Architektura aplikacji

System powiadomień

Kamil Szarek, 21 listopada 2013

Plan prezentacji

1. Internet i aplikacje mobilne
2. Jak działa typowe API, architektura *pull*
3. Architektura *push*, PubSubHubbub
4. Problemy z PubSubHubbub i jak sobie z nimi radzić
5. Implementacja systemu powiadomień USOSapi

Internet w 2013

- Użytkownicy
  - 38.8% całej populacji
  - 76.8% w krajach rozwiniętych
- Trzech użytkowników z dostępem bezprzewodowym na jednego z dostępem przewodowym
  - Przyczyna (przynajmniej częściowa): smartfony

Aplikacje mobilne

Table 2. Mobile App Store Revenue, Worldwide, 2011-2017 (Millions of Dollars)

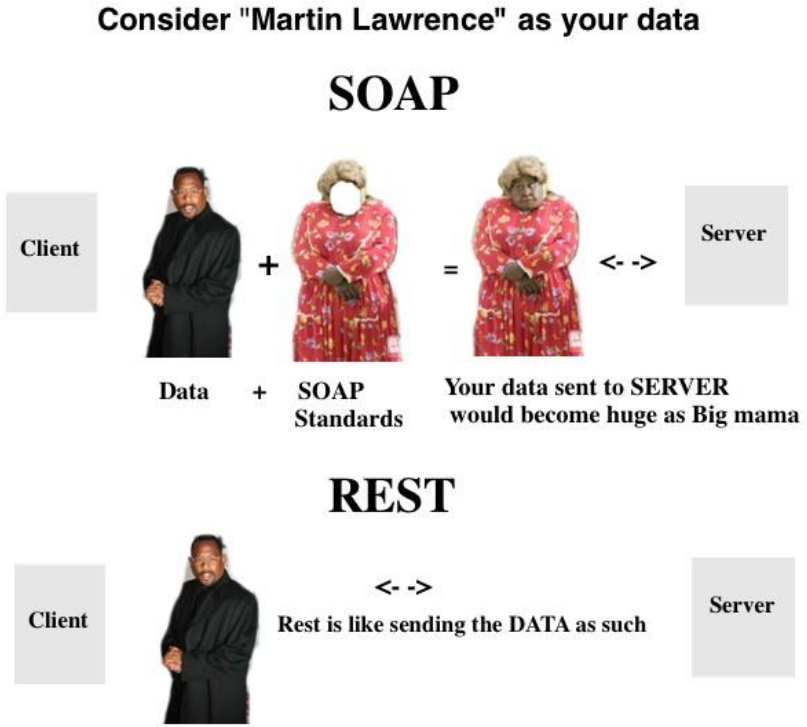
Downloads	2011	2012	2013	2014	2015	2016	2017
Paid-for	7,139	15,375	20,240	24,314	26,990	27,664	28,935
In-app purchases	712	2,111	4,591	7,856	14,001	23,771	36,887
Advertising	467	1,073	1,851	2,819	4,375	6,772	10,694
Total revenue (millions of dollars)	**8,318**	**18,559**	**26,683**	**34,988**	**45,366**	**58,207**	**76,517**
Paid-for (%)	85.8%	82.8%	75.9%	69.5%	59.5%	47.5%	37.8%
In-app purchases (%)	8.6%	11.4%	17.2%	22.5%	30.9%	40.8%	48.2%
Advertising (%)	5.6%	5.8%	6.9%	8.1%	9.6%	11.6%	14.0%

Source: Gartner (September 2013)

Aplikacje mobilne

- Samodzielne aplikacje (np. gry)
- Aplikacje korzystające z zasobów dostępnych przez Internet
  - API - *application programming interface*
    - Protokół komunikacji z backendem (baza danych, dodatkowa logika itp.)
    - Typowo HTTP + SOAP / REST

SOAP kontra REST

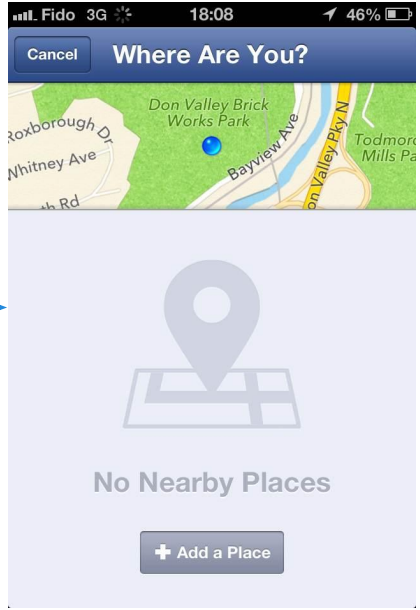


REST

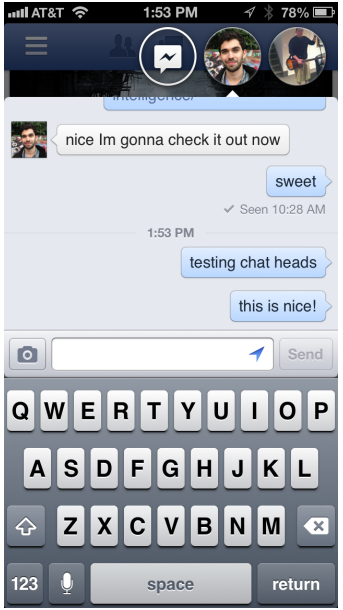
- Definiujemy tzw. zasoby identyfikowane przez URL
  - http://www.example.com/users/ - wszyscy użytkownicy
  - http://www.example.com/users/1/ - użytkownik o ID 1
- Wykonujemy żądania HTTP pod adresy zasobów
- Metoda HTTP oraz (potencjalna) treść definiują operację
  - GET, PUT, POST, DELETE

