Seminarium Problemowe
Sprawozdanie

ARKADIUSZ RUSIN, 106644
BARTOSZ WOŹNIAK, 106089
ADRIAN SZYMCZAK, 98758

Spis treści

Omówienie Eksperymentu 1
Przebieg eksperymentu 1
Sposób zbierania danych 2
Napotkane trudności 3
Zastosowane narzędzia 4
Zapis danych do bazy danych 4
Analiza danych i przykładowe wykresy 5

Omówienie Eksperymentu

Tematem naszego projektu było: "Zbieranie, przetwarzanie i wizualizacja danych klasy Big Data". Nasza grupa miała za zadanie przeprowadzenia analizy i wizualizacji, zmieniających się w czasie dnia i w zależności od linii trasy, czasów przejazdu komunikacją miejską w Poznaniu. Eksperyment ten składał się z dwóch etapów. Pierwszy etap obejmował zbudowanie aplikacji do kolekcjonowania logów danych o faktycznych godzinach przejazdów komunikacji miejskiej, a także zaprojektowanie struktury danych wykorzystywanej do przechowywania otrzymywanych danych z zewnętrznego serwera. Drugim etapem było przetwarzanie uzyskanych danych, w tym ich wizualizacja.

Przebieg eksperymentu

Pierwszym krokiem pierwszego etapu projektu, było zebranie danych reprezentujących dany przystanek na mapie. Informacje te udało się uzyskać, poprzez automatyczne przeszukanie stron znajdujących się na witrynie http://www.mpk.poznan.pl/. Następnie, z uzyskanych danych, należało wyciągnąć unikalne adresy i kody poszczególnych przystanków. Drugim krokiem danego etapu było ustalenie adresu, na który należy wysyłać zapytanie, a także format zapytania i odpowiedzi. Ustalono, że właściwym adresem jest **HTTP://WWW.PEKA.POZNAN.PL/VM/METHOD.VM?TS=1426074314843** gdzie „ts” oznacza znacznik czasowy – timestamp, który jest nieistotny z naszego punktu widzenia i w każdym zapytaniu może być stały (prawdopodobnie służy analizie danych po stronie zdalnego serwera). Na adres ten należy wysłać wiadomość HTTP GET z payloadem jak w łańcuchu tekstu:

STRING S = „METHOD=GETTIMES&P0=%7B%22SYMBOL%22%3A%22" + NAZWA_PRZYSTANKU + "%22%7D"

Nazwa przystanku składa się z od 2 do 5 znaków alfanumerycznych i 2 cyfr na końcu. Wyjątkiem jest SWMN(x)01. Niestety nie istnieje adres z którego można łatwopobrać listę przystanków, a próba odpytania każdej możliwości jest obliczeniowo zbyt kosztowna. Dlatego zebranie nazw przystanków wykonano. Uzyskano w efekcie listę około 1400 nazw przystanków. Zostały one zapisane w kodzie programu gdyż rzadko ulegać będą zmianie, jednak w razie potrzeby łatwo zmienić logikę na okresowe odczytywanie ich z pliku lub bardziej wyrafinowaną metodę (obecnie natomiast uznano je natomiast za zbędne).

Odpowiedzi mają format JSON i przykładowa wygląda następująco:

{"success":{"bollard":{"symbol":"URWO01","tag":"URWO21","name":"Urząd Wojewódzki","mainBollard":false},"times":[{"realTime":true,"minutes":1,"direction":"Os. Sobieskiego","onStopPoint":false,"departure":"2015-05-13T01:49:00.000Z","line":"235"},{"realTime":false,"minutes":15,"direction":"Mogileńska","onStopPoint":false,"departure":"2015-05-13T02:03:00.000Z","line":"233"}]}}

Odpowiedź można interpretować następująco:

Z przystanku Urząd Wojewódzki o symbolu URWO21(w inną stronę jest inny przystanek) mają odjechać 2 autobusy. Jeden o numerze 235 ma odjechać za minutę i drugi o numerze 233 ma odjechać za 15 minut. Wartość onStopPoint oraz realTime, mogące przyjmować wartości true lub false dostarczają dodatkowej informacji na temat tego, czy pomiar odbył się na przystanku czy w trasie, oraz czy pochodzi z nadajnika umieszczonego w autobusie czy z przystanku i jest w efekcie jedynie oszacowany. Te posiadające wartość true zawierają dodatkowo informacje o minutach do odjazdu.

