2 センシングデータ処理基盤技術
—ストリームデータ処理—

北川 博之[1] 川島 英之[1] 天笠 俊之[1]
1 筑波大学大学院システム情報工学研究科

ストリームデータ処理

各種センサデータ，Twitter，Ustream，実時間株価情報配信，移動体位置データ，監視カメラ，ログやアクセス履歴情報，ソフトウェアから生成されるイベント列等，実世界の状況を継続的に配信する情報源が多数出現している．本稿ではこのような情報源をセンシング情報源と呼ぶ．センシング情報源はデータを絶え間なく連続的にストリームとして配信する．実世界の状況は時々刻々と変動するため，ストリームデータを利用したモニタリング応用等のデータ処理においては，ストリームデータを連続的かつ即時的に処理する必要がある．

センシング情報源の種類が限られ，また，そのデータ利用の目的や方法が限定されている場合には，専用の応用システムを構築することも考えられ，現にこれまではこのようなアプローチが主流であった．しかし，多様なセンシング情報源を対象とし，かつその利用目的も多様でオープンな場合は，異種のセンシングデータを統一的に扱い種々の応用利用を支援する枠組みが必要になる．

従来の大規模データ応用構築を支援するための処理機構としては，オンライントランザクション処理やバッチ処理を提供するデータベース管理システム（DBMS）がある．しかし，DBMSをストリーム処理に用いた場合には，いくつかの問題が発生する．

DBMSでは，通常，2次記憶に置かれたデータに対する問合せや更新処理が想定されている．このため，

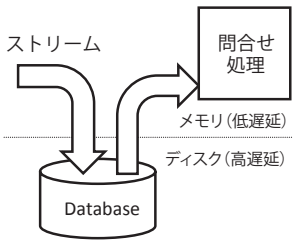

図-1(a) DBMS

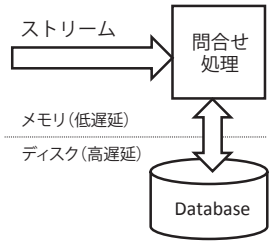

図-1(b) ストリーム処理

データベース管理システムを用いてストリームに対する問合せを実行する場合には，図-1 (a)[6]に示すように，まず入力ストリームを2次記憶に格納し，必要に応じて索引付けを行う必要がある．しかし，このような前処理を行った場合には，ストリーム処理において要求される即時性を満たすことが難しくなる．また，DBMSは応用プログラムからの問合せ要求に対応して問合せ結果を返すという受動的なコンポーネントであるため，応用プログラム側が同じ問合せを繰り返し発行する必要が生じる．さらに，問合せ記述によっては問合せ結果が重複することもあり，最新の状況変化を差分として応用プログラム内で陽に抽出する処理が必要になるようなことも考えられる．

ストリームデータ処理は，図-1 (b)[6]に示すように，ストリーム情報源から到着したデータを主記憶上で即時的に処理するものである．この際，データ分析や履歴管理等の目的のために，必要に応じてストリームデータを直接あるいはDBMSを通して2次記憶内に格納することもあるが，2次記憶への格納を前提としていない点が本質的に異なる．大量のデータを

主記憶上においたデータ処理技術として，主記憶データベース技術がある．主記憶上のデータ処理という観点からは，両者に共通点も見られるが，ストリームデータ処理が「流れてきたデータ」を即時に処理することを主目的にしているのに対し，主記憶データベースはあくまで「定常的に存在しているデータ」の処理を主目的にしている点で，両者の実現に必要な技術は異なってくる．

ストリームデータ処理のポイントを簡潔に述べると次の通りである．(1) システムに到着したデータをディスクに書き込むことなく，即時的に主記憶上で処理して結果を出力する．(2) 問合せは応用プログラムの処理要求に応じて事前にシステムに登録され，新規データの到着ごとに自動的かつ継続的に評価される．(3) 論理的には無限の長さを持ち得るストリームデータを適当な単位で分割し，毎回の問合せの評価対象とする．

