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

2008－OS－108 (21)
2008／4／24

カーネル由来情報に基づくバックグラウンドタスクの制御

山田 浩史　　阿部芳久*　　河野 健二

慶應義塾大学 理工学部 情報工学科
E-mail: {yamada,yoshiabe}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

PC の高性能化に伴い，ユーザが利用しているにもかかわらず，PC が長時間遊休状態になっていることが知られている．この遊休時間中にウィルスチェックや SETI@home などのバックグラウンドタスク (BG タスク) を動作させることで，PC の使用率を上げることができる．しかし，これらのアプリケーションの制御を適切に行わないと，ユーザのタスク (フォアグランドタスク，以下 FG タスク) のパフォーマンスは著しく低下する．既存の多くのオペレーティングシステム (OS) のスケジューラは優先度に基づいているが，BG プロセスの優先度を下げるだけでは，FG プロセスのパフォーマンス劣化を十分に防ぐことはできない．本論文では，FG プロセスのパフォーマンス劣化を抑えながら BG プロセスを実行する機構を提案する．提案機構はユーザレベルで取得可能な OS の内部情報から，FG プロセスと BG プロセスとの計算機資源の競合状態を見つけ出す．提案機構を Solaris 10 上に実装し，実験を行ったところ，FG タスクのパフォーマンス低下を 16.9% 以下に抑えつつ，BG タスクを実行することができた．

Background Task Regulation using Information Available from Kernel

Hiroshi Yamada　　Yoshihisa Abe　　Kenji Kono

Department of Information and Computer Science, Keio University
E-mail: {yamada,yoshiabe}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

Idle resources can be exploited not only to run important local tasks such as data replication and virus checking, but also to make contributions to society by participating in open computing projects like SETI@home. When executing background processes to utilize such valuable idle resources, we need to explicitly control them so that the user will not be discouraged from exploiting idle resources by foreground performance degradation. Unfortunately, common priority based schedulers lack such explicit execution control. In this paper, we argue that we can reasonably detect resource contention between foreground and background processes and properly control background process execution at the user-level. We infer the existence of resource contention from the approximated resource shares of background processes. To show that our system effectively avoids the performance degradation of foreground activities, we implemented the prototype on Solaris 10 and conducted some experiments. Our prototype keeps the increase in foreground execution time due to background processes below 16.9%, or much lower in most of our experiments.

1 はじめに

PC の高性能化に伴い，ユーザが利用しているにもかかわらず，PC が遊休状態になっていることが知られている．この遊休時間中に，バックアップソフトやウィルスチェックなどのシステム保守を行うアプリケーションを実行することで，ユーザのタスクに干渉することなくシステムを健全に保つことができる．また，遊休資源を SETI@home [1] や Folding@home [2] といった PC グリッドアプリケーションに提供することができる．以下，遊休資源を利用して動作すべきアプリケーションのことをバックグラウンドタスク (BG タスク) と呼ぶ．対して，現在動作しているユーザのタスクをフォアグラウンドタスク (FG タスク) と呼ぶ．

BG タスクを実行することは効果的であるが，FG タスクのパフォーマンスを劣化することなく BG タスクを実行することは困難である．BG タスクを適切に制御しなければ，FG タスクと計算機資源の競合を起こし，FG タスクのパフォーマンスは著しく低下する．既存の多くのオペレーティングシステム (OS) は優先度に基づくスケジューリング手法を採用しているが，BG プロセスの優先度を低く設定しても，FG タスクのパフォーマンスを低下させてしまう．なぜなら，これらのスケジューラはプロセスの飢餓状態 (starvation) を避けるために，FG プロセスの使用している資源を BG プロセスに割り当てることがあるからである．ディスクやネットワークといったデバイス資源の競合は特にパフォーマンス劣化が大きい．そのため，FG タスクのパフォーマンス劣化を避けるために，ユーザが BG タスクを利用しなくなってしまう．これにより，バックアップソフトやウィルススキャンが動作せず，ハードウェアの故障やウィルスへの対策が施されなくなる．また，SETI@home 等のプロジェクトへの参加の妨げとなりうる．

本論文では，FG タスクのパフォーマンス劣化を抑えながら BG タスクを制御する機構を提案する．提案機構は，ユーザレベルで取得可能な OS の内部情報から BG プロセスの資源使用率を見積もり，BG プロセスと FG プロセスとが CPU やディスク帯域，ネットワーク帯域といった計算機資源を競合していないか否かを判定する．競合していると判定すると

