

Bachelor project– Autumn 2017

Prof. Anastasia Ailamaki

Phd. student Utku Sirin

Students: Mouadh Hamdi, Mariem Belhajali.

Content

Contents	1
1.Introduction	3
2.Application architecture	3
3.Database schema	4
3.1 Attributes description :	4
3.2 Entity-relationship model :	4
4.Queries	5
4.1 Server process queries:	5
4.2 Client process queries:	7
5.Graphical user interface (Gui)	8
5.1 Login :	10
5.2 Real-time mode :	11
5.3 History mode :	13
5.3.1 Daily average :	13
5.3.2 hourly average :	14
5.3.2 minutely average :	15

DIAS: Data-Intensive Applications and Systems Laboratory

School of Computer and Communication Sciences

Ecole Polytechnique Fédérale de Lausanne

Building BC, Station 14

CH-1015 Lausanne

URL: <http://dias.epfl.ch/>

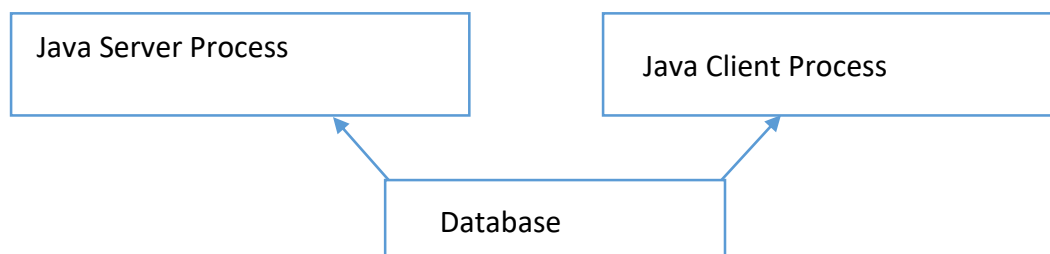


6. Performance evaluation	16
6.1 Data analysis	16
6.2 Performance testing	17
6.2.1 Experimental methodology	17
6.2.2 Results	18
6. Conclusion	19

1.Introduction

Data analytics is inspecting, cleaning, transforming and modeling data in order to extract some meaningful information. **“We are drowning in data, but starving for knowledge!”** By John Naisbett. It is mostly used in commercial industries to better the decision-making process and used by scientists to study theories and scientific models. This project will present two different modes for data analysis reunited in a Java application. The history mode will be treating all the data already saved in the system while the real-time mode is dynamic analysis and reporting based on fresh data recently entered into the system. The data studied in this project will be provided by PocketCampus company, an application deployed at École polytechnique fédérale de Lausanne (EPFL) used by thousands of students and employees.

2.Application architecture



In our application we designed two Java Applications to interact with our database:

- The Java Server Process: This can be seen as the database manager interface, It will allow him to insert data with a chosen speed rate.
- The Java Client: This is the user or the observer interface because it will allow him to see different plots of the data according to some preprocessed queries (average, 95 percentile, 99 percentile)

3.Database schema

3.1 Attributes description:

We created one table named data with the following 6 attributes:

id: A unique integer starting from 0, added to the database as primary key for the tuples since duplicates exists for all other attributes.

date: of type Date, Format yyyy-mm-dd, describes the day of the transaction.

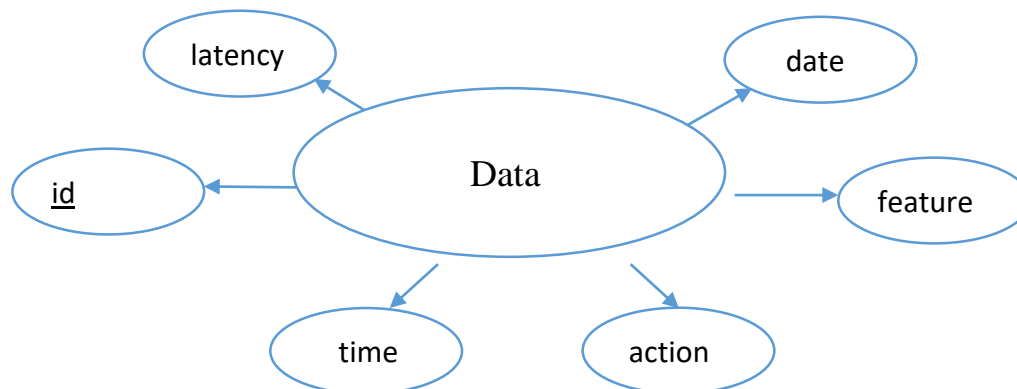
time: of type Time 00:00:00.0000000 through 23:59:59.9999999, describes the time of the transaction.

feature: of type varchar (100), describes the feature of the transaction.

action: of type varchar (100), describes the action of the transaction.

latency: float, describes how much time the transaction took in ms.