上記 (2) を実現するため，連続的問合せ (Continuous Query) という概念が導入され，その問合せ評価方式が開発された．多くのストリーム処理システムでは，DBMSの標準言語であるSQLをベースとした問合せ記述を用いている．しかし，ストリームデータ処理においては，いわば，連続的問合せが「永続的」であり，新規データが到着するごとにすでに登録済みの連続的問合せが評価される．一般のDBMSでは問合せが発行されるタイミングで処理が行われるので，問合せとデータの関係が逆になっている．さらに，リレーショナルモデルに基づいたデータモデル化を行った場合，ストリームは無限長を持つリレーションと捉えることができる．しかし，一部のリレーショナル演算子 (集約・結合等) は無限長のデータを扱うことができないため，上記(3)を実現するため，ストリームデータから有限長のタプルを切り出す窓演算子 (Window Operator) が導入された．

ストリームデータ処理を実現するミドルウェアソフトウェアとして，ストリーム処理エンジン (SPE) が研究開発されてきた．SPEの概要を図-2に示す．SPEはまずユーザに登録された問合せを解析し処理木の形式で保存する．その際，問合せ最適化が行われて，

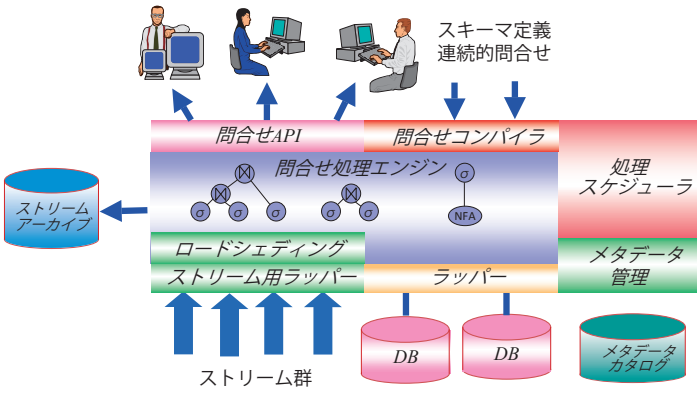

図-2　ストリーム処理エンジン

適切な演算子実行順序や複数問合せ間で共通な処理の抽出等も行われる．一方，ストリーム情報源からストリームデータがSPEに到着すると，それはラッパーにより内部形式に変換され，処理木中の演算子によって処理される．処理木全体の演算処理が終了した際には，その結果がユーザへ返却される．性能向上のために処理スケジューラが処理木中の演算子の実行優先順位を制御する．また，ストリーム情報源からの到着データ量がシステムの処理能力を超えるような過負荷時には，一部入力データを破棄するようなロードシェディング処理が行われることもある．

以下では，まず初期のリレーショナルモデルに基づく代表的ストリーム処理研究であるSTREAMに関して，ストリームデータ処理のモデル化や問合せ記述を中心に説明する．初期のSPEによって基本的なモデルとアーキテクチャが確立された後，分散化による高性能化・高信頼化に関する研究が展開された．これらを次に解説する．さらに，ストリームデータ処理の中には，リレーショナルモデルに基づくアプローチとは別に，アクティブデータベースにルーツを持つ複合イベント処理に基づく研究がある．その代表であるSASE+のアプローチを解説する．最後に，筆者らが過去数年間にわたり研究開発を行ってきたイベント駆動に基づくリレーショナルストリーム処理を特徴とする分散型SPEであるStreamSpinnerについて述べる．

リレーショナルSPE：STREAM[1)]

リレーショナルモデルへのストリームの導入

リレーショナルモデルを拡張してストリーム処理のためのモデルを構築する研究が初期のSPEでは行われた．その例としては，Stanford大学のSTREAM，Brown大学／Brandeis大学／MITのAurora，UC BerkeleyのTelegraphCQ等が挙げられる．本章ではその中で代表的なSPEであるSTREAMのアプローチを取り上げて解説する．

データモデル

STREAMは，ストリームデータ処理で扱われるデータとして，ストリームとリレーションの2種類のデータを定義する．ストリームは，ストリーム情報源から連続的に配信されるストリームデータを表すテーブルである．