*現在，ニューヨーク大学所属

直ちにその計算機資源を利用しているBGプロセスの動作を停止し，FGプロセスにその資源を譲る．停止後もOSの内部状態を監視し，資源が遊休状態になると再びBGプロセスを再開する．例えば，BGプロセスのCPU使用率を見ることで，CPUが競合しているか否かを判定する．BGプロセスのCPU使用率が一定値より低くなると，BGプロセスを停止する．

本研究では，BGタスクを実行したときのFGタスクのパフォーマンス劣化を抑えることに着目している．つまり，提案機構はFGタスクのパフォーマンス劣化を抑えながらBGタスクを可能な限り実行することで，システム全体のスループットを改善する．本研究ではFGタスクの応答時間を保つことを目的としておらず，システムのスループット改善と応答時間の保証とを両立するのは困難であるため，後者は本研究の対象外とする．

提案機構のプロトタイプをSolaris 10上に実装した．プロトタイプを用いてSETI@homeやfsckといった実際のアプリケーションを用いて実験を行ったところ，FGタスクのパフォーマンスの低下を16.9%以下に抑えることができた．

2 背景と関連研究

2.1 優先度に基づくスケジューラ

多くのOSは優先度に基づくスケジューリングを行う．しかし，優先度に基づくスケジューラは，FGプロセスのパフォーマンスを低下させることなく，BGプロセスを実行することは困難である．なぜなら，これらのスケジューラはプロセスのCPU使用率しか考慮していないからである．そのため，ディスク帯域やネットワーク帯域が競合したときには，優先度問わず公平にそれらを割り当てる．これらの資源の競合はプロセスのパフォーマンスを著しく低下させる．

また，既存のスケジューラの多くはプロセスの飢餓状態を回避するために，プロセスの優先度を動的に変更する．そのため，常にユーザの指定した優先度でプロセスが動作するとは限らない．結果として，予めBGプロセスの優先度を低く設定していても，FGプロセスより優先的にBGプロセスが資源を獲得し得る．

2.2 関連研究

これまでにBGプロセスを制御する手法が数多く提案されている．アイドル時間(idletime)スケジューリング[3]は，BGタスクを実行するために，横取り遅延期間(preemption interval)と呼ばれる遅延を用いる．横取り遅延期間はFG要求の処理が終了した後に始まる期間である．その期間中，FG要求は処理されるが，BGタスクの処理は遅延される．横取り遅延期間が終了すると，はじめてBGタスクが処理される．

フリーブロックスケジューリング(Freeblock scheduling)[4, 5]は，FGプロセスのパフォーマンスを劣化することなく，BGプロセスのディスクアクセスを行う手法である．フリーブロックスケジューリングでは，データアクセスに要する時間，ポジショニング時間(positioning time)を利用する．まず，FGプロセスの連続したディスクアクセス要求からポジショニング時間を算出する．次に，この2つの要求の間にBGプロセスの要求を挿入したときのポジショニング時間を算出する．この値がFG要求のみのポジショニング時間より小さければBGプロセスの要求を挿入する．

TCP Nice [6]はBGプロセスの輻輳ウィンドウを制御する機構である．ボトルネックとなっているルータでのパケット送信数を見積もり，TCP Vegasよりも敏感にこれらのボトルネックに反応し，BGプロセスの輻輳ウィンドウを小さくする．これにより，FGプロセスが利用できる帯域幅を確保する．

MS Manners [7]はBGプロセスの進捗に基づいて，ユーザレベルでBGプロセスを制御する手法である．BGプロセスの進捗が低下すると，MS MannersはFGプロセスと資源を競合していると判断してBGプロセスを停止する．

SETI@home [1]やFolding@home [2]はスクリーンセーバとして動作することで，FGタスクとの資源競合を避けている．これらはユーザのキー入力が一定時間なければ，ユーザはPCを利用していないと仮定している．この方法はOSやアプリケーションの変更を要さず，広く利用することができる．

オフィス環境やクラスタ環境上での遊休資源を積極的に利用するための手法も提案されている．Condor [8]では遊休状態にあるワークステーションにBGタスクを配置する．BGタスクが配置されたワークステーションにFGプロセスが現れると，他の遊休状態にあるワークステーションにBGタスクを再配置する．Stealthスケジューラ[9]は，FGプロセスのパフォーマンス低下を防ぐために，CPUだけでなく，仮想記憶とファイルキャッシュ領域も優先度に基づいて管理する．

2.3 問題点

上述した手法は導入が困難であったり，適用範囲が限定的であるといった問題点をそれぞれもっており，これらは大きく 4 つに分類することができる．1 つめはユーザの既存の環境を大きく変更する必要がある点である．アイドル時間スケジューリングやフリーブロックスケジューリング，TCP Nice はカーネル内で動作する手法である．そのため，これらの手法を利用するには，ユーザは OS を変更しなければならず，遊休資源を効率よく使うために OS を再インストールする必要がある．

