実世界から発生するビッグデータをリアルタイムに処理するストリーム分析基盤

喜田　弘司[†a)]　藤山健一郎[†]　中村　暢達[††]

Data-Stream Processing Platform for Analyzing Huge-Scale Continuous Sensor Data in Real-Time

Koji KIDA[†a)], Kenichiro FUJIYAMA[†], and Nobutatsu NAKAMURA[††]

あらまし　ユビキタス社会においてますます増大しつつある，センサ，端末等から連続かつ大量に発生する比較的小さなサイズのデータ（データストリーム）を高速に収集・分析する技術を提案する．本技術は，従来のソリューションのようにいったんデータベースにデータを蓄積して分析するのではなく，大量に連続発生するデータを収集する過程でオンザフライに逐次的に分析することにより，桁違いに高速な処理が可能となる．我々は，ストリーム処理をより簡単にシステム開発できるように，データストリーム処理基盤（DSPP）を開発した．DSPP は，複数の計算機で分散実行される処理ノードの負荷を分散させるフロー制御機構や，データストリーム処理用のメモリ管理機構をもつことが特徴である．これらにより，データの流量や計算機リソースなどが動的に変化した場合においても高性能に分散処理できることを実車による ITS 実験データをもとに検証した．

キーワード　データストリーム，ビッグデータ，プローブ情報，ITS

1. まえがき

2020 年までに全世界で生成されるディジタルデータは 35 ZB になるという試算がある [1]．こうした，従来の常識を超えるような大量データを最近は「ビッグデータ」と呼び，ビッグデータを活用したサービスの創出，価値創造への期待は大きく膨らんでいる．

ビッグデータの活用対象はインターネット上のデータにとどまらず，実世界をセンシングしたデータにも及んでいることが特徴である．こういった実世界から発生したセンサデータは「データストリーム」と呼ばれ，リアルタイムに処理することにより，実世界で「今，何が起きているのか」を認識し安全・安心，便利・快適，正確・効率，エコ・省エネに活用することが期待されている．例えば，在庫管理において，従来は 1 週間単位に把握すればよかったものが，現在では数分単位にリアルタイムで処理し，きめ細かくビジネスの効率化や最適化を図ることが期待されている．

ところが，データストリームをリアルタイムに処理するシステムを開発することは簡単ではない．データストリームは，たとえ一つのデータは数百バイトと小サイズであっても，数百万箇所からデータを収集すると，秒当り数十万件，数十万メガバイトのデータを処理する必要があり，これらをリアルタイムに処理することは非常に難しい．

この課題に対し，我々は，根本的にリアルタイム性を向上させるためには，システム・アーキテクチャから見直す必要があると考える．従来のソリューションのようにいったんデータベースにデータを蓄積して分析するのではなく，大量に連続発生するデータを収集する過程でオンザフライに逐次分析するフロー型のアーキテクチャにより，桁違いに高速な処理が可能となる．

本論文では，フロー型のアーキテクチャでより簡単にシステムを開発できる基盤ソフトウェアであるデータ

† 日本電気株式会社クラウドシステム研究所，川崎市
Cloud System Research Laboratories, NEC Corporation, 1753 Shimonumabe, Nakahara-ku, Kawasaki-shi, 211–8666 Japan
†† 日本電気株式会社第三 IT ソフトウェア事業部，川崎市
3rd IT Software Division, NEC Corporation, 1753 Shimonumabe, Nakahara-ku, Kawasaki-shi, 211–8666 Japan
a) E-mail: kida@da.jp.nec.com

 ⓒ一般社団法人電子情報通信学会 2012

ストリーム処理基盤 DSPP（Data Stream Processing Platform）を提案する．更に実車を使った ITS 実証実験のデータ用いて本方式のスケーラビリティに関する性能評価結果を報告する．

2. データストリーム処理

本章では，データストリームをオンザフライに逐次的に処理するフロー型の計算モデルを述べる．

2.1 ストック型の処理とフロー型の処理

実世界から発生したデータストリームは，実世界の「今」の状況を表しておりリアルタイムで処理・活用されてこそ意味がある．ところが従来方式は，データをいったんデータベース等へ蓄積し，この蓄積したデータを後で一括処理するいわゆるバッチ処理による分析が主流であり，こういった “ストック型” のアーキテクチャはリアルタイム処理には向かない．リアルタイム性の向上を目指し現在のシステム構築のトレンドは，Hadoop [2]，KVS，メモリデータベースを使う方式が実用されている．しかし，こういった方式もデータをいったんストックするアーキテクチャであることは変わらず，夜間バッチによる 1 日の処理タイムラグが，数時間のタイムラグに改善される程度のリアルタイム性を実現するにすぎない．