ストリームの定義は次のように与えられる．

> 定義（ストリーム）あるストリームSは要素$<s, \tau>$の多重集合である．なお，多重集合とは，重複を許す集合のことである．ただしsはSのスキーマに属するタプルであり，τは要素の時刻印である．

ストリームデータのうち，情報源から得られるものを実ストリーム，問合せによって生成される結果を導出ストリームと呼ぶ．

一方，従来のリレーションの定義は次のように与えられる．

> 定義（リレーション）あるリレーションRは，各時刻におけるストリームから，有限個のタプルの多重集合への写像である．

時刻τにおけるタプルの多重集合を$R(\tau)$とする．ストリームと同様に実リレーション，導出リレーションが定義される．

ストリームはリレーショナルモデルにおけるリレーションと同様のテーブルであるが，論理的には無限長であるため，それを直接リレーショナルモデルにおけるリレーショナル演算子で処理することができない．そこで，STREAMでは，時刻τにおける連続的問合せの実行を次の3ステップからなるものとして，ストリーム処理をモデル化している．(1) 実ストリームから，時刻τにおける連続的問合せの対象となるタプルを切り出し，実リレーションに変換する．(2) 前ステップで得られた実リレーションを問合せに対応するリレーショナル演算子群で処理し，導出リレーションを得る．(3) 時刻τにおける導出リレーションから，（必要に応じて時刻$\tau-1$における導出リレーションを参照して）時刻τにおける導出ストリームを得る．このように，ストリームとリレーションをつなげる演算子を定義することで，既存のリレーショナル演算子を適用可能にする．

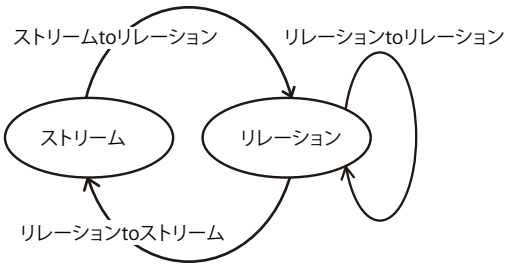

図-3　データと演算子の抽象的意味論

上記の3ステップのそれぞれにおいて用いられるのが，図-3に示す，ストリームtoリレーション演算子，リレーションtoリレーション演算子，そしてリレーションtoストリーム演算子である．

ストリームtoリレーション演算子は，時刻τにおける有限個のタプルからなるストリームのスナップショットからリレーションを得る演算子であり，窓演算子（Window Operator）と呼ばれる．STREAMのモデルでは3種類の窓演算子が提供される．タプル窓演算子は直近の一定数のタプルを切り出すことでストリームをリレーションに変換する．時間窓演算子は過去の一定時間内に到着したタプルを切り出すことでストリームをリレーションに変換する．分割窓演算子はキーに基づいてストリームをタプルグループに分割する．窓演算子については，このほかにも関連研究において多数のバリエーションが提案されている．詳細は文献6)の2章を参照されたい．

リレーションtoリレーション演算子は，リレーショナル演算子群であり，リレーショナルDBMSにお

けるSQL記述に対応するデータ操作を実現する演算子群である．リレーショナル演算子の詳細は標準的なデータベースの教科書を参照されたい．

次に，リレーションtoストリーム演算子について解説する．これらは，時刻τにおける導出リレーションから，（必要に応じて時刻$\tau-1$における導出リレーションを参照して）時刻τにおける導出ストリームを得る演算子である．STREAMでは，下記3種類のリレーションtoストリーム演算子を提案している．これらの中ではIstream演算子が最も頻繁に使われる．

1. **Istream演算子**．これは時刻$\tau-1$から時刻τの間にRへ追加されたタプルの多重集合を得る演算子である．Istream演算子は次のように定義される．

$$\mathrm{Istream}(R) = \cup_{\tau \geq 0} (R(\tau) - R(\tau-1)) \times \{\tau\}$$

