D-015

RTA:オープンデータへのダイレクトなアクセス手法の提案

RTA:Method of Direct Inquiring Open Data Source

小坂 祐介† 村上 柊† ロラン トマ† 五嶋 研人† 遠山 元道†
Yusuke Kosaka Shu Murakami Thomas Laurent Kento Goto Motomichi Toyama

1. はじめに

オープンデータとは、特定のデータが誰もが望むように利用、または再掲載出来るような形で入手出来るべきであるという考え方である。例えば、そのようなデータの例には、政府機関が公開する統計データや気象情報などが挙げられる。

公開データは、主に現在 4 つの方法で公開されている。最初の 2 つの方法は、ダウンロード可能なファイルとしてデータを利用できるような形式での公開である。そして、それらをさらに２つに分けると、PDF や HTML などの人間が読める形式のものと、CSV や XML などのより構造的で機械的な形式のものに大別される。3 番目の方法は、Web サービスへの API 呼び出しによってデータを利用可能にすることであり、そして最後の方法は、Linked Open Data[1]という技術による公開である。

次にこれら４つの方法で公開されているデータをユーザーが利用するにあたり、どのようなメリット、デメリットがあるかを説明する。最初の 2 つの方法では、ファイルをダウンロードする必要があるだけで、データを簡単に取得できるようになっているが、その他の難点がある。実際に、ファイルが取得されると、ユーザーはそれらを解析して処理して、ローカルデータと統合するか、またはそれらを任意の種類のアプリケーションで使用する必要がある。この解析プロセスは、PDF ファイルなどのファイルでは不可能ではないにしても、非常に困難である。また最新のデータを保つ為には、データが更新されるたびにデータを取得し、解析する必要もあり、データの最新性を保つことも簡単には出来ない。また第 3 の方法では、既存のフレームワークにデータをより良く統合することが可能であるが、データの公開者は Web サービスと API を設計し、実装し、維持するためのそのデータ特有のプログラムを書く必要がある。さらに、ユーザーが予期していない方法でデータを取得する必要がある場合、API の機能によって、その利用が制限される可能性がある。最後に、LOD は複数のデータソースの公開と統合のためのフレームワークを提供するが、ユーザーの手元にあるデータと統合して、利用したいといった場合に不向きである。

関係データベースは現在、さまざまな状況やアプリケーションで使用されており、我々はオープンデータとローカルリレーショナルデータとの簡単なインタフェースが必要であると考えている。

我々は、オープンデータの現在の配布方法は、この必要性に対する効率的な解決策を提供しないと考える。実際、ファイルとして公開されたデータは、多かれ少なかれ整理されており、それらのデータを解析してリレーショナル形式にフォーマットすることは、かなりの労力を要する可能性が高い。オープンデータを公開するための一般的なフォーマットがないため、データの利用者はデータソースごとに、それに合ったデータ取得の方法をとる必要がある。同様に、API コールを使用してデータにアクセスする場合、データ利用者は新しい API ごとに新しいプログラムを作成する必要があり、多数のデータソースを統合する必要がある場合には、利用者側には大きな負担がかかる。最後に、LOD はリレーション形式でフォーマットされていないデータを提供するため、データを手持ちのデータとリレーショナル・パラダイムに統合するためのデータ管理に関するかなりの知識が必要である。

これらの問題を解決するために、我々は Remote Table Access（RTA）と呼ばれるシステムを提案する。 RTA では、データ公開者は Web アプリケーションを介してオープンデータをリレーショナルデータテーブルとして登録し、このテーブルへの読み取り専用アクセスをすべての RTA ユーザーに開放することができる。一方で、データの利用者は、オープンデータを、自分のデータベースに格納されているかのように、ローカルのデータと共に、単一の SQL クエリーを通じて照会することができる。このような仕組みを採用することで、上記の問題を解決し、オープンデータを公開者と利用者の両者が円滑に利用できることを目指している。

RTA は、ユーザーのクエリを処理してリモートデータをフェッチするクライアントアプリケーション、データの安全な公開を可能にするオプションのリモート側モジュール、および公開されたすべてのデータソースを登録する Public Table Library (PTL)の 3 つのシステムからなる。

