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

2008－ITS－34（1）
2008/9/10

データストリーム処理による 大規模プローブカーシステムの開発と評価

喜田 弘司　藤山 健一郎　今井 照之　中村 暢達

NEC サービスプラットフォーム研究所 〒630-0101　奈良県生駒市高山町 8916-47

E-mail: {kida@da, k-fujiyama@cw, t-imai@cp, n-nakamura@ab}.jp.nec.com

あらまし　車両をセンサーに見立て車両の走行データをサーバで分析することにより渋滞情報などの交通情報を提供するプローブ情報システムが期待を集めている．従来のプローブ情報システムは地域や車両を限定した高々1 万台程度の小規模な実証実験システムであり，実用化するためには大規模化へ対応することが重要な技術課題のひとつである．この技術課題の解決を目指し本稿では，大量かつ連続して発生するイベントデータ（データストリーム）を，データベースを使うことなく並列分散処理する基盤ソフトウェア(DSPP)を使った大規模プローブ情報システムを提案する．DSPP の特徴は，(1)データフローアーキテクチャ上でのパイプライン処理，(2)逐次処理による時系列データ分析，(3)データベースを使わないオンメモリ処理であり，本基盤を用いることで高速化が可能となる．名古屋地区のプローブタクシー1700 台を使った実証実験システムを DSPP 上で再構築した結果，データベースを使った従来のシステムと比較し，(1)により 6.0 倍，(2)により 4.6 倍，(3)により 5.8 倍の高速化に成功し，これらの組み合わせで最大 160 倍の高速化の見込みを得た．

キーワード　データストリーム，プローブ情報，高度道路交通システム

Development and Evaluation of High Performance Floating Car Data System Based on Data-stream Processing

Koji KIDA, Ken-ichiro FUJIYAMA , Teruyuki IMAI and Nobutatsu NAKAMURA

NEC Service Platforms Research Laboratories　8916-47 Takayama-Cho, Ikoma, Nara, 630-0102 Japan

Abstract　Floating car data systems that regard each vehicle as moving sensors and provide traffic, weather information, etc. generated by analyzing floating car data are attracting much attention. In this paper, we propose a high performance floating-car data analyzing technologies for next generation floating-car systems that have an important issue of handling of very large-scale data such as several hundreds of thousands of floating-cars' data on a national basis. To tackle the issues, we apply our data stream processing platform (DSPP) that is based on inbound processing model that data records are processed before they are stored in a repository. The DSPP has following three key technologies: (1) Dataflow architecture and pipe-line processing, (2) Successive analyzing, and (3) On-memory processing. The prototype floating-car system based on the DSPP has successfully worked. The floating-car system estimates traffic jam according to probe data of 1700 floating-taxes in Nagoya. Our experience shows that the performance of our proposal method is excellent since its processing time is only 1/160 of the processing time of a conventional system.

Keyword　Data-stream, Floating-car, ITS

1. はじめに

道路を走っている一台一台の車両をセンサーに見立て，サンプリングした走行データをサーバへ収集・分析することにより渋滞情報や，気象情報などを提供するプローブ情報システムが期待を集めている．プローブ情報システムは従来の交通情報提供サービスと比較し，リアルタイムかつ高精度な情報提供が可能であり，ドライブが快適になるだけではなく，省エネルギーなど社会的にも大きな貢献が期待されている．

プローブ情報システムは現在，その有効性を検証している段階であり様々な実証実験が実施されている[1][2][3]．例えば，[1]では，名古屋地区のタクシー1700台からの車両位置，速度のプローブ情報を使って 5 分周期で渋滞情報を生成できることを検証している．

実用化を目指した次のステップは，全国規模で数百万台のプローブ情報を解析することであり，大規模化への対応が重要な技術課題である．プローブ情報は，たとえひとつのデータは数百バイトと小サイズであっても，数百万カ所からデータを収集すると，秒当たり数十万件，数十万メガバイトのデータを処理する必要があり実現が非常に難しい．

この問題に対し我々は，大量かつ連続して発生するイベントデータ（データストリーム）を高速に処理する基盤ソフトウェアの研究開発を進めており，この基盤ソフトウェアを使って，[1]のプローブ情報システム（以下，名古屋実証実験システムと呼ぶ）の高速化を試みた．

本稿では，大規模データの収集・分析基盤であるデータストリーム処理基盤 DSPP(Data Stream Processing Platform)の技術を紹介し，DSPP を用いて試作した渋滞情報提供システムについて方式，性能評価結果を報告する．

