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

2008－HPC－116 (17)
2008／8／6

分散計算機環境InTrigger上の資源共有ルールの評価

鴨志田 良和† 佐伯 勇樹† 田浦 健次朗†

多拠点分散計算機環境 InTrigger では，バッチキュー経由で起動されたプロセスとそうでないプロセスを共存させるために，利用可能な CPU 時間に関するリアルタイムなルールを設けて運用している．ところが，我々が蓄積したリソースの使用状況のログを解析したところ，必ずしもそのルールを守っていないプロセスもある程度存在することが明らかになった．我々はこの状況を改善するため，低優先度で長時間実行するジョブを計算機の遊休時間を利用して実行する Nicer を，バッチキューの情報を考慮するように改良した．改良版 Nicer の性能の評価を行ったところ，他のプロセスが動作している間は 32 秒に 1 度ポーリングを行う設定で，平均 CPU 使用率は 0.12%程度で動作することがわかった．また，バッチキューと Nicer を併用することで，停止中の Nicer プロセスをルールに違反しない形で再開させることができるなど，より柔軟な実行制御を行えることを示した．

Evaluation of Resource-Sharing Rules on Distributed Computing Environment "InTrigger"

Yoshikazu Kamoshida,† Yuki Saeki† and Kenjiro Taura†

InTrigger, a multi-site distributed computing environment, is operated with real-time resource sharing rules assigning available CPU time to each process based on its type of execution to allow coexistence of processes spawned from batch schedulers and directly from shells. According to log data of resource usage on InTrigger, we found a problem that there are a certain number of processes which do not hold by these rules. To solve this problem, we improved "Nicer," a process controller to run long-running low priority processes, to be aware of the state of batch queueing systems. In the result of performance evaluation of Nicer, it runs with average CPU usage 0.12% when its polling interval is set to 32 seconds. In addition, it is shown that more flexible execution control can be achieved by using batch queueing system and Nicer simultaneously.

1. はじめに

大規模なクラスタ計算機などで資源を共有する場合，システムのリソースを管理して適切にジョブの割り当てを行う必要がある．このために，従来より Grid Engine[3], TORQUE[4], Condor[6] などの，バッチキューイングシステムが利用されてきた．バッチキューイングシステムの実行制御の方法は，中央のジョブ制御用のサーバがジョブを待ち行列に保持して，適切なリソースが利用可能になるとジョブの実行を開始するものである．このようなジョブの実行制御下では，獲得した資源を，あたかも専有しているような状態で実行することが可能である．その一方で，リソースの使用率が高く，混雑している場合はジョブをキューイングしたときにすぐに実行が開始される保証がない．例えば，バッチキューイングシステムの制御化では，多数のノードを使うがすぐに終了するようなジョブが実行されにくいという問題がある．また，要求されたリソースが揃うまでの待ち時間のために，本来は他の計算に使用することができるのに使用されないままになってしまうリソースが存在する場合がある．投入されるジョブに予定実行時間を詳細に指定しないと，このようなリソースを効率的に使用することは困難である．

上で述べたような問題はキューを使用しない場合は起こらない．実行結果をすぐに確認することができるし，他のジョブの実行を大きく邪魔することなく長時間かかるジョブを実行したい場合は，プロセスの優先度を低くして実行することができる．このようにすることで，リソースの遊休時間に優先度の低いジョブを実行することができるようになる．一方，このように明示的なジョブの実行制御を行わない場合は，あるジョブが他のジョブの実行を邪魔してしまうことがあるという問題が存在する．

† 東京大学
University of Tokyo

我々は両者の長所を有効に活用するため，インタラクティブな実行，優先度は低いが長時間かかる実行，キュー経由での独立性の高い実行を，低いシステム負荷で共存させるための研究を行っている．これを実現することにより，計算資源の利用効率を損なうことなく，より高い柔軟性でユーザが大規模な並列計算環境を利用できるようになると考えている．

