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

2008－DPS－134 (9)
2008－CSEC－ 40 (9)
2008／3／6

分散データストリーム処理における適応型リソース制御方式の検討

内山 寛之† 赤間 浩樹† 山室 雅司†

† 日本電信電話株式会社 NTT サイバースペース研究所 〒 239-0847 横須賀市光の丘 1-1
E-mail: †{uchiyama.hiroyuki,akama.hiroki,yamamuro.masashi}@lab.ntt.co.jp

あらまし センサや IC タグ等から発生する多量の情報，携帯機器の発達により収集される様々な個人の活動履歴情報，ブログのようにコンシューマが生成する多量の情報など，ユビキタス社会の進展に伴って時々刻々と多種多様な情報が発生している．これらのデータストリームを受け取り，ユーザによって定義されたオペレーションを実行し，結果を蓄積またはアラートするためのアーキテクチャにおいて，データストリームの変化や性質の違いを考慮した適応型リソース制御手法として，(1) 各種データストリームの到着率を加味した適応型負荷分散手法，(2) 総処理能力が総負荷を大きく上回る場合に，総処理能力を適切に抑制する適応型処理能力抑制手法を提案する．実験により，適応型負荷分散手法のラウンドロビン手法に対する有効性を示し，適応型処理能力抑制手法により，1 パラメータで負荷に処理能力を追随させることが可能であることを示す．

キーワード データストリーム管理システム，スケーラビリティ，PC クラスタ

Adaptive Resource Managemnt Methods for Distributed Data Stream Processing on PC Clusters

Hiroyuki UCHIYAMA†, Hiroki AKAMA†, and Masashi YAMAMURO†

† NTT Cyber Space Laboratories, 1-1 Hikarinooka Yokosuka-Shi Kanagawa 239-0847 Japan
E-mail: †{uchiyama.hiroyuki,akama.hiroki,yamamuro.masashi}@lab.ntt.co.jp

Abstract Enormous and various data streams are generated from miscellaneous data sources such as web logs, cellular phones logs and web cams. In our architecture which receives data streams, executes flexible operations defined by users and stores/alerts the results to views, we propose two Adaptive Resource Managemnt methods. One is the Adaptive Load Sharing method which balances streams' load based on average arrival time and service time. The other is the Adaptive Self-Control of Processing Power which puts restraints on total processing power when it is much larger than total load.

Key words Data Stream Management System, Scalability, PC cluster

1. はじめに

我々は，ネットワークに大量発生するデータストリームを PC クラスタにより，逐次的に処理を行い複数の情報を企業や個人の各々の目的・条件に応じたフィルタリングや検索などを可能にする情報統合管理サービスを目指して，追記・参照型データ管理システム (以下 DMS: Data Management System) の研究開発を進めている [9], [10]．図 1 は，センサ情報をはじめとする様々なストリームデータを DMS が処理を行った上で各アプリケーションへ提供する様子を示している．

SQL の拡張を行ったクエリに対するストリーム処理プロセッサとして多くの提案がなされている [1], [3], [5], [6]．これらのプロダクトと比較して，DMS の特徴は，SPJ 以外のオペレーションの利用を前提に設計を進めてきた点にある．例えば，映像監視サービスの場合には顔画像検出や物体検出，音声応答サービスなどの場合には，音声合成・認識などのオペレーションを利用する．DMS がオ

ペレーションに対して分散処理環境を提供することで，オペレーション開発者は，物理的なネットワークトポロジなどを意識することなく開発を進めることが可能である．オペレーション組み込みに対するシンプルなインターフェースを提供した上で，DMS は以下のような特徴を有する [10].

