2003－OS－94　(12)
2003／8／4

他のプロセスに与える影響が少ない実行時ミラーリングシステム

柳澤佳里[†]　千葉滋[†]

ファイルシステムのミラーリングではその実行に関与している資源が多いが、そのひとつひとつを適切に考慮したスケジューリングを行う機構を作るのは困難である。そこで、特定のリソースに依らずミラーリングの進捗を見ることでスケジューリングを行う progress-based regulation という手法が提案されている。この手法ではリソースの競合が起きたときに進捗が悪くなることを利用してスケジューリングを行っているが、大域的に進捗を見るためシステムの負荷が変動してからスケジューリングが行われるまでに大きなタイムラグがあるという難点がある。そこで、我々はミラーリングに関連が深い資源である mbuf の使用量の変化をもあわせてみることでこのタイムラグを減らすことを試みた。その結果、若干であるがタイムラグを減らすことに成功した。

Runtime Mirroring System with Low Influence to Other Processes

Yoshisato Yanagisawa[†] and Shigeru Chiba[†]

Since mirroring a file system to a remote backup server uses several system resources, scheduling the mirroring process appropriately is not straightforward. The progress-based regulation is known as a good algorithm for such scheduling. It adjusts the priority of the mirroring process by monitoring the progress of that process since the progress slows down if the resource usage of that process conflicts with that of other processes. However, this scheduling algorithm cannot quicly adjust the priority when the system load suddenly changes. This paper presents a new scheduling algorithm for our mirroring system; it is a hybrid of the progress based regulation and a traditional algorithm based on resource consumption, in our case, network resource. This paper also shows that our algorithm could adjust the priority quicker than previous algorithms.

1. はじめに

近年、インターネットを利用した商取引がますます盛んになる中、そのデータの重要性というのは以前よりも格段に高くなっているだろう。そのような世の中ではデータの保守管理は必須のことであるように思われる。また、インターネット上での取り引きが盛んになるとその扱うデータは時々刻々と目覚しく変わることが予測される。そこでは実行時にファイルシステムのミラーリングを行うことは必須であるように感じられる。本研究はユーザの誤消去などに対処するためにテープにバックアップを行うことや RAID1 を用いてデータの複製を行うミラーリングを対象としたスケジューリングを行うわけではない。本研究のスケジューリング対象は実行時にネットワークを介して遠隔地に複製を作成するようなミラーリングである。ネットワークを介した遠隔地へのミラーリングは地震や火事のような物理的な破壊に耐えうるミラーリングを行うには必須であろう。また、時々刻々と変わるデータのミラーリングを作成するには cron などで定期的にミラーリングを行うのではなく実行時にミラーリングを行うことが不可欠であろう。

しかしながら、ミラーリングでは使用するリソースが多岐にわたるため従来型の CPU ベースのスケジューリングでは HDD のアクセスにおいて他のプロセスとの競合が発生するものの CPU への負荷が少ないという場合に対処できない。さらに、ミラーリングが使うリソース全てに対してスケジューリングを行うような仕組みを入れることは非常に手間がかかりたやすいことではない。そこで、本システムでは progress-based regulation という仕組みを使うことで多岐にわたるリソースのスケジューリングを行えるようにした。

progress-based regulation とは進捗状況が悪い場合

† 東京工業大学院 情報理工学研究科 数理・計算科学専攻
Dept. of Mathematical and Computing Sciences. Tokyo Institute of Technology

に資源の競合が起きているとして何らかのアクションを起こすスケジューリング方法である。このスケジューリングでは検定を用いてアクションを起こすかを決めている。その中にはスケジューリングでは現在の進捗は進捗が良いときの進捗であるという帰無仮説を検定し、検定により進捗が悪いと判定された場合には止めるということをしているものもある。しかし、このprogress-based regulation のみのスケジューリングでは検定の時間にスケジューリングが出来ないという問題や微弱な変化の場合には進捗が悪いと判定できずうまくスケジューリングが働かないという問題がある。

そこで、本研究では既存のリソース監視によるスケジューリングと progress-based regulation によるスケジューリングを組み合わせるスケジューリング技法を用いたミラーリングシステム Tottotto を提案する。Tottto はネットワークサーバーでのミラーリングを想定しているため、まずネットワークの流量の変化を見てスケジューリングを行う。そして、これと組み合わせて progress-based regulation によるスケジューリングも行う。これによりネットワークの流量を監視したスケジューリングが失敗した場合にも progress-based regulation によるスケジューリングにてスケジュールできることを見込んでいる。