本稿では，バッチキューイングシステムを使ったジョブとそうでないジョブを共存させるための要件を整理し，ジョブを実行するユーザにとって負担の少ない方法でこれを実現するための支援ツールを提案する．

本稿の構成を以下に示す．2章では提案するツールの評価実験を行った並列分散計算環境 InTrigger についての概要と，その上で守られるべきとされている資源共有のルールについて説明する．3章では優先度の低いジョブを長時間実行させるための支援ツール Nicer とその改良について説明し，4章で改良版 Nicer を InTrigger で動作させ，性能評価を行う．そして，5章で本稿のまとめを行う．

2. InTrigger プラットフォームでの資源共有

2.1 InTrigger プラットフォームの概要

InTrigger プラットフォーム[10] は，日本国内 11 拠点の様々な教育・研究機関に展開する並列分散計算のためのテストベッドで，現在 300 以上の計算ノードが稼働しており，今後 20-30 拠点，1000 コア以上の環境を構築することを目指している．分散計算機環境を構築する試みは他国でも行われている．例えば，フランスには Grid'5000[1] があり，オランダには DAS-3[2] がある．大規模な分散環境を構築するという点では InTrigger もこれらの試みと似ているが，拠点数が Grid'5000 や DAS-3 より多いという点が特徴的である．

InTrigger では，拠点が分散していてハードウエアが均一でないという状況の下で，各計算ノードの設定を均質に保つことが必要となる．これを低い管理コストで実現するため，Lucie[9] を使用したネットワーク経由での OS インストールと，Puppet[7] による構成管理の集中化など，遠隔管理や自動管理の仕組みを積極的に取り入れている．

ジョブスケジューラとして TORQUE を拠点毎に動作させているが，TORQUE を経由しないで直接各ノードにログインしてジョブを実行することも許されている．

2.2 資源共有ルール

InTrigger ではキュー経由で投入されたジョブとそうでないものが共存する．そのため，資源の共有について以下のようなルールを定めている．

(1) キュー優先ルール
- キュー以外から起動されたプロセスは，キュー経由のプロセスの邪魔をしてはいけない
  - キュー以外のプロセスが利用可能な CPU 使用率を，(CPU 数 - キューから予約された CPU 数)×100% と定義する
  - キュー経由以外のプロセスの CPU 使用率はこれを越えてはならない

(2) キューの実行時間制限

キュー経由以外のプロセスにも実行時間を配分するため，キューに時間制限を設けて長時間占有させないようにする．実行時間は，並列実行を推奨するために，[sqrt(使用プロセッサ数) × 使用時間] を基準に，この値がほぼ一定になるような時間に設定する．
- 占有できる時間
  - 400 プロセッサ → 1 時間
  - 100 プロセッサ → 2 時間
  - 4 プロセッサ → 10 時間

キュー以外のプロセスはこの制限を受けないものとする．

上記のルールに違反した場合，利用可能分を超過した CPU 時間に比例するポイントをアカウントごとに付与し，ポイントが多いユーザをルールを守らないユーザとして公開することで公平な利用を促すものとする．

2.3 資源共有ルールの評価

我々は上記のルールと，蓄積した過去の各ノードのリソース利用率に関するデータを基に，ルールに違反したプロセスがどの程度あるかを評価した．表 1 は，ある 1 日のあるノードでルールに違反して実行された CPU 時間である．この日は，キュー経由で実行されたプロセスの CPU 使用率 (キュー経由で実行されたプロセスの CPU 時間の和/キューで実際に予約されていた時間の和) が 85.2% であった．一方，表 2 は，別な日の同じノードでルールに違反して実行された CPU 時間である．この日は，キュー経由で実行されたプロセスの CPU 使用率が 12.1% であった．

多くのキュー経由でないプロセスは優先度を下げて実行しているが，キュー経由で実行されたプロセスの CPU 使用率が低い場合はその分キュー経由でないプ

☆ 現時点ではまだアカウント毎のポイントの公表は行っていない．

表 1　CPU 時間 1 秒以上違反実行したユーザ (1)

