DEIM Forum 2021 A11-2

分散RDFストリーム処理システムにおける複数フィルタリング最適化

平方　俊行†　　天笠　俊之††

† 筑波大学大学院 理工情報生命学術院 システム情報工学研究群　〒 305-8577 茨城県つくば市天久保 1 丁目-1-1
†† 筑波大学 計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1 丁目 1-1
E-mail: †hirakata@kde.tsukuba.ac.jp, ††amagasa@u.tsukuba.ac.jp

あらまし　RDF ストリーム処理システムは，ストリームデータと知識ベースの統合を容易に行えるため，それらのデータを活用したアプリケーションで広く利用されている．ストリームデータを使用するアプリケーションでは，一般的に大規模なストリームデータに対して多くの問合せが発行されるため，フィルタリング処理を連続で行うことが想定される．しかし，多くの既存 RDF ストリーム処理システムは，単一マシン上での問合せ処理を想定しているため，大規模なストリームデータに対して複数のフィルタリング処理が最適化されていない．大規模なストリームデータを処理するには，複数マシンを利用した並列処理が有効であると考えられる．そこで本研究では，大規模ストリームデータに対応するために分散処理環境での RDF ストリーム処理システムの複数フィルタリング最適化手法を提案する．

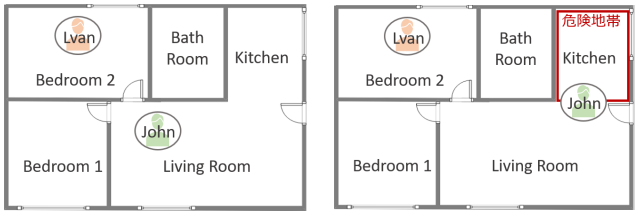
キーワード　ストリームデータ，RDF，並列分散処理

1 はじめに

近年，スマートフォンのようなセンサデバイスやソーシャルネットワーキングサービスの普及に伴い，継続的に送信され続けるストリームデータのデータ量が増加している [1]．またセンサデバイスの性能向上により，エッジセンサ上で機械学習や Web 上の知識を管理する知識ベースを用いた分析処理が実行可能となった [2]．よって，様々なストリームデータのデータ統合や推論処理を容易に処理できる RDF 形式のストリームデータである RDF ストリームを採用したセンサデバイスが注目されている．

そのため，RDF ストリームを採用したセンサデバイスを用いた，システムが多く開発されている [3] [4]．図 1 では，ヘルスケアにおいて病棟内にいる 2 人の患者がもつセンサデバイスから配信されるデータをモニタリングする例を示している．このシステムは，ある病院にいる患者が病院内の危険な場所へ行かないようにモニタリングしている．時刻 t では，ユーザ Lvan と John は危険地帯である Kitchen から遠い場所にいるため，システムは安全な状態であると判定できる．時刻 t+1 では，ユーザ John が危険地帯 Kitchen の領域に入ったため，システムは安全な状態でないと判断してアラートを発して従業員に知らせることができる．このようなシステムに RDF ストリームを採用することで，処理中での突然のデータ構造の変更や要求の変更への対応が可能になるため，容易に大規模モニタリングシステムの構築や管理が可能になる．

RDF ストリームを利用したシステムは，一般的に数百から数千のセンサデバイスに対して，同じく数百から数千のユーザが処理要求を発行されることが想定される．そのため，リアルタイムに送られる RDF ストリームに対して効率的なフィルタリングが望まれる．したがって，RDF ストリームを効率的に処理するための RDF ストリーム処理システムが多く提案されている．



(a) 時刻 t　　(b) 時刻 t+1

図 1: モニタリングシステムの例

しかし，多くの RDF ストリーム処理システムの既存手法 [5] [6] [7] [8] [9] は，単一マシンでの動作を想定しているものが多く，大規模な RDF ストリームや複数問合せを効率的に処理することができない．また，並列分散処理を想定した RDF ストリーム処理システムの既存研究 [10] [11] も，RDF ストリームがバーストしたときは特定のインスタンスに対して大きな負荷がかかり，全体の処理性能が大きく低下するという問題を解決できていない．実際のストリーム処理では，ストリームデータの到着レートが動的に変化する事が多くあるため，バーストへの対応は重要である．

