Silnik rekomendacyjny do filmów, zespół nr 1

1. Opis analizy problemu

Celem naszego projektu jest wykonanie silnika rekomendacyjnego z wykorzystaniem narzędzi stosowanych w celu rozwiązania problemów o skali Big Data.

Biorąc pod uwagę skalę projektu zdecydowaliśmy się na implementację silnika typu content-based filtering (opartego na porównywaniu podobieństwa dostępnych treści i preferencji kinematograficznych danego użytkownika). Program docelowy przyjmie na wejściu listę filmów, które spodobały się użytkownikowi, i co do których chciałby on obejrzeć coś możliwie podobnego, np. w liście znalazł się jedynie "Hobbit", to zakładamy, że aplikacja powinna polecić filmy z cyklu "Władcy Pierścieni". Jako że by móc odnaleźć coś "podobnego", potrzebna jest referencja do czegoś, względem czego tego podobieństwa szukamy. Pusta lista na wejściu zostanie więc uznana za nieprawidłowy argument, co stanowi rozwiązanie problemu zimnego startu poruszonego w komentarzu do poprzedniego raportu.

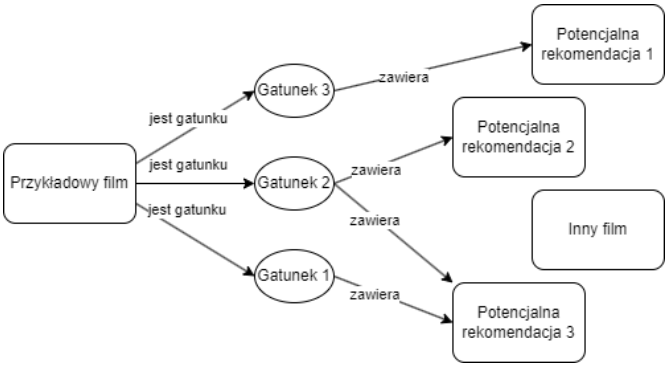
Oprócz skuteczności, chcieliśmy, aby nasze narzędzie cechowała aktualność, dlatego sami zebraliśmy dane z publicznych API; oraz możliwie szybko uzyskiwany wynik aktualnej rekomendacji, dlatego zdecydowaliśmy się na zastosowanie bazy grafowej jako serving layer.

Ostateczny wynik podobieństwa między filmami, na podstawie którego wybierane będą rekomendacje może zostać policzony z poniższego wzoru. Może on oczywiście ulec zmianie, jeśli ten okaże się nieoptymalny.

$$P = \frac{1}{G} * g + \frac{1}{A} * a + p$$

Gdzie:
P – wynik rekomendacji
g – ilość wspólnych gatunków (bo według TMDB film może należeć to więcej niż jednego)
G – waga podobieństwa gatunkowego, np. maksymalna ilość gatunków dla jednego filmu
a – ilość wspólnych aktorów
A – waga podobieństwa *obsadowego*, np. maksymalna ilość aktorów w jednym filmie (czyli w naszym przypadku 7 – zescrapowanie tylko 7 aktorów z każdego filmu i tak zapewniło nam blisko 200 tys. krawędzi w bazie)
p – wartość podobieństwa kosinusowego między fabułami jeśli zdecydujemy się umieszczać ją w bazie, 0 lub 1 w przeciwnym wypadku.

Wagi mają na celu znormalizowanie ważności każdej z trzech części wyniku – nie chcemy, żeby gatunki i aktorzy byli parę razy ważniejsi niż faktyczne podobieństwo fabuł. Podobieństwo to zostanie policzone dla każdego filmu, który zostanie zwrócony w wyniku trawersacji grafu w poniższy sposób, a wybrane zostanie N filmów o najwyższym wyniku.

