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

2007－DBS－143 (55)
2007／7／3

分散データストリーム処理アーキテクチャの提案

内山 寛之† 赤間 浩樹† 西岡 秀一*† 内藤一兵衛† 谷口 展郎†
長谷川知洋† 三浦 史光† 山室 雅司† 櫻井 紀彦†

† 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}@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を提案する．特徴としては，(1) 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，(2) 動的なオペレーションの追加，変更，削除，(3) オペレーション実行バイナリの自動配布などが挙げられる．本アーキテクチャに基づいたプロトタイプシステムを実装し，映像監視アプリケーションへ適用した．この適用例に対して，PC サーバの台数に対するスケーラビリティ及びオペレータの動的な変更に対するオーバーヘッドを評価する．

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

An Architecture for Distributed Data Stream Processing on PC Clusters

Hiroyuki UCHIYAMA†, Hiroki AKAMA†, Shuichi NISHIOKA*†, Ichibe NAITO†, Noburou TANIGUCHI†, Tomohiro HASEGAWA†, Fumiaki MIURA†, Masashi YAMAMURO†, and Norihiko SAKURAI†

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

Abstract Enormous and various data streams are generated from miscellaneous data sources such as RFID sensors, web logs, cellular phones logs and web cams. To deal with these data streams, we propose an architecture which receives data streams, executes flexible operations defined by users and stores/alerts the results to views/applications via simple interfaces. This architecture is composed of PC servers and has features:(1)scalability for the processing of the operations, (2)dynamic insertions/updates/deletions of the operations, (3)automatic distribution and execution of executable file which includes all operations. We describe a prototype system based on the architecture and a video monitoring application implemented on the prototype system. Finally, we evaluate the scalability for the numbers of PC servers and the distribution overheads of operators.

Key words Data Stream Management System, Scalability, PC cluster

1. はじめに

情報爆発時代に向けた新しい IT 基盤の研究プロジェクトや，データストリーム向けシステムの研究開発が活発化している．ネットワークに大量発生するログ情報を，収集・構造化・半永久保存し，複数の情報を同期・相互連携し，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，時間・空間を跨った検索などを可能にする情報統合管理サービス (図 1) 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]．図 1 の下側には情報源となる各種のセンサ群があり，図 1 の上側には情報を活用する各種応用サービスが存在する．DMS は，それら情報源と応用サービスの間に存在する．現在，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 [4]．これらは，SQL の拡張を

: *現 NTT レゾナント株式会社 〒100-0004 東京都千代田区大手町 1-6-1 大手町ビルヂング.
NTT Resonant Inc., 6-1-3F Otemachi building Otemachi 1-chome Chiyoda-ku, Tokyo Japan

