Vol.2010-UBI-26 No. 5
2010/5/28

ストリーム処理エンジンにおける集約演算の実時間処理に関する一考察

川 島 英 之[†1,†2] 猿 渡 俊 介[†3]

スマートグリッドに代表される実世界状況監視技術においては，過去 x 分間の電力平均値を y 分周期で収集できるような技術が求められる．そのような技術の一つに問合せ処理の実時間性を考慮したストリーム処理エンジン (RT-SPE) がある．RT-SPE の代表的な研究には，Virginia 大学の Stankovic 教授らにより開発された RTSTREAM がある．RTSTREAM は，EDF 問合せスケジューリングとアドミッション制御により，RT-SPE における実時間性を実現している．RTSTREAM を詳細に観察すると，それが問合せを処理の最小単位とみなしており，問合せを構成する演算子の処理について言及をしていない点が認識される．そこで本研究は集約演算に着目し，その演算子自体を効率化することで，RTSTREAM の性能を改善することを試みる．提案は，集約演算処理を効率化するために，処理を子演算子に受け渡すことである．提案により，時間計算量を $O(n)$ から $O(1)$ に削減する．さらに，空間計算量を $O(mn)$ から $O(m)$ に削減する．提案手法が効率的であることはシミュレーションによっても示される．

A Consideration with Real Time Processing of Aggregate Operations on Stream Processing Engines

HIDEYUKI KAWASHIMA[†1,†2] and SHUNSUKE SARUWATARI[†3]

THIS paper proposes an efficient algorithm to compute aggregates over data streams. The essential concept of our proposal is to embed aggregate computation into its child operator so that the aggregate computation is processed together with child operator. Our proposal reduces time complexity from $O(n)$ to $O(1)$, and it also reduces spatial complexity from $O(mn)$ to $O(m)$. The efficiency of our proposal is also shown by simulation.

1. はじめに

人間-ロボット-コンピュータの相互作用解明[5]，ユビキタスコンピューティングの実現[22]，そしてサイバー物理システムの設計[15] に関する研究が行われている．これらの研究では，実世界で生じるイベントを検知すべく，屋内外のセンサデータを収集・解析する仕組みが必要になる．これを本論文では RED(Real-time Event Detection) と表記する．

RED を支援するため，センサデータ処理基盤に関する研究が行われている．その例には，信号処理に特化した WaveScope[14]，センサデータからのノイズ除去や補完処理を行う MauveDB[12]，複合イベント処理（CEP）を対象とする SASE[13] や Cayuga[8]，関係ストリーム処理を支援する STREAM/CQL[6] や Borealis[1]，そして筆者が開発した KRAFT[24] や，筑波大学で開発されてきた異種情報統合基盤 StreamSpinner[20] が挙げられる．また，実時間性を考慮した基盤システムとしては RTSTREAM[21] が挙げられる．

RED はその目的である実世界で生じるイベントの検知を行うため，周期的に集約演算を用いることがある．集約演算とは，最大値，最小値，平均値などの集約値を計算する演算を表す．上記のセンサデータ処理基盤はいずれも集約演算を支援する．周期的な集約演算は，STREAM や Borealis などの関係ストリーム処理基盤でも定義されている．しかしこれらは周期的な処理を実現するために必要な，デッドラインに基づく問合せ処理を実現していない．デッドラインに基づくストリーム処理を実現している研究は，我々の知る限り，RTSTREAM に限定される．

RTSTREAM の方式は，各問合せのデッドラインに基づく EDF スケジューリングと，アドミッション制御である．これにより，図 1 のような CQL ライクな問合せに対して RTSTREAM はある程度の実時間性を保証する[21]．

RTSTREAM はある程度の実時間性を保証するが，その性能改善には余地がある．それは，RTSTREAM が問合せ処理自体の高速化を検討していない点である．

そこで本研究では RTSTREAM[21] の集約演算に関する効率化を試みる．具体的には集約演算を効率的に実行する技法を提案する．提案技法により，時間計算量と空間計算量を削減す

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba
†3 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology

Vol.2010-UBI-26 No. 5
2010/5/28

```
SELECT AVG(S0.value), MIN(S0.value), MAX(S0.value)
FROM S0 [RANGE 2 SECOND], R0
WHERE S0.ID = R0.ID
RTSPEC STIME 0 second ETIME 301 second PERIOD 2 second DEADLINE 1 second
IMPORTANCE 0;
```

