2004－DBS－134　(I)　(32)
2004／7／13

連続的問合せを用いたテキストストリームに対する類似検索

渡辺 陽介†，北川 博之†

概要

近年，Web 上のニュースサイトやメールマガジン，データ放送など，テキストを提供するデータストリームが増加し，テキストストリームに対する問合せ処理要求が高まっている．そのための枠組みとして，現在，連続的問合せが注目されているが，連続的問合せに関するこれまでの多くの研究では，単純なリレーショナル代数演算等を用いた問合せしか対象としていないため，文書の類似度に基づく結合処理やキーワードを用いたランキングなど，テキストストリームに対する問合せ要求を実現するための十分な機能を提供できていない．本研究では，我々の研究グループが構築しているデータストリーム統合システムに，テキストストリーム処理のための類似検索やランキングの概念を導入する．また，本稿ではそれらの処理を含んだ問合せの最適化手法についても検討する．

Similarity Search on Continuous Queries for Text Streams

Yousuke Watanabe†, Hiroyuki Kitagawa†

Abstract

Today, there are a lot of data streams, which provide information changing over time. Examples of data streams are web-based news sites, data broadcasting services, mailing-lists, and so on. Continuous query is one of solutions for query processing on data stream, but in previous researches, it only provides simple functions based on relational algebra. We need more powerful query facilities for integrating text streams based on similarities between arriving documents, and ranking them by keywords. In this paper, we introduce a notion of similarity search into continuous query, and we show an architecture of our data stream integration system which we are currently developing. We also discuss an optimization method for such queries.

1　まえがき

ネットワークを介した情報交換が活発に行われるようになり，我々は Web のニュースサイトやメールマガジン，データ放送，センサーネットワークなどから時々刻々と変化する情報を容易に入手することができるようになった．新しい情報が逐次利用者の手元に到着する，このような形態の情報源はデータストリームと呼ばれ，従来の RDBMS のような利用者の手元にあらかじめ蓄積された情報を扱う情報源とは大きく異なっている．データが時々刻々と提供されるということ以外にも，データストリームの主な特徴として，サービスが長期間に渡り提供されるため，最終的には膨大なデータが届く点，最新情報を扱うため，可能な限り速い情報の配信が求められる点などが挙げられる．現在，データストリームの数と種類が膨大になってきており，データストリームから届いたデータに対するフィルタリングや複数のデータストリーム同士の統合などの問合せ処理要求が高まっている．特に，文書データを配信するテキストストリームは利用の幅が広く，テキストストリームに対する問合せ処理の需要は高い．

データストリームに対する問合せ処理のための枠組みとして，連続的問合せ (Continuous Query)[1] がある．連続的問合せは，ストリームからの新規到着データに対して問合せ処理を適用することを繰り返し，前回実行時からの差分となる処理結果を徐々に生成していくものである．これまでに何種類もの連続的問合せが提案されているが，XML ストリームを対象としているものを除けば，基本的にストリームからの配信単位をリレーションの 1 タプルとみなし，リレーショナル代数演算の範囲内で扱える問合せのみを対象としてきた．しかし，テキストストリームに対する問合せ要求に応えるためには，通常のリレーショナル代数演算だけでは十分ではない．指定したキーワード集合に対する類似度に基づいたフィルタリングやランキング，テキスト同士の類似度による結合処理といった，従来のテキストに対する類似検索で行われてきた処理に相当する機能が必要である．

†筑波大学システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

本稿の目的は，テキストストリームに対する連続的問合せ処理を実現することである．我々の研究グループが構築しているデータストリーム統合システム [9, 10, 11, 12] に，テキストの類似検索やランキングの演算を導入する．また，我々がこれまで提案してきた連続的問合せの複数問合せ最適化手法を拡張し，これらの演算を含んだ問合せを対象とする最適化について述べる．