2. **Dstream演算子**．これは時刻$\tau-1$から時刻τの間にRから削除されたタプルの多重集合を得る演算子である．Dstream演算子は次のように定義される．

$$\mathrm{Dstream}(R) = \cup_{\tau > 0} (R(\tau-1) - R(\tau)) \times \{\tau\}$$

3. **Rstream演算子**．これは時刻τにおいてRに含まれるタプルの多重集合をそのままストリームとする演算子である．Rstream演算子は次のように定義される．

$$\mathrm{Rstream}(R) = \cup_{\tau \geq 0} R(\tau) \times \{\tau\}$$

問合せ言語

これまで述べてきた演算子を操作するための問合せ言語として，STREAMはContinuous Query Language（CQL）というSQL風の宣言的問合せ言語を新たに提供する．下記にCQLを用いた例を示す．

例（1）：毎時65マイルより速い車からのデータが新しく届いたら教えよ．

```
SELECT    Istream(*)
FROM      PosSpeedStr[Range Unbounded]
WHERE     speed > 65
```

この問合せは次のように評価される．まず2行目のPosSpeedStr[Range Unbounded]はPosSpeedStrというスキーマに対して無限長の窓演算を適用して，リレーションを切り出す．次に3行目の選択演算子（述語“speed > 65”を有するフィルタ）が評価される．そして最後に1行目のIstream演算子により，前回実行時には含まれなかった新たなタプルからなるリレーションがストリームに変換される．

例（2）：高速道路から出て行った車のIDを教えよ．

```
SELECT    Dstream(vehicleid)
FROM      PosSpeedStr[Range 30 Seconds]
```

この問合せは次のように評価される．まず2行目で時間窓演算子が評価され，PosSpeedStrストリームから過去30秒間以内に到着したデータからなるリレーションが切り取られる．次に，1行目でそのリレーションに対して射影演算子が適用されてvehicleid属性が選択される．最後に1行目のDstream演算子により，前回実行時には存在していたが，今回はなくなったvehicleid，すなわち高速道路から出て行った車のIDが出力される．

連続問合せ処理

窓演算子を実現するには，窓演算子により指定された数だけ，過去に到着したタプルを保持しておく必要がある．この過去のタプルを保持する機構はシノプシスと呼ばれる．シノプシスはDBMSでの処理では不要であり，ストリーム処理で新たに導入された．Istream演算子を用いた処理においては，直積演算子，結合演算子，そして集約演算子にのみシノプシスが必要となる．シノプシスを用いたこれらの演算はそれぞれ窓直積（Window Cartesian Product），窓結合（Window Join），そして窓集約（Window Aggregate）と呼ばれる．これらの演算はストリーム処理でのみ現れる．窓演算子により拡張された演算（窓結合等）を効率的に実現する方法に関しては，多数の研究が行われてきた．

また，演算子のスケジューリング等による応答時間改善・使用メモリ量削減や，過負荷に対応するためのロードシェディング技術も開発された．STREAMに関する研究プロジェクトは2006年1月に完了したが，同プロジェクトはストリームデータ処理の礎を築いた．STREAMに関する論文ならびにソフトウェアは下記URLから利用可能である：http://infolab.stanford.edu/stream/

分散型SPE：Borealis

Borealisの概要

STREAM[1]を代表とする初期のSPEでは，データモデル，問合せインタフェース，演算子スケジューリング，ロードシェディング等が研究されてきた．その後の発展研究において，複数のノードを利用して高性能化や高信頼化を達成可能な分散化技術の研究が行われた．本章では分散化を実現した代表的なSPEであるBorealisを解説する．BorealisはBrown大学／Brandeis大学／MITの研究グループにより開発された初期SPEであるAuroraを核としている．Borealisのアーキテクチャは分散化を指向して設計されており，グローバルカタログ，高可用化器，近傍最適化器等をはじめとする多数の分散化用コンポーネントが基本部分に組み込まれている．本章ではBorealisに関して，分散問合せ最適化による高性能化と，3つの高信頼化技法を解説する．Borealisの情報は下記から取得可能である：http://www.cs.brown.edu/research/borealis/public/

分散問合せ最適化[2]