これに対し，根本的にリアルタイム性を向上することを目的に，データを一括処理する “ストック型” のアーキテクチャではなく，データの発生時にリアルタイム処理する “フロー型” のデータ処理アーキテクチャ（= ストリームデータ処理）がある（図 1）．

ストック型の処理がクエリ発行時に全データを見て全結果を抽出する一括処理であるのに対し，ストリームデータ処理はデータ到達時にそのデータを見て関係する処理のみを実行する差分処理をすることにより，リアルタイム処理を実現する．

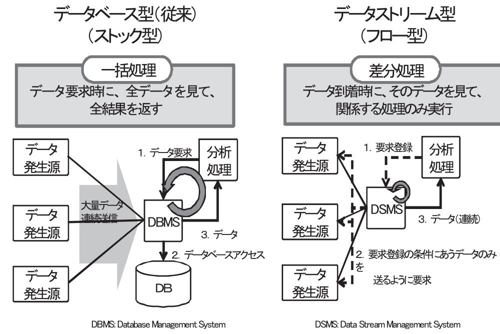

図 1 ストック型とフロー型のアーキテクチャ比較
Fig. 1 Data stock architecture and data flow architecture.

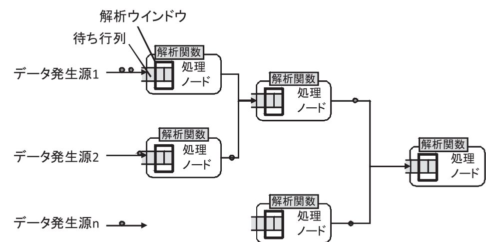

図 2 データフロー計算モデル
Fig. 2 Data flow computation architecture.

2.2 データフロー計算モデル

データストリーム処理を実現するには，ユーザは図 2 に示したデータフロー計算モデルでプログラミングをする必要がある．

本計算モデルでは，処理ノードと呼ばれる解析処理の単位を複数用意し，これらを処理する順にリンクすることで処理ノードネットワークを構築する．各処理ノードは，解析待ちイベントデータを記憶する待ち行列と，待ち行列のデータから一度に解析するデータセットを生成する解析ウインドウ生成部と，待ち行列のデータを解析する解析関数とから構成される．

各処理ノードの待ち行列へは，前にリンクされている処理ノードから非同期にデータが追加される．解析ウインドウ生成部は，待ち行列の状態を常に監視し，一度に解析するデータがそろえば，解析関数に対象データを渡し解析を依頼する．解析関数は渡されたデータを解析し，後ろに接続された処理ノードの待ち行列へ解析結果を書き込む．これを繰り返し，全体として解析が行われる．

3. データストリーム処理の課題

前記のように，データストリーム処理はデータを蓄積する前に処理することにより，蓄積するオーバヘッドがなくリアルタイム処理に向いている．しかしデータストリーム処理をすれば高性能なリアルタイム処理ができるとは限らない．本章では，ITS システムを例にデータストリーム処理でリアルタイム処理を実現するための課題を明らかにする．

3.1 ITS システムの例

ITS システムとは，クルマをセンサに見立て，クルマからアップロードされる位置，速度，ワイパーの利用状況などの車両情報をリアルタイムに分析し，渋滞情報や天候情報を生成するシステムである．これを実

現するシステムの構成をデータフロー計算モデルで考えると図3のようになる．各クルマがデータの発生源であり，これをサーバで受け取り，地図上のどこを走行しているのか判定するマップマッチングと呼ばれる処理を行い，最後に，渋滞情報や天候情報などを生成する各アプリケーションプログラムへデータが流れる．

3.2 課題分析

図3は，あくまでも論理的なアーキテクチャである．これを実現させるためには，物理的に大量の計算機でどう分散動作させるか設計する必要がある．この際，以下の要件を満たしながらスループットなどの性能が高くなるように設計する必要がある．