そこで本研究では，RDF ストリームのバーストに対応可能な大規模並列 RDF ストリーム処理システムを提案する．具体的には，膨大な RDF ストリームと多数の問合せに対する処理を可能にするため，分散 RDF ストリーム処理システムで現在最も高速な CQELS+ [11] の改善を行う．CQELS+は，RDF

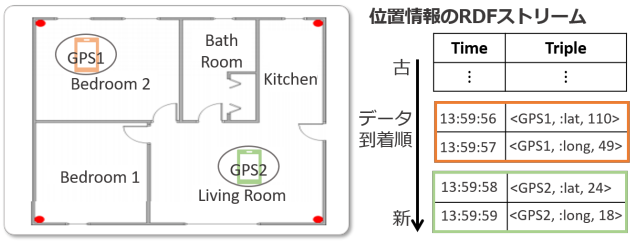


図 2: RDF ストリームの例

ストリームの到着レートが急激に増加した時，一部のインスタンスに急激な負荷がかかり，システム全体の処理性能が低下するという問題を解決できていない．そのため，到着レートが急激に増加した場合でもシステム全体の処理性能低下を防ぐために，RDF ストリーム処理システムの負荷分散手法を提案する．

本稿の構成について説明する．2 章では，関連研究について説明する．3 章では本稿で必要となる前提知識を定義する．4 章では，本稿でのクエリ再配置アルゴリズムについて説明する．5 章では，実データを用いた評価実験の計画について説明する．最後に 6 章で本稿についてまとめる．

2 前提知識

本章では，RDF ストリーム処理エンジン 2.1 節では，本研究で対象とする RDF ストリームの定義を行う．2.2 節では，RDF ストリームのフィルタリング方法についての定義を行う．

2.1 RDF ストリーム

本節では，本研究が対象とする RDF ストリームを説明する．RDF ストリームは，データ表現のためにさまざまなデータの表現に Web 上でデータ交換をするための標準モデルである RDF（Resource Description Framework）[12] を用いる．RDF は，主語 (subject), 述語 (predicate), 目的語 (object) の三つの要素で表され，これをトリプルと呼ぶ．トリプルは以下のように表される．

$$< subject, predicate, object >$$

トリプルの各要素は，それぞれ主語では Web 上のデータ識別子である URI(Uniform Resource Identifier) または空白ノードで表現される．述語は URI で表現される．目的語は URI またはリテラルまたは空白ノードで表される．リテラルは，URI のような形式化された識別子ではなく文字列のことを指す．

このように RDF では，この三つの要素を使うことにより，様々なデータ表現を行う事ができる．しかし RDF 形式のデータ表現のみでは，ストリームデータのような時系列ごとに継続的に流れてくるデータを表すことができない．

そこで Gutierrez らは，イベントの発生時刻とともに RDF を記録する方法を提案した [13]．この表現方法を時間トリプルという．以下に時間トリプルの定義を示す．