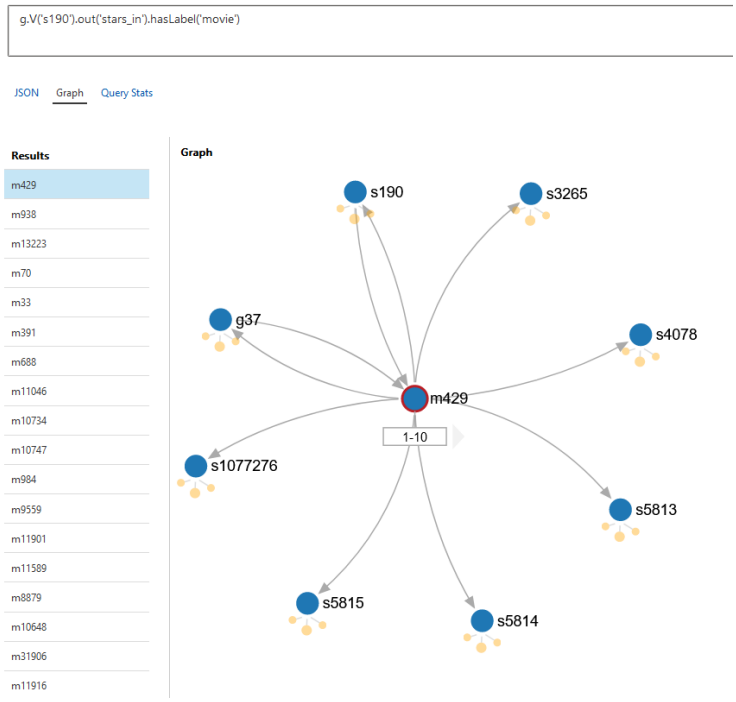


2. Co zostało wykonane, wykorzystana infrastruktura

Dane dotyczące filmów, gatunków i aktorów, a także streszczenia fabuł zostały zescrapowane z publicznych API zapewnianych przez portal TMDB oraz bibliotekę Cinemagoer, które dokładniej opisaliśmy w poprzednim raporcie. Sposób pobierania przechowywania i przesyłania dalej opisany w następnym punkcie.

Scrapowanie okazało się całkiem czasochłonnym procesem, dlatego wszystkie scrapery zostały napisane w taki sposób, że są odporne na przerwania i zaczynanie pracy od momentu, na którym skończyły. Fabuły zostały zapisane do osobnych plików tekstowych z myślą o przetwarzaniu we frameworku MapReduce. Dodatkowo scraper fabuł działa wielowątkowo, co znacznie (o rząd wielkości) przyspiesza jego działanie.

Postawiona została baza grafowa Cosmos DB for Gremlin (oparta o Apache TinkerPop). Zdecydowaliśmy się na to rozwiązanie ze względu na prosty język zapytań oraz na darmowy tier Storage Account w chmurze Azure. Nie jest to niestety rozwiązanie bez wad, o czym więcej we wnioskach. Poniżej dowód działania bazy: zapytanie zwracające wszystkie filmy, w których grał Clint Eastwood.



Zrzut ekranu z Data Explorera z Azure Portalu, jako że jeszcze nie mamy interfejsu