2. データストリーム処理基盤(DSPP)の概要

本章では，プローブ情報システムの大規模化を支えるデータストリーム処理と呼ばれる技術に関して説明する．まずは，大量，連続発生するデータをデータの流れとして処理する「データストリーム処理」の概念を説明し，次にデータストリーム処理を実現する基盤ソフトウェア DSPP に関して説明する．

2.1. データストリーム処理とは

時系列データの処理として，従来はデータベースを利用したバッチ処理が一般的である（図１）．このアプローチは，①イベントデータをストレージに蓄え，②効率的に検索するためにインデックスを構築し，③処理プロセスがこのストレージにクエリを発行して，④クエリの結果のデータ集合を処理プロセスのメモリ空間へコピーするという処理フローになる．

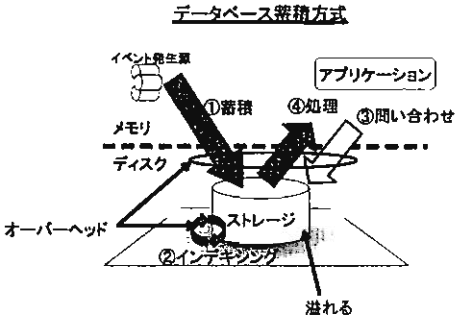

図 1　データベースによる方式

ところが，イベントデータが連続に大量に発生し解析をリアルタイムに行う必要がある場合，データベースを利用したアプローチでは以下の理由から問題がある．

問題１：①，④におけるディスクとメモリ間のデータコピーのオーバヘッドが大きい

問題２：データが更新される度に実行するインデックスの更新②のオーバヘッドが大きい

問題３：すべてのデータを蓄える必要があり，ディスクが溢れるといった問題が発生する

これに対し，データベースを用いずに，すなわちデータを貯めることなく，発生したデータをデータの流れとして順次処理する方式が「データストリーム処理」である（図 2）．

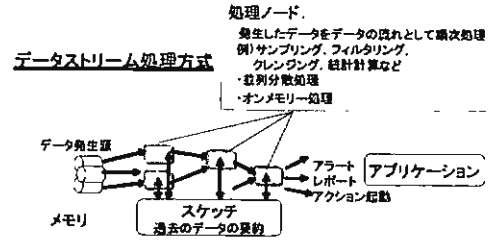

図 2　データストリーム処理による方式

通常，データの発生源は分散しており，これらを解析サーバへ収集する必要がある．このデータを収集する過程で，データをサンプリング，フィルタリング，クレンジング，統計計算などの処理を実行する．連続に大量に発生するイベントデータをリアルタイムに解析できる理由は以下のとおりである．

- DB へのアクセスがなくオンメモリ処理が可能である（問題１の解決）．
- 過去のデータへはアクセスせずに，最新のデータを逐次処理するため高速処理が可能である．過去のデータが必要な場合には，過去のデータの要約を作成しておくことで高速に処理が可能となる（問題２，３の解決）．
- 並列分散処理による高速処理が可能である．

2.2. DSPP のデータフロー計算モデル

DSPP を使ってデータストリーム処理を実現するには，ユーザはデータフローアーキテクチャと呼ばれるデータ駆動型の計算モデルを利用する（図３）．

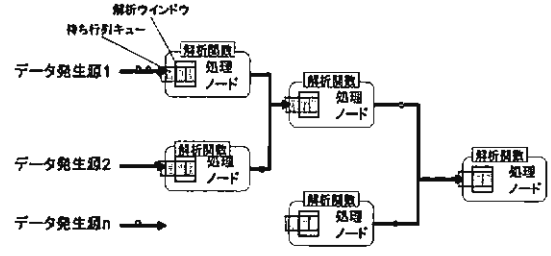

図 3　データフローアーキテクチャ

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．各処理ノードはパイプであり，解析待ちイベントデータを記憶する待ち行列キューと，待ち行列キュー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，待ち行列キュー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キューへは，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機部は，待ち行列キューの状態を常に監視し，一度に解析するデータが揃えば，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キュー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．

データフロー計算モデルをコンピュータ上で実行させるために，DSPP は，アプリケーションに依存しない基本機能を提供する（2.3 節）．この基本機能を使ってアプリケーションに依存する部分，すなわち，解析ウインドウ部，解析関数と，処理ノード間のネットワークを開発することでアプリケーションを構築する．