本稿の構成は以下の通りである．まず，2 で本研究の目的を明確にするため，テキストストリームに対する問合せ処理要求の例を示す．次に 3 では，我々の研究グループが構築してきたデータストリーム統合システムについて概説する．そして，4 で，本研究で導入する類似検索のための演算について述べ，5 ではそれらの問合せに対する最適化手法を説明する．6 で関連研究を紹介し，最後に 7 でまとめと今後の課題を述べる．

2 利用例

本研究の目的は，テキストストリームに対する問合せ処理を可能にすることである．より具体的に説明するため，以下に本研究が想定するテキストストリームの利用例を示す（図 1）．

例 1: あるニュース記事が届いたときに，同じ日に届いた別のサイトからのニュースに類似する記事があれば一緒に提供してほしい（記事の類似度に基づく統合）．

例 2: 過去 24 時間以内に届いたニュースから特定のキーワードまたはその類義語との関連度が高いものトップ 10 を 1 時間おきに求め，新規にランキングに入ったニュースがあれば通知してほしい（キーワード指定によるランキング）．

これらの要求は，類似度やランキングの概念を含んでいるため，リレーショナル代数演算だけでは実現できない．3 では，我々のデータストリーム統合システムにおける，これらの問合せ処理要求の記述法について述べる．

3 データストリーム統合システム

本節では，我々がこれまでに構築してきたデータストリーム統合システム [9, 10, 11, 12] について述べる．テキストストリームに対する類似検索の機能については 4 で説明する．

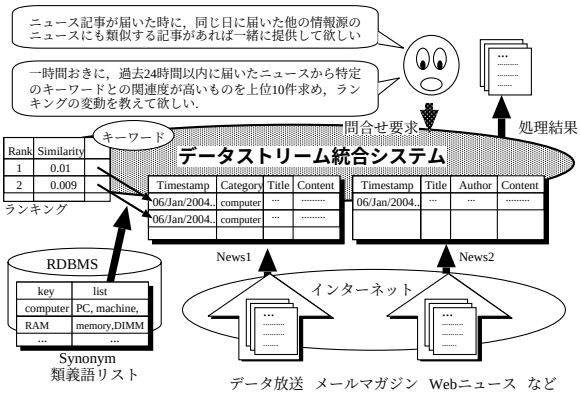

図 1: 統合利用例

3.1 連続的問合せ

本研究では，データストリームを無限にタプルが挿入されるリレーションとみなし，データストリームから到着する 1 配信単位をリレーションの 1 タプルとする．また，各タプルはそのタプルの到着時刻を格納した属性 TS を持つものとする．

本研究における連続的問合せ $CQ(\mathscr{M}, E, w)$ は，マスタ情報源集合 $\mathscr{M}$ と問合せ式 E，ウインドウ w から構成される．マスタ情報源 $M \in \mathscr{M}$ は，CQ を実行するきっかけとなるイベントを与えるデータストリームであり，E 中で参照していない情報源 (タイマーなど) も含めることができる．連続的問合せ CQ はマスタ情報源 M からタプル m_i が到着するたびに実行され，実行時刻 $t(= m_i[TS])$ から w 単位時間前までに届いたタプルを対象に問合せ式 E を適用する．そして，前回までの実行によって生成された結果との差分にあたる，新規到着データを元に生成されたタプルを時刻 t における実行結果として返す．

$$result_{CQ}(t) = E(t) - \bigcup_{t_i \in prev\_exec(t)} E(t_i)$$

ただし，$E(t)$ は時刻 t に問合せ式 E を適用した時の結果を表し，$prev\_exec(t)$ は t より以前の実行時刻の集合 $\{m[TS] \mid m \in M \wedge m[TS] < t\}$ を表すとする．また，本研究では E はリレーショナル代数演算と 4 で述べる類似検索のための演算からなるとする．ここでは議論を簡略化するため，リレーショナル代数演

```
MASTER     master_source_1 ,...
SELECT     attr_1 ,...
FROM       source_1 { [windowsize_1] } , ...
WHERE      conditions
[RANK BY   sort_attr k ]
```

図 2: 問合せ記述

```
MASTER     News1
SELECT     *
FROM       News1[now], News2[24hour]
WHERE      sim(News1.Content, News2.Content) > 0.1
```

