データストリーム処理とバッチ処理における動的負荷分散に向けて

松浦　紘也[†]　鈴村　豊太郎[‡]
東京工業大学[†]　IBM東京基礎研究所／東京工業大学[‡]

1. はじめに

環境や経済など様々な側面から計算機の効率的な運用が求められており、計算機の稼働時間を抑えつつ実行効率を高めなければならない。こうした負荷分散を要求する処理のうち、本研究ではデータストリーム処理を含むものを扱う。

2. データストリーム処理

データストリーム処理とはデータを蓄積せずに逐次処理するという新しい計算パラダイムで、リアルタイムの応答が要求される場合や、前後の僅かなデータのみを参照すればよい計算、データの蓄積が困難な処理に適している。

処理系にはIBM ResearchのSystem Sが存在するが、System Sはデータフロー図から直感的に処理を記述できるSPADE[1]という宣言的言語を持ち、自動性能最適化を行うSPADEコンパイラとSPC[2]という実行基盤を用いてデータストリーム処理を実現する。SPADEでは組み込みオペレータで処理を記述するが、C++やJavaといった汎用言語を用いた独自のオペレータや関数により高度に柔軟な処理も実装できる。これよりデータストリーム処理やミドルウェアの研究に適するため、本研究ではSystem Sを利用した。

3. データストリーム処理とバッチ処理の負荷分散

3.1 負荷分散の必要性

データストリーム処理では一般に負荷の変動が予測困難なので静的なノードの割り当てが難しく、運用の効率化には動的負荷分散が必要不可欠だ。低負荷時には余ったノードを止めるという選択肢も存在するが、資源活用の観点から他の処理の実行が求められる場合が多い。バッチ処理との組み合わせが考えられるものとしては、通話記録の処理、コールセンターの通話ログの処理、防犯カメラの自動監視、工場の異常検知などが挙げられ、本研究ではこうした処理を対象に負荷分散システムを実装/評価する。

3.2 データストリーム処理に起因する課題

データストリーム処理には他の演算とは異なる特有の性質があり、正しい計算結果を得るにはこれを加味して処理しなければならない。

(1) レイテンシクリティカル

バッチ処理でも処理時間全体にデッドラインが設けられることがあるが、データストリーム処理では個々の要素への演算に対しても応答時間が定められる場合がある。この特性をレイテンシクリティカルと呼び、負荷の変化を予測しながら負荷分散する必要がある。

(2) ウィンドウ

データの平均値や合計を求める場合があるが、データストリーム処理では始点や終点のないデータを扱うため一定の幅でデータを区切り、その中での最大値や平均値などを計算する。これをウィンドウと呼ぶが、計算がウィンドウに依存するためマイグレーションが複雑になる。

(3) スレッドとレイテンシ

多数のセンサの監視など同時に処理すべきスレッド数が固定の場合、バッチ処理ではノード数に応じてスレッドの実行順序を前後させればよいが、データストリーム処理では実行順序によってレイテンシが制約条件を破ってしまうと実行結果そのものが無意味になってしまう。これはノード数や処理能力が足りない場合には根本的解決は不可能で、資源が限定される場合には、ロードシェディング[6]という負荷を間引いてレイテンシを抑える手法を用いる。

データストリーム処理を扱うには以上の特性に注意しつつ、実装を行わなくてはならない。関連研究には[3]が挙げられ、バッチ処理とトランザクション処理の負荷分散について述べられているが、上記に書かれたデータストリーム処理やその特性については研究されていない。

4. 設計と実装

4.1 設計方針とアーキテクチャ

負荷分散を必要とする様々な処理に対応するために、計算部分以外をミドルウェアとして隠蔽する。計算ノードの割り当て部分は3.2の条件を満たすよう設計し、負荷に応じて計算ノードに適切な処理を割り当てられるようにする。