定義 **1** [時間トリプル]．主語 s，述語 p，目的語 o とした時，時間トリプルを以下と定義する．

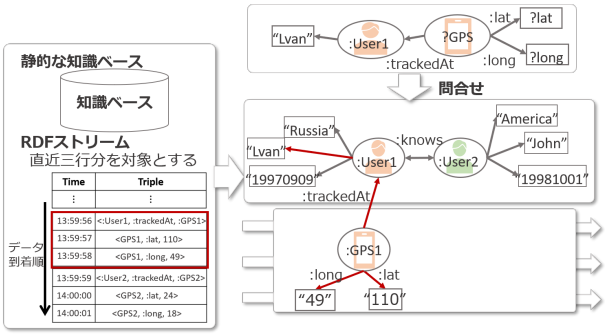


図 3: RDF ストリームのフィルタリング処理

RDF トリプル $< s, p, o >$ に対して，時間ラベル t を付与したとき，$<< s, p, o >, t >$ を時間トリプルとする．

そして RDF ストリームは，特定の期間に生成された時間トリプルの集合となる．以下に，RDF ストリームの定義を示す．

定義 **2** [**RDF** ストリーム]．開始時刻 t_1，終了時刻 t_2 とする時，RDF ストリームを以下と定義する．

$$\{<< s, p, o >, t > | t_1 < t < t_2\}$$

図 2 のように，2 種類のセンサデバイスから，RDF ストリームを取得した際，各位置情報を示すトリプルに一意のタイムスタンプを付与されている．この表現方法は，既存の RDF ストリーム処理システムにおいて多く利用されており，本研究でも対象とする．

2.2 RDF ストリームへのフィルタリング要求

膨大な RDF ストリームから，該当のデータを取得するために，RDF ストリーム処理システムは，連続的問合せが使用を使用する．この問合せは，静的な RDF 形式のデータに対する検索言語である SPARQL を拡張した形式であり，ウィンドウサイズを指定して対象データを作成することで，処理を行う．

図 3 では，直近三行のみを対象とする RDF ストリームと対応する知識ベースに対して連続的問合せを用いてフィルタリング処理を行う例を示す．連続的問合せでは，ユーザ Lvan が持っているセンサデバイスの ID と位置情報をフィルタリングするという要求を SPARQL 形式で登録されている．このとき要求に該当するデータは，静的な知識ベースから人間関係のデータ，RDF ストリームからセンサデバイスから発生する位置情報となる．そして，連続的問合せが指定するグラフパタンは，刻々と変化する RDF ストリームと知識ベースを統合することで作成できるデータセットをフィルタリングすることで結果を取得できる．該当する結果が得られない場合は，RDF ストリームで送信されなくなるまで処理し続ける．

3 関連研究

本節では，RDF ストリームを対象にした処理エンジンに関する既存研究について示す．3.1 章では，単一マシンを想定した RDF ストリーム処理エンジンの既存研究を示す．3.2 章では，分散処理を対象とした RDF ストリーム処理エンジンの既

存研究を示す．

3.1 単一マシンでの RDF ストリーム処理

単一マシンを想定した RDF ストリーム処理システムの研究は多く存在する [5] [6] [7] [8] [9]．その中でも，代表的な RDF ストリーム処理システムである C-SPARQL [5] と CQELS [6] を示す．

3.1.1 C-SPARQL

C-SPARQL [5] は，Balduini らによって提案された初めての RDF ストリーム処理システムである．このシステムは，まず与えられた問合せに基づき，RDF ストリームからスナップショットを作成する．そして，作成したスナップショットを RDF 処理システムである SPARQL に処理を行わせることで， RDF ストリーム処理の実現を可能とした．しかし，この RDF ストリーム処理システムは，処理の一部を SPARQL に任せるため，処理がブラックボックス化するため，処理の最適化出来ないという問題が存在した．

3.1.2 CQELS

C-SPRAQL の問題を解決するために，De-Phuoc らは処理を SPARQL に処理を行わせない，ホワイトボックスの RDF ストリーム処理システムである CQELS [6] を提案した．このシステムは，リレーショナルストリームデータを対象にした処理システム Eddy [14] を参考にしており，入力された RDF ストリームの選択率をから，実行計画を作成することを可能とした．そのため，実行計画を動的に変更することが可能となり，高速な処理が可能となった．

3.2 並列処理による RDF ストリーム処理

複数マシン上での問合せ最適化に関する研究は，単一マシンを想定した研究よりも少ないものの存在する．その中でも，CQELSCloud [10] や CQELS+ [11] は単一マシン上で最も高速な RDF ストリーム処理システムである CQELS を基にして複数マシン上で実行可能にした研究である．

3.2.1 CQELSCloud

CQELSCloud [10] は，CQELS を複数インスタンスに配置することで，膨大な RDF ストリームに対応可能した研究である．しかしこのシステムは，コンパイル時にクエリプランを決定し，その後変更を行うことがない．したがって，データの特性を考慮して動的にクエリプランを変更することができないため，マシンの性能を最大限に活かすことはできないという問題が存在する．