- データストリームの増加に伴う オペレーション処理に対するスケーラビリティ
- 動的なオペレーションの追加，変更，削除
- オペレーション実行バイナリの自動配布

以上のような特長により，DMS を用いて構築されたシステムは，サービスの拡大やデータストリームの変化に対して，サービスをとめることなく PC サーバの追加による対応やオペレーションの更新を行うことが可能となった．本論において取り組んでいる二つの課題 (A,B) について述べる．

(A) 様々なデータストリームの処理において，各データストリームごとに処理時間や到着率に違いが生じ，時刻により変化する．DMS においては，各 PC クラスタの負荷分散はラウンドロビン方式で行ってきた．しかしながら，データストリームの性質の違いを吸収することが出来ない場合がある．本論では，データストリームの性質に応じて各 PC サーバの CPU リソースを動的に変更し，データストリームに対する PC クラスタ全体の CPU リソース割り当てを行う手法を検討し，ラウンドロビンに比較して同数の PC サーバを用いた場合に，より多くのデータを処理できることを実験により示す．

(B)DMS では，処理を行うプロセス (以下，フィルタプロセス) が処理するべきデータストリームを決定し，取得及びオペレーションの実行を行う [10]．このことにより，処理 PC サーバ間の独立性を高めてオペレーションがクラッシュした場合においても PC クラスタ全体としては問題なく動作することが可能となった．しかしながら，フィルタプロセスはデータストリームを一時的に保持するプロセス (以下，キュープロセス) がデータを保持していなかったとしてもデータの取得を試みようとする．

- フィルタプロセスは，常に動作するため CPU の無駄な電力消費をもたらす．
- フィルタプロセスからの問い合わせ処理自体がキュープロセスへ負担をかける．

PC クラスタ全体に入力される処理負荷とフィルタプロセス群の持つ処理能力の差が大きい場合に，フィルタプロセス群に適切なスリープを入れることで，上記の課題を解決する手法を提案する．待ち行列理論によれば，処理負荷が処理能力に近づくにつれてキュープロセスが保持するデータストリームが増大する．つまり，処理負荷と処理能力を一致させれば良いわけではなく，本手法では，予測される平均待ち行列長を制御するパラメータを

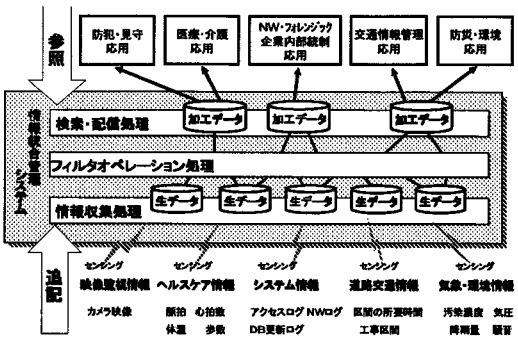


図 1　情報統合管理サービス概要

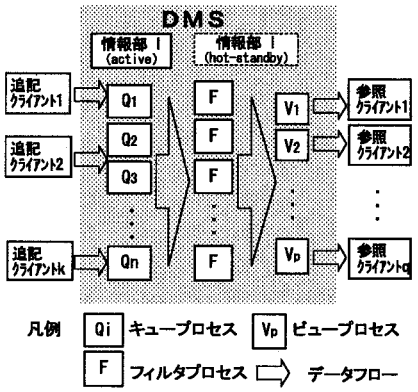


図 2　アーキテクチャ概要

提供することを特徴としている．実験において，このパラメータにより DMS 全体としてフィルタプロセスの無駄な動作を抑えつつ，キュープロセスの平均待ち時間を制御できることを示す．

2. アーキテクチャ

2.1 概　　要

図 2 には，DMS の構成要素とデータフローが示されている．追記クライアントは，データストリームをキュープロセスへ送信する．送信されたデータは，キュープロセスからフィルタプロセスへ送られ，フィルタプロセスで処理されたデータストリームはビュープロセスへ送信される．フィルタプロセスは，追記クライアントから入力されるデータストリームの各種処理を行う．この処理をフィルタオペレーションと呼び，後述のインターフェースを用いて開発を行う．これらは，データストリームに対して複数連続して適用できる．

データストリームを DMS に入力するために，追記クライアントはキュープロセスを要求し，追記を開始する．キュープロセスに入力されたデータストリームは，フィルタプロセスからの取得要求によって，フィルタプロセ