Komunikacja

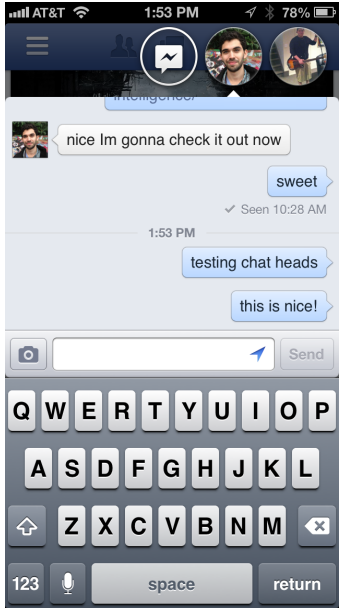
Uruchomienie aplikacji, pobranie danych



Nowa zakładka, pobranie danych



Wysłanie wiadomości, wysłanie danych



Odświeżenie wiadomości, pobranie danych

Architektura *pull*

- Pobieranie danych na życzenie klienta
- Klient zawsze inicjuje komunikację
- Nie pasuje do pewnych sytuacji
  - Powiadomienia

Architektura *push*

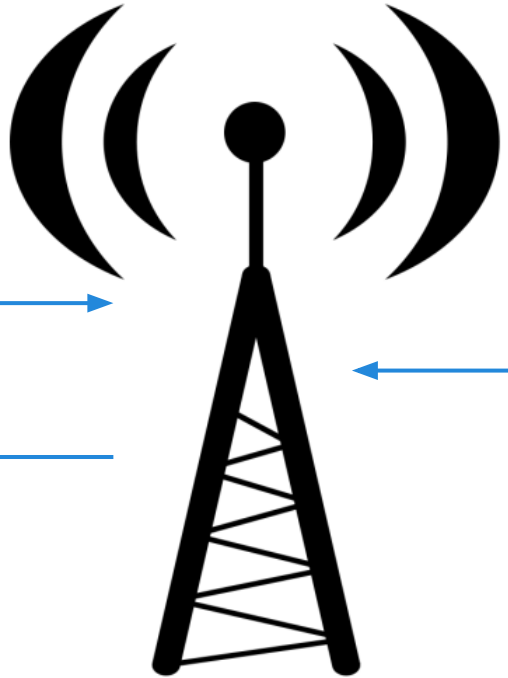
- Dopełnienie architektury *pull*, nie zastępstwo
- API inicjuje komunikację z klientem
- Potrzebne pewne założenia