2.3. DSPP のアーキテクチャ

DSPP のアーキテクチャを図 4 に示す．DSPP がユーザへ提供する基本機能は以下のとおりである．

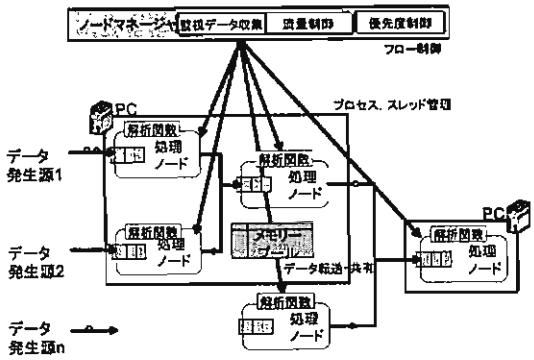

図 4 DSPP アーキテクチャ

処理ノードのベースクラスを提供：アプリケーションを構築する際には，このクラスに独自の解析関数をプラグインして各処理ノードを構築する．さらに，他の処理ノードへリンクする API を利用して処理ノードネットワークを構築する．

処理ノードの実行プロセス，スレッド管理機能：各処理ノードを非同期に独立に動作させる．

処理ノード間のデータ転送，データ共有機能：小サイズのデータを逐次処理することに特化した独自開発のメモリープールを利用して，高速にデータの受け渡しおよび，各処理ノードのキュー管理を行う．

処理ノード間のフロー制御機能：ノードマネージャと呼ばれるモジュールが，処理ノードネットワーク全体の処理の実行状態を管理する．ノードマネージャは，各処理ノードの待ち行列キューが溢れないように，あるいは系全体のパフォーマンスを向上させるために，各処理ノードへ流れるデータ量，処理ノードの処理優先度の制御を行う．

3. DSPP を用いた渋滞情報システムの高速化の提案

本章では，従来のアーキテクチャ，すなわちデータベースを使った渋滞情報システムである名古屋実証実験システムを DSPP により高速化する方法を提案する．まずは，改造の対象となった名古屋実証実験システムの構成を説明し，高速化の観点から分析する．次に，DSPP を使った高速化の方法を説明する．

3.1. 名古屋実証実験システムの概要

3.1.1. 利用イメージ

図 5 に，プローブカー情報を使った旅行時間予測結果を表示したスクリーンショットを示す．旅行時間予測とは，道路を数十メートル単位の道路セグメントに分割し，その道路セグメント間のクルマの移動時間を予測することである．図 5 では，矢印ひとつひとつがその道路セグメントの移動速度を表しており，速度に応じて色分けがされている．ユーザは図 5 の画面の左のペインから，出発地，目的地を入力することで経路案内サービスを受けることができる．カーナビなどの固定的な道案内サービスと異なり，時々刻々と変化する移動時間を考慮して最短時間で移動できる道案内ができる点が特徴である．

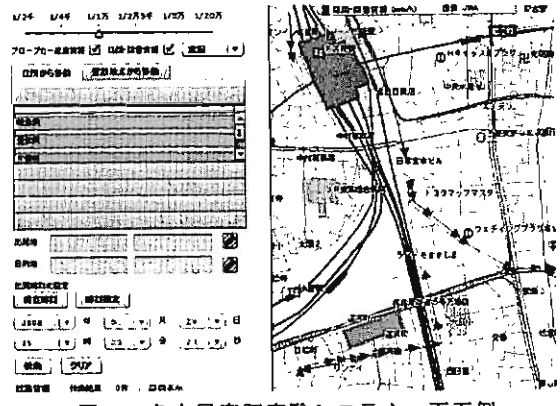

図 5 名古屋実証実験システム 画面例

3.1.2. システム構成

図 6 に，プローブ情報を分析するプローブ情報加工処理のシステム構成図を示す．タクシーから収集したプローブカー情報はいったんデータベース（プローブデータ DB）に蓄積される．プローブ情報加工処理の入力は，このプローブデータ DB からのプローブカー情報である．プローブ情報加工処理の出力は，プローブカー情報から各道路セグメントの平均旅行時間を求め旅行時間 DB として登録する．

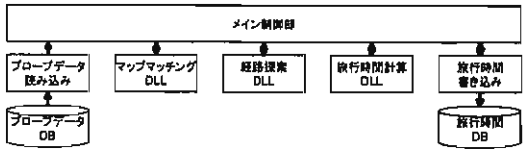

図 6 名古屋実証実験システム
プローブ情報加工処理の構成

各処理の概要は以下の通りである．

マップマッチング:プローブカー情報を地図上にマッピングする処理．プローブカー情報の位置情報は，誤差を含んでいるために，必ずしも地図の道路上あるとは限らない．そこで，プローブカー情報毎に，最短距離にある道路セグメントを検索し，道路セグメント上に存在するようにプローブカー情報の位置を補正してから，プローブ情報と道路セグメントの対応を生成する．

