞれ $T_1 = \{4, 1, 4\}$, $T_2 = \{16, 7, 16\}$ とする．この時の CPU 利用率は $U = \{\sum_{i=1}^{m} C_i/T_i\}$ より，$1/4 + 7/16 = 0.68$ となる．

汎用 OS 上では，TS プロセスの優先度は，RT プロセス優先度より低い．そのため，TS プロセスの最悪応答時間は，ある時間内 ($t0$ から t) でリリースされた全ての RT プロセスの仕事量の総和値 (最悪応答時間) の定義である次式を利用できる[4]．

$$Workload_m(t) = \sum_{i=1}^{m} \lceil t/T_i \rceil C_i = t \qquad (1)$$

ここでは，$0 \leq t \leq D_m$ が成立する t が全ての m に対して存在すれば $m = i$ の際に求められた時間が i の最悪応答時間を示す．図 1 に示したように RTOS 上のタスクを汎用 OS 上の RT プロセスに対応づけし，全ての RT プロセスを同時にリリースした際の最悪応答時間を求めると，(1) より，$Workload_2(10) = \lceil 10/4 \rceil \times 1 + \lceil 10/16 \rceil \times 7 = 10$ となる．これは汎用 OS 上で動作する TS プロセスの最悪応答時間となる．

2.3 従来の対処方法

汎用 OS 上でリアルタイム性能と応答性を同時に満たすために，従来は，RTOS 上のタスクを汎用 OS 上の RT プロセスに対応づけした上で，TS プロセスの応答性を考慮する手法が取られてきた．TS の応答性を考慮する具体的な方法には (1)RT プロセスの CPU の利用率を下げる (2)TS プロセスに,RT プロセスよりも高い優先度をつけるといった対処がある．

対処 (1) に対応するためには，RT アプリケーション側の書き換えが一般的である．例えば，プログラムの実行時間や，周期が長い場合には，その部分の処理を分割することで実行時間，周期を調整する．

また，対処 (2) では，応答性が必要な箇所で TS プロセスの優先度が一時的に RT プロセスよりも上昇するように優先度を調整する．しかしこの場合，RT の優先度を持った TS プロセスは，RT スケジューリングの対象となるため，リアルタイムのスケジューリングも調整する必要がある．また，必要な箇所で優先度操作をするため，アプリケーションの改変が必要である．

2.4 アプリケーションの改変に伴う問題

このように従来は，スケジューリング調整において，アプリケーションの改変が行われてきた．しかし改変は，制御・処理構造の変更作業などが発生するため，困難な作業であり，手間，コスト上昇の要因となる．また，修正作業では，洩れや抜けなどの人為的なミスが混入しやすく，ソフトウェアの品質の低下につながる．こうした修正は，ソフトウェア全体の規模の増大に比例して増加するため，同様に修正に含まれるミスも増加するため，極力回避しなくてはならない．

スケジューラの改変の場合，修正による品質低下以外にも，既存スケジューラの挙動に依存して開発されているアプリケーションの動作に影響がでる恐れがある．それらを回避するためには，長期に渡るテストが必要となり，開発期間へ影響を及ぼす恐れがある．

こうした問題を避けるためにも，移行時は既存資産にできるだけ手を入れずに移行を行う事が重要な課題となる．

3. 資産移行に関する検討

本研究では，資産移行を支援するシステムを提供するにあたり，満たすべき条件を次の 3 点とした．

(1) 既存の資産の改変を行わない
(2) RT アプリケーションの時間制約
(3) TS アプリケーションの応答性

条件 (1) における既存の資産は，移行側の RTOS 上のソフトウェア資産と，移行先のソフトウェア資産が含まれる．本節では，これらの条件を満たすためのスケジューリング理論を検討する．理論検討にあたり，スケジューリング理論に沿い，スケジューリングの実行単位をタスクとし，周期処理が必要なタスクを周期タスク，それ以外を非周期タスクとする．RT プロセス，TS プロセスはそれぞれ周期タスク，非周期タスクに対応するものとする．

3.1 Rate Monotonic(RM)

RM は，周期タスク数が複数となった場合であっても，全体の CPU 利用率が約 69%以下の場合には，スケジューリングが可能である事が証明されている[2]．また，m 個の周期タスクがスケジューリングされる際の最悪応答時間の計算は 2.2 節 (1) 式より導出でき，条件 (3) の応答性が予測可能である．

RM は周期処理を前提としたタスクを対象としており，非周期タスクの応答性を特に考慮しておらず，非周期タスクに対しては周期タスクの空き時間を割当てる．そのため，特に (1) の改変は不要である．しかし，その応答性については，RM は周期タスクの CPU 利用率を約 69%最大まで利用した場合，決して高いとはいえず，条件 (3) を満たしているとはいえない．