User	Overrun(sec)
user1	10.21
user2	5.26
user3	2.20
root	1.18
user4	1.02

キュー経由の CPU 時間/キューの予約時間 = 85.2%

表 2　CPU 時間 1 秒以上違反実行したユーザ (2)

User	Overrun(sec)
user1	575.81
user3	6.55
user2	4.86
root	1.32

キュー経由の CPU 時間/キューの予約時間 = 12.1%

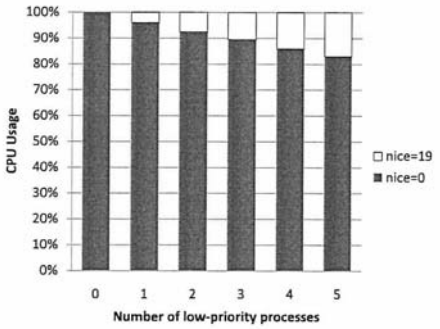

図 1　CPU usage of nice processes

ロセスに CPU 時間が割り当てられることになり，結果としてルール違反の CPU 時間が増加することが判明した．このように，表 2 の日では，ルールが適切に守られているとは言えない状態になっている．我々は，この原因の一つに，ルールを守りつつ十分な CPU 時間を確保して自分の計算を行うことが難しいということがあり，ルールを守らせるためには，ルールを作るだけではなくて，容易にそれを守らせるツール等の仕組みが必要となるのではないかと考えている．

2.4　ルールを守らせるためのツール

キューを経由しないで実行されるジョブのうち，短時間で実行が終わるものについてはリアルタイムモニタリングシステムを活用することにより資源共有ルールを極力守りながら実行を行うことが可能である．InTrigger ではリアルタイムモニタリングシステム VGXP[5] を提供しており，更新間隔 2 秒程度の詳細度で全ノードの負荷の分布を一覧することができる．このような情報から負荷が比較的低い状態にある計算機群を選択してジョブを実行することが可能である．

一方，優先度は高くないが実行時間が長時間に渡るジョブについては，キュー経由で実行されたジョブとうまく共存できるためのツールは提供されていなかった．我々は，この問題を解決するためにプロセスの実行制御ツールである Nicer[8] を改良することにした．

3.　Nicer とその改良

3.1　Nice

実行時間が長期にわたるプロセスは，UNIX の `nice` コマンドや `renice` コマンドを使用してプロセスの優先度を低く変更することにより，他のプロセスの実行を大きく妨げずに実行することができる．しかし，プロセスの優先度を変更するだけでは，優先度が高いプロセスに実行を完全に明け渡すことはできない．例えば 1CPU のマシン上でビジーループを行って CPU を使用するプロセスを，通常の優先度，低い優先度 (nice=19) で実行した場合，低い優先度のプロセスも 3.5%から 4%の CPU を使用する．また，低い優先度のプロセスが増加しても，低優先度プロセス 1 つあたりの CPU 使用時間はあまり変わらないため，低優先度プロセスの数が多いと通常の優先度のプロセスの使用可能な CPU 時間はかなり削られてしまう．(図 1)

3.2　Nicer

Nicer[8] は，`nice` コマンドの拡張で，他のユーザが実行するプロセスへの影響をなるべく抑えつつ CPU の遊休時間を活用するツールである．Nicer は目的のプロセスを最低の優先度で実行させるだけでなく，ノード内の資源の利用状況を定期的に監視し，CPU 時間や最後にコマンドを実行した後のアイドル時間など，利用状況に応じてプロセスを一時停止させたり再開させる機能を持つ．例えば，CPU 時間によるプロセスの実行制御では，ユーザ x が Nicer 管理下のプロセスを実行中のとき，ノード内で使用可能な CPU コアの数を N，ユーザ u の CPU の使用率 (CPU 時間/経過時間) を C_u として，

$$\sum_{p}^{p \neq x} C_p/N > th_{stop}$$

が成り立つと Nicer はプロセスを一時的に停止させる．また，Nicer 管理下のプロセスが停止中の時，

$$\sum_{p}^{p \neq x} C_p/N < th_{start}$$

が成り立つと Nicer はプロセスの実行を再開させる．プロセスの停止と再開は，管理下のプロセスにシグナルを送ることで行うため，停止中のプロセスは `nice` のように CPU を使用することはない．プロセスを Nicer