我々は NetBSD 1.6 上にこのスケジューリングを利用した実行時ミラーリングシステムを作成した。そして、ネットワークの流量を監視するスケジューリングを用いた場合に progress-based regulation のみによるスケジューリングよりも早くシステムへの負荷が高くなっていることを検出することができた。その結果、progress-based regulation のみでスケジューリングを行っているものに比べて数秒早くスケジューリングを行え、数%の処理量の向上が行えた。

以下、2 章では、既存のスケジューリング技法によるスケジューリングの難点について述べる。3 章では、Tottotto でどのようにスケジューリングを行っているかについて述べる。4 章では、Tottotto におけるスケジューリングおよびミラーリングの実装に付いて述べる。5 章では、progress-based regulation との比較を行った実験に付いて述べる。6 章で関連研究に触れ、7 章で本論文をまとめて、今後の課題を述べる。

2. ミラーリングシステムでのスケジューリングの困難性

2.1 従来のスケジューリング方法

既存の OS には CPU の使用時間に応じたスケジューリングを行う機構が入っている。その機構では重要度の高いものにより多くの回数 CPU 時間を割り当てるという方法でスケジューリングしている。[3] 全てのプロセスは CPU を使って動作するためこのようなスケジューリングは非常に効率の良いものに思えるかも知れない。

しかしながら、この CPU の使用時間を用いたスケジューリングでは CPU の使用時間が少ないにもかかわらず他の資源の使用率が高く、他のプロセスへの影響が大きいようなプロセスに対しては思ったようにスケジューリングを行うことが出来ない。CPU 以外の資源に対しても優先度の高い順に一定時間ずつ割り当てるようなスケジューリングをしようとしたとしても個々に対処するのは困難であるし、それをまじめに実現しようとするとクラスタリングなどの OS の高速化機構が使えなくなり、そのオーバーヘッドによりプログラムの実効速度を著しく損なうことになるだろう。また、個別に対処できたとしてもそれらをどのように関連づけて実際のスケジューリングを行うかを決めるのは非常に困難である。

2.2 Progress-based regulation によるスケジューリング方法

そこで、それには依らないスケジューリング方法である progress-based regulation が必要となる。progress-based regulation はアプリケーションの進捗を見てスケジューリングを行う方法であり、どのような資源をどの程度使っているかを直接みてスケジューリングを行っているわけではない。よって、個々の資源に縛られないスケジューリングを行うことが期待できる。

progress-based regulation は現在の進捗状況を進捗がよいという帰無仮説のもとで検定することによって調べている。検定により調べることで特定のリソースの使用量に依らないスケジューリングであるもののそれだけ安心できるものになっている。この進捗というのは前にスケジューラーが呼ばれて今回呼ばれるまでにされた単位時間あたりの仕事量で表されていて、より短い時間にたくさんの仕事をすればするほど進捗はよくなる。

progress-based regulation では進捗のよいときの進捗 P_{good} を次のような数式で推定して、検定に使っている。

$$P_{good} = \frac{(n-1)P_{good} + P_{now}}{n} \qquad (1)$$

ここで、P_{now} は現在の進捗状況である。また、n はこれまでの進捗とどの程度の割合で混ぜるかを決める定数である。この式はこれまでの進捗の良いときの進捗の値と今回の進捗の指数平均をとっている。このよ

うにすることで進捗が良いときの進捗に漸近的に近づけて行くことが出来る。

Progress-based regulation ではこのように進捗が良いときの進捗を求めた上で進捗がよいという帰無仮説と進捗が悪いという帰無仮説を検定し、進捗が悪いと判断した場合は前回の停止時間の 2 倍の停止時間で停止する。進捗がよいと判断した場合は停止時間を初期値にリセットし、処理を続行する。

しかしながら、検定を行うだけのデータを集めるためある程度のデータが集まるまではなんのスケジューリングも働かないという問題がある。これにより急激に状態が変化したような場合、対処するのに若干の遅れが出る恐れが出る。また、検定を使って判定しているためリソースが使われている度合が軽い場合には他のプロセスに影響を与えているか否かどっちとも判定できない状態が続き、スケジューリングを行うことができない。

また、progress-based regulation のみでのスケジューリングは使用量がある閾値を越えると急激に使えなくなるようなリソースのスケジューリングには使えない。このようなリソースのスケジューリングの場合はリソースの使いはたしのために重要度の低いプロセスのために重要度の高いプロセスがまったくリソースを使えない状況が起こる恐れがある。