スへ送信される．データストリームがどのように処理されるかは，事前に設定されている．フィルタプロセスは，その設定に基づき処理を行い，ビュープロセスへ処理済データストリームを送信する．データストリームは，スキーマ定義済みレコード群である．フィルタプロセスが取得したレコード群に対する処理が完了すると，フィルタプロセスは，また別のキュープロセスへの取得を試みる．DMS 内の全てのフィルタプロセスが同一のフィルタオペレーション集合を持ち，データストリーム取得先キューの決定は，フィルタプロセスが自律的に決定する．フィルタプロセスの実行バイナリは情報部に登録され，適切なタイミングで各 PC サーバへ配布がおこなわれる．上記の各プロセスは，PC クラスタ上での配置は自由に行うことができる．サービス初期には 1 台でスタートし，サービスの拡大に従ってスケールする．大規模 PC クラスタにおいては，PC サーバのクラッシュや各サーバでの設定コストなどが課題となる．各プロセスが疎に連携することをアーキテクチャの基本としている．上記の観点に基づいて，キュープロセス及びフィルタプロセスについて述べる．

フィルタプロセスは，全てのオペレーションを組み込んだ実行バイナリからなり，キュープロセスに保持されたデータストリームを取得する．取得したストリームデータの処理が完了した時，次のキュープロセスを選択しデータストリームの取得を行う．組み込まれたオペレーションがクラッシュした場合，他のフィルタプロセスが補完することが可能となる．また，特定のデータストリーム処理を割り当てるわけではないため，PC サーバを増加させることでフィルタプロセス群の処理能力をスケールすることが可能となっている．キュープロセスは，フィルタプロセスからデータストリームの要求があった場合に，データの送信を行うがフィルタプロセスでの処理が終了するまではデータを保持する．フィルタプロセスにおいて，エラーやクラッシュが生じた時，データストリームを別のフィルタプロセスに再送信を行う．複数回失敗した場合には，キュープロセスは送信を中止する．よって，必ずクラッシュするデータに対して DMS 全体の停止を防いでいる．クラッシュしたフィルタプロセスは，再起動され通常状態に戻る．

2.2 プログラミングモデル

図 3 に DMS が提供するデータ処理モデルを示す．矢印は，データフローである．キューはデータを受け取り保持を行う．フィルタから要求があった場合にデータをフィルタへ送信する．フィルタは，取得したデータに対してオペレーションを適用し，ビューへ処理済データを送信する．この時，ビューはキューから得られたデータストリームのウィンドウに対応するデータのみに対して処理を行うことができる．ビューでは，あるデータストリームをフィルタで処理された結果が全て集まることになる．DMS の処理モデルは MapReduce [4] と類似性がある．しかしながら，DMS は，ストリームデータに特化したものになっている [10].

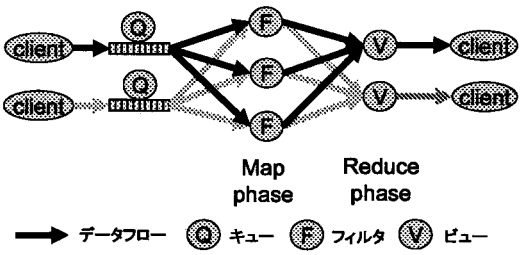


図 3 処理モデル

2.3 インターフェース

株価をクライアントからキュープロセスへ追記し，フィルタプロセスでウィンドウ幅における株価の平均を計算し，V ではそれらを出力するというアプリケーション例におけるオペレーションの記述例は，以下のようになる．

```
Record* doExecute(Record** recs,int len){
 int sum=0;
 for(int i=0; i<len; i++){
  sum = sum + recs[i]->getInt(''stock'');
 }
 Record* output = new Record(m_outschema);
 output->setDouble(''avg'', sum/len);
 return output;
}
```

入力スキーマは，(int stock) で定義されており，オペレーションの出力スキーマは，(double avg) で定義される．doExecute() はコールバック関数として機能し，処理を行うべきレコードがフィルタプロセスに送られてくると，この関数を経由して，情報源 ID から特定されるフィルタプロセスへデータを渡す．recs は，レコードの配列が入力され，len は，ウィンドウサイズを示している．得られた stock データ群から平均を算出し結果レコードを返す．次に，V での記述例を示す．

```
void receiveData(Record** recs, int len){
 double avg;
 for(int i=0; i<len; i++){
  avg = recs[i]->getDouble(''avg'');
  printf(''(%f)'', avg);
 }
}
```