図 1 RTSTREAM の問合せ

る．n 個のタプルがあるとき，時間計算量を $O(n)$ から $O(1)$ に削減する．また，m 個の集約演算がある時，空間計算量を $O(mn)$ から $O(m)$ に削減する．

本論文の構成は次の通りである．2 節では関連研究について述べる．3 節では研究課題を定式化する．4 節では提案技法を述べる．5 節では提案技法を評価する．6 節では開発中の RT-SPE の設計を述べる．7 節では RT-SPE において実時間スケジューリングを行う方法を検討する．最後に 8 節ではまとめと今後の課題を述べる．

2. 関 連 研 究

2.1 ストリームデータ処理

近年，大量に流れ来るデータを効率的に処理する技術としてデータストリーム技術が注目されている．データストリームに関する主たる研究は 2002 年頃[7] から始まった．当初は関係データ演算をストリームモデルに適応させ，そのモデルに基づく処理系である **Stream Processing Engine**(以後，**SPE** と略記) において性能を追求する研究が行われたが，現在では多様なアプリケーションに特化した専用問合せ言語を持つ専用システムに関する研究が多数行われている．

データストリーム研究はおおまかに 3 世代に分けることができる．第一世代 SPE に関する研究では，関係データモデルを拡張して，ストリームを扱うためのデータモデルが提案された．そして同モデル上における性能向上とメモリ使用量の制御が研究された．このようなモデル化が行われた理由は，汎用的である関係データモデルを拡張することで，データストリーム処理の汎用的なモデルを構築しようとしたからである．この例には STREAM[6], Aurora[2], Telegraph[9] が挙げられる．

第二世代 SPE に関する研究では，第一世代 SPE で提案されたデータモデルに対して，分散処理を導入することで，高性能化および耐故障化に関する研究が推進された．この研究動向は現在もまだ継続されている．この例には Borealis[1], StreamSpinner[20] が挙げられる．

そして第三世代 SPE に関する研究では，第一世代，第二世代とは一線を画し，明確なアプリケーションを想定した，専門用途システムの構築が研究されている．用途例には，RFID, ネットワークトラフィック，そして動物の監視など，幅広いアプリケーションが含まれる．この例には 信号処理用途の WaveScope[14], パケット処理用途の GigaScope[10] およびサービス指向ルータ[17]，そして実時間処理用の RTSTREAM[21] が挙げられる．

2.2 実時間データ処理

ストリーム処理の概念が現れる前に行われてきた実時間データ処理には，インクリメンタル問合せ処理を行う APPROXIMATE[19] や，実時間性に加えて時間的一貫性を同時に考慮する研究[11] が挙げられる．他にも実時間トランザクションスケジューリング[3] や実時間並行実行制御[18] が研究されてきた．

しかしながら，我々が知る限り，実時間性を考慮したストリーム処理は RTSTREAM[21] に限られる．そこで本論文では RTSTREAM をベースにし，その性能を向上させる方式を検討する．

3. 研 究 課 題

本研究で取り組む課題は，集約演算を含む問合せ処理に関して，デッドラインミス率を削減することである．本論文で扱う問合せが有する演算は，集約演算と窓演算に限定する．

この研究課題を本論文では次のように定式化する．s 本の能動的情報源があるとする．各能動的情報源のデータ発生周期を p とする．k 本の問合せ $q_1, \ldots, q_i, \ldots, q_k$ が登録されるとする．全ての問合せは連続的問合せだとする．各問合せ q_i は m 個の集約演算 $a_1^i, \ldots, a_j^i, \ldots, a_m^i$ を有するとする．各問合せの実行時間を $e_1, \ldots, e_i, \ldots, e_k$ とする．各集約演算の窓幅長を n タプルとする．本論文では時間ベース窓は扱わない．各集約演算の窓移動長を $\frac{n}{2}$ タプルとする．即ち，窓内のタプルは毎回吐き出されるとする．各問合せのデッドラインは d だとする．