以下、 ２節では、現在のオープンデータリモートアクセス技術の概要とこの分野の課題を示し、３節では RTA のユースケースの説明を行い、４節で具体的なアーキテクチャとクエリ処理について説明する。 ５節では、プロトタイプ実装を使って RTA を評価し、最後に６節でまとめと今後の課題について述べる。

2. 関連研究

この節では、現在のリモートおよびオープンのデータアクセスシステムとテクノロジーについて説明する。 最初に、Linked Open Data の技術を紹介し、次に関係データベースへのリモートアクセスのさまざまな方法を紹介する。

2.1 Linked Open Data

Linked Data はウェブ上でコンピュータ処理に適したデータを公開・共有するための技術である。Linked Data は、Resource Description Framework（RDF）[2]を使用して、データを subject / predicate / object の３つの要素で関係情報を表し、それは triple と呼ばれる。Wikipedia から Linked Data として抽出された情報を公開する DBPedia[3]のような大規

†慶應義塾大学 Keio University

模プロジェクトは、Linked Data を共有し、Linked Open Data の世界を創造することを目指している。

SPARQL[4]は、SQL によって提供される表現能力と検索能力を Linked Data にも適用する事を目的とした RDF の照会言語である。 しかし、ほとんどのデータは現在リレーショナルデータとして保存されているため、Linked Data とそれらのデータを統合するのは難しい作業である。本研究で、リレーショナルオープンデータに焦点を当てているのは、そういった理由からである。

2.2 関係データベースへのリモートアクセス

現在関係データベースでリモートデータを照会するための多くの技術が存在するが、ここではその中でも一般的なものを紹介する。

Remote Database Access（RDA）は、リモートデータベースの情報にアクセスするための標準プロトコルである。[5]のような概念実証のための実装を構築する研究が行われていたのにもかかわらず、プロトコルは主要な DBMS によってほとんど無視されているのが現状である。したがって、技術的にオープンなリレーショナルデータの基礎を提供しているにもかかわらず、RDA は現在の実際のシステムでは使用できない。

ISO / IEC 9075- 9 : 2016 規格は、SQL における外部データの管理の規格を定めている。この規格では、SQL の外部データ用のラッパーという概念を定義している。これは、現在の実装では postgres_fdw、Oracle DBLINK または MySQL の Federated 機能として定義されている。これらのラッパーは、RTA のように、リモートデータとローカルデータを SQL に統合する手段を提供するが、データ公開の仕組みは提供せず、また RTA では必要のない設定作業をユーザーに要求する。

Federated Database は、データベースのグループを単一のエンティティとして扱う管理方法を実現する。これにより、複数箇所に分散されているデータへの照会が容易になる。この Federated Database のアーキテクチャは[6] で提案された。 また RTA は、複数のデータソースをシームレスに統合する機能を提供するが、後で述べる理由から、読み取り専用の方法で提供する。代わりに、RTA では、Federated Database のモデルにはないデータの公開という概念が導入されている。

Graywulf[7]は、Microsoft SQL Server に格納された科学的データの効率的な格納、変換、統計分析、および表示のための再利用可能なコンポーネントを提供するプラットフォームである。分散データの透過的な表現を提供する拡張された SQL を使用して、データへの統一されたユーザーアクセスを提供するというコンセプトは、我々の提案する RTA と似ている。しかし、このプロジェクトでは 1 つの DBMS のみを考慮し、またデータの公開を意図していないため、RTA よりもはるかに機能が制限されてしまっている。

CloudMdsQl[8]は、複数の異種クラウドデータストアから単一の SQL ライクなクエリでデータを取得できるようにするシステムである。 CloudMdsQl クエリは、各ストアのクエリに分割され、各システムのネイティブクエリ言語に変換される。この原則は RTA の機能に似ているが、CloudMsQl は、RTA が提案しているように、データ公開の仕組みやエンドユーザー向けの簡単で細かい設定の必要なクエリを提供していません。クエリの最適化などの分野で CloudMdsQl で行われた作業は、本システムに適合させることができ、現在検討を行っている。