[要件1] データの発生頻度が変化してもそれに応じて高性能であること．特に，事前に発生頻度が不明な場合においても高性能である必要がある．例えば，駅前などデータ量が多いと事前に予測できる地域もあれば，渋滞が起き突発的にデータ量が増える場合もある．すなわち，設計時に想定できないデータ量の動的変化にシステムが対応して性能劣化を抑え高性能であることが必要である．

[要件2] ユーザが多くなった場合などサービスを拡張する際に，再設計の必要がないこと．計算機の台数を増やすだけでそれに比例して性能があがる必要がある．また，計算機の故障などにより，台数が減る場合もあり得る．すなわち，計算機の構成変更にシステムが対応して高性能であることが必要である．

以上のように，データストリーム処理システムの課題は，データを流し処理が実行されている段階で，データの流量や計算機リソースなどが動的に変化した場合においても高性能に分散処理できることである．

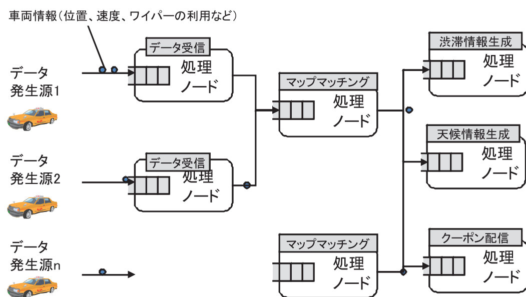

図3 ITSシステムのデータフロー
Fig. 3 Data flow computation architecture for ITS system.

4. データストリーム処理基盤（DSPP）

4.1 設計方針

DSPPでは，前記の要件を可能な限りシステム構築者が意識することなく構築できることを目的とする．すなわち，システム構築者は，DSPPのAPIを使って設計を行うと，DSPPがシステム実行時に，データや計算機リソースなどが動的に変化した場合においても，DSPP内でこれらの変化を吸収し高性能に分散処理できるようにする．

4.2 DSPPの概要

DSPPのアーキテクチャを図4に示す．システム構築者がDSPPのAPIを使ってデータフロー計算モデルでシステムを設計する「ユーザレイヤ」と，これを実際の計算機で実行し，実行状態を監視して，高性能に実行できるように調整する「実行レイヤ」からなる．

4.2.1 DSPPのユーザレイヤ

ユーザレイヤでは，C++のクラスライブラリとして提供されるDSPPのAPIを使ってシステム構築者はプログラムを作成する．具体的には，処理ノードネットワークの設定，各処理ノードのコーディング，各処理ノードの実行計算機条件の指定の三つを開発する．

まず，処理ノードネットワークの設定は，システム全体をどのような処理ノードに分けるのが良いのか検討し，処理ノードのネットワーク構成を設計する．図3の例では，データ収集ノード，マップマッチングノード，ユーザ集計ノード，渋滞計算ノード，サービスノードからなる．

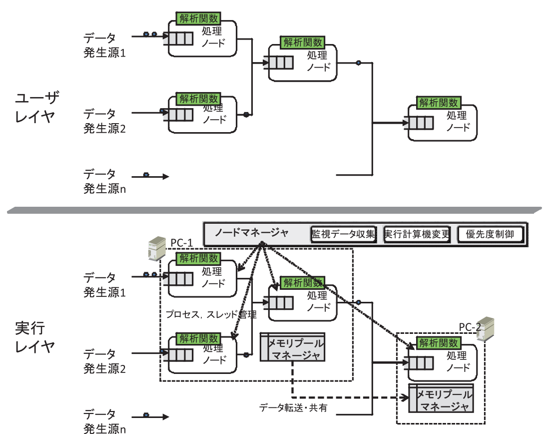

図4 DSPPアーキテクチャ
Fig. 4 DSPP (data-stream processing platform) architecture.