4.2 実装の詳細

(1) 負荷の割り振り（負荷分散）

レイテンシをもとに適切な計算ノード数を求め、ラウンドロビンで制御情報とデータを計算

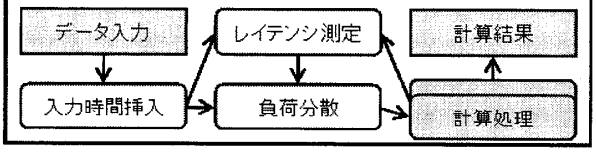
ノードに送り、適切な負荷分散を実現する。バッチ処理の実行内容もここで操作可能だが、今回はデータストリーム処理のレイテンシにのみ着目しており、こちらは実装しない。

(2) レイテンシ測定

分散環境での時間計測は困難だが、入力時に計算データに時間データを付与し、計算終了後に入力部のノードへフィードバックして計算時間を測定することでこれを解決した。時間データ分とフィードバック部分のオーバーヘッドが発生するが、測定ノード間の時刻の差異の影響を受けない点において優れている。

図1 元のプログラムのデータフロー図

図2 負荷分散適用後のデータフロー図



5. 性能評価

5.1 測定環境

8台の計算ノードでは1ノードにつき2プロセスずつ計算処理を行い、その他の処理を負荷分散ノードが行う。各ノードは 1GbE で接続される。

測定環境
OSとソフトウェア: CentS 5.2 kernel 2.6.18-92.el5 SMP AMD64 InfoSphere Streams 1.0.1 (IBM System S) , gcc 4.1.2
負荷分散ノード: Athlon 2.7GHz L2 512KB, Mem 1GB
計算ノード: Opteron 242 1.6GHz L2 1MB 2 sockets, Mem 8GB

5.2 ターゲットアプリケーション

性能評価のためのアプリケーションとして、SST[4,5]を用いた。SST とは行列計算(特異値分解)を用いて異常検知を行うプログラムで、本研究の対象アプリケーションのうち計算の負荷と粒度の変更が容易であるためこれを利用した。SST はウィンドウを持ちスレッド数の固定が必要な計算だが、今回は入力にランダムデータを用いるためこれらを無視して負荷に応じてプロセス数やデータの計算先を変化させているが、計算負荷は同じなのでレイテンシに影響はない。

5.3 測定結果

図3は 8 ノードで負荷分散を実行した結果で、負荷（入力タプル数）に応じてノード数が増減し、計算資源がうまく利用されていることがわかる。図4は負荷分散ありと 4 ノードで負荷分散なしを実行した場合のレイテンシをまとめたもので、負荷分散なしと比べてレイテンシそのものやレイテンシの発散が抑えられていることがわかる。

図4	負荷分散あり	負荷分散なし
レイテンシの平均値	97.17ms	133.7ms
レイテンシが200ms以上のものの割合	4.31%	7.77%

6. まとめと今後の展望

負荷に応じてノードが割り振られ、レイテンシの低下を実現できた。今回はバッチ処理が実装されていないため、ノードの稼動効率の定量的なデータを示せなかったが、レイテンシを抑えつつノードの使用効率を最適化する負荷分散ミドルウェアの実装を目指す。今後は実際にバッチ処理も実装し、処理の粒度や計算の性質が異なる実アプリでの評価や、タプル数などを利用し負荷の変化を先取りするようなノード割り振りや、ミドルウェアとしての実装の隠蔽、ウィンドウやスレッド数などのマイグレーション問題を解決する必要がある。また、利用可能なノード数を超えた負荷がかかった場合に有効なロードシェディング[6]の実装も必要となる。

参考文献

[1] Bugra Gedik, et. al. SPADE: The System S Declarative Stream Processing Engine, SIGMOD 2008

[2] Lisa Amini, etc SPC : A Distributed, Scalable Platform for Data Mining, DM-SSP, 2006

[3] David Carrera et al. Enabling Resource Sharing between Transactional and Batch Workloads Using Dynamic Application Placement, Middleware, 2008

[4] T. Ide et al. Knowledge Discovery from Time-series Data using Nonlinear Transformations, The 4th Data Mining Workshop of JSSST, 2004

[5] 森田 康介、など 「データストリーム処理を用いた変化点検知の実装とGPUによる性能最適化」、情報処理学会 全国大会 72回

[6] Nesime Tatbul. Load Shedding in a Data Stream Manager. VLDB. 2003.

図3 SST+負荷分散