2 つめは実際の資源の利用状態を考慮していない点である．SETI@home や Folding@home は，ユーザのキー入力が一定時間なければ PC は遊休状態であるという仮定に基づき，スクリーンセーバとして動作する．しかし，実際はキー入力がある状態でも遊休状態にある資源は存在し，スクリーンセーバが起動しているときでも FG タスクが動作している可能性はある．また，Condor では FG プロセスが現れると BG プロセスを再配置するが，FG プロセスが利用していない資源は遊休状態のままである．

3 つめは特定の資源にしか適用できない点である．フリーブロックスケジューリングや TCP Nice は特定の資源にのみ適用可能な手法である．そのため，これらの手法を適用できない資源上では，BG プロセスと FG プロセスとが競合してしまい，FG プロセスのパフォーマンスは劣化する．

4 つめは様々なワークロードへの対処が困難な点である．MS Manners は，BG タスクの資源使用状況ではなく進捗に基づいて制御するため，予め BG タスクの挙動を知る必要がある．そのため，未知の BG タスクの適切に制御することは困難である．

遊休資源を効率よく利用でき，かつユーザが遊休資源を積極的に利用するには，以下の 4 つの点を満たす BG タスクの制御機構が必要である．(1) ユーザの環境を変更しないように，制御機構導入時には OS やアプリケーションの変更を要さない，(2) 実際の資源の使用状況を考慮する，(3) 特定の資源に依存しない，(4) 様々なワークロードに対応する．

3 アプローチ

本論文では，前節で述べた 4 つの必要事項を満たしながら，BG タスクを制御する機構を提案する．(1) OS，アプリケーションの変更を要さない点と (2) 実際の資源の使用状況を考慮する点を満たすために，ユーザレベルで BG タスクの資源使用率を推測する．具体的には，提案機構は OS のプロファイリングツールであるプローブ機構を利用して BG タスクの資源使用率を見積もる．例えば，BG プロセスが取得したディスクブロック数からディスク帯域の使用率を見積もる．OS のプローブ機構は近年多くの OS で採用されている [10, 11, 12, 13]．これらのプローブ機構はシステム稼働中にオンラインで挿入可能であり，OS の変更や再起動は必要ない．

また，(3) 特定の資源に依存しない点と (4) 様々な BG ワークロードに対応する点を満たすために，前述したプローブを用いて取得した資源使用率から資源競合を推測する．もし，BG プロセスの使用率が低ければ，FG プロセスと資源を競合しているとみなして BG プロセスを停止する．提案機構は BG プロセスを停止するために，閾値を設けて，それより BG プロセスの資源使用率が低くなると FG プロセスと資源競合が起きているとみなす．BG プロセスを停止することで，FG プロセスのパフォーマンス劣化を抑える．

4 提案機構

4.1 取得する OS 内部情報

本論文では，CPU，ディスク帯域，ネットワーク帯域の 3 種類の資源を対象とし，以下の OS の内部情報を取得し，BG タスクの資源使用率を見積もる．

- CPU: 各プロセスがスケジュールされていた時間．
- ディスク帯域: 各プロセスが同期的にアクセスしたディスクブロック数．
- ネットワーク帯域: ソケットにバインドされたディスクリプタに対する `write()` と `send()` の発行回数．

これらの情報は対象の資源の使用率を正確に表しているものではないが，資源使用率を十分に反映している情報である．実際の資源使用率を表すプロファイラ情報も取得できるが，その取得はオーバーヘッドが高い．そのため，低コストで取得できる情報で，かつターゲットの資源の使用率を十分に反映している上記のプロファイラ情報を用いる．

本研究では非同期なディスクアクセスとネットワーク受信は考慮していない．非同期なディスクアクセスから要求を発行したプロセスを特定することは難しい．そのため，これらの情報を含めると競合状態を適切に検出できなくなる可能性がある．また，BG プロセスのネットワーク受信を制御しても，FG プロセスのネットワーク受信のパフォーマンスが向上するとは限らない．なぜなら，BG プロセスのネットワークパフォーマンス低下の原因が途中経

路のルータによるものである場合，BG プロセスをただ止めてしまう場合がある．本研究ではこれらの事象を対象外とする．

4.2 特定のプロセスの無視

FG プロセスと BG プロセスとの資源競合をより正確に検出するために，提案機構は，BG プロセスの資源使用率を算出するときに，いくつかのプロセスの資源使用率を考慮しない．具体的には，常にシステム内で動作しているが FG タスクではないプロセスである．BG プロセスの資源使用率を計算する際に，これらのプロセスのプロファイラ情報を含めると，BG プロセスの資源使用率は低くなり，FG プロセスとの資源競合が明確に現れない．