以上の条件において，本研究の課題は e_i を最小化することである．

本論文で扱う集約値計算処理の全体像を図 2 に示す．本研究の課題は 5 行目に示されている集約計算の効率化である．RTSTREAM では特段の工夫点がないため，このナイーブ方式を実装していると仮定する．

5 行目に示されているナイーブな実現方式は，n 個のタプルを線形走査するアルゴリズム

1: **For all** q_i such that $1 \leq i \leq n$
2: **While** (Current time is earlier than start time)
3: Release resource for other queries;
4: **End while**
5: **For all** a_j^i such that $1 \leq j \leq m^i$
6: Compute an aggregate value for a_j^i;
7: **End for**
8: **End for**

図 2 Complete Picture

になる．時間計算量は $O(n)$ となる．集約演算が m 個あれば，空間計算量は $O(mn)$ となる．

この方式を用いる場合，本論文の想定環境では，時間計算量と空間計算量が大きくなる．

例えば a_j^i の窓幅長と移動長が 1 秒間であり，これに対して，1 MHz でデータストリームがシステムに流れ込んでくるとする．このとき，a_j^i は毎秒 100 万件のデータを処理することになる．商用ストリーム処理エンジンである Coral8 のピーク性能が 100 万タプル/秒程度であることを考えると，この時間計算量は削減されることが望ましいと考えられる．また，システムは 100 万 $\times m \times$ タプルサイズ分のメモリを常に確保する必要がある．m が大きくなりディスクアクセスが発生すると SPE の性能は急激に劣化する．これに伴い，$safe$ は満足されなくなる．従って，本論文の目的を達成するには，この空間計算量は削減されることが望ましいと考えられる．

4. 提　　案

前節では，本研究の課題が e_i の極小化である事を述べた．また，ナイーブなアルゴリズムを用いると，時間計算量と空間計算量がそれぞれが $O(n)$ と $O(n*m)$ になることを述べた．

本節では時間計算量と空間計算量をそれぞれ $O(1)$ と $O(m)$ にするアルゴリズム Embed Pushdown Aggregate (EPA)[⋆1]を提案する．提案アルゴリズムの基本的な概念は，子演算子に集約演算をさせる点にある．集約演算子は演算木において葉にはならないため，子演算子は必ず存在する．また，子となり得る演算子は，選択，射影，和，結合，直積，窓，等の集約演算子以外の演算子である．これらの演算子はいずれも入力 n タプルを全て走査する必要があるため，集約演算の支援は可能である．図 5 に EPA アルゴリズムの例を示す．同図では提案アルゴリズムを図 4 に示す．同図を用いて提案アルゴリズムの時間計算量と空間計

⋆1 文献[16] では文献[23] が aggregate に関する push-down 方式を用いている旨が述べられているが，同方式は Group-by と Join の実行順序制御に関する研究であり，本提案とは異なる．

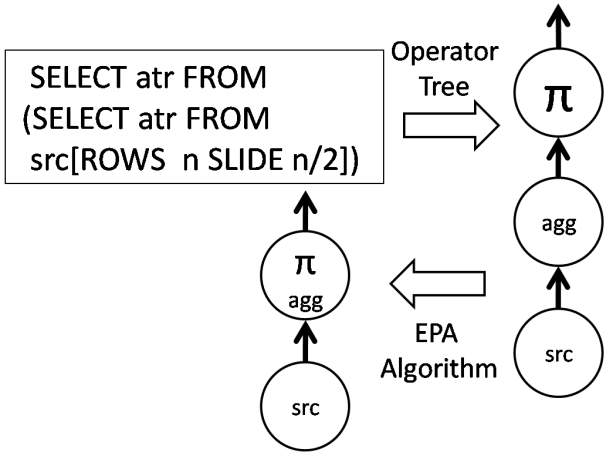

図 3 Example of EPA Algorithm

1: Pushdown an aggregate operator and embed it to its child;
2: **Forall tuples** t_i **such that** $0 \leq i \leq n$
3: Child operator scans t_i;
4: **If** it is true **then**
5: Compute aggregate using t_i and set it to agg;
6: **Endif**
7: **End for**
8: Output agg to the input queue of the aggregate operator;