オペレーションと同様にコールバック関数でデータストリームがレコードとして渡される．avg の値を取得し標準出力へ送る．

3. 課　　題

入力されるデータストリームの到着率，サービス時間等は，時間とともに変化するが，DMS を利用する開発者は設計時に上記の性質を正確に記述することが困難である．そこで，DMS を構成する PC クラスタが持っているフィルタプロセスの CPU リソースをデータストリームのの変化に合わせてリソース配分を行う．フィルタプロセス群があるキュープロセスへリソースを与えることは，フィルタプロセスがそのキュープロセスへストリームデータを取得し，処理を行うことである．この時以下の課題がキュープロセス及びフィルタプロセス間に生じる．(A) 現状のラウンドロビン手法においては，負荷が比較的小さいキュープロセスへの不必要なデータ取得処理が発生し，キュープロセスの数が増加した場合にフィルタプロセスのオーバヘッドが増大する．(B) キュー群に入力される総処理負荷がフィルタ群の持つ総処理能力よりも極端に小さい場合に無駄な空振りが発生し，キュープロセスへの負荷を上げてしまう．また，空振りのために CPU リソースを 100%使うことになり無駄な電力消費を生んでしまう．上記の課題に対し，(A) フィルタ処理のリソースを負荷割合に応じて配分し，無駄な空振りを抑え，必要となる CPU 数を抑え，(B) 総処理能力を総処理負荷にあわせて抑えつつ，キューの平均待ち時間を制御する．

4. 適応型制御手法

4.1 負荷モデル

$q_i \in Q(i = 1, 2..|Q|)$ 及び $f_i \in F(i = 1, 2, ..., |F|)$ は，それぞれキュープロセス，フィルタプロセスを表す．ここで，$|Q|$，$|F|$ は，DMS に参加しているキュープロセスの数及びフィルタプロセスの数とする．q_i の負荷は，$\lambda_i x_i$ で定義する．ここで，λ_i，x_i は，q_i で計測される平均到着率及び単位レコードあたりの平均サービス時間であるとする．$\lambda_i x_i$ は，1 秒間に行われるべき処理時間を表しているといえる．[1]

4.2 適応型負荷分散手法

CPU リソースは，フィルタプロセスがどのキュープロセスからデータを取得するかで制御を行う．フィルタプロセスは，取得したデータの処理が完了すると，次に処理をするべきデータをどのキューから取得するかを決定する．q_i の負荷 $\lambda_i x_i$ の変化が設定された閾値を超えた場合，q_i からマルチキャストを行い全ての $f_i \in F$ へ負荷の変化が通知される．フィルタプロセスは全キューの負荷を保持するキュー負荷テーブルを用いて現状の全ての負荷の和 $S = \sum_{i=1}^{|Q|} \lambda_i x_i$ を計算する．1 から S に含まれるランダム値 r を作成し，$\sum_{i=1}^{h} \lambda_i x_i \leqq r < \sum_{i=1}^{h+1} \lambda_i x_i$ を満たすような h をひとつ見つける．次にフィルタプロセスが取得するキュープロセスを q_h とする．フィルタプロセスが取得するたびに上記の計算を行うことは，オーバーヘッドとなる．そこで，キュープロセスからの負荷変動通知の際に，フィルタプロセスにおいて複数回の q_h 決定処理を行い，取得先キューリストを q_h の要素として保持する．キュープロセスからの変動通知が無い限り，リストをラウンドロビンによってたどる．以上により，各フィルタプロセスは，どのキュープロセスからデータを取得するかについてのスケジューリングを他のプロセスと一切同期せずに決定する．DMS 全体としては，負荷に応じた取得処理を行うことができる．ラウンドロビン手法は，各キュープロセスの平均サービス時間の比がリソース割り当ての比となる．本手法では，平均到着率を考慮したリソース割り当てを行うことが出来る．