3.2 Earliest Deadline First (EDF)

先に述べたように RM は，CPU の利用率は約 69%までしか保証しない．これに対し，EDF ではリアルタイム性能を満たしつつ，利用率を 100% $\left(U = \sum_{i=1}^{m} C_i/T_i \leq 1\right)$ まで利用することができる[2]．応答性に関しては，周期タスクの空き時間を利用することができるが，最悪応答時間を一意に求められない問題があり，予測可能性が低い[2]．また，オンラインスケジューリングであるため，スケジューラの改変が必要となり条件 (1) に不適合である．

3.3 Deferrable, Sporadic Server

条件 (2),(3) の両立を前提とした理論としては，Deferrable Server[6]，Sporadic Server[1] 方式がある．これらの理論は，周期的なタスクのスケジューリングの時間制約を満たしつつ，非周期タスクの最悪応答時間を改善することを目的としている．具体的には，非周期タスクを処理する非周期的なタスクサーバをたて，それを周期タスクとともにスケジューリングの対象とする．これらのサーバは優先度を最大とすることで，非周期タスクが到着した場合の応答性を保証することができる．Sporadic Server は，周期をまたいだ場合の容量の確保も行う柔軟性がある．

しかし，これらのサーバ系のアルゴリズムでは，非周期タスクはサーバタスクへ対応づけや時間管理が必要となる．そのため，対象となる非周期タスクが増大するほど設計の複雑さが増大する．これは，RM や EDF のように非周期タスクが，周期タスクの空き時間を利用する場合と対照的である．また，対応づけのための仕組みをアプリケーション側もしくはスケジューラ側で持つ必要があり，条件 (1) に適合しない．

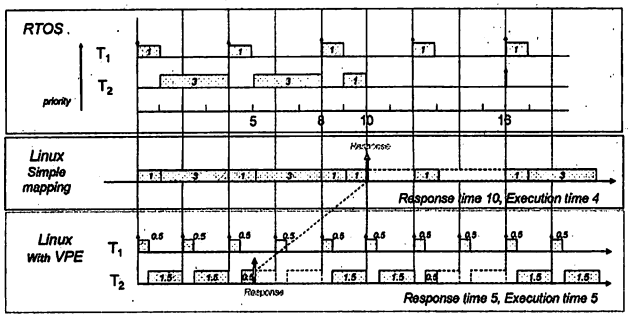

図 2 VPE によるスケジューリング例
Fig. 2 The VPE Scheduling

4. Virtual Periodic Execution(VPE)

4.1 VPE の提案

3.1 節で，我々は RM 応答性が十分改善しない事について述べた．しかし，RM では，非周期タスクは周期タスクの実行時間の空き時間を利用するため，特別な改変が不要である．さらに，非周期タスクの最悪応答時間は，周期タスクの実行時間の総和でもあるため，その周期と実行時間の時間を分割することで改善が可能である．RM において，周期及び実行時間の変更が可能であることは，既に Liu らによって示されている[3]．本研究では，周期タスクの周期と実行時間の時間の分割により応答性を改善する手法を仮想周期実行 (Virtual Periodic Execution：VPE) とし，これが条件 (1),(2),(3) を満たせると考えた．

VPE における分割では，分割する値を k ($k > 0$) とし，分割係数とする．分割係数により周期 T 及び実行時間 C を分割する場合，分割により得られる CPU 利用率は，次の式で示すことができる．

$$U = \sum_{i=1}^{m} (C_i/k)/(T_i/k) \leq m\left(2^{1/m} - 1\right) \quad (2)$$

ここで，k は消えるため，RM と同じ式となる．また，Workload も同様に周期と実行時間を k で分割すると次式となる．

$$Workload_m(t) = \sum_{i=1}^{m} \lceil t/(T_i/k) \rceil C_i/k = t \quad (3)$$

t は $0 \leq t \leq D_m$ とする．この時，t は分割前の 1/k 倍となる．

以上より VPE による周期タスクの周期と実行時間の k による分割は，RM と比較して CPU 利用率は変更せず，最悪応答時間を 1/k 倍改善するといえる．

4.2 VPE スケジューリング例

図 2 は，図 1 同様，RTOS 上で動作する 2 つのタスク T_1, T_2 をそれぞれ $T_1 = \{4, 1, 4\}$,$T_2 = \{16, 8, 16\}$ とした例を示した．VPE 分割係数=2 とした場合，T_1, T_2 の値はそれぞれ $T_1 = \{2, 0.5, 2\}$, $T_2 = \{8, 3.5, 8\}$ となる．この時の CPU 利用率は (2) 式より 0.5/2+3.5/8 = 0.68 となり，分割前の CPU 利用率と等しい．