3.2.2 CQELS+

CQELS には，複数の結合処理を実行する際に再帰的な処理を実行しており，多くの処理時間がかかるという問題が存在する．そこで Chan らは，その問題を解決するため RDF ストリーム処理システム CQELS+ [11] を提案した．この処理システムは，CQELS を改良してフィルタリング結果を共有バッファに保存することで複数問合せの最適化を可能にした．そして，膨大な RDF ストリームを効率的に処理するために，複数インスタンスに RDF ストリームを配置して，並列分散処理を

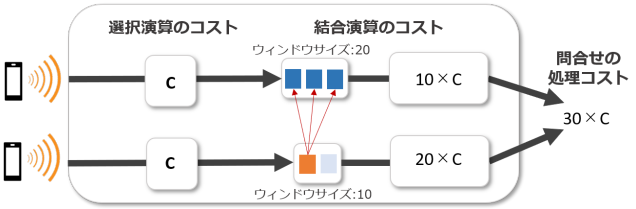


図 4: 1 トリプルあたりの処理コスト算出例

可能にした．問合せの配置方法は，以下の三種類を提案した．順番に配置を行うラウンドロビンの方法．平均レイテンシが最も低いインスタンスを優先的に選択する方法．そして，データバッファ内の要素の総数が最小となるインスタンスを選択する方法である．しかしこれらの方法では，RDF ストリームの到着レートが急激に増加した場合，一部のインスタンスに急激な負荷がかかるため，システム全体の処理性能が低下するという問題が発生する恐れがある．そのため，マシンの性能を最大限活かす事ができない．

4 提 案 手 法

本章では，問合せ再配置のアルゴリズムについて示す．4.1 節では，システムの概要を示す．4.2 節では，パフォーマンスを示す統計量を取得方法を示す．4.3 節では，高負荷なインスタンスの検出手順を示す．最後に 4.4 節では，新しい問合せの追加方法を示す．

4.1 システム概要

分散処理環境における RDF ストリーム処理システムでの負荷分散手法の解説を行う．システムの実行手順は以下のとおりである．

(1) 事前準備として先行研究と同様に問合せを各インスタンスに配置する．
(2) 各インスタンスのパフォーマンスを示す統計量を取得する．
(3) 取得した統計量より高負荷なインスタンスを検出した場合，そのインスタンスの中で高負荷な問合せの特定を行う．
(4) コスト式を基にしてその問合せの他インスタンスへの再配置方針を決める．
(5) 実際に処理を止めて問合せの再配置を実行する．

手順 (1) は，[11] と同様の方法で配置を行うため説明は省く．手順 (2)-(5) の処理内容は，下記で詳しく説明する．

4.2 パフォーマンスを示す統計量を取得

各インスタンスのパフォーマンスを確認するために継続的に取得する統計量について説明する．まず，RDF ストリームの到着レートを基に問合せの処理コストを算出する．考慮する演算は，選択演算と結合演算とする．RDF ストリーム処理システムでは，ストリームデータに対する差分計算を適応する．そのため，選択演算の処理コストは $O(1)$ となり，結合演算の処理コストは，バイナリの場合ウィンドウサイズを W とすると $O(W)$ となる．これらの処理コストを組み合わせることで問合せの処理コストを算出する．

図 4 では，RDF ストリームの 1 トリプルあたりの処理コストの算出例を示す．選択演算では，到着した RDF ストリームを逐次選定していくため，1 トリプルあたり定数コスト C で計算が可能である．次に結合演算では，選択演算で処理されたトリプル同士の比較を行うため，ウィンドウサイズ分のトリプルを参照する必要がある．その結果，結合演算の各処理コストは $10C$ と $20C$ となる．最後に，各処理コストを足し合わせることで，1 トリプルあたりの問合せの処理コスト $30C$ が算出できる．この処理コストを到着した RDF ストリームに対して行うことで問合せ全体の処理コストを算出する．