図 4 Embed Pushdown Aggregate

算量を議論する．まず，時間計算量を議論する．8 行目より，集約演算子の入力キューへ挿入されるタプル数は 1 であることがわかる．従って，集約演算子の処理コストは $O(1)$ となる．子演算子の時間計算量を含めると，ナイーブ方式の時間計算量は $O(2n)$ となる．一方，提案方式の時間計算量は $O((1+\alpha)n)$ となる．ここで α は図 4 の 5 行目で必要になるコストである．4 行目の真偽判定は子演算子に課せられた役割であるから，提案方式によるコスト増加とは関係しない．但し条件分岐はコストを増加させる．次に，空間計算量を議論する．4 行目より，集約演算の中間値は agg に格納されることが明らかである．従って 1 つの集約演算に求められる空間コストは 1 タプルとなる．従って m 個の集約演算に求められる空間コストは $O(m)$ となる．更に，m 個の集約演算の中で，l 個が同一演算である場合，空間コストは $O(m-l+1)$ となる．

5. 評　　価

図 に示した問合せについて，ナイーブな方式と提案方式の性能をシミュレーションにより比較した．シミュレータは C++言語により開発した．実験に用いたマシンは Intel(R) Xeon(TM) MP CPU 3.00GHz, 6GB RAM, Red Hat Enterprise Linux AS release 4 である．実験結果を図 56 に示す．両実験の違いは，射影演算により削減するバイト数である．削減幅が小さい程，負荷は高くなる．なぜなら削減幅が小さい程，射影演算後のタプルサイズが大きいからである．タプルサイズが大きいほど メモリ確保ならびにメモリコピーに要する時間は長くなる．図 5 に示した実験では，射影演算においてタプルサイズを 40008 バイトから 10000 バイトまで削減している．40008 バイト程度のサイズを有するタプルは近年珍しくない．例えば我々が扱っている JPEG ストリームでは JPEG 用 CLOB に 40000 バイトを割り当てている．図 5 はナイーブ方式のグラフがタプル数 6000 で下がる現象が生じた．この現象は再現性を持ったが，原因は不明である．また，図 6 に示した実験では，射影演算においてタプルサイズを 40008 バイトから 40004 バイトまで削減している．いずれの実験においても，処理時間に関して，提案手法がナイーブ手法よりも優れた性能を示すことがわかる．

この実験では示されていないが，提案手法のメモリ使用量は高々4 バイトである．一方，ナイーブ手法が必要なメモリ使用量は本実験では $\frac{n}{2}\times$ タプルサイズである．従って提案手法はメモリ使用量においてもナイーブ手法よりも優れる．

6. SPE 処理系の設計

本節では我々が開発している RT-SPE の設計に関して述べる．なお，同 RT-SPE が提供する演算子は，選択，射影，結合，直積，集約，に加えて時系列パタンがある．これは，時系列タプル系列に対してクリネ閉包と順接を有する．

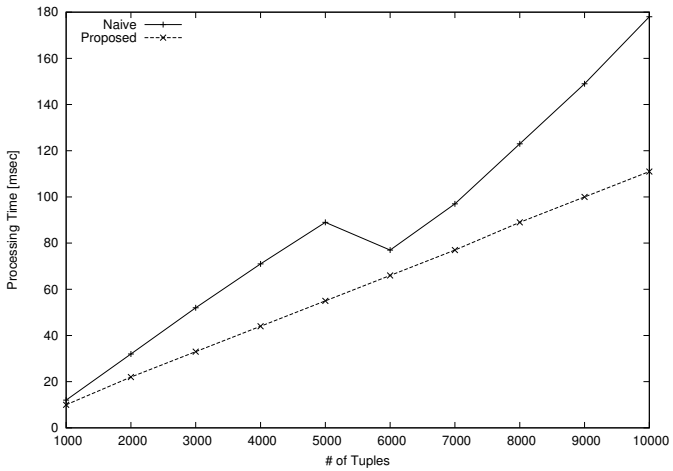

図 5　実験結果 (40008 Bytes → 10000 Bytes)