3.2 Entity-relationship model:



General comments:

-The id attribute is an attribute added by our Java Server Process, since we the attributes we extracted from

URL: <http://dias.epfl.ch/>

our data contain duplicates values. The idea behind adding this id is to somehow sort our data since we are interested in the n last added values.

-While doing the data cleaning we observed that some latency values had a wrong float format so we ignored them.

4. Queries

4.1 Server process queries:

The following commands need to be executed in order to set up the SQL Server database parameters and being able to execute queries as given.

For Windows:

Open SQL Server Management Studio (SSMS).

Open a fresh shell to excute SQL queries.

Run the create database query: CREATE DATABASE PocketCampus.

➤ use PocketCampus

➤ go

Run the created table query mentioned below.

Run the primary key query to set the is as a primary key.

When trying to run procedure we may get interrupted by some permissions issues. In order to avoid that we may need to run the next query:

-grant create procedure to BachelorProject // to allow create procedure statements

To have the possibility to do the grant procedure we need to give our database the following permissions :

-Alter any database

-create any database

URL: <http://dias.epfl.ch/>

Those permissions can be granted as follow:

select page securable explicit--> security--> login--> user (double click) --> securable and alter and create

ReadMe: Both localhost and port number.

For Linux:

To set up SQL Server you need to follow the instructions in <https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-ubuntu>.

You need also to run the following queries in the shell:

- `sqlcmd -S localhost -U user -P Password`
- run the create database query: `CREATE DATABASE PocketCampus`
- Use Pocketcampus
- `go`

Run the created table query metionned below.

Run the primary key query to set the is as a primary key.

- To create the database table:

```
CREATE TABLE data
(
    id          INT NOT NULL,
    date        DATE,
    time        TIME,
    feature     VARCHAR (100),
    action      VARCHAR (100),
    latency     FLOAT
)
```

- To create the primary key:

```
ALTER TABLE data
ADD CONSTRAINT pk_data PRIMARY KEY (id)
```

URL: <http://dias.epfl.ch/>

4.2 Client process queries:

The following queries are automatically executed by the Java Client Process, it will be done due to a call to the main function of the class SqlFiles. The queries are in the queries.sql file.

- To insert tuples into the table.

```
CREATE PROCEDURE Insertionpr @id          INT,  
                             @date        DATE,  
                             @time        TIME,  
                             @feature     VARCHAR (100),  
                             @action      VARCHAR (100),  
                             @latency     FLOAT  
  
AS  
BEGIN  
    BEGIN TRANSACTION  
  
    INSERT INTO data  
    VALUES      (@id,  
                 @date,  
                 @time,  
                 @feature,  
                 @action,  
                 @latency)  
  
    COMMIT TRANSACTION  
END;
```

- To find the average latency for the last 100 entries of the table of all or a specific action and feature.

```
CREATE PROCEDURE Avgpr @feature VARCHAR (100),  
                      @action  VARCHAR (100)  
  
AS  
BEGIN  
    SELECT Avg(latency)  
    FROM    (SELECT TOP 100 *  
             FROM      data  
             ORDER BY id DESC) data  
    WHERE   action LIKE @action  
           AND feature LIKE @feature  
END;
```

- To find the 99 percentile value of the latency for the last 100 entries of the table of all or a specific action and feature.



URL: <http://dias.epfl.ch/>

```
CREATE PROCEDURE NinetyNinePr @feature VARCHAR (100),
                                @action  VARCHAR (100)
AS
BEGIN
    SELECT TOP 1 latency
    FROM    (SELECT TOP (99) PERCENT latency
            FROM    (SELECT TOP 100*
                    FROM    (SELECT TOP 100*
                            FROM    data
                            ORDER BY id DESC) data
                        ORDER BY latency ASC) AS table
                    ORDER BY latency DESC) AS latency
    ORDER BY latency DESC
END;
```

- To find the 95 percentile value of the latency for the last 100 entries of the table of all or a specific action and feature.

```
CREATE PROCEDURE NinetyFivePr @feature VARCHAR (100),
                                @action  VARCHAR (100)
AS
BEGIN
    SELECT TOP 1 latency
    FROM    (SELECT TOP (95) PERCENT latency
            FROM    (SELECT TOP 100*
                    FROM    (SELECT TOP 100*
                            FROM    data
                            ORDER BY id DESC) data
                        ORDER BY latency ASC) AS table
                    ORDER BY latency DESC) AS latency
    ORDER BY latency DESC
END;
```

- To find the **daily average latency** of a certain month of the year of all or a specific action and feature:

```
CREATE PROCEDURE Avgdaily @year      INT,
                            @mon       INT,
                            @feature   VARCHAR (100),
                            @action    VARCHAR (100)
AS
BEGIN
    SELECT Avg(latency),
           Datepart (day, date)
    FROM    data
```