4.3 適応型処理能力抑制手法

本節では，DMS 全体に流入するデータに対する全負荷に比べて，フィルタプロセスに用意された CPU リソースが大きく上回る場合に，フィルタプロセスが自動的にデータ取得処理を抑える手法について述べる．フィルタプロセスに 1 つの CPU が割り当てられている場合，1 秒間に処理できる仕事量は最大で 1 秒である．フィルタプロセスに割り当てられる CPU の数を G とすると，並列に処理できるのであれば，1 秒間において G 秒に相当する処理が可能となる．この G を DMS の持つ総処理能力とする．総処理能力 G と総負荷 S の差 $G-S$ が大きい場合に，その差を小さくするための各フィルタプロセスのリソース抑制率 R を以下のように定義する，$R := \frac{G-S}{G}$．抑制率 R に従い，総処理能力を抑制した抑制後処理能力 G' は，

$$G' = G(1-R) = G(1 - \frac{G-S}{G}) = S$$

となる．これは，総負荷 S に一致することを示しており，1CPU 上にそひとつ配置されている G 個のプロセスが 1 秒間あたり，S 秒の処理時間を与えることになる．このように，抑制率 R に従い総処理能力を抑制すれば，総処理能力は，総負荷に一致することが可能となる．1 秒間あたり，R だけスリープするのは困難であるため，この抑制は取得先リストに適当なスリープを示すような要素を追加することで実現する．取得先リストに含まれるキュープロセスに対する処理時間の和 L_S は，$L_S = \sum_{i \in L} x_i$ とかける．但し，取得先リストには，重複したキューが配置されることがある．このデータ処理プロセスは，L_S 秒間に，L_S 秒のリソースを利用していることになる．この時，取得先リストに，$\frac{R}{1-R} L_S$ 秒スリープする要素を追加する．この時，

$$L_S + \frac{R}{1-R} L_S = \frac{1}{1-R} L_S$$

1: 例えば，$x_i = 5, \lambda = 3$ は，5 秒かかる仕事が毎秒 3 個入っていることを示している．5 × 3=15 が毎秒，処理しなければならない処理時間となる

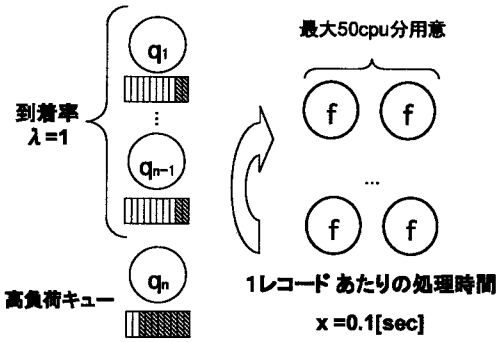


図4　実験概要図：適応型負荷分散手法

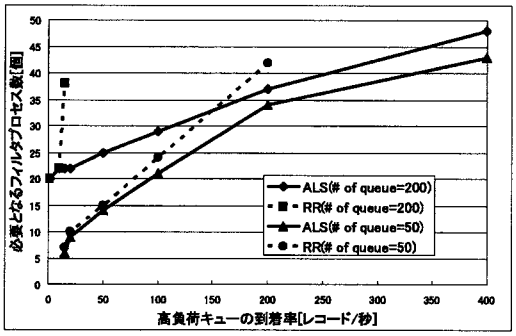


図5　適応型負荷分散手法とラウンドロビン手法

秒間に L_S 秒のリソースを利用していることになる．ここで，1秒間あたりのリソース使用率は，

$$L_S/(\frac{1}{1-R}L_S) = 1-R.$$

よって，1台あたりの1秒間におけるリソース使用率が，$1-R$ となり，G 台あれば，1秒間当たり $G(1-R)$ のリソース使用になり，式(1)と一致する．待ち行列理論[7]によれば，到着率及びサービス時間が一般分布であるときの，平均待ち時間の上限 W は，