6.1 設　　計

6.1.1 演算木の設計

開発中の SPE は代数式を入力し，その代数式から演算木 (Operator Tree) を作成する．

演算木の入力はタプルストリームであり，これはウィンドウ演算子によりリレーションに変換される．ウィンドウ演算子は必ず演算木の葉に置かれるため，実行木内にはストリームは存在せず，データは全てリレーションとして扱われる．オペレータの実行順序は，下から上である．同じ高さに存在する演算子については，それらの間に実行順序の優先度は存在しない．即ちどの順序で実行されても構わない．この規則に従ってオペレータ実行順序を定める．実行順序は実行キューにより管理され，連続的問合せが終了するまで順序が変わることはない．演算木の出力はタプルストリームになる．本研究では CQL における ISTREAM[6] のみを対象とする．すなわち，本研究における演算木は新しい入力に関する結果のみを出力する．

ストリームの定義は，名前とデータ型の組を記述することで行われる．例を示す．`test1 (id int) (v double)` と `test2 (id int) (v double)` は二つのストリー

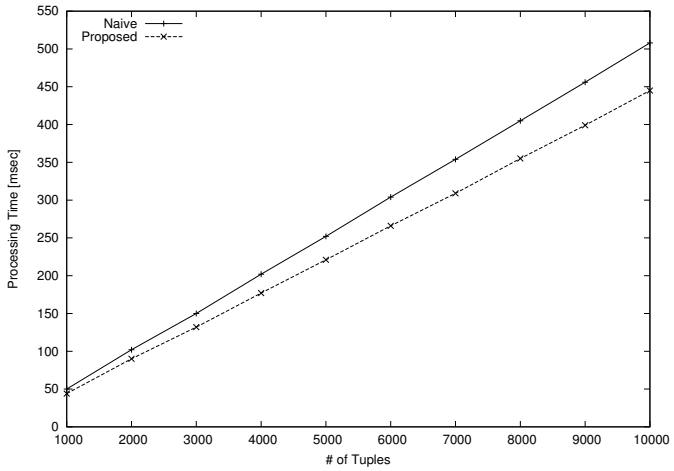

図 6 実験結果 (40008 Bytes → 40004 Bytes)

ム，test1 と test2 を定義している．いずれも int 型属性 id と double 型属性 v を有する．本 SPE が実現した型は，int 型，double 型，固定長テキスト型，そしてパケットペイロード用あるいは JPEG ストリーム用の CLOB である．

実装は C++言語により行っている．排他制御のために pthread ライブラリの mutex を使用している．総行数は 3000 行程度である．タプル形式はスキーマにより決定されるため，データ領域は void 型とし，全属性のサイズの総和長の連続領域により実現した．同連続領域は属性サイズにより区切られる．

7. 実時間スケジューリングの実現に関する検討

7.1 設　　計

今後，実システムを用いた実験で必要な実時間スケジューリングを行うために，本研究では SPE レベルと OS レベルの両面からアプローチする．具体的には Scheduler Activation に基づく機構を利用する．Scheduler Activation[4] とは，pthread レベルのスレッド実行順序がカーネルと同一になる仕組みの事をいう．

7.1.1 SPE レベルのアプローチ

周期的発信を効率的に実行するために，エグゼキュータを呼び出す処理と，エグゼキュータが得た結果を転送する処理を異なるスレッドで実行する．以後，前者をモニタスレッド，後者を転送スレッドと表記する．

```
1: スケジューラにリリースタイムを通知
2: do {
3:   スケジューラの呼出
4: } while(現時刻 < リリースタイム)
5: エグゼキュータを呼び出す
6: エグゼキュータの結果を結果リストに追加
```

図 7 モニタスレッドのアルゴリズム

モニタスレッドと転送スレッドはパイプライン処理することにより効率的な周期的発信を実現する．両者の機能が同一スレッドで実現された場合の問題は，転送に要する時間がわからないことである．エグゼキュータの結果を転送するのに要する時間は，クライアントへのネットワーク的距離とデータ量に依存するが，基本的には長くかかり，転送時間が監視をおこなう周期を越える可能性も考えられる．そこで両者の機能を分割して実現することにより，モニタリングスレッドの負荷を軽減し，周期的なエグゼキュータ実行を実現する．