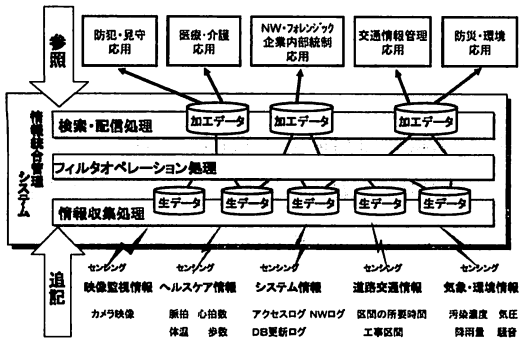

図1 情報統合管理サービス概要

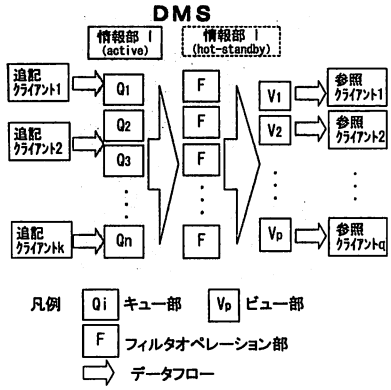

図2 アーキテクチャ概要

行ったクエリに対してストリーム処理を行う．しかしながら，ストリーム処理においては，SPJ 以外のオペレーションを要求される場合がある．例えば，映像監視の場合には顔画像検出や物体検出，音声応答などの場合には，音声合成・認識が挙げられる．DMS は，このようなモジュールに対して分散処理環境を提供する．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り，組み込まれたオペレーションに対するデータストリームが増加した場合でも，サーバを追加し後述するように DMS のプロセスを立ち上げるだけでスケールすることが可能となる．また，ストリーム向けサービスにおいては，オペレーションの改良や追加などをサービスを完全停止することなく実行することが求められる．多数の応用に長期的に対応し続けるための要件は，情報源，応用サービスやデータへの処理内容に対して，追加，変更や削除をサポートすることである．分散ストリーム開発環境を提供するという目的に対しては，シンプルなオペレーション定義用インタフェースを提供・経由して逐次引き渡されるレコード群を処理する．これにより，開発者はネットワークプログラミングを意識することなくオペレーションを実装することが可能となる．本論では，以上の特徴を持った DMS アーキテクチャを示し，現在開発中であるプロトタイプシステムを用いて，スケーラビリティやオペレーションの入替によるオーバーヘッドに対する評価を示す．

2. アーキテクチャ

図 2 には，DMS の構成要素とデータフローが示されていている．追記クライアント (AC) は，後述するライブラリ群を用いてセンサデータをキュー部 (Q) へ送信する．送信されたデータは，キュー部からフィルタオペレーション部 (F) へ送られ，F で処理されたデータストリームはビュー部 (V) へ送信される．参照クライアント (RC) へは，アラートまたは検索といった参照方式を提供する．

DMS を使ったサービス構築は，情報源側の AC 開発者，フィルタ処理の要素機能であるフィルタ部オペレーション (FOP) 作成者，応用サービス側でビューを利用する RC 作成者，DMS 管理者によって行われる．ここで，FOP とは F において定義されているオペレーションを指す．例えば，顔画像検出や移動検出に対してそれぞれ FOP が開発されることになる．これらの FOP は，データストリームに対して複数連続して適用できる (移動検出 FOP を適用した後で，顔画像検出を行う等)．DMS に対して入力される多種のデータストリームを適宜処理しデータフローを制御するために，DMS 管理者は，情報源 ID を定義する．情報源 ID をキーとする制御情報は，(情報源 ID，情報源のスキーマ定義，Q の永続化フラグ，処理 ID 群，window サイズ) から構成される．処理 ID 群において，情報源 ID で識別されるデータストリームがどのように処理されるかを規定している．処理 ID をキーに持つ処理制御情報は，(処理 ID，バージョン，FOP の系列) から構成される．FOP の系列には，定義済 FOP が複数記述される．この系列に従い，データストリームの処理が規定される．

上記の Q，F は，それぞれプロセスとして構成されているので，PC クラスタ上での配置は自由に行うことができる．よって，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，新たな開発をすることなしに PC サーバの追加と mapper プロセス (M) を実行するだけで，Q，F の配置，起動を行うことが可能である．M は，Q，F を制御するために各 PC サーバに一つ存在し，I の位置を特定し Q，F の配置情報を取得し，Q，F を立ち上げる等の制御を行う．大規模 PC クラスタに対応していく場合，運用中に PC サーバがクラッシュする問題や各サーバでの設定コストの問題などが考えられる．これらの問題に対応するために，DMS では，前出の I，Q，F，V 等のプロセスがそれぞれ独立して動作することをアーキテクチャの基本としている．I は DMS 全体を完全に制御するのではなく，緩やかに連携をとるように設計を行っている．例えば，M は起動後 I の位置すら情報として与えられない状況で起動する．I は，一定周期で現在 DMS に参加しているプロセス群のリストをマルチキャストする．その情報を M が受信して初めて I の場所を特定することができる．M は，I が管理している情報を基にして，Q や F の制御を開始する．また，I が送信するプロセスリストにおいて，Q，F の生存確認を行うために応答を要求するフラグを持っている．フラグが ON のとき，対象となった Q，F は I へ生存していることを通知する．もし，生存していない場合にはプロセスリストから該当する Q，F は削除される．緩やか