例えば，スワッパプロセスや X サーバなどがある．スワッパプロセス (Solaris 10 では *sched* プロセスに相当) は，CPU が遊休状態にあるときに CPU が割り当てられる．また，X Server は FG プロセスや BG プロセスの存在に関係なく，計算機資源を使う．そのため，提案機構ではこれらのプロファイラ情報を，BG プロセスの資源使用率を計算する際に加味しない．

4.3 CPU の遊休状態の検出

提案機構は，BG プロセスの資源使用率が低下すると，FG プロセスと資源競合を起こしていると判断する．そして，BG プロセスを停止して FG プロセスにその資源を利用させる．この動作は，ディスクやネットワークといった，BG プロセスのリクエストが FG プロセスのパフォーマンスを大きく劣化させる資源には有効であるが，CPU に関しては有効でない場合がある．例えば，BG プロセスが CPU 使用率 5%，FG プロセスが CPU 使用率 60% で動作しているとする．この場合，BG プロセスは FG プロセスのパフォーマンスを劣化することなく動作できるので BG プロセスを止める必要はないが，上述の方法では，BG プロセスの資源使用率が低いので BG プロセスを停止してしまう．

CPU の競合が起きたときにだけ BG プロセスを停止するために，スワッパプロセスの CPU 使用率を指標として用いる．提案機構は，もし BG プロセスの CPU 使用率が低いとき，スワッパプロセスの CPU 使用率がある閾値より低いか否かを確認する．もし低ければ，FG プロセスが CPU を要求しているとみなし，BG プロセスを停止する．そうでなければ，BG プロセスを停止しない．現在，この閾値を 25% としている．

4.4 閾値

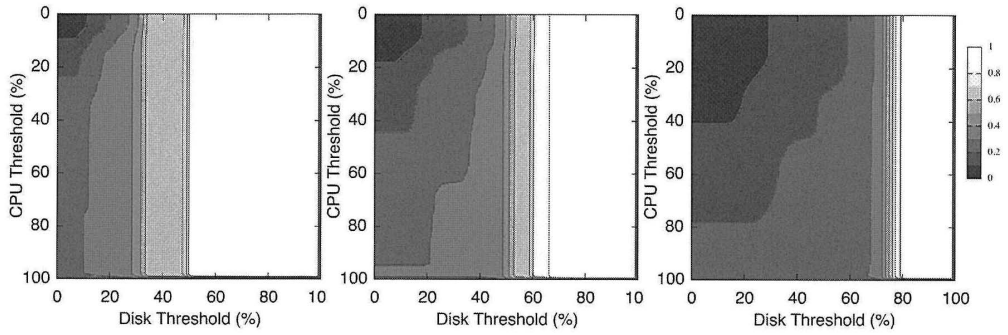
4.4.1 測定方法

BG プロセスを停止する閾値を資源ごとに決めるために，2 台の PC を用いて予備実験を行った．用いた PC は Pentium 2.4 GHz の CPU，512 MB のメモリ，40 GB の 7200 RPM のハードディスクを備えており，互いにギガビットイーサで接続されている．2 種類のプログラムを用意し，実際にプローブ機構を用いて上述したプロファイラ情報を取得した．1 つめのプログラムは，2 GB のファイルを 8 KB ごとに 1 バイト読み込む．このプログラムをディスクベンチマークと呼ぶ．2 つめは TCP コネクション上で Discard 通信を行う．このプログラムをネットワークベンチマークと呼ぶ．それぞれ CPU の使用率を指定して，ディスク帯域とネットワーク帯域の使用量を調整した．指定する CPU 使用率を**資源使用度**と呼ぶ．資源使用度を変更することで，様々なワークロードを生成することができる．

閾値を決めるために，実行する BG プロセス数とプログラムの資源使用度を変更しながら，プロファイラ情報を取得した．BG プロセスは優先度を最低にして，そして同時に FG プロセスを通常の優先度で 1 つ起動し，資源を競合させる．ここで，FG プロセスが 1 つである状況は提案手法が資源競合を一番検出しにくいケースである．もし FG プロセスが多ければ，BG プロセスの使用する資源使用率は減り，判定がしやすくなるからである．FG，BG プロセスの資源使用度をそれぞれ 12.5% おきに，12.5% から 100% まで変更した．また，FG プロセスを実行せずに，BG プロセスのみを実行したときのプロファイラ情報も取得した．これらの取得したプロファイラ情報に様々な閾値を適用して，資源競合を正しく検出しているか否かを検証した．