3. 研究目的とユースケース

3.1 研究目的

現在、様々な統計データや気象データなどのオープンデータと呼ばれる大量の公開データがある。ただし、このようなデータは、CSV や XML ファイルなどのダウンロード可能なファイルや Web サービスへの API 呼び出しなど、さまざまな方法で配布される。このようなデータを利用したいユーザーは、そのデータをリレーショナルデータとして利用するためにフェッチ、処理、統合するためのデータ固有のプログラムを作成する必要があり、データの利用者にとって、この作業は負担になる。このような事から、我々は現在のオープンデータが真にオープンであるとは考えていない。

したがって、私たちは、ユーザーとデータ公開者の両方に公開データの使用を容易にするシステムを提供することを考えた。データ所有者は、データを格納する主な手段の 1 つであるリレーショナルデータベースを直接公開することができ、データ利用者がリレーショナル形式（SQL を介して）で直接データにアクセスできるようにすることで、我々は、オープンデータの世界が拡張する事を期待している。

リレーショナルオープンデータのためのこのようなフレームワークは、気象情報を掲載しているサイトのような公開データを中心とした多数のアプリケーションの容易な実装を可能にする。さらに、データを公開する負担を軽減することは、データ所有者にデータの公開を促し、今以上に多くのオープンデータが流通することにも繋がると考えられる。

3.2 ユースケース

ここでは。いくつかのユースケースを説明することによって、RTA のビジョンを説明する。最初にリレーショナルオープンデータの概念を取り入れた RTA について記述し、次にモバイル用の RTA について述べ、最後に、組織内のリレーショナルデータを公開し共有するための「プライベート RTA」への拡張について述べる。

3.2.1 Relational Open Data のための RTA

このアプリケーションは、RTA の最初のアプリケーションである。この使用例では、データ公開者は RTA の公開インスタンスを使用してデータへの無制限の読み取りアクセスを提供し、エンドユーザーはそのオープンデータを自身のリレーショナルデータと簡単に統合することを可能となる。このユースケースでは、データは完全にオープンである。

その後、RTA を使って気象データや株価などの情報を公開することができる。このデータは、Web サービスによって簡単に統合されて表示されるか、または任意の利用者によって照会および分析される。

RTA がデータ公開者のデータベースへのアクセスに必要な情報(ホスト情報やアクセスするためのユーザー名やパスワード)を公開することは、セキュリティ上の懸念が生じる可能性があり、これについては考慮する。またエンド

ユーザーのプールは事実上無制限なので、データソースが普及した場合や「ホットスポット」になった場合は、パフォーマンスと同時実行の問題も考慮する必要がある。

3.2.2 Mobile RTA

リレーショナルオープンデータのフレームワークである RTA は、モバイルデバイスにも適用できる。モバイルネットワークの性質を考慮すると、この場合に取り出されるデータ量は当然のことながら小さくなる。このユースケースでは、RTA は、たとえば公共交通機関のスケジュールなどのデータを照会して表示するアプリケーションの設計を容易にすることができる。

このような状況に RTA を適用させるためには、モバイル機器環境（例えば、SQLite）用の RTA を実装し、インタフェースで接続することが必要であると考えている。

3.2.3 Private RTA

RTA の上記２つ以外の使用方法は、組織内で簡単にデータを共有するためのプライベートインスタンスとしての用途である。組織のデータが公開されることはめったにないため、このユースケースでは閉鎖されたプライベート RTA が必要である。一方、組織外のエンティティが内部で公開されたデータにアクセスできないようにするには、アクセスメカニズムが必要である。このように、RTA は読み取り専用のフェデレーテッドデータベースと見なすことができる。

また、社内共有ツールとして RTA を使用しながら、データの一部を世界中に公開するという利用も考えられる。このユースケースでは、RTA は公開データと機密データの両方にアクセスできるため、セキュリティの問題に直面するであろう。

4. アーキテクチャ

この節では、RTA のアーキテクチャについて説明する。まず、一般的なアーキテクチャについて説明し、システムのモジュールを列挙する。そして、最後に各モジュールについての説明を行う。

4.1 アーキテクチャの概要

RTA は、リモートのオープンデータを照会するクライアント(L)、リモートのデータ公開者(R)、そして、R のアクセス情報を格納し、L が R にアクセス出来るようにするための機能を持つライブラリ（PTL）の３つの要素で構成される。

