Graph Databases

graphdb

2023 年 06 月 22 日

目次

第 1 章 Virtuoso 3
1.1 Installation . . . 3
1.2 Configuration . . . 4
1.3 Miscellaneous . . . 5

第 2 章 GraphDB 7
2.1 Installation . . . 7

第 3 章 Stardog 9
3.1 Installation . . . 9

第 4 章 Blazegraph 11
4.1 Installation . . . 11

第 5 章 AllegroGraph 13
5.1 Installation . . . 13

第 6 章 Fuseki 15
6.1 Installation . . . 15

第 7 章 RDF4j 19
7.1 Installation . . . 19

第 8 章 rdfstore-js 21
8.1 Installation . . . 21

第 9 章 Neptune 23
9.1 Setup . . . 23

第 10 章 Oracle 25
10.1 Installation . . . 25

第 11 章 PGX 27
11.1 Installation . . . 27

第 12 章 Neo4j 29

12.1 Installation 29
12.2 Configuration 32

第 13 章 ArangoDB 33
13.1 Installation 33

第 14 章 OrientDB 35
14.1 Installation 35

第 15 章 JanusGraph 37
15.1 Installation 37

第 16 章 TigerGraph 39
16.1 Installation 39

第 17 章 Summary: トリプルストア及びグラフデータベースに関する技術調査 41
17.1 背景 41
17.2 トリプルストアの各実装 41
17.3 グラフデータベースの各実装 55
17.4 データセット 60
17.5 クエリ 61
17.6 計測結果 66
17.7 考察と課題 68

There are plethora of implementations for graph databases and triplestores (RDF stores).

See https://graphdb.dbcls.jp/ for the details of the list.

Here we summarize documentations that include installation processes of those implementations.

第 1 章

Virtuoso

ウェブサイト

バージョン

- 7.2.5.1 (2018/08/15)

ライセンス

- GPLv2

その他必要条件

- automake, libtool, gperf (インストールされていない場合は apt でインストールする)

1.1 Installation

- ソースを取得

```
git clone git@github.com:openlink/virtuoso-opensource.git
cd virtuoso-opensource
git checkout v7.2.5.1
```

- コンパイル

```
./autogen.sh
./configure --prefix=/path/to/install/directory --with-readline
make
make install
```

ポート 1111 を既に使用していると、make の途中で失敗する。（テストに 1111 を使おうとするため）

configure に –with-port=1112 オプションを付けるなどすれば失敗を回避できるかもしれないが、1111 を空けておくのが確実。

- 起動

```
cd /path/to/install/directory/var/lib/virtuoso/db/
/path/to/install/directory/bin/virtuoso-t +wait
```

http://localhost:8890/sparql で SPARQL エンドポイントが起動していることを確認する。

- ロードデータの取得

```
curl -LOR http://example.com/example.ttl
```

```
isql 1111 dba dba
SQL> DB.DBA.TTLP_MT(file_to_string_output('example.ttl'), '', 'http://example.com/
↪example.ttl', 0);
```

1.1.1 Troubleshooting

./configure で OpenSSL に関するエラーが出ることがある。この場合は、libssl1.0-dev をインストールする。

```
sudo apt install -y libssl1.0-dev
```

Ubuntu 20.04 LTS では、パッケージ libssl1.0-dev が見つからないことがある。この場合は、/etc/apt/sources.list の末尾に以下を追加して apt update を行う。

```
deb http://security.ubuntu.com/ubuntu bionic-security main
```

```
sudo apt update
sudo apt install -y libssl1.0-dev
```

1.2 Configuration

デフォルトから設定を変更したい場合

- 設定ファイルの編集

```
vi /path/to/install/directory/var/lib/virtuoso/db/virtuoso.ini
```

問い合わせによって、virtuoso-temp.db のサイズが肥大化していくことがある。その場合、virtuoso.ini の中で TempAllocationPct(virtuoso.db に対して、何%まで許容するか) を設定する。参照

```
TempAllocationPct        = 100
```

1.3 Miscellaneous

1.3.1 X-SPARQL-MaxRows について

- クエリの結果が ResultSetMaxRows(デフォルト 10000) で切れた場合は、レスポンスヘッダーに X-SPARQL-MaxRows: 10000 が設定される。
- クエリの末尾を OFFSET 10000 LIMIT 10000 と修正して再度投げると、次の 10000 件を取得することができる。
- X-SPARQL-MaxRows が現れなくなるまで OFFSET をずらしていけば、全件を取得することができる。

1.3.2 VALUES に与える要素の最大数について

Virtuoso ソースコード libsrc/Wi/sparql_core.c に 0xFFF(=4095) とハードコーディングされている。

VALUES に与える要素の数が 4095 以上のとき、下記のようなエラーが返される。

```
Error: 400
Virtuoso 37000 Error SP030: SPARQL compiler, line 0: Too many arguments of a standard␣
↪built-in function in operator()
```

以下は、実験的に最大値を増やしてみた際の記録である。

変更 1

libsrc/Wi/sparql_core.c で以下の部分をコメントアウトする。

```
      if (argcount > sbd->sbd_maxargs)
        sparyyerror_impl (sparp, NULL, t_box_sprintf (100, "Too many arguments of a␣
↪standard built-in function %s()", sbd->sbd_name));
```

1 万要素でも渡せる。

変更 2

要素数をさらに増やすと、以下のエラーに遭遇する。

```
Error: 400 Bad Request
Virtuoso 37000 Error SP031: SPARQL: Internal error: The length of generated SQL text␣
↪has exceeded 10000 lines of code
```

libsrc/Wi/sparql2sql.h で以下の部分を削除する。

```
    if (SSG_MAX_ALLOWED_LINE_COUNT == ssg->ssg_line_count++) \
      spar_sqlprint_error_impl (ssg, "The length of generated SQL text has exceeded␣
↪10000 lines of code"); \
```

(次のページに続く)

(前のページからの続き)

すると、さらに以下のエラーが出る。

```
Error: 500 Internal Server Error
Virtuoso ..... Error SQ200: Query too large, variables in state over the limit
```

変更 3

libsrc/Wi/wi.h で以下の部分を修正する。

```
#ifdef LARGE_QI_INST
#define MAX_STATE_SLOTS 0xffffe
#else
#define MAX_STATE_SLOTS 0xfffe
#endif
```

MAX_STATE_SLOTS を 0xffffe にする (16 倍にする)。

すると以下のエラーが出る。

```
Error: 500 Internal Server Error
Virtuoso 42000 Error SQ199: Maximum size (32767) of a code vector exceeded by 3967441␣
↪bytes. Please split the code in smaller units.
```

変更 4

libsrc/Wi/sqlexp.c の以下の部分をコメントアウトする。

```
      if (BOFS_TO_OFS (byte_len) > SHRT_MAX)
        {
          sqlc_new_error (sc->sc_cc, "42000", "SQ199",
              "Maximum size (%ld) of a code vector exceeded by %ld bytes. "
              "Please split the code in smaller units.", (long) SHRT_MAX, (long) (byte_
↪len - SHRT_MAX));
        }
```

以上 4 カ所の修正で、結果的には、数万から 10 万要素でも答えが返ってくるようになる. ただし数万以上ともなると急激に遅くなり、何分もかかる。

1.3.3 Performance Tuning

http://vos.openlinksw.com/owiki/wiki/VOS/VirtRDFPerformanceTuning

第 2 章

GraphDB

ウェブサイト

更新履歴

本記事の対象バージョン：9.3.1 (2020/06/18 リリース)

ライセンス

- 無料版が存在するが、同時に発行できるクエリは 2 つまでとの記述有り

2.1 Installation

- GrahDB のフリーエディションを使用するため、サイト上でユーザ登録をする https://www.ontotext.com/products/graphdb/graphdb-free/
- ユーザ登録が完了すると、登録したメールアドレスにインストール用バイナリのダウンロード用リンクが送られてくる
- Download as a stand-alone server のリンクをクリックしてダウンロード（Desktop installation ではない方）
- ダウンロード後、zip を解凍してそのフォルダへ移動

```
$ cd path/to/unzipped/folder
```

- GraphDB をワークベンチモードで起動

```
$ sudo ./bin/graphdb
```

- ブラウザで localhost:7200 にアクセスすると、GraphDB のホーム画面が表示される
- 新しいリポジトリを作るため、左のメニューから、Setup->Repositories->Create new repository と選択する
- リポジトリの ID を入力し、最下部の Create をクリック

- ターミナル上で、以下のコマンドを実行して ttl データをインポート
  - なお、loadrdf コマンドではインポートするグラフに名前をつけることができない模様。
  - このため、グラフ名をつける必要がある場合はブラウザ上の http://localhost:7200/import から行う必要あり？$ sudo ./bin/loadrdf -f -i <repository_name> -m parallel
- インポートが完了したら、ブラウザに戻り左のメニューから SPARQL をクリックする
- リポジトリのリストが表示されるのでリポジトリを選択する
- SPARQL のエディタが表示されるので SPARQL を入力後、RUN で実行できる
- API から SPARQL を実行したい場合は、http://localhost:7200/repositories/<リポジトリ名> に GET リクエストを送ればよい

第 3 章

Stardog

ウェブサイト

Getting Started

本記事の対象バージョン：7.3.2（2020/07/01 リリース）

ライセンス

- 完全にフリーなライセンスは存在しないが、30 日分の体験版、1 年のアカデミックライセンスの体験版がある。

3.1 Installation

- Stardog の最新版をダウンロードして解凍して移動

```
wget https://downloads.stardog.com/stardog/stardog-latest.zip
unzip stardog-latest.zip
cd stardog
```

- Stardog のサーバを起動。

```
./bin/stardog-admin server start
```

- 初回起動時にはライセンスに関する質問をされるので、利用規約への同意やメールアドレス、Stardog から送られる情報をメールで受け取りたいかなどを聞かれるので、答える。
- この時点で 5820 番ポートでサーバが起動するが、このままの状態では Web ブラウザからアクセスしても何も表示されない。
- GUI が必要な場合は、別途 Stardog Studio をインストールする必要があるらしい https://www.stardog.com/studio/

- なおデフォルトのユーザ名/パスワードは admin/admin
- コマンドライン上でクエリを実行できればいい場合は、以下を参照のこと。
- データのロードをして DB を作るため、以下のようなコマンドを使用する。

```
./bin/stardog-admin db create -n myDB /path/to/some/data.ttl
```