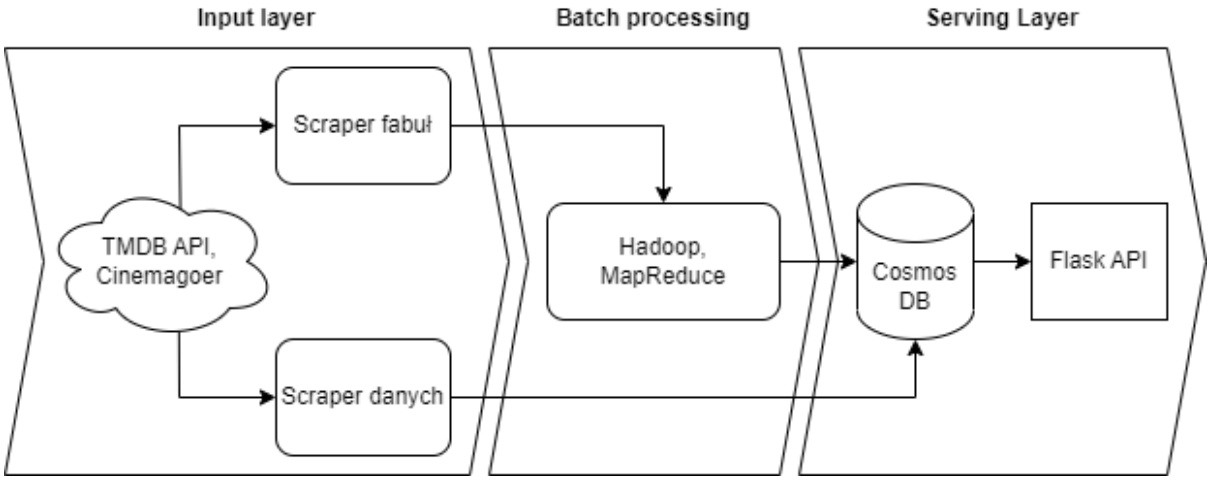
Warto dodać, że w tym momencie baza jest już gotowa do wykonywania trawersacji. Krawędzie między filmami nie zostały jeszcze dodane, gdyż podobieństwo nie zostało jeszcze policzone, ale relacje potrzebne do policzenia podobieństwa gatunkowego i *obsadowego* już zostały wprowadzone, a to właśnie baza sama w sobie jest przecież silnikiem do przetwarzania grafów.

Korzystając z chmury Azure postawione zostały trzy maszyny wirtualne, a na nich skonfigurowany został Hadoop. Niestety nie mogliśmy skorzystać z serwisu HDInsight, gdyż nie jest dostępny w subskrypcji studenckiej. Maszyny postawiliśmy w chmurze zamiast lokalnie, by przetwarzanie mogło chodzić nieprzerwanie tyle czasu, ile będzie tego potrzebować, a przy okazji łatwiej tak wspólnie pracować na jednym środowisku. Poniżej lista węzłów dostępnych dla YARNa jako dowód poprawnej konfiguracji maszyn i Hadoopa.

```
mihawb@OBIEZYSWIAT → ssh hadoop@52.158.42.242 "/home/hadoop/hadoop/bin/yarn node -list"
2023-05-21 21:51:10,278 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at nn1/10.0.0.4:8032
Total Nodes:3
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
       dn1:39607                RUNNING          dn1:8042                                  0
       dn2:46729                RUNNING          dn2:8042                                  0
       nn1:45185                RUNNING          nn1:8042                                  0
```

Pierwsze próby otrzymania podobieństwa fabuł za pomocą MapReduce zostały podjęte. Przygotowana została implementacja wyliczająca podobieństwo dokumentów korzystająca z paradygmatu MapReduce, która w fazie redukcji wyodrębnia z każdego pliku pojedyncze słowa (z pominięciem wielkości liter i znaków interpunkcyjnych), a w fazie mapowania oblicza podobieństwo pomiędzy parami dokumentów na podstawie wyekstrahowanych słów. Przygotowany program działa, ale występują problemy ze stabilnością maszyn wirtualnych. Niemniej jednak, uzyskany został tutaj potencjalnie istotny wniosek - znacznie dłuższy czas przetwarzania przykładowego tekstu niż oczekiwany.

3. Potok przetwarzania



Scrapery oraz connector do bazy zostały napisane w taki sposób, by dane od razu były gotowe do załadowania w miejsce docelowe. Napisaliśmy także skrypty uruchamiające wszystko w odpowiedniej kolejności, te jednak trzeba uruchomić ręcznie.

Input layer można by rozszerzyć o czasowe sprawdzanie nowych produkcji w serwisie TMDB, można by też dodać automatyczne triggerowanie scrapowania oraz przetwarzania wsadowego. Wspominaliśmy o tym w poprzednim raporcie jako możliwości future-proofingu rozwiązania. Termin oddania projektu jednak zbliża się wielkimi krokami, a nam nie brakuje roboty, dlatego pozostawiamy te opcje jako możliwości rozwoju.