図１と２は、それぞれ RTA のアーキテクチャを示している。L と R のデータのやり取りの方法が両者で異なるが、その部分の詳細の説明は 4.6 で行う。ここでは、データ公開のプロセスを説明する。(以下の説明で用いる番号と、図１、２の通し番号は全く別のものである。

(0) (B 方式のみ) WEB サービスをインストール
(1) データ公開者が PTL に公開するテーブルのスキーマ情報やアクセスに関する情報を登録
(2) L のユーザーがクエリ Q を実行すると、RTA クライアントは、リモートテーブルをスキャンする(クエリ中で#で示される)
(3) RTA クライアントが、PTL から R の接続情報をフェッチする

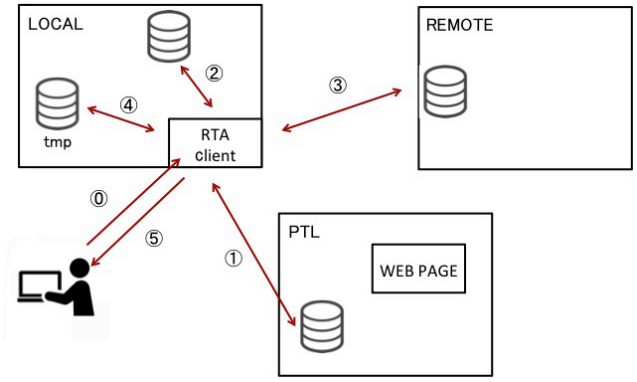

図１　RTA アーキテクチャ(A 方式)

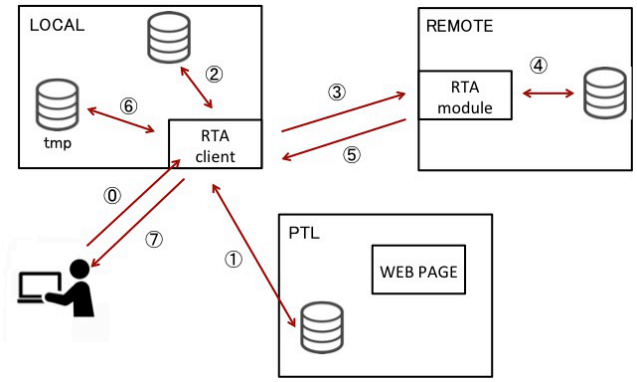

図２　RTA アーキテクチャ(B 方式)

(4) RTA クライアントは、フェッチした接続情報を元に R から必要なデータを取り出し、ローカルの一時データベースに格納
(5) リモートデータとローカルデータが テンポラリデータベースで結合され、フィルタリングされる
(6) クエリ結果がユーザーに返される

この後、各モジュールとステップの詳細について述べる。

4.2 Public Table Library

Public Table Library は、RTA を使用して公開されたすべてのテーブルのリポジトリである。公開者が新しいテーブルを登録して既存の公開テーブル情報を管理し、エンドユーザーが公開されたテーブルとそのアクセス名を見つけることを可能にする Web インターフェイスを提供する。

アクセス名は、RTA システム内のテーブルを一意に識別し、登録時に公開者によって設定され、エンドユーザーがクエリで使用する。

図３は、新しい公開テーブルをシステムに登録するために提案されたインターフェイスを示している。公開者は、DBMS、ホスト名、資格情報などのテーブル、アクセス情報、および公開するテーブルのリストを入力する。次に、公開者は、エンドユーザーの可読性を高めるために、そのテーブルの短い説明を追加出来る。この情報は PTL に保存され、テーブルは公開されたテーブルのリストに追加される。現在のバージョンでは、PTL はデータベース資格情報を使用して格納するため、公開者は RTA 専用の別個のユーザーを使用することが望ましい。