$$W \leqq \frac{\sigma_a^2+\sigma_b^2}{\frac{2}{\lambda}(1-\lambda x)}$$

となることが知られている．σ_a, σ_b は，それぞれ平均到着時間及び平均サービス時間の分散を示している．σ_b を小さくするため，

$$\frac{R}{1-R}\frac{L_S}{|L|} \qquad (1)$$

秒間スリープする要素を取得先リストの要素間に追加する．ここで，$|L_S|$ は，取得先リストの要素数であるとする．

5. 評　　価

5.1 適応型負荷分散手法

図4には，実験の概要を示す．図中のq及びfは，それぞれプロセスをあらわしている．キュー数 n は，キュープロセスの数を表し，$n-1$ 個の到着率 $\lambda=1$ とする．残りの一つのキュープロセスの到着率は可変とする．フィルタプロセスに対しては，最大50プロセスを用意する(1CPU/1プロセスで配置)．q_n の負荷を上げていった場合に，用意されたフィルタプロセス数での最大処理限界を調べる．このとき，どれか一つのキュープロセスの待ち行列長が600を越えた時点(10分間分のデータストリームに相当)のフィルタプロセス数を「必要なフィルタプロセス数」と定義する．取得先キューリストの大きさは50とする．

図5に，ラウンドロビン手法(RR)及び提案手法(ALS)の必要なフィルタプロセス数を示す．x軸は，q_n の到着率を示している．y軸は，必要なフィルタプロセス数を示している．ラウンドロビン手法の場合(点線)，キュープロセス数を50から200に変更した時，処理能力に大きな劣化がみられる．これは，負荷の高い q_n から取得を行わず，q_1〜q_{n-1} へ取得を行い無駄な取得処理が発生し，システム全体のオーバヘッドとなっているためである．対して適応型負荷分散手法は，キュープロセス数が増えた場合でも高負荷キューへ適切に取得を行っている．フィルタプロセス群の持つ総処理能力を到着率を考慮した負荷分散により，CPUリソースの配分をしている．

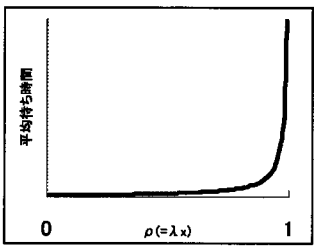


図6　平均滞留時間

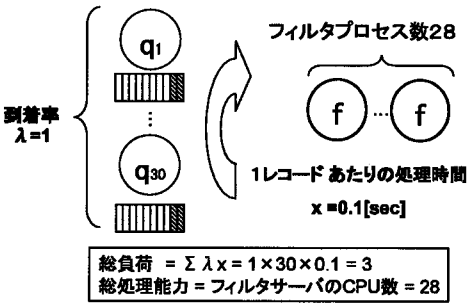


図7　実験概要図：適応型処理能力抑制手法

5.2 適応型処理能力抑制手法

総負荷と総処理能力を近づけた場合，待ち行列の平均待ち時間の上限値は図6のようになる．これより，前述

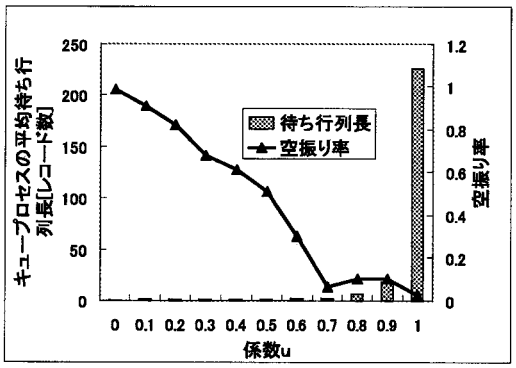


図 8 パラメータ u に対する待ち行列長と空振り率

の手法により総負荷と総処理能力を近づけると滞留時間が極めて大きくなることが分かる．そこで，式 (1) とパラメータ $u(0 < u < 1)$ との積をスリープ時間とし，発散すること防ぐ．u は，アプリケーションの性質によりチューニングされるパラメータとなる．

図 7 には，実験概要図を示している．キュープロセスは，30 個用意し，到着率は $\lambda = 1$ である．フィルタプロセス数は 28 個で，CPU 数も 28 個であり，サービス時間は，$x = 0.1$ である．ここで，総負荷及び総処理能力は定義により，それぞれ 3 及び 28 である．総負荷に比較して総処理能力が大きく高い状態である．

