3X-6

クラウド環境を利用したElasticなデータストリーム処理

石井 惇志† 鈴村 豊太郎†‡

†東京工業大学 ‡IBM 東京基礎研究所

1. はじめに

データストリーム処理とは、時々刻々と送られるデータに対して、リアルタイムに処理を行う計算手法であり、この手法を用いたアプリケーションの運用では、既存の Web サイト、その他のアプリケーションに比べて、リアルタイムの応答性が重要になる。データストリーム処理において、レイテンシの増大は致命的であり、その為、時間とともに変化するデータレートが局所的に増大した場合でも、可能な限りレイテンシの増大を避け、リアルタイム性を維持することが要求される。

しかし、データレートの急激な増加に対して、計算資源を物理的に、かつ即時に追加するのは、配置スペース、手続き上の問題で現実的でない。そこで、本研究では、データレートの急激な増加に対して、クラウド上の仮想マシン（Virtual Machine：以下、VM）を追加することで、レイテンシの発散を抑えることを提案する。ただし、クラウド環境を利用するには経済的コストが伴う。具体例として、Amazon EC2[2]などのパブリッククラウドは従量課金制である。レイテンシの発散を抑えられるとはいえ、必要以上に VM を起動してしまうと、その分コストが大きくなってしまう。そのため、SLA(Service Level Agreement)となるレイテンシを維持した上で、経済的コストを最小限に抑える必要が生じる。この問題に対処するため、本研究では、レイテンシと経済的コストのトレードオフを最適化問題として定式化し、必要十分な VM を確保する手法を提案する。

2. データストリーム処理系とクラウド環境

2.1 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[3]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[4]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。このように、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 を利用する。

2.2 クラウド環境

クラウド環境とは、ネットワークを介して計算資源を仮想マシンとして提供する環境であり、従量課金制などの課金方式によって外部に計算資源を提供するパブリッククラウドと、社内などの一定の範囲内で利用されるプライベートクラウドなどに分類される。既存のクラウド環境の例としては、パブリッククラウドである Amazon EC2[2]やオープンソースの Eucalyptus[1]などが存在する。本研究では、実際のクラウド環境での有効性を示すため、クラウド環境として Amazon EC2 を使用することとした。

3. ElasticStream システムの概要

本稿で提案する、クラウド環境を用いた伸縮性のあるデータストリーム処理系を、"ElasticStream"と呼ぶことにする。本章では、本研究における、ElasticStream システムを実装する上での仮定と前提について述べ、その上で ElasticStream システムの論理的な構成について述べる。

最初に、本研究において、サービスの質の基準となる SLA(Service Level Agreement)はレイテンシとする。また、本研究におけるクラウドの定義は、Amazon EC2、Eucalyptus などの IaaS(Infrastructure as a Service)のみを指す。また、自身の保有する計算資源の数は固定的で、変化させないものとする。また、それらの計算資源では処理が追いつかない場合のみを対象とする。つまり、保有する計算資源だけでは SLA を守れず、必ずクラウド環境を使用することになることを前提とする。

設計方針として、アプリケーションの機能を、データの受信部分と計算処理部分に分割し、計算処理部分を外部に抽出、分散させることで並列化を図る。この並列化された計算処理を担当するアプリケーションの一部を、クラウド環境上で実行することで、動的な並列処理を実現する。

4. コスト最適な VM 数の計算手法

4.1 対象とするアプリケーションの分類

まず、ElasticStream システムの対象アプリケーションを、データ並列型アプリケーションと、タスク並列型アプリケーションの二種類に分類する。データ並列型アプリケーションは、計算処理の負荷が軽く、入力されるデータストリームの量が多い、ネットワークインテンシブなアプリケーションが該当し、データストリーム処理の性質上、多くのアプリケーションがこちらの型に分類される。入力されるデータストリームを任意の割合で分散させ、各マシンの性能を最大限に引き出し、最小限のマシン数でレイテンシを維持することを図る。具体的なアプリケーションとしては、Twitter の投稿からの文字列マッチングや株式取引における VWAP（Volume Weighted Average Price：出来高加重平均価格）の計算などがある。タスク並列型アプリケーションは、主として入力されるデータストリームの量が少なく、独立した計算処理が複数あり、処理負荷が大きい CPU インテンシブなアプリケーションが該当する。データ並列型アプリケーションには分類できない、例外的なものがこちらに分類される。