よって、このような閾値を越えるリソースを使っているもののスケジューリングをするには progress-based regulation によるスケジューリングに加えてそのようなリソースの使用量をみたスケジューリングを行わなくてはいけない。

3. The Tottotto Mirroring System

本システムでは progress-based regulation に加え、従来のリソースベースのスケジューリングによるスケジューリングを行っている。これにより progress-based regulation によるスケジューリングが遅いという難点とリソース監視によるスケジューリングでそのリソースをあまり使わないがためにスケジューリングし損ねるという問題の両者を解決することを見込んでいる。

3.1 ネットワーク流量監視によるスケジューリング

本システムにおけるリソースベースのスケジューリングは mbuf の使用量を見ることで行っている。mbuf とは BSD 系 Unix でプロセス間通信を行うときに使われるデータ構造で、IP パケットに一つ以上割り当てられる。そのため、ネットワークの流量が多い場合にはより多く消費される。mbuf はその内部に IP パケットなどのデータを持つことが出来るが、あまりに大きいパケットの場合には外部にメモリー領域を確保し、そこにデータを保存する。このようにして確保される領域を mbuf cluster という。この mbuf cluster を管理する領域は kernel 構築時などにその容量が決められるため、システム動作中に変えることが出来ない。しかし、この mbuf cluster の使用量を使い果たすと全くプロセス間通信が出来なくなるという問題がある。

本システムで mbuf を監視することでスケジューリングを行っているのはこのシステムがネットワークサーバーで使われることを想定しているからである。サーバーにおける重要度の高いプロセスはサーバーアプリケーションであり、ミラーリング業務は重要度の低いプロセスに分類される。このサーバーアプリケーションが活発に動くようになるときにはまず mbuf の使用量が急増するという変化が現れるはずである。ゆえに mbuf の使用量の変化を監視することでサーバーアプリケーションの活動の忙しさを見ることが出来るのではないかと考えた。

また、mbuf cluster はその使用量がある閾値を越えたら急に新たな確保が難しくなる資源である。よって mbuf cluster の使用量が一定量を越えた場合には重要度の低いプロセスであるミラーリングを停止させている。その結果、ネットワークを介したミラーリングによって使われていた mbuf cluster が消費されなくなる。このようにして mbuf cluster の使用量が閾値を越えるのを助長するようなネットワークミラーリングを行わないようにしている。

3.2 progress-based regulation によるスケジューリング

Tottotto では 3.1 節で述べた流量監視によるスケジューリングに加え、progress-based regulation によるスケジューリングも行っている。これにより mbuf 以外の資源の競合により重要度の低いプロセスが重要度の高いプロセスの実行に影響を与えている場合でもきちんとスケジューリングすることができる。mbuf の使用量を監視するスケジューリングではこのような場合にスケジューリングを行うことが出来ない。

この progress-based regulation によるスケジューリングは通常の場合で mbuf 以外の資源の使用が極端に高くなってしまった場合に対してのスケジューリングを行うことを考えている。よって、mbuf の使用量が急激に増えるようなある種の攻撃を受けている場合のスケジューリングは流量を監視するスケジューリングにて行う。

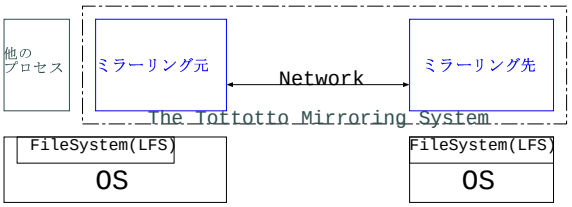

図 1 Tottotto ミラーリングシステムの構成
Fig. 1 Desgin of the Tottotto Mirroring System

ネットワークの流量を監視することによるスケジューリングでは HDD などネットワーク以外の資源の競合で重要度の高いプロセスの実行が妨げられるような場合にはその競合を見落とす恐れがある。しかしながら、progress-based regulation によるスケジューリングを行うことで重要度の低いプロセスが使っている資源が多岐にわたっていてもスケジューリングを行うことが出来る。なぜなら、資源の競合が起こるような場合には重要度の低いプロセスは重要度の高いプロセスと資源の競合を起こし、その進捗が悪化するからである。

4. the Tottotto Mirroring System の実装

本研究では上記のようなスケジューリングを実装したネットワークミラーリングシステムを作成した。このネットワークミラーリングシステムはシステム内でスケジューラーを動かしている。そして、OS のスケジューラーとは別に独自でスケジューリングを行い、他のプロセスに資源を譲るようになっている。