SMS

1. Uruchomienie

2. Wyszukiwanie sieci

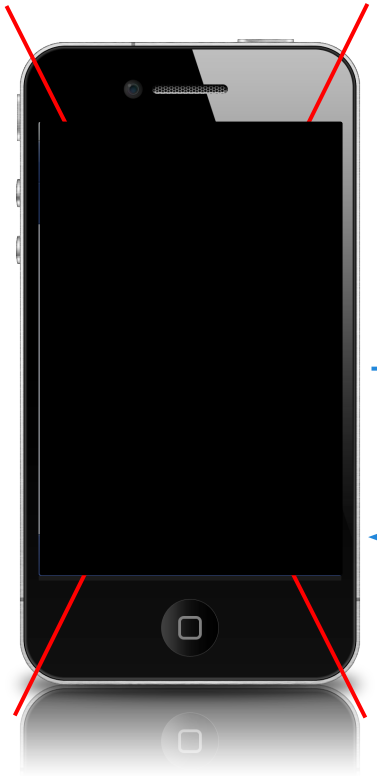
3. Rejestracja w sieci



4. SMS

5. SMS trafia do telefonu

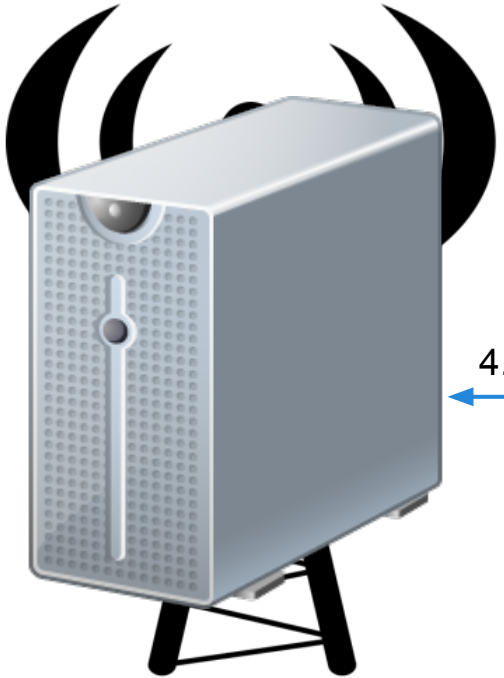
Powiadomienie



1. Uruchomienie aplikacji

2. Wyszukiwanie możliwych subskrypcji

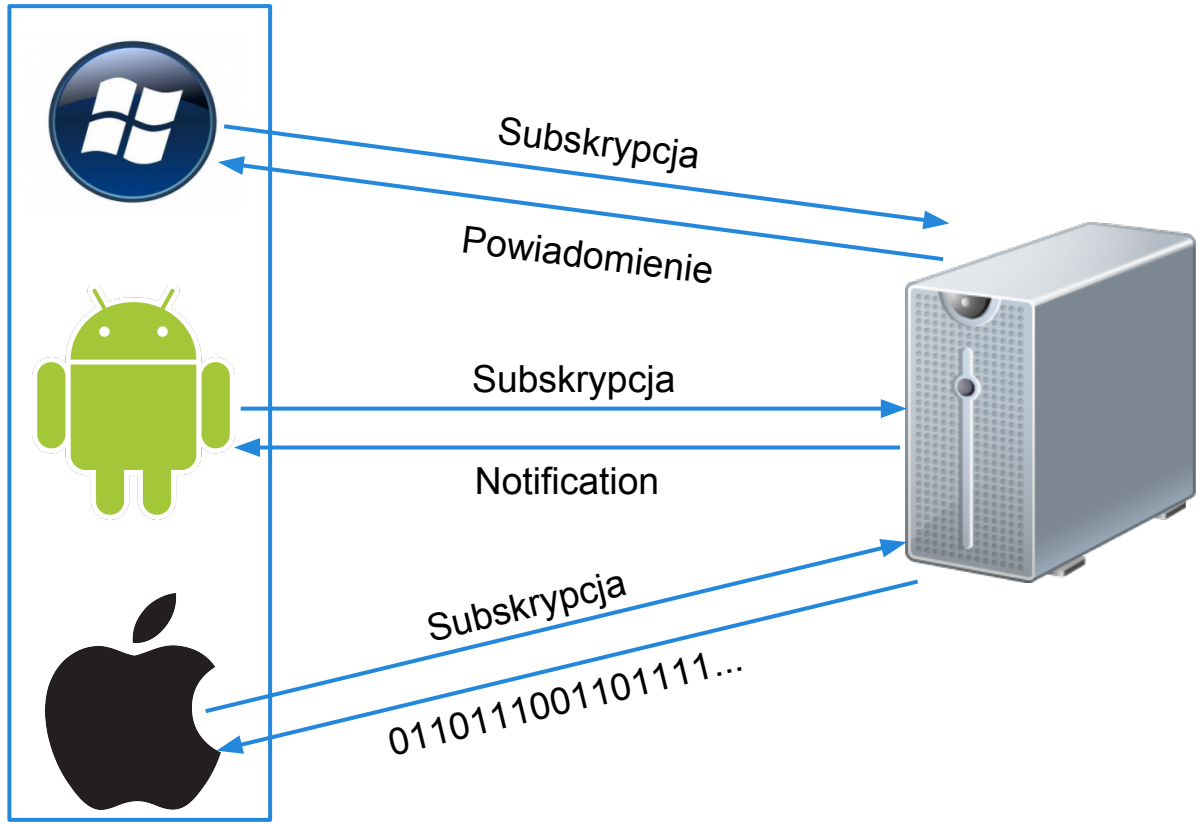
3. Subskrypcja powiadomień



4. Zdarzenie

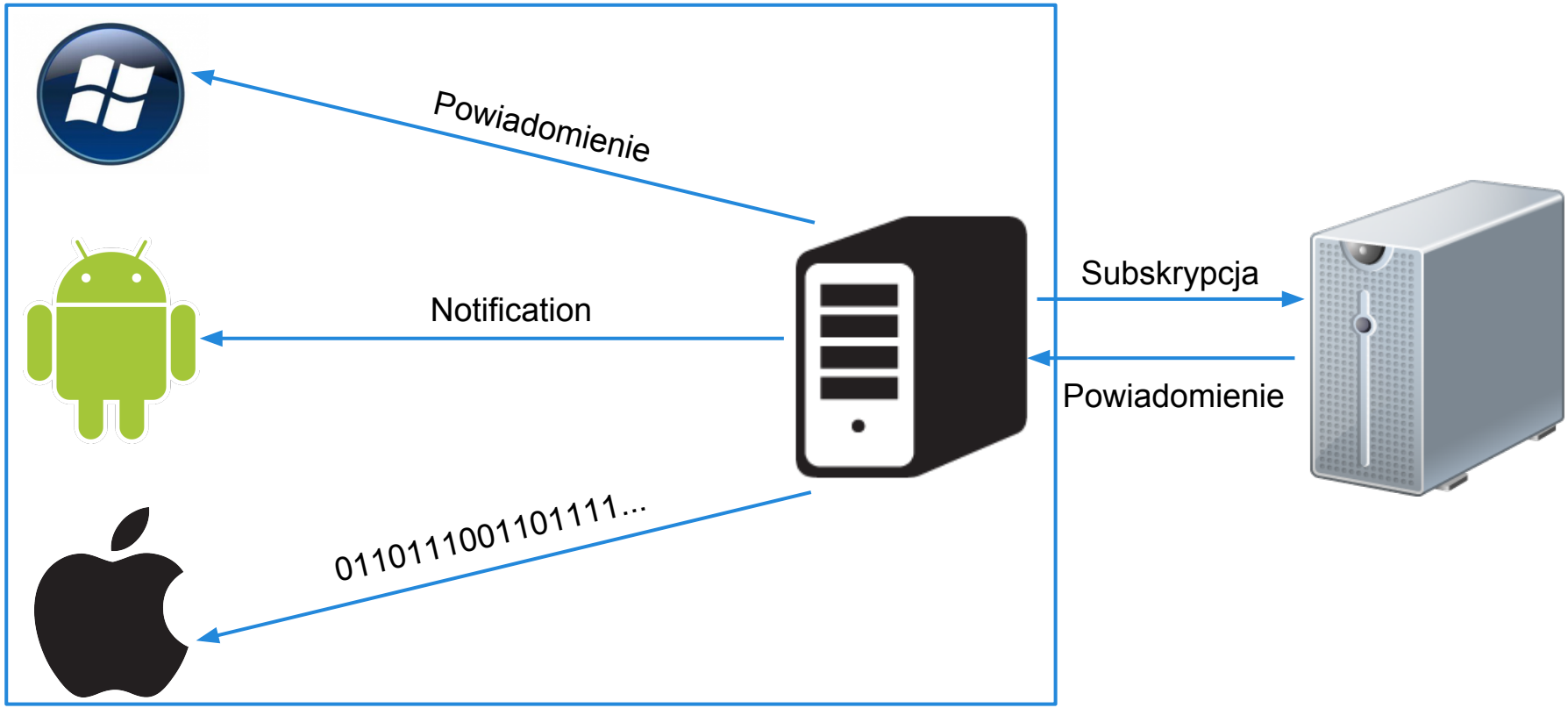
5. Powiadomienie trafia do aplikacji

Powiadomienie



ta sama aplikacja,
różne platformy

Powiadomienie



PubSubHubbub

- Pub - *Publisher*
- Sub - *Subscriber*
- Hub - *Hub*
  - Zarządza subskrypcjami
  - Odbiera informacje o zdarzeniach
  - Wysyła powiadomienia
- Hubbub - pol. zamieszanie, poruszenie

PubSubHubbub

1. Publisher w nagłówkach zasobu podaje URL Huba
2. Subscriber pobiera zasób w standardowy sposób
3. Subscriber widzi nagłówek-URL Huba
4. Jeśli chce subskrybować informacje o zmianie zasobu, wykonuje odpowiednie żądanie HTTP do Huba
5. Hub weryfikuje żądanie subskrypcji
6. Publisher informuje Hub o zmianie zasobów
7. Hub rozsyła powiadomienie (jako żądanie HTTP) do wszystkich Subscriberów zasobu

PubSubHubbub - szczegóły

- Weryfikacja adresu URL Subscribera
  - Żądanie HTTP pod adres z dodatkowymi parametrami
    - hub.challenge - weryfikacja, czy serwer subskrybenta obsługuje protokół
    - hub.topic - weryfikacja, czy żądanie subskrypcji pochodzi od faktycznie zainteresowanego subskrybenta
- Weryfikacja powiadomienia
  - hub.secret
  - HMAC, SHA1(treść żądania)
  - X-Hub-Signature
- Hub własny lub otwarty

PubSubHubbub - problemy

- Autoryzacja, poziomy uprawnień
  - OAuth
- REST
- Bezpieczeństwo przesyłu powiadomienia
- Rozwiązanie: własny Hub, adaptacja protokołu

OAuth

- Standard autoryzacji i udzielania uprawnień klientom w imieniu użytkowników