Borealisを利用して分散環境で問合せ最適化を行う研究として，Pietzuchらにより提案された手法を解説する．広域に分散して配置された多数のノードが連携して問合せ処理を最適化する場合を考える．任意のノードに任意の演算子を割り当てることが可能であるという前提があるとき，演算子をどのような基準でノードに割り当てるかが研究課題となる．

Pietzuchらは，オペレータ配置問題における最適化は，1問合せあたりのネットワーク帯域使用量$u(q) = \sum_{l \in L} DR(l)Lat(l)$を最小化することと設定した．ただしLはノード間のリンクの集合，DR(l)はリンクlにおけるデータレート，そしてLat(l)はリンクlにおける遅延を表す．

上記の最適化を行うために，PietzuchらはSBON（Stream-Based Overlay Network）アーキテクチャを提案した．SBONでは演算子を各ノードに配置する前に，遅延空間と呼ばれる多次元空間上に通信遅延や負荷に基づいてノードをマップし，通信遅延に関する見積りを行う．その後，遅延空間において，生産者と消費者がオペレータをばねで引き合っていると考えるばねモデルを用い，同モデル上でオペレータの持つ位置エネルギーが最小となるノードにオペレータを配置することで最適化を実現する．SBONアルゴリズムはBorealisを利用して実装され，PlanetLabを用いた実験とシミュレーションにより，その有効性が示された[2]．

高信頼化機構[3]

Borealisを提案したMITの研究グループは，分散環境におけるストリームデータ処理に対する高信頼化手法を提案した．高信頼化の意味は，あるノードが停止故障を起こした後に，(1) そのノードが出力すべきだった結果を他ノードが出力することと，(2) そのノードで稼働していた演算子処理を引き継ぐことである．ここでは文献3)で示された3つの高信頼化手法を解説する．なお，主として問合せ処理を行うノードをプライマリ，プライマリの故障時に問合せ処理を引き継ぐノードをセカンダリ，プライマリにデータを送信するノードを上流ノード，プライマリから結果を受信するノードを下流ノードと表記する．

Passive Standby方式（図-4）は，プライマリが問合せ処理と並行して，定期的にプライマリの内部状態をセカンダリへ複製する方式である．プライマリの障害発生を検知すると，セカンダリは最新のコピーを用いて処理を再開する．このとき，コピー後にプライマリが処理したデータはセカンダリのコピーに反映されていないため，そのまま再開しては処理データが失われる．この損失を防ぐため，セカンダリは上流ノードに対してそれらのデータを処理再開時に再送要求する．

Active Standby方式（図-5）は，上流ノードからプライマリが受信するデータをすべてセカンダリにも受け取らせ，セカンダリはプライマリと並列に同内容の問合せ処理を行う．プライマリが生存している場合には，セカンダリは処理結果を出力せずバックアップデータとして蓄積する．

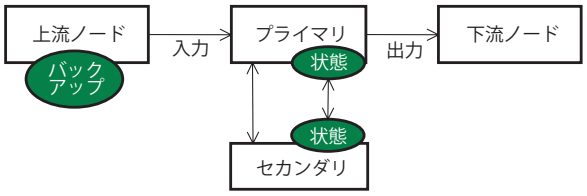

図-4 Passive Standby 方式

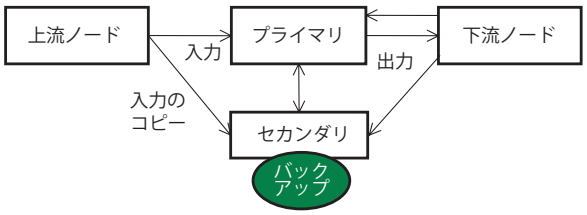

図-5 Active Standby 方式

Upstream Backup 方式(図-6)は，プライマリが生存している場合には，セカンダリは待機しながらプライマリの生存確認のみ行う．セカンダリは，Passive Standby 方式のような内部状態のコピーや Active Standby 方式のようなプライマリと並列した問合せ処理は行わない．その代り，上流ノードは，障害回復からの再開時に必要となるプライマリへの入力データをバックアップデータとして蓄積しておく．プライマリに障害が発生した場合は，上流ノードはバックアップデータをセカンダリに再送し，セカンダリがそのデータを処理することで復旧する．