URL: <http://dias.epfl.ch/>

```
WHERE Datepart (month, date) = @mon
      AND Datepart (year, date) = @year
      AND action LIKE @action
      AND feature LIKE @feature
GROUP BY Datepart (day, date)
ORDER BY Datepart (day, date)
END;
```

- To find the **hourly average latency** of a certain date for all or a specific action and feature:

```
CREATE PROCEDURE Avghourly @date DATE,
                           @feature VARCHAR(100),
                           @action VARCHAR(100)
AS
BEGIN
    SELECT Avg(latency),
           Datepart (hour, time)
    FROM data
    WHERE date = @date
          AND action LIKE @action
          AND feature LIKE @feature
    GROUP BY Datepart (hour, time)
    ORDER BY Datepart (hour, time)
END;
```

- To find the **minutely average latency** of a certain hour in a certain date for all or a specific action and feature:

```
CREATE PROCEDURE Avgminutely @date DATE,
                              @feature VARCHAR (100),
                              @action VARCHAR (100),
                              @hour INT
AS
BEGIN
    SELECT Avg(latency),
           Datepart (minute, time)
    FROM data
    WHERE date = @date
          AND action LIKE @action
          AND feature LIKE @feature
          AND Datepart (hour, time) = @hour
    GROUP BY Datepart (minute, time)
    ORDER BY Datepart (minute, time)
END;
```

URL: <http://dias.epfl.ch/>

5. Graphical user interface (GUI)

5.1 Login:

Once we run the Java Client process or the Java Server Process a connexion interface will pop-up to allow us to enter our logins as follow:

Host: The address of the server to connect to. This could be a DNS or IP address, or it could be localhost or 127.0.0.1 for the local computer. If not specified in the connection URL, the server name must be specified in the properties collection.

Database name: The name of the data base that you want to connect to it.

User: The user that have access to data base mentioned above.

Password: The password associated to user.

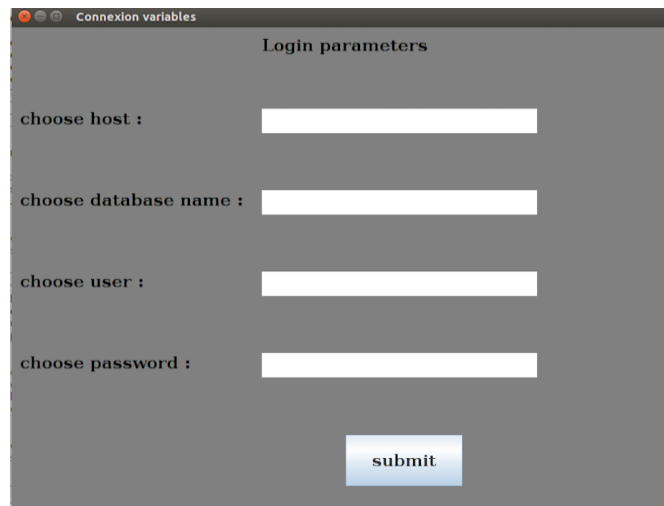
Example:

Choose host : localhost

choose database name : PocketCampus

Choose user: sa

Choose password : *****



The image shows a Java Swing window titled "Connexion variables". Inside the window, there is a section titled "Login parameters". Below this title, there are four labels with corresponding text input fields: "choose host :", "choose database name :", "choose user :", and "choose password :". At the bottom right of the window, there is a "submit" button.

Figure 1 : Login main page

-Figure 1 is the login main page which will appear every time we run a Java Process. Once we enter the adequate logins we press submit.