- W uproszczeniu
  - Klient otrzymuje klucz publiczny i prywatny, współdzielony z API
  - Klient może poprosić API o to, by poprosiło (!) w jego imieniu o uprawnienia użytkownika
  - Jeśli użytkownik się zgodzi, klient otrzymuje żeton dostępowy dla użytkownika
  - Zapytania o wrażliwe dane muszą być podpisywane tym żetonem

Facebook Real-time Updates

- Subskrybujemy zmiany wybranych pól w jednej z dostępnych encji
  - Nie ma URL zasobu
- OAuth
  - Jeśli użytkownik nie autoryzował klienta, to klient-subskrybent nie dostanie powiadomienia
- Bezpieczeństwo przesyłu powiadomienia
  - W treści nie ma wrażliwych danych

Facebook Real-time Updates

- Przykładowe powiadomienie
  - Grupowanie powiadomień
  - Parametry niezbędne, by pobrać dane

```
{
  "object": "user",
  "entry": [
    {
      "uid": 1335845740,
      "changed_fields": [
        "name",
        "picture"
      ],
      "time": 232323
    }, {
      "uid": 1234,
      "changed_fields": [
        "friends"
      ],
      "time": 232325
    }
  ]
}
```

Inne serwisy

- Flickr
  - Bliżej do PubSubHubbub
  - Tzw. strumienie zdjęć jako zasoby
  - Atom 1.0
- Foursquare
  - Powiadomienia mogą zawierać wrażliwe dane
  - Wymagane HTTPS
- Instagram
  - Jak Facebook

USOSapi

- API do USOS :)
- Python, Django
- Metody, nie zasoby
- OAuth
  - Poziomy uprawnień
  - Klienci administracyjni

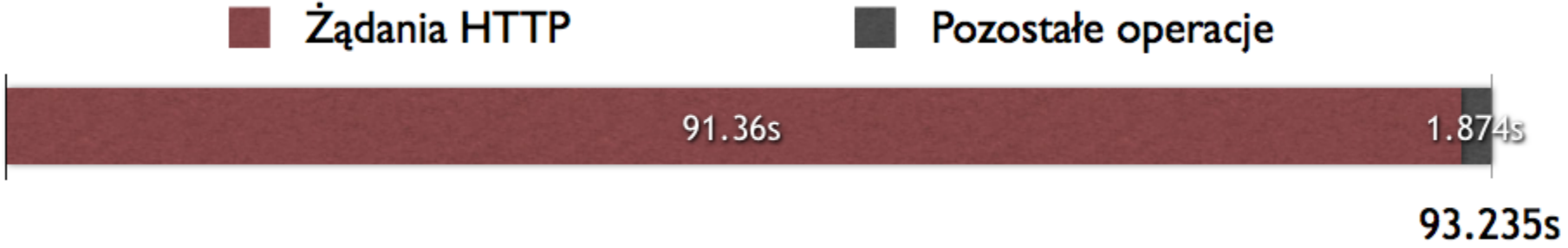
System powiadomień USOSapi

- Subskrypcje zmian wyników metod dla parametrów
  - Np. zmienia się liczba punktów za zadanie 123 dla użytkownika 321
  - Powiadomienie zawiera nazwę metody (“typ zdarzenia”) i parametry
  - OAuth i użytkownicy związani ze zdarzeniem
- Nowe metody do zarządzania subskrypcjami
- Rozsyłacz powiadomień