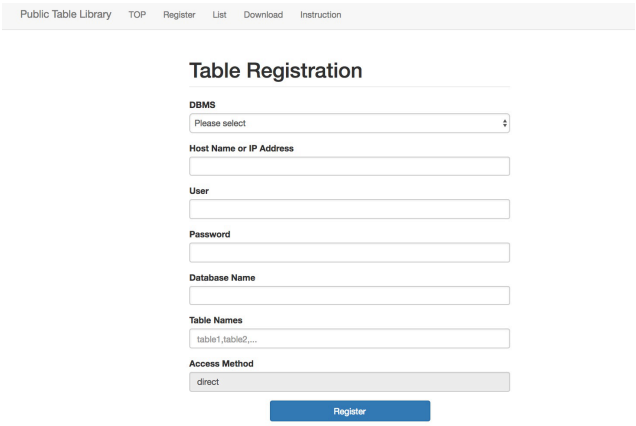

図３　公開テーブルの登録画面

登録されると公開されている表のリストをエンドユーザーは見ることができ、さらにそこから各公開テーブルの詳細を閲覧できる。ここでは、テーブルの各カラムの名前、型、説明を見ることができる。

4.3 RTA クエリの構文解析

RTA は SQL の構文を使用してローカルと公開データに跨るクエリを表現し、＃をテーブル名の先頭に付けることで、公開データソースであることを示す。 RTA クエリでは、リモートテーブルは、テーブル名の前に＃が付加されている。 この文字は、通常の SQL テーブル名とリモートマーカーである＃の間の衝突を避けるために、SQL 構文のテーブル名の無効な最初の文字を用いている。 したがって、RTA クエリの解析は、通常の SQL クエリの解析と同様である。

我々の実装では、若干の修正を加えた JSQLParser[9]を使用して、RTA クエリを解析している。

4.4 RTA クエリの実行

クエリ Q が解析されると、結果の表現は L で分解され、ローカルクエリ Q_l とリモートクエリ Q_r が生成される。クエリ Q_r は、実行されるリモートマシンに送信され、結果はテンポラリのローカルテーブルに格納される。次に、キャッシュされたリモート結果とローカルデータを結合して Q を実行する。それでは、アルゴリズム 1 のクエリを分解するプロセスを図４のクエリ Q を例にとり、詳しく説明する。

まず、Q の FROM 句を調べ、＃でマークされたすべてのリモートテーブルについて、テーブルが PTL に存在することを確認し、それぞれに対して RemoteConnector を作成し、テーブルを表し、その接続情報を含むオブジェクトをチェックする。この例では、stocks テーブルのために RemoteConnector を作成し、すべてのローカルテーブルに対して、LocalConnector にテーブルを追加する。その後、select 句の各カラムについて、そのカラムがリモートテーブルに属している場合、それを対応する RemoteConnector の selectItem に追加し、ローカルデータに属する場合は LocalConnector の selectitem に追加する。ここでは、s.ending_price を株式の RemoteConnector に、u.id、u.name、us.number を LocalConnector に追加する。最後に WHERE

```
SELECT u.id, u.name,
SUM (us.number * s.ending_price) as sum
FROM users u, user_stocks us, #stocks s
WHERE u.id = us.user id AND us.code = s.code
GROUP BY u.id, u.name
```

図４　クエリ Q

```
SELECT s.ending price, s.code FROM stocks
```

図５　クエリ Q_r

```
SELECT u.id, u.name, us.number, us.user id, us.code
FROM users u, user_stocks us
```

図６　クエリ Q_l

句を調べ、リモートテーブルまたはローカルテーブルからフェッチするカラムを、同様に各 connector の selectitem に追加する。ここで、s.code が RemoteConnector に追加され、us.user_id および us.code が LocalConnector に追加される。

このようにクエリが処理されると、RemoteConnector 毎にリモートクエリ Q_r が生成され、LocalConnector からローカルクエリ Q_l が生成される。この例のクエリ Q_r と Q_l は、図５と６にそれぞれ示す。

リモートクエリは現在、必要なすべてのテーブルをフェッチしてクエリ全体を完了させ、多くの不要なデータをフェッチする。このプロセスは、[10]や[11]で定義されている分散データベースのための十分に検討されたクエリ最適化技術を使用して、大幅に最適化することができる。最初に、それを最適化する直接的な方法は、リモートデータだけに関係する RTA where 節をクエリ Q_r に統合することである。そして、もう 1 つは、最初にクエリ Q_l をローカルで実行し、このクエリの結果を結合条件に関連付けてローカルデータと結合できるリモートデータのみを使用する方法である。これは、Q_r の IN 句を使用するか、リモートサーバーで半結合を実行することによって実現できる。しかし、後者の方法ではリモートマシン上の RTA にデータ挿入の権限が与えられる必要がある。