URL: <http://dias.epfl.ch/>

5.2 Real Time mode:

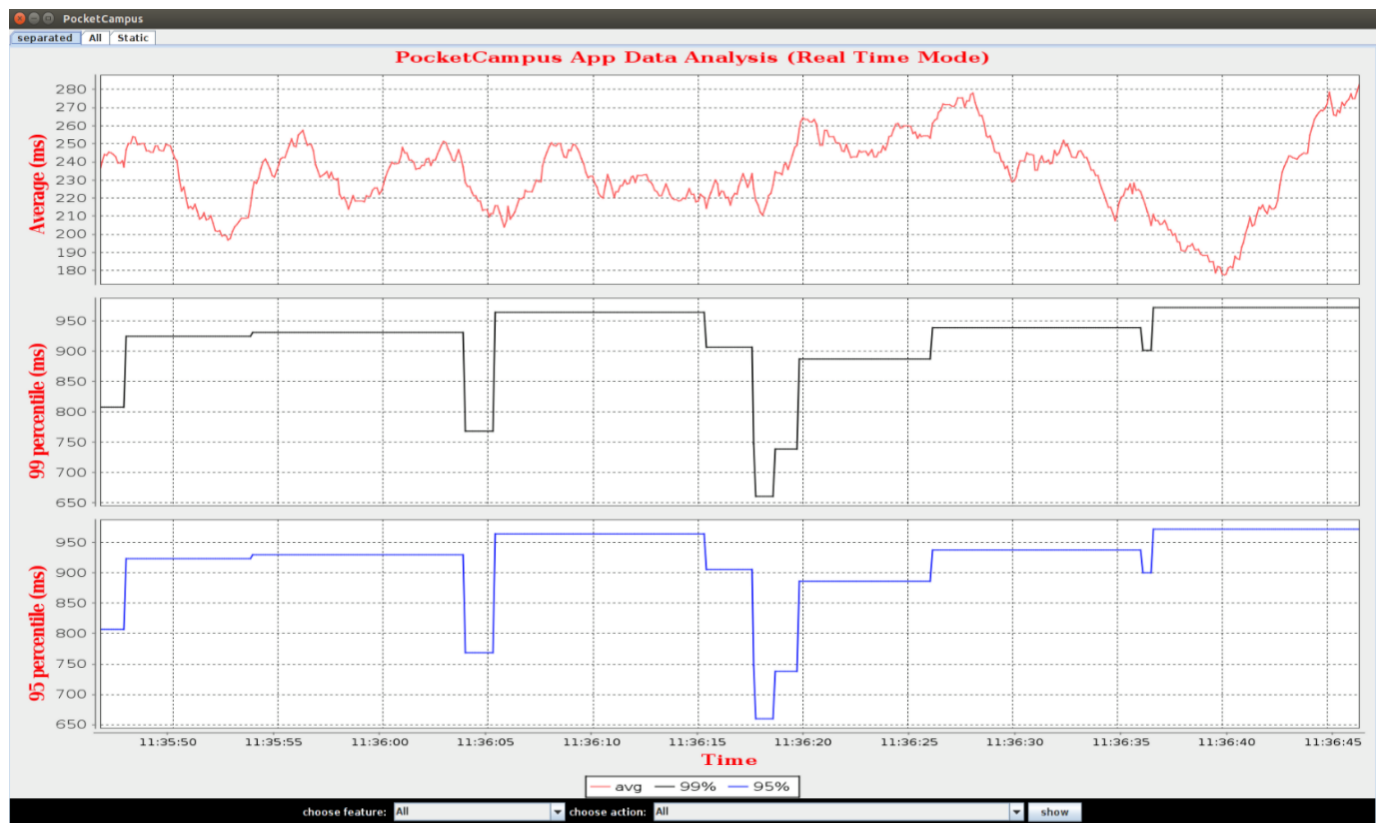


Figure 2: Dynamic charts (separated mode)

1. The first chart will contain the dynamic graph of the average value expressed in ms drawn in **red**. This average value is the result of the procedure `avgPr` mentioned in section 3.2
2. The second chart will contain the dynamic graph of the 99 percentile value expressed in ms drawn in **black**, which is the result of the procedure `ninetyNinePr` mentioned in section 3.2
3. The third chart will contain the dynamic graph of the 95 percentile value expressed in ms drawn in **blue**, which is the result of the procedure `ninetyFivePr` mentioned in section 3.2