に DMS 内部のプロセスリストが更新されていく．これらのリストはマルチキャストで通信されるため，起動後は Q，F は各自プロセスリストを管理する．よって，I がダウンしたとしても，すでに起動した Q，F は動作し続ける．Q，F，V，I，M の各部は以下の機能と特徴を持つ．

2.1 追記部 (Q)

AC からの新たな接続があった場合には系内にマルチキャストでその接続情報を共有する．AC からの接続は情報源 ID を伴って行なわれるため，Q はその情報源 ID で指定される制御情報に定義されたスキーマに割当てられる．AC のコネクションと Q は常に 1 対 1 に対応するが，1 つの AC が複数の Q とのコネクションを持つことは可能である．Q は AC から追記されたデータを受け取り，系内管理用のレコード ID を付与すると共に，受取の時刻印を付与し，F 向けにキューイングすると同時に，生データとして DISK に書込む．入力データが処理されたかどうかを F の応答によって判別し，失敗した場合には別の F へデータを送る．但し，F のもつ FOP はクラッシュする可能性がある[1]ため，指定回数以上の再送信は行わない．

2.2 フィルタ部 (F)

F は Q からデータを取得する前半部分と，受け取ったデータを処理する後半部分からなる．前半部分は自律的に系内の Q 群から 1 つ Q を決定し，その Q 上のキューからデータを情報源 ID と共に取り出す (Pull)．F はその際，自らが持つ F 内の処理 ID のバージョンを Q に通知し，Q は自分の持つデータに必要とされる処理 ID のバージョンと比較し，バージョンチェック結果も返却する．不整合が生じた場合には，F の実行バイナリを更新・F の再起動を行う．後半部分は受け取ったデータに対応する情報源 ID に登録された処理 ID 群，その中に含まれる FOP 系列を順に適用する．アーキテクチャ上の重要な特徴は，

- DMS 内の全ての F が同一の FOP 集合を持つ．
- 取得先 Q の決定は，F が自律的に決定する．

である．F は，DMS 内で複数存在するが，その中のどれがクラッシュしたとしても他の F が処理を継続することが可能である．また，DMS 内に存在する Q の状態を見て取得先を変更することができるため，Q の負荷が変化した場合に F の割り当てを動的に変化させることが可能となる．他の特徴としては，以下のようなものがある．

- F は FOP の組込みインタフェース (I/F) を持ち，ユーザ定義処理の組込みが可能．複数の FOP が組み込まれた F は実行バイナリとして構成され，次節で述べる情報部に登録しておく．
- タプルカウントベースのウィンドウ処理が可能．

2.3 ビュー部 (V)

F からデータを受け取り，RC に提供するものである．機能と特徴は以下の通りである．

- V は組込み I/F を持ちユーザ定義のビューの組込みが可能．
- V は複数の FOP からのデータを受け取ることが可能．これにより複数の F に分散したデータをまとめなおすことが可能になる．データ合流のためのバッファを持ち，バッファ内での合流・整列処理も可能．

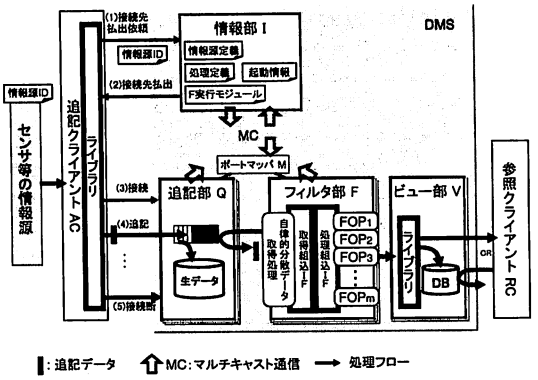

図 3 アーキテクチャ詳細

二つ目の特徴は，F が全ての FOP を処理できる構成のため，どのレコードがどの F を経由するのかは DMS は関与しない (どの F を経由しても問題ない) という F での処理と関係したものである．これにより，V で時系列データを情報源 ID ごとにまとめ直し，かつデータストリームとして出力することが必要になる．顔検出においては，顔が検出された場合とされない場合で処理時間が大きく変わる．つまり，新しいレコードが先にビューへ到着してしまう可能性がある．しかし，サービス上，時系列として参照することが必須であるため，整列し時系列データとして出力する必要がある．