図 8 には，適応型処理能力抑制手法の結果を示している．x 軸は，パラメータ u を示している．y 軸には，待ち行列長と空振り率を示している．ここで，空振り率とは，フィルタプロセスがキュープロセスへの取得処理を行ったときにキュープロセスからレコードが一つも取得できなかった割合を示している．パラメータ u が 1 に近づくと，待ち行列長が大きく劣化する．図 6 で示された傾向と一致する．但し，フィルタプロセスの抑制により，無駄な空振りは減少している．u が 0 に近づくと空振り率は大きく上昇するが，待ち行列長はきわめて小さい．既存の抑制を行わない場合が，$u = 0$ の場合であり，98.8% もの空振りが発生している．空振り率と待ち行列長は，トレードオフの関係にあるが，u により制御可能となる．

6. 関連研究

これまで，STREAM [6] をはじめとする SQL の拡張ストリーム処理言語をベースとした研究が盛んに行われてきた [1], [3], [5], [6], [8]．これらは，与えられたデータストリーム向けに拡張された SQL に対するプランナの構成や最適化を行っている．DMS では，カラム，レコード及びスキーマは規定しているが，一般的な処理を組み込むことを前提としている．分散 DSMS としては，Borealis [3] などがあるが，それらは主にネットワークコストの最小化という観点から SQL オペレーションの配置を最適化をはかるるものであり，状況が変わった場合には SQL オペレーションの分割などを行い SQL オペレーションの再配置を行う．このような課題は，特定のオペレーションは必ず一つのサーバ上で実行されるという前提から生じている．分散プログラミング環境における負荷分散技術 [2] においても何らかのオペレーションが存在し，そのオペレーションを PC クラスタ上でどのように配置を行うか，負荷状況が変化した場合にどのようにオペレーションの移行を行うかについて検討されてきた．DMS では，フィルタプロセスに全てのオペレーションを組み込むことでオペレーションの配置という課題ではなくフィルタプロセスの負荷分散という課題に取り組んでいる．単純なデータモデルを採用することで本論で述べたような特徴を得ている．MapReduce [4] は，分散配置されたデータブロックをそれぞれ処理することで負荷分散を行っている．動的な負荷分散に対応することは考えられていない．

7. まとめ

分散データストリーム処理システム上において，適応型リソース制御手法の検討を行った．スケールする手法の確立やヘテロな環境におけるレスポンスタイムを意識したデータストリームの送信手法を検討していく．

文献

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: A New Model and Architecture for Data Stream Management". In *VLDB Journal (12)2: 120-139*, 2003.

[2] D. Gupta and P. Bepari. "Load Sharing in Distributed Systems". In *National Workshop on Distributed Computing, Jadavpur University, Calcutta*, 1999.

[3] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. X. and S. Zdonik. "The Design of the Borealis Stream Processing Engine". In *CIDR*, 2005.

[4] Sanjay Ghemawat Jeffrey Dean. "MapReduce: Simplified Data Processing on Large Clusters". In *OSDI*, 2004.

[5] S. Chandrasekaran, et. al. "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World". In *CIDR*, 2003.

[6] The STREAM Group. "STREAM: The Stanford Stream Data Manager". In *IEEE Data Engineering Bulletin*, 2003.

[7] Wiley-Interscience.

[8] 山田 真一, 渡辺 陽介, 北川 博之. "ストリーム処理とデータベースを統合した実世界情報管理基盤". 電子情報通信学会 DEWS, 2006.

[9] 赤間 浩樹, 内山 寛之, 三浦 史光, 西岡 秀一, 内藤 一兵衛, 谷口 展郎, 山室 雅司, 櫻井 紀彦. "追記・参照型データ管理プラットフォーム アーキテクチャの提案". 情報処理学会 DPSWS, 2006.

[10] 内山 寛之, 赤間 浩樹, 西岡秀一, 内藤一兵衛, 長谷川知洋, 谷口展郎, 兵藤正樹, 三浦史光, 山室雅司, 櫻井 紀彦. "分散データストリーム処理アーキテクチャの提案". In *DBWS*, 2007.