ストリームデータ処理のためのプロセススケジューリングに関する一考察*

河合栄治

奈良先端科学技術大学院大学

1 研究の目的

連続的に発生するデータの処理の仕組みとして，ストリームデータ処理 [1] と呼ばれるアーキテクチャが注目を集めている．ストリームデータ処理のアプリケーション例として，金融情報や気象情報，プローブカー情報など刻一刻と変化するデータの情報処理，計算機システムのイベントログ管理，RSS 等を用いたプッシュ型のコンテンツデータの処理などがある．ストリームデータ処理は，複数の処理エンティティが，ソフトウエアパイプライン的なデータ処理を行うという特徴をもつ．そのため，処理エンティティのスケジューリングは，サービス性能に対して大きな影響を与える．

一方で，計算機アーキテクチャの観点からは，プロセッサコアのクロック高速化に限界があることから，コア数およびキャッシュ容量の増加による性能向上が指向されるようになってきた．そのため，高性能でスケーラブルなストリームデータ処理を実現するためには，スケジューリングにおいてそうした点を考慮することが重要となってきている．

本研究では，ストリームデータ処理に適したシステムアーキテクチャの検討の第一歩として，マルチスレッドで実装した単純なソフトウエアパイプラインによるマイクロベンチマークを用いて，ソフトウエアパイプラインの深さやデータサイズ等のパラメータと性能の関連について議論する．

2 ストリームデータにおけるスケジューリングの問題

先に述べたように，ストリームデータ処理では，複数の処理エンティティが，ソフトウエアパイプライン的なデータ処理を行う．このようなシステムの実装において，モノリシックなシングルスレッドプロセスを用いることも可能であるが，現在および将来のコンピュータアーキテクチャの方向性を考慮すると，マルチスレッドを用いてスケーラビリティを確保するのが重要である．そのため本研究では，マルチスレッドによるソフトウエアパイプラインがストリームデータ処理システムの基本アーキテクチャであるという前提をおく．このようなシステムをマルチプロセッサ/マルチコア環境で実行する場合，処理の局所性と処理の順序という二つの基本的問題がある．

2.1 処理の局所性

処理の局所性を考慮することはどんなシステムにおいても重要であるが，近年プロセッサキャッシュの有効利用が性能向上の重要な要素になっている．ストリームデータ処理においては，多数のスレッドが，継続的なデータをパイプライン処理するため，どのスレッドおよびデータをどのプロセッサ上で処理するのが最適かを考慮する必要がある．

一般的に，スレッドが（キャッシュを共有しない）他のコアに移動すると，命令やスレッド固有のデータ等のキャッシュが無駄となり，処理対象のデータが移動するとそのキャッシュが無駄となる．また，スレッドとデータの間の依存関連があるため，性能面のみを考慮して個別に配置することができるわけではない．

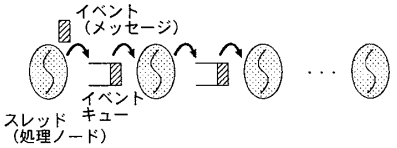

図1 ソフトウエアパイプライン

2.2 処理の順序

ストリームデータ処理の利点の一つは，従来の関係データベースを用いたシステムと比較して，データベースの一貫性をそれほど気にすることなく，必要なデータを必要なところにイベント駆動で供給することができる点にある．そのため，多数のユーザがシステム詳細を気にすることなく自由に処理エンティティを追加することができるシステムが望ましく，システムのパイプライン構造は巨大化することが予想される．こうしたシステムにおいて，スループット性能は規模拡張により解決がある程度可能であるが，そのサービス遅延（最初のデータ入力から最終のデータ出力までの時間）は，パイプラインが深くなればなるほど大きくなりがちであり，最適化が求められる．

パイプラインにおける処理遅延の最適化では，いくつかのポリシーが考えられる．代表的なものは，最終のデータ出力が早く得られるパイプラインから優先的に処理を行うものや，もっともサービス遅延が大きなパイプラインから優先的に処理を行うものである．重要なのは，システム全体で一貫性のある設定可能な指標を定めることと，処理エンティティ間の依存関係を正しく扱うことである．

3 現行の優先度スケジューリングの振る舞い

2節で述べた問題について考察する予備実験として，本研究ではソフトウエアパイプライン（図1）のマイクロベンチマークを実装し，実験を行った．実験では，4つの Xeon MP 1.4 GHz（Foster MP，2次キャッシュ 256kB，3次キャッシュ 512kB，HT 無効）と 4GB のメモリを搭載したホストを用い，Linux（CentOS 5.2/Linux 2.6.18-92.1.18.el5）上で行った．

3.1 ソフトウエアパイプラインマイクロベンチマーク

実装したソフトウエアマイクロベンチマークは，簡単な構造を持ち，表1に示すパラメータをとる．実装では，ランキューの振舞いとの同期を防ぐためにパイプラインを構成するスレッドの順番をランダム化するなど細かな工夫もしている[1]．

---

* A Brief Consideration of Scheduling for Stream Data Processing
Eiji Kawai, Nara Institute of Science and Technology

[1] 今回の実験では使用していないが，各スレッドがキュー上のデータを一度にすべて処理をする方式や，スタックサイズ調整による起動スレッド数の上限の増加なども実装している．

表1　ベンチマークパラメータ

スレッド数	パイプラインを構成するスレッドの数
イベント数	処理するイベント（データ入力）の数
イベントサイズ	各イベントデータのサイズ
データ処理の重さ	各スレッドにおけるデータ処理の重さ
処理後の yield	データ処理後にプロセッサを自主的に明け渡す（連続処理を禁止する）か否か