図 3: 問合せ例 1

算のうち，選択演算，射影演算，直積演算，結合演算のみ考慮する．

本システムにおける連続的問合せ記述は，図 2 の構文に基づいて行う．問合せの MASTER 節には，マスタ情報源を与える．SELECT–FROM–WHERE の各節の意味は SQL とほぼ同様である．ウインドウ [6] は FROM 節の情報源記述の後に “[...]” を用いて指定するものとする．情報源ごとに異なった幅のウインドウを指定することが可能で，ウインドウ幅の指定を省略した際は，無限大のウインドウ幅を持つと解釈するものとする．RANK BY 節は，ストリームに対して *sort_attr* 属性の値に基づいてランキングを行い，トップ k 位に入るタプルを通知するよう指定するためのものである．RANK BY 節に対応する，ランキング演算については 4 で詳しく述べる．

2 で示した利用例について，問合せ記述を行うと図 3, 4 のようになる．ウインドウ指定の部分の *now* は現在時刻のみを含む，最小単位のウインドウを表し，24hour は 24 時間の時間幅を表す．関数 *sim* は類似度を求めるためのユーザ定義関数であり，詳細は 4 で述べる．Clock1h は，本システムが提供するクロックストリームで，1 時間おきに時間の経過を通知するストリームである．

3.2 システムアーキテクチャ

次に，本システムの構成について説明する．本システムはメディエータ／ラッパー方式を採用している．各情報源ごとに対応するラッパーが存在し，それぞれが新規到着データの検出およびデータを内部形式に変換する処理を行う．新規情報の到着を検出

```
MASTER     Clock1h
SELECT     *
FROM       News1[24hours], Synonym
WHERE      Synonym.key = 'Computer'
RANK BY    sim(News1.Content, Synonym.list), 10
```

図 4: 問合せ例 2

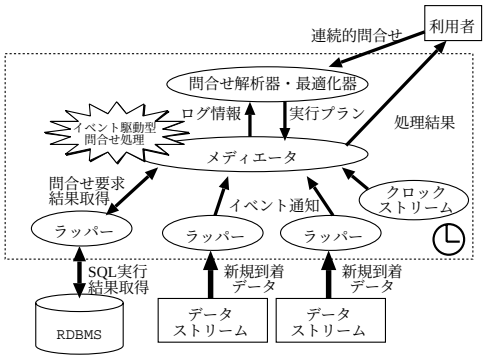

図 5: システムアーキテクチャ

すると，ラッパーはメディエータにイベントを通知する．メディエータはイベント通知を受けて，イベントをマスタ情報源とする連続的問合せを起動し，到着タプルに対して問合せ式の演算を適用した処理結果を利用者へ配信する．また，本システムはクロックストリームという，時刻の経過に応じて定期的にデータを配信する情報源を提供している．クロックストリームをマスタ情報源として用いることにより，図 4 の問合せ例 2 のような定期的に実行される連続的問合せを記述することが可能である．

メディエータ内部の処理について説明する．メディエータはイベント発生に応じて連続的問合せの問合せ式 E 中の演算を起動する．各演算は新規到着タプルを保持するための入力キューを持つ（図 6）．演算を一度適用すると入力キューの中のタプルは空になり，結果となるタプルが上位の演算の入力キューへ出力される．結合演算や直積演算は入力キューの他に，問合せで指定されたウインドウ w の範囲内に届いたタプルを保持するためのウインドウリレーションを持つ（図 6(II)）．これは新規到着タプルだけで結合演算や直積演算の結果を正しく生成することができないためである．