の管理下に入れるには，

```
nicer <command> [arg...]
```

のように，起動時に nicer の引数にコマンドラインを与えることで，nicer の子プロセスとして実行することで行う．

Nicer を使用すれば，通常のプロセスが実行されていない場合に限り，プロセスを実行することができる．しかし，これだけではバッチキューイングシステムから起動したプロセスの実行を妨害することなくプロセスを共存させることができない場合が存在する．例えば，キューを経由して起動したプロセスが，CPU 時間をあまり消費していない場合，CPU がキュー経由で予約されているにもかかわらず，Nicer はプロセスが実行可能であると判断してしまう可能性がある．これはキューから起動したプロセスの実行を妨害することにつながる．

3.3 Nicer の改良

キュー経由で実行されたプロセスのために割り当てられた CPU は，たとえアイドル状態であっても近い将来使用される確率が高いことから，他のプロセスは使用しないことが望ましい．InTrigger プラットフォームでも，2.2 節のルール (1) で述べたようにキュー経由のプロセスが使用中の CPU を使用すべきでないと規定している．したがって，これを簡単に守ることができるツールを提供することで，より多くのユーザがルールを守りつつ資源を効率的に共有できることになる．我々は本稿で，キューの状態を考慮してプロセスの実行制御を行う改良版 Nicer を提案する．

改良版 Nicer は，実行中のキュー経由のジョブが予約している CPU 数を全体の CPU 数から差し引いてから Nicer のアルゴリズムを適用する点が今までの Nicer と異なっている．改良版 Nicer の動作概要は以下のとおりである．(x は Nicer 管理下に置くプロセスの所有者である．)

- 「非キュー管理下 CPU 時間」の計算
  - ノード内のプロセス CPU 使用率をユーザ毎に集計 (C_u の計算)
  - TORQUE で実行中のジョブがあるユーザの場合，C_u から予約している CPU コア数を引き，この値を C'_u とする (負になった場合は 0 とみなす)
  - この値の和 ($\sum_p^{p\neq x} C'_p$) を「非キュー管理下 CPU 時間」とする
- 「非キュー管理下 CPU 数」の計算
  - ノードの CPU コア数から toque で予約されているコア数を引き，この値 N' を「非キュー管理下 CPU 数」とする
- 上記 2 つの値を CPU 時間，CPU 数として，Nicer のアルゴリズムを適用する

キューを通さないプロセスが，利用可能な CPU 時間を超過している場合は

$$\sum_p^{p\neq x} C'_p > N'$$

が成り立つ．InTrigger における，利用可能な CPU 時間を超過している場合の違反ポイントは，超過時間を比例配分して計算する．すなわち，ユーザ x に配分される超過時間 (違反ポイント) は，

$$(\sum_p^{p\neq x} C'_p - N') \times C'_x / \sum_p^{p\neq x} C'_p$$

となる．

改良版 Nicer を使用することで，バッチキューイングシステムの存在下でも，キューを経由せずに起動した優先度の低い長時間ジョブを共存させることができる．ただし，プロセス停止中でもプロセス自体は存在しているため，低優先度のプロセスでもメモリは消費してしまう．