4.5 L から PTL への問い合わせ

L は、クエリで使用されたアクセス名を使用してリモートマシン（R）への接続情報について PTL に照会する。 これは、PTL 上の公開されたエンドポイントへの GET 要求によって行われる。 PTL は接続メソッド（4.6 を参照）と必要な接続情報（R の URL または IP、データベースに接続するためのユーザー名/パスワードの組み合わせ、またはテーブルが登録されていない場合のエラー）を返す。

4.6 L から R への問い合わせ

PTL からアクセス情報を得た後、L は R から必要なデータを取り出し、それをローカルのテンポラリデータベース

に格納する。これを達成するための 2 つの方法を提案する。データ公開者は、データを公開する方法を選択することができる。この方法の違いは、エンドユーザーにとって無関係である。

メソッド A と呼ばれる最初のメソッドは、従来の直接的なデータベース接続である。これには、リモートデータベースのアクセス資格情報をクライアントに伝える必要がある。しかし、これらのアクセス情報が傍受され、第三者によって悪用される可能性があるため、セキュリティのリスクがある。またこれらのアクセス情報に関連付けられているユーザーの権限を変更する必要や、ファイアウォールがデータベースへの外部接続を許可していることを確認する必要があり、それらをデータ公開者に強いることになる。

メソッド B と呼ばれる 2 番目の方法は、リモートマシンにインストールされているモジュールを介してデータにアクセスする。ローカルマシンが PTL に照会すると、このモジュールのホスト名を受け取り、REST 要求を使用してデータを取得する。実行時にオーバーヘッドが発生するが、データベースの資格情報が伝達されず、読み取り機能のみが公開されるため、システムのセキュリティが向上する。モジュールをインストールし、データベースの情報を設定ファイルに記述するだけで使うことができ、アクセス許可やファイアウォールについて心配する必要はない。このメソッドは REST 要求を利用するが、SQL 問合せを作成するだけで済むエンドユーザーと、これらの要求の複雑さを隠蔽する。

4.7 R におけるクエリ実行

A 方式を使用する場合、R は RDMS によって単に処理される。一方で B 方式を使用すると、リモートモジュールはクライアントの REST 要求を受け取り、データベースに必要なデータを照会し、クエリ結果を JSON 表現に変換し、クライアントへの応答としてデータを返す。

4.8 R におけるクエリ実行

リモートデータが検索されて、ローカルのテンポラリデータベースに格納されると、ローカルデータと結合されてクエリ結果が返される。 これは、図 7 に示す最終クエリ Q_{fi} によって行われる。 デンポラリテーブルは元のテーブルの名前に加え、異なる実行を区別するためにタイムスタンプが付加される。 これらのテーブルは現在、実行ごとに再作成される。

5. 実験

この節では、RTA のプロトタイプを使用して実際にクエリ実行を行った実験について説明し、得られた結果についての考察を述べる。 この現在のプロトタイプは mysql と postgresql とのみ互換性がある。

5.1 実験環境

我々は、2.7GHz の Intel Core i5、16GiB の RAM で、OS X 10.11 の Macbook pro 2015 を MySQL 14.14 を実行するクライアントマシン L として使用した。また 2.4Mhz、3.7GiB RAM、CentOS 7.2.1511 でクアッドコアの仮想マシンを、postgreSQL 9.2.15 と mysql 14.14 を実行するリモートマシン R として使用した。