次に，各処理ノードのコーディングは，前記の処理ノードネットワークを構成する各処理ノードをプログラミングする．処理ノードの基底クラスは，前記のデータフロー計算モデルに従い，解析待ちイベントデータを記憶する待ち行列，この待ち行列のデータから一度に解析するデータセット取得する解析ウィンドウ，待ち行列のデータを解析する解析関数とから構成される．解析関数を処理ノードごとにコーディングすることで処理ノードをプログラミングする．例えば，図3のマップマッチング処理ノードは，イベントデータの位置情報のフィールドから，その位置とマッチした地図上の道路のIDを計算し，イベントデータにこの道路IDを付加する解析関数を作成する．

最後に，各処理ノードの実行計算機条件の指定は，処理ノードごとに実行する実際の計算機を指定する．この指定は，必ずその計算機で実行する必要がある場合（計算機指定）と，いくつかの計算機群を用意しておき，どの計算機でもかまわない場合（機能指定）とを指定できる．図3の例では，データ収集ノードは，クルマの位置周辺の基地局に置かれた計算機で受信するように計算機指定し，マップマッチングは，どの計算機で実行してもかまわないため，マップマッチング用の計算機群を機能指定する．更に，ユーザ集計ノードは，どの計算機で実行してもかまわないが，イベントデータのユーザIDの値が同じデータは同じ計算機で実行するように条件パラメータを付けて機能指定する．

4.2.2 DSPPの実行レイヤ

実行レイヤは，C++のラインタイムライブラリとして提供され，ユーザレイヤで作成したプログラムを実行するために，メモリ管理機能とプロセス管理機能を提供する．

まず，メモリの管理機能はデータストリーム処理に特有の以下の問題を解決する機能を提供する．

- システムコールのオーバヘッド問題：小サイズのデータが大量に発生し大量に削除されるため，メモリの確保，解放のシステムコールが多く呼び出され，処理が重くなる．
- メモリリーク問題：データストリーム処理は，大量のデータの繰返し処理であり，プログラマのミスによるメモリリークは致命的である．
- メモリ転送問題：論理的にはある処理ノードから別の処理ノードへのデータの転送であるが，物理的には同じ計算機内の転送もあれば，異なる計算機間の転送もあり，これらを透過的に扱いたい．

以上の観点から，OSに標準の汎用的なメモリ管理を使うのではなく，データストリーム処理に特化したメモリ管理を行う（**4.4**参照）．

次に，プロセス管理機能は，処理ノードの実行状態を監視し適宜処理ノードを制御する機能（ノードマネージャ）である．例えば，要件1で挙げた例のように，突発的に，ある処理ノードへ流れるデータ量が多くなり負荷が高いことが検知されれば，別の負荷の低い処理ノードを実行している計算機に，この処理ノードも実行するように物理構成を変更することで，全体の負荷のバランスをとる．以上のようにノードマネージャは，各処理ノードの待ち行列があふれないように，あるいは系全体のパフォーマンスを向上させるために，処理ノードの優先度や，各処理ノードを実行する計算機の割当を変更する（**4.3**参照）．

4.3 ノードマネージャによる処理ノードフロー制御

本節では，ノードマネージャが行う処理ノードフロー制御のアルゴリズムを述べる．処理ノードのフロー制御は，各計算機内で動作させるサブノードマネージャと，処理ノードネットワーク全体を制御する全体で一つだけ実行している統合ノードマネージャから構成される．処理ノードフロー制御のアルゴリズムは，状態把握，計画，制御の三つの機構からなる．

まず，状況把握は，サブノードマネージャが常駐し，その計算機内の各処理ノードの実行状態と，計算機リソースの実行状態を監視する．

処理ノードの実行状態は，処理ノードの待ち行列の長さの変化を監視し，あらかじめ設定された待ち行列の最大長を超えるタイミングを最小二乗法などで予測する．ある一定時間以内に処理があふれると予測された処理ノード（制御対象ノード）は，サブノードマネージャが実行されている計算機内で解決できないか計画を立て（後述），解決できない場合に統合ノードマネージャによる制御対象としてリストアップする．計算機リソースの実行状態は，SNMPを使ってCPUの負荷，ネットワークの負荷を一定期間（例えば，1分間隔）監視し，その平均値と前回との差分を計算する．