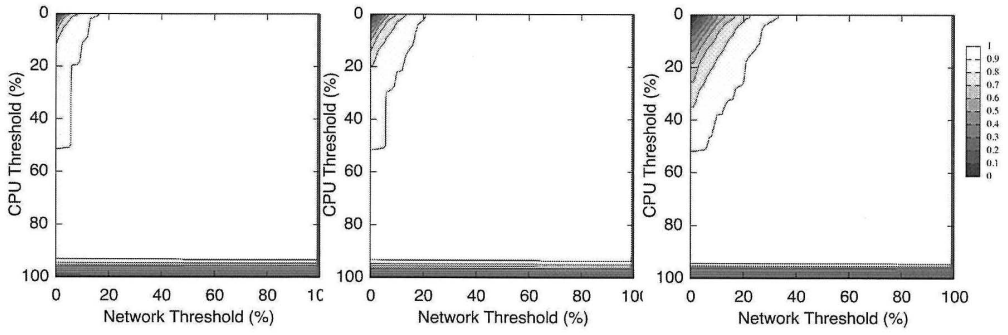
4.4.2 閾値の決定

結果を図 1(a)，(b) に示す．横軸がディスク，ネットワークの閾値，縦軸が CPU の閾値である．グラフの色の濃淡は閾値の精度を表している．薄ければ薄いほど，資源競合を検出する精度が高いことを示す．図 1(a) より，ディスクに関しては，ディスクの閾値が大きく検出精度に影響することがわかった．ディスクの閾値を徐々に大きくすると，CPU の閾値に関わらず，検出精度が向上する．ディスクの閾値が低いときには，CPU の閾値を徐々に大きくすると資源競合の検出精度は向上する．また，図 1(b) より，ネットワークに関しては，CPU，ネットワーク両者の閾値が精度に影響すると言える．CPU，ネッ



Disk: 1 BG process　Disk: 2 BG processes　Disk: 4 BG processes

(a) ディスクベンチマーク



TCP: 1 BG process　TCP: 2 BG processes　TCP: 4 BG processes

(b) ネットワークベンチマーク

図 1: 閾値の決定

トワークの閾値を大きくしていくと，資源競合の検出精度は上昇する．CPU の閾値を 90% から 100% の間に設定すると，BG プロセスの動作に敏感になりすぎて，資源競合が起きていないにも関わらず，資源競合として報告する．実験より，CPU の閾値は 80% から 90%，ディスクとネットワークの閾値はそれぞれ 100% とした．

最適な閾値は PC の性能や環境によって異なるかもしれないが，これらの閾値が大きくずれることはないと考えられる．提案手法は，システム特有の値や PC の性能に依存した値を扱うのではなく，システム全体での BG プロセスの資源使用率を指標としている．そのため，これらの閾値は様々な環境上でも適用可能であることが期待できる．

5 実装

提案機構を Solaris 10 上に実装した．提案機構はクライアントとデーモン，プローブプロセスからなる．クライアントを通して BG タスクを起動することで，提案機構の管理下に置くができる．クライアントはプロセス ID をデーモンに通知し，nice 値を最大値に設定したあと，自身のメモリイメージを BG タスクのメモリイメージにすり替える．デーモンはシグナルを用いて BG プロセスの停止，再開を行う．プローブプロセスはプローブ機構を用いて OS の内部情報を取得する．

プローブプロセスはプローブ機構として Dtrace [10] を用いた．プロセスのスケジューリングされていた時間は，*sched:::on-cpu* プローブでタイムスタンプを保存した値と，*sched:::off-cpu* プローブで得た現在のタイムスタンプとの差をその値とした．ディスクブロック数を取得するために *io:::wait-done* を，`write()` と `send()` の発行回数を取得するために *syscall::write:entry* プローブと *syscall::send:entry* プローブを用いた．プローブプロセスはこれらの情報を定期的にデーモンに通知し，デーモンはこれらの情報から BG プロセスの資源使用率を算出する．

また，BG プロセスの資源使用率を用いて BG プロセスの停止，再開を行うアルゴリズムを 2 種類用意した．1 つめは MS Manners [7] で採用されている Exponetially Increasing Interval (EII) アルゴリズム，