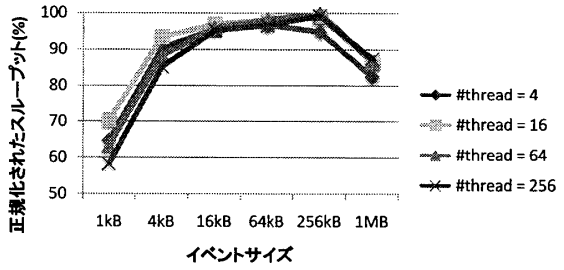

図2　実験結果（最大スループット）

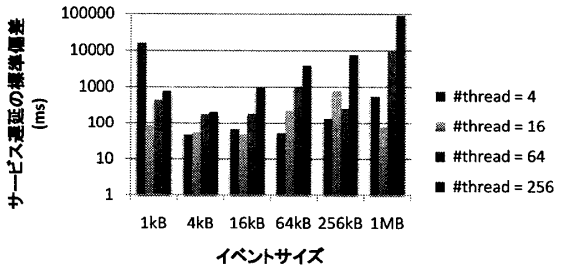

図3　実験結果（サービス遅延の標準偏差）

3.2　実験結果

実装したマイクロベンチマークにおいて，スレッド数（4, 16, 64, 256）およびイベントサイズ（1 kB, 4 kB, 16 kB, 64 kB, 256 kB, 1 MB）を変化させ，評価を行った．

3.2.1　スループット

まず代表的な性能指標として，最大スループット（1 秒間に処理したデータ量）を図 2 に示す．イベントサイズが小さい場合（1 kB～16 kB）にスループットが悪いのは，スレッド間のイベントの受け渡しや，スレッドのスケジューリングなどで発生するオーバヘッドが，データの処理と比較して相対的に大きいためである．スレッド数がプロセッサ数と同じである 4 の場合を除き，スレッド数が多くなると若干の性能低下が見られるのも同じ理由である．また，イベントサイズが大きい場合（1 MB）にもスループットが低下している．これは，今回用いたプロセッサのキャッシュサイズが影響していると考えられる．

3.2.2　サービス遅延の分散

次に，サービス遅延（処理の遅延）がどのように発生しているかを見るために，正規化されたサービス時間の標準偏差を図 3 に示す．全般的に，スレッド数が多くなるとその分サービス遅延の分散も大きくなっている．これは，スレッド数の増加にともない，処理順序のばらつきが大きくなるということを裏付けてはいる．

一方で，今回の実験では十分なサンプル数（イベント数）が得られていない可能性がある場合もあることには注意が必要である．実験では，ホストに搭載しているメモリ容量の制約や，計算量の制約から，イベントサイズやスレッド数に応じたイベント数を用いている．そのため，イベントサイズおよびスレッド数が大きい場合には，そうでない場合と比較してイベント数が小さく設定せざるを得なかったという問題がある．

4　スケジューリングの改善手法

最後に，これまでの考察をふまえ，ストリームデータ処理のためのスケジューリング機構について提案する．

4.1　ストリームデータ処理における基礎統計情報の取得および活用

一般的なプログラムでは，その処理における妥当な単位を得るには関数呼び出しを用いるのが一般的であり，そうしたアプリケーションの例としてプロファイラがある．しかし，そうした情報はスケジューリングに活用するには粒度が細かすぎるという欠点がある．

一方で，ストリームデータ処理においては，個々の処理ノードにおける処理の単位は，ミドルウエアの呼び出しにおいて厳密に得ることができ，かつスケジューリングに活用するのに十分な粒度がある．そこで，ストリームデータ処理ミドルウエア，スレッドライブラリ，さらにはオペレーティングシステムと協調して，スケジューリングに活用可能な統計情報を取得する．具体例としては，各処理ノード間の連結関係や，そこでやり取りされるイベントのサイズ，頻度などであり，これによってストリームデータ処理におけるデータフロー，処理フローが明らかになる．

4.2　スレッドとメモリオブジェクトのバインディング

先に述べた基礎統計情報を活用してスケジューリングを行うが，一般的には，フロー情報が与えられたとしても最適なスケジューリングを求めることは NP 困難である [2]．また，近代的なオペレーティングシステムでは一般的に優先度スケジューラが採用されている．そこで，ストリームデータ処理の基礎統計情報から，スケジューラへのヒントを与え，スケジューラは有効に利用するという方式とする．具体的には，従来スケジューリングにおいては，プロセッサにおける Affinity しか考慮されていなかったが，メモリにおける Affinity も考慮するものとする．すなわち，スレッド（処理ノード）のスケジューリングにおいて，扱うメッセージのキャッシュを考慮する．これは，スレッドがプロセッサ間を渡り歩くモデルを実現可能にするものである．

5　今後の課題

今回実装したマイクロベンチマークは単純なものであるが，実際のストリームデータ処理のフローグラフはもっと複雑なものである．汎用性と結果の有効性のトレードオフがあるが，ストリームデータ処理における性能面での設計空間をより考察を深めていく必要がある．また，今回提案したスケジューリング機構を具体的に設計，実装し，検証したいと考えている．

参考文献

[1] Chaudhry, N. A., Shaw, K. and Abdelguerfi, M. eds.: *Stream Data Management*, Springer (2005).

[2] Sinnen, O.: *Task Scheduling for Parallel Systems*, Wiley-Interscience (2007).