モニタスレッドと転送スレッドの設計方式を以下で述べる．

7.1.1.1 モニタスレッド

周期的にエグゼキュータを呼び出すモニタスレッドのアルゴリズムを図 7 に示す．

1 行目でスケジューラに自分のリリースタイムを通知する．スケジューラはこれによりスレッドのリリースタイムを知り，この情報をもとにスケジューリングを行う．2 行目から 4 行目では，リリースタイムまでの待機を行う．スケジューラがモニタスレッドをリリースタイム前に実行スレッドとして選択した場合，そのモニタスレッドはスケジューラを呼び出し，スケジューラに再度，実行スレッドの選択を行わせる．リリースタイムが過ぎた後，5 行目においてモニタスレッドはエグゼキュータを呼び出し，その結果を 6 行目にあるように結果リストに追加する．当然ながらこのときリストを施錠/解錠する．

7.1.1.2 転送スレッド

モニタスレッドが得た結果をクライアントに転送する，転送スレッドのアルゴリズムを図 8 に示す．

2 行目から 4 行目において，結果リストに答えがあるかチェックし，もしあればそれをクライアントに転送する．そして結果があろうとなかろうと，5 行目において，次に動かすス

```
1: while (1) {
2:   if (結果リストに答えがある)
3:     答えをクライアントに転送
4:   スケジューラの呼出
```

図 8 転送スレッドのアルゴリズム

レッドを選択させるためにスケジューラを呼び出す．

7.1.2 OS レベルのアプローチ

OS レベルのアプローチとして本論文はユーザレベルスケジューラの修正を行なう．

スレッドスケジューラのアルゴリズムを図 9 に示す．1〜3 行目に示されている通り，スケジューラを呼び出したのが転送スレッドであれば，スケジューラはそのスレッドを実行キューの末尾に追加する．この処理に要する計算量は $O(1)$ である．なぜならばこの処理に必要な操作は，実行キューの末尾オブジェクトの次要素への当該スレッドの追加のみであるからである．

4〜6 行目に示されている通り，スケジューラを呼び出したのが SPE サーバスレッドであれば，スケジューラはそのスレッドを実行キューの先頭に挿入する．なぜなら SPE サーバスレッドが最優先されなければ，データベースシステムは新しいクライアントからのコネクション要求に応答できないからである．SPE サーバスレッドはクライアントからのコネクション要求がなければスケジューラを呼び出すことによりモニタスレッドや転送スレッドを実行させる．この処理に要する計算量は $O(1)$ である．なぜならばこの処理に必要な操作は，実行キューの先頭への当該スレッドの挿入のみであるからである．

7〜9 行目に示されている通り，スケジューラを呼び出したのがモニタスレッドであれば，スケジューラはそのスレッドを実行キューの先頭から，その中のモニタスレッドに対してリリースタイムを比較していき，実行キュー内でモニタスレッドがリリースタイムの早い順に並ぶような位置へ挿入する．すなわち最もリリースタイムが早いものが実行キューの先頭に並べられる．この方式を ERTF(Earliest Release Time First) と表記する．この処理に要する計算量は，モニタスレッドの数を n とすると $O(n+1)$ である．なぜならば，この処理に必要な操作は，最も計算量がかかる場合において，実行キュー内の全てのモニタスレッドと比較する必要があるからであり，実行キューの先頭は SPE サーバスレッドであり，その後には N 個のモニタスレッドが並んでいるからである．

```
1: if (スケジューラを呼び出したのは転送スレッド)
2:   実行キューの末尾に追加
3: else if (スケジューラを呼び出したのは SPE サーバスレッド)
4:   実行キューの先頭に挿入
5: else if (スケジューラを呼び出したのはモニタスレッド)
6:   そのスレッドのリリースタイムに合わせてキューに追加
```

図 9 スケジューラのアルゴリズム

7.2 実装方法

FreeBSD ソースコードにおいて”/usr/src/lib/libpthread”以下を修正する．FreeBSD5.3Release より，-pthread により選択されるスレッドライブラリが libc_r ではなく kse になったため作業手順は次の通りになる．

```
1: ソースコードを編集
2: make && sudo make install
3: gcc -pthread ... として My-RT-SPE を make
```

FreeBSD5 系からは，効率的なスケジューリングを実現するために Scheduler Activation(SA)[4] ベースの設計である，Kernel Scheduling Entity(KSE) が使われている．これは基本的に SA と同様であり，カーネルからメールボックスを通じてユーザレベルへの通知を行う．libc_r と異なり，マルチプロセッサアーキテクチャならばスレッドを異なるプロセッサに割り振ることができる．ユーザスレッドのスケジューリングはユーザレベルスケジューラ (UTS) により実現される．