4. 実　　験

我々は先に述べた改良を Nicer に加え，その性能を評価した．評価環境は InTrigger 上の 1 ノードで，CPU は Intel Core2 Duo 6400(2.13GHz) である．ノード上では 2 つの通常優先度のプロセスをキュー経由で実行している状態で，5 つの CPU インテンシブなプロセスを Nicer から実行した．5 つの各プロセスに Nicer が必要になるため，Nicer も 5 プロセス実行される．この状態で，Nicer とその子プロセスの CPU 時間を計測した．図 2 で，横軸は実行したプロセスの種類，縦軸は CPU 使用率である．評価環境は 2CPU コアの計算機なので，CPU 使用率は 0 から 200%までとしてある．実行したプロセスの種類は，上の行は，通常優先度のプロセスの種類で，“Busy” はビジーループ，“Sleep” は何もしないプロセスである．下の行は Nicer の種類で，“Nicer” は改良版 Nicer，“Nicer-” は改良前の Nicer である．Nicer のポーリング間隔は，プロセス実行時は 2 秒，停止時は 32 秒，$th_{stop} = 33.3\%, th_{start} = 25\%$ とした．

通常優先度のプロセスが CPU インテンシブなものである場合は，Nicer プロセスは正しくそのプロセスの存在を認識し，子プロセスの実行を停止した．CPU 使用率も 5 プロセスの合計でもほぼ 0.6%であった．こ

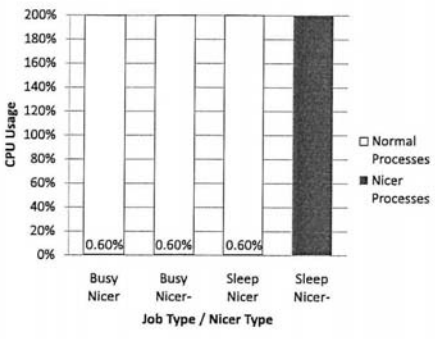

図 2 CPU usage of Nicer processes

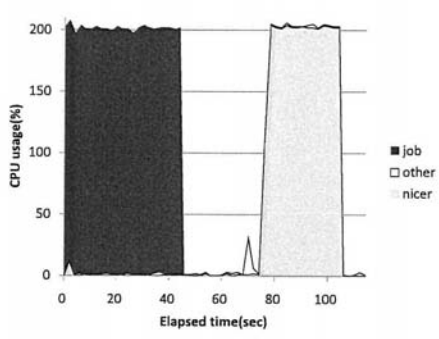

図 3 Waking up a sleeping nicer process

れは，1 プロセスあたりに換算すると 0.12%であり，`nice` を使用したときと比べると十分に低いことが分かる．また，通常優先度のプロセスが CPU インテンシブでない場合でも改良版 Nicer はキュー経由のプロセスによって CPU が使用されていると判断して子プロセスの実行を停止していることが分かる．また，その時のオーバーヘッドも他の場合と同様に低い．

Nicer の仕組みによって，キューを通さないで開始されたジョブが，キューを通して投入されたジョブを大きく妨害することなく実行できることが確認された．

Nicer はユーザごとに使用可能な CPU 時間をカウントして子プロセスの停止と再開の判定を行う．このため，自分の Nicer プロセスが動いている状態で，キューに「何もしない」ジョブを投入することで，そのジョブが実行可能な間は使用可能な CPU 時間が増えることになり，結果として Nicer プロセスを「起こす」ことができる．これを確かめるため，他人のジョブが実行中の時，自分のスリープしかしないジョブをキューに投入し，Nicer ジョブを開始するという実験を行った．図 3 は経過時間約 2 秒の時点で Nicer ジョブを起動したがその時は他人のジョブが実行中だったためすぐに一時停止状態になった．その後，約 46 秒の時点で他人のジョブが終了し，自分のジョブが開始された．そして，約 96 秒の時点から Nicer 経由で開始したプロセスが実行を再開した．しばらく何も動作していない時間があるのは，Nicer が，CPU 使用率が定期的に変動するような他人のプロセスが実行している時には自身の実行を控えるという使用になっており，そのようなプロセスが存在しないことを確認するために時間がかかるからである．

