AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

Obliczenia rozproszone w języku Haskell
Distributing tasks with Haskell

Autor: *Konrad Lewandowski*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Piotr Matyasik*

Kraków, 2017

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wprowadzenie

W czasach kiedy przewidywania Gordona Moore'a dotyczące dalszego wzrostu mocy obliczeniowej pojedynczych komputerów przestają się sprawdzać, coraz częściej wykorzystujemy metody wykonywania programów oparte na jednoczesnym przetwarzaniu rozproszonym na wielu połączonych ze sobą maszynach.

Pomimo pozornej prostoty takiego rozwiązania istnieje bardzo niewiele narzędzi ułatwiających programowanie w modelu rozproszonym.

1.1. Cel i założenia pracy

Celem poniższej pracy jest implementacja w języku Haskell podstawowej biblioteki do rozpraszania zadań na wielu komputerach z wykorzystaniem brokera RabbitMQ, umożliwiającą zlecanie zadań do wykonania, przerywanie zadań będących w trakcie wykonywania, raportowanie na bieżąco postępów wykonania, przesyłania wyników oraz wprowadzanie zależności pomiędzy zadaniami.

1.2. Istniejące rozwiązania

Poniższa lista zawiera skrócony opis istniejących bibliotek programistycznych, służących do obliczeń rozproszonych:

1.2.1. Celery

Celery jest asynchroniczną kolejką zadań opartą o rozproszone komunikaty przesyłane między komputerami, napisaną w języku Python. Działa w czasie rzeczywistym, jednak umożliwia również szeregowanie zadań. Interfejs programistyczny pozwala na zlecanie zadań zarówno w sposób synchroniczny jak i asynchroniczny, oraz przesyłanie wyników. Wspiera wiele brokerów wiadomości (np. RabbitMQ, Redis, MongoDB).

1.2.2. Resque

Resque jest biblioteką języka Ruby, wykorzystującą bazę Redis w charakterze brokera wiadomości. Umożliwia asynchroniczne zlecanie powtarzalnych zadań na innych komputerach.

Biblioteka ta jest często używana przez programistów serwisów internetowych do wykonywania długotrwałych operacji (np. generowanie miniaturek zdjęć, rozsyłanie newslettera e-mail, tworzenie raportów, etc...)

1.2.3. Cloud Haskell

Cloud Haskell to biblioteka języka Haskell, udostępniająca warstwę transportową do komunikacji między węzłami, mechanizm serializacji domknięć pozwalający na zdalne uruchamianie procesów oraz API do programowania rozproszonego.

1.3. Zawartość pracy

W rozdziale 1 cośtamcośtam...

2. Programowanie rozproszone

Programowanie rozproszone to styl programowania umożliwiający [TODO]

2.1. Architektura

2.1.1. Klient-serwer

W tym podejściu oprogramowanie klienckie łączy się z serwerem celem pobrania danych wejściowych, a po ich przetworzeniu odsyła wyniki z powrotem

2.1.2. Peer-to-peer

2.2. Modele obliczeń rozproszonych

2.2.1. Zcentralizowany

(jeden węzeł nadzoruje pracę innych)

2.2.2. Równoległy

(jeden węzeł wykonuje to samo zadanie na wielu komputerach)

3. Specyfikacja projektu

3.1. Architektura

Projekt składa się z dwóch części - oprogramowania instalowanego na każdym węźle (zawierającego statycznie skompilowaną bibliotekę możliwych do uruchomienia zadań) oraz biblioteki klienckiej pozwalającej na uruchamianie zadań z poziomu kodu użytkownika.

3.1.1. Oprogramowanie węzła

Oprogramowanie węzła ma za zadanie połączyć się ze skonfigurowanym wcześniej brokerem a następnie nasłuchiwać komunikatów zlecenia lub przerwania zadania.

3.1.2. Biblioteka kliencka

Biblioteka kliencka umożliwia łączenie się z brokerem wiadomości a następnie wysyłanie komunikatu zlecającego lub przerywającego zadanie w odpowiednim, ustalonym wcześniej formacie

3.2. Sekwencja zdarzeń

3.3. Konfiguracja

Oprogramowanie węzła wymaga pliku konfiguracyjnego taskell.conf w następującym formacie:

```
taskell {
  rabbitmq {
    host     = "localhost"
    vhost    = "/"
    username = "guest"
    password = "guest"
  }
  abortExchange = "taskell.abort"
  parallelism = 1
```

```
  queues = ["taskell.q1", "taskell.q2"]
}
```

taskell.rabbitmq.host – Nazwa sieciowa lub adres IP brokera

taskell.rabbitmq.vhost – URL hosta wirtualnego jeśli na fizycznej maszynie działa więcej niż jeden broker

taskell.rabbitmq.username – Nazwa użytkownika skonfigurowana na brokerze

taskell.rabbitmq.password – Hasło użytkownika skonfigurowana na brokerze

taskell.abortExchange – Nazwa wymiennika do którego każdy węzeł przypina swoją kolejkę celem nasłuchu zleceń przerwania zadana

taskell.queues – Lista kolejek, które węzeł ma nasłuchiwać w oczekiwaniu na zadanie

taskell.parallelism – Liczba zadań, które węzeł może przetwarzać jednocześnie

3.4. Serializacja zadania

3.5. Wątek nadzorujący

3.6. Śledzenie zatrzymań

3.7. Raportowanie wyników

4. Język Haskell

4.1. Programowanie funkcyjne

4.2. Podstawowe cechy języka

4.3. Opis zastosowanych bibliotek

5. RabbitMQ

RabbitMQ to otwartoźródłowy broker wiadomości - oprogramowanie zapewniające odporny na zakłócenia mechanizm komunikacji sieciowej. Został zaimplementowany w języku Erlang z wykorzystaniem Open Telecom Platform. Posiada biblioteki dla wszystkich znaczących języków programowania - w tym dla Haskella (biblioteka amqp - od ang. Advanced Message Queuing Protocol). Dwie najważniejsze koncepcje implementowane przez RabbitMQ to - **kolejki** (ang. queues) oraz **wymienniki** (ang. exchanges). Kolejki to przetrzymywane przez serwer struktury FIFO z sieciowym protokołem dostępu. Są najbardziej podstawową abstrakcją stosowaną w rozproszonym programowaniu współbieżnym, a w przypadku projektu będącego przedmiotem tej pracy pozwalają wielu węzłom ”konkurować” o zadania w bezpieczny sposób (nigdy nie dojdzie do sytuacji, w której dwa zadania są w tym samym czasie wykonywane na dwóch różnych węzłach). Wymienniki są pośrednikami dostępu do zbiorów kolejek - wiadomość, która zostanie dodana do wymiennika może zostać powielona i rozdystrybuowana do połączonych z wymiennikiem kolejek w jeden z czterech możliwych sposobów zależnie od predefiniowanego typu wymiennika:

direct – wiadomość trafia tylko do kolejek z określonym kluczem routującym, identycznym z kluczem zawartym w nagłówku wiadomości

fanout – wiadomość trafia do wszystkich podłączonych kolejek

topic – wiadomość trafia do konkretnej kolejki tylko wtedy, kiedy jej klucz routujący spełnia wzorzec określony dla tej kolejki

headers – wiadomość trafia do konkretnej kolejki na podstawie innych niż klucz routujący parametrów zawartych w nagłówku