Sposób zbierania danych

Główna logika odpytywania zdalnego serwera odbywa się w metodzie ObtainData klasy WebDataObtainer. Sama strona odświeża swoje dane periodycznie co 20 sekund więc częstsze odpytywanie nie ma sensu. Z drugiej strony należy w ciągu co najwięcej 20 sekund odpytać wszystkie przystanki, by nie tracić żadnych możliwych do pozyskania danych.

Należy odpytać około 700 przystanków na temat ruchu w każdą ze stron. Ponieważ w jednym zapytaniu można uzyskać informację na temat jednego przystanku w jedną stronę należy wystosować ponad 1400 zapytań.

Generalna logika działania programu jest następująca:

Lista przystanków wczytywana jest do pamięci, a następnie w nieskończonej pętli wywoływana jest metoda ObtainData (opis poniżej), której wynik, będą listą maksymalnie 1400 odpowiedzi ze wszystkich przystanków w formacie podanym powyżej, zapisywany jest do pliku. Zapis do pliku odbywa się z pomocą biblioteki służącej do logowania Log4j i jej klasy RollingFileAppender, która pozwala na ustalanie ilości plików logów, ich sumarycznego maksymalnego rozmiaru oraz maksymalnego rozmiaru pojedynczego pliku, a także zajmuje się kompresją starszych logów. Kompresja jest o tyle istotna, iż dane w formacie JSON o jednakowej strukturze są łatwo kompresowane i współczynnik kompresji wynosił około 95%. Pliki tekstowe logu i ich archiwa używane są jako format pośredni, który jest szczególnie użyteczny z racji możliwości zmiany decyzji odnośnie używanej bazy danych. Sama metoda ObtainData działa następująco:

Maksymalnie trzykrotnie (linia 12) następuje próba odpytania wszystkich przystanków ze zbioru przystanków(linia 14), od których jeszcze nie udało uzyskać się odpowiedzi (który jest pomniejszany o dany przystanek, za każdym razem gdy uda się dla niego uzyskać odpowiedź). Ponadto bez względu na sukces lub porażkę odpytania (success / error / timeout) co 282 zapytania nastąpi przerwa w odpytywaniu na zadany parametrem wywołania programu czas lub w przypadku jego braku na 2 sekundy(linia 2, 15-21)

Napotkane trudności

Wpierw zastosowano stałe odstępy czasowe pomiędzy ciągłymi zapytaniami, niestety metoda ta nie zadziałała. Dane były pobierane zbyt i serwer blokował nasze zapytania. Poprzez pomiar czasu spędzonego na odpytywaniu, oraz biorąc pod uwagę, że należy w ciągu 20 sekund odpytać wszystkie przystanki, metodą prób i błędów mającą na celu uniknięcie blokowania, dobrano powyżej opisaną metodę odpytywania z parametryzowanym czasem usypiania między grupami zapytań (o wielkości 282).