- 名前付きグラフにロードしたい場合は、入力する RDF ファイルの名前の前に@<グラフの URL>のようにする。以下は http://examples.org の例

```
./bin/stardog-admin db create -n myDB @http://examples.org /path/to/some/data.ttl
```

- クエリはコマンドライン上から以下のように実行できる

```
./bin/stardog query myDB "SELECT DISTINCT ?s WHERE { ?s ?p ?o } LIMIT 10"
```

- Web API 越しに SPARQL を実行したい場合は http://localhost:5820/<DB 名>/query に対して GET リクエストを送る

第 4 章

Blazegraph

ウェブサイト

Quick start

本記事の対象バージョン：2.1.6 (2020/02/04 リリース)

ライセンス

- オープンソース (GPL2.0 ライセンス)

その他必要条件

- Java 9 以上

4.1 Installation

- Blazegraph のリポジトリから最新リリースの jar ファイルをダウンロードする (https://github.com/blazegraph/database/releases)。
- 以下は wget でダウンロードする例。

```
wget https://github.com/blazegraph/database/releases/download/BLAZEGRAPH_2_1_6_RC/
↪blazegraph.jar
```

- ダウンロードされた jar ファイルを、以下のコマンドで実行する
- 実行すると Web サーバが起動するので、http://localhost:9999/ でアクセスできる
- Go to http://<**自分のサーバの** IP>/blazegraph/ のような表示も出るが、こちらでもアクセス可能

```
java -server -Xmx4g -jar <downloaded_dir>/blazegraph.jar
```

- データをロードするにはブラウザ内で、Update タブを選択して以下のようなコマンドを入れて下の方の Update ボタンをクリックする。

```
load <file:///path/to/your.ttl>
```

- 名前付きグラフにする場合は、以下のように INTO GRAPH <グラフの URL>をつける

```
load <file:///path/to/your.ttl> INTO GRAPH  <http://examples.org>
```

- 実行すると、一番下に小さく `Running update...` と表示されるので完了するまで待つ。
- クエリを実行する場合は、Query タブに移動してクエリを入力後、Execute ボタンをクリックして実行できる。
  - Web API から実行したい場合は、http://localhost:9999/sparql に対して GET メソッドでリクエストを送ればよい。

第 5 章

AllegroGraph

ウェブサイト

Quick Start

本記事の対象バージョン：7.0.0

ライセンス

- Free, Developer, Enterprise の 3 形態あり。Free だと 5000,000 トリプルと 3 サーバまでの制限あり (https://allegrograph.com/allegrograph-editions/)

5.1 Installation

- 以下の URL に名前とメールアドレスを入力して、tar ファイルをダウンロード

```
https://franz.com/franz/agraph/ftp/pri/acl/ag/ag7.0.0/linuxamd64.64/agraph-7.0.0-
↪linuxamd64.64.tar.gz.lhtml?l=agfree
```

- tar ファイルを解凍後、解凍したフォルダに移動し、インストールしたいディレクトリを指定してインストール

```
cd agraph-7.0.0
sudo ./install-agraph /path/to/install/directory
```

- インストールの際、いくつか質問されるので答えていく。ユーザ名とパスワード以外はデフォルトで良さそう。
  - ユーザ名とパスワードに関しては、後でデータセットをロードする時に必要になる。
  - ユーザはデフォルトだと agraph になるが、シェルのユーザ名と同じにしておくと良い。
- インストールしたディレクトリに移動後、以下のコマンドでサーバを起動

```
sudo ./bin/agraph-control --config ./lib/agraph.cfg start
```

- ブラウザで、http://127.0.0.1:10035 にアクセスする。
- ユーザ名とパスワードを入力して、サインインする。
- リポジトリを好きな名前で作成する。
- コマンドラインに戻り、以下のコマンドでロードを行う。repository_name と path/to/dataset は適宜読み替える。（ブラウザ上の import RDF からロードすることも可能）

```
./bin/agtool load repository_name path/to/dataset
```

- グラフ名を指定したい場合は、 -g オプションを使用する

```
 ./bin/agtool load repository_name -g http://example.org path/to/dataset
```

- ロード完了後、ブラウザ上で queries のタブを選択し、クエリを入力して実行する。
  - Web API から SPARQL を実行したい場合は、 http://localhost:10035/repositories/<リポジトリ名> に対して POST で実行する（query パラメータにクエリの内容を含める）。GET で送るとエラーになるので注意。
- 実行中のサーバを止めたい場合は、以下のコマンドを実行する。

```
sudo ./bin/agraph-control --config ./lib/agraph.cfg stop
```

第 6 章

Fuseki

ウェブサイト

クイックスタート

本記事の対象バージョン：3.16.0 (2020/07/09 リリース)

ライセンスなど

- オープンソース（Apache 2.0 ライセンス）

必要なもの

- Java と Tomcat のインストール

6.1 Installation

- https://jena.apache.org/download/ から Fuseki を探してダウンロード
  - 例えば、wget で 3.16.0 をダウンロードする場合、以下のようにする

```
wget https://ftp.yz.yamagata-u.ac.jp/pub/network/apache/jena/binaries/apache-jena-
↪fuseki-3.16.0.tar.gz
```

- 解凍したフォルダ内で、以下のコマンドを実行するとサーバが立ち上がるので、Web ブラウザ上で `http://localhost:3030` からアクセスできるようになる。

```
./fuseki-server
```

- デフォルトでは localhost 以外からのアクセスが一部禁止されている（トップページは表示されるが、データセットのロードができない）ため、ホストの外からアクセスしたい場合は `./run/shiro.ini` の以下の行をコメントアウトする。

```
/$/** = localhostFilter
```

- データセットをロードするには、ブラウザからアクセスしたあと、manage datasets メニューの add new dataset タブから好きな名前と Dataset type を選んで Dataset を作成する。その後、dataset メニューに移動して upload files タブからファイルをアップロードできる。
- クエリの実行は dataset のメニューから、データセットごとに実行できる。
- Web API 越しに実行する場合は、http://localhost:3030/<データセット名>/query に GET リクエストを送ると実行できる

6.1.1 Tomcat を使う場合

- ダウンロードした tar ファイルを解凍後、中に入っている fuseki.war を Tomcat の webapps ディレクトリにコピーする
  - 例えば、Tomcat のインストールディレクトリが/opt/tomcat/ なら

```
cp ./fuseki.war /opt/tomcat/webapps/
```

- fuseki の実行に/etc/fuseki が必要らしいので、ディレクトリを作成してから tomcat のユーザに権限を付与する

```
sudo mkdir /etc/fuseki && sudo chgrp tomcat /etc/fuseki && sudo chown tomcat /etc/
↪fuseki
```

- これで、ブラウザから http://localhost:8080/fuseki でアクセスできるようになる。
  - デフォルトでは localhost 以外からのアクセスが一部禁止されている（トップページは表示されるが、データセットのロードができない）ため、ホストの外からアクセスしたい場合は /ets/fuseki/shiro.ini の以下の行をコメントアウトする。

```
/$/** = localhostFilter
```

6.1.2 コマンドラインからデータのロードを実行する場合

- tdbloader を含んだ apache-jena のパッケージをダウンロードする

```
wget [https://apache.cs.utah.edu/jena/binaries/apache-jena-3.16.0.zip](https://apache.
↪cs.utah.edu/jena/binaries/apache-jena-3.16.0.zip)
```

- zip ファイルの展開後、./bin/tdb2.tdbloader を使ってロードする。

- 名前付きグラフとしてロードする場合は、tdb2.tdbloader 実行時に–graph=グラフの URL のようにオプションを追加する

```
cd <apache-jena*.zip を解凍した場所>
mkdir tmp # いったん tmp にロード
./bin/tdb2.tdbloader --loc ./tmp <RDF データファイルのパス>
```

- ロードしたデータを fuseki サーバから参照したい場合は、ブラウザ上で manage datasets → add new dataset から好きな名前で（ここでは sample_dataset とする）データセットを作る。データセットタイプは Persistent（TDB2）にする。
- 新しいデータセットを作成後、以下のコマンドでデータセットのディレクトリに先ほど tdb2.tdbloader で作成したファイルを移動する

```
mv ./tmp/* <fuseki のインストール場所>/run/databases/sample_dataset/
```

- 一度 fuseki-server のプロセスを再起動すると、データセットがロードされた状態になるはず

第 7 章

RDF4j

ウェブサイト

本記事の対象バージョン：3.3.0

ライセンス

- オープンソース（Eclipse Distribution License (EDL), v1.0.）

必要なもの

- Java と Tomcat のインストール

7.1 Installation

7.1.1 Web サーバとして実行する場合

RDFJ Server and Workbench を使う

- まず、https://rdf4j.org/download/ から RDF4J 3.3.0 SDK (zip) をダウンロード
- zip ファイルを解凍後、war/rdf4j-*.war を Tomcat の webapps ディレクトリに以下にコピーする。
  - 例えば Tomcat のインストール先が/opt/tomcat なら以下のようなコマンドでコピーできる

```
cp rdf4j-*.war /opt/tomcat/webapps/
```

- ブラウザから、`http://localhost:8080/rdf4j-workbench` にアクセスするとトップページが表示される。
  - Repositories -> New repository から、ID とタイトルなどを入力してリポジトリを作成する。
  - リポジトリ作成後、Modify->Add からファイルをアップロードしてデータをインポートできる。

7.1.2 コンソールからデータをロードする場合

- zip を解凍したフォルダ内の bin/console.sh を実行すると CUI が起動する。
  - まずサーバに接続

```
> connect http://localhost:8085/rdf4j-server
Disconnecting from default data directory
Connected to http://localhost:8085/rdf4j-server
```

- 次にリポジトリを作成する。最初にリポジトリのタイプ（native, memory など）を選択してから、リポジトリ名など必要事項を入力する。デフォルトで良い場合は何も入力しない。

```
> create native
Please specify values for the following variables:
Repository ID [native]:
Repository title [Native store]:
Query Iteration Cache size [10000]:
Triple indexes [spoc,posc]:
EvaluationStrategyFactory [org.eclipse.rdf4j.query.algebra.evaluation.impl.
↪StrictEvaluationStrategyFactory]:
Repository created
```

