Polsko-Japońska Akademia Technik Komputerowych

# Przetwarzanie strumienia danych z użyciem Apache Spark, Kafka i bazy danych Cassandra

Mateusz Pierzchala
6-10-2019

# Contents

# Cele

Celem niniejszej pracy jest przedstawienie przetwarzania strumienia danych przy pomocy Apache Spark, Kafka i Cassandra DB. Przedstawione zostaną podstawy teoretyczne, aplikacja przetwarzająca strumień danych napisana w języku Scala wraz z opisem działania oraz wnioskami. Przykładowa aplikacja działać będzie w architekturze lambda.

# Wstęp teoretyczny

## 2.1 Przetwarzanie strumienia danych

Aby opisać na czym polega przetwarzanie strumienia danych należy przedstawić w pierwszej kolejności definicję samego strumienia danych. Istniejące definicje przedstawiają go najczęściej jako sekwencję zdarzeń znakowanych stemplem czasowym.

Np. wg. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations: *”Strumień jest nieograniczonym zbiorem elementów* ***(s,t)*** *gdzie* ***s*** *jest krotką należącą do schematu strumienia,* ***t*** *należące do* ***T*** *jest stemplem czasowym elementu.”*

Przykładowo – jeśli zdarzeniem jest otrzymanie zamówienia w sklepie internetowym, to sekwencja komunikatów o poszczególnych zamówieniach utworzy strumień danych (zwanym też strumieniem komunikatów).

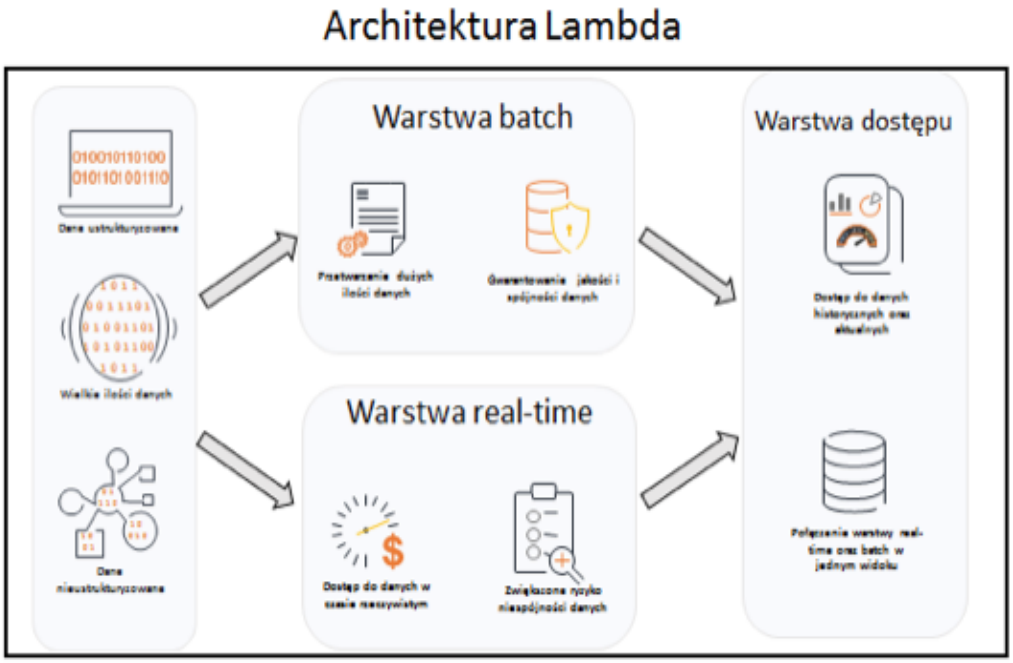
Tradycyjne podejście do analizy i reagowanie na dane, najczęściej polega na zapisaniu ich w pewnym zasobie (baza danych, system plików, itp.) i wykonywaniu analiz/zapytań na większym, utrwalonym zbiorze danych.

Czasami takie podejście może być jednak niewystarczające (np. ze względu nas czas reakcji). Strumieniowe przetwarzanie danych polega na procesowaniu danych „w locie”, natychmiast w przypadku wystąpienia pewnego zdarzenia (lub korelacji pewnych zdarzeń).