Polityka serwera blokuje dany IP po kilku tysiącach zapytań początkowo na krótko, a następnie na coraz dłużej, aż w końcu uniemożliwia je całkowicie, dlatego należy wziąć pod uwagę możliwość ponownej konieczności estymacji wartości wszystkich parametrów w przyszłości. Dodatkowo można uruchomić program na wielu maszynach z różnymi publicznymi adresami IP (po transformacjach NAT) lub zastosować serwery proxy. By to łatwo umożliwić długości przerwy w odpytywaniu oraz wielkości grup są parametrem wywołania programu.

```
1 public Collection<String> ObtainData(Collection<Stop>
stopsCollection) {
2 int counter; int repetitions = 3; int sleepEveryN = 282;
3 Collection<String> result = new LinkedList();
4 ArrayList<Stop> stops = (ArrayList<Stop>) stopsCollection;
5 ArrayList<Stop> newStops = new ArrayList<Stop>(stops.size());
6 for(Stop s : stops) newStops.add(new Stop(s));
7 try { url = new URL(urlString);
8 } catch (MalformedURLException ex) {
9   Logger.getLogger(WebDataObtainer.class.getName())
10  .log(Level.SEVERE, null, ex);
11 }
12 for(int i = 0; i <repetitions; i++){
13   counter = stops.size()-1;
14 while(stops != null && stops.size() != 0 && counter >= 0){
15   totalCounter++;
16   if(totalCounter % sleepEveryN == 0) {
17   try { Thread.sleep(Constants.politnessMilisecondsSleep);
18   } catch (InterruptedException ex) {
19    Logger.getLogger(WebDataObtainer.class.getName())
20    .log(Level.SEVERE, null, ex);
21   } }
22   payload = Constants.payloadTemplate
23   .replace(Constants.replacementTemplate,
24   stops.get(counter).getSymbol());
25   sb = new StringBuffer();
26   try {
27      connection = url.openConnection();
28      connection.setDoInput(true);
29      connection.setDoOutput(true);
30      connection.connect();
31      os = connection.getOutputStream();
32      pw = new PrintWriter(new OutputStreamWriter(os));
33      pw.write(payload);
34      pw.close(); os.close();
35      is = connection.getInputStream();
36      reader = new BufferedReader(new InputStreamReader(is));
37      String line = null;
38      while ((line = reader.readLine()) != null) sb.append(line);
39      is.close();
40      response = sb.toString();
41      if(!response.startsWith("{\"success\""))
42         response += (" " + stops.get(counter).getSymbol());
43      result.add(response);
44      newStops.remove(counter);
45   } catch (IOException ex) {
46      Logger.getLogger(WebDataObtainer.class.getName())
47      .log(Level.SEVERE, null, ex);
48   } finally {
49      counter--;
50      if (pw != null) pw.close();
51      if (os != null) {
52         try { os.close();
53         } catch (IOException ex) {
54            Logger.getLogger(WebDataObtainer.class.getName())
55            .log(Level.SEVERE, null, ex);
56         } }
57      if (is != null) {
58         try { is.close();
59         } catch (IOException ex) {
60            Logger.getLogger(WebDataObtainer.class.getName())
61            .log(Level.SEVERE, null, ex);
62         } } } }
63 stops = newStops;
64 if(stops != null && stops.size() != 0) {
65 newStops = new ArrayList<Stop>(stops.size());
66 for(Stop s : stops) {
67   newStops.add(new Stop(s));
68 } } } return result; }
```

Listing kodu metody ObtainData 1

Zastosowane narzędzia

1. Jako rozproszonego systemu zarządzania wersji użyliśmy Gita - projekt jest hostowany na Githubie, przez co dostępny jest publicznie pod adresem https://github.com/AdrianSzymczak/pekamonitoring
2. Środowiska programistycznego NetBeans 8.0.1
3. JDK i JRE w wersji 7 (Oracle)
4. Bibliotek do obsługi baz danych i formatu JSON
5. Tableau - do analizy danych (w wersji darmowej)
6. Microsoft Excel – do przechowywania prostych arkuszy np. z nazwami przystanków

Ewentualne przeniesienie projektu z NetBeans do Eclipse jak i zastosowanie innych bibliotek do obsługi JSONa i bazy danych nie jest problemem i powinno być możliwe do wykonania w ciągu kilkunastu minut w razie potrzeby.