サブノードマネージャは定期的に（例えば，5分ごとに），統合ノードマネージャへ制御対象処理ノードと，計算機リソースの状態を報告する．

次に，計画は，処理ノードのフロー制御は，1台の計算機内の制御をサブノードマネージャで行い，複数台にまたがる制御を統合ノードマネージャで行う．

サブノードマネージャは，その計算機内で動作して

いる他の処理ノードとのプロセスの優先度を調整することにより解決を図る．具体的には，制御対象処理ノードと他の処理ノードとの接続関係から，制御対象処理ノードと接続していない処理ノードを選び出し，この中から処理があふれると予測されていない処理ノードのプロセスの優先度を落とし，制御対象ノードの優先度を上げる計画を立てる．もし，負荷の低い処理ノードが見つからなかった場合には，この計算機内では調整不可能と判断し，統合ノードマネージャへ調整を依頼する．

統合ノードマネージャは，処理ノードが実行される計算機の割当を変えることで解決を図る．具体的には，制御対象処理ノードの実行計算機条件（計算機指定，機能指定，条件付き機能指定）により制御計画を立てる．計算機指定の場合，その計算機で実行することが指定されているために調整が不可能である．システムとしては管理者へ警告し，管理者が別途物理構成などの変更を検討する．機能指定の場合には，制御対象ノードを実行できる計算機の中から負荷の低い計算機へ実行を変更する計画を立てる．条件付き機能指定の場合には，条件のパラメータを調整する．例えば，制御対象処理ノードの振り分け条件のパラメータがユーザ ID が 500 から 1000 までであったとき，500 から 750 というように，制御対象処理ノードへ流れるデータが少なくなるように制御し，負荷が余裕のある計算機で実行している，別の処理ノードでユーザ ID が 750 から 1000 を処理できるように計画を立てる．

最後に，制御は，計画に従い処理ノードのフロー制御を実行する．サブノードマネージャによる優先度の変更は即実行される．統合ノードマネージャによる実行計算機の変更は，実行状態を継続しながら変更する必要があり，メモリプールマネージャ（**4. 4** 参照）と連動して実行する．

まず，制御対象処理ノードを変更後の計算機で動作させネットワークを変更元への接続を変更後の計算機につなぎかえる．ただし，変更後の処理ノードはまだ解析は行わず待ち行列にデータをため続ける．元の制御対象処理ノードは待ち行列のデータがなくなるまで処理を続ける．待ち行列のデータがなくなった段階で，メモリプールマネージャが，この処理ノードが使っているメモリを変更後の処理ノードのメモリへコピーを行う．これによりこの処理ノードの実行状態が引き継がれたことになる．最後に，変更元の処理ノードを終了させ，変更後の処理ノードの解析を開始する．

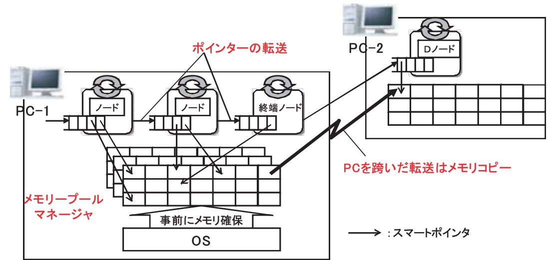

図 5　メモリプールマネージャ
Fig. 5　Memory management for datastream processing.

4. 4　メモリプールマネージャ

前記のメモリ管理の問題点の解決を目指し，メモリプールを用いたデータストリーム処理用のメモリ管理方式を提案する．

メモリプールとは，システム起動時に処理ノードで利用するメモリ全体を OS からヒープ領域として確保し以後この確保したメモリ領域を DSPP の責任でストリーム処理に利用する方法である（図 5）．

OS へのメモリ確保や，メモリ開放のシステムコールが発生しないためにシステムコールのオーバヘッドがなくなる．

一方，どのメモリ領域が使用中で，どの領域が未使用なのかの管理は DSPP で行うことになる．DSPP ではメモリ領域ごとに，処理ノードから参照されているポインタの数をカウントしている．カウントが 0 になるとどの処理ノードからも参照されていないことを意味し，未使用となる．この機構により，各処理ノードの解析関数を作成するプログラマは，スマートポインタとしてストリームデータへアクセスする．つまり，メモリの解放をプログラマが明示的にする必要はなくなりメモリリーク問題は解決する．

