5J-6

プロセス追跡型割り込みを考慮したプロセス起床時ロードバランス機構の改善

合田　和也†　　田胡　和哉†
†東京工科大学大学院 バイオ・情報メディア研究科

1　はじめに

近年，CPU のマルチコア化，10Gbits/sec を超える NIC の普及が進んでいる．このような環境でネットワーク通信を伴うアプリケーションを並列に効率よく扱う技術として，Receive Flow Steering(RFS) などが利用されている．しかし，現在の OS のスケジューラはこのような技術を想定しておらず，意図しないプロセスマイグレーションが行われる可能性がある．

本稿では，RFS を利用する場合に発生するスケジューラの問題を改善し，その効果を向上させることを目標とする．そこで，まず RFS を利用した場合にスケジューラがどのような動きをするかを示す．その後，改善手法を提案し，その効果を評価する．

2　プロトコルスタック処理の最適化

プロトコルスタック処理の流れを図 1 に示す．プロトコルスタックは，プロトコル処理とアプリケーションでコンテキストが分かれている． A.Foong らは，複数の CPU に分かれて処理されることがオーバヘッドになると指摘し，アフィニティを重視すると約 30%性能が向上すると述べている [1].

この問題を解決する技術として，RFS[2] がある．RFS ではパケットをその宛先プロセスが動いている CPU のソフトウェア割り込みで処理させることで, プロトコルスタックとユーザプロセスが同じ CPU で実行されることを実現している．プロセスがどの CPU で実行されているかという情報は, システムコール発行時に更新される．しかし，このように割り込み先を変えるとパケットのリオーダが発生する可能性があり，その場合には最大で約 4%の性能低下を招くことが指摘されている [3]. そのため RFS では，フロー毎に処理中のパケットの数をカウントしておくことで，パケットのリオーダが発生しないことを保証している．このように，情報の更新タイミングやパケットのリオーダを防ぐため，プロセスマイグレーションが起きても割り込み先は直ぐには変更されという性質を持つ．

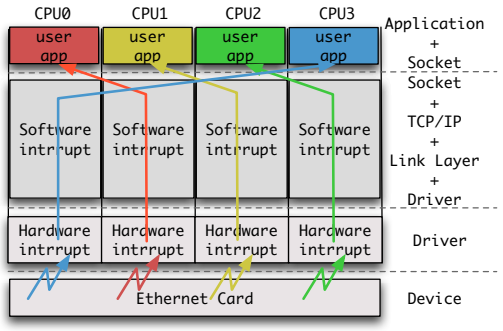

図 1: プロトコルスタック処理の流れ

3　プロセススケジュールの問題点と評価

3.1　既存スケジューラのスケジュール方針

プロセス起床時はロードバランス契機の一つであるが，ネットワーク通信を行うアプリケーションではプロセスの起床が頻繁に発生するためプロセスマイグレーションが発生しやすくなる．起床先 CPU の選択は，起床処理を行なっている CPU，プロセスが前回実行されていた CPU，その他の同一スケジュールドメインの CPU の順に選択される．これはハードウェアキャッシュなどを考慮し，出来るだけ起床処理を行なっている CPU で起床するようにするためである．

3.2　既存スケジューラの問題点

前項で述べたプロセススケジュール方針により，RFS を利用すると意図しないプロセスマイグレーションが発生する可能性がある．例を図 2 に示す．

	起床処理 CPU	プロセスが実行されていた CPU	起床先CPU
#1	0	0	1
#2	0	1	0
#3	1	0	1
#4	0	1	0

図 2: 起床処理 CPU と起床先 CPU の推移

図 2#2 において起床処理 CPU が CPU1 になっているのが理想であるが，2 章で述べたようにプロセスが

Improve Wake-up Load Balaning Mechanism for Process Traking Interrupt
†Kazuya GODA　†Kazuya TAGO
†Graduate School of Bionics, Computer and Media Science, Tokyo University of Technology

マイグレーションされた場合に直ぐには割り込み先が変更されないためCPU0となることがある．この場合，出来るだけ起床処理を行なっているCPUにスケジュールするという方針により，#2では起床先がCPU0となる．さらに#3においてプロセスを追跡してCPU1で起床処理が行われると，起床先はCPU1となる．このように，RFSにより起床処理CPUと起床先CPUが毎回入れ替わり，マイグレーションが増えることで性能が低下する．

3.3 評価

RFSを利用した場合に，意図しないプロセスマイグレーションがどの程度起こっているかを評価した．評価環境を表1に示す．ネットワーク通信として，iperfを用いて512フローの受信処理を行った．評価基準として，図2のようなスケジュール毎に起床処理CPUと起床先CPUが入れ替わっていく場合を意図しないプロセスマイグレーションとした．評価結果を表2に示す．全体のプロセスマイグレーションのうち8%は意図しないものであった．また，意図しないマイグレーションが発生した場合，安定するまで平均1.21回，最大9回かかることが確認された．

表1: 評価環境

CPU	Intel Core i7 980X - 6 Core/12 Thread
Memory	24G Byte
Kernel	Linux Kernel 3.2-rc7
NIC Chipset	Intel 82599 10 Gigabit Ethernet Controller

表2: 意図しないマイグレーションの評価結果

Unintended in Total	Average Settling Times	Max Settling Times
8%	1.21	9

4 プロセス起床時のスケジュール改善

4.1 改善方法の提案

3章で示した問題を解決するため，割り込み(プロセス起床処理)がプロセスを追ってくるまでそのプロセスのマイグレーションを禁止する方法を提案する．

4.2 提案手法の評価

3.3章と同じ条件でiperfを利用したベンチマークテストを行った．表3に既存スケジューラと提案手法スケジューラを使用した場合のそれぞれの，マイグレーション回数とラストレベル及びTLBのキャッシュミス回数を示す．同様に表4にCPU使用率を示す． 提案手法によりプロセスマイグレーションの回数が48%減っていることが確認できる．これにより，LLCのストアミス回数は5%減少，データTLBではロードミスが9%，ストアミスは8%減少した．また，CPU使用率は5%の改善となった．

表3: マイグレーションとキャッシュミス回数

Event	Existing Method [Events/sec]	Proposed Method [Events/sec]	Improvement
Process Migrations	2200	1142	48%
LLC Cache Load Misses	7228	7085	2%
LLC Cache Store Misses	6171	5857	5%
Data TLB Cache Load Misses	6714	6124	9%
Data TLB Cache Store Misses	6648	6114	8%

表4: CPU使用率

Existing Method	Proposed Method	Improvement
25.3%	24.0%	5%

5 おわりに

RFSなど割り込みをプロセスと同じCPUで実行する技術において，割り込み先のCPUが変わることにより意図しないプロセスマイグレーションが発生する事を確認した．また，これを改善するため「割り込み処理がプロセスを追ってくるまでプロセスのマイグレーションを禁止する方法」を提案し，効果があることを示した．

参考文献

[1] A, Foong et al. Architectural Characterization of Processor Affinity in Network Processing. In *ISPASS 2005. IEEE*, pp. 207 –218.

[2] Jake Edge. Receive flow steering. 2010. `http://lwn.net/Articles/382428/`.

[3] Wenji Wu et al. Why Can Some Advanced Ethernet NICs Cause Packet Reordering? *IEEE*, Vol. 15, No. 2, pp. 253 –255, 2011.