経路探索:複数のプローブ情報からクルマが実際に走った経路を推定する処理である．この処理が必要な理由は，プローブ情報が少ないためである．プローブカー情報は，分単位（例えば1分）にアップロードされるために，アップロードされていない期間は，プローブ情報が存在せず，旅行時間も計算できない．そこで，アップロードされていない期間のプローブ情報の位置と道路セグメントの対応関係を，経路探索により算出する

旅行時間計算:クルマごとにマップマッチング，経路探索旅行時間を計算した結果を基に，道路セグメント毎に移動時間の平均を求める処理．

メイン制御部:メイン制御部は，解析全体の流れを制御する役割である．P-DRGS は，基本的には，解析はバッチ処理を繰り返す．まず，プローブデータ DB にプローブ情報を蓄積し，ある一定量蓄積されるとそのプローブ情報全体を読み込み，マップマッチング処理を実行する．次に，この結果全体を対象に経路探索を行い，旅行時間を計算して，最後に，旅行時間 DB へ結果を書き込む．

最後に，実装方法について説明する．上記の各処理は，それぞれウインドウズの DLL として実装されている．従って，メイン制御部は，シーケンシャルに DLL へデータを渡し処理を依頼している．

3.2. 名古屋実証実験システムの速度面の課題

名古屋実証実験システムの速度面の問題点を分析すると以下の3点である．

問題点1:分析処理がバッチ処理であるため，タイムラグが発生

名古屋実証実験システムでは，処理対象のデータがそろうまで待ち，すべてそろってから処理をスタートするスケジューリングになっている．5分間のデータを基に旅行時間を計算した場合の例を図7に示す．必要なデータを蓄積している間は処理をせずに待っており，5分経過後に最近の5分のデータを順に，マップマッチング(MM)，経路探索(R)，旅行時間(TT)の計算を実行する．これらの処理時間の合計が，タイムラグとなる．

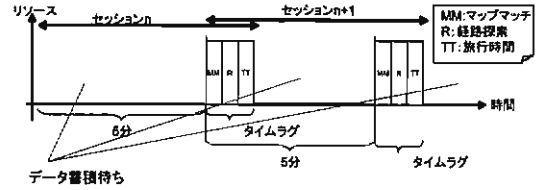

図 7 名古屋実証実験システムタイムチャート

問題点2:繰り返し処理の無駄

名古屋実証実験システムでは図8のように，データストリームをある一定サイズのウインドウを順にスライドさせて，各ウインドウ内のデータを対象データとして解析する．図8の場合，n回目の解析ウインドウとn+1回目の解析ウインドウの両方に含まれるデータが繰り返し計算されることになる．例えば，旅行時間の計算を，過去5分間のデータ集合の計算結果の平均とすると定義し，これを毎分更新する場合に，毎回，5分間分のプローブデータをプローブデータ DB から読み出し，マップマッチング(MM)，経路探索(R)，旅行時間(TT)の計算を実行する．この場合，過去4分間のデータはすでに計算済みであり無駄な処理である．

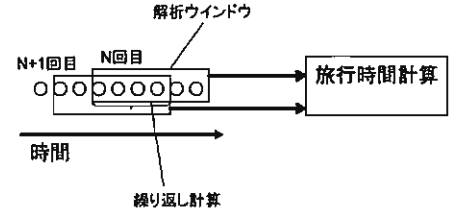

図 8 繰り返し計算説明図

問題点3:データベースアクセスの I/O オーバヘッド

データベースアクセスのファイル I/O が処理を遅くしている可能性がある．特に，データ量が多い，プローブデータ DB からのプローブデータの読み込みが問題である．

3.3. 提案システムの設計

3.3.1. 分散並列処理アーキテクチャ

図9に名古屋実証実験システムを DSPP による分散並列処理アーキテクチャで実現した構成図を示す．解析の種類毎，すなわちマップマッチング，経路探索，

旅行時間の計算それぞれが処理ノードである．各処理ノードの解析方法は，名古屋実証実験システムの解析と同じ方法にそろえるために，名古屋実証実験システムの解析ライブラリ(DLL)で実現する．