また、複数ある処理を各マシンに分散させ、入力データストリームは複製して全てのマシンに送るため、複製したデータストリームの合計が、ネットワークのバンド幅を超えてはいけない。分散されたマシンでは、各々が割り当てられた処理のみの結果を返し、最終的に出力を統合して最終的な出力とする。具体的なアプリケーションとしては、多次元要素への SST（Singular Spectrum Transformation）による異常検知などがある。

4.2 最適化問題の定式化

以下では、レイテンシと経済的コストのトレードオフの最適化問題の定式化について述べる。本研究では、この問題を VM インスタンスタイプごとの起動数の線形計画問題として解く方針をとる。定式化の主旨は、[16]で用いられている SDAR(Sequentially Discounting AR Model)アルゴリズムを利用し、未来のデータレートを予測し、ローカルの計算資源で処理できる量を求め、残りをクラウド上の VM に委譲する、というものである。アプリケーションがデータ並列の場合は、ローカルで処理できる分のデータレートを計算すればよく、タスク並列の場合は、ローカルで処理できるタスク数を求めればよい。

図 1 が定式化した目的関数と束縛条件である。目的関数は、起動時間とアップロードのトラフィック量を最小化するものである。ダウンロードのトラフィック量は、アプリケーション依存であるから最適化問題としては考慮しない。束縛条件は、ローカルの計算資源が処理できないデータストリームの量、もしくはタスクを、クラウド VM の側が全て処理できていることである。また、定式化された線形計画問題を解く実装には、C++で実装された、オープンソースである lp_solve[13]を利用する。

$$Min:$$
$$Cost = \sum_{type}^{VMtype} (P_{type} + P_{Nin} \times D_{type}) \times x_{type} \quad ...(1)$$
$$Where:$$
$$\forall x_{type} \geq 0, \qquad \forall x_{type} \in N,$$
$$\sum_{type}^{VMtype} (D_{type} \times x_{type}) \geq (D_{next} - D_{local}) \quad ...(2)$$

図1 最適化問題の定式化

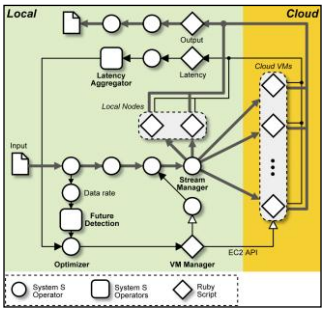

図2 ElasticStream アーキテクチャ

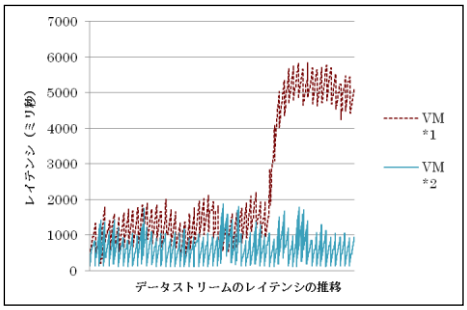

図3 クラウド環境での並列化

5. ElasticStream システムの実装

本章では、ElasticStream システムの実装構成について述べる。図 2 は、本稿で提案する ElasticStream システムのアーキテクチャ図である。基本となるデータストリーム処理系には System S を、System S に無い機能については Ruby スクリプトを利用して外部的に実装している。動作の基本的な流れは、入力データストリームを StreamManager がローカル計算資源、クラウド VM にそれぞれ分散させ、計算結果を再びローカル側で集約する。

一方で、現在のデータレートからこれからのデータレートを予測し、必要に応じて VM 数を変更する。また、実際のレイテンシをフィードバックし、最適化問題のレイテンシのパラメータを自動的に更新する。

6. 性能評価

本章では、実アプリケーションでの実験前に行ったベンチマークの結果について述べる。ベンチマークに使用したアプリケーションは、受け取った擬似データに正規表現マッチングを行い、マッチしたデータをローカル環境に結果として送信するものである。また、負荷を加えるために、単純なループ処理を追加している。

実験は、Amazon EC2 上の VM インスタンスの台数を、1 台から 4 台に変えながら同一のデータレートで入力を与えて行った。図 3 は、VM 数が 1 台と 2 台の時の、データストリームのレイテンシの推移を示している。データストリーム内の個々のデータであるタプルのデータサイズは、一つ当たり 0.1KB から 10KB に変えながら事前測定を行ったが、レイテンシに影響は無かったので、本章では 1KB に設定している。また、データセンターとの送受信のオーバーヘッドは約 0.1 秒程である。VM1 台の場合は途中からレイテンシが発散しているが、2 台以上で並列化した場合にはレイテンシは発散せず、かつ台数が多いほど低い値を保っていることが確認された。これにより、ローカル計算資源でレイテンシが発散する場合には、クラウド環境の VM を一定数追加すれば、レイテンシの発散を抑えることが実際に可能であることが分かり、更に、レイテンシの発散を抑える、必要最小限の VM 数も求められることが分かる。