以下ではこの実装について述べる。このネットワークミラーリングシステムはミラーリング元からミラーリング先へネットワークを介してデータを送ることでミラーリングを行っている (**図 1**)。本システムでは NetBSD 上の LFSv2[1] にて作られたファイルシステムを対象としている。LFS とは Log-structured FileSystem といってログ構造に作られたファイルシステムである。LFS ではファイルシステムへの変更は追記することで表されている。ミラーリングのときは LFS のブロック単位である segment 単位でミラーリングを行う。

このシステムではミラーリングの作業の単位としてラウンドとステップの 2 つの表現を用いている。ラウンドというのは前回ミラーリングを行った箇所からの差分を調べ、その差分およびそのときのスーパーブロックをミラーリング先に送る作業を数える単位で、ステップというのは差分を送る作業の中の 1 segment 分を送る作業を数える単位である。

4.1 Tottotto におけるスケジューリング

Tottotto ではラウンドごとにその進捗を報告することで progress-based regultion によるスケジューリングを行っている。ラウンドはラウンド開始時のファイルシステムの内容をミラーリング先にミラーリングするまとまった単位である。進捗の評価はまとまった単位が終わった時点で行うのがよいと考えラウンドごとに progress-based regulation によるスケジューリングを行うことにした。

Tottotto ではネットワーク通信により mbuf を使用する前に現在の使用量を確認し混雑が見込まれる場合には停止するという意図の元、ステップごとにネットワークの流量を監視することによるスケジューリングを行っている。ネットワークの流量を監視するスケジューリングは前の mbuf の使用量と現在の mbuf の使用量を比較し、その増え方が急激である場合には停止するという方法で行っている。なお、停止する時間は急増していることを観測するごとに binary exponential backoff にて増加させていく。なお、ここでの停止時間はラウンドの処理時間に加えられるので、ネットワークの流量監視によるスケジューリングで停止した場合には進捗が若干悪くなる。この動作により progress-based regulation に対してヒントを与えている。

4.2 Tottotto におけるミラーリング方法

Tottotto では LFS[4] にて構築されたファイルシステムをミラーリングの対象としている。LFS の構造を利用することで前回ミラーリングを行った箇所を覚えておけば今回行う箇所の差分を非常にたやすく求めることができる。

Tottotto のラウンドは次のような手順で行われている。

(1) superblock を読み、ミラーリング元で現在書き込みが行われている segment の位置を確認
(2) superblock をミラーリング先に送る
(3) 前回ミラーリングを行った時との差分を求める
(4) 差分を segment 単位でミラーリング先に送り、書き込む
(5) ミラーリング先にて 2 で送られた superblock を書き込む

ここで superblock を送った後に segment を送る操作を行っているのはラウンドの途中でファイルシステムへの変更があったときにファイルシステムの整合性が取れなくなる可能性があるからである。superblock には現在追記による書き込みが行われている segment の位置が格納されている。しかし、上記の手順 3 にて差

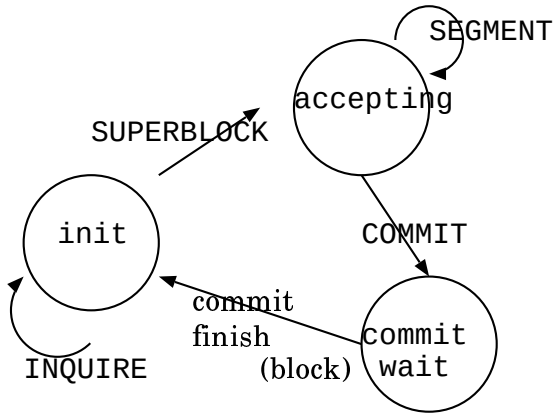

図 2 ミラーリングに於ける状態遷移
Fig. 2 State Transition with Mirroring

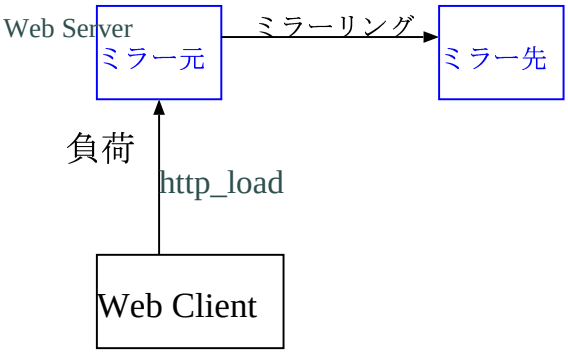

図 3 実験環境
Fig. 3 Experiment Environment

分を求めた後にファイルシステムへの書き込みが行われると superblock には新しく書かれている segment の情報が格納されているものの差分情報は古いままである。よって、そのまま転送を行うと実際にミラーリング元からミラーリング先に送られた segment とその後に送る superblock の情報に不整合が生じてしまう。これを防ぐために先に segment を送っている。