7.3 スケジューラ

7.3.1 スケジューラの呼び出し方法

クライアントプログラムが UTS を呼び出すには，FreeBSD5.3Release の pthread 実装では pthread_yield をコールすればよい．pthread_yield コールから UTS までの関数遷移を図 10 に示す．

7.3.2 リリースタイムの設定

問合せ処理を行うスレッドが実行開始時刻を設定するために，pthread_set_release(long long release) というインタフェースを作成する．スレッドの構造は struct pthread により定義されており，struct pthread の中には struct pthread_attr 型である attr という名前の属性が存在す

```
1: pthread_yield (クライアントプログラムが実行)
2: _thr_sched_switch
3: _thr_sched_switch_unlocked
4: _thread_enter_uts (ユーザレベルスケジューラへの遷移)
5: _i386_enter_uts (CPU が Intel である場合)
6: kse_sched_multi(スケジューラ本体)
```

図 10 yield によるスケジューラ呼び出し

```
1: kse_sched_multi
2: kse_switchout_thread
3: if (スレッドは走行状態である (== PS_RUNNING))
4:   _pq_schedule_ertf (筆者により追加する関数)
```

図 11 実行キューの操作

る．この属性にリリースタイム用のデータ構造として long long 型で release という名前の属性を追加する．

pthread_set_release が呼び出されると，スケジューラを施錠し，ユーザが設定したリリースタイムを release を代入し，スケジューラを解錠する．そして UTS はこの値をみてそのスレッドをスケジューリングする．

7.3.3 実行キュー操作

前述の手続きを実行するにあたり，優先度キューを複数もつ必要はない．

`kse->sched_queue->sq_runq->pq_lists`

が実行キューであり，キューの本数は

`THR_MAX_PRIORITY-THR_MIN_PRIORITY+1`

で設定される．そこで，`THR_MIN_PRIORITY`

と `THR_MAX_PRIORITY`

を 0 に設定することで，実行キューを 1 つにする．

実行キューの操作はスケジューラである関数である kse_sched_multi から図 11 のように呼び出される．

_pq_schedule_ertf の実装を図 12 に示す．

```
1: _pq_schedule_ertf(pq_queue_t *pq, pthread_t pthread){
2:   int prio = THR_MIN_PRIORITY;
3:   pthread_t work;
4:   PQ_SET_ACTIVE(pq);
5:   work = TAILQ_FIRST(&(pq->pq_lists[prio].pl_head));
6:   while (1)
7:   if ((work == NULL) || (work->attr.release < 0) || (work->attr.release > pthread->attr.release))
8:       break;
9:    work = TAILQ_NEXT(work, pqe);
10:  if (work == NULL)
11:    TAILQ_INSERT_TAIL(&(pq->pq_lists[prio].pl_head), pthread, pqe);
12:  else
13:    TAILQ_INSERT_BEFORE(work, pthread, pqe);
14:  if (pq->pq_lists[prio].pl_queued == 0)
15:    pq_insert_prio_list(pq, prio);
16:  pq->pq_threads++;
17:  pthread->flags |= THR_FLAGS_IN_RUNQ;
18:  PQ_CLEAR_ACTIVE(pq);
19: }
```

図 12 ERTF の実装

8. まとめと今後の課題

本研究は集約演算に着目し，その演算子自体を効率化することで，RTSTREAM の性能を改善する方法を提案した．集約演算処理を効率化するために，処理の中間状態を保持する機構を子演算子に導入したことである．同機構により，時間計算量を $O(n)$ から $O(1)$ に削減し，空間計算量を $O(mn)$ から $O(m)$ に削減した．提案手法が効率的であることはシミュレーションによっても示した．

今後の課題は実システムと実データセットを用いた評価である．現在作成中の SPE に実時間性を与えるため，Scheduler Activation に基づく方式を検討している．