- なお、すでにリポジトリを作成済みの場合は `open <repository_name>`のようにすると開くことができる）

- 最後に、load コマンドでロードを行う。into 以下は省略可能

```
> load <input_path> into <name_of_graph>
Loading data...
Data has been added to the repository (2565 ms)
```

7.1.3 Java プログラムから使用する場合

- TBA

第 8 章

rdfstore-js

ウェブサイト

本記事の対象バージョン：0.9.17 (2016/09/04 リリース)

ライセンス

- オープンソース（MIT ライセンス）

8.1 Installation

- npm をインストール

```
$ sudo apt install npm
```

- rdfstore-js で作業するためのディレクトリを作って移動

```
$ mkdir ./rdfstore-js
$ cd rdfstore-js
```

- npm を使って必要なパッケージをインストール

```
$ npm install nodejs
$ npm install rdfstore
```

- テキストエディタなどで、以下のようなファイルを作成する。ファイル名は test.js とする。
  - `/path/to/dataset.ttl` はロードしたい ttl ファイルのパスに変更する。

```
const rdfstore = require('rdfstore');
const fs = require('fs');
```

(次のページに続く)

(前のページからの続き)

```
rdfstore.create(function(err, store){
  const rdf = fs.createReadStream('/path/to/dataset.ttl');
  store.load('text/turtle', rdf, function(s,d){
    console.log(s,d);
    store.execute("SELECT * WHERE { ?s ?p ?o } LIMIT 10",
      function(success, results){
        console.log(success, results);
      });
  });
});
```

- ファイル作成後、以下のコマンドで実行する。
  - 大きめのファイルを扱う場合は、Node.js のメモリ制限に引っかかることがあるので --max-old-space-size を指定する（単位は MB）

```
$ node --max-old-space-size=4096 test.js
```

第 9 章

Neptune

9.1 Setup

前提として、AWS にアカウントがあることが必須となる.

AWS コンソールから、Neptune を起動する.

新しい VPN の作成を選べば、簡単に作成できる.

第 10 章

Oracle

ウェブサイト

- 参考 1 (RDF の利用)
- 参考 2 (PG の利用)
- 参考 3 (Oracle Cloud の利用)

バージョン：18.4.0 XE

ライセンス：Oracle Free Use Terms and Conditions (Permitted Features)

10.1 Installation

10.1.1 Docker 版を利用

Get Dockerfile to build Docker image of Oracle Database (needs 4GB memory).

```
$ git clone https://github.com/oracle/docker-images.git
$ cd docker-images/OracleDatabase/SingleInstance/dockerfiles/18.4.0/
$ docker build -t oracle/database:18.4.0-xe -f Dockerfile.xe .
```

Launch Oracle Database on a docker container.

```
$ docker run --name oracle -p 1521:1521 -e ORACLE_PWD=Welcome1 -v $HOME:/host-home␣
↪oracle/database:18.4.0-xe
```

Once you got the message below, the database is ready (you can quit with Ctl+C).

```
#########################
DATABASE IS READY TO USE!
#########################
```

Configure the database as a triplestore.

```
$ docker exec -it oracle sqlplus sys/Welcome1@XEPDB1 as sysdba @/host-home/icgc/
↪scripts/setup.sql
```

Create a user.

```
$ docker exec -it oracle sqlplus sys/Welcome1@XEPDB1 as sysdba @/host-home/icgc/
↪scripts/00_user.sql
```

第 11 章

PGX

本記事の対象バージョン

- 20.4.0

ライセンスなど

- OTN license

必要なもの

- Java のインストール

11.1 Installation

11.1.1 Ubuntu 18.04 の場合

- 2021/1 現在、公式に配布されているのは rpm パッケージのみなので、Ubuntu にインストールするためにまず deb パッケージ に変換してからインストールする。
- 参考：[https://phoenixnap.com/kb/install-rpm-packages-on-ubuntu]

```
sudo add-apt-repository universe
sudo apt-get update
sudo apt install alien
sudo alien --scripts oracle-graph-20.4.0.x86_64.rpm # --scripts オプションをつけないとインストール時に権限周りで失敗する
sudo dpkg -i oracle-graph_20.4.0-1_amd64.deb
```

- インストール後、標準では /opt/oracle/graph/ にインストールされる。とりあえずコマンドライン上で試すのであれば、bin/opg-jshell が試しやすい

```
sudo /opt/oracle/graph/bin/opg-jshell
```

- sudo systemctl start pgx のようにしてサーバプロセスとして起動することもできるが、その場合 /etc/oracle/graph/server.conf や /etc/oracle/graph/pgx.conf にセキュリティ関係の設定を適切に編集する必要があるほか、Oracle DB などを IdentityProvider として使用する必要があるため割愛。
- opg-jshell 内で PGQL を実行する場合は、例えば以下のようにする

```
opg-jshell> var G = session.readGraphWithProperties("/tmp/example.pgx.json") // 読み込みたい pgx 用の json を指定
G ==> PgxGraph[name=sample.pgx,N=554,E=1528,created=1612683135450]
opg-jshell> G.queryPgql("SELECT * MATCH (a)-[]->(b) LIMIT 10").print() // 任意のリレーションを最大 10 個取得して表示
```

第 12 章

Neo4j

ウェブサイト

本記事の対象バージョン

- 4.1.0 Community Edition

ライセンスなど

- オープンソース（Community Edition に限る。GPL v3）

必要なもの

- Java のインストール

参考：Manual

12.1 Installation

12.1.1 Ubuntu 18.04 の場合

参考：Manual

- Neo4j のパッケージリポジトリを追加する

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.com stable latest' | sudo tee -a /etc/apt/sources.list.
↪d/neo4j.list
sudo apt-get update
```

- apt でインストール

```
sudo apt-get install neo4j
```

- Neo4j をサービスとして起動する

```
sudo service neo4j start
```

- 停止する場合は以下のコマンド

```
sudo service neo4j stop
```

- localhost:7474 にアクセスすると Neo4j Browser が立ち上がるはず。
- デフォルトだとユーザ名とパスワードは両方とも neo4j になっているので、これでログイン（ログイン後にパスワードの変更を求められる）

12.1.2 NeoSemantics を使用する場合

参考：Tutorial

参考：Configuration

- NeoSemantics は RDF データを変換しつつ Neo4j にインポートできるようになる Neo4j プラグイン
- まず NeoSemantics の jar をダウンロード

```
wget https://github.com/neo4j-labs/neosemantics/releases/download/4.1.0.1/neosemantics-
↪4.1.0.1.jar
```

- これを Neo4j の plugins フォルダに移動

```
sudo mv neosemantics-4.1.0.1.jar /var/lib/neo4j/plugins/
```

- Neo4j の再起動

```
sudo service neo4j restart
```

- うまくいっていれば、Neo4j Browser 上で `call dbms.procedures()` というコマンドの実行結果の中に、n10s から始まるものが含まれているはず。
- ロードする前に config を初期化しておく必要があるので `n10s.graphconfig.init` を呼び出す。

```
CALL n10s.graphconfig.init();
```

- また、Resource の uri にユニーク制約を張っておく。

```
CREATE CONSTRAINT n10s_unique_uri ON (r:Resource)
ASSERT r.uri IS UNIQUE
```

- ロードする際にはプレフィックスを省略表記にするかなどをオプションで設定できる (参考：Reference)。
- 例えば、次のような形で指定可能

```
CALL n10s.graphconfig.init({
  handleVocabUris: 'MAP'
})
```

- あとは n10s.rdf.import.fetch でデータをロードする。リモートのデータをロードする場合は url の指定を http:// から始める。ローカルのデータをロードする場合は file:// から始めれば良い。以下は/home/user_name/file_name.nt をロードする場合の例。

```
CALL n10s.rdf.import.fetch(
  'file:///home/user_name/file_name.nt',
  'Turtle'
)
```

12.1.3 curl でクエリを実行する方法

実行時間を計測したい場合など、ブラウザを経由せず、curl 等を用いてコマンドラインからクエリを実行したほうがよい場合がある。

- Neo4j Browser や Neo4j Console を使用する場合、一応クエリごとの実行時間は表示してくれるが、これはあまり信用してはいけないらしい（以下の URL で don't rely on that exact timing と言われている）
  - https://neo4j.com/developer/neo4j-browser/
  - なお Enterprise 版だと、dbms.logs.query.*のようなオプションが用意されているらしい。
- コマンドラインのスクリプトから起動する場合など、curl でクエリを実行したい場合には curl の-d, --data オプションにクエリを指定して、/db/data/transaction/commit に対して実行すればよい。
  - 例えば、以下のようなスクリプトを手元に用意する。名前は仮に curl_cypher.sh とする。

```
query=$(cat $1 | tr -d '\n')
param="{ \"statements\": [ {\"statement\": \"$query\"}]}"
curl -u neo4j:neo4j -H 'Content-type: application/json;charset=utf-8' -d "$param"␣
↪http://localhost:7474/db/data/transaction/commit
```

- また、サンプルのクエリとして以下のようなファイルを、sample_query.cyp として保存する。

```
MATCH (n) RETURN n LIMIT 10
```

- その後、sh ./curl_cypher.sh sample_cypher.cyp のようにして実行することで、結果が JSON 形式で表示される。
  - 実行時間を計測したい場合は、time sh ./curl_cypher.sh sample_cypher.cyp のようにすればよい。

12.2 Configuration

12.2.1 外部から接続したい場合

- （以下を行うと、任意のホストからの接続を許可することになるので、セキュリティ上の問題が起きないよう注意すること）
- /etc/neo4j/neo4j.conf を開くと、以下のような記述があるので