複合イベント処理(CEP)：SASE+[4)]

STREAM や Borealis のようなリレーショナルデータベースに基づくストリームデータ処理が存在する一方，アクティブデータベースにルーツを持つ，複合イベント処理 (Complex Event Processing (CEP)) が提案されている．アクティブデータベースでは入力イベント列を合わせて複合イベントを出力する．CEP はそれに時系列という要素を導入している．CEP はある条件を満足するイベントの繰り返しや，複数イベントの時間的順序関係を表現できる．CEP には SASE+ や Cayuga をはじめとして多数の研究があるが，本稿では紙面の都合から代表的研究である SASE+[4)] について述べる．

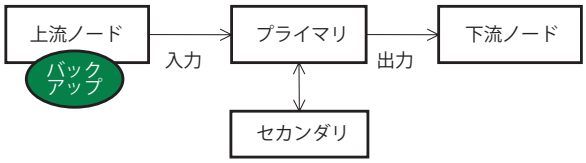

図-6 Upstream Backup 方式

SASE+ には2つの特徴的な演算子が導入されている．それらは Sequence 演算子と Kleene plus 演算子である．Sequence 演算子はタプル A の到着の後にタプル B が到着したことを検出する演算子である．Kleene plus 演算子はある条件が成り立つタプルが繰り返し連続して到着したことを検出する演算子である．これらの演算子を用いた SASE+ の問合せ例を図-7 (a) に示す．

この問合せは複雑な株式市場の傾向を把握するために記述されている．把握したいシーケンスは，取引量が多い状態で始まり，価格が徐々に上がるが，取引量が急落するようなものである．ただし，シーケンスの長さは1時間以内とされている (WITHIN 句)．この問合せで求めるシーケンスには2つの要素があり，SEQ (Stock+ a[], Stock b) と記載されている．価格が徐々に上がり続けるシーケンスを把握するための，タプルの繰り返し (Stock+ a[]，Kleene plus 演算子で取得) と，それに続いて取引量急落を把握するタプル (Stock b，Sequence 演算子で取得) である．“a[]” と “b” に格納されるべきタプルのパタンは WHERE 句に記述される．ここで “a[1]” は最初のタプル，“a[i]” は過去に選択したタプルの平均値よりも高い価格を有するタプル，そして “a[a.LEN]” は最後のタプルを表す．また “skip_till_next_match” は条件に適合しないタプルをスキップする意味を表す．

問合せは NFA にバッファを追加した NFA^b という抽象機械へ変換される．図-7 (b) は，図-7 (a) の問合せが NFA^b へ変換された様子を示す．θ_{begin} および θ_{take} は入力タプルを消費すると同時にそれをバッファに格納し，次の状態へ遷移する．θ_{ignore} は入力タプルを消費するが，それをバッファには格納しない．これは前述の “skip_till_next_match” が指定されたとき，条件に合わないタプルをスキップするために使

```
PATTERN SEQ(Stock+ a[], Stock b])
WHERE   skip_till_next_match(a[], b) {
            a[1].volume > 1000
            and a[i].price > avg(a[..i-1].price)
            and b.volume < 80% * a[a.LEN].volume }
WITHIN  1 hours
```

図 -7 (a)　問合せ

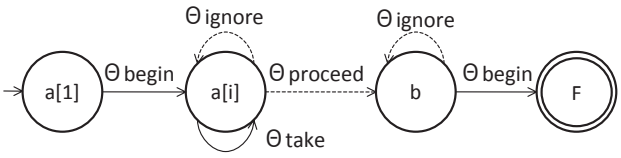

図 -7 (b)　問合せの NFA^b 表現

われる．θ_{proceed} は入力タプルを消費せず，評価のみ行い，状態遷移を試みる．文献4）では NFA^b を効率的に処理する手法が提案されている．SASE+ の情報は次から取得できる：http://sase.cs.umass.edu/