謝　　辞

本研究の一部は科学研究費補助金基盤研究 (#22700090) による．ここに記して謝意を表す．

参 考 文 献

1) D.Abadi, Y.Ahmad, M.Balazinska, U.Cetintemel, M.Cherniack, J.-H. Hwang, W.Lindner, A.Maskey, A.Rasin, E.Ryvkina, N.Tatbul, Y.Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *Proc. of the International Conference on Innovative Data Systems Research*, 2005.
2) Daniel J. Abadi, Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and StanleyB. Zdonik. Aurora: a new model and architecture for data stream management. *International Conference on Very Large Databases Journal*, Vol.12, No.2, 2007.
3) RobertK. Abbott and Hector Garcia-Molina. Scheduling real-time transactions: a performance evaluation. *ACM Trans. Database Syst.*, Vol.17, No.3, pp. 513–560, 1992.
4) ThomasE. Anderson, BrianN. Bershad, EdwardD. Lazowska, and HenryM. Levy. Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, pp. 95–109, 1991.
5) Yuichiro Anzai. Human-Robot-Computer Interaction: A New Paradigm of Research in Robotics. *Advanced Robotics*, Vol.8, No.4, pp. 357–369, August 1994.
6) A.Arasu, S.Babu, and J.Widom. Cql: A language for continuous queries over streams and relations. In *Proc. of International Workshop on Database Programming Languages*, pp. 1–19, 2003.
7) Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *Proc. of Symposium on Principles of Database Systems*, 2002.
8) Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. Cayuga: a high-performance event processing engine. In *Proc. of the 2007 ACM SIGMOD International Conference on Management of Data International Conference on Management of Data*, pp. 1100–1102, 2007.
9) Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellestein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of the International Conference on Innovative Data Systems Research*, 2003.
10) Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: a stream database for network applications. In *Proc. of the 2003 ACM SIGMOD International Conference on Management of Data international conference on Management of data*, pp. 647–651, 2003.
11) Anindya Datta and IgorR. Viguier. Providing Real-Time Responce, State Recency and Temporal Consistency in Databases for Rapidly Changing Environments. *Information Systems*, Vol.22, No.4, pp. 171–198, 1997.
12) Amol Deshpande and Samuel Madden. Mauvedb: supporting model-based user views in database systems. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 73–84, 2006.
13) Yanlei Diao, Neil Immerman, and Daniel Gyllstrom. Sase+: An agile language for kleene closure over event streams. In *UMass Technical Report 07-03*, 2007.
14) Lewis Girod, Yuan Mei, Ryan Newton, Stanislav Rost, Arvind Thiagarajan, Hari Balakrishnan, and Samuel Madden. The case for a signal-oriented data stream management system. In *Proc. of Conference on Innovative Data Systems Research*, pp. 397–406, 2007.
15) EdwardA. Lee. Cyber physical systems: Design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, 2008.
16) Samuel Madden, MichaelJ. Franklin, JosephM. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, Vol.36, No.SI, pp. 131–146, 2002.
17) Tomoaki Makino, Michihiro Koibuchi, Hideyuki Kawashima, Koichi Inoue, and Hiroaki Nishi. Hardware architecture for supporting high-speed database insertion on serviceoriented router for future internet. In *Proc. the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'10)*, 2010.
18) Özgür Ulusoy and GenevaG. Belford. Concurrency control in real-time database systems. In *CSC '92: Proceedings of the 1992 ACM annual conference on Communications*, pp. 181–188, 1992.
19) Susan Vrbsky. A data model for approximate query processing of real-time databases. *Data and Knowledge Engineering*, Vol.21, No.1, pp. 79–102, 1996.
20) Yousuke Watanabe and Hiroyuki Kitagawa. Query result caching for multiple event-driven continuous queries. *Information Systems*, Vol.35, No.1, pp. 94–110, 2010.
21) Yuan Wei, SangH. Son, and JohnA. Stankovic. Rtstream: Real-time query processing for data streams. In *ISORC '06: Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pp. 141–150, 2006.
22) Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, Vol.36, No.7, pp. 74–84, 1993.
23) WeipengP. Yan and P.Larson. Eager aggregation and lazy aggregation. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pp. 345–357, 1995.
24) 川島英之, 今井倫太, 遠山元道, 安西祐一郎. センサデータベースシステム kraft の設計と実装. 情報処理学会論文誌：データベース, Vol.45, No. SIG 14 (TOD 24), pp. 39–53, 2004.