2.4 情報部 (I)

I は緩やかなプロセス管理・制御情報管理を行うと同時に，AC からの接続要求に応じて空き Q プロセスの割り当てを行う．AC はその Q プロセスに接続を行なう．これにより AC と分散する Q との動的な対応付けを行なう．情報部まで含めたデータの流れの処理を図 3 に示す．AC ライブラリを通じて，I や Q と接続を行うが，これらの詳細なやりとりはライブラリによって隠蔽される (3.1 節参照)．Q のその他の機能と特徴は以下の通りである．

- 数十秒～数分に 1 度の契機で系内のプロセス構成をマルチキャストで共有し，新たに追加されたマシン上の M に対して系への参加を促す．
- 系内各マシンの起動情報や情報源情報，処理制御情報，スキーマ情報，最新の F 実行バイナリを管理し，系内の各プロセスに HTTP で公開する．

なお，情報部サービスが停止しても，DMS は既存のサービスを継続できる．ただし，新しい Q への接続と，新しい F プロセスの動的な系への参加はできなくなる．その制約を避けるため情報部は Hot-Standby 構成を採ることができる．

2.5 ポートマッパ (M)

各マシンにはそのマシン内の Q や F 等の管理者として M が 1 プロセス存在し，系内にマルチキャストで流れる情報を受信し，系内の分散プロセスの情報や Q の輻輳状態を配下のプロセ

[1] 例えば，FOP が外部のライブラリを利用している場合，ある特定のデータに関してセグメンテーション違反を発生する可能性ある

```
 1: int main(int argc, char* argv[]){
 2:  AddDriver* driver=new AddDriver(Propeties);
 3:  AddConnection conn = driver->getConnection();
 4:  Schema* schema = driver->getSchema;
 5:  Record* rec=new Record(schema);
 6:           //get stock data and set to the val.
 7:  rec->setInt(''stock'', val);
 8:  conn->sendRecords(rec, 1);
 9:  delete rec;
10:  driver->releaseConnection();
11: }
```

図4 AC コーディング例

```
 1: Record* doExecute(char* opt,
 2:                   Record** recs,
 3:                   int ws){
 4:  int sum=0;
 5:  for(int i=0; i<ws; i++){
 6:   sum = sum + recs[i]->getInt(''stock'');
 7:  }
 8:  Record* output = new Record(m_outschema);
 9:  output->setDouble(''avg'', sum/ws);
10:  return output;
11: }
```

図5 FOP コーディング例

スに伝える．その他，以下の処理も行なう．

- I の設定を参照して各マシン内の F や Q の起動を行う．各マシンでは M を起動するだけで良くなるため運用が簡単になるという効果がある．
- F が Q からデータを受け取る際に処理 ID の版番号が古いことを発見した場合には，マシンの代表である M が F 実行バイナリ入替えの処理を行なう．

I からの系内プロセス構成情報を M を経由して受信した Q や F は，自らがプロセスリストに含まれていない場合に自らを含めるように系内にプロセス情報の修正をマルチキャストで I や M に依頼する．

3. インターフェース

本節では，AC，FOP，RC に対する開発者へ提供されるインターフェース及び実装例を示す．以下の例は，株価を AC から Q へ追記し，F 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ものである．入力スキーマは，(int stock) で定義されており，FOP の出力スキーマは，(double avg) で定義される．これらの情報は，開発者が設計しあらかじめ I に登録しておき，運用時には I から取得可能となっている．これらコードは，C++で記述されている．開発者は，以下のコードを書くことで分散ストリーム処理を行うことが可能となる．

3.1 追　　記

図 4 は，AC のコーディング例を示している．AddDriver は，AC ライブラリのオブジェクトであり，キューの払い出し等を提供する．Properties には，I の IP アドレス，ポート，情報源 ID を格納している．driver を経由して I に問い合わせを行い Q

```
1:void receiveData(Record** recs, int ws){
2: double avg;
3: for(int i=0; i<ws; i++){
4:  avg = recs[i]->getDouble(''avg'');
5:  printf(''(%f)'', avg);
6: }
7:}
```