## 2.2 Architektura lambda

Zapewnienie równoległego przetwarzania dużych, nieustrukturyzowanych zbiorów danych oraz zapewnienia do nich nieustannego dostępu w czasie rzeczywistym możliwe jest poprzez zastosowanie architektury Lambda.

Założenie polega na stworzeniu dwóch osobnych przepływów, gdzie jeden odpowiedzialny jest za przetwarzanie danych w trybie wsadowym, natomiast drugi za dostęp do nich w trybie rzeczywistym. Stale napływający strumień danych kierowany jest do obu warstw.




Rys. 1 Architektura lambda

W warstwie batch obliczenia wykonywane są na całym zbiorze danych. Odbywa się to kosztem czasu, ale otrzymane w zamian dane zawierają pełną historię i wysoką jakość. Zakłada się, że zbiór danych znajdujący się w warstwie batch ma formę niepodzielną, którą należy jedynie rozszerzać, aniżeli usuwać z niej dane. W ten sposób można zapewnić pełną historyzację i spójność danych.

Warstwa real-time przetwarza napływające dane w trybie rzeczywistym. Oferuje ona niski czas dostępu do danych, co przekłada się na możliwość szybszego pozyskania informacji. Niestety, brak dostępu do danych historycznych sprawia, że nie wszystkie obliczenia są możliwe do wykonania. Często jakość oraz wiarygodność danych pochodzących z warstwy real time nie jest tak wysoka, jak z warstwy batch. Te drugie należy uważać za bardziej wiarygodne, ale ze względu na dłuższy czas potrzebny na ich załadowanie, warstwa real-time okazuje się nieocenioną pomocą, chcąc zagwarantować możliwość przetwarzania danych w trybie rzeczywistym.

Warstwa dostępu jest miejscem, w którym tworzone są widoki na podstawie warstwy batch oraz real-time. Dane są agregowane w taki sposób, aby końcowy użytkownik widział je jako jedną, spójną całość, aniżeli dwa niezależne, całkiem odrębne systemy. Widoki powinny być przygotowane w taki sposób, aby zapewnić możliwość wykonywania wszelkiego rodzaju analiz ad-hoc, ciesząc się przy tym szybkim dostępem do danych.

Koncepcja architektury Lambda zapewnia wiele zalet, przede wszystkim doskonały kompromis między przetwarzaniem wsadowym i real-time. Największą i najczęściej wspominaną wadą jest konieczność utrzymywania dwóch niezależnych aplikacji – jednej do zasilania warstwy batch, natomiast drugiej do warstwy real-time. Narzędzia wykorzystywane w poszczególnych warstwach różnią się między sobą, więc ciężko jest dobrać jedno, które może być wykorzystane do dwóch celów. Jest to możliwe w przypadku Apache Spark, gdzie po zdefiniowaniu logiki, według której dane mają być przetwarzane, można ją wywołać w trybie batchowym, jak i w trybie Spark Streaming, który ze strumienia danych wejściowych tworzy tzw. micro-batch i pozwala na przetwarzanie ich w czasie niemalże rzeczywistym.

## 2.3 Apache Spark

Apache Spark to rozproszona, ogólnodostępna platforma obliczeniowa. Spark zapewnia interfejs do programowania całych klastrów z ukrytą równoległością danych i odpornością na błędy. Pierwotnie opracowany na Uniwersytecie Kalifornijskim w AMPLab w Berkeley, Spark został później przekazany Fundacji Apache Software, która utrzymuje go od tego czasu.

Apache Spark ma za podstawę architekturę elastycznego rozproszonego zestawu danych (RDD), wieloczęściowego elementu danych tylko do odczytu, rozprowadzanego w klastrze maszyn, który jest utrzymywany w sposób odporny na błędy.

Spark i jego RDD zostały opracowane w 2012 r. W odpowiedzi na ograniczenia w modelu obliczeniowym klastra MapReduce, który wymusza określoną strukturę liniowego przepływu danych na programach rozproszonych: programy MapReduce odczytują dane wejściowe z dysku, mapują funkcję na danych, redukują wyniki mapowanie i zapisywanie wyników redukcji na dysku. RDD Sparka działają jako zestaw roboczy dla programów rozproszonych, który oferuje (celowo) ograniczoną formę rozproszonej pamięci współdzielonej.

Apache Spark obecnie składa się obecnie z czterech modułów:

- SparkSQL
- Spark Streaming
- Mllib (machine learning)
- GraphX (grafy)

Oraz Spark Core który jest podstawą działania całego ekosystemu Sparka.

Spark Streaming korzysta z możliwości szybkiego planowania Spark Core w celu wykonywania analiz strumieniowych. Pobiera dane w mini batchacg i wykonuje transformacje RDD na tych mini partiach danych. Ten projekt umożliwia wykorzystanie tego samego zestawu kodu aplikacji do analizy wsadowej w analizie strumieniowej, ułatwiając w ten sposób łatwą implementację architektury lambda. Jednak ta wygoda wiąże się z karą za opóźnienie równą czasowi trwania mini-partii. Inne silniki danych strumieniowych, które przetwarzają zdarzenia według zdarzeń, a nie w mini-partiach, to Storm i składnik strumieniowy Flink. Spark Streaming ma wbudowane wsparcie do korzystania z gniazd Kafka, Flume, Twitter, ZeroMQ, Kinesis i TCP / IP.

W Spark 2.x udostępniono oddzielną technologię opartą na zestawach danych, zwaną Strumieniowaniem Strukturalnym, która ma interfejs wyższego poziomu do obsługi przesyłania strumieniowego.

## 2.4 Apache Kafka

Apache Kafka to rozproszona, silnie skalowalna kolejka komunikacyjna zdolna do obsługi ogromnych ilości komunikatów i jednoczesnych połączeń klienckich. Kafka pozwala podjąć wyzwanie stawiane przez Big Data, tam gdzie zawiodły technologie brokerów oparte o standardy JMS czy AMQP. Technologia doskonale wpisuje się w dynamicznie zmieniający się rynek usług, adresując olbrzymie wolumeny informacji generowane przez środowiska mobile czy IoT (Internet Of Things).

Apache Kafka, BigData

Rozwiązanie implementuje model komunikacji typu publish-subscribe z akcentem położonym na maksymalną przepustowość i minimalne opóźnienie w dostarczeniu komunikatów. Kafka gwarantuje doręczenie wiadomości do systemu subskrybenta dzięki mechanizmowi dziennika komunikatów. Trwały zapis danych w węzłach Kafka pozwala na ich przetwarzanie również w trybie „batch" w sposób analogiczny do narzędzi ETL.

Kafka znalazła zastosowanie w wielu organizacjach, gdzie niezbędna okazała się obsługa strumieni terabajtów danych klienckich w sposób maksymalnie niezawodny. Przykład to Spotify, Uber czy PayPal.

Potencjalne zastosowania Apache Kafka:

- Gromadzenie danych na potrzeby analizy w czasie rzeczywistym (-> Apache Spark)
- Strumieniowe przetwarzanie danych (-> Apache Storm)
- Przyjęcie roli Message Oriented Middleware (MOM)
- Agregacja logów aplikacyjnych

## 2.5 CassandraDB

Apache Cassandra to rozproszona baza danych stworzona w 2008 roku przez inżynierów firmy Facebook. Lawinowo rosnąca liczba użytkowników pokazała, że tradycyjne, relacyjne silniki bazodanowe oraz istniejące rozwiązania nie są w stanie zapewnić odpowiedniej wydajności rozwiązania. Po udanym uruchomieniu zespół rozwijający Cassandrę zdecydował się na udostępnienie projektu szerokiej społeczności poprzez Apache Incubator. W chwili obecnej Apache Cassandra jest najbardziej wydajną bazą danych NoSQL klasy „widerow" przy jednoczesnym zachowaniu pełnej skalowalności na sprzęcie dowolnej klasy.

Apache Cassandra została zaprojektowana tak aby pokryć wszystkie obszary wymagań niefunkcjonalnych. Do głównych cech rozwiązania należą:

- Skalowalność - twórcy Cassandry postawili na skalowalność poziomą i dzięki temu dodanie kolejnego węzła zwiększa przepustowość zarówno odczytu jak i zapisu bez niedostępności dla aplikacji

- Replikacja i bezpieczeństwo danych - strategia replikacji dla klastra Cassandry jest zaprojektowana w taki sposób aby umożliwić projektantowi systemu konfigurację idealnie dopasowaną do potrzeb z uwzględnieniem wielu centrów danych oraz rozmieszczenia serwerów w szafach. Przy odpowiednio skonfigurowanej replikacji Cassandra zapewnia bezpieczeństwo danych.

- Brak pojedynczego punktu podatności na awarie (SPOF) - każdy element klastra pełni identyczną rolę więc awaria jednego węzła nie powoduje problemów z dostępnością systemu

- Transakcyjności i spójność danych - zapisy i odczyty mają możliwość konfiguracyjnego sterowania poziomem blokowania rekordów – od rozwiązanie, w którym operacja zapisu zawsze kończy się powodzeniem aż po blokowanie wszystkich replik do momentu zakończenia. Najbardziej popularnym rozwiązaniem jest wykonywanie operacji z poziomem spójności quorum(połowa plus jeden), które pozwala na poprawną pracę w większości przypadków biznesowych. Jednocześnie projektant systemu ma możliwość sterowania poziomem spójności nawet na poziomie pojedynczych zapytań co pozwala na użycie Cassandry w wielu typowych oraz nietypowych zastosowaniach.

Jednym z większych ułatwień dla twórców aplikacji było wprowadzenie w wersji 0.8 języka CQL (Cassandra Query Language). Dzięki wprowadzeniu dodatkowej warstwy abstrakcji pomiędzy wewnętrznymi strukturami danych a aplikacjami klienckim Cassandra zyskała wielu zwolenników zarówno wśród entuzjastów oprogramowania Open Source jak i decydentów w korporacjach, które do tego czasu korzystały wyłącznie z relacyjnych baz danych.

Czytelna i kompletna dokumentacja w połączeniu z szerokim wsparciem społeczności powoduje, że koszty uruchomienia pilotażowych wdrożeń są relatywnie niskie. Jednocześnie duża popularność platformy spowodowała, iż na rynku bezproblemowo można znaleźć zarówno specjalistów znających technologię jak i firmy świadczące kompletne wsparcie rozwiązania.

Apache Cassandra posiada szeroką bazę sterowników dla różnych technologii wydanych zarówno jako oficjalne wersje przez zespół Datastax rozwijający podstawowe funkcjonalności platformy jak i wersji alternatywne wydawane przez społeczność. Dzięki szerokiemu wachlarzowi dostępnych opcji Cassanda pozwala na przyjazną integrację dla wszystkich popularnych technologii i platform.

Solidne podstawy przy projektowaniu oraz wysoka stabilność rozwiązania sprawiły, że baza Apache Cassandra została wybrana przez wielu dużych graczy z różnych domen biznesowych.

Cassandra została z powodzeniem wdrożona przez wiele organizacji m.in. CERN, Apple, Netflix, eBay, GitHub, Reddit. Jednocześnie dzięki dojrzałości rozwiązanie zostało zaaprobowane i wdrożone przez ponad 1500 firm z sektora finansowego m.in. ING, UBS, Credit Suisse.

## 2.6 Inne technologie użyte w projekcie

**Scala** - język programowania łączący cechy języków funkcyjnych i obiektowych. Scala działa na Wirtualnej Maszynie Javy, a także na Java Platform, Micro Edition Connected Limited Device Configuration i platformie .NET. Nazwa ma za zadanie podkreślać skalowalność języka, stąd Scala ("scalable language").

Scala został stworzony w 2001 roku na École Polytechnique Fédérale de Lausanne przez Martina Odersky'ego. Język ten został upubliczniony w styczniu 2004 roku na platformie Javy, a w czerwcu tego samego roku na .NET. Wersja druga została udostępniona w marcu 2006 roku.

**Apache Hadoop**

Apache Hadoop jest otwartą platformą służącą do rozproszonego przechowywania i przetwarzania dużych zbiorów danych.

Główne zalety platformy Apache Hadoop to:

- skalowalność
  ponieważ jest to darmowa platforma oparta o architekturę klastrową, klaster Hadoop może być w łatwy sposób rozbudowywany o kolejne serwery w sposób przeźroczysty dla zapisanych już danych i zdefiniowanych procesów

- elastyczność
  mnogość narzędzi wchodzących w skład ekosystemu Hadoop sprawia, że umożliwia on przetwarzanie danych zarówno ustrukturyzowanych jak i nieustrukturyzowanych (z którymi najczęściej mamy do czynienia w Big Data)

- odporność na awarie
  dzięki replikacji danych i narzędziom umożliwiającym pracę klastra w trybie High Availability (HA) zapewniony jest spójny i ciągły dostęp do przechowywanych danych, mimo awarii któregokolwiek z serwerów

- szybkość przetwarzania danych
  przetwarzanie danych w sposób rozproszony sprawia, że przetwarzanie bardzo dużych wolumenów danych jest dużo szybsze niż w przypadku standardowych mechanizmów ETL i przetwarzania wsadowego

- wydajne zarządzanie zasobami
  zadania są odpowiednio dzielone pomiędzy maszyny, aby w pełni wykorzystać moc klastra.

Powyższe cechy sprawiają, że Apache Hadoop jest jednym z najczęściej wybieranych rozwiązań w budowaniu szkieletu kompleksowych rozwiązań związanych z Big Data.

Wśród firm wykorzystujących Apache Hadoop w swoich produktach można wymienić Adobe, Ebay, Facebook, Google, IBM, Spotify, Twitter, Yahoo i wiele innych znanych firm z branży IT.

Główne komponenty
Hadoop składa się z czterech podstawowych modułów:

- Hadoop Common
  zestaw bibliotek i narzędzi do obsługi pozostałych modułów

- Hadoop Distributed File System (HDFS)
  rozproszony system plików, który dzieli dane na mniejsze bloki i składuje je w rozproszony i równomierny sposób na węzłach klastra z odpowiednim poziomem replikacji

- MapReduce
  implementacja paradygmatu programistycznego, który umożliwia rozproszone przetwarzanie dużych ilości danych

- YARN (Yet Another Resource Navigator)
  platforma do zarządzania zasobami klastra.

**Zeppelin**
Apache Zeppelin to webowy notes do przechwytywania, eksplorowania, wizualizacji i udostępniania danych opartych na Hadoop i Spark.

**Docker**
Docker jest najbardziej rozbudowaną platformą do zarządzania kontenerami oprogramowania. Kontener oferuje pełne środowisko do poprawnego działania każdego systemu. W ramach kontenera znajdują się:
– biblioteki systemowe,
– narzędzia systemowe,
– runtime

Są one tożsame z konkretnym obrazem systemu operacyjnego. Dzięki temu Docker gwarantuje, że system będzie działał dokładnie tak samo w ramach każdej instancji kontenera.

# Aplikacja

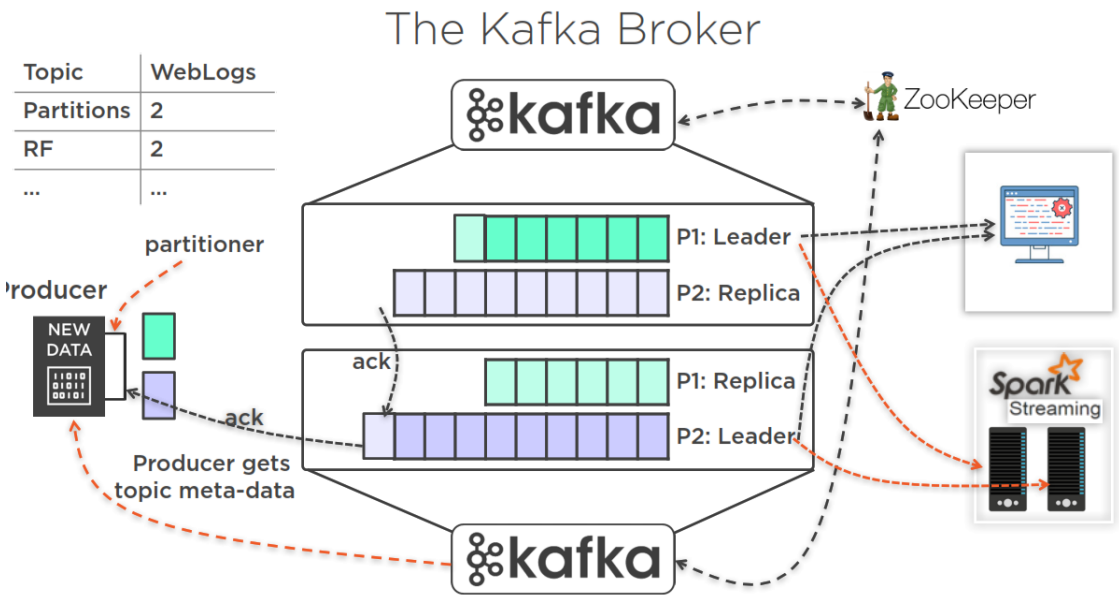
Aplikacja przygotowana w ramach projektu składa się z kilku modułów. Jej głównym zadaniem jest przetwarzanie logów ze sklepu internetowego w czasie rzeczywistym oraz w trybie batchowania, a następnie zapis do bazy danych Cassandra.

## 3.1 Log Producer

Log Producer jak sama nazwa wskazuje posłuży nam do preparowania logów ze sklepu internetowego. Z dwóch wcześniej przygotowanych plików CSV pobiera on dane na temat produktów i referentów (czyli strony z której do naszego sklepu trafil użytkownik) a następnie w losowy sposób je kombinuje dodając równocześnie dodatkowe losowe cechy. W efekcie dostajemy ramkę danych składającą się z :

- Polecającego (czyli strony z której do naszego sklepu trafil użytkownik)
- Akcji – kupno,dodanie do koszyka lub wejście na stronę
- Poprzedniej strony
- Unikatowy id odwiedzającego
- Obecnej strony
- Produktu

Tak przygotowana ramka danych trafia do brokera Kafki który dalej przesyła ją do Spark Streaming



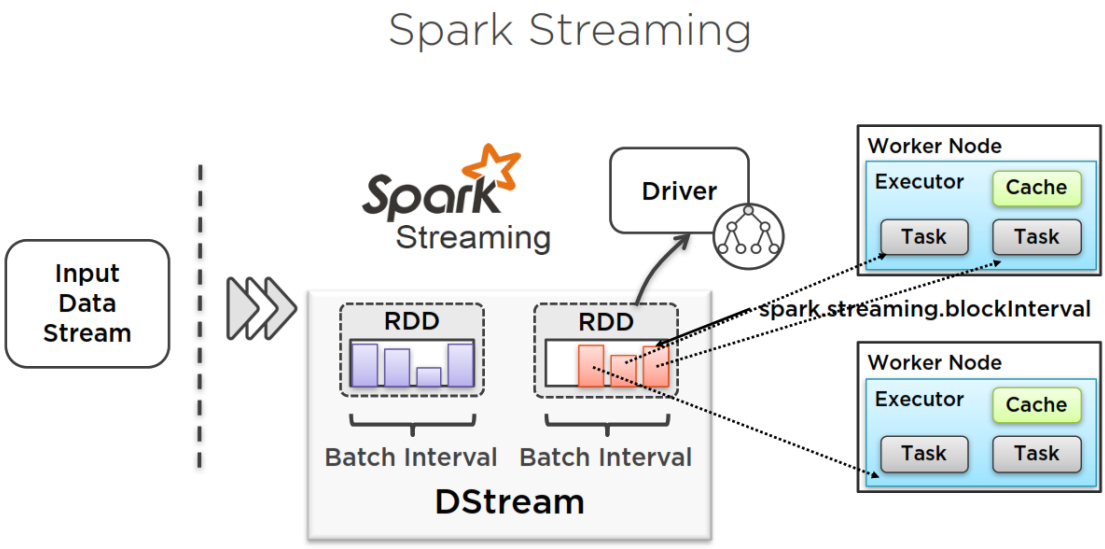

Rys. 2 Przepływ strumienia danych przez Apache Kafka

## 3.2 Speed layer

Moduł StreaminJob jest „jobem” w Spark Streaming- pobiera ramki danych z brokera Kafki w czasie rzeczywistym za pomocą micro batchingu. Pobrane dane przetwarza za pomocą SparkSQL tworząc tabelę ActivityByProduct sumującą akcję powiązane z produktem tj. Liczby obejrzeń, dodań do koszyka i zakupów konkretnych produktów.

Dodatkowo job oblicza ilość unikatowych użytkowników wchodzących na stronę kolejnych przedmiotów za pomocą algorytmu HyperLogLog ograniczającego zużycie pamięci.

Następnie przetworzony strumień danych zostaje przesłany do bazy danych Cassandra.