Rozsyłacz powiadomień USOSapi

1. Pobierz nieprzetworzone zdarzenia (z bazy danych)
2. Dla każdego zdarzenia
    a. Zdeserializuj do obiektu
    b. Pobierz subskrypcje założone przez klientów dla tego typu zdarzenia
    c. Dla każdej subskrypcji
        i. Sprawdź uprawnienia klienta, jeśli to konieczne
        ii. Wyślij powiadomienia, jeśli nie ma problemu z uprawnieniami
3. Usuń przetworzone zdarzenia

Profilowanie rozsyłacza

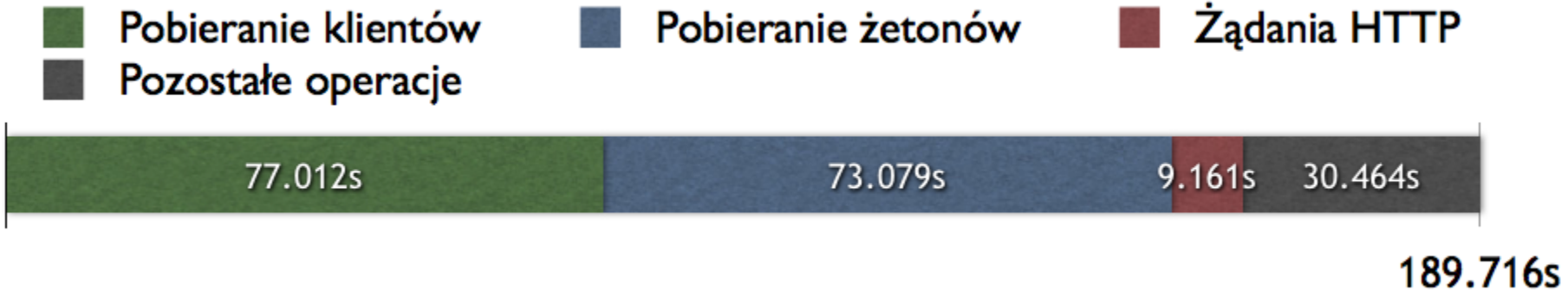


- Python, cProfile
- 3 klientów, 10 użytkowników, 60 zdarzeń
- Problem: zbyt dużo żądań HTTP

Grupowanie powiadomień

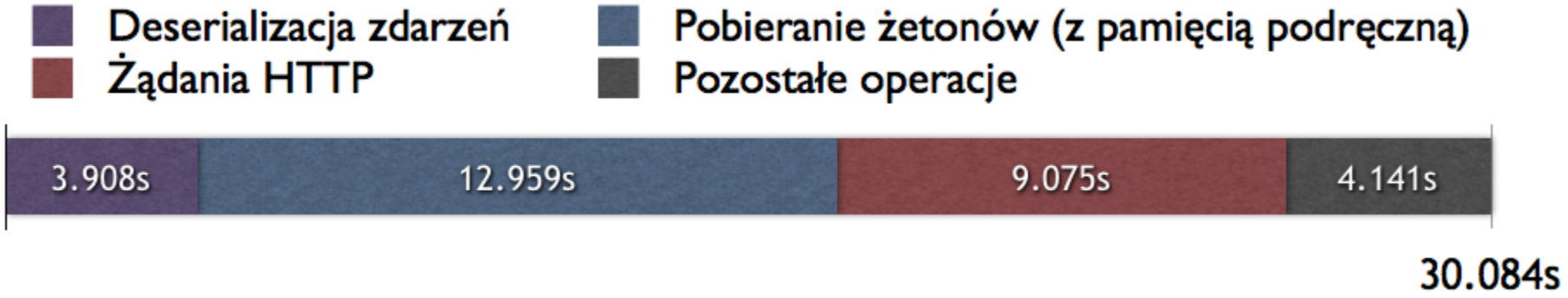
- 3 klientów, 10 użytkowników, 60 zdarzeń
- Kolejkowanie powiadomień i wysyłanie w grupach
- 28 razy szybciej

Grupowanie powiadomień



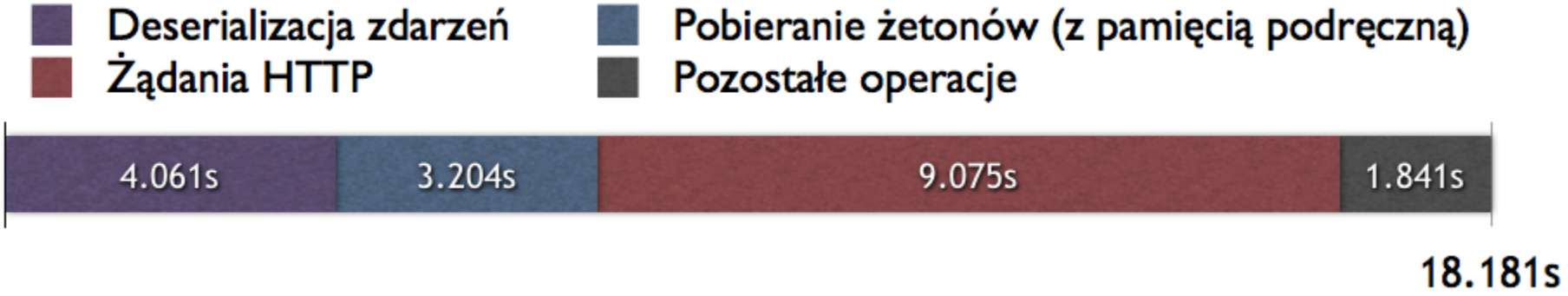
- 3 klientów, 1 000 użytkowników, 6 000 zdarzeń
- Problem: pobieranie danych klientów/żetonów