StreamSpinner[5)]

これまで海外の代表的な研究を解説してきたが，国内でも SPE は研究開発がされている．ここでは筆者らが研究開発を行っている SPE である StreamSpinner について解説する．StreamSpinner のデータモデルは STREAM のモデルに基づくが，次の点で異なる．

（1）問合せの実行タイミングの明示的な指定ができる点

StreamSpinner の問合せ言語は，新規データが到着するたびに起動する問合せや，タイマーにより定期的な問合せ実行，その両方で起動する問合せなどをすべて統一的な記述方式（MASTER 節，後述）で与える．

（2）動的な情報源選択機能

センシング応用等では，問合せ記述時に対象となる情報源を特定できず，情報源からの入力に基づいて参照すべき情報源を動的に切り替えるような処理が必要となることがある．そのための問合せ記述方式を提供する．

StreamSpinner への問合せには，CQL が提供する SELECT，FROM，WHERE，GROUP BY 節に加えて，新しく MASTER 節が導入されている．これはイベント駆動型問合せ処理を実現する節であり，問合せ評価の実行契機となる情報源を指定する．これにより，「ニュース到着時に」や「毎日 12 時に」などの契機で問合せ評価を実行できる．

StreamSpinner のアーキテクチャ概要を図 -8 に示す．StreamSpinner は上記問合せに基づき，フィルタリング，リレーショナルデータベースへの蓄積，そしてストリームとリレーショナルデータベースの統合という 3 種類の基本機能を提供する．さらにラッパーを経由することにより，映像，音声等さまざまな情報源から流れくるメディアストリームとの統合も目指している．

StreamSpinner プロジェクトの中では，イベント駆動型連続的問合せを対象とした複数問合せ最適化機構を提案している．複数問合せ最適化とは，複数の問合せから共通演算を抽出して共有化し，本来複数回必要だった処理を一度で済ます手法である．従来の DBMS でも用いられている手法であるが，イベント駆動型連続的問合せ処理においては，その実行タイミングに着目することが重要であることを示したことがポイントである．提案方式では，他の問合せの再利用できそうな実行結果をそうでない実行結果から切り分け，前者のみをキャッシュに保持するアルゴリズムを開発し，大きな性能向上が得られることを示した[5)]．

StreamSpinner プロジェクトでは，このほか，ストリームをデータベースへ連続的に書き込みを行う場合に，書き込み処理の共有と遅延，ビューの利用等を用いて，その効率化を図る技術を提案している．また，分散ストリーム処理のための問合せ処理や高信頼化についても研究を行っている．高信頼化に関しては，データ転送量とリカバリ時間をバランス可能な新たな高信頼化技術を提案している．さらに，映像ストリームとタプルストリーム処理の統合，上記に記載の動的な情報源選択機能における効率的なラッパー制御等の方法についても研究を行っている．

StreamSpinner の情報は次から入手可能である：http://www.streamspinner.org/

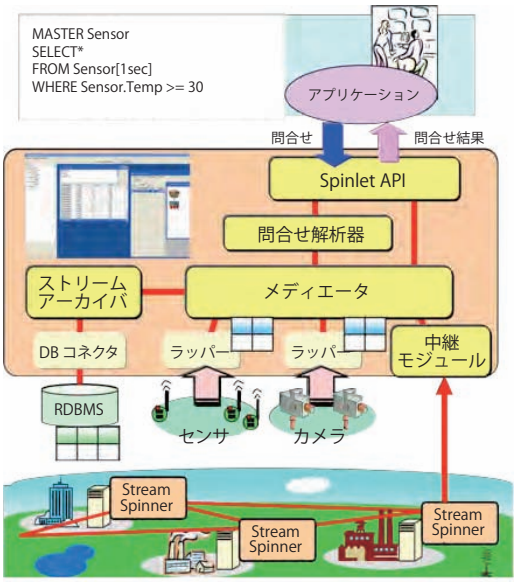

図 -8　StreamSpinner のアーキテクチャ

まとめと展望

本稿ではセンシングデータ処理基盤技術であるストリームデータ処理について解説を行った．この分野における研究論文は，SIGMOD，VLDB，ICDE などのデータベース関係の国際会議やジャーナルに多く発表されている．なお，SPE には商用版があり，日立製作所，Oracle，IBM，Sybase，Microsoft，EsperTech 等の各社から販売されている．