4. Wnioski

- Podobieństwo uzyskane z fabuł może okazać się nie do końca praktyczne - duże rozbieżności w długości opisów (od ponad 11000 słów do jedynie 3)
- Znacznie dłuższy czas przetwarzania przykładowego tekstu z wykorzystaniem MapReduce niż się spodziewaliśmy. Trzeba będzie więc możliwie wcześnie skończyć implementację jego działania, by móc rozpocząć proces jak najszybciej i móc doczekać wyników przed nadejściem terminu oddania projektu, lub odpowiednio okroić zbiór filmów – z 10 tys. do np. 3 lub 5 tys.
- Paradygmat mapowania i redukcji jest ciężki do zrównoleglenia dla obliczeń podobieństwa cosinusowego ze względu na konieczność obliczeń miary dla każdej pary dokumentów. Niemniej

jednak, jego wykorzystanie jest umotywowane faktem wykorzystania możliwie dużo narzędzi Big Data.

- Warto korzystać ze współbieżności! Znaczące przyspieszenie pobierania tekstów fabuł w stosunku do programu na jednym wątku.
- TinkerPop sam w sobie jest tragicznie nie-transakcyjny, a jego implementacja w Cosmos DB dodatkowo go spowalnia. Jest to wygodne narzędzie do przetwarzania grafów, natomiast wgrywając dane trzeba uzbroić się w cierpliwość – załadowanie naszego raczej niewielkiego zbioru zajęło niemal sześć godzin.
- Cosmos DB for Gremlin nie wspiera Gremlin Bytecode (i patrząc na tempo wdrażania nigdy nie będzie wspierać ani tego, ani wielu innych funkcji Gremlina). Z tego względu nie można korzystać z bibliotecznych obiektów i metod, a zamiast tego trzeba tworzyć stringi z zapytaniami i wysyłać je do serwera. Przysporzyło nam to wiele problemów z serializacją danych, a potrzeba ręcznego konsumowania odpowiedzi świetnie obfuskuje kod. Konkurencyjne rozwiązanie – Neptune DB w chmurze AWS – posiada pełną implementacje Gremlina. Będziemy wiedzieć na przyszłość.

5. Dalsze kierunki badań

Określanie podobieństwa między filmami na podstawie streszczeń fabuł to zdecydowanie najciekawsza część tego projektu, ale równie wymagająca. Korzystając z tego, że system może działać bez wyników tego procesu (aczkolwiek nie tak dobrze jak mógłby z nimi), zostawiliśmy implementację tego algorytmu na koniec. Na tym etapie więc faza przetwarzania wsadowego nie została jeszcze zakończona.

Wynikowe podobieństwa wstawione zostaną do bazy jako krawędzie między wierzchołkami filmów z wartością podobieństwa, lub tylko tam, gdzie podobieństwo przekroczy próg. Przeanalizujemy, które rozwiązanie będzie dawało trafniejsze wyniki.

Baza grafowa jest także z definicji silnikiem przetwarzania grafów więc algorytm wybierania rekomendacji będzie praktycznie polegał na wykonaniu odpowiedniej trawersacji przez zapytania. Na zapytania natomiast dobrze będzie nałożyć jakiś prosty w obsłudze interfejs. Pozwoli to również na zabezpieczenie bazy, bo interfejs będzie nawiązywał z bazą połączenie tylko do odczytu. Connector, który został wykorzystany przy zapełnianiu bazy napisaliśmy w Pythonie. Jako interfejs zdecydowaliśmy postawić proste API we Flasku, co pozwoli na wykorzystanie funkcjonalności connectora. API będzie wystawiało parę endpointów, które będą zwracały listę rekomendowanych filmów, szczegółowe informacje konkretnego filmu lub aktora itp. Problem zimnego startu zostanie rozwiązany przez zwracanie błędu np. 418 po otrzymaniu od użytkownika pustej listy. Takie rozwiązanie pozwoli na proste podłączenie naszego silnika rekomendacji do pełnej strony lub do większego serwisu. API będzie można upublicznić korzystając z Azure App Service lub Azure Functions.

Autorzy:
Michał Banaszczak
Daniel Ślusarczyk
Weronika Skiba
Patryk Chojnicki

Repozytorium:
[movie-recs-engine @ GitHub]