もう 1 つは Idle Period Detection (IPD) アルゴリズムである．EII では，資源が遊休状態であるか否かを確認するために，BG プロセスを短時間実行する手法である．4 節で述べた手法で資源競合を検出し，BG プロセスを停止した後，EII は少し時間が経過した後に再び BG プロセスを再開する．再開直後に取得したプロファイラ情報から資源競合が起きていると判断すれば，BG プロセスを前回停止させた時間の 2 倍の時間停止させる．停止時間は予め決めておいた上限まで増え続ける．もし，BG プロセス再開直後に資源競合が起きていないと判断すれば，停止時間を初期値に戻す．現在，その初期値を 1 秒，上限を 16 秒としている．

IPD は，BG プロセス以外のプロセスの挙動から，各資源が遊休状態であるか否かを調べる手法である．IPD では，CPU が遊休状態にあるか否かを調べるためにスワッパプロセスの CPU 使用率を考慮する．スワッパプロセスに割り当てられた CPU 使用率が高ければ，CPU は遊休状態とみなす．予備実験として，FG タスクを 30 分間起動せずに，スワッパプロセスの CPU 使用率を計測したところ，62% から落ちなかったので，この値を閾値とする．スワッパプロセスの CPU 使用率がこの値より大きいとき，IPD は CPU が遊休状態にあるとみなす．ディスクとネットワークに関しては，リクエストの有無を遊休状態であるかないかの指標とする．全資源が遊休状態であるとみなしたときにのみ，BG プロセスを再開する．

6 実験

提案機構の有効性を検証するために，2 台の PC を用いて実験を行った．PC の設定は 4 節で述べた環境と同じである．

6.1 マイクロベンチマーク

提案機構が様々なワークロードに対応できるかを検証するために，4 節で用いたディスクベンチマーク，ネットワークベンチマークを用いた．FG プロセス，BG プロセスとしてこれらプログラムを起動し，資源使用率と BG プロセスの数を変更しながら両者の実行時間を計測した．比較として，FG プロセスのみの実行時間 (Alone) と提案機構を用いなかった場合，つまり BG プロセスの優先度を下げただけのときの実行時間 (Low Prio.) も計測した．

結果を図 2 に示す．図 2(a) がディスクベンチマーク，図 2(b) がネットワークベンチマークの結果である．図より，提案機構が FG プロセスのパフォーマンス低下を抑えつつ BG プロセスを実行していることがわかる．図 2(a) より，ディスクに関しては，提案機構を用いない場合，FG プロセスの実行時間の増加は最大で約 6 倍となっている．提案機構を用いると，EII では FG プロセスの実行時間の増加は 12.5% 以下に，IPD では 1% 以下となる．EII と IPD の FG プロセスの実行時間を比べると，IPD の方が FG プロセスが早く終了している．これは，EII では資源が遊休であるか否かを検出するために，BG プロセスを繰り返し実行することに起因する．これにより，短時間ではあるが，FG プロセスと BG プロセスが資源競合を起こし，FG プロセスのパフォーマンスが低下する．また，積極的に BG プロセスを実行する分，BG プロセスの実行時間は EII の方が短い．Low Prio. と提案機構を用いたときとの結果を比べると，提案機構は FG，BG プロセス両者のパフォーマンスを改善していることがわかる．これは，提案機構はディスク競合を起こりにくくしているからである．

図 2(b) より，ネットワークに関しては，提案機構を用いない場合，FG プロセスの実行時間は，最悪約 45% 増加する．これに対し，提案機構を用いると，EII では FG プロセスの実行時間の増加は 5.7% 以下に，IPD では 6.1% 以下となる．ネットワークベンチマークの実験では，ディスクベンチマークの実験とは異なり，IPD の方が EII と比べると BG プロセスの実行時間は短い．これは，EII が FG プロセス終了後すぐに BG プロセスを実行できないことに起因する．EII では指数関数的に停止時間を長くするので，資源が遊休状態にあるにもかかわらず，BG プロセスが停止し続けることがある．

