UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Traitement d'un flux de données spatio-temporel avec PostgreSQL

Post documentation: Install solution

Rémy Detobel

Promoteur :
Prof. Esteban Zimanyi

Année académique 2018 - 2019

Contents

1 Introduction

1.1 Français

Ce document est une documentation complémentaire au mémoire "Traitement d'un flux de données spatio-temporel avec PostgreSQL". Le travail lié à ce mémoire a nécessité plusieurs tests et diverses mises en places.

Ce document vise donc à expliquer les différentes structures créées et leurs mises en place. Tous les codes ont été regroupés sur un repos Github: https://github.com/detobel36/MobilityDBComparison
Pour des raisons de facilité, ce document sera écrit en anglais. Notez que le mémoire est lui en français.

1.2 English

This document is a complementary documentation for the master thesis "Traitement d'un flux de données spatio-temporel avec PostgreSQL". The work linked to this master thesis required multiple tests and various set up.

This document try to explain the different architecture and how to set up them. All code has been pushed on a Github repository: https://github.com/detobel36/MobilityDBComparison
For easy of use, this document has been written in English. Notice that the master thesis is in French.

2 Architecture

In this section we will see the existing structure but also the architecture that could be set up.

2.1 Servers

The project has evolved in time. At the beginning a server (from ULB) was used to try some solutions: "Tristan23". But after several tests, it seems that Barefoot was consuming a lot of memory. Thus, tests were set up on this first server ("Tristan23") and "production" was set up on another server: "GeoData".

These two servers are managed by the ULB and for security reason they aren't accessible with web protocol (without a complicated authentication set up). So to display easily results, a new server was used. This new server is a private server (rent by "R. Detobel").

To summarize, in this project, three servers were used:

- GeoData: for production;
- Tristan23: for testing;
- Private: for web server.

2.2 Existing architecture

The image 1 show the end architecture set up. This architecture makes computation on Geo-

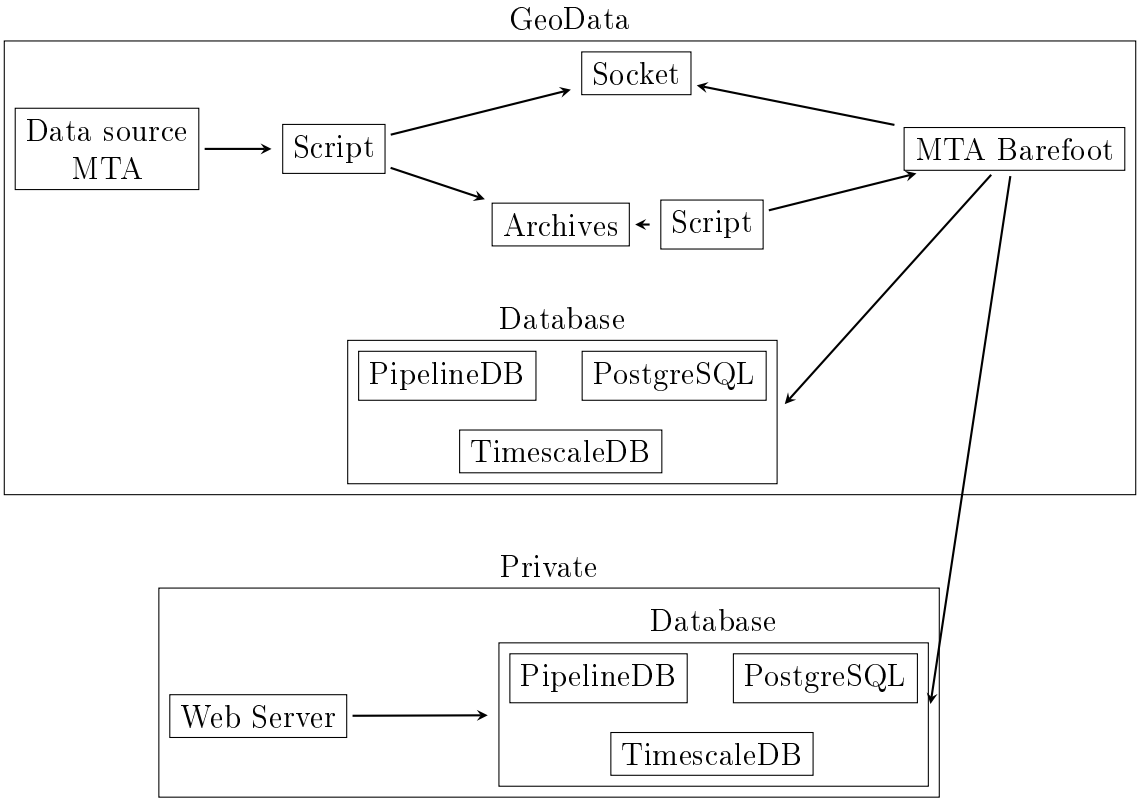

Figure 1: Existing architecture

Data server and results are displayed on private server. Notice that we need to choose: store the result on GeoData (to measure performance) or store result on private server to display results.