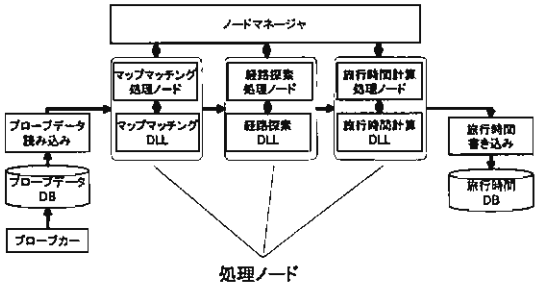

図 9　DSPP による構成

これら処理ノードは，DSPP の，ノードマネージャの管理下で並列実行する．各処理ノードは互いに独立に解析できるわけではなく順序関係があるため，他の処理ノードの処理状況次第では，「待ち」状態に陥ることがありパフォーマンスが低下する．こういったことが発生しないように，ノードマネージャは，処理ノードの処理待ち状態を極力少なくするように処理全体を制御する．

以上のように，本アーキテクチャは，解析の種類単位に処理ノードとし，これらの実行をノードマネージャで調整することで高速な並列分散解析が可能となる．

3.3.2. ノードマネージャのパイプライン処理による高速化

問題点 1 は，処理ノードによるパイプライン処理により対処できる（図 10）．パイプライン処理は，データの発生と同時に解析を順にはじめる方式である．最後のデータが発生したときには，すでに多くのデータは解析が完了しているため，タイムラグを非常に少なくすることが可能である．パイプライン処理は，各処理にかかる期間がほぼ同じである必要がある．例えば，連続している２つの処理ノードにおいて，前段の処理ノードの方が後段の処理ノードの期間より処理期間が短くて済む場合，後ろの処理ノードの処理待ちキューは溢れ，システムは止まってしまう．DSPP ではノードマネージャが調整することでこの問題を解決する．

3.3.3. スケッチを使った繰り返し計算の削除による高速化

データストリーム処理では，1 回の解析単位をウインドウと呼び，ウインドウをスライディングさせて繰り返し計算を実行する．DSPP では，過去のウインドウの計算結果を，将来の計算で再利用するために記憶しておくスケッチと呼ばれる機能を持っている．問題点２は，スケッチを使った繰り返し計算の削除により解決できる．どのようにスケッチを利用するか処理ノードの種類毎に検討する．

マップマッチングは，完全に過去のデータと独立に計算できるため特にスケッチは必要としない．経路探索は，車ごとに処理をする必要があり，前回の終点が今回の始点として経路を探索する．従って，前回のプローブ情報の位置のみをスケッチとして覚えておけばよい．アルゴリズムは，スケッチの値と最新のデータの位置で経路探索を行い，最新のデータをスケッチと置き換える．この処理をデータが来る度に繰り返せばよい．旅行時間の計算は，過去の全データの平均であるために，過去のデータの合計値とデータ数をスケッチとして覚えておけば逐次的に計算できる．

3.3.4. データベースレスによるオンメモリ処理による高速化

図 6 のように，名古屋実証実験システムでは，プローブカーからのデータはプローブデータ DB を介して解析する．このオーバヘッドをなくすために，プロー

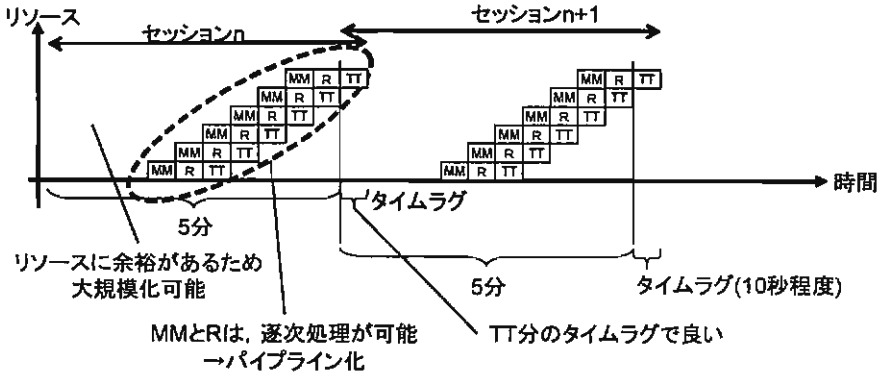

図 10 DSPP によるパイプライン処理のタイムチャート

プ情報を直接メモリ上でマップマッチング処理ノードへ入力できるように改造する．ただし，名古屋実証実験システムにこの改造をすることは難しいため，オフラインのシミュレーション環境を構築した（図 11）．このシミュレーション環境では，数時間分のプローブ情報をあらかじめプローブデータ DB からメモリに読み込んでおき，このプローブデータを定期的にマップマッチング処理ノードへ入力する．