Algorithm 1 Query decomposition algorithm

```
1: Input: Parsed RTA Query (Q)
2: Output: Local and remote SQL queries(Q_l, [Q_{r,n}]_{n=1..})
3: for all tableName in Q's FROM clause do
4:     if tableName begins with # then
5:         if access name is registered with the PTL then
6:             add a new RemoteConnector to the list of RemoteConnectors with the
               connection information and access name
7:         else
8:             return 'This access name is invalid'
9:         end if
10:    else
11:        add the tableName to the LocalConnector
12:    end if
13: end for
14: for all column in Q's SELECT clause do
15:    local ← true
16:    for all remoteConnector[n] in remoteConnector do
17:        if the column belongs to table remoteConnector[n] then
18:            add column to remoteConnector[n]'s selectItem list
19:            local ← false
20:        end if
21:    end for
22:    if local then
23:        add column to localConnector's selectItem list
24:    end if
25: end for
26: for all column in Q's WHERE clause do
27:    local ← true
28:    for all remoteConnector[n] in remoteConnector do
29:        if column ∈ remoteConnector[n] AND column ∉ remoteConnec-
           tor[n].selectItem then
30:            add column to remoteConnector[n]'s selectItem list
31:            local ← false
32:        end if
33:    end for
34:    if local AND column ∉ localConnector.selectItem then
35:        add column to localConnector's selectItem list
36:    end if
37: end for
38: for all remoteConnector[n] in remoteConnector do
39:    create remote SQL query Q_{r,n}
40: end for
41: create local SQL query Q_l
```

```
SELECT T1.id, T2.name,
SUM (T2.number * T3.ending_price) as sum
FROM users_20170418172910 T1,
user_stocks_20170418172910 T2,
stocks_20170418172910 T3
WHERE T1.id = T2.user id AND T2.code =
T3.code
GROUP BY T1.id, T2.name
```

図 7　クエリ Q_{fi}

PTL は、リモートデータベースと同じマシン上で運用を行った。

我々は、以下の 4 つの異なるネットワーク環境で実験を行った。

- ネットワークが使用されず、すべてのデータがローカルマシン L にある場合(LOCAL)
- R と L が同一ネットワーク上にある場合(LAN)
- R と L が同一ネットワーク上にない場合(WAN)
- L がモバイルネットワークを利用している場合(MOBILE)

LAN、WAN、MOBILE は、3.2 で述べたユースケースを反映しており、LOCAL は他 3 つの結果と比較を行う際のベースラインとして、機能する。

5.2 プロトタイプとクエリの最適化

RTA の概念証明として、我々は、クエリをローカルで実行する前にリモートマシン R からテーブル全体をフェッチするプロトタイプを実装した。このプロトタイプはアーキ

テクチャの概念証明にはなるが、実際に許容できるパフォーマンスを提供できていない。 このため、我々は将来的に RTA に統合される可能性のある最適化の効果を確かめる為に予備実験を行うことにした。

今回想定した最適化は、２種類である。１つ目は、WHERE 句を R に送ることで、リモートデータだけに適用される句が R に渡され、データの取得が少なくするものであり、２つ目は 、すべてのローカルデータが最初にフェッチされ、IN 句を使用してリモートデータを選択するために使用される。 そして、その２種類を両方行ったものを OPTIM と呼び、また実際のプロトタイプを NAIVE と呼ぶ。

5.3 実験と考察

本実験では、RTA のスケーラビリティの評価を行う。このため、R への接続情報、使用するネットワーク、全ての最適化のバージョンの全ての組み合わせで実験を行った。

この実験で使用したデータは、TPC–C ベンチマークのデータであり、https://github.com/AgilData/tpcc にあるベンチマークの Java 実装を使用した。この実験では、その中の stock テーブルをローカルに、item テーブルをリモートに配置し、実験を行った。item テーブルには 10000 タプルのデータが格納されており、stock テーブルのタプル数は 2000 から 100000 までの６パターンで実験を行った。

実験の結果を図８、９に示す。まず、ネットワーク環境の影響を考察する。 RTA は LOCAL の場合が一番高速に動作するが、これはオープンなリレーショナルデータの利用を反映していない。他の環境では、通信オーバーヘッドが生まれ、システムはより高いスループットとより低い待ち時間の環境でより良い性能を発揮する。我々は、このオーバーヘッドはオープンなリレーショナルデータ利用固有のものであり、妥当なものであると考えている。

次に、ローカルマシンとリモートマシン間の通信方法に基づいて結果を比較する。B 方式は、リモートデータベースとローカル・クライアントの間に RTA モジュールを導入し、データベースのレスポンスを中間表現に変換して、ローカルに送る。この工程の追加により、かなりのオーバーヘッド（平均では 18%のオーバーヘッド）が生まれるが、それでも、これにより提供されるセキュリティとスケーラビリティを考慮すれば、このオーバーヘッドは妥当なものであると、我々は考えている。