The "Tristan23" server isn't displayed in this image because it is only used to set up new solution and test that they work.

2.3 Possible architecture

The section 2.2 present the current solution, which respects some hardware constraint (server access, performances...).

The solution presents by the figure 1 could be simplified to have only one server. It is this solution that is presented in the image 2. The different programs used in this image will be detailed in the following points. Note that they must not all be on the same server. Most of the connections made between the different programs can be adapted to a remote solution (socket, SQL query, file recovery (especially archives)...).

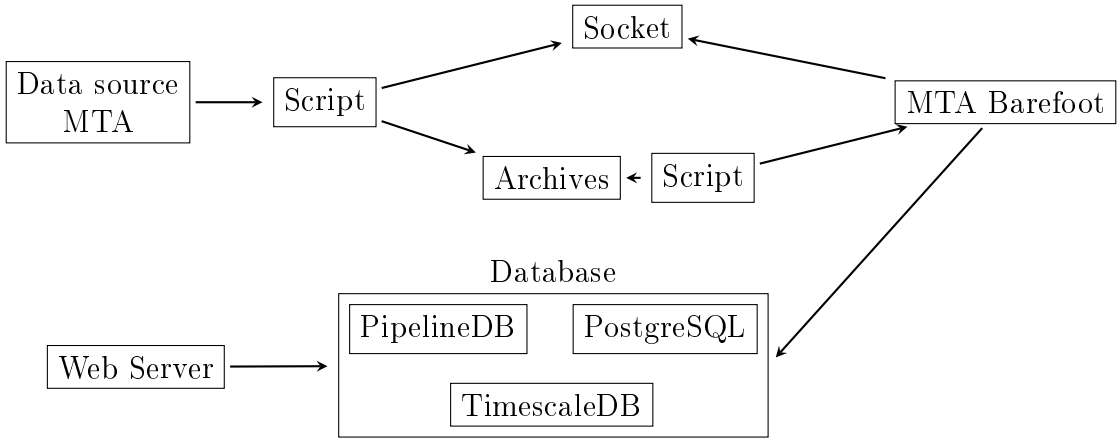

Figure 2: Possible architecture

3 Programs and configurations

This section will describe programs mentioned before. The goal is to explain the different steps to set up and configure these programs.

3.1 Script to fetch data source

In the figure 1 and 2, first elements are: "Data source MTA", "Script" (twice) and "Archives". These elements are set up by ULB.
This document will not details how it work but, to summarize it is simply a GET query send to MTA server which fetch information and create a stream socket in addition to store the content of the result into a file.

3.2 MTA Barefoot

"MTA Barefoot" is a modified version of Barefoot. In fact, it is more of a Java program using Barefoot as a library. The goal is to allow different methods to send information into Barefoot.

3.2.1 Build

The code of "MTA Barefoot" is accessible here: https://github.com/detobel36/MTABarefoot
To get the executable file (with the extension ".jar"), you only need to execute the following command: `mvn clean build`
You will get a folder "target" with two executable. One of them directly includes the Barefoot library (the one containing, logically, "with-dependencies" in its name).

3.2.2 Start

To start the executable you can execute the following command:

```
java -jar <jar file> </path/to/server/properties> </path/to/mapserver/properties>
    [INPUT: server/mta/socket] [OUTPUT: tcp/sql]
```

The first parameter is the name of the executable. The second and the third parameters define some configuration file. The two following parameters define which method must be used to (respectively) get input information and push output information.

For the configuration, the second argument reference the server configuration. "MTA Barefoot" is based on "Track server" (from Barefoot). The same configuration could be used. An example is available here: https://github.com/detobel36/MobilityDBComparison/tree/master/Tool/config

For the third parameters, it is information to connect to database. An example of configuration could be found here: https://github.com/detobel36/MobilityDBComparison/blob/master/Tool/config/map_server.properties
Notice that in image 1 the database could be on localhost or in remove. To switch easily between these two configurations, the script `switchConfig.sh` was used (available here).

There are three different type of input for "MTA Barefoot":

- "server" to have a listen port socket (define in server configuration);
- "mta" which will fetch directly information on MTA server;
- "socket" which will listen to "local" socket, define by the script (define in point 3.1).