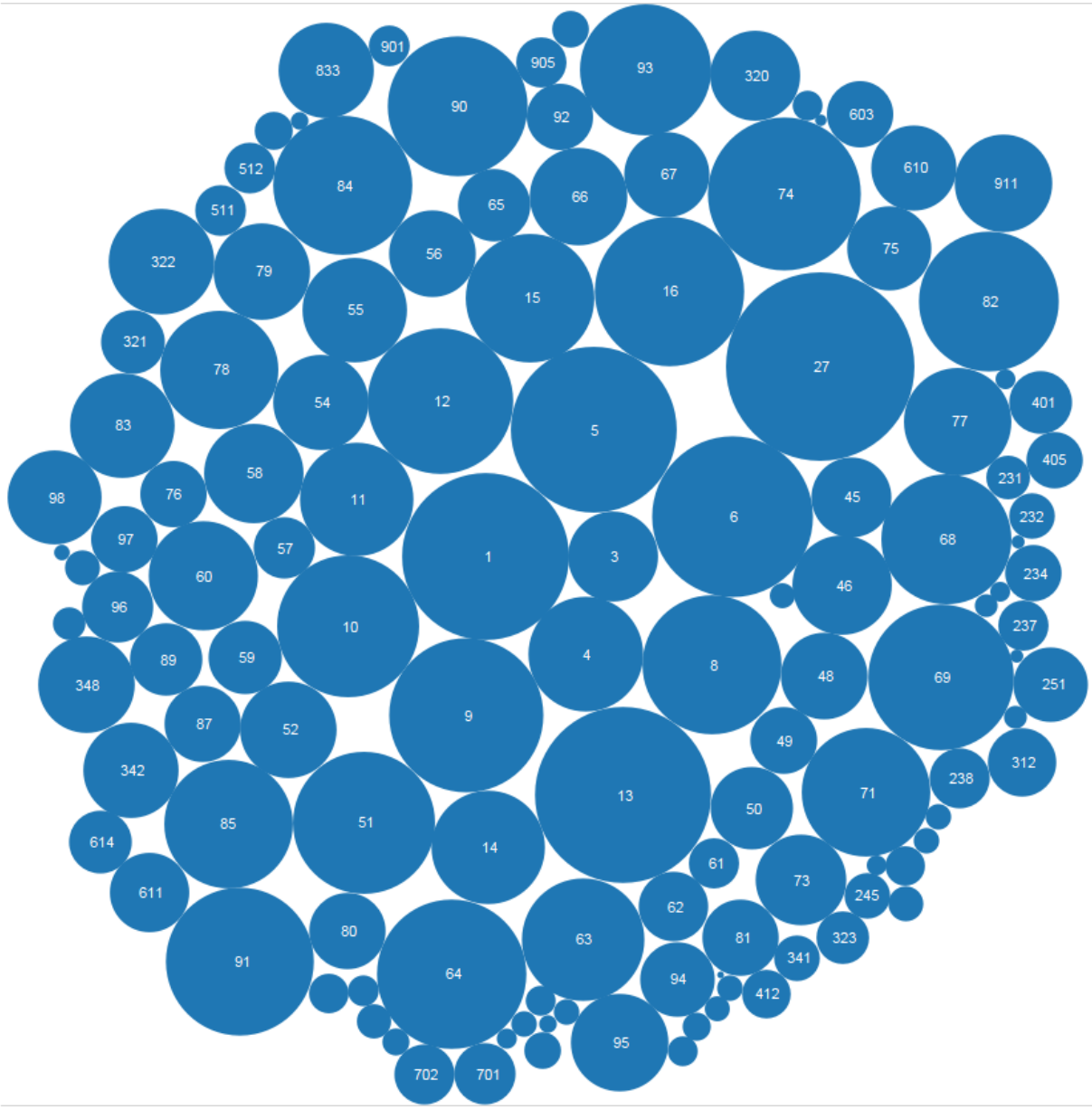
Zapis danych do bazy danych

Do sprawnego importowania zebranych danych do bazy danych mongoDB przygotowano skrypt w języku python. Do jego poprawnego wywołania wystarczy podać w argumencie ścieżkę folderu z danymi.