ウインドウリレーション中のタプルは，到着時刻からウインドウ w だけ時間が経過すると不要になるので，適宜削除される．長期間起動しない問合せで

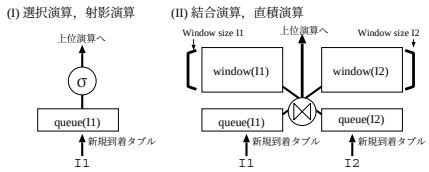

図 6: 演算子の構成

は，入力キューへ一旦蓄積したタプルも実行を待つ間にウインドウの範囲外へ出る可能性があり，その場合も同様に削除される．

4 テキストストリームに対する類似検索

本節では，テキストデータを扱うために本研究で新たに連続的問合せの枠組みへ導入する，類似検索のための演算について述べる．

4.1 類似度生成

本システムにおいては，文字列から類似度を求めるための処理はユーザ定義関数 $sim(x_1,x_2)$ として与えられるものとする．具体的な類似度の計算法については，情報検索の分野で一般的な手法であれば特に規定されないが，ここでは，文字列に含まれる単語の出現頻度ベクトルを求め，ベクトル同士の内積の値を類似度とする．この場合，ユーザ定義関数 sim は，文字列型から不要語除去を行い，重み付きベクトルを生成する関数 $vector(x)$ と，2 つのベクトルの内積を求める関数 $inner\_product(x,y)$ を用いて以下のように与えられる．

$$sim(x_1,x_2) = inner\_product(vector(x_1),vector(x_2)),\ (\in [0,1])$$

ただし，x_1 と x_2 はタプルの文字列型属性または利用者が指定するキーワード集合である．

関数 sim は，ユーザ定義関数呼び出しのための演算 $\hat{\pi}$ により評価される．関数評価演算 $\hat{\pi}$ は以下のように定義される．

$$\hat{\pi}_{A_f \leftarrow f(arg_1,\ldots)}(I) = \{t[A_1,\ldots,A_n,A_f] | t \in I \wedge t[A_f] = f(arg_1,\ldots)\}$$

$\hat{\pi}$ は，タプル t の属性または定数値を引数としてユーザ定義関数 f を評価し，その結果を属性 A_f として追加する演算である．

4.2 ストリームにおけるランキング

ランキング演算は，問合せ実行時刻からウインドウ w だけ過去の時刻までに届いたタプル集合 I に対してソートを行い，k 位以内に入っているタプルだけを通過させるというものである．

$$\hat{\sigma}_{A_i,k}(I) = \{t | t \in I \wedge rank(I,t[A_i]) \leq k\}$$

ここで，$rank(I,t[A_i])$ は I 中のタプルを属性 A_i の値に基づいて降順でソートした際の t の順位を表すものとする．定義から，同順位のタプルが複数あった場合，ランキング演算は k 個以上のタプルを出力する．連続的問合せでは前回までの実行結果との差分を出力する必要があるため，ランキング演算は k 位までのタプルのうち，新規にランキングに入ったタプルがあった場合にだけ，そのタプルのみを出力するものとする．

本システムにおけるランキング演算の実装について説明する．ランキング結果を得るために必要な処理として，以下の 2 つがある．

1. ウインドウの範囲内に入るタプルをソートする処理
2. トップ k 位以内に入る新規タプルを選び出す処理

(1) の処理では，連続的問合せが繰り返し起動する点を生かして，前回のソートの結果を利用することができる．ランキング演算は前回の実行までに届いたタプルのソート結果をランキングリレーションとして保持している．問合せが実行された時に，入力キューに新規到着タプルがあれば，そのタプルの属性 A_i の値に基づいて，ランキングリレーションの順位が保存されるように適切な場所へ挿入する．ランキングリレーションには，k 位以内のタプルだけなく，ランク外のタプルも保持される．これは，ランキング上位のタプルが時間の経過に伴ってウインドウ w の時間範囲に入らなくなり，処理対象から除外された影響で，以前ランク外だったタプルの順位が繰り上がる可能性があるためである．順位の繰り上がったタプルであっても，新規にランク入りした場合は処理結果として出力する．

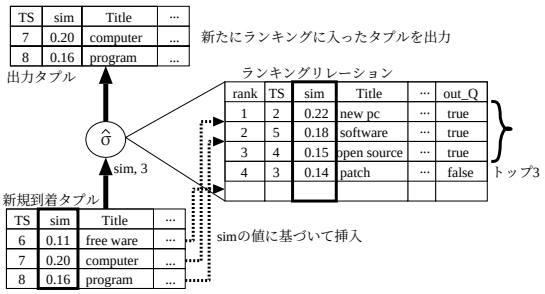

図 7: ランキング演算の動作例

(2) の処理について述べる．ランキングリレーションからトップ k 位までのタプル集合を取得することは，先頭から必要なタプル数だけスキャンすることで容易に行うことができる．また，そこから新規にランク入りしたタプルだけを抽出する処理については，一度でもランク入りしたことのあるタプルにフラグを立てて区別することで行う．

実際には，ランキングリレーションに保持しなければならないタプルは，ウインドウ w の範囲内のすべてのタプルではなく，一部だけで十分である．あるタプル t について，t よりも到着タイムスタンプが新しいかまたは同じ値を持ち，かつ t より上位にランクされるようなタプル集合が k 種類以上の A_i の値を持つ場合，もはや繰り上がりによって t が k 位以内にランクインすることはありえなくなるので，そのようなタプル t は廃棄することができる．

$$disposable(t) = count(S) \geq k$$

$$(t \in I \wedge S = \{s[A_i] \mid s \in I \wedge s[TS] \geq t[TS] \wedge s[A_i] > t[A_i]\})$$

ランキング演算の動作を図 7 を用いて説明する．ここでは，sim の値の上位 3 位までを出力するものとし，すでに $TS = 2,3,4,5$ のタプルがランキングリレーションに格納されており，また，$TS = 6,7,8$ のタプルが入力キューにたまっているとする．まず (1) の処理によって入力キュー中の新規到着タプルがランキングリレーション中の適切な場所へ挿入される．そして (2) の処理によって $TS = 7$ と $TS = 8$ のタプルが新規に上位 3 位にランクインしたことがわかるため，これらのタプルが出力される．このとき，$TS = 3$ のタプルは，自分より新しく，かつ自分より sim の値が高いタプルが 4 件存在するため，廃棄可能となる．

4.3 問合せプランの構築

本システムの連続的問合せは，基本的にリレーショナル代数に基づいているため，問合せプランの構築についてはほぼリレーショナル代数の枠組みに従う．ここでは，ユーザ定義関数評価演算 $\hat{\pi}$ とランキング演算 $\hat{\sigma}$ について述べる．

まず自明な制約として，演算 $\hat{\pi}$ は，ユーザ定義関数の評価結果を利用する選択演算よりも先に実行されなければならず，また，2 つ以上のストリームをまたいで引数をとる関数の場合，結合演算や直積演算の後でなければならない．これらの制約を満たす問合せプランの候補が複数ある場合に，どのプランを選択するべきかは，ユーザ定義関数自身のコストに基づいて判断する必要がある．一般に，ユーザ定義関数の評価コストの見積りは，ストリームのログに対して関数を適用するなどの方法を用いて行う．関数の評価コストが通常の射影演算や選択演算などと同程度である場合には，関連する選択演算と共に演算のプッシュダウンを行う．テキストの類似度を求める関数 sim については，ストリームという環境下で巨大な文書が頻繁に配信されてくることは稀であるという仮定に基づいて，本研究では選択演算等と同程度のコストとみなす．

ランキング演算 $\hat{\sigma}$ の場合，$\hat{\sigma}$ によって出力されたランキング結果に対して，さらに選択演算や結合演算のような演算を適用すると，タプルが途中で削除される可能性があり，正しくトップ k 位以内を出力できない．この問題については，従来の RDB におけるランキング演算を含んだ問合せの最適化 [5] と同様である．しかし，RDB では参照整合性制約が成立する場合は，結合演算の下位にランキング演算をプッシュ可能であるのに対し，ストリームでは，そもそも参照整合性制約を仮定すること自体が難しい．よって，本システムにおいては，問合せ処理の単純化のためにランキング演算は常に問合せプランの上位に配置する．

5 複数問合せ最適化手法

大量の問合せが与えられた場合，それらを個別に実行することは非常に手間がかかり，特に次々とタプルが到着するような状況では，1 つの処理が完了する前に次々と処理が発生し，システムのリソースを使い尽くしてしまう可能性がある．大量の問合せ

を効率的に実行するための技術として，複数問合せ最適化 [9, 10, 11, 12] と呼ばれる手法がある．我々の研究グループでは，これまでに連続的問合せに対する複数問合せ最適化手法を提案しており，複数問合せ最適化を用いて問合せに含まれる共通演算を共有し，処理結果を再利用することで処理の効率化を図ることが可能である．

本研究では類似検索を行うために，ユーザ定義関数 $sim(x_1, x_2)$ を導入した．異なったパラメタが関数 sim に与えられた場合には，当然異なった結果が生成されるので，複数問合せ最適化により関数の評価結果を共有することはできない．しかし，ユーザ定義関数の中にはパラメタが異なっていても共有可能な処理が含まれている可能性があり，その部分を共有することができれば，問合せ処理をさらに効率化することができると考える．本研究では，これまでの複数問合せ最適化の枠組みを拡張し，ユーザ定義関数中の共通演算も抽出して共有化の対象とする．

5.1 概要

ここでは，我々の研究グループがこれまでに複数問合せ最適化方式の概要を説明する．

連続的問合せに複数問合せ最適化を適用する際に問題となるのは，マスタ情報源の異なる問合せ同士では，共通演算を含んでいても，実行タイミングが離れすぎているために処理結果が共有できない場合があるということである．共通演算の処理結果が共有できるケースとして，以下の 2 つが挙げられる．

case 1 MASTER 節に同一のマスタ情報源を与えられた問合せ同士が，同じタイミングで実行されることは明らかである．もし，それらに共通演算が含まれているならば，生成される処理結果は共有可能である．

case 2 異なったマスタ情報源を与えられた問合せ同士であっても，非常に近いタイミングで実行されるものであれば，共通演算から生成される処理結果はかなり近いものとなる．この場合，処理結果の一部が共有可能である．

我々の最適化手法 [10, 11, 9, 12] はこれらの点を考慮し，問合せ集合に対する前処理として，2 段階のグループ化を行った後，生成されたグループごとに共通演算の共有化処理を適用する．

1. **ベースグループの生成**
   ベースグループは case 1 に相当する問合せの集合である．同一のマスタ情報源をもち，さらに共通の情報源を参照している問合せを，同一のベースグループに分類する．このような条件を満たす問合せの集合は，各問合せの記述を解析するだけで生成することができる．ベースグループは後述する到着ログを用いた問合せのクラスタリングにおける最小単位となる．

2. **到着ログを用いた問合せのクラスタリング**
   ベースグループの条件は厳しいので，一つのベースグループに属する問合せの数は平均的に少なく，このままでは共通演算の共有による恩恵が小さい．そこで，さらに case 2 の関係にあるより大きな問合せのグループを生成する．すでに問合せ群がベースグループの集合へ分類されているため，ここではベースグループ間で比較を行い，マスタ情報源の到着時刻が近く，かつ，処理結果が類似しているかどうかを調べる．

   しかし，クロックストリームのような等間隔で到着するストリームを除き，ほとんどのストリームデータの到着タイミングを事前に知ることは困難である．そこで本手法では，到着データと問合せの実行状況に関するログを用いて，ベースグループ同士の実行タイミングおよび参照するデータの時間範囲の関係を求め，参照範囲の類似度 (文書の類似度とは異なる) を調べる．

   全てのベースグループ同士で参照範囲の類似度を計算すると，m 個のベースグループに対して $m \times m$ の類似度行列が得られる．この類似度行列を用いて，通常の階層的クラスタリング [8] を行い，類似度の高いベースグループ同士のクラスタを生成する．

3. **共通演算の共有化**
   クラスタ内の問合せに出現する共通演算の処理結果は，ほぼ共有できることが保証されている．よって，クラスタごとに共通演算のマージを行い，それぞれ最適な問合せ実行プランを導出する．ただし，ストリームにおいては演算のコスト見積りが困難であるため，コスト計算に基づく最適化を行うことはできない．本手法では，問合せから共有される演算の数が最大となるよ

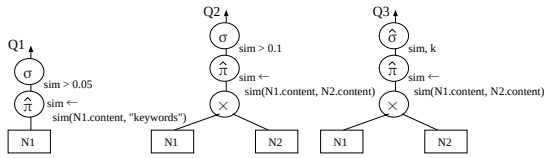

図 8: 問合せプラン（初期状態）

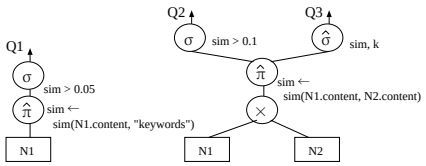

図 9: 共有された問合せプラン（関数分解なし）

うな問合せ実行プランを最適としている．

5.2 ユーザ定義関数の分解と共有

sim のような，複数のユーザ定義関数からなる関数を評価する際の戦略として，ひとつの $\hat{\pi}$ 演算ですべての関数を一度に計算するやり方と，複数個の $\hat{\pi}$ 演算にわけて段階的に計算するやり方の 2 種類が考えられる．連続的問合せを単体で実行する場合には通常前者を用いるが，複数問合せ最適化によって関数の途中結果が大量の問合せ間で共有可能な状況では，後者の方のプランを用いた方が効率がよくなる場合がある．本研究では，複数問合せ最適化の (3) の過程において，ユーザ定義関数を分解し，ユーザ定義関数の途中結果を共有できるようなプランを導出する．ただし，ひとつの関数を任意の細かさに分解可能だとしてしまうと，生成可能なプランの候補が爆発的に増えるため，問合せ最適化の処理が複雑になる．本研究では，ユーザ定義関数の設計者が，あらかじめ分解や共有の最小単位となる関数を定義し，システムに登録しておくことを想定している．関数 $sim(x_1, x_2)$ の場合は，$vector(x)$ と $innor\_product(x_1, x_2)$ である．

例えば，図 8 のような 3 つの問合せが与えられた状況を考える．関数を一括評価するポリシーでは，図 9 のように共有するしか選択肢がないので，このプランを最適とするほかにない．この場合，$Q2, Q3$ の *sim* は直積演算の下にはプッシュダウンすることもできないし，また，引数の異なる問合せ同士では *sim* の結果を共有することはできない．しかし，*sim* の処理を関数 *vector* と *inner_product* に分解し，2 段階で評価するポリシーを用いた場合，まず $Q2, Q3$ の関数 *vector* がプッシュダウン可能となるため，図 10 のように書き換えることができる．さらにプッシュダウンされた関数 $vector(N1.content)$ の部分に関しては，3 つの問合せで共通であり，その結果，図 11 のようなプランを導出することができる．

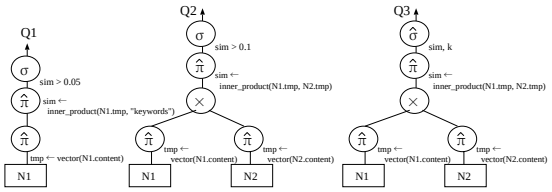

図 10: 問合せプラン（関数分解後）

5.3 ランキング演算の共有

ランキング演算は，問合せが完全に同一でなく，マスタ情報源やランキング演算のパラメタ k が異なっていても，一部の処理が共有可能である．4 で述べた，ランキング演算の (1),(2) の処理のうち，(1) の部分は単なるソート処理であるため，問題なく共有できる．ただし，(2) の部分については問合せごと個別に，k 位までに入った新規タプル探す処理を行う必要がある．

6 関連研究

連続的問合せを用いた代表的なデータストリーム処理システムとして，TelegraphCQ[3], Aurora[2], STREAM[7], NiagaraCQ[4] がある．これらのシステムのうち，TelegraphCQ, STREAM はリレーショナル代数の範囲内での問合せ記述をサポートとしており，NiagaraCQ では XML ストリームを対象とした問合せ，Aurora では独自に提案する代数による問合せ記述を行う．いずれも，テキストストリームに対する類似検索の機能は提供していない．また，これらのシステムは，多数の問合せを効率的に実行するための，複数問合せ最適化の機能を提供している．ただし，本研究で提案したようなユーザ定義関数の分解による共通演算の抽出までは考慮していない．

連続的問合せ以外でテキストメッセージを扱うシステムとして，publish/subscribe システムがあげられる．publish/subscribe システムでは，キーワードでテ

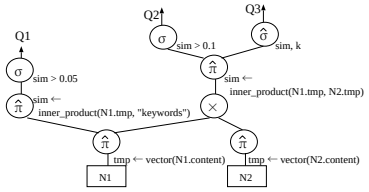

図 11: 共有された問合せプラン（関数分解後）

キストに対するフィルタリング条件を指定することができる．しかし，これらのシステムには他の情報源との統合を実現する機能はない．

RDBMS においてトップ k 件を取得するための研究として [5] がある．[5] では，k 件の結果が得られた時点で問合せ処理をとめるための Stop 演算を提案し，また，Stop 演算を含めた問合せの最適化手法を提案している．本研究の目的はデータストリームにおいて繰り返しランキングを計算することであり，RDBMS の環境における研究とは目的や前提が異なっている．

7 むすび

本稿では，テキストストリームに対する問合せ処理を実現するために，類似検索の概念を導入した連続的問合せを提案した．また，複数問合せ最適化においてユーザ定義関数を分解し，共有可能な部分を抽出するためのアイデアを提案した．

今後の課題として，提案手法の実装および実験・評価が挙げられる．

謝辞

本研究の一部は，日本学術振興会特別研究員奨励費 (15・330)，科学研究費補助金特定領域研究 (2)(#16016205)，基盤研究 (B)(#15300027)，CREST「自律連合型基盤システムの構築」による．

参考文献

[1] A. Arasu, S. Babu and J. Widom. “The CQL Continuous Query Language: Semantic Foundations and Query Execution,” Technical Report, http://dbpubs.stanford.edu/pub/2003-67 , 2003.

[2] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. “Aurora: a new model and architecture for data stream management,” VLDB Journal Vol.12, No.2, pp. 120–139, 2003.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. “TelegraphCQ: Continuous Dataflow Processing for an Uncertain World,” Proc. Conference on Innovative Data Systems Research 2003.

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. “NiagaraCQ: A Scalable Continuous Query System for Internet Databases,” Proc. ACM SIGMOD Conference, pp. 379–390, 2000.

[5] M. J. Carey, and D. Kossmann. “On Saying “Enough Already!” in SQL,” Proc. ACM SIGMOD International Conference, pp.219–230, 1997.

[6] J. Kang, J.F. Naughton, and S.D. Viglas. “Evaluating Window Joins over Unbounded Streams,” International Conference on Data Engineering, 2003.

[7] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. “Query Processing, Resource Management, and Approximation in a Data Stream Management System,” Proc. Conference on Innovative Data Systems Research 2003.

[8] G. Salton. “Automatic Information Organization and Retrieval,” McGraw-Hill Book Company, 1968.

[9] Y. Watanabe, and H. Kitagawa. “A Multiple Continuous Query Optimization Method Based on Query Execution Pattern Analysis,” Proc. International Conference on Database Systems for Advanced Applications, LNCS 2973, pp. 443–456, 2004.

[10] 渡辺陽介，北川博之．“連続的問合せに対する複数問合せ最適化”，DEWS 2003.

[11] 渡辺陽介，北川博之．“ストリームの動的特性変化を考慮した連続的問合せ最適化方式”，DBWS 2003.

[12] 渡辺陽介，北川博之．“問合せ最適化機構を備えたデータストリーム統合システムの開発”，DEWS 2004.