図6 V コーディング例

との接続を行い (ℓ3)，情報源 ID の利用する入力スキーマを取得している (ℓ4)．送信するためのレコードオブジェクトを作成し，stock カラムへ値の設定を行い (ℓ7)，レコードの送信を行う (ℓ8)．ここで，第 2 引数はレコードの数を示している．複数のレコードを一括で送信するときにレコード数を設定する．5〜8 行目を繰り返すことで，データストリームを Q へ追記することが可能である．

3.2 フィルタオペレーション

図 5 は，処理 ID をキーに持つ FOP の作成例である．この関数は，仮想関数として定義されている．FOP 開発者はこの関数をオーバーライドすることで，FOP の実装を行う．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 F 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 FOP へデータを渡す．引数 opt は，処理 ID に対して引数として渡された文字列となる[2]．第 2 引数 recs は，rec レコードの配列が入力され，ws は，ウィンドウサイズを示している．doExecute(...) は，仮想クラスに定義されており，メンバ変数として，m_inschema 及び m_outschema を持っている．それぞれ，FOP の入力/出力スキーマである．これらは，F 起動時に自動的に I から取得されるため，ユーザは特にスキーマの取得を意識する必要はない．ウィンドウに含まれる株価の和を計算し (ℓ5-ℓ7)，m_outschema に対応するレコードを作成，平均を計算し値をセットする (ℓ8-ℓ9)．出力を return を用いて行う．この output レコードは，処理 ID 群に示されている次の FOP の入力データとして渡される．例えば，組み込み FOP である Emitter(...) は，引数に V の送信先を指定することが可能である．よって，処理 ID 群に，

```
avg() Emitter(''ipaddress:port'')
```

と記述することで，avg の処理結果を適切なビューへ送信することが可能となる．

3.3 ビュー

図 6 は，V のインタフェースであり，FOP と同様にコールバックドリブンでデータストリームがレコードとして渡される．recs 及び ws は，FOP で処理されたレコードとそのレコード数である．avg の値を取得し標準出力へ送る (ℓ3-ℓ6)．receiveData(...) には，各 F で処理された平均値を格納したレコードが Emitter(...) により送られてくる．F に入力されたレコードの順番と F から出力されたレコードの順番は一致していない場合がある．そこ

---

2 例えば，この FOP に対する処理 ID が "avg" であるとする．この時，情報源 ID をキーとして持つ制御情報における処理 ID 群において avg("hello") と書けば，opt に "hello" が入力される

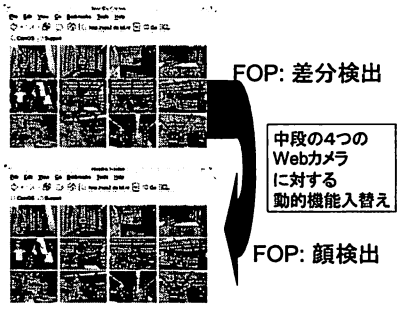

図7 FOPの動的入替例

で，receiveData(...) へレコードを渡す前に，V において得られたレコードを時系列順に並びなおす機能を有している．

4. プロトタイプと映像監視への適用

4.1 プロトタイプ

2. 章で述べたアーキテクチャに従いプロトタイプを作成した．DMS プロトタイプは C++を使用し，IA マシン (EM64T) 上の Linux(Fedora Core) で動作している．系内通信や AC との通信には標準データ形式 (XDR) を使っているため，32ibit/64bit マシンの混在環境へも対応可能である．分散構成を基本とする DMS は現在 50 台の環境での動作実績があり，1 台構成のスモールスタートからの動的なスケールアウト (マシンを追加することで系の処理能力が増加することをここではスケールアウトという) が可能である．簡単な SQL 文でフィルタ条件を指定可能な FOP である SqlFilter や，処理結果を V 組込みライブラリに送信する FOP である Emitter，PostgreSQL と接続した V，一定量を越えたデータを高速に削除するためにリングバッファを用いた V などが実装済みである．