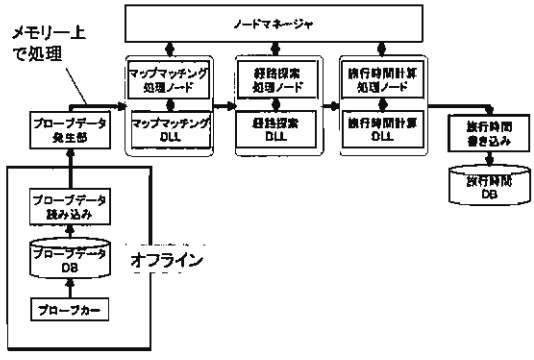

図 11データベースを利用しない場合の構成

4. DSPP の基盤技術

本章では，問題点 1，問題点 2 の解決に利用した DSPP の技術の補足説明する．

4.1. ノードマネージャによる処理ノードの調整方法

ノードマネージャは，各処理ノードの実行状態を監視し，系全体でパフォーマンスが出るように各処理ノードの動作を制御する．各処理ノードの実行状態の監視は，処理待ちキューの長さを監視し，処理ノードの制御は処理待ちキューの長さに応じて処理ノードを停止，再開の制御をする．詳細なアルゴリムズを以下に示す．

ステップ1:各処理ノードは，定期的に処理待ちキューの使用率をノードマネージャへ報告

ステップ2:ノードマネージャは，溢れそうな処理ノードを特定．溢れる基準は，「バッファサイズ」というパラメータであらかじめ外部から与えておく．

ステップ3:ノードマネージャは処理ノードの接続関係から溢れそうな処理ノードの上位ノードを特定する．

ステップ4:溢れそうな処理ノードの上位ノードに停止命令を出す．

ステップ5:停止命令を受けた処理ノードは処理を一時停止する（一時停止状態のときには，バッファへのデータの追加はできるがバッファからデータを取り出し解析することはできない）．

ステップ6:ノードマネージャは，一時停止した処理ノードの一覧を記憶しておく．

ステップ7:ノードマネージャは，一時停止した処理ノードの処理待ちキューの使用率がある一定値以下になると，解析を再開させる（この閾値を「再開閾値」と呼ぶ）．

ステップ8:処理ノードは，再開命令を受け取ると，処理を再開する．

4.2. スケッチによる計算結果の再利用方法

データストリーム処理は，到着したデータへ繰り返し問い合わせを適用する．問い合わせのタイミングは，ある一定期間毎に問い合わせたり(Time-Base Window)，ある一定量のデータ到着毎に問い合わせる(Tuple-Base Window)方法がある[4]．この一回の解析の範囲をウインドウと呼び，次々に到着するデータに対してウインドウを順にスライディングさせて解析する．例えば，過去 5 分間のデータを使って毎分旅行時間を更新する場合には，ウインドウの種類は Time-Base Window であり，ウインドウのサイズは 5 分であり，ウインドウのスライディング幅は，1 分となる．

ウインドウのサイズよりウインドウのスライディング幅の方が小さい場合，同じデータを複数回解析することになる．この無駄をなくすために，過去のデータの解析結果をスケッチとして記憶しておき再利用する．スケッチを使った計算方法は以下のとおりである．

ステップ1:n-1 回目の解析以降に追加された最新のデータを得る

ステップ2:n-1 回目までのスケッチデータを得る

ステップ3:ステップ 1 の最新のデータと，ステップ 2 のスケッチのデータから解析する．

ステップ4:n 回目のスケッチデータを作成する．

5. 性能評価実験

我々は，3 章で提案した DSPP によるデータストリーム型のプローブ情報解析システムを，実際の名古屋実証実験システムとまったく同じ環境で動作させた．本章では，その性能評価を説明する．

5.1. 実験目的

3 章で提案した以下の 3 つの技術の有効性を検証することが，本実験の目的である．

検証-1:分散並列処理アーキテクチャとノードマネージャのパイプライン調整による高速化(3.3.1 節，3.3.2 節)