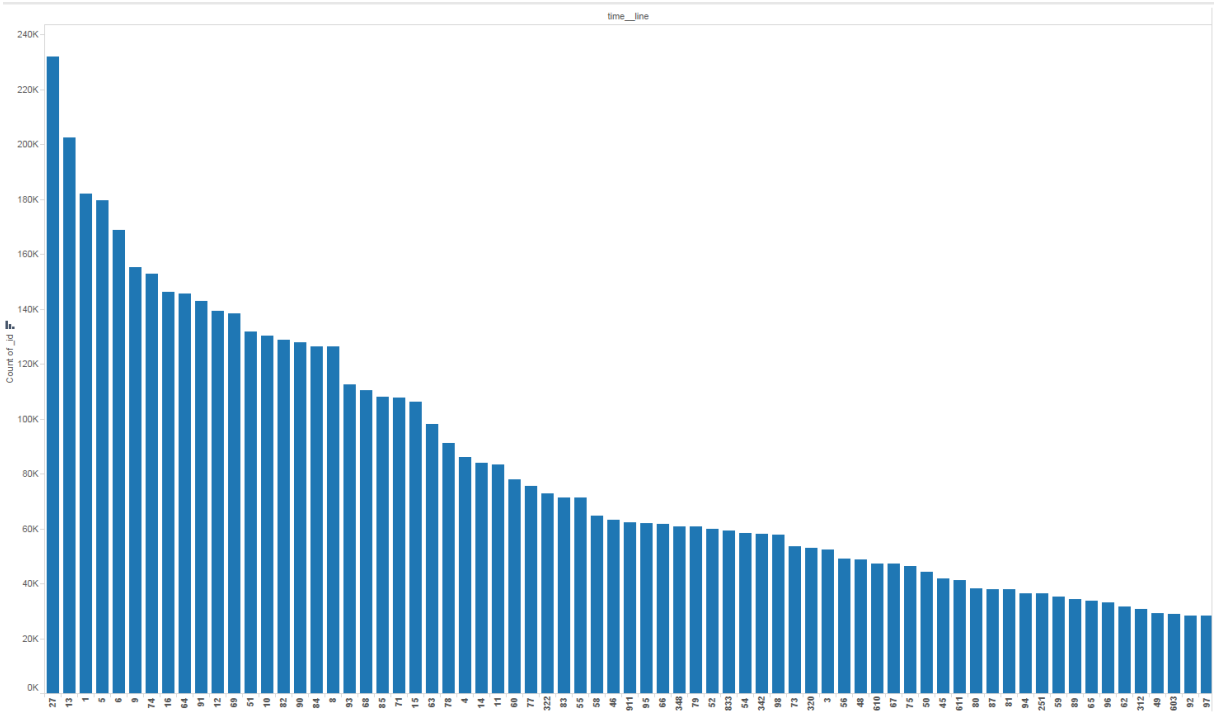
```
import sys
import pymongo
from pymongo import MongoClient
import json
client = MongoClient('localhost', 27017)
db = client['peka_db']
collection = db['peka_details']
for arg in sys.argv:
	if arg == sys.argv[0]:
		continue;
	num_lines = sum(1 for line in open(arg,'r'))
	count = 1;
	percentage_progress = 0;
	search_lines = open(arg,'r')
	for line in search_lines:
		count = count+1
		progress = count/float(num_lines)*100
		if percentage_progress != int(progress):
			print 'File: '+str(arg)+' '+str(percentage_progress)+'%'
			percentage_progress = int(progress)
		if not line.strip():
			continue
		success = line[26:33]
		if success == 'success':
			date = line[0:23]
			details = line[24:]
			details_json = json.loads(details)
			insert_row = {}
			insert_row['bollard'] = (details_json['success']['bollard'])
			insert_row['timestamp'] = date

			for time in details_json['success']['times']:
				insert_row['time'] = time
				collection.insert(insert_row)
				del insert_row['_id']
		if count == num_lines:
			print 'File: '+str(arg)+' 100% DONE'
print '...loading complete!'
```