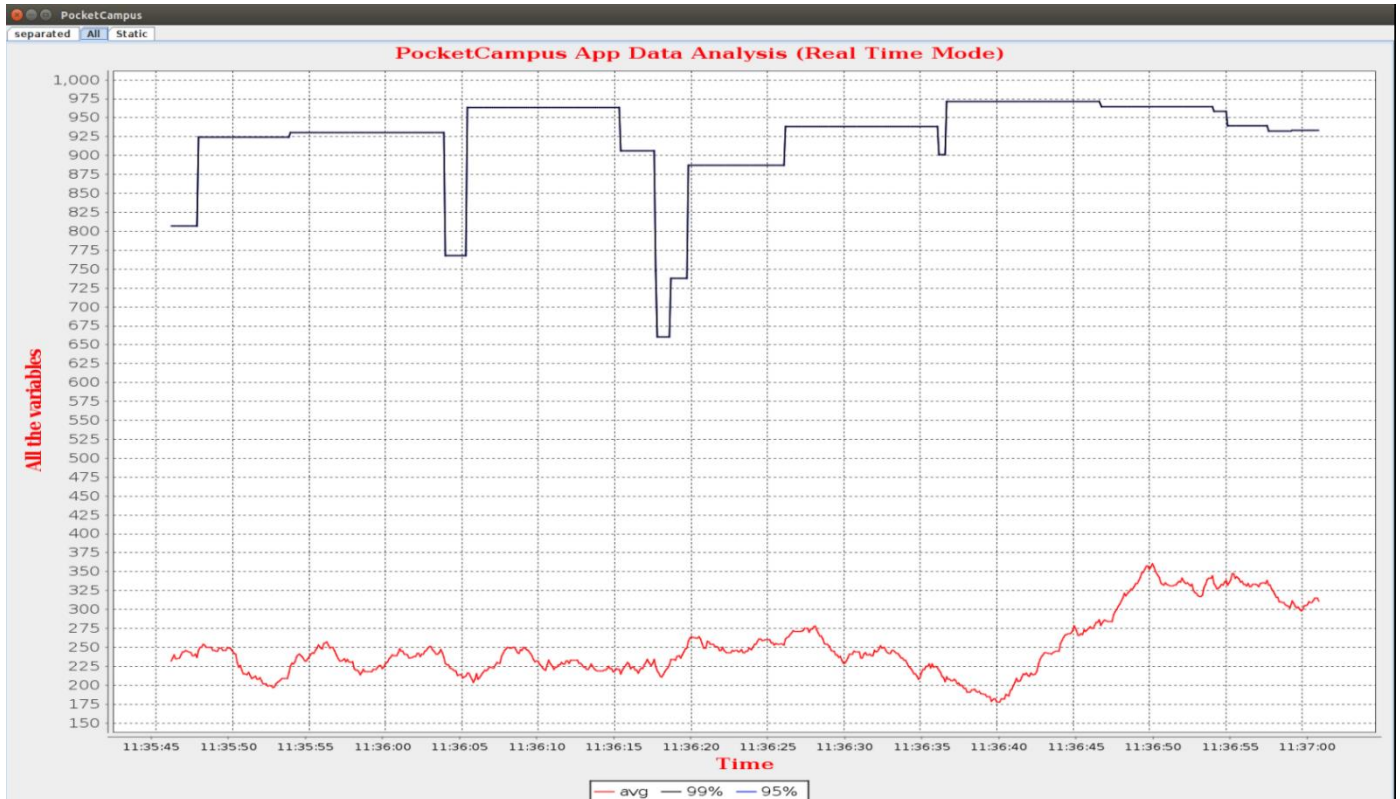


Figure 3: Dynamic charts (combined mode)

this figure reunites the previous plots on the same chart preserving the same scale and colors.

The values we chose to plot are representative for the data since:

- The **average** is a number expressing the central or typical value in a set of data, which is calculated by dividing the sum of the values in the set by their number.
- The **n percentile** is a measure used in statics indicating the value below which a given percentage of

URL: <http://dias.epfl.ch/>

observations in a group of observations fall. For our example, the 99th percentile is the value of latency below which 99% of the observations may be found.

-In the previous charts the user can filter the mentioned values according to features, actions, or a combination of both to have a more specific result if needed.

-Initially, the given results are for all actions and all features.

-The combined figures represented in Figure 3 were plotted in order to help the user visualize the gap between the three values. For example, one can notice that the 99 and 95 percentile values are very close.

5.3 History mode:

The history mode has a lot of features such as debugging. If some bug occurs in real time, we need to go to the previous registered values to be able to debug and to learn more about the issue. For that we decided to implement a history mode with multiple displays. In this mode, the user can specify the year and the month to observe the daily average latency of that month on a histogram mode. By adding the day in the right format (from 00 to 31), the hourly average latency of that day will be plotted. By specifying the hour of the day (from 00 to 23) minutely average hour will be plotted.