Cache klientów i żetonów



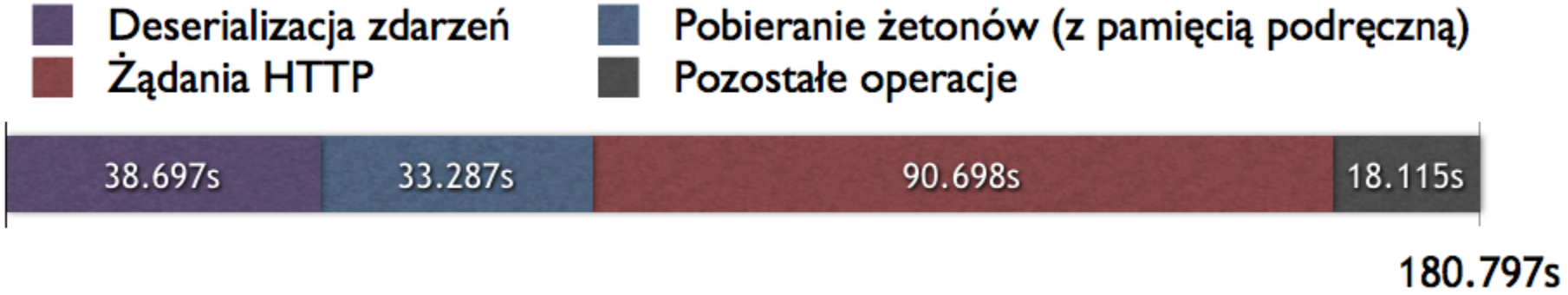
- 3 klientów, 1 000 użytkowników, 6 000 zdarzeń
- Proste zapamiętywanie obiektów w pamięci
- 6 razy szybciej
- Problem: nadal pobieranie żetonów

Bezpośrednie zapytania SQL



- 3 klientów, 1 000 użytkowników, 6 000 zdarzeń
- Rezygnacja z ORM
- 1.6 raza szybciej