これに対して，最悪応答時間は $Workload_2$ (5) = $\lceil 5/2 \rceil \times 0.5 + \lceil 5/8 \rceil \times 3.5 = 5$ となり，分割前の TS プロセス最悪応答時間 10 より 1/2 倍の改善となる．

4.3 オーバーヘッドの考慮

本節では，VPE による分割について検討する．VPE では，実行時間の分割を行う際にプロセスの切り替えも伴う．そのため，分割数 k が増大するほど，オーバーヘッドは増大する．ゆえに，分割数を示す k の決定にあたっては，分割数が増大しても，スケジューラビリティが保証されるようにする必要がある．

ある周期内でプログラムの実行時間 C は，実行そのものにかかる時間 C_r と，実行切り替えのオーバーヘッド C_o とに分けられる．ここでのオーバーヘッドは，VPE の割込み制御，スケジューラ起動や処理，コンテキストスイッチ処理等が含まれる．この関係を次式に示す．

$$C = C_o + C_r \quad (4)$$

オーバーヘッドと，分割係数 k との関係を式に反映した場合，C_r は分割による影響は受けないのに対し，C_o は k の増加に従って増加する．ゆえに次式となる．

$$C = kC_o + C_r \quad (5)$$

VPE は RM をベースとしているため，スケジューリング可能条件は次式で与えられる．

$$U = \sum_{i=1}^{m} C_i/T_i \leq m\left(2^{1/m} - 1\right) \quad (6)$$

(5) 式を (6) 式に代入すると (7) 式が得られる．

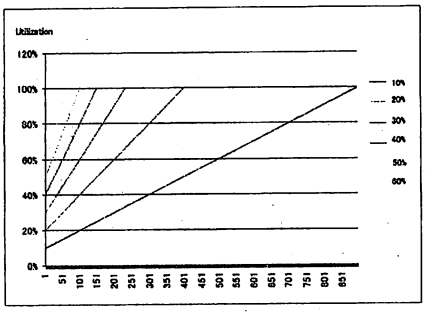

図 3 例 $C_o : C_r(1:100)$
Fig. 3 The case of $C_o : C_r(1:100)$

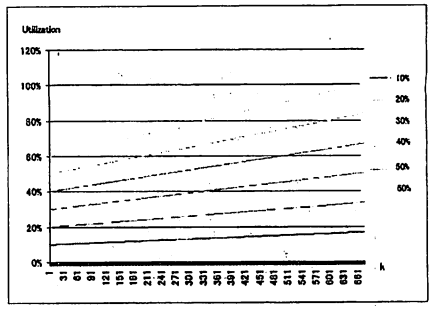

図 4 例 $C_o : C_r(1:1000)$
Fig. 4 The case of $C_o : C_r(1:1000)$

$$U = \sum_{i=1}^{m} (kC_o i + C_r i)/T_i \leq m\left(2^{1/m} - 1\right) \quad (7)$$

VPE による周期，実行時間の分割では上記の条件を満たす k を決定する必要がある．k の値の変化と U の増加率は，タスク数が 1 の最小ケースでは，C_o と C_r に依存する．そのため，比率を各々 1 : 100, 1 : 1000 と固定した場合の U の増加の推移により，U の増加率に対する k の値の変化を知る事ができる．図 3, 図 4 に結果を示した．

結果より C_o と C_r の比率の差が大きいほど k の増加率は低く，CPU 使用率 C_r/T の比率が低く設定されている程 k の増加率は低くなる．

VPE では，こうした結果を前提とし分割値 k を決定する．オーバーヘッドの増分を考慮して k を決定することにより，分割値 k が増大した場合でも，VPE はリアルタイム性能を保証できるようにする．

5. 設　　計

提案に基づき VPE システムを設計した．システムは次の 3 つのレイヤーで構成する．

- Policy Layer（ポリシー層）
- Simulation Layer（シュミレーション層）
- Enforcement Layer（実施層）

3 つのレイヤーを含む全体図を図 5 に示す．Policy Layer は，ユーザからの指定によりサポートするポリシーとパラメータを設定する．Simulation Layer はパラメータに従って分割を行い，オーバーヘッドを考慮した VPE の値を決定・保存する．Enforcement Layer は制御機構で，決定された値を読み取り，値に従って周期と実行時間を制御する．

次に各レイヤーの詳細を述べる．Policy Layer では，VPE の他に，RM と VPE の拡張である Virtual Periodic Execution with Priority Importance(VPE-PI) ポリシーも選択できるものとした．VPE-PI は重要度に応じた優先度設定機能を持つ．

Simulation Layer では，VPE モジュールがパラメータの決定を行う．モジュールの詳細を図 6 に示した．VPE モジュールは，システム特有のパラメータである ISR(Interrupt Service Routine) 時間，コンテキストスイッチなど収集し，その値をオーバーヘッドの値とする．これらの値を元にシステム側でシミュレーションを行いスケジューラビリティチェックの上で，分割が可能であれば再度分割を行い，値を決定する．