5.3.1 Daily average:

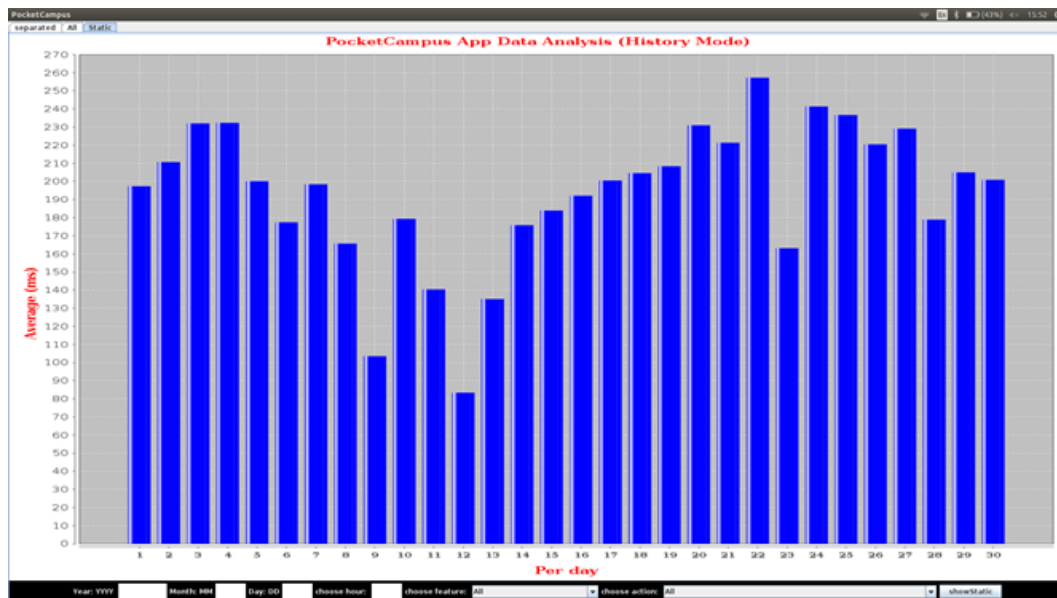


Figure 4: Daily average chart (History mode)

The previous histogram displays the average latency of all the days of a month chosen by the user.

In the daily average mode, the user can display the average values of all days of a certain month, he need to run the Java Client Process by first opening the Static tab. Then, he needs to enter the year and the month then click on show to update the chart.

By default, we display the mode per day of the the first month.

Sometimes we choose the per day mode but we don't have all the data of that month, if they getting inserted by clicking on show we will see progress.

5.3.2 Hourly average:

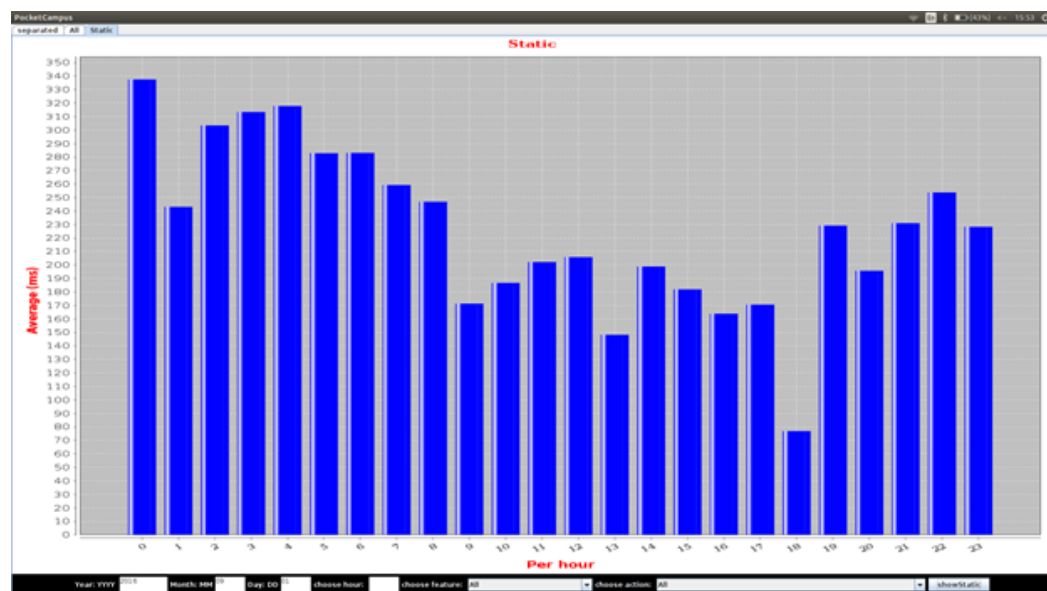


Figure 4: Hourly average chart (History mode)

This figure is a screen shot of the per hour mode of the static chart.

This Figure represent the average values in all hours of a selected day.

The user need to specify the year, the month and the date and to process show in order to apply changes.