```
#dbms.default_listen_address=0.0.0.0
```

- コメントアウトを解除する

```
dbms.default_listen_address=0.0.0.0
```

- Neo4j を再起動する

```
sudo service neo4j restart
```

第 13 章

ArangoDB

13.1 Installation

第 14 章

OrientDB

14.1 Installation

第 15 章

JanusGraph

15.1 Installation

第 16 章

TigerGraph

16.1 Installation

第 17 章

Summary: トリプルストア及びグラフデータベースに関する技術調査

17.1 背景

近年、RDF 関連のソフトウェア開発が進展し、世界中で様々な RDF ストア（トリプルストアと呼ばれる）が公開されてきた。よく用いられている実装の一例としては OpenLink Software Virtuoso があるが、その他にも多くの新しい実装が活発に開発されてきており、多くの実装について、性質や性能が自明ではない。また、グラフを基礎としたデータモデルを採用しているデータベースとしては、プロパティ・グラフモデルに基づくデータベース実装も出てきており、近年より活発に開発が行われている。トリプルストアに加えて、プロパティ・グラフモデルに基づくデータベース実装に言及するとき、グラフデータベースと呼ばれることが多い。

このように活発に開発が続けられているトリプルストア及びグラフデータベースの分野は、進展が早くまた製品も多様性に富むため、現状を把握するのは容易ではないが、各データベースの有用性を評価するために、標準準拠状況、性能の観点から継続的に調査を行うことが望ましい。こうした評価情報は、RDF 等の基盤技術を利用している研究機関において共有されるべき重要な情報である。

17.2 トリプルストアの各実装

17.2.1 Virtuoso

Virtuoso は、OpenLink Software 社の開発するデータベースエンジンである。Open-source 版と commercial 版がある。本調査では、open-source 版を用いている。RDBMS を基盤としているが、RDF も扱えるようになっている。もともと row-wise であったが、Virtuoso 7 からは column-wise を用いるように拡張された。graph identifier g と、(s, p, o) からなる 4 つ組 (quads) をストアする。インデックスは、<g, s, p, o>と<o, g, p, s>である。

多くの RDF ワークロードでは、削除が少なく、大量の読み込みを必要とするため、デフォルトのインデックススキームは、これらに最適化されている。このような状況で、この方式はスペースを大幅に節約し、より良い作業セットを実現する。通常、このレイアウトは、4 つのフルインデックスを使用したレイアウトの 60〜70 %の

スペースで済む。これが現実的ではない場合、インデックススキームはフルインデックスのみを持つべきである。つまり、各キーがクワッドの主キーのすべての列を保持する。これは、CREATE INDEX 文で DISTINCT NO PRIMARY KEY REF オプションが指定されていない場合に当てはまる。このような場合、削除されてもすべてのインデックスは厳密に同期される。

http://docs.openlinksw.com/virtuoso/rdfperfrdfscheme/

https://www.amazon.co.jp/RDF-Database-Systems-Triples-Processing/dp/0127999574

ウェブサイト

http://vos.openlinksw.com/owiki/wiki/VOS

バージョン

7.2.5.1 (2018/08/15)

ライセンス

GPLv2

その他必要条件

automake, libtool, gperf パッケージを apt でインストールする必要がある。

Installation

ソースを取得

```
git clone git@github.com:openlink/virtuoso-opensource.git
cd virtuoso-opensource
git checkout v7.2.5.1
```

コンパイル

```
./autogen.sh
./configure --prefix=/path/to/install/directory --with-readline
make
make install
```

ポート 1111 を既に使用していると、make の途中で失敗する。（テストに 1111 を使おうとするため）

configure に –with-port=1112 オプションを付けるなどすれば失敗を回避できるかもしれないが、1111 を空けておくのが確実。

起動

```
cd /path/to/install/directory/var/lib/virtuoso/db/
/path/to/install/directory/bin/virtuoso-t +wait
```

http://localhost:8890/sparql で SPARQL エンドポイントが起動していることを確認する。

データの取得

```
curl -LOR http://example.com/example.ttl
```

ロード

```
isql 1111 dba dba
SQL> DB.DBA.TTLP_MT(file_to_string_output('example.ttl'), '', 'http://example.com/
↪example.ttl', 0);
```

Configuration

デフォルトから設定を変更したい場合

- 設定ファイルの編集

```
vi /path/to/install/directory/var/lib/virtuoso/db/virtuoso.ini
```

問い合わせによって、virtuoso-temp.db のサイズが肥大化していくことがある。その場合、virtuoso.ini の中で TempAllocationPct(virtuoso.db に対して、何%まで許容するか) を設定する。参照

```
TempAllocationPct        = 100
```

Troubleshooting

./configure で OpenSSL に関するエラーが出ることがある。この場合は、libssl1.0-dev をインストールする。

```
sudo apt install -y libssl1.0-dev
```

Ubuntu 20.04 LTS では、パッケージ libssl1.0-dev が見つからないことがある。この場合は、/etc/apt/sources.list の末尾に以下を追加して apt update を行う。

```
deb http://security.ubuntu.com/ubuntu bionic-security main
```

```
sudo apt update
sudo apt install -y libssl1.0-dev
```

17.2.2 GraphDB

GraphDB は、Ontotext 社の開発したトリプルストアであり、ライセンスとしては Free 版と Standard 版、Enterprise 版が存在する。実装言語は Java のため、JVM の動作する基板上であれば使用可能である。

GraphDB Free Edition 9.3.1 (https://www.ontotext.com/products/graphdb/graphdb-free/) に関して、弊社内 Linux サーバへのインストールを行い、インストール、Web サーバの起動、GUI へのアクセス方法、SPARQL クエリを発行する方法を GitHub 内の docs/md/graphdb.md に記載しました。また、貴研究所より指定されたデータセットに関して、GraphDB でロードした場合に要する時間の調査を行い、環境情報とともにスプレッドシートに記載しました。

ウェブサイト

https://www.ontotext.com/products/graphdb/

更新履歴

http://graphdb.ontotext.com/documentation/free/release-notes.html

本記事の対象バージョン

9.3.1 (2020/06/18 リリース)

ライセンス

無料版が存在するが、同時に発行できるクエリは 2 つまでとの記述有り。有料ライセンスは Standard 版と Enterprice 版の二種

Installation

- GrahDB のフリーエディションを使用するため、サイト上でユーザ登録をする https://www.ontotext.com/products/graphdb/graphdb-free/
- ユーザ登録が完了すると、登録したメールアドレスにインストール用バイナリのダウンロード用リンクが送られてくる
- Download as a stand-alone server のリンクをクリックしてダウンロード（Desktop installation ではない方）
- ダウンロード後、zip を解凍してそのフォルダへ移動

```
$ cd path/to/unzipped/folder
```

- GraphDB をワークベンチモードで起動

```
$ sudo ./bin/graphdb
```

- ブラウザで localhost:7200 にアクセスすると、GraphDB のホーム画面が表示される
- 新しいリポジトリを作るため、左のメニューから、Setup->Repositories->Create new repository と選択する
- リポジトリの ID を入力し、最下部の Create をクリック
- ターミナル上で、以下のコマンドを実行して ttl データをインポート

- なお、loadrdf コマンドではインポートするグラフに名前をつけることができない模様。
- このため、グラフ名をつける必要がある場合はブラウザ上の http://localhost:7200/import から行う必要あり？

```
$ sudo ./bin/loadrdf -f -i <repository_name> -m parallel <path to dataset>
```

- インポートが完了したら、ブラウザに戻り左のメニューから SPARQL をクリックする
- リポジトリのリストが表示されるのでリポジトリを選択する
- SPARQL のエディタが表示されるので SPARQL を入力後、RUN で実行できる
- API から SPARQL を実行したい場合は、http://localhost:7200/repositories/<リポジトリ名> に GET リクエストを送ればよい

17.2.3 Stardog

Stardog 7.3.2 (https://www.stardog.com/get-started/) に関して、弊社内 Linux サーバへのインストールを行い、インストール方法を docs/md/stardog.md に記載しました。

Linux サーバへインストールを行った Stardog 7.3.2 に関して、データのロード時間を計測いたしました。

ウェブサイト

https://www.stardog.com/

Getting Started

https://www.stardog.com/docs/#_setting_path

本記事の対象バージョン

7.3.2（2020/07/01 リリース）

ライセンス

完全にフリーなライセンスは存在しないが、30 日分の体験版、1 年のアカデミックライセンスの体験版がある。

Installation

Stardog の最新版をダウンロードして解凍して移動

```
wget https://downloads.stardog.com/stardog/stardog-latest.zip
unzip stardog-latest.zip
cd stardog
```

Stardog のサーバを起動。

```
./bin/stardog-admin server start
```

- 初回起動時にはライセンスに関する質問をされるので、利用規約への同意やメールアドレス、Stardog から送られる情報をメールで受け取りたいかなどを聞かれるので、答える。
- この時点で 5820 番ポートでサーバが起動するが、このままの状態では Web ブラウザからアクセスしても何も表示されない。
- GUI が必要な場合は、別途 Stardog Studio をインストールする必要があるらしい https://www.stardog.com/studio/
- なおデフォルトのユーザ名/パスワードは admin/admin
- コマンドライン上でクエリを実行できればいい場合は、以下を参照のこと。
- データをロードして DB を作るため、以下のようなコマンドを使用する。

```
./bin/stardog-admin db create -n myDB /path/to/some/data.ttl
```

- 名前付きグラフにロードしたい場合は、入力する RDF ファイルの名前の前に@<グラフの URL>のようにする。以下は http://examples.org の例

```
./bin/stardog-admin db create -n myDB @http://examples.org /path/to/some/data.ttl
```

- クエリはコマンドライン上から以下のように実行できる

```
./bin/stardog query myDB "SELECT DISTINCT ?s WHERE { ?s ?p ?o } LIMIT 10"
```

- Web API 越しに SPARQL を実行したい場合は http://localhost:5820/<DB 名>/query に対して GET リクエストを送る

17.2.4 AllegroGraph

AllegroGraph は Franz 社の開発したマルチモデルデータベースであり、ドキュメントデータとトリプルデータの両方をサポートしている。ライセンスとしては Free,Developer, Enterprise の三種類が存在する。実装言語は Java, Python, Common Lisp である。

AllegroGraph Free Edition の 7.0.0（https://franz.com/agraph/downloads/）に関して、弊社内 Linux サーバへのインストールを行い、インストール方法、データセットのロード方法、クエリの発行方法を docs/md/allegrograph.md に記載しました。

6 月に弊社 Linux サーバへインストールを行った AllegroGraph 7.0.0 に関して、データのロード時間を計測いたしました。ただし、フリー版のライセンスでは５００万トリプル以上のデータをロードすることができず、対象データ全体を扱えなかったため、参考情報として行数を先頭から５００万行分に限定したファイルを使用してロード時間を計測させていただきました。

貴研究所担当者様より頂いた AllegroGraph Enterprise Edition のライセンスを利用し、弊社で Enterprise Edition を用いたロード時間の測定を行いました。この際、AllegroGraph 開発者より入力ファイルの形式が turtle ファイルか ntriples ファイルかでデータのロードの並列性能に差があるという連絡を受けたため、入力ファイルを turtle から ntriples に変換したものも用意し、ntriples でのみ並列化の効果があることを確認しました。

ウェブサイト

https://allegrograph.com/products/allegrograph/

Quick Start

https://franz.com/agraph/support/documentation/current/agraph-quick-start.html

本記事の対象バージョン

7.0.0

ライセンス

Free, Developer, Enterprise の 3 形態あり。Free だと 5000,000 トリプルと 3 サーバまでの制限あり (https://allegrograph.com/allegrograph-editions/)

Installation

- 以下の URL に名前とメールアドレスを入力して、tar ファイルをダウンロード

https://franz.com/franz/agraph/ftp/pri/acl/ag/ag7.0.0/linuxamd64.64/agraph-7.0.0-linuxamd64.64.tar.gz.lhtml?l=agfree

- tar ファイルを解凍後、解凍したフォルダに移動し、インストールしたいディレクトリを指定してインストール