そして，問合せの処理コストを基に下記の三種類の統計量を算出する．一つ目は，インスタンスに登録されている問合せの累計コストである．累計コストの計算方法は，各インスタンスに登録されている問合せの処理コストの加算とする．二つ目は，インスタンス内の移動対象問合せの処理コストである．三つ目は，移動対象とした問合せを実際に移動させる際に発生する処理コストである．移動させる際に発生する処理コストは，登録されている問合せを処理するのに必要な時間とする．これらの三種類の統計量を用いて問合せの再配置の方針を決定する．

4.3 高負荷なインスタンスを検出

次に，負荷を分散する必要がある高負荷なインスタンスの検出を行う．本研究では，先ほどインスタンスに登録されている問合せの累計処理コストが最大となるインスタンスを高負荷なインスタンスとして負荷分散の対象とする．インスタンスの負荷を削減するためには，インスタンス内で負荷をかけている問合せ取除く必要がある．そのため，コストが高い問合せを優先的に移動対象の問合せとする．

4.3.1 コスト式を基に問合せの再配置方針を決定

検出した高負荷なインスタンス内の問合せの再配置方針をコスト式を用いて決定する方法を説明する．高負荷なインスタンスに登録されている問合せの再配置を行うには，高負荷なインスタンスの累計処理コストから高負荷な問合せを他にインスタンス移動させたときの各インスタンスの処理コストを計算すれば良い．このとき，再配置方針の決定に使用するコスト式は以下のように定義する．下記のコスト式では，コスト式では，I_i を i 番目のインスタンス，$P(I_i)$ をインスタンスに登録されている問合せの累計コスト，$S(I_i)$ をインスタンス内の移動対象の問合せ処理コスト，$R(I_i)$ を移動対象とした問合せを実際に移動させる際に発生する処理コスト，そして $I_{i_{max}}$ を問合せの累計コストの最大値を持つインスタンスとする．

$$P(I_{i_{max}}) > P(I_i) + S(I_{i_{max}}) + R(I_{i_{max}}) \qquad (1)$$

コスト式 (1) は，最も高負荷なインスタンスに登録されている問合せを他インスタンスに移動させた場合，全体の処理コストが削減されるかどうかの判定を行うことができる．つまり，(1) が成り立つ場合は，問合せのシステムの処理性能が向上できる可能性が高いと考えられる．

4.3.2 問合せの再配置を実行

システムの処理コストを削減するために，コスト式を基に作成した方針により問合せの再配置を行う．

アルゴリズム 1 問合せの再配置

Input: インスタンス : I, 問合せ: Q

```
1: i_max = 1
2: for i ∈ [1, 2..., |I|] do
3:     P(I_i) = Σ_q^{Q_{I_i}} (C(q))
4:     S(I_i) = max_q^{Q_{I_i}} (C(q))
5:     R(I_i) = Σ_q^{Q_{I_i}} (LC(q))
6:     if P(I_i) > P(I_{i_max}) then
7:         i_max = i
8:     end if
9: end for
10: for i ∈ [1, 2..., |I|] do
11:     if P(I_{i_max}) > P(I_i) + S(I_{i_max}) + R(I_{i_max}) then
12:         インスタンス I_{i_max} 内の処理コストが最大の問合せ q_max を
            インスタンス I_i に再配置
13:     end if
14: end for
15: return I
```

アルゴリズム 1 では，問合せ再配置を実行するアルゴリズムを示す．インスタンス内の現時点での三種類の統計量を取得する (3-5 行目)．その中で，インスタンス t の累積処理コスト $P(t)$ が全インスタンス内で最大であるかどうかの判定を行う (6-8 行目)．インスタンス内の高負荷な問合せを移動させることで全体の処理コストが低下するかどうか確認して，コスト式より低下すると考えられる．そのため，高負荷をかける問合せ q_max を他インスタンス t に再配置する (11-13 行目)．問合せの再配置は，既に到着している RDF ストリームの処理を終えた後に行う．

以上の流れを連続的な問合せが実行され続ける限り，処理し続ける．

4.4 新しい問合せの追加

新たな問合せが追加されたとき，各インスタンスで管理する問合せを更新する方法を紹介する．各分散配置手法の方法に沿って問合せの配置を行う．例えば，ラウンドロビン方式の場合，新たに問合せが追加されると，各インスタンスに順番に新たな問合せを配置する．