最後に，複数の計算機間でのメモリプールの連携機能を説明する．複数の計算機間でのメモリのコピーが必要なのは，異なる計算機上で動作している処理ノード間のリンクをデータが流れる場合と，統合ノードマネージャによるフロー制御により処理ノードの実行計算機が変更になった場合である．前者は，処理ノードの待ち行列へイベントデータを追加するタイミングで送信元の処理ノードはソケットで送信先の計算機へデータを送り，受信側の処理ノードでは，自分が管理するメモリプールへコピーする．後者は，移動元の処理ノードが参照しているメモリをシリアライズし，ソケットで送信先の計算機へデータを送り，受信側の処

理ノードでは，受信したデータをデシリアライズして自分が管理するメモリプールへコピーする．デシリアライズする際に，送信元と送信先のメモリプール内のアドレスが変わる点に注意が必要である．以上の動作は各処理ノードの解析関数を作成するプログラマは全く意識しなくてよく，メモリ転送問題は解決する．

5. 性能評価実験

5.1 実験目的

検証 1：ノードマネージャによる処理ノードフロー制御の効果をスケールアウトの観点で検証する

検証 2：メモリプールマネージャの有効性を高速性の観点で検証

5.2 実験方法

検証 1 は，名古屋地区で行った，実車を使った表 1 の ITS 実証実験システムの実データをもとに実験を行った [3]．ただし，この実験データは大規模ではないため，数日分のデータを 1 日分のデータとして扱うことにより大規模データを作成した．

検証 1 は，地域を南北，東西に 4 地域（2 × 2），9 地域（3 × 3），25 地域（5 × 5），36 地域（6 × 6）に分割し，地域ごとに計算機を配置した．図 6 にシステム構成を示す．本方式と従来方式の違いは，ノードマネージャによるフロー制御機能のオンとオフである．性能の計測は，データの流量を徐々に増していき，処理しきれない処理ノードが出現した段階を 1 秒当りの最高処理性能（スループット）とした．更に，このときの各計算機の CPU 利用率を計測した．

検証 2 は，検証 1 のシステムを計算機 1 台構成で実験を行った．本方式と従来方式の違いは，メモリプールマネージャを使った場合と，OS のシステムコールを使った場合である．検証 1 と同様の方法により，1 秒当りの最高処理性能（スループット）を計測した．更にこのときの各計算機の CPU 利用率を計測した．

5.3 実験結果

[検証 1 の結果]

図 7 にスループット性能の実験結果を示す．横軸が計算機の台数であり縦軸がスループットである．データとデータの間は，二次元の多項式近似曲線である．

本方式は，計算機の台数に対してほぼリニアに性能が上がっているのが分かる．一方，従来方式は，25 台目あたりで性能が頭打ちになっている．本方式と従来方式の性能差は，台数が多いほど差が開き，36 台では 2 倍強の性能差であった．また，従来方式（チューニング）は，名古屋駅周辺など計算機のリソースが不足しがちな 3 箇所の計算機リソースを増やすチューニングを行った．従来方式よりは性能は改善されたが，やはり 28 台目付近から性能が頭打ちになっている．本方式との性能差は，36 台構成時点で本方式が 1.6 倍程度高速である．

図 8 に計算機の CPU の利用率の実験結果を示す．横軸が計算機の台数であり，縦軸が CPU 利用率である．データとデータの間は，二次元の多項式近似曲線である．本方式がほぼ 100%利用されているのに対し，従来方式は台数が増えるほど利用率が下がっている．

表 1 実データ
Table 1 Stream data specification of the field trial system.

車両数	4000 台
収集データ	位置，速度，方向
データサイズ	128 バイト
収集頻度	車両から 1 分間隔でアップロード
道路区間数	12 万区間

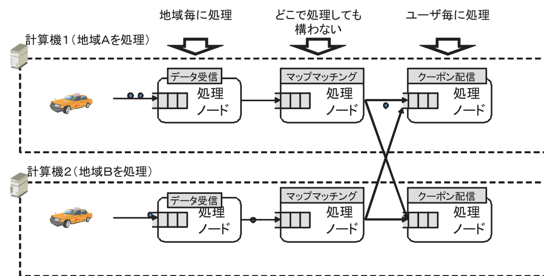

図 6 実験システムの構成
Fig. 6 Configuration for the evaluation experiment.