5.3.2 Minutely average:

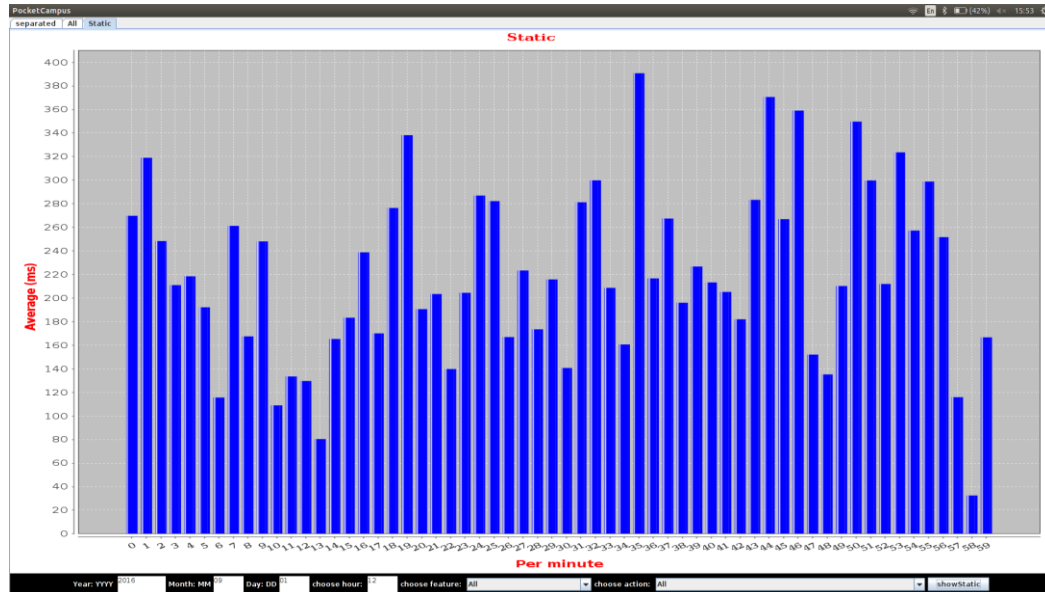


Figure 4: Minutely average chart (History mode)

This figure is a screen shot of the per minute mode of the static chart.

If the user chooses the hour besides the year the month and the day, we will the plot the average value of the 60 minutes corresponding to the chosen hour.

General comments:

The following libraries are essential for the project:

- sqldbc4-3.0.jar: To access data from a SQL Server Database by using the Microsoft JDBC Driver for SQL Server

URL: <http://dias.epfl.ch/>

- jfreechart-1.0.14.jar: JFreeChart is a Java chart library that makes it easy for developers to display professional quality charts in their applications.
- jcommon: JCommon is a general purpose Java class library that is used in several projects at www.jfree.org, including JFreeChart

6. Performance evaluation

6.1 Data analysis

- Number of transactions per day e.g 2016-09 :

Query:

```
SELECT    Count (*)
FROM      data
WHERE     Datepart(month, date)=09
AND
where     datepart(year, date)=2016
GROUP BY  datepart(day, date)
```

Result: The number of transactions per day for the september is represented in the screenshot. As we can see in average we have from 11000 to 205000 transactions per day.

- Number of transactions per hour e.g 2016-09-01:

Query:

```
SELECT    Count (*)
FROM      data
WHERE     Datepart(month, date)=09
AND
where     datepart(year, date)=2016
AND       datepart(day, date)=01
GROUP BY  datepart(hour, time)
```

Result: the number of transactions per hour for 2016-09-01 is represented in the screenshot . As we can see in average we have from 125 to 1337 transactions per hour.

- Number of transactions per minute e.g 2016-09-01 for 12 pm:

Query:

```
SELECT    Count(*)
FROM      data
WHERE     Datepart(month, date) = 09
AND       Datepart(year, date) = 2016
AND       Datepart(day, date) = 01
```


URL: <http://dias.epfl.ch/>

```
AND Datepart(hour, time) = 12  
GROUP BY Datepart(minute, time)
```

Result: the number of transactions per minute for 2016-09-01 at 12pm is represented in the screenshot . As s

6.2 Performance testing

In this section, we modified slightly our project to visualize how the latency of the average query varies according to the type of tables and the insertion rate. Tests will be made for two types of database tables: Disk Based and Column Store Disk Based table .In a disk-based table ,the data is stored primarily on disk, and usually the server copies only small pieces of data at a time into memory. In a column store Disk Based table, the data tables are stored by column rather than by row.

6.2.1 Experimental methodology

1.First we create **two tables** on SQL server :

- Disk based Table:

```
CREATE TABLE diskbased  
(  
    id          INT NOT NULL PRIMARY KEY,  
    date        DATE,  
    time        TIME,  
    feature     VARCHAR(100),  
    action      VARCHAR(100),  
    latency     FLOAT  
)
```

- Column store disk based Table:

```
CREATE TABLE diskbasedcs  
(  
    id          INT NOT NULL PRIMARY KEY,  
    date        DATE,  
    time        TIME,  
    feature     VARCHAR(100),  
    action      VARCHAR(100),  
    latency     FLOAT  
)
```

```
CREATE NONCLUSTERED columnstore INDEX diskbasedcsi ON diskbasedcs (id, date,  
time, feature, action, latency);
```

2.The experiment is based on data extracted from the logs.txt file provided by PocketCampus. **Initially, both tables will contain 3 months of data which is equivalent to 7 million tuples.**

3.Once the tables are ready, we start inserting by running the JavaServerProcess class for 30 seconds, the insertion rate will be modified at each time by pausing the program at the beginning of the loop before the execution of the insertion query trying, four different values will be tested: **0 ms (no pause), 1 ms, 10 ms , 100 ms.**

4.For each of those different insertion rate we will run the JavaClientProcess class for 2 minutes, with slight modifications of the java code , we are able to calculate the average time it takes to execute AvgPr query in the JavaServerProcess class over 2 minutes which as explained in the queries section calculates the average latency for the **last 100 entries** of the table for 2 minutes . In this experiment each time we will vary the number of entries in the query trying three different values: **Last 100 , last 10 000 and last million.** The steps will be executed for both tables.