For the output solution, two methods are available: tcp and sql. The first one come from original code of Barefoot. The goal is to have a server (NodeJS) which listen to a TCP connection and directly display result (on web). In this configuration nothing is done to save data.
The "SQL" output allow to execute one or multiple SQL Query when a position is treated. Queries which must be executed are stored in the file `request.sql`, present in the "config" folder. You will find an example here. MTA Barefoot will automatically detect if there is one or multiple query (based on ";" symbol).
Variable could be used in queries. The variable have the following syntax: "`:name` ". Notice the space after the variable name.

Possible variables in queries process by Barefoot:

- `:trip_id` (given by MTA API);
- `:start_date` (given by MTA API (type: integer));
- `:route_id` (given by MTA API);
- `:direction_id` (given by MTA API);
- `:bearing` (given by MTA API);
- `:stop_id` (given by MTA API);
- `:time` (time of the point);
- `:point` (encode as `POINT(x, y)`);

- `:route` (sequence of point that compose road);
- `:candidates` (JSON with candidate point (never test on SQL query));
- `:distance` (distance between last and current point);
- `:timeDiff` (difference in time between last and current point);
- `:speed` (speed average in km/h between last and current point (based on two last information));
- `:timeRoad` (contain all "road" point (and information) between last and current point (never test on SQL query));
- `:timeCoordinate` (contain all point (on the road) between last and current point with the MobilityDB format).

3.3 Database

The database is a PostgreSQL database with different extensions. Install instruction of extensions are available directly on respective documentation:

- MobilityDB, more information here: https://github.com/ULB-CoDE-WIT/MobilityDB
- PipelineDB, more information here: http://docs.pipelinedb.com/installation.html#install-postgresql
- TimescaleDB, more information here: https://docs.timescale.com/v1.3/getting-started/installation

In this project different solutions have been tested. Tables are not the same for each solution. But to test these solutions, some other data (and thus tables) are required.

- `usa_adm` which contains counties of USA.
  The script `import.sh` in the folder NewYorkCounty could be used to create the table and import the data.
- `gtfs_shape_geoms` which contains bus road of New York.
  The folder "Tool" contain a script to create and import easily the data. https://github.com/detobel36/MobilityDBComparison/blob/master/Tool/importGtfsMta.sh

As already mentioned, the database is based on PostgreSQL. Extensions are then added to improve or add elements to the database. Each extension allows you to create a new solution. For some solutions, variants can also be tested. Concretely, there are 5 choices:

- PostgreSQL with pre-treatment;
- PostgreSQL with post-treatment;
- PipelineDB;
- TimescaleDB with pre-treatment;

- TimescaleDB with post-treatment.

For each solution, 3 files have been created in the folder Script: one for the initialization, one for the insert query and the last for the selection.

The setup of a solution is described in the point 4.

3.4 Website

To set up a website to view result you just need to install a web server (the current solution have been tested with Apache2) and install PHP 7.0 (or upper).

Then you need to add following folder in your "web home" folder.

A small configuration is needed:

- For database, you need to update the file `model/bdd.php` with your own PostgreSQL configuration.
- For website parameter, you need to update the file `controllers/useful.php`, where you need to update Google Key and Website URL.

All files in this website are not up-to-date. The interesting pages have the prefix `visual`. For instance "visualDashboard" (the URL will be `http://your_website.com/visualDashboard`) allow you to see the first case develop in the "Result" section of the master Thesis.

In the master thesis, 3 cases were studied: bus informations, bus deviation and time in county (also named "zone"). Each case has two files: one to made the SQL query and another to display result. Notice that a Javascript code is also defined for each case (check the folder assets/js.

- **Bus information:**
  The SQL query is in the file getVisualDashboardData.php
  The display page is the file visualDashboard.php
- **Bus deviation:**
  SQL Query: getDeviationData.php
  Display file: visualDeviation.php
- **Time in county:**
  SQL Query: getCountyData.php
  Display file: visualCounty.php

4 Link all technologies

The structure has already been described by the image 2. Previous points describe how to set up different technologies.
The next step is to modify configuration and set up database to make everything work together. With this new step we will be able to made test but also to have concrete solution.

In all the situation we will use:

- Data source or archive;
- Barefoot;
- PostgreSQL with MobilityDB;
- Webiste (not required, only to show results).

In addition to these programs, we can use some extension in the database like TimescaleDB or PipelineDB.

Concretely, the first step when we want to test a solution is to set up the database. To do that, check the initialization file linked to your solution (check the folder Script).
When it is done, set the configuration of Barefoot. In the folder `config` you need to set the insert query in the file `request.sql` and check that `map_server.properties` have right access to your database.

If you would like to see the result, do not forget to update the SQL query in the web server (see point 3.4). When all is done you can start Barefoot and (normally) it is ok.