```
cd agraph-7.0.0
sudo ./install-agraph /path/to/install/directory
```

- インストールの際、いくつか質問されるので答えていく。ユーザ名とパスワード以外はデフォルトで良さそう。
- ユーザ名とパスワードに関しては、後でデータセットをロードする時に必要になる。
- ユーザはデフォルトだと agraph になるが、シェルのユーザ名と同じにしておくと良い。
- インストールしたディレクトリに移動後、以下のコマンドでサーバを起動

```
sudo ./bin/agraph-control --config ./lib/agraph.cfg start
```

- ブラウザで、`http://127.0.0.1:10035` にアクセスする。
- ユーザ名とパスワードを入力して、サインインする。

- リポジトリを好きな名前で作成する。
- コマンドラインに戻り、以下のコマンドでロードを行う。repository_name と path/to/dataset は適宜読み替える。（ブラウザ上の import RDF からロードすることも可能）

```
./bin/agtool load repository_name path/to/dataset
```

- グラフ名を指定したい場合は、 -g オプションを使用する

```
./bin/agtool load repository_name -g http://example.org path/to/dataset
```

- ロード完了後、ブラウザ上で queries のタブを選択し、クエリを入力して実行する。
- Web API から SPARQL を実行したい場合は、 http://localhost:10035/repositories/<リポジトリ名> に対して POST で実行する（query パラメータにクエリの内容を含める）。GET で送るとエラーになるので注意。
- 実行中のサーバを止めたい場合は、以下のコマンドを実行する。

```
sudo ./bin/agraph-control --config ./lib/agraph.cfg stop
```

17.2.5 Blazegraph

Blazegraph は SYSTAP 社が開発したオープンソースのトリプルストアであり、Wikidata の SPARQL エンドポイントとして採用されている。実装言語は Java であり、SPARQL に加えて Gremlin API もサポートしている。

BlazeGraph 2.1.6 に関して、弊社内 Linux サーバへのインストールを行い、インストール方法、データセットのロード方法、クエリの発行方法を docs/md/blazegraph.md に記載しました。また、データのロード時間の計測を行いました。

BlazeGraph 2.1.6 に関して、弊社内 Linux サーバへのインストールを行い、インストール方法、データセットのロード方法、クエリの発行方法を docs/md/blazegraph.md に記載しました。また、データのロード時間の計測を行いました。

ウェブサイト

https://blazegraph.com/

Quick start

https://github.com/blazegraph/database/wiki/Quick_Start

本記事の対象バージョン：2.1.6 (2020/02/04 リリース)

ライセンス

オープンソース (GPL2.0 ライセンス)

その他必要条件

Java 9 以上

Installation

- Blazegraph のリポジトリから最新リリースの jar ファイルをダウンロードする (https://github.com/blazegraph/database/releases)。
- 以下は wget でダウンロードする例。

```
wget https://github.com/blazegraph/database/releases/download/BLAZEGRAPH_2_1_6_RC/
↪blazegraph.jar
```

- ダウンロードされた jar ファイルを、以下のコマンドで実行する
- 実行すると Web サーバが起動するので、http://localhost:9999/ でアクセスできる
- `Go to http://<`**自分のサーバの** `IP>/blazegraph/` のような表示も出るが、こちらでもアクセス可能

```
java -server -Xmx4g -jar <downloaded_dir>/blazegraph.jar
```

- データをロードするにはブラウザ内で、Update タブを選択して以下のようなコマンドを入れて下の方の Update ボタンをクリックする。

```
load <file:///path/to/your.ttl>
```

- 名前付きグラフにする場合は、以下のように INTO GRAPH <グラフの URL>をつける

```
load <file:///path/to/your.ttl> INTO GRAPH  <http://examples.org>
```

- 実行すると、一番下に小さく `Running update...` と表示されるので完了するまで待つ。
- クエリを実行する場合は、Query タブに移動してクエリを入力後、Execute ボタンをクリックして実行できる。
- Web API から実行したい場合は、http://localhost:9999/sparql に対して GET メソッドでリクエストを送ればよい。

17.2.6 Apache Jena Fuseki

Apache Jena Fuseki は Apache ソフトウェア財団のコミュニティによって開発されたトリプルストアである。より正確には、Apache Jena がトリプルストアとしての機能の核となる部分であり、Fuseki は http のインターフェース部分を指す。実装言語は Java であり、単独の Web サーバとしても実行できるほか、Tomcat などの Servlet としても使用することができる。オープンソースであり、ライセンスは Apache ライセンスを採用している。

Apache Jena Fuseki 3.16.0 に関して、弊社内 Linux サーバへのインストールを行い、インストール方法、データセットのロード方法、クエリの発行方法を docs/md/fuseki.md に記載しました。ロード時間に関しては、ブラウザ経由でのアップロードに関しては実験いたしましたが、他のトリプルストアに対しての実験の公平性のためコマンドラインからのロードを追って調査する予定です。

先月に弊社 Linux サーバへインストールを行った Apache Jena Fuseki 3.16.0 に関して、コマンドライン経由でのデータのロードを行う方法をドキュメントに追記した上で、ロード時間の計測を行いました。

ウェブサイト

https://jena.apache.org/documentation/fuseki2/

クイックスタート

https://jena.apache.org/documentation/fuseki2/fuseki-quick-start.html

本記事の対象バージョン

3.16.0 (2020/07/09 リリース)

ライセンスなど

- オープンソース（Apache 2.0 ライセンス）

必要なもの

- Java と Tomcat のインストール

Installation

- https://jena.apache.org/download/ から Fuseki を探してダウンロード
- 例えば、wget で 3.16.0 をダウンロードする場合、以下のようにする

```
wget https://ftp.yz.yamagata-u.ac.jp/pub/network/apache/jena/binaries/apache-jena-
↪fuseki-3.16.0.tar.gz
```

- 解凍したフォルダ内で、以下のコマンドを実行するとサーバが立ち上がるので、Web ブラウザ上で `http://localhost:3030` からアクセスできるようになる。

```
./fuseki-server
```

- デフォルトでは localhost 以外からのアクセスが一部禁止されている（トップページは表示されるが、データセットのロードができない）ため、ホストの外からアクセスしたい場合は `./run/shiro.ini` の以下の行をコメントアウトする。

```
/$/** = localhostFilter
```

- データセットをロードするには、ブラウザからアクセスしたあと、manage datasets メニューの add new dataset タブから好きな名前と Dataset type を選んで Dataset を作成する。その後、dataset メニューに移動して upload files タブからファイルをアップロードできる。
- クエリの実行は dataset のメニューから、データセットごとに実行できる。
- Web API 越しに実行する場合は、http://localhost:3030/<データセット名>/query に GET リクエストを送ると実行できる

Tomcat を使う場合

- ダウンロードした tar ファイルを解凍後、中に入っている fuseki.war を Tomcat の webapps ディレクトリにコピーする
- 例えば、Tomcat のインストールディレクトリが/opt/tomcat/ なら

```
cp ./fuseki.war /opt/tomcat/webapps/
```

- fuseki の実行に/etc/fuseki が必要らしいので、ディレクトリを作成してから tomcat のユーザに権限を付与する

```
sudo mkdir /etc/fuseki && sudo chgrp tomcat /etc/fuseki && sudo chown tomcat /etc/
↪fuseki
```

- これで、ブラウザから http://localhost:8080/fuseki でアクセスできるようになる。
- デフォルトでは localhost 以外からのアクセスが一部禁止されている（トップページは表示されるが、データセットのロードができない）ため、ホストの外からアクセスしたい場合は /ets/fuseki/shiro.ini の以下の行をコメントアウトする。

```
/$/** = localhostFilter
```

コマンドラインからデータのロードを実行する場合

- tdbloader を含んだ apache-jena のパッケージをダウンロードする

```
wget [https://apache.cs.utah.edu/jena/binaries/apache-jena-3.16.0.zip](https://apache.
↪cs.utah.edu/jena/binaries/apache-jena-3.16.0.zip)
```

- zip ファイルの展開後、./bin/tdb2.tdbloader を使ってロードする。
- 名前付きグラフとしてロードする場合は、tdb2.tdbloader 実行時に–graph=グラフの URL のようにオプションを追加する

```
cd <apache-jena*.zip を解凍した場所>
mkdir tmp # いったん tmp にロード
./bin/tdb2.tdbloader --loc ./tmp <RDF データファイルのパス>
```

- ロードしたデータを fuseki サーバから参照したい場合は、ブラウザ上で manage datasets → add new dataset から好きな名前で（ここでは sample_dataset とする）データセットを作る。データセットタイプは Persistent（TDB2）にする。
- 新しいデータセットを作成後、以下のコマンドでデータセットのディレクトリに先ほど tdb2.tdbloader で作成したファイルを移動する