Analiza danych i przykładowe wykresy

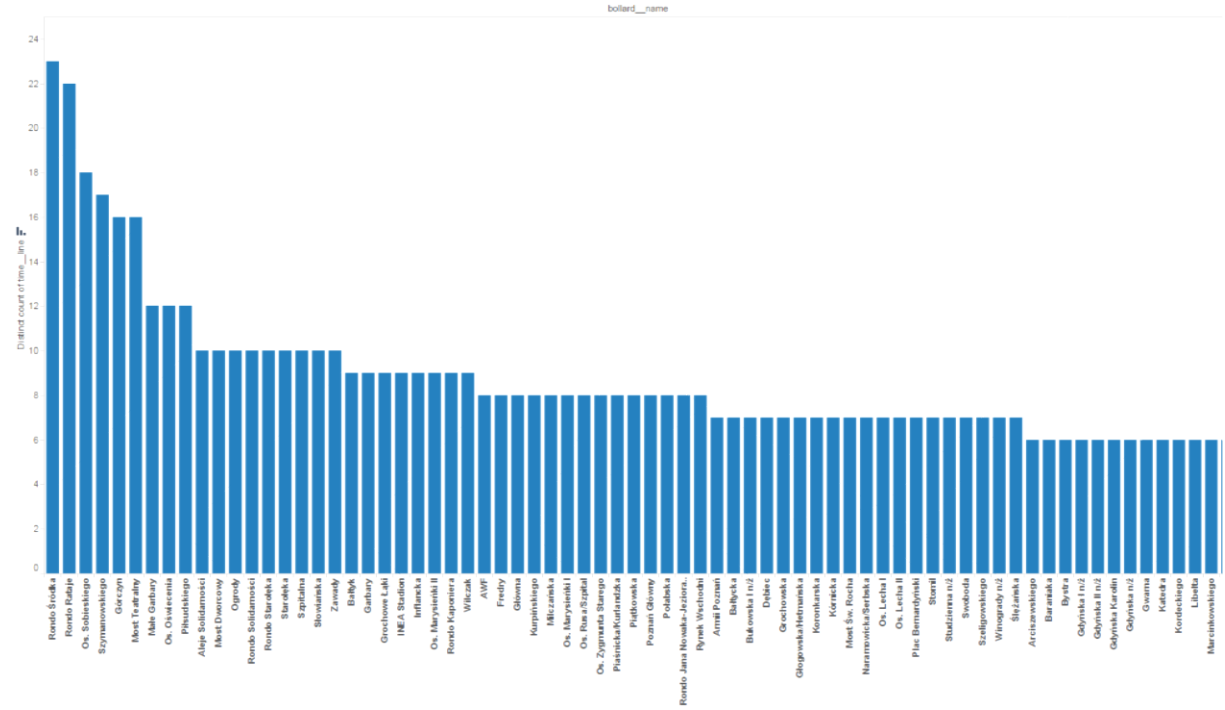
Do analizy danych wykorzystano narzędzie Tableau. Jest to nowoczesne narzędzie BI pozwalające na sprawne raportowanie oraz wizualizację danych z wielu źródeł. Poniżej zamieszczono przykładowe wykresy uzyskane za pomocą narzędzia Tableau. Do ich utworzenia wykorzystano zebrane dane z okresu jednego tygodnia.