4.2 映像監視への適用

映像の監視ストリームを下記のように DMS にマッピングさせて動作確認を行なった．

- AC として Web カメラの各フレームを JPEG 画像 (blob 型) として DMS に投入．
- FOP として連続する 2 枚の JPEG 画像の差分を比較し，閾値未満であれば，縮小画像を作成して画像一覧 V に転送．閾値以上であれば縮小画像に赤枠を付与して画像一覧 V に転送し，オリジナルサイズの画像を差分映像保存 V に転送．この際，window を利用し，FOP 内で直前の Jpeg フレームを参照．
- FOP として JPEG 画像の中から画像解析 [1] によって顔画像を検出し，検出した顔に赤枠を付与して画像一覧 V に転送．
- 画像一覧 V として Web ブラウザで多数の映像を一覧表示．
- 差分映像保存 V として一定時間のオリジナル映像を保存し，検索により表示．

図 7 は，映像監視システムの実行画面を上下に 2 画面分記載している．12 個の Web カメラ映像を情報源と考え，上段および中段の 8 情報源が映像差分検出 (差分があった画像に枠をつける)，下段の 4 情報源が顔検出 (検出した顔に枠をつける) で動作している．さらに，その後の任意のタイミングで中段の 4 つの情報源に対する処理情報を映像差分検出から顔検出に無中断で変更した場合の画面を示している．

5. 評　　価

映像監視モデルをベースに Q と F のスケールアウト能力と情報源の追加や処理の変更に伴うオーバーヘッドの評価を行なった．共通する測定環境を以下に述べる．PC サーバ:CPU Pentium4 相当 3GHz，RAM 2GB，OS:FedoraCore5，追記データ:PPM 画像 (900KB/枚) が 1 レコード，FOP：Magick++ を用いた画像差分の抽出，ビュー部：なし (フィルタ部の出力は行わない)．

5.1 Q のスケーラビリティ

図 8 は，Q プロセス数に対する追記レコード数のスループットを示している．AC に 2 台用いて，各 Q に 200 レコードを投入．Q プロセスは，1 サーバに 1 つ起動．Q における永続化，書込み確認を行う．本評価では，Q のスケーラビリティを確認するため，F がボトルネックとならないように F のサーバとして 20 台を用意した．よって，本スケーラビリティは，単に追加してためるだけではなくフィルタが動作している状況において測定されている．AC で書込みを始めてから終了するまでの時間を測定しスループットに変換している．Q が系のボトルネックとなるのは，Q のプロセス数が非常に多い場合，Q の永続化負荷が非常に高い場合，Q に対する F の Pull 要求が多すぎる場合などがある．今回は Q の永続化負荷が高い場合を想定した性能測定を行なっている．DISK 書込み応答時間のバラツキはあるものの，グラフが示すとおり，Q のリソースを増やすことで，追記レコードに対する Q のスループットが増加している．

5.2 F のスケーラビリティ

図 9 が F プロセス数に対する処理レコード数のスループットを示している．AC に 1 台用いて，Q に 1000 レコードを投入．本評価では F のスケーラビリティを確認するため，Q がボトルネックとならないように Q のサーバとして 2 台を用意した．この際，Q における永続化は行っていない．F は，1 サーバに一つ起動した．Q に最初のデータが到着してから，Q 上のデータが無くなるまでの時間を測定しスループットに変換．図 9 において，F のリソースを増やすことで，処理レコードのスループットが増加していることが確認できる．

5.3 Q の追加と既存 Q への影響

図 10 は DMS の Q の追加に伴う処理スループットの変化を表している．x 軸は，時刻を示している．時刻 -10 秒から AC1 の追記を開始し，時刻 0 秒で AC2 の追記を開始した．AC1(AC2) は，それぞれ 0.2 秒 (0.1 秒) ごとに 1 回の追記を行う Q(F) は，2(5) プロセス用意し，1 プロセスに 1 サーバを用意した．1 秒毎に処理スループット値を測定し，y 軸にプロットしている．図 10 において，既に起動中の Q に対して，新たな Q による影響がほとんど無いことを示している．