5 性能評価について

今後に行う予定の実験の計画を示す．5.1 節では，本研究の実験環境について説明する．5.2 節では，本実験で使用するベンチマークを示す．5.3 節では，実験時に指定するパラメータについて解説する．5.4 節では，比較手法について示す．5.5 節では，評価実験の結果を示す．

5.1 実 験 環 境

本実験の実行環境を表 1 に示す．本実験は，CentOS 7.9, Intel Xeon CPU E5-2620 2.10 GHz，および 64 GB RAM を搭載した計算機を用いた．全てのアルゴリズムは Java で実装

した．各インスタンスは，JVM を立てて処理を行い，4 GB ずつメモリを割り振った．

表 1: 実行環境

CPU	Intel Xeon CPU E5-2620 2.10 GHz 6 core ×1
OS	CentOS 7.9
RAM	64 GB

5.2 ベンチマーク

実験時に利用するベンチマークを紹介する．ベンチマークは，CityBench [15] を使用する．CityBench は，CityPulse EU FP7 プロジェクトにより作成された，IoT 機器を使用したシステムに対する実問合せとデータセットからなる RDF ストリーム処理システム専用のベンチマークである．このベンチマークは，データが生成される環境やそれを使用するアプリケーションの要件が動的に変化する可能性を考慮した評価が可能である．ソースコードは，Github 上にて公開されているため自由に利用する事ができる [1]．

CityBench のデータセットは，デンマークのオーフス市に設置されているセンサーから収集された RDF ストリームと地形情報を管理する知識ベースで構成されている．CityBench の概要を図 5 に示す．構成要素は，RDF ストリームから自動車から得られるセンサデータ，駐車場から得られるデータ，天候データ，市内から計測される大気汚染のデータと人の位置情報データからなる．

CityBench の問合せは，上記のデータセットを用いたスマートシティアプリケーションで利用される十二種類の問合せからなる．各問合せは，交通情報，駐車場情報，地域の環境情報に関する検索を対象としている．問合せの内容は以下となる．

（1） 予定していた地域の各道路の渋滞レベルの通知

（2） ある地点の経路の混雑度や天候の通知

（3） 目的地までの平均的な混雑度と移動時間の目安の通知

（4） ある地点に最も近い観光地の通知

（5） あるイベント X が開催されている道路の混雑度の通知

（6） 現在地から 1 k m圏内の駐車場の状況の通知

（7） 目的地近くの駐車場が満車になると通知

（8） 図書館 X の近くにある駐車場を通知

（9） ある地点から値段が一番安い駐車場の空き状況の通知

（10） 市内で最も汚染されている地域を通知

（11） ある地点でセンサを検知できなかった場合に通知

（12） ある道路の混雑度が過去 20 分以内に 3 回以上，閾値を超えた場合に通知

1 : CityPulse EU FP7 Project. CityPulse Dataset Collection. http://iot.ee.surrey.ac.uk:8080/software.html(閲覧日：12 月 23 日)

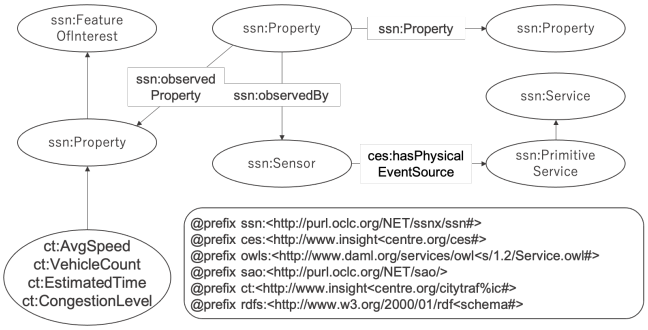


図 5: CityBench データセットの概要図

5.3 パラメータ設定

表 2 に，本実験で用いるパラメータを示す．インスタンス数は，実際に処理を行うインスタンスの数である．クエリ数は，問合せをランダムに該当の数だけ生成する．到着レートは値が大きくなるほど，単位時間あたりの到着するトリプル数が増加する．太字で表している値は初期値である．各パラメータを変更する際は，その他のパラメータ値は初期値を用いる．