Enforcement Layer は実際にプログラムの制御を行う．制御機構は CABI[5] 機構をベースとしている．CABI はプロセスの実行を周期にそって正確に管理するための機構として開発したもので，VPE においては実施機構として機能する．

6. 今後の課題

4 点の課題を次に述べる．1 つ目は，スケジューリングに関する課題として，ロック区間の延長の問題への対処がある．VPE では，プロセスの実行を強制的に分断するため，RT プロセスがロックを保持した状態でブロックした場合，該当ロックを必要とする TS プロセスの遅延が延長する問題がある．これに対して，システム側でロックを発見する機能を設け，ロックが発見された場合はプロセスをブロックしない等の対応が必要である．また，優先度継承についても対処が必要である．

2 点目に分割係数の決定について検討が必要である．4.3 節に示したように，分割値はオーバーヘッドを見越して決定する．しかし，分割数の目標となる値は TS プロセスが必要とする応答性をもとにして決定する事が望ましい．ゆえに TS プロセスが必要とする応答性の検証が必要である．また，システムかユーザのいずれで決定すべきかの議論が必要となる．

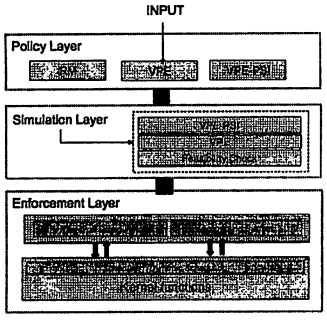

図 5 VPE システムアーキテクチャ
Fig. 5 Whole Architecture of VPE System

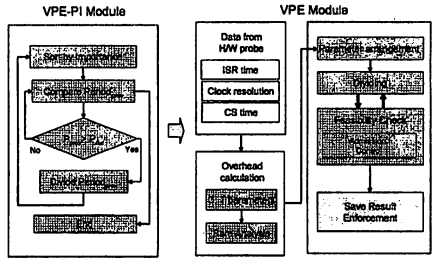

図 6 VPE のモジュール設計
Fig. 6 Module Design in VPE

3 点目として，オーバーヘッドの検証がある．周期の粒度はオーバーヘッドとのトレードオフがあり，線形に粒度を詳細化する事は困難であるとされている[8]．そのため，本研究では理論値に加えてオーバーヘッドを考慮したモデルを追加し，オーバーヘッドを固定値とした．しかし，オーバーヘッドは，実際はカーネル内のプリエンプティブ遅延やロックの保持による遅延，優先度逆転などが生じるケースでは，固定値ではなく，動的な値となる事も考えられる．こうした可能性も含め，検証及びパラメータとして適切な値の検証やモデルについての検討が必要である．

4 点目として，本手法が既存手法よりも実践的であることを示すために，従来の応答性向上のための手法との比較が必要である．比較に際しては，ソフトウェア資産の改変の程度についても議論が必要である．さらに，具体的なアプリケーションによる評価を行う必要がある．

7. 結　　論

本研究は資産移行時のスケジューリングの課題について，VPE システムを提案した．VPE システムは，情報系組込み機器上でのスケジューリングの課題となっているリアルタイム性能と応答性という要求を同時に満たすこと，資産に手を加えないことで再利用性を高め，開発者の負担を減らすことを目的とした．現在，VPE システムの開発及び 6 章に示した課題への対処を行っている．

参 考 文 献

1) B. Sprunt, Aperiodic Task Scheduling for Real-Time Systems, Ph.D. thesis, Dep. of Electrical and Computer Engineering, Carnegie Mellon University (1990).
2) C. L. Liu, James W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Envirnment, Journal of the Association for Computing Machinery, Vol. 20, No. 1, January 1973.
3) Lui Sha, John Lehoczky, Rangunathan Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling., In Proc. 7th IEEE Real-Time Systems Symposium. IEEE Computer Society Press, 1986.
4) Nimal Nissanke , Realtime Systems, International Series in Computer Science, Prentice Hall; 1st edition (September 1997).
5) Midori Sugaya, Shuichi Oikawa, Tatsuo Nakajima: Accounting System: A Fine-Grained CPU Resource Protection Mechanism for Embedded System. ISORC 2006: 72-84.
6) Strosnider, J. K., Lehoczky, J. P., and Sha, L. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Transactions on Computers 44, 1 (January 1995).
7) William Stallings, Operating Systems:(International Edition) 5th, Prentice Hall. July 2004.
8) Yoav Etsion, Dan Tsafrir, and Dror G.Feitelson, Effects of clock resolution on the scheduling of interactive and soft real-time processes, SIGMETRICS 2003.