5.At the beginning of each test we make sure that the insertions made in the third step of the tests are not considered, in order to have the same initial conditions the the tests.

6.2 Result

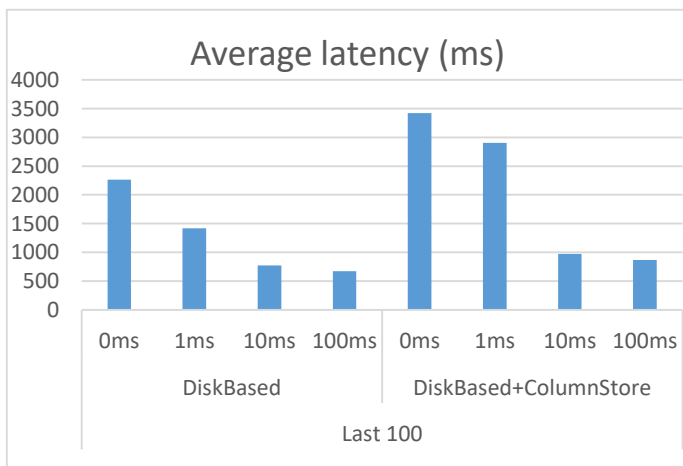


Figure 5: Results of the experience for the last 100 entries

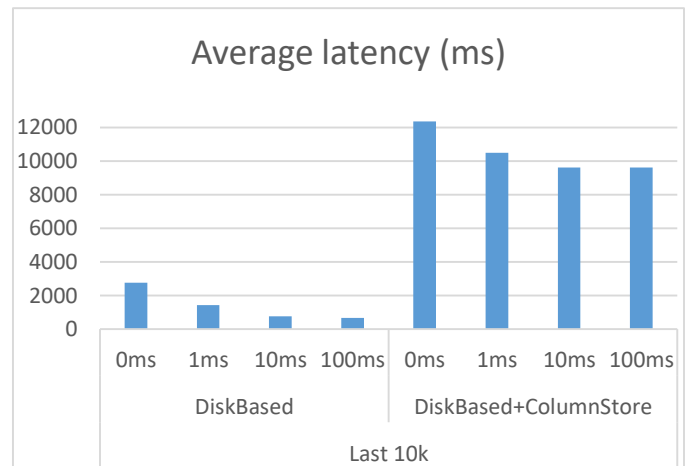


Figure 6: Results of the experience for the last 10k entries

URL: <http://dias.epfl.ch/>

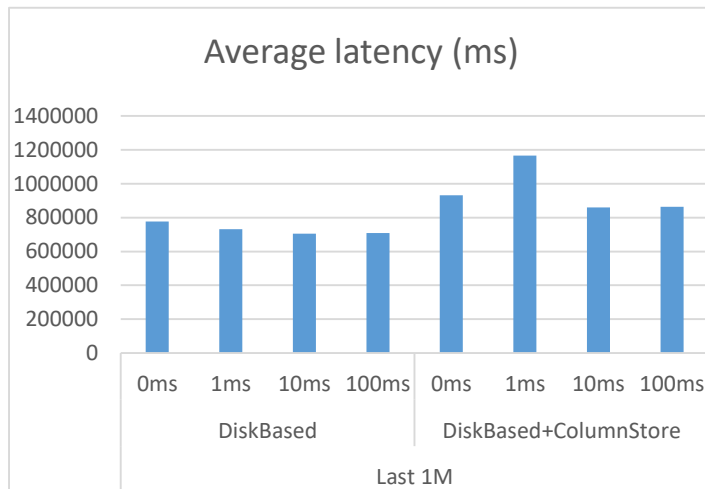


Figure 7: Results of the experience for the last million qentries

Two main results are shown in the graphs:

1. As insertion rate decreases the execution of query gets faster.
2. Surprisingly, column store disk based table is slower than the disk based table.

-To clarify the origin of those results and the reason that column store is slower in our case, more research and tests must be made but due to the lack of time, this will not be a part of our project.

7. Conclusion

This Java application can analyze PocketCampus data giving a clear view of its performance according to multiple factors in real-time and history mode.

DIAS: Data-Intensive Applications and Systems Laboratory

School of Computer and Communication Sciences

Ecole Polytechnique Fédérale de Lausanne

Building BC, Station 14

CH-1015 Lausanne

URL: <http://dias.epfl.ch/>