このような Nicer の利用方法で，Nicer プロセスの優先度を一時的に上げたり，`qsub` コマンドでジョブ投入することに対応していないような並列プログラムを他のキュー経由のジョブと同様の高い優先度で実行させることが可能になる．また，実行時間制限を越えた後は通常の Nicer プロセスとして実行させるといった柔軟なプロセス制御も可能になる．

5. おわりに

本稿では分散計算機環境 InTrigger 上で定められているキュー経由のプロセスとそうでないプロセスが共存するためのルールと，現在の利用状況を比較したところ，日によっては 600 秒程度違反しているプロセスが実行されている場合があることが判明した．そして我々はこの原因のひとつにはルールを容易に守らせるためのツールが不足していることがあると考え，キューの状態を考慮する Nicer を実装した．性能評価を行った結果，Nicer 自体のオーバーヘッドは 0.12%程度でキューを通したプロセスを邪魔せずにプロセス共存を行うことが可能であることを確認した．また，Nicer とキューを併用することでより柔軟なプロセスの実行制御ができる可能性を示した．このような柔軟な制御ができるため，InTrigger 以外の並列環境でも Nicer を使う利点があると考えられる．今後，InTrigger 内で改良版 Nicer の利用を普及させ，どの程度資源共有ルールが守られるようになるかを検証していきたい．

今後の課題としては，CPU 使用率以外の情報の活用がある．我々はプロセス情報の他に，netstat, vmstat 等のコマンドで取得できる統計量についても過去のデータを取得している．今後は，過去のログを解析することにより，これらの情報も考慮したプロセスのスケジューリング方針を検討する予定である．

また，ノード間の通信を伴うような並列プログラムを Nicer で実行することも，複数ノードに存在するプロセスを同時に実行する必要があるため，チャレンジングな課題であると考えている．

謝辞　本研究の一部は文部科学省科学研究費補助金特定領域研究「情報爆発に対応する新 IT 基盤研究プラットフォームの構築」の助成を得て行われた.

参 考 文 献

1) Bolze, R., Cappello, F., Caron, E., Dayde, M., Desprez, F., Jeannot, E., Jegou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E. G. and I., T.: Grid'5000: a large scale and highly reconfigurable experimental Grid testbed., *International Journal of High Performance Computing Applications*, Vol. 20, No. 4, pp. 481–494 (2006).
2) DAS-3, http://www.cs.vu.nl/das3.
3) Grid Engine, http://gridengine.sunsource.net.
4) Cluster Resources Inc.: TORQUE Resource Manager, http://www.clusterresources.com/pages/products/torque-resource-manager.php.
5) Kamoshida, Y. and Taura, K.: Scalable Data Gathering for Real-time Monitoring Systems on Distributed Computing, *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2008)*, Lyon, France, pp. 425–432 (2008).
6) Livny, M., Basney, J., Raman, R., and Tannenbaum, T.: Mechanisms for High Throughput Computing, *SPEEDUP Journal*, Vol. 11, No. 1 (1997).
7) Reductive Labs, LLC.: Puppet, http://reductivelabs.com/projects/puppet/.
8) 吉田仁: 共有クラスタの空き時間を効率的に利用するツールの実装と大規模並列計算への応用, 東京大学工学部 学士論文 (2005).
9) 高宮安仁, 真鍋篤, 松岡聡: Lucie: 大規模クラスタに適した高速セットアップ・管理ツール, 情報処理学会論文誌：コンピューティングシステム, Vol. 44, No. SIG 11 (ACS 3), pp. 79–88 (2003).
10) 斎藤秀雄, 鴨志田良和, 澤井省吾, 弘中健, 高橋慧, 関谷岳史, 頓楠, 柴田剛志, 横山大作, 田浦健次朗: InTrigger：柔軟な構成変化を考慮した多拠点に渡る分散計算機環境, 情報処理学会研究報告 HPC-111 (SWoPP 2007), 旭川 (2007).