Bezpośrednie zapytania SQL

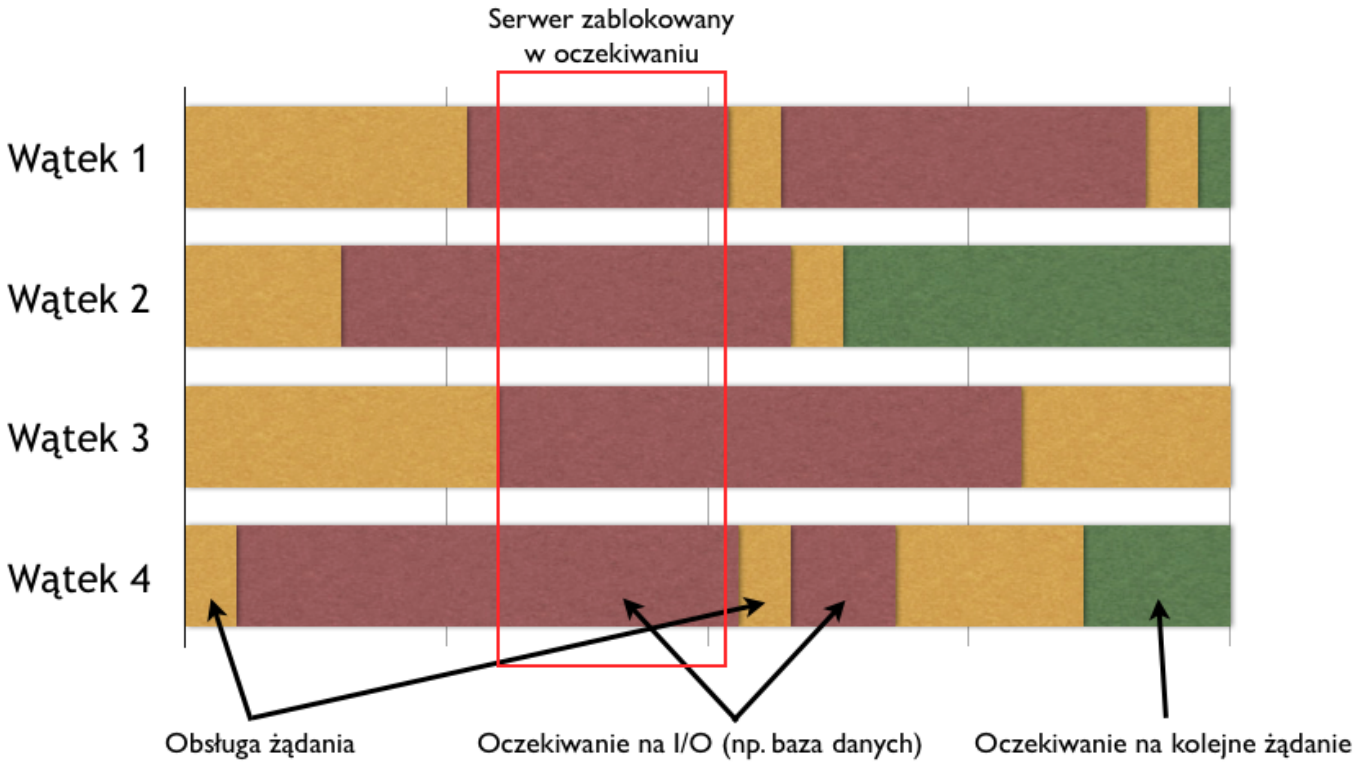


- 3 klientów, 10 000 użytkowników, 60 000 zdarzeń
- Problem: znowu żądania HTTP

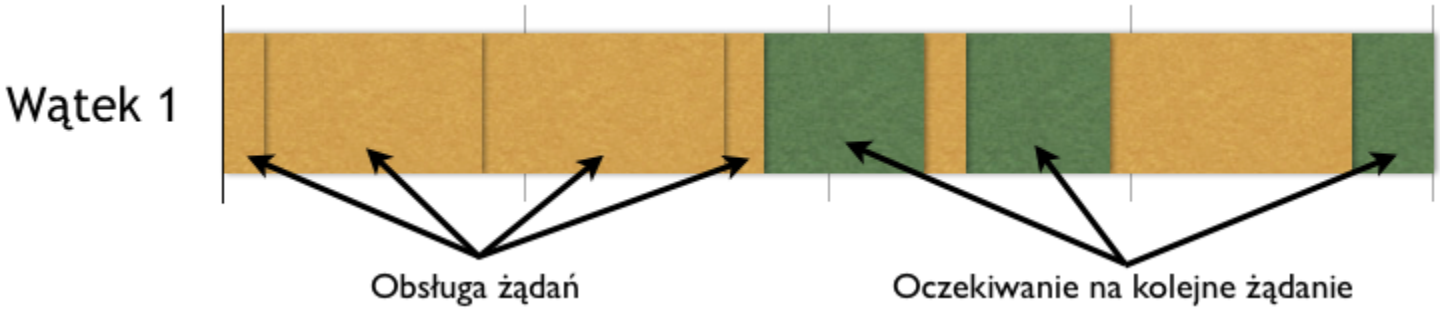
Szybkie wykonywanie żądań

- Wątki?
  - *Global Interpreter Lock* w CPython
  - Procesy?...
- Dualna operacja: obsługa żądań
  - Problem C10K
  - Operacje wejścia-wyjścia są **bardzo** kosztowne w porównaniu do innych

Synchroniczny serwer HTTP



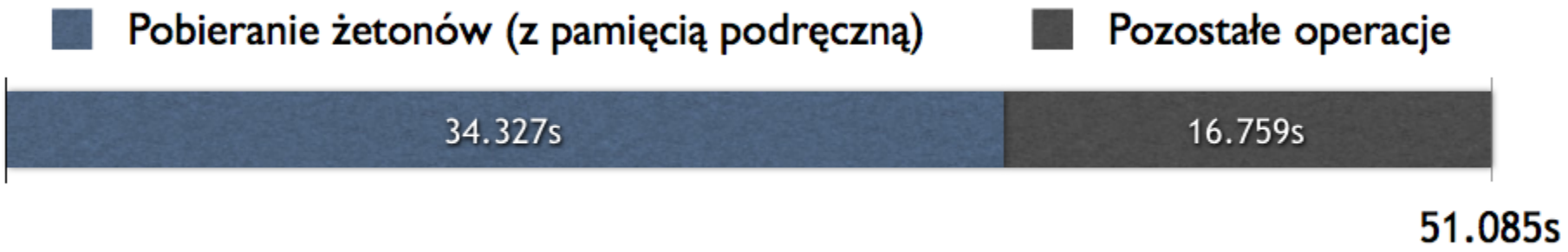
Asynchroniczny serwer HTTP



Asynchroniczne żądania HTTP

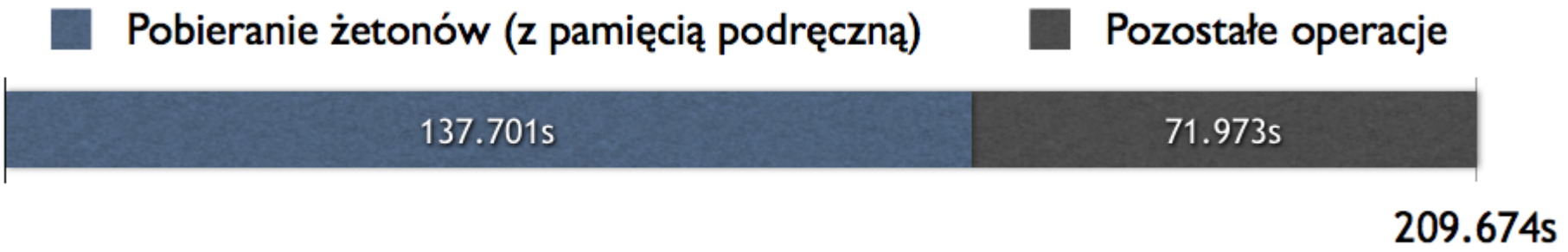
- 3 klientów, 10 000 użytkowników, 60 000 zdarzeń
- Tornado
- Prawie 2 razy szybciej
- Problem: deserializacja zdarzeń (Python, pickle)