この分野における注目される研究の動向としては，以下のような点がある．第 1 の観点として，RFID 等各種センシング情報源から得られる不確実なストリームデータを処理する技法が最近注目を集めつつある．現状ではマルコフ性を考慮した確率的データストリーム処理システム Lahar やパーティクルフィルタにおけるストリーム処理を実現した CLARO 等が提案されている．今後，不確実なデータを産出するセンシング情報源の増大に伴って不確実ストリーム処理技術の研究がさらに展開すると考えられる．第 2 の観点として，リレーショナル演算や複合イベント処理では記述困難な応用に対応した拡張性や応用処理との融合の枠組みの構築がある．特定の応用処理に特化した SPE に関する既存研究としては，信号処理を対象とした MIT の WaveScope やパケット処理を対象とした AT&T の GigaScope 等が挙げられる．各種メディア処理やマイニング処理等を含めて，多様な応用処理と基盤エンジン技術の融合は大きな課題である．第 3 の観点として，情報爆発の中，今度一層増大するであろう大規模ストリームデータのスケーラブルな処理がある．性能向上のために GPGPU や FPGA 等のハードウェアを利用する研究が行われつつある．その中には，FPGA を用いてストリーム処理を高性能化する ETH Zurich の研究等がある．分散・並列処理ストリーム処理やクラウド環境への展開等も含めて，興味深い研究課題である．

参考文献

1) Arasu, A., Babu, S., and Widom, J. : The CQL Continuous Query Language : Semantic Foundations and Query Execution, The VLDB Journal, Vol.15, No.2, pp.121-142 (2006).
2) Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M. and Seltzer, M. : Network-Aware Operator Placement for Stream-Processing Systems, In Proceedings of the 22nd International Conference on Data Engineering (2006).
3) Hwang, J., Balazinska, M., Rasin, A., Cetintemel, U., Stonebraker, M. and Zdonik, S. : High-Availability Algorithms for Distributed Stream Processing, In Proceedings of the 21st International Conference on Data Engineering (2005).
4) Agrawal, J., Diao, Y., Gyllstrom, D. and Immerman, N. : Efficient Pattern Matching over Event Streams, In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (2008).
5) Watanabe, Y. and Kitagawa, H. : Query Result Caching for Multiple Event-driven Continuous Queries, In Information Systems, Vol.35, No.1, pp.94-110 (2010).
6) Chakravarthy, S. and Jiang, Q. : Stream Data Processing : A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing, 1st. Springer Publishing Company, Incorporated.

（平成 22 年 7 月 16 日受付）

北川 博之（正会員）　kitagawa@cs.tsukuba.ac.jp
筑波大学大学院システム情報工学研究科教授（計算科学研究センター兼務）．1980 年東京大学理学系研究科修了．理学博士．データベース，データ工学に関する研究に従事．本会，電子情報通信学会フェロー．
http://www.kde.cs.tsukuba.ac.jp/~kitagawa/

川島 英之（正会員）　kawasima@cs.tsukuba.ac.jp
筑波大学大学院システム情報工学研究科講師（計算科学研究センター兼務）．博士（工学）．2004 年慶應義塾大学大学院理工学研究科修了．2007 年より現職．http://www.kde.cs.tsukuba.ac.jp/~kawasima/

天笠 俊之（正会員）　amagasa@cs.tsukuba.ac.jp
筑波大学大学院システム情報工学研究科准教授（計算科学研究センター兼務）．1999 年群馬大学大学院工学研究科修了．博士（工学）．奈良先端科学技術大学院大学助手，筑波大学大学院システム情報工学研究科講師を経て，2009 年より現職．
http://www.kde.cs.tsukuba.ac.jp/~amagasa/