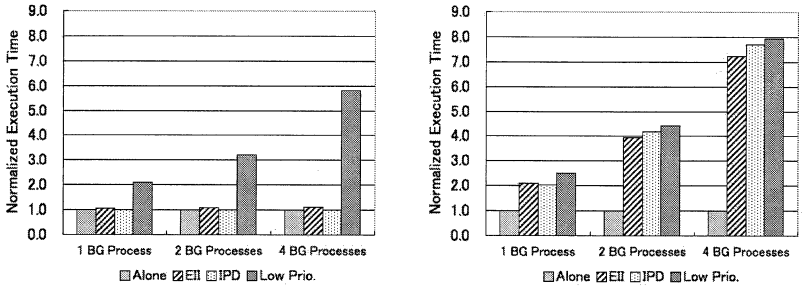
6.2 マクロベンチマーク

実際の状況下での提案機構の有効性を検証するために，実アプリケーションの制御を行った．

6.2.1 SETI@home

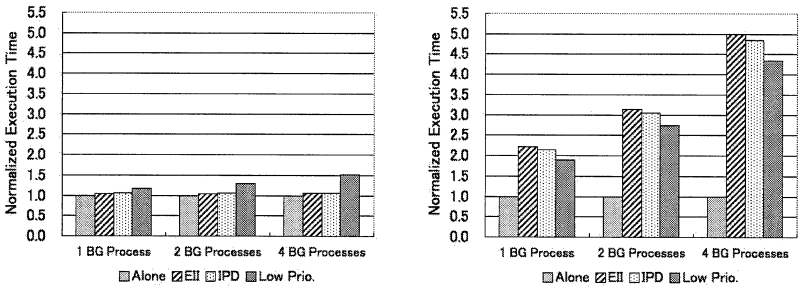
SETI@home[1] を BG タスクとして制御しながら，同時に FG タスクを起動して実行時間を計測した．SETI@home は PC グリッドアプリケーションの一つで，電波望遠鏡のデータを分析するプログラムである．FG タスクとして `fft-wsdom`, `make`, `pcregrep` を用意した．`fft-wsdom` はフーリエ変換を行う CPU バウンドタスク，`make` は Apache 2.2.2 のコンパイルを行う．`pcregrep` は Linux 2.6.16 のソースツリーから指定された文字列を探す．

結果を図 3 に示す．結果より，提案機構が FG タスクのパフォーマンス劣化を抑えながら SETI@home を実行しているのがわかる．SETI@home は CPU バ



FG execution time　　BG execution time

(a) ディスクベンチマーク



FG execution time　　BG execution time

(b) ネットワークベンチマーク

図 2: マイクロベンチマークの結果

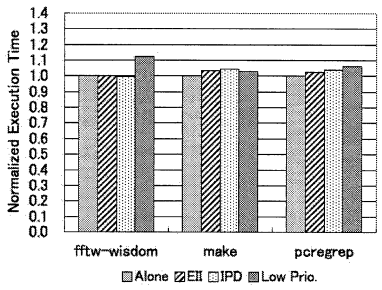


図 3: BG タスクが SETI@home のときの結果

ウンドタスクであるため，`fft-wsdom` への影響が大きい．そのため，提案機構を用いないと，`fft-wsdom` の実行時間は約 12% 増加する．提案機構を用いると，その増加は 3.1% 以下となる．また，提案機構を用いると，`make` の実行時間が最大で 3.1% 増加する．これは `make` のプロセス生成による提案機構のオーバヘッドであると考えられる．

6.2.2 fsck

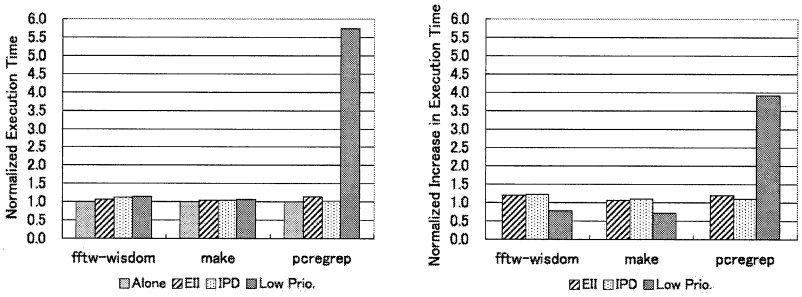
次に，BG タスクを `fsck` として提案機構を動作させた．FG タスクは前節と同じプログラムとし，両者の実行時間を計測した．ここで，`fsck` がチェックをするディスクと FG タスクがアクセスするディスクは同一である．

結果を図 4 に示す．結果より，提案機構が FG タスクのパフォーマンス劣化を抑えながら `fsck` を実行しているのがわかる．`fsck` はディスクバウンドタスクであるため，同じディスクバウンドタスクである `pcregrep` への影響が大きく，実行時間はほぼ 6 倍となっている．提案機構を用いると，EII では最大でも 13.5%，IPD では 3.3% の増加となる．