Tottotto は TCP にてミラーリング元とミラーリング先が通信することでミラーリングを行っている。その通信は**図 2** のような状態遷移を経て行われる。Tottotto が起動したときはまず init 状態にある。この状態のときにミラーリング元からミラーリング先に superblock が送られることで accepting 状態に遷移する。accepting 状態はミラーリング元からミラーリング先へ segment 単位でのコピーが行われる状態である。前回からの差分すべてのコピーが終わった時点でミラーリング元は以上のコピーを反映させるために superblock を書き込むよう COMMIT 命令をミラーリング先へ送る。このことにより commit wait 状態へ遷移する。ミラーリング先で superblock の書き込みが終わった時点でミラーリング元へ終わったことを通知し、init 状態に戻る。

5. 実　　験

progress-based regulation を用いたスケジューリングとネットワークの流量を監視するスケジューリングを結合したものが progress-based regulation のみによるスケジューリングに対しどの程度効果があるのか実験により確かめた。また、ネットワークの流量を監視するスケジューリングだけでネットワークをあまり使わない状況下でどの程度スケジューリングが出来るのかも実験により確かめた。

5.1 実 験 環 境

実験はミラーリング元、ミラーリング先、Web Client を準備し、全く負荷をかけない状態で一定時間おいたのちに Web Client から http_load にてミラーリング元に急激に負荷をかけ、http_load によるリクエストの処理量を測った (**図 3**)。これによりネットワークの負荷の変化にどれだけ早く気づいてスケジューリングを行えるかを調べた。なお、ミラーリング元、ミラーリング先、Web Client とも CPU は AMD Athron 1.8GHz を用いていて、メモリーは 1GB を用いている。使っている OS はミラーリング元、ミラーリング先は NetBSD 1.6R で、Web Client は FreeBSD 4.7R である。

5.2 ネットワークの流量が多い場合

まず、ミラーリング元の Web Server に受け取ったデータをファイルに書き込み、送り主にも送り返すような CGI を設置し、それを http_load で呼ぶことでその時間あたりの処理量の変化を調べた。この CGI では処理が終わるときにその時の時刻をファイルに書き込むようにしている。ここではこのデータを元に時間ごとののべ処理量の推移を調べた。

調べた結果は**図 4** の通りである。負荷がかかっていない状態、つまり Tottotto によるミラーリングを全く行わなかった場合はほぼ直線で延びていっていたのでこのグラフはミラーリングを行わなかった場合とそれ以外の場合の差をミラーリングを行わなかった場合の処理量で割ったもので描いた。ここで、nottt というのが全くミラーリングを行わなかった場合で、hybrid というのが mbuf 使用量を監視するスケジューリングと progress-based regulation を併用した場合、pb というのが progress-based regulation のみの場合、noreg というのが全くこれらのスケジューリングの機構を使