```
mv ./tmp/* <fuseki のインストール場所>/run/databases/sample_dataset/
```

- 一度 fuseki-server のプロセスを再起動すると、データセットがロードされた状態になる

17.2.7 RDF4j

RDF4j は Eclipse 財団の支援するコミュニティによって開発されたトリプルストアであり、実装言語は Java である。３条項 BSD ライセンスを採用したオープンソースプロジェクトである。

RDF4j 3.3.0 に関して、弊社内 Linux サーバへのインストールを行い、インストール方法、データセットのロード方法を docs/md/fuseki.md に記載しました。ロード時間に関しては、Fuseki と同様ブラウザ経由でのアップロードに関しては実験いたしました。

RDF4J 3.3.0 に関してもコマンドライン経由でのデータのロードを試みましたが、実行時間の関係でご指定いただいた約 3,000,000 トリプルの RDF データセット全体をロードすることはできなかったため、一旦約 100,000 トリプルのデータセットに制限してロード時間を計測させていただきました。

ウェブサイト

https://rdf4j.org/

本記事の対象バージョン

3.3.0

ライセンス

オープンソース（Eclipse Distribution License (EDL), v1.0.）

必要なもの

Java と Tomcat のインストール

Installation

Web サーバとして実行する場合

RDFJ Server and Workbench を使う

- まず、https://rdf4j.org/download/ から RDF4J 3.3.0 SDK (zip) をダウンロード

- zip ファイルを解凍後、war/rdf4j-*.war を Tomcat の webapps ディレクトリに以下にコピーする。
- 例えば Tomcat のインストール先が/opt/tomcat なら以下のようなコマンドでコピーできる

```
cp rdf4j-*.war /opt/tomcat/webapps/
```

- ブラウザから、http://localhost:8080/rdf4j-workbench にアクセスするとトップページが表示される。
- Repositories -> New repository から、ID とタイトルなどを入力してリポジトリを作成する。
- リポジトリ作成後、Modify->Add からファイルをアップロードしてデータをインポートできる。

コンソールからデータをロードする場合

- zip を解凍したフォルダ内の bin/console.sh を実行すると CUI が起動する。
- まずサーバに接続

```
> connect http://localhost:8085/rdf4j-server
Disconnecting from default data directory
Connected to http://localhost:8085/rdf4j-server
```

- 次にリポジトリを作成する。最初にリポジトリのタイプ（native, memory など）を選択してから、リポジトリ名など必要事項を入力する。デフォルトで良い場合は何も入力しない。

```
> create native
Please specify values for the following variables:
Repository ID [native]:
Repository title [Native store]:
Query Iteration Cache size [10000]:
Triple indexes [spoc,posc]:
EvaluationStrategyFactory [org.eclipse.rdf4j.query.algebra.evaluation.impl.
↪StrictEvaluationStrategyFactory]:
Repository created
```

- なお、すでにリポジトリを作成済みの場合は open <repository_name>のようにすると開くことができる）
- 最後に、load コマンドでロードを行う。into 以下は省略可能

```
> load <input_path> into <name_of_graph>
Loading data...
Data has been added to the repository (2565 ms)
```

Java プログラムから使用することも可能であるが、これについては実施していない。

17.2.8 rdfstore-js

rdfstore-js 0.9.17（https://github.com/antoniogarrote/rdfstore-js）に関して、弊社内 Linux サーバへのインストールを行い、インストール方法と Node.js からの実行方法を docs/md/rdfstore-js.md に記載しました。

ウェブサイト

https://github.com/antoniogarrote/rdfstore-js

本記事の対象バージョン

0.9.17 (2016/09/04 リリース)

ライセンス

オープンソース（MIT ライセンス）

Installation

- npm をインストール

```
$ sudo apt install npm
```

- rdfstore-js で作業するためのディレクトリを作って移動

```
$ mkdir ./rdfstore-js
$ cd rdfstore-js
```

- npm を使って必要なパッケージをインストール

```
$ npm install nodejs
$ npm install rdfstore
```

- テキストエディタなどで、以下のようなファイルを作成する。ファイル名は test.js とする。
- /path/to/dataset.ttl はロードしたい ttl ファイルのパスに変更する。

```
const rdfstore = require('rdfstore');
const fs = require('fs');

rdfstore.create(function(err, store){
  const rdf = fs.createReadStream('/path/to/dataset.ttl');
  store.load('text/turtle', rdf, function(s,d){
    console.log(s,d);
    store.execute("SELECT * WHERE { ?s ?p ?o } LIMIT 10",
      function(success, results){
        console.log(success, results);
```

(次のページに続く)

(前のページからの続き)

```
      });
  });
});
```

- ファイル作成後、以下のコマンドで実行する。
- 大きめのファイルを扱う場合は、Node.js のメモリ制限に引っかかることがあるので `--max-old-space-size` を指定する（単位は MB）

```
$ node --max-old-space-size=4096 test.js
```

17.2.9 Neptune

前提として、AWS にアカウントがあることが必須となる.

AWS コンソールから、Neptune を起動する.

新しい VPN の作成を選べば、簡単に作成できる.

17.3 グラフデータベースの各実装

17.3.1 Neo4j

Neo4j は Neo4j 社によって開発されたプロパティグラフを扱うグラフ DB であり、クエリ言語として Cypher を利用することができる。ライセンスとしてはオープンソースの Community Edition に加え、有償の Enterprise 版やクラウド上での利用を想定した Aura が存在する。実装は Java 言語によって行われている。

Neo4j ＋ Neosemantics の使い方に関するドキュメントの作成

トリプルストアと異なるデータモデルであるプロパティグラフを扱うデータベースである Neo4j に対し、RDF データをプロパティグラフに変換しながらロードを行うプラグインである Neosemantics を利用する方法に関し、インデックスの張り方などを含めた調査およびドキュメントの作成を行いました。また、既存の各種トリプルストアに関するドキュメントに関しても、API を通したクエリの実行方法や、名前付きとしてのロード方法などを追記いたしました。

Neo4j のクエリ実行時間を CUI から計測する方法に関する記載

Neo4j に対して、curl コマンドを使用してコマンドラインからクエリを実行する方法に関して、docs/md/neo4j.md に追記させていただきました。

ウェブサイト

https://neo4j.com/

本記事の対象バージョン

4.1.0 Community Edition

ライセンスなど

オープンソース（Community Edition に限る。GPL v3）

必要なもの

Java のインストール

参考

Manual

https://neo4j.com/docs/operations-manual/current/installation/

Installation

Ubuntu 18.04 の場合

参考

Manual

https://neo4j.com/docs/operations-manual/current/installation/linux/debian/

- Neo4j のパッケージリポジトリを追加する

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.com stable latest' | sudo tee -a /etc/apt/sources.list.
↪d/neo4j.list
sudo apt-get update
```

- apt でインストール

```
sudo apt-get install neo4j
```

- Neo4j をサービスとして起動する

```
sudo service neo4j start
```

- 停止する場合は以下のコマンド

```
sudo service neo4j stop
```

- localhost:7474 にアクセスすると Neo4j Browser が立ち上がるはず。

- デフォルトだとユーザ名とパスワードは両方とも neo4j になっているので、これでログイン（ログイン後にパスワードの変更を求められる）

NeoSemantics を使用する場合

参考：Tutorial https://neo4j.com/labs/neosemantics/tutorial/

参考：Configuration https://neo4j.com/docs/labs/nsmntx/current/config/

- NeoSemantics は RDF データを変換しつつ Neo4j にインポートできるようになる Neo4j プラグイン
- まず NeoSemantics の jar をダウンロード

```
wget https://github.com/neo4j-labs/neosemantics/releases/download/4.1.0.1/neosemantics-
↪4.1.0.1.jar
```

- これを Neo4j の plugins フォルダに移動

```
sudo mv neosemantics-4.1.0.1.jar /var/lib/neo4j/plugins/
```

- Neo4j の再起動

```
sudo service neo4j restart
```

- うまくいっていれば、Neo4j Browser 上で `call dbms.procedures()` というコマンドの実行結果の中に、n10s から始まるものが含まれているはず。
- ロードする前に config を初期化しておく必要があるので `n10s.graphconfig.init` を呼び出す。

```
CALL n10s.graphconfig.init();
```

- また、Resource の uri にユニーク制約を張っておく。

```
CREATE CONSTRAINT n10s_unique_uri ON (r:Resource)
ASSERT r.uri IS UNIQUE
```

- ロードする際にはプレフィックスを省略表記にするかなどをオプションで設定できる (参考：Reference)。
- 例えば、次のような形で指定可能

```
CALL n10s.graphconfig.init({
  handleVocabUris: 'MAP'
})
```

- あとは `n10s.rdf.import.fetch` でデータをロードする。リモートのデータをロードする場合は url の指定を http:// から始める。ローカルのデータをロードする場合は file:// から始めれば良い。以下は/home/user_name/file_name.nt をロードする場合の例。

```
CALL n10s.rdf.import.fetch(
  'file:///home/user_name/file_name.nt',
  'Turtle'
)
```

curl でクエリを実行する方法

実行時間を計測したい場合など、ブラウザを経由せず、curl 等を用いてコマンドラインからクエリを実行したほうがよい場合がある。

- Neo4j Browser や Neo4j Console を使用する場合、一応クエリごとの実行時間は表示してくれるが、これはあまり信用してはいけないらしい（以下の URL で don't rely on that exact timing と言われている）
- https://neo4j.com/developer/neo4j-browser/
- なお Enterprise 版だと、`dbms.logs.query.*`のようなオプションが用意されているらしい。
- コマンドラインのスクリプトから起動する場合など、curl でクエリを実行したい場合には curl の`-d, --data` オプションにクエリを指定して、`/db/data/transaction/commit` に対して実行すればよい。
- 例えば、以下のようなスクリプトを手元に用意する。名前は仮に `curl_cypher.sh` とする。

```
query=$(cat $1 | tr -d '\n')
param="{ \"statements\": [ {\"statement\": \"$query\"}]}"
curl -u neo4j:neo4j -H 'Content-type: application/json;charset=utf-8' -d "$param"␣
↪http://localhost:7474/db/data/transaction/commit
```

- また、サンプルのクエリとして以下のようなファイルを、`sample_query.cyp` として保存する。

```
MATCH (n) RETURN n LIMIT 10
```

- その後、`sh ./curl_cypher.sh sample_cypher.cyp` のようにして実行することで、結果が JSON 形式で表示される。
- 実行時間を計測したい場合は、`time sh ./curl_cypher.sh sample_cypher.cyp` のようにすればよい。

Configuration

外部から接続したい場合

以下を行うと、任意のホストからの接続を許可することになるので、セキュリティ上の問題が起きないよう注意すること

/etc/neo4j/neo4j.conf を開くと、以下のような記述があるので

```
##dbms.default_listen_address=0.0.0.0
```