7. 関連研究

[15]では、System S のオペレータとして負荷分散機能を組み込んでいる。この研究では単一の計算資源の中で処理を分散させているのに対し、本研究では処理の委譲先をクラウド環境に求めている点で異なる。ジョブスケジューリングは、[9]で実行時間の最小化の線形計画問題を用いたスケジューリング手法について述べている。クラウド環境への処理の委譲に関しては、[11]ではバッチ処理を対象として、コスト最適なジョブスケジューリングを行っている。[11]では最適化問題によるスケジューリングを事前処理として最初に行っているが、本研究ではデータストリーム処理という処理の終わりがないジョブを対象とするため、その都度最適解をアップデートしているという点で異なる。[14]では、データストリーム処理系におけるオペレータの分割について焦点を当てている。また、[8]では、データの一部を間引いて処理を行う Load Shedding の技法を用いながらオペレータの分割を行っている。また、[10]では、複数クラウド環境を想定した VM の配置について検討している。本研究では単一のクラウド環境を想定しているが、Amazon EC2 という実環境上で評価を行ったという点で異なる。また最適化の対象として[12]のように消費電力に着目するという研究もある。

8. まとめと今後の展望

本稿の貢献として、データストリーム処理を用いたアプリケーションの負荷分散手法としての ElasticStream システムのアーキテクチャの提案をしたこと、処理の委譲先であるクラウド環境の利用コストとレイテンシのトレードオフを最適化問題として定式化し、コスト最適な負荷分散手法を提案したこと、現行のデータストリーム処理系に無い、実行中の動的な計算ノードの追加を外部的に実装したこと、最後に、それらを実クラウド環境上で実装、性能評価を行ったことが挙げられる。

今後の展望として、データレート予測アルゴリズムの改善、マルチコアに対応したアプリケーションの最適化が挙げられる。また、本稿において、ElasticStream システムはミドルウェアである System S 上で実装しているが、これは本来ミドルウェアの機能として内包されるべき機能である。よって、実際のデータストリーム処理系の内部に ElasticStream システムの機能を適用し、ミドルウェアとして利用可能にすることも今後の課題である。

参 考 文 献

[1] Daniel Nurmi, et al., The Eucalyptus Open-source Cloud-computing System, Proceedings of the 2009 9th IEEE/ASCM

[2] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/

[3] Daniel J. Abadi, et al., The Design of the Borealis Stream Processing Engine, 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05), Asilomar, CA, January 2005

[4] Bugra Gedik, et al., SPADE: The System S Declar ative Stream Processing Engine" SIGMOD 2008

[5] Lisa Amini, et al., SPC: A Distributed, Scalable Platform for Data Mining DM-SSP 2006

[6] J. L. Wolf, N. Bansal, et al, SODA : An Optimizing Scheduler for Large-Scale Stream-Based Distributed Computer Systems, Middleware 2008.

[7] Bugra Gedik, A Code Generation Approach to Optimizing High-Performance Distributed Data Stream Processing, ASCM CIKM 2009

[8] Barzan Mozafari, et al., Optimal Load Shedding with Aggregates and Mining, ICDE 2010

[9] JOSE R. CORREA, et al., LP-Based Online Scheduling: From Single To Parallel Machines, Mathematical Programming: Series A and B Volume 119 Issue 1, February 2009

[10] Sivadon Chaisiri, et al., Optimal Virtual Machine Placement across Multiple Cloud Providers, Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific

[11] Ruben Van den Bossche, et al., Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads, 2010 IEEE 3rd International Conference on Cloud Computing

[12] Gaurav Dhiman, et al., vGreen: A System for Energy Efficient Computing in Virtualized Environments, ISLPED '09

[13] Lp_solve, http://sourceforge.net/projects/lpsolve/

[14] Vincenzo Gulisano, et al., StreamCloud: A Large Scale Data Streaming System, In Proceedings of ICDCS'2010. pp.126~137

[15] Scott Schneider, et al., Elastic Scaling of Data Parallel Operators in Stream Processing, IPDPS 2009.

[16] 松浦紘也, et al., データストリーム処理系 System S と Hadoop の統合実行環境, Comsys 2010