最後に、クエリの最適化は、ローカル実行と比較して RTA の絶対実行時間とオーバーヘッドの両方を削減するため、提案された最適化が効率的であることを示しており、これらの結果は、RTA のパフォーマンスのために通信量を制限することの重要性を示す。

6. 結論

本論文では、リレーショナルデータを公開し、公開されたデータをローカルデータと統合するためのフレームワークを提案することで、リレーショナルデータの公開の可能性を示した。そして、 このフレームワークをプロトタイプとして実装し、TPC-C ベンチマークを用いて、実験を行った。我々の結果は、簡単なクエリと妥当なオーバーヘッドによって、公開されたリモートデータをローカルデータと統合できることを示している。

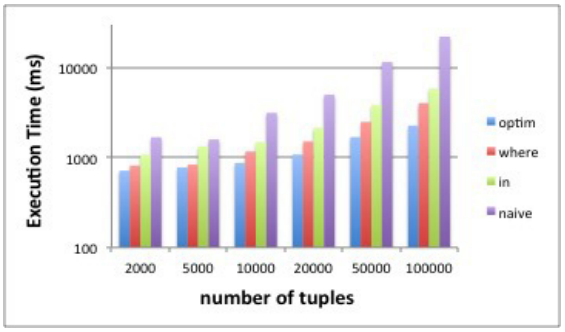

図 8　スケーラビリティ実験(A 方式)

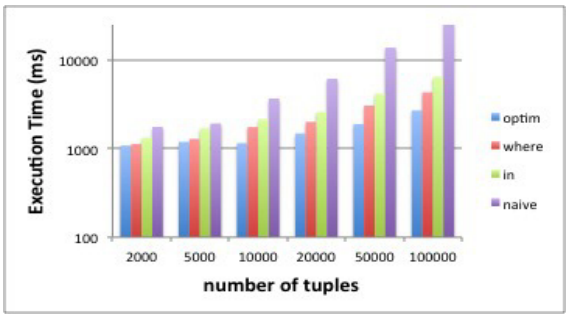

図 9　スケーラビリティ実験(B 方式)

今後は、ローカルマシンとリモートマシン間の通信を制限するためのクエリの最適化や、異なるユースケースや設定をカバーするためのフレームワークの拡張が必要と考えている。

参考文献

[1] Tom Heath and Christian Bizer. Linked data: Evolving the web into a global dataspace.Synthesis lectures on the semantic web: theory and technology, 1(1):1–136,2011
[2] World Wide Web Consortium et al. Rdf 1.1 concepts and abstract syntax. 2014
[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak,and Zachary Ives. Dbpedia: A nucleus for a web of open data.The semantic webpages 722–735, 2007.
[4] Eric Prud, Andy Seaborne, et al. Sparql query language for rdf. 2006
[5] Dean Arnold, Philip Cannata, Leigh Anne Glasson, Gary Hallmark, Bill McGuire,Scott Newman, Robert Odegard, and Harjit Sabharwal. Sql access: An implementation of the iso remote database access standard.Computer, 24(12):74–78,1991.
[6] Amit P Sheth and James A Larson. Federated database systems for managingdistributed, heterogeneous, and autonomous databases.ACM Computing Surveys(CSUR), 22(3):183–236, 1990
[7] Yogesh Simmhan, Roger Barga, Catharine van Ingen, Maria Nieto-Santisteban, Laszlo Dobos, Nolan Li, Michael Shipway, Alexander S Szalay, Sue Werner, and Jim Heasley. Graywulf: Scalable software architecture for data intensive computing. In System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on, pages1–10. IEEE, 2009.
[8] Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jiménez-Peris,Raquel Pau, and José Pereira. The cloudmdsql multistore system. In Proceedings of the 2016 International Conference on Management of Data, pages 2113–2116. ACM, 2016.
[9] JSQLParser. https://github.com/JSQLParser/JSqlParser, 2011. Last accessed 28 March 2017.
[10] Philip A Bernstein, Nathan Goodman, Eugene Wong, Christopher L Reeve, and James B Rothnie Jr. Query processing in a system for distributed databases (sdd-1). ACM Transactions on Database Systems (TODS), 6(4):602–625, 1981
[11] M Tamer Özsu and Patrick Valduriez. Principles of distributed database systems. Springer Science & Business Media, 2011.