表 2: パラメータ設定

パラメータ	値
インスタンス数	3,**6**,9,12
クエリの数	**300**, 600, 900, 1200

5.4 比 較 手 法

比較手法は，CQELS+ [11] にて提案されていた RDF ストリームに対する問合せの分散配置手法である，クエリが到着すると順番にインスタンス選択を行い配置していく手法であるラウンドロビン方式を採用した．この手法は，[11] にて実験で最も安定した手法であったため，本実験の比較手法とした．また RDF ストリーム処理エンジン CQELS+ は，オープンソフトウェアとして公開されているため，実験時に利用した．

5.5 評 価 方 法

測定方法は，全問合せの連続的な問合せの処理化完了するまでの処理性能を評価する．一定時間に均等に問合せが配置された際の，RDF ストリーム処理エンジンが処理できるスループットとレイテンシの測定を行う．評価項目は，パラメータを変化させたときのスループット，レイテンシとする．以前の問合せが RDF ストリーム処理システムに登録されると，新しいクエリを逐次登録する．全ての問合せが登録された後，観測を行った．

5.6 インスタンス数の影響

インスタンス数を変化させたときのスループットとレイテンシ計測結果を表 3 に示す．

インスタンス数を変化させたときのスループットは，比較手法と比べて提案手法は一秒あたりの処理できるトリプル数が増加していることが分かる．これは，インスタンス数が増加すると，各インスタンスで管理する問合せ数が少なくなるため，ス

表 3: インスタンス数を変化させたときのレイテンシ・スループット測定結果

	throughput(triple/sec)			latency(msec)		
サーバ数	提案手法	比較手法	処理効率	提案手法	比較手法	高速化率
3	2742.43	703.33	**3.90**	27.61	587.20	**21.27**
6	7437.96	704.01	**10.57**	84.33	822.11	**9.75**
9	9542.01	1032.13	**9.24**	86.91	851.24	**9.79**
12	11526.35	1385.06	**8.32**	54.70	796.47	**14.56**

表 4: 同時実行問合せ数を変化させたときのレイテンシ・スループット測定結果

	throughput(triple/sec)			latency(msec)		
クエリ数	提案手法	比較手法	処理効率	提案手法	比較手法	高速化率
300	7437.96	704.01	**10.57**	85.79	822.11	**9.58**
600	19301.10	2212.01	**8.73**	217.36	1201.17	**5.53**
900	15451.10	485.75	**31.81**	194.78	1144.93	**5.87**
1200	14772.78	184.50	**80.07**	310.63	1568.64	**5.05**

ループットが増加する．また，各問合せの処理コストを考慮することで，再配置先を正確に見積もることができる．そのため，サーバ数を増加させた場合の処理効率は，比較手法よりも効率化することが確認できた．

インスタンス数を変化させたときのレイテンシも，比較手法と比べて提案手法が低下していることが確認できた．これも，スループットと同様, インスタンスた増加すると，各インスタンスで管理する問合せ数が少なくなるためにより，レイテンシが低下していることが確認わかる．

5.7 問合せ数の影響

同時実行問合せ数を変化させたときのスループットとレイテンシ計測結果を表 4 に示す．

問い合わせを変化させたときのスループットは，比較手法と比べて提案手法が 1 秒あたり処理できるトリプルすうが増加していることが確認できた．これはクエリ数を増加させた場合，比較手法が一部のインスタンスに高負荷な問合せが集中した場合対応できない．その一方提案手法は，問合せの処理コストを正確に判定する事ができるため，低負荷なインスタンスに高負荷な問合せを再配置することができる．

6 おわりに

本稿では，並列分散処理環境において，複数のインスタンスを用いて問合せを分散管理して処理を分担することで，大規模 RDF ストリームデータを効率的に処理する問題に取り組んだ．