6.2.3 scp

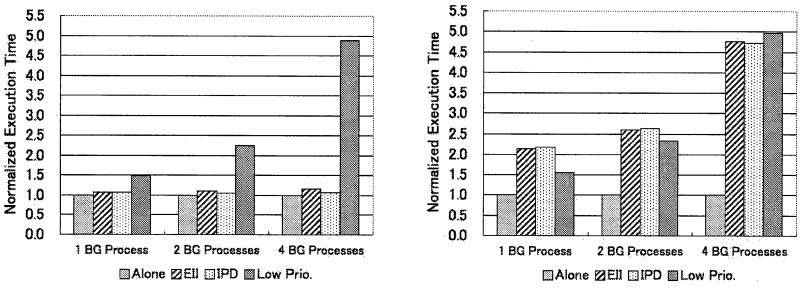
FG，BG タスクを `scp` とし，ネットワーク通信を行うアプリケーションに関しても実験を行った．この `scp` は同一の Linux 2.6.16 のソースツリーを別ホストにコピーする．本実験では，BG タスク数を変化させながら，両者の実行時間を測定した．

結果を図 5 に示す．結果より，提案機構が FG タスクのパフォーマンス劣化を抑えながら BG タスクの `scp` を実行していることがわかる．提案機構を用いないと，FG タスクの実行時間は最大で約 5 倍となる．提案機構を用いると，EII では FG タスクの実行増加は最大でも 16.9%，IPD では 6.7% となる．BG タスクが 4 つのとき，提案機構を用いると BG タスクの実行時間が短くなっている．これは，他の状況よりネットワーク帯域，またコピー先のノード



FG execution time　　BG execution time

図 4: BG プロセスが `fsck` のときの結果



FG execution time　　BG execution time

図 5: FG タスク，BG タスクが `scp` のときの結果

でのディスク I/O の競合が原因であると考えられる．

7　まとめ

本論文では，FG タスクのパフォーマンス劣化を抑えながら BG タスクを実行する機構を提案した．提案機構は，OS をプロファイルするプローブ機構を用いて，ユーザレベルから BG タスクと FG タスクとが起こす資源競合を検出する．検出すると，シグナルを用いて BG タスクの動作を停止して FG タスクに資源を譲る．提案機構のプロトタイプを Solaris 上に実装し，実験を行ったところ，FG タスクのパフォーマンス劣化を抑えつつ，BG タスクを実行しているがわかった．今後は，他の OS 上での有用性を確かめる予定である．

参考文献

[1] Korpela, E., Werthimer, D., Anderson, D., Cobb, J. and Lebofsky, M.: SETI@HOME-Massively Distributed Computing for SETI, *IEEE Computing in Science and Engineering*, Vol. 3, No. 1, pp. 78–83 (2001).

[2] Larson, S. M., Snow, C. D., Shirts, M. and Pande, V. S.: Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology, *Computational Genomics* (2002).

[3] Eggert, L. and Touch, J. D.: Idletime Scheduling with Preemption Intervals, *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, pp. 249–262 (2005).

[4] Lumb, C. R., Schindler, J., Ganger, G. R. and Nagle, D. F.: Towards Higher Disk Head Utilization: Extracting Free Bandwidth From Busy Disk Drives, *Proceedings of the 4th Symposium on Operating System Design and Implementation*, pp. 87–102 (2000).

[5] Lumb, C. R., Schindler, J. and Ganger, G. R.: Freeblock Scheduling Outside of Disk Firmware, *Proceedings of the 1th Symposium on File and Storage Technologies*, pp. 10–22 (2002).

[6] Venkataramani, A., Kokku, R. and Dahlin, M.: TCP Nice: A Mechanism for Background Transfers, *Proceedings of the 5th USENIX Symposium on Operating System Design and Implementation*, pp. 329–344 (2002).

[7] Douceur, J. R. and Bolosky, W. J.: Progress-based regulation of low-importance processes, *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pp. 247–260 (1999).

[8] Litzkow, M. J., Livny, M. and Mutka, M. W.: Condor - A Hunter of Idle Workstations, *Proceedings of the 8th International Conference on Distributed Computing Systems*, pp. 104–111 (1988).

[9] Phillip Krueger and Rohit Chawla: The Stealth Distributed Scheduler, *Proceedings of the 11th International Conference on Distributed Computing Systems*, pp. 336–343 (1991).

[10] Cantrill, B., Shapiro, M. W. and Leventhal, A. H.: Dynamic Instrumentation of Production Systems, *Proceedings of the USENIX Annual Technical Conference*, pp. 15–28 (2004).

[11] Apple - Mac OS X Leopard - Developer Tools - Instruments: Apple - Mac OS X Leopard - Developer Tools - Instruments. `http://www.apple.com/macosx/developertools/instruments.html`.

[12] Linux Technology Center : Welcome: Kprobe. `http://sourceware.org/systemtap/kprobes/`.

[13] Moore, R. J.: A Universal Dynamic Trace for Linux and other Operating Systems, *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pp. 297–308 (2001).