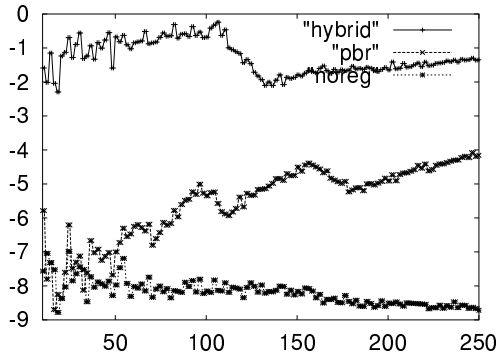

図 4 流量が多い場合の比
Fig. 4 Ratio at high amount of flowing

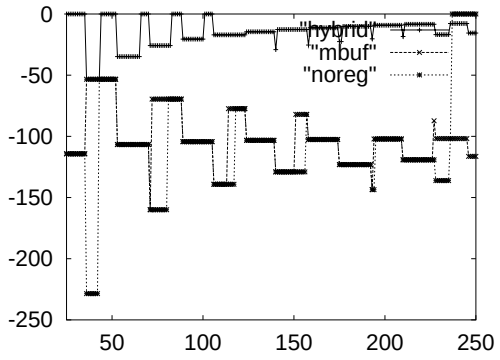

図 5 流量が少ない場合の比
Fig. 5 Ratio at low amount of flowing

わなかった場合である。

この結果より progress-based regulation のみの場合では mbuf の使用量をも見て動作制限を行うものに比べてのべ処理数が若干遅れて延びているということがわかる。これは hybrid の方ではネットワークの流量を見ていたことでより早くスケジューリングを行うことに成功し、Tottotto を全く使わなかった場合と遜色無く CGI が実行できたためであると考えられる。また、progress-based regulation により動作制限をかけたものは負荷が高くなったことを検出することに気づかず progress-based regulation により検出できるほどのデータが揃った時点でスケジューリングが開始されているからであると考えられる。よって progress-based regulation に加えて mbuf の使用量を見ることでスケジューリングをするようにしたことで純粋な progress-based regulation よりも速く反応することが出来るようになったと言えるであろう。

5.3 ネットワークの流量が少ない場合

このように書くと mbuf の使用量による動作制限だけでもスケジューリングが行えるのではないかと思われるかも知れない。それを次の実験にて確かめる。ネットワークの流量が少ないものの他のリソースへのアクセスが多いような場合に、ネットワークの流量を監視することによるスケジューリングのみでどの程度うまくスケジューリングできるか調べた。この実験ではミラーリング元の Web Server に受け取った文字列が含まれているファイルが/usr/src 以下にあるか find(1) にて調べ、調べた結果をファイルに書き込む CGI を設置し、それを http_load で呼んでその処理量を見た。

その結果は**図 5** の通りである。このグラフも Tottotto によってミラーリングをしなかった場合の CGI 処理量と各々の場合の CGI 処理量の差をミラーリングをしなかったもので割った比 (%) で表してある。ここで、nottt というのが全くミラーリングを行わなかった場合で、hybrid というのが mbuf 使用量を監視するスケジューリングと progress-based regulation を併用した場合、mbuf というのが mbuf の使用量を監視してスケジューリングをした場合、noreg というのが全くこれらのスケジューリングの機構を使わなかった場合である。

この結果よりネットワークの流量を監視するスケジューリングのみの場合はネットワーク以外の資源を重点的に使う場合に他のプロセスとの資源の競合を検出することが出来ず正しくスケジューリングすることができないと考えられる。しかしながら、hybrid のグラフが大域的に見て右肩上がりであることからこのような場合でも progress-based regulation を用いたスケジューリングは検出がゆっくりではあるものの着実に他のプロセスとの資源の競合を検出しスケジューリングを行っていけていることが見て取れる。

6. 関 連 研 究

MS-Manner[2] は progress-based regulation を利用してスケジューリングを行う代表的な手法である。MS-Manner ではプログラム内から適当な間隔で testpoint という関数を呼ぶことで現在の進捗状況を報告し、その進捗が悪い場合にはプログラムの停止を行うシステムである。このシステムでは検定によりプログラム進捗の度合を測っているためある程度のデータが集まるまでは検定が行えないという難点がある。

A Feedback-driven Proportion Allocator for Real-Rate Schduling[5] は進捗を見て CPU 時間の割り当てを変えるシステムである。これは従来のプライオリティーベースのシステムへの反省として出来た。

このシステムは OS と Userland のプログラムを分けるというポリシーに従っているので、Userland のプログラム自身の進捗ではなくそれが使っているリソースの処理の進捗状況を見てスケジューリングを行う。しかしながら、この手法はカーネルの改造が必要であるため導入がたやすい方法ではない。

7. おわりに

本稿ではネットワークサーバーにおける実行時ネットワークミラーリングのスケジューリングを通し、progress-based regulation に加えて mbuf の使用量監視というスケジューリングを行うことで素の progress-based regulation によるスケジューリングに比べて速くスケジューリングを行えることを確認した。また、ネットワークの流量を監視するスケジューリングのみの場合にはネットワーク以外のリソースを大量に使うようなプログラムに対処できないことも確認した。

参考文献

1) Netbsd lfsv2. `ftp://ftp.jp.netbsd.org/pub/NetBSD/NetBSD-release-1-6/src/sys/ufs/lfs/R%EADME`.
2) John R. Douceur and William J. Bolosky. Progress-based regulation of low-importance processes. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 247–260. ACM Press, 1999.
3) Maurice J.Bach. *UNIX* **カーネルの設計**. 共立出版株式会社, 1991.
4) Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):26–52, 1992.
5) David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Operating Systems Design and Implementation*, pages 145–158, 1999.