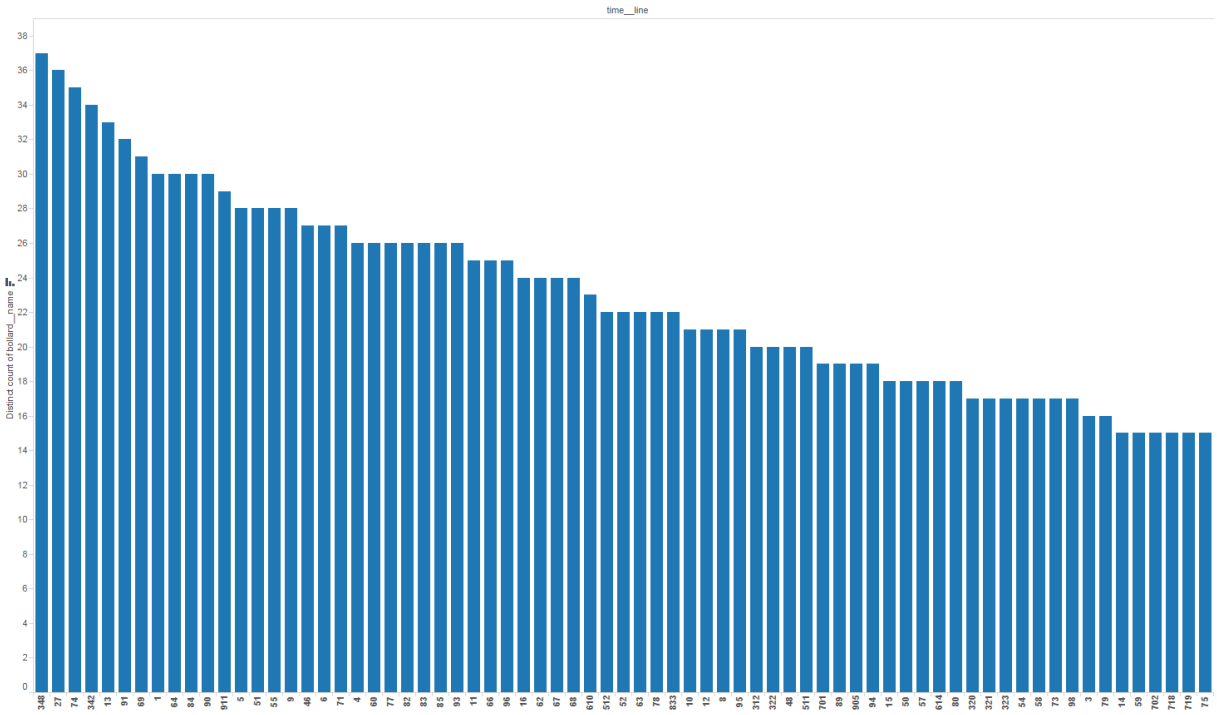
Wykres 1 - graf z kołami symbolizującymi linie tramwajowe lub autobusowe, im większe koło tym więcej wpisów w bazie dla danej linii.



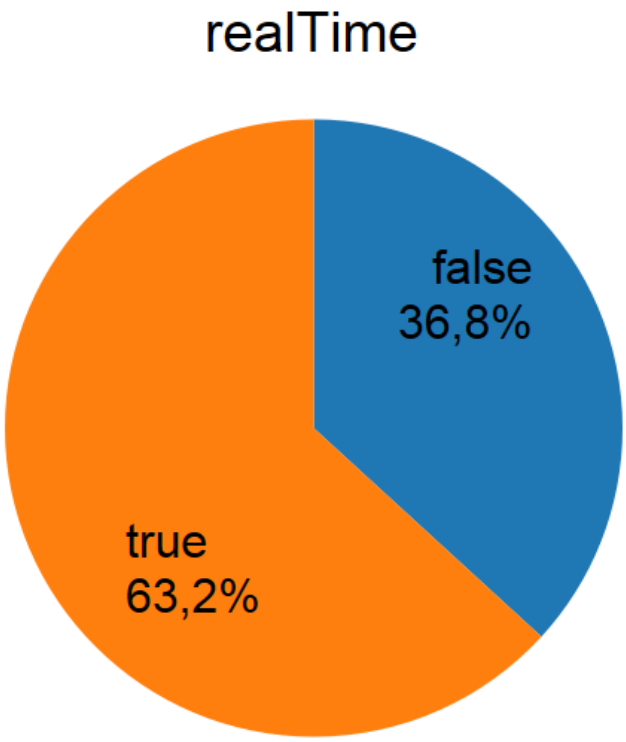
Wykres 2 – reprezentujący jednakowe dane jak wykres 1, ale posortowany malejąco. Widzimy, że najwięcej informacji udało się zebrać o linii 27.



Wykres 3 - obrazujący ile linii przebiega przez dany przystanek. Przez Rondo Śródka przebiega najwięcej (23) linii. Wedle oczekiwań Rondo Rataje oraz Śródka, na których znajdują się dworce autobusowe, są największymi pętlami komunikacyjnymi.



Wykres 4 – przedstawiający ile przystanków ma dana linia. 348 ma aż 37 przystanków (informacja potwierdzona poprzez skonfrontowanie wyniku z danymi z serwisu jakdojade.pl)



Wykres 5 – porównanie ile spośród zebranych logów ma wartość realTime ustawioną na false, a ile na true. Te posiadające wartość true zawierają dodatkowo informacje o minutach do odjazdu.