Lepszy model danych



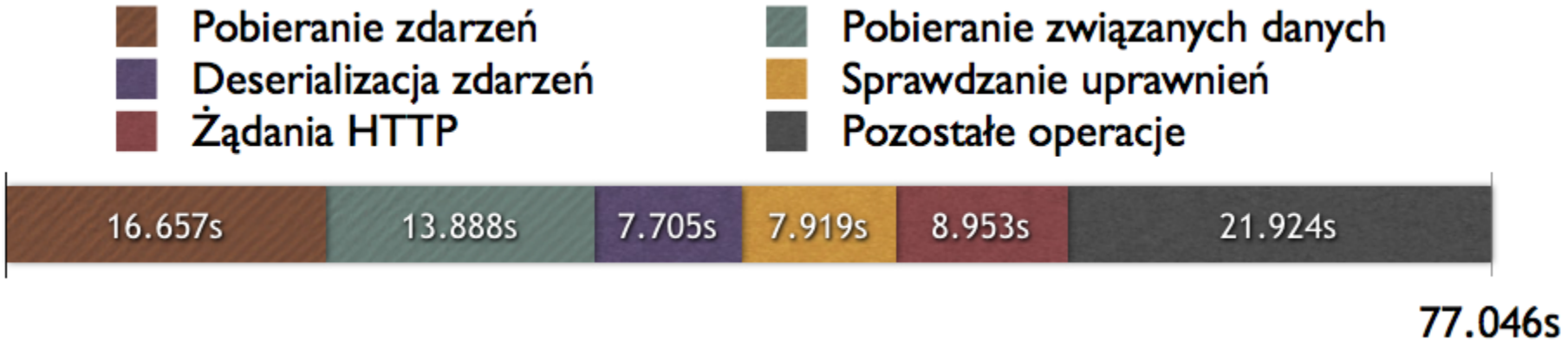
- 3 klientów, 10 000 użytkowników, 60 000 zdarzeń
- JSON zamiast zserializowanego obiektu w bazie
- 1.8 raza szybciej

Lepszy model danych



- 3 klientów, 40 000 użytkowników, 250 000 zdarzeń
- Problem: znowu pobieranie żetonów

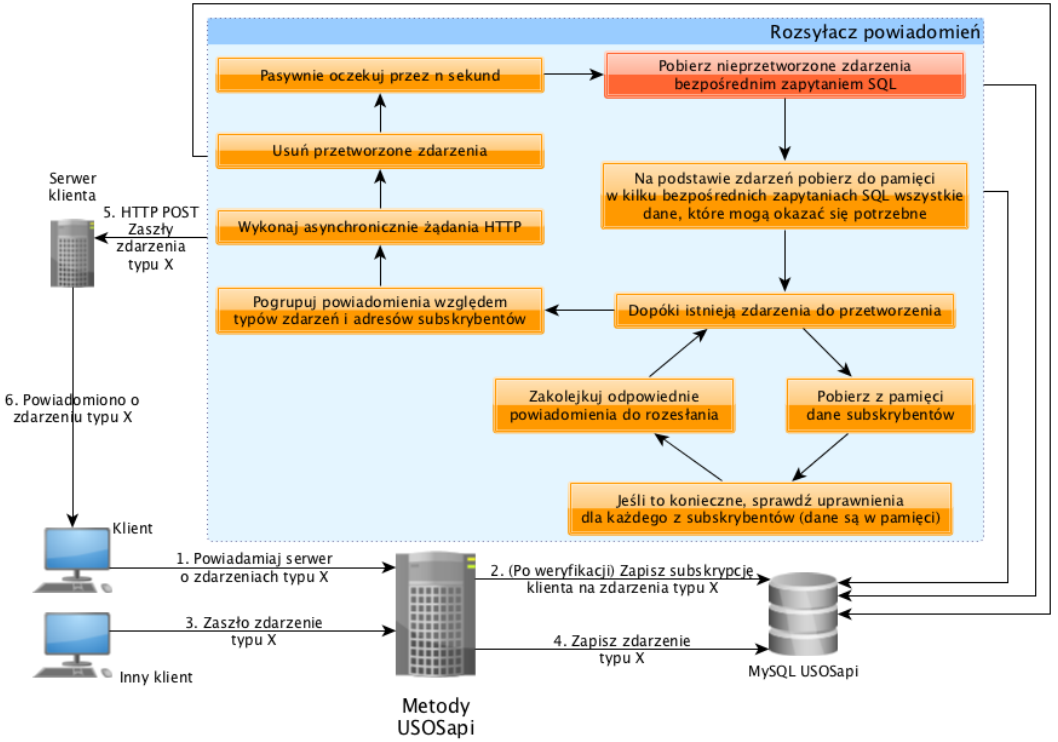
Preprocessing



- 3 klientów, 40 000 użytkowników, 250 000 zdarzeń
- Pobranie do pamięci wszystkich potrzebnych danych jednym zapytaniem
- 2.7 raza szybciej

System powiadomień USOSapi

- Początek: 60 zdarzeń dot. 10 użytkowników w 94 sekundy
- Po poprawkach: 250 000 zdarzeń dot. 40 000 użytkowników w 78 sekund
- USOS: 247 685 zmian ocen dot. 40 248 użytkowników w czerwcu 2013



Pomysły i ciekawostki

- Klasy zamiast typów zdarzeń
- Filtrowanie powiadomień
- Twitter
  - Strumienie danych przez HTTP
  - Konieczność podtrzymywania połączenia

Kilka odnośników

- Statystyki ITU - http://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx
- Raport Gartner - http://www.gartner.com/newsroom/id/2592315
- SOAP kontra REST - http://stackoverflow.com/questions/209905/representational-state-transfer-rest-and-simple-object-access-protocol-soap/
- REST - http://en.wikipedia.org/wiki/REST
- PubSubHubbub - https://pubsubhubbub.googlecode.com/git/pubsubhubbub-core-0.4.html
- OAuth - http://tools.ietf.org/html/rfc5849
- Facebook Real-time Updates - http://developers.facebook.com/docs/reference/api/realtime/
- USOSapi - http://usosapps.uw.edu.pl/developers/api/
- C10K problem - http://www.kegel.com/c10k.html
- Tornado - http://www.tornadoweb.org/en/stable/
- Twitter Streaming API - https://dev.twitter.com/docs/streaming-apis

Dziękuję za uwagę