検証-2:スケッチを使った繰り返し計算の削除による高速化（3.3.1 節）

検証-3:データベースレスによるオンメモリ処理による高速化（3.3.4 節）

5.2. 実験方法

5.2.1. 予備実験

パイプライン処理のパラメータを予備実験によりチューニングした．設定するパラメータは，ノードマネージャが処理ノードを調整するための以下の２つである．

バッファサイズ:各キューの処理待ちのデータの最大数を意味するパラメータ．

再開閾値:停止した状態を再開するタイミングを設定するパラメータ．キューにいくつのデータが残っているときに再開するかを設定する

以下の本実験では，この予備実験でチューニング済みのシステムで評価する．

5.2.2. 本実験

実験１:検証-1 の実験

以下の２つのシステムの処理時間を測定する．

- DSPP-NO-SKETCH：ノードマネージャで制御し，かつスケッチ機能を OFF にした DSPP
- 名古屋実証実験システム

データは，実際のデータ１日分であり，最も混雑期のデータであり，名古屋実証実験システムでは処理が間に合わなかった日である．

実験２:検証-2 の実験

以下の２つのシステムの処理時間を測定する．

- DSPP-NO-SKETCH：ノードマネージャによるパイプライン処理制御し，かつスケッチ機能を OFF にした DSPP
- DSPP-WITH-SKETCH：ノードマネージャによるパイプライン処理制御し，かつスケッチ機能を ON にした DSPP

旅行時間の生成は，最近の５分間のデータを使って計算し，１分間隔で計算する．データは，実験１と同じデータを利用する．

実験３:検証-3 の実験

- DSPP-WITH-SKETCH：実験２と同じ
- DSPP-WITH-SKETCH-NO-DB：図 11 のシステム．データベースを利用しないこと以外は，DSPP-WITH-SKETCH と同じ．

5.3. 実験結果

【実験1の結果】

プローブカーのデータ数に対する処理時間を図 12 に示す．従来方式に対して、本方式は平均 6 倍高速である．5 分(300 秒)間隔に解析処理を実行するため処理は 5 分以内に完了する必要があり，図 12 より従来システムは 8000 プローブのデータ（1600 台程度）で処理が溢れている．一方，本方式は最大混雑時期においても、100 秒の処理時間，すなわち要件の 1/3 程度の時間で処理可能であった．

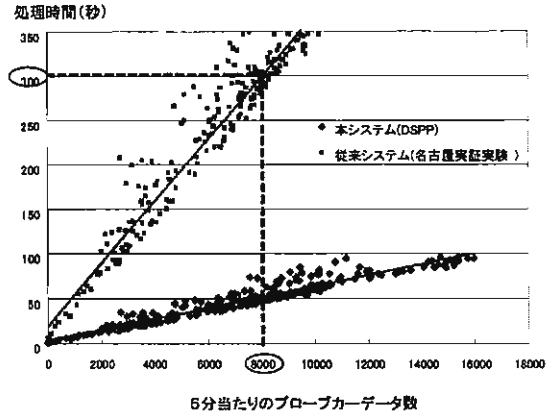

図 12　実験１の結果

【実験2の結果】

プローブカーのデータ数に対する処理時間を以下に示す．スケッチによる計算結果の使い回しにより平均 4.6 倍高速になった．今回は，５分間のデータを１分間隔に計算するために，各データは５回繰り返し計算される．従って，最高５倍高速化が可能であるが，スケッチの管理といった別の処理が発生するために 4.6 倍程度の結果となったと思われる．

また，実験１の本システムと比べると，実験２の本システムは５倍の頻度で計算しているにも関わらず，1.1 倍のレスポンスタイムとなった．これはスケッチにより，ほとんど実験１の場合と同等の計算量に抑えられたからである．

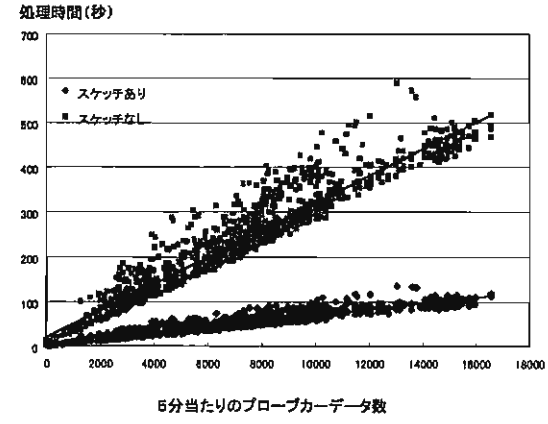

図 13　実験２の結果

【実験3の結果】

プローブカーのデータ数に対する処理時間を以下に示す．データベースを利用しない場合，平均 5.8 倍高速になった．なお，データベースを利用しない場合

は，プローブカーデータの数を 5 分当たり 2000 から 10000 まで 2000 間隔で人工的に発生させたデータを利用．従って，今回の比較実験は，プローブデータは同じではないが結果には影響は少ないと考えられる．