コメントアウトを解除する

```
dbms.default_listen_address=0.0.0.0
```

Neo4j を再起動する

```
sudo service neo4j restart
```

17.3.2 Oracle Graph Server and Client (Oracle Labs PGX)

本記事の対象バージョン

20.4.0

ライセンスなど

OTN license

https://www.oracle.com/downloads/licenses/standard-license.html

必要なもの

Java のインストール

Installation

Ubuntu 18.04 の場合

- 2021/1 現在、公式に配布されているのは rpm パッケージのみなので、Ubuntu にインストールするためにまず deb パッケージ に変換してからインストールする。
- 参考：[https://phoenixnap.com/kb/install-rpm-packages-on-ubuntu]

```
sudo add-apt-repository universe
sudo apt-get update
sudo apt install alien
sudo alien --scripts oracle-graph-20.4.0.x86_64.rpm # --scripts オプションをつけないとインストール時に権限周りで失敗する
sudo dpkg -i oracle-graph_20.4.0-1_amd64.deb
```

- インストール後、標準では /opt/oracle/graph/ にインストールされる。とりあえずコマンドライン上で試すのであれば、bin/opg-jshell が試しやすい

```
sudo /opt/oracle/graph/bin/opg-jshell
```

- sudo systemctl start pgx のようにしてサーバプロセスとして起動することもできるが、その場合 /etc/oracle/graph/server.conf や /etc/oracle/graph/pgx.conf にセキュリティ関係の設定を適切に編集する必要があるほか、Oracle DB などを IdentityProvider として使用する必要があるため割愛。
- opg-jshell 内で PGQL を実行する場合は、例えば以下のようにする

```
opg-jshell> var G = session.readGraphWithProperties("/tmp/example.pgx.json") // 読み込みたい pgx 用の json を指定
G ==> PgxGraph[name=sample.pgx,N=554,E=1528,created=1612683135450]
opg-jshell> G.queryPgql("SELECT * MATCH (a)-[]->(b) LIMIT 10").print() // 任意のリレーションを最大 10 個取得して表示
```

また、クエリの実行時間計測は以下のコードで行える。なお、query 変数には事前にクエリの文字列を代入しておくものとする。

```
var start = System.currentTimeMillis(); G.queryPgql(query);
var end = System.currentTimeMillis(); end - start
```

17.3.3 ArangoDB

ArangoDB は ArangoDB GmbH が開発するオープンソースのマルチモデル DB であり、キーバリュー型のデータ、ドキュメントデータ、グラフデータを同じ DB エンジン上で扱うことを可能とする。クエリ言語は AQL（ArangoDB Query Languade）と呼ばれ、異なる 3 種のデータモデルに対し、共通の言語でクエリを発行できるようになっている。ライセンスはオープンソースの Community に加え、並列実行などの機能を追加した Enterprise、フルマネージド版の Oasis が存在している。実装言語としては C++ が用いられている。

17.4 データセット

パフォーマンステスト用のデータセットとして、NCBI taxonomy data を ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz を用いた。

https://graphdb.dbcls.jp/benchmark/taxonomy_2019-05-01.ttl (1.2GB, 30M triples)

https://graphdb.dbcls.jp/benchmark/taxonomy_2019-05-01.nt (3.8GB, 30M triples)

https://graphdb.dbcls.jp/benchmark/taxonomy_2019-05-01.pg (935MB, 15M nodes, 20M edges)

https://graphdb.dbcls.jp/benchmark/taxon_40674.nt (907KB, 166894 triples)

また、Oracle Labs PGX に同等のデータをロードするため、ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

を変換する Ruby スクリプト taxdump2pg.rb を作成した。本スクリプトは、https://github.com/dbcls/rdfsummit/blob/master/taxdump2owl/taxdump2owl.rb を元に作成しており、PG tools (https://pg-format.readthedocs.io/en/latest/) で扱うことのできる pg フォーマットで taxonomy データを作成する。さらにこの pg フォーマットから。pg2pgx コマンドを使用することで、Oracle Labs PGX にロード可能なデータ形式への変換を行った。

17.5 クエリ

17.5.1 SPARQL クエリ

count_taxa

```
PREFIX taxon: <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
SELECT (COUNT(?taxid) AS ?count)
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  ?taxid a taxon:Taxon .
}
```

count_classes

```
SELECT (COUNT(?instance) AS ?count) ?class
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  ?instance a ?class .
}
GROUP BY ?class
ORDER BY DESC(?count)
```

count_from_graph

```
SELECT (COUNT(*) AS ?count)
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  ?s ?p ?o
}
```

count_in_graph

```
SELECT (COUNT(*) AS ?count)
WHERE {
  GRAPH <http://ddbj.nig.ac.jp/ontologies/taxonomy/> {
    ?s ?p ?o
  }
}
```

count_in_taxon

```
PREFIX taxid: <http://identifiers.org/taxonomy/>
SELECT (COUNT(DISTINCT ?taxid) AS ?count)
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  ?taxid rdfs:subClassOf* taxid:9443 .
}
```

count_in_taxon_label

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT (COUNT(DISTINCT ?taxid) AS ?count) ?taxon
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  ?taxid rdfs:subClassOf* ?taxon .
  ?taxon rdfs:label "Primates" .
}
GROUP BY ?taxon
```

get_hierarchy

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX taxid: <http://identifiers.org/taxonomy/>
PREFIX taxon: <http://ddbj.nig.ac.jp/ontologies/taxonomy/>

SELECT ?order ?order_name ?family ?family_name ?species ?name
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  ?species taxon:rank taxon:Species ;
      rdfs:label ?name ;
      rdfs:subClassOf+ taxid:40674 ;
      rdfs:subClassOf+ ?family ;
      rdfs:subClassOf+ ?order .
  ?family taxon:rank taxon:Family ;
      rdfs:label ?family_name .
  ?order taxon:rank taxon:Order ;
      rdfs:label ?order_name .
}
ORDER BY ?order_name ?family_name ?name
```

filter_regex

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT *
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
```

(次のページに続く)

(前のページからの続き)

```
WHERE {
  ?taxid rdfs:label ?label .
  FILTER regex(?label, "Homo ")
}
```

common_ancestor

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX taxid: <http://identifiers.org/taxonomy/>

SELECT ?ancestor ?p ?o
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  ?ancestor ?p ?o .
  ?tax1 rdfs:subClassOf ?ancestor .
  ?tax2 rdfs:subClassOf ?ancestor .
  taxid:9606 rdfs:subClassOf* ?tax1 .
  taxid:511145 rdfs:subClassOf* ?tax2 .
  FILTER(?tax1 != ?tax2)
}
```

taxon_info

```
PREFIX taxid: <http://identifiers.org/taxonomy/>

SELECT *
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  taxid:9606 ?p ?o .
}
```

taxon_info_ordered

```
PREFIX taxid: <http://identifiers.org/taxonomy/>

SELECT ?p ?o
FROM <http://ddbj.nig.ac.jp/ontologies/taxonomy/>
WHERE {
  taxid:9606 ?p ?o .
}
ORDER BY ?p ?o
```

17.5.2 Cypher クエリ

count_taxa

```
MATCH (n:Taxon) RETURN COUNT(n)
```

count_classes

```
MATCH (n:Resource) RETURN DISTINCT count(labels(n)), labels(n)
```

count_from_graph および count_in_graph（Neo4j + Neosemantics では名前付きグラフとしてロードされないため、今回は同一クエリとして計測）

```
MATCH (n)-[r]->() RETURN COUNT(r)
```

count_in_taxon

```
MATCH (tax:Resource)-[r:subClassOf*..]->(n2:Resource
  {uri:'http://identifiers.org/taxonomy/9443'})
  RETURN DISTINCT COUNT(tax)
```

count_in_taxon_label

```
MATCH (taxid:Resource)-[rdfs:subClassOf*..]->(:Resource {label:'Primates'})
RETURN COUNT(DISTINCT(taxid))
```

get_hierarchy

```
MATCH
  (species:Resource)-[:subClassOf*1..]->(:Resource
     {uri:'http://identifiers.org/taxonomy/40674'}),
  (species:Resource)-[:rank]->(taxon:Resource
     {uri:'http://ddbj.nig.ac.jp/ontologies/taxonomy/Species'})

MATCH
  (species)-[:subClassOf*1..]->(family:Resource)-[:rank]->
    (:Resource {uri:'http://ddbj.nig.ac.jp/ontologies/taxonomy/Family'})
MATCH
  (species)-[:subClassOf*1..]->(order:Resource)-[:rank]->(:Resource
    {uri:'http://ddbj.nig.ac.jp/ontologies/taxonomy/Order'})
RETURN
  order.uri, order.label, family.uri, family.label, species.uri, species.label
```

filter_regex

```
MATCH (taxid:Resource)
WHERE taxid.label =~ '.*Homo .*'
RETURN taxid
```

common_ancestor

```
MATCH (tax9606:Resource {uri:'http://identifiers.org/taxonomy/9606'}),
      (tax9606)-[:subClassOf*1..]->(ancestor:Resource)
MATCH
      (tax511145:Resource {uri:'http://identifiers.org/taxonomy/511145'}),
      (ancestor)<-[:subClassOf*1..]-(tax511145)
MATCH
      (ancestor:Resource)-[p]->(o)
RETURN ancestor, p, o
```

taxon_info

```
MATCH (n:Resource
{uri:'http://identifiers.org/taxonomy/9606'})-[r]->(n2:Resource)
RETURN r, n2
```

taxon_info_ordered

```
MATCH (n:Resource
{uri:'http://identifiers.org/taxonomy/9606'})-[r]->(n2:Resource)
RETURN r, n2 ORDER BY r.uri, n2.uri
```