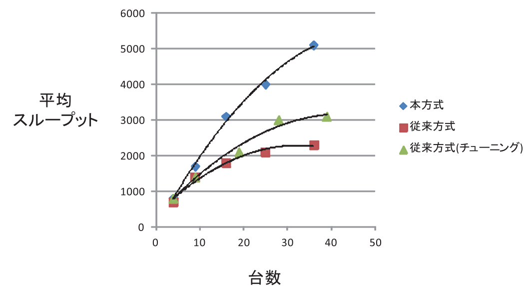

図 7 実験結果（スループット）
Fig. 7 Results (Throughputs).

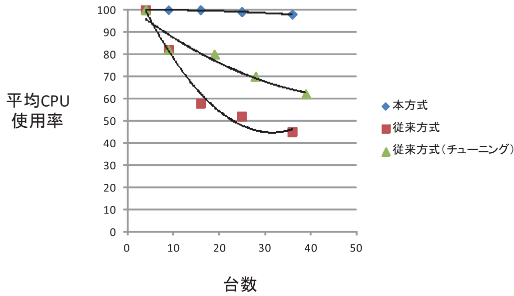

図 8　実験結果（CPU 使用率）
Fig. 8　Results (CPU usage rate).

[検証 2 の結果]

本方式の平均スループットは，平均毎秒 221 処理であるのに対し，従来方式は，平均毎秒 30 処理であった．CPU 使用率は共に 100%であった．CPU の使用率の内訳を調査すると本方式が Privileged Time が 10%程度であるのに対し，従来方式が 80%を超えていたことから，本方式はシステムコールのオーバヘッドが改善されていることが分かる．

6. 考　　察

6.1　課題の検証

3.2 で設定した二つの要件に関して検証する．

要件 1 に関して，従来方式は，流量が多い地域の計算機の負荷が高くなりこれをバランスできる機構がないためにスループットが低くなったと考えられる．一方，本方式は，データの流量に地域ごとにばらつきがあるにもかかわらず，どの計算機もほぼ 100%の CPU 使用率であり，うまくバランスできていると考えられる．したがって，要件 1 のデータ量の動的変化にシステムが対応しているといえる．

要件 2 に関して，図 7 より計算機の台数に応じてほぼリニアにスループットが上がっていることから，要件を満たしているといえる．

6.2　スケールアウト性能の特性

6.1 では図 7 より，ほぼスケールアウトすることを述べたが，厳密には，36 台構成では，理想的には秒 7200 処理に対し実際には秒 5100 処理であり，70%程度の性能に落ちている．一方で，図 8 より CPU の使用率はほぼ 100%であることから，フロー制御機能により CPU 負荷を分散させることに成功しているといえる．詳細データは示していないが，ネットワークはいずれの実験も余裕があったので，大規模化した際の限界性能は，CPU の性能に起因といえる．また，フロー制御機能の CPU のオーバヘッドにより 36 台規模で 30%程度の性能が落ちるといえる．

今回は，一つのデータサイズは 128 バイトと小さいためにこういった結果となったが，データサイズが大きい，あるいはネットワークの性能が低い場合には，CPU ではなくネットワークがボトルネックとなり，スケールアウトの限界性能が決定すると考えられる．

6.3　関 連 研 究

データストリーム処理の研究は，米国を中心に研究プロジェクトがある [4], [5]．これらプロジェクトでは，データストリーム処理のアーキテクチャ，連続問合せ言語及び問合せ処理の最適化，マイニングと多岐にわたるが，いずれも基礎研究であり実システムへの応用例がない．一方，実システムへの応用としては，金融分野において，株価の変動に応じてルール処理により株の売買するアルゴリズムトレードがよく知られている．しかし，汎用技術ではなく専用のハードウェアも含めた作り込みによる高速化である．

これに対し本研究の成果は，データストリーム処理に関して，特別な知識がなくても簡単に高性能なシステムを開発できることが特徴である．データストリーム処理システムの開発で，特に難しいのは，データの流量や計算機リソースなどが動的に変化した場合においても高性能に分散処理できることである．これに対し，DSPP は，可能な限りシステム構築者がこういったことを意識することなく構築できるように工夫されている点が特徴である．