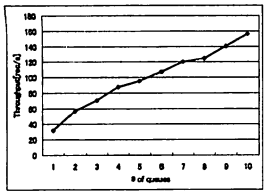

図8 Qのスケールアウト

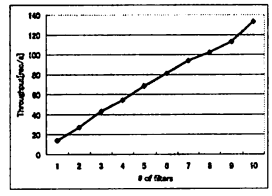

図9 Fのスケールアウト

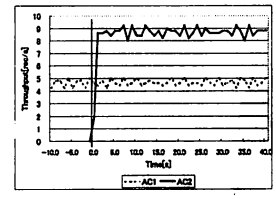

図10 Qの追加とオーバーヘッド

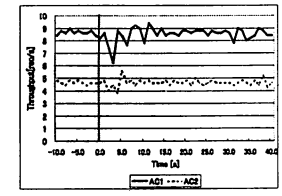

図11 Fの入れ替えとオーバーヘッド

5.4 Fの動的入替とオーバーヘッド

図11はDMSのFの動的な入替に伴う処理スループットの変化を表している．AC1(AC2)は，それぞれ0.1秒(0.2秒)ごとに1回の追記を行うQ(F)は，2(5)プロセス用意し，1プロセスに1サーバを用意した．FOP1(FOP2)は，PPM画像の差分検出であり，差分検出された場合に赤枠(青枠)を付加するオペレーションである．FOP1を含んだ実行バイナリをFOP2を含んだ実行バイナリに入れ替える．図11での測定方法について以下に述べる．時刻-10秒からAC1およびAC2の追記を開始．FOP2を含んだ実行バイナリをIに配置，処理制御情報のFOP1をFOP2に変更し，情報源IDのバージョンを変更し，管理コマンドで系内に通知を実施．これによりバージョンがQに通知される．Fではデータ取得時点でバージョン不整合が発生し，Mを経由してF実行モジュールの入替を実施する．その入替後のF実行バイナリの実行が開始されたタイミングを時刻0秒としている．1秒毎に処理スループット値を測定．F実行バイナリの動的な入替の際には，Fプロセスの部分的な再起動が発生しているが，このAC1のグラフが示すように，若干のスループットの変動は見られるものの，大きな変動は見られない．また，同時に動作しているAC2からの処理に関してもスループットへの影響がほとんど見られない．

6. 関連研究

これまで，STREAM[7]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[2],[4],[6]〜[8]．これらは，与えられたクエリに対するプランナの構成や最適化を行っている．応用サービスにおいて，既に存在するライブラリをデータストリームに適用するというニーズがあり，拡張SQLのみではそれらのライブラリを直接組み込むことは難しかった．DMSでは，カラム，レコード及びスキーマは規定しているが，処理自体は一般的なライブラリやアルゴリズムを組み込むことが可能となっている．分散DSMSとしては，Borealis[4]などがあるが，それらは主にネットワークコスト(スループットやレイテンシ)の最小化という観点からオペレーションの配置を最適化をはかるものである．このような課題は，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[3]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，F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F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．現時点では，joinに相当するオペレーションは提供されていない．MapReduce[5]は，mapにおいて単純なオペレーションを行い，reduceにおいてそれらをまとめるという処理モデルを採用している．DMSでは，Fにおいて，単純な処理を行い，Vにおいてそれらをまとめるという処理モデルであり類似性がある．しかし，MapReduceは，蓄積データに対して処理のスケーラビリティを求めており，動的な負荷分散に対応することは考えられていない．MapReduce及びDMSでは，join相当のオペレーションが存在しない．

7. まとめ

センサからの追記型のデータに対して，蓄積・管理するとともに，利用者の目的に合った形で情報を統合し，検索や分析といった参照を可能にするプラットフォームの構築に向けて，DMSアーキテクチャの提案を行った．今後は，センサ情報の負荷変動に対して適応型負荷分散方式，及び，スケーラビリティを保持しつつjoin相当のオペレーションを提供できるようなアーキテクチャを検討して行きたい．

文献

[1] 遠隔映像モニタリング高付加価値化技術. http://www.ntt.co.jp/RD/OFIS/active/2006pdf/hot/ap/04.html.

[2] 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.

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

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

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

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

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

[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.