17.5.3 PGQL クエリ

Oracle Labs PGX で性能測定を行うために記述したクエリ群を以下に示す。

common_ancestor.pgql

```
SELECT * MATCH (tax9606)-/:subClassOf+/->(ancestor)<-/:subClassOf+/-(tax511145) WHERE␣
↪id(tax9606) = '9606' AND id(tax511145) = '511145'
```

count_classes.pgql

```
SELECT DISTINCT COUNT(n.label_), n.label_ MATCH (n) GROUP BY n.label_
```

count_in_graph.pgql

```
SELECT COUNT(r) MATCH (n)-[r]->()
```

count_in_taxon_label.pgql

```
SELECT COUNT(DISTINCT(taxid)) MATCH (taxid)-/:subClassOf+/->(r) WHERE r.label =
↪'Primates'
```

count_in_taxon.pgql

```
SELECT COUNT(tax) MATCH (tax)-/:subClassOf*/->(n2) WHERE id(n2) = '9443'
```

count_taxa.pgql

```
SELECT COUNT(n) MATCH (n) WHERE n.label_ = 'Taxon'
```

filter_regex.pgql

```
SELECT id(n) MATCH (n) WHERE JAVA_REGEXP_LIKE( n.label, '.*Homo .*' )
```

get_hierarchy.pgql

```
SELECT order.label, id(family), family.label, id(species), species.label MATCH␣
↪(species)-/:subClassOf+/->(ancestor), (species)-/:subClassOf+/->(family), (species)-/
↪:subClassOf+/->(order) WHERE species.rank = 'Species' AND id(ancestor) = '40674' AND␣
↪family.rank = 'Family' AND order.rank = 'Order'
```

taxon_info_ordered.pgql

```
SELECT * MATCH (n)-[r]->(n2) WHERE id(n) = '9606' ORDER BY id(r), id(n2)
```

taxon_info.pgql

```
SELECT * MATCH (n)-[r]->(n2) WHERE id(n) = '9606'
```

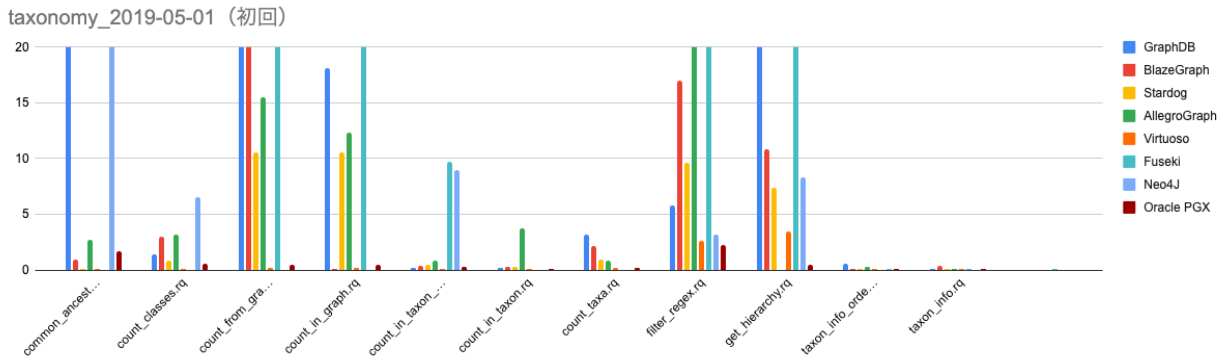
17.6 計測結果

17.6.1 taxonomy_2019-05-01.nt での計測結果

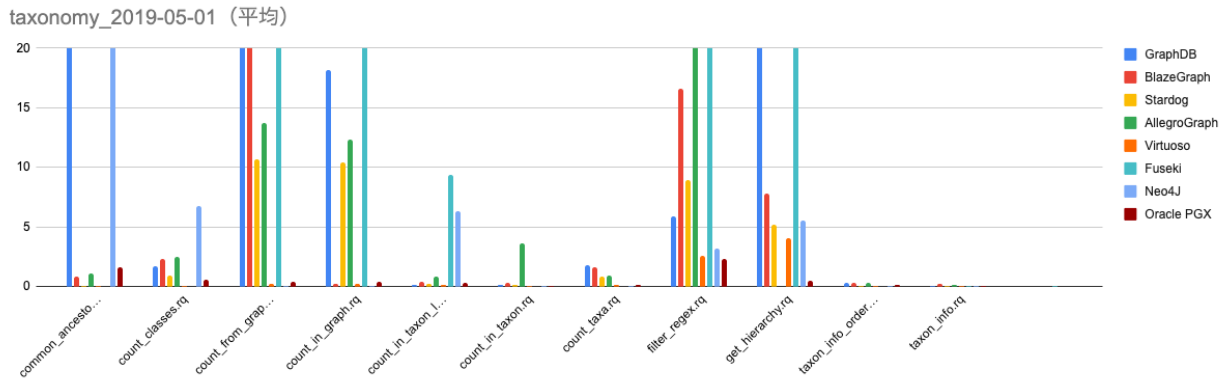
各種データベースに taxonomy_2019-05-01.nt データを用いてロード速度を計測したところ、以下の通りになった。なお、トリプルストアとの比較のために Neo4j ＋ Neosemantics 、および Oracle Labs PGX を用いたものに関しても計測を行った。当該データに関し、表や図の中では単に「Neo4j」「Oracle PGX」と表記する（Oracle PGX に関しては、クエリの実行時間のみ調査を行った）。

| トリプルストア | ロード速度 (M triples/min) | |—|—:| | Virtuoso | 5.812 | | Blazegraph | 3.299 | | Stardog | 1.885 | | GraphDB | 1.102 | | Fuseki | 3.386 | | AllegroGraph | 3.999 | | RDF4J | 2.333 | | Neo4j | 1.863 |

さらに各種データベースに関して、クエリセットの実行時間の調査を行った。調査対象は上記 11 クエリになっており、各クエリに関して 3 回ずつ実行した上で、平均を調べた。トリプルストア間の実行時間の比較を下図に示す。なお、トリプルストアによってはクエリの結果がキャッシュされる場合があるためか、初回と 2 回目以降の時間が大きく変わるものがあったため、初回のみの時間と平均の時間の両方を記載する。縦軸の単位はミリ秒である。



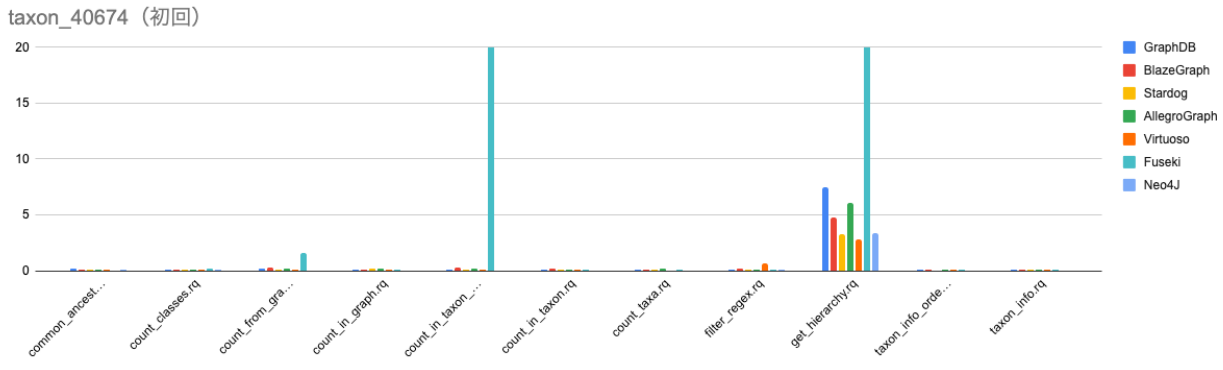
../img/taxonomy_2019-05-01_1st.png



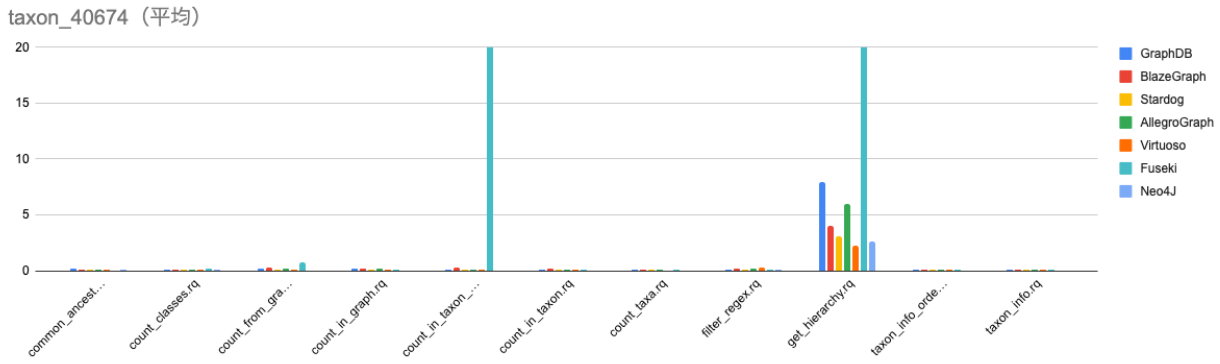
../img/taxonomy_2019-05-01_mean.png

17.6.2 taxon_40674.nt を用いた計測結果

小規模データとして taxon_40674.nt を利用した場合の各種トリプルストアのロード速度とクエリの実行時間の調査を行った。計測は 3 回行い、初回のみの値と平均の値をグラフとした。縦軸の単位は秒になっている。



../img/taxon_40674_initial.png



../img/taxon_4067

17.7 考察と課題

今回は、特定の Linux マシンの上で、パフォーマンスの測定を行った。しかし、再現性を確保する観点から、任意のマシンの上で同じ実験を遂行することができることが望ましい。そこで、各データベースエンジンのインストールプロセスを Docker 化したうえで、パフォーマンス測定まで行うことができるようにする。

また、今回は特定のデータセットのみを用いて測定を行った。一般的な RDF に適用できる情報ではあるが、どこまでデータサイズに関して、スケールするかについては未検討である。そこで、特にパフォーマンスの良かったエンジンについては、大メモリマシンで、データセットをスケールさせてパフォーマンスを調査することが必要になる。

17.7.1 各種トリプルストアおよびプロパティグラフ DB の Docker 化

Docker 環境に対して、トリプルストアやプロパティグラフ DB を容易にデプロイできることを目的とし、以下のソフトウェアの Dockerfile、および必要なファイル群を作成の上、graphdbs リポジトリの docker_files ディレクトリに配備しました。

- Virtuoso 7.2.5.2
- Neo4j 4.2.3 Community Edition
- GraphDB 9.6.0 Free Edition
- Blazegraph 2.1.5