6.4　適 用 領 域

本論文ではデータストリーム処理に関して ITS システムを例に述べたが，データストリームの活用は様々な分野で期待されており，実証実験等がなされている．

サーバ運用管理：サーバのアクセスログ，エラーログなどから，障害解析，運用改善などに活用 [6]．

モバイル広告：携帯電話ユーザへ，そのユーザの状況に適した広告を携帯電話へ配信 [7]．

WEB アクセス解析：WEB サイトのアクセス履歴や，検索キーワードをリアルタイムに分析して宣伝効果の高いバナー広告を表示 [8]．

環境監視：環境センサからのデータを入力とし，CO_2 の発生状況把握，電力消費などを監視．

見守りサービス：子供，高齢者のもつ携帯端末や街中のセンサ，カメラなどで見守り，安全・安心の社会作りに活用．

金融：時々刻々と変化する金融商品の価格を入力とし，アルゴリズム・トレーディング，リアルタイム・リスク分析，各種金融指標の算出へ活用．

流通：配送物の位置，IC タグによる出入管理を入力とし，配送状況の把握，商品発注・配送計画の策定へ応用．

7. むすび

ストリーム処理システムをより簡単に開発できるデータストリーム処理基盤（DSPP）を提案した．DSPP は，複数の計算機で分散実行される処理ノードの負荷を分散させるフロー制御機構や，データストリーム処理用のメモリ管理機構をもつことが特徴である．これらにより，データの流量や計算機リソースなどが動的に変化した場合においても高性能に分散処理できることを実車による ITS 実験データをもとに検証した．

今後，更なる高速化のためには，各解析処理のアルゴリズムを工夫する方法が考えられ，我々も，マップマッチングのアルゴリズムを研究している [9]．また，データベースではなく，ネットワークのボトルネック，すなわち広域に発生するデータの収集の高速化の研究も始めている [7]．

謝辞　本研究開発成果の一部は，平成 20～22 年度総務省委託研究「ユビキタスサービスプラットフォーム技術の研究開発」による．

文　献

[1] IDC と EMC の共同調査報告書，“The diverse and exploding digital universe,” http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf, 2007.

[2] Apache ソフトウェア財団，Hadoop 公式サイト http://hadoop.apache.org/

[3] 姚　恩建，佐藤　影，“プローブ情報活用システム「PROROUTE」の開発，” NEC 技報 ITS 特集，vol.61, no.1, pp.35–39, 2008.

[4] Stanford Univ., “STREAM project homepage,” http://www-db.stanford.edu/stream/

[5] MIT, Brown Univ., “Aurora project homepage,” http://nms.lcs.mit.edu/projects/borealis/

[6] 中村暢達，喜田弘司，竹村俊徳，藤山健一郎，“大規模 VM 負荷予測配置制御技術によるシンクライアントデータセンタのグリーン化，” NEC 技報，vol.62, no.3, pp.101–104, 2009.

[7] 見上紗和子，佐藤　正，落合敏功，喜田弘司，“スケーラブルなストリーム Pub/Sub プラットフォームの開発と評価，” 信学技報，IN2011-04，April 2011.

[8] 今井照之，海老山知生，喜田弘司，藤山健一郎，中村暢達，“データストリーム処理手法を用いた Web アクセス解析システム，” 情報処理学会，情報科学技術フォーラム（FIT），2009.

[9] 喜田弘司，藤山健一郎，三津橋晃丈，中村暢達，“次世代プローブ情報システム (2)～大規模高速マップマッチングアルゴリズムの提案，” 情報処理学会，マルチメディア分散協調とモバイルシンポジウム論文集，2007.

（平成 24 年 4 月 2 日受付，6 月 25 日再受付）

喜田　弘司

1993 岡山大学大学院情報工学研究科修士課程了．同年日本電気（株）入社．エージェント，検索エンジン，レコメンドエンジン，システムセキュリティの研究に従事．現在，M2M システムの研究に従事．博士（工学）．

藤山健一郎

2001 筑波大学大学院工学研究科博士課程修士了．同年日本電気（株）入社．現在，M2M，データストリーム処理の研究に従事．

中村　暢達

1991 東京大学大学院工学研究科精密機械工学専攻修士課程了．同年日本電気（株）入社．マルチメディア通信，ユーザインタフェース，コンテンツ保護，耐障害システム，ユビキタスシステムに関する研究に従事．2007 奈良先端科学技術大学院大学博士後期了．現在，日本電気（株）第三 IT ソフトウェア事業部．博士（工学）．