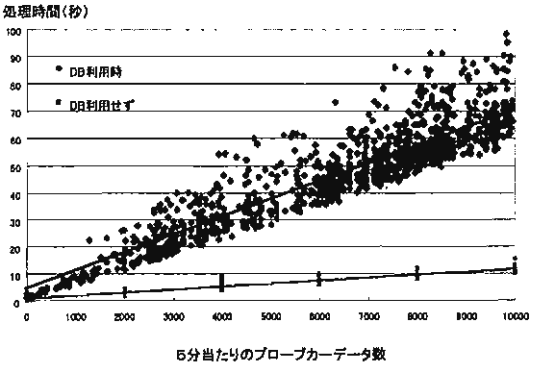

図 14　実験 3 の結果

6. 考察

6.1. さらなる高速化を目指して

本研究は，マップマッチングや経路探索などの各解析処理はブラックボックスとして扱い，これらの実行スケジューリングを制御してパイプライン化することにより高速化をはかった．つまり，各解析処理は従来と同じである．さらなる高速化のためには，これら各解析処理のアルゴリズムを工夫する方法がある．我々も，マップマッチングのアルゴリズムを提案している[5][6]．これらを組み合わせることでさらなる高速化がきたいできる．

また，本研究の高速化は，従来システムのボトルネックのひとつであるデータベースを使った方式に対する対策の提案であった．しかしながら，従来システムのボトルネックはデータベースだけではなく，プローブデータを収集する際のネットワークもボトルネックである．プローブデータの収集の高速化のための，プローブ情報の発生源でのフィルタリングや，データのパッキングなどの研究が今後重要であると考えている．

6.2. 関連研究

本研究の成果は，プローブカーデータの分析をデータストリーム処理することによりこれまでにない大規模化に成功したことである．世界中でプローブカーを使った実証実験を行っているが，いずれもデータベースを利用した方式であり，我々のアプローチはオリジナリティが高い．プローブ情報を使ったリアルタイムサービスの有効性はこれら実証実験で実証されており，本研究が大規模化に成功したことはプローブ情報システムの実用化へむけた意義は大きい．

一方，データストリーム処理の研究は，米国を中心にいくつか研究プロジェクトがある[7][8]．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，連続問い合わせ言語および問い合わせ処理の最適化，ストリームマイニングと多岐にわたるが，いずれも基礎研究段階であり実システムへの応用例がない．本研究は，実際のプローブタクシーと連携したシステムであることが特長である．

7. まとめ

我々が研究しているデータストリーム処理基盤 DSPP を，名古屋地区のタクシープローブデータを活用した高精度なプローブ情報システムへ適用した例と，その性能評価実験の結果を紹介した．技術的特徴は，以下の 2 点である．

- パイプライン処理による処理時間の削減
- スケッチによる繰り返し計算の削減

上記の技術を実際のタクシープローブ 1 日分のデータを使って評価実験を実施した結果，以下のとおり高速化に成功した．

- パイプライン処理により，従来の 6.0 倍
- スケッチにより 4.6 倍

さらに，データベースを使わない理想的なデータストリームアーキテクチャでの処理性能を評価した結果，従来の 5.8 倍の高速化に成功した．これにより，トータルで最大 160 倍の高速化(6.0 x 4.6 x 5.8) が期待できる．

文　献

[1] 姚恩建,佐藤彰典:プローブ情報活用システム「PROROUTE」の開発,NEC 技法 ITS 特集,Vol.61 No.1,pp.35-39 (2008)

[2] 熊山治良，横田孝義: 交通情報提供サービス，日立機関誌 Uvalere（ユーヴァレール）,Vol.8(2007)

[3] 中島かおり，尾林俊文，三浦寿：プローブへの取組みと ITS センタ構想，雑誌 FUJITSU ITS 特集，VOL.59, NO.4，pp465-480(2008)

[4] B.Babcock, et al, "Models and Issues in Data Stream Systems", ACM PODS'02,pp1-16(2002)

[5] 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(DICOMO2007)シンポジウム論文集，Vol.2007，pp.104-109 (2007)

[6] 今井照之，藤山健一郎，喜田弘司，中村暢達：大規模プローブカー情報を処理するための高速リンクマッチング手法，情報処理学会第 70 回全国大会 第 3 分，pp37-38(2008)

[7] Stanford Univ. : STREAM Project Homepage, http://www-db.stanford.edu/stream/

[8] MIT, Brown Univ. : Aurora Project Homepage, http://nms.lcs.mit.edu/projects/borealis/