既存手法では，問合せを再配置することができないため，RDF ストリームがバーストすると，システムの処理性能が低下するという問題が発生する．そのため本研究では，高負荷なインスタンス内の問合せをコスト式を元にして，低負荷なインスタンスに再配置を行い，全体の処理コストを削減する手法を提案した．

評価実験の結果，インスタンス数を増やしていくと既存手法と比べて最高で約 10 倍のスループット向上を確認できた．また，問合せ数を増やした場合でも，最高で約 80 倍のスループットの向上が確認できた．

今後の課題は，さらなる性能評価を行う予定である．性能評価では，ラウンドロビン方式以外の負荷分散手法を実装して実験を行う予定である．また，RDF ストリームのデータ量を大幅に変化させた際の性能評価も行う予定である．また，本手法は問合せ数を増加させた時，処理性能が向上しない場合が存在した．そのため，それらを改善するためにより細かい粒度でも負荷分散手法の考案を行う予定である．

謝　辞

この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP20006）の結果得られたものです．

文　献

[1] Sherif Sakr, Marcin Wylot, Raghava Mutharaju, Danh Le Phuoc, and Irini Fundulaki. Processing of RDF Stream Data. *Linked Data*, pp. 85–108, 2018.

[2] Daniele Dell' Aglio Marco Balduini Marco Brambilla Emanuele Della Valle Karl AbererAndrea Mauri. TripleWave: Spreading RDF Streams on the Web. In *Proceedings of the 12th International conference on The Semantic Web*, p. 140–149, 2016.

[3] Tarek Elsaleh, Shirin Enshaeifar OrcID, Roonak Rezvani, Sahr Thomas Acton, Valentinas Janeiko, and Maria Bermudez-Edo. IoT-Stream: A Lightweight Ontology for Internet of Things Data Streams and Its Use with Data Analytics and Event Detection Services. *sensors*, Vol. 5, No. 953, 2020.

[4] Edna Ruckhaus, Jean-Paul Calbimonte, Raúl García Castro, and Oscar Corcho. From streaming data to Linked Data - A case study with bike sharing systems. In *Proceedings of the 5th International Workshop on Semantic Sensor Networks*, Vol. 904, pp. 109–114, 2012.

[5] Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: A Continuous Query Language for RDF Data Streams. Vol. 4, No. 1, pp. 3–25, 2010.

[6] Josiane Xavier Parreira Manfred Hauswirth Danh Le Phuoc, Minh Dao-Tran. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In *Proceedings of the 10th International conference on The Semantic Web*, p. 370–388, 2011.

[7] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. EP-SPARQL: A unified language for event processing and stream reasoning. pp. 635–644, 2011.

[8] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming SPARQL - Extending SPARQL to Process Data Streams. In *Proceedings of the 5th International conference on European Semantic Web Conference*, pp. 448–462, 2008.

[9] Srdjan Komazec, Davide Cerri, and Dieter Fensel. Sparkwave: Continuous schema-enhanced pattern matching over RDF data streams. *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pp. 58–68, 2012.

[10] Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Chan Le Van, and Manfred Hauswirth. Elastic and scalable processing of Linked Stream Data in the Cloud. In *Proceedings of the 12th International conference on The Semantic Web*, p. 280–297, 2013.

[11] Chan Le Van, Feng Gao, and Muhammad Intizar Ali. Optimizing the Performance of Concurrent RDF Stream Processing Queries. In *Proceedings of the 14th International conference on European Semantic Web Conference*, pp. 238–253, 2017.

[12] W3C. *RDF 1.1 Concepts and Abstract Syntax* . https://www.w3.org/TR/rdf11-concepts/.

[13] Claudio Gutierrez, Carlos A. Hurtado, and Alejandro Vaisman. Introducing Time into RDF. In *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, Vol. 19, pp. 207–218, 2007.

[14] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously Adaptive Continuous Queries over Streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, p. 49–60, New York, NY, USA, 2002. Association for Computing Machinery.

[15] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. CityBench: A Configurable Benchmark to Evaluate RSP Engines Using Smart City Datasets. In *Proceedings of the 14th International conference on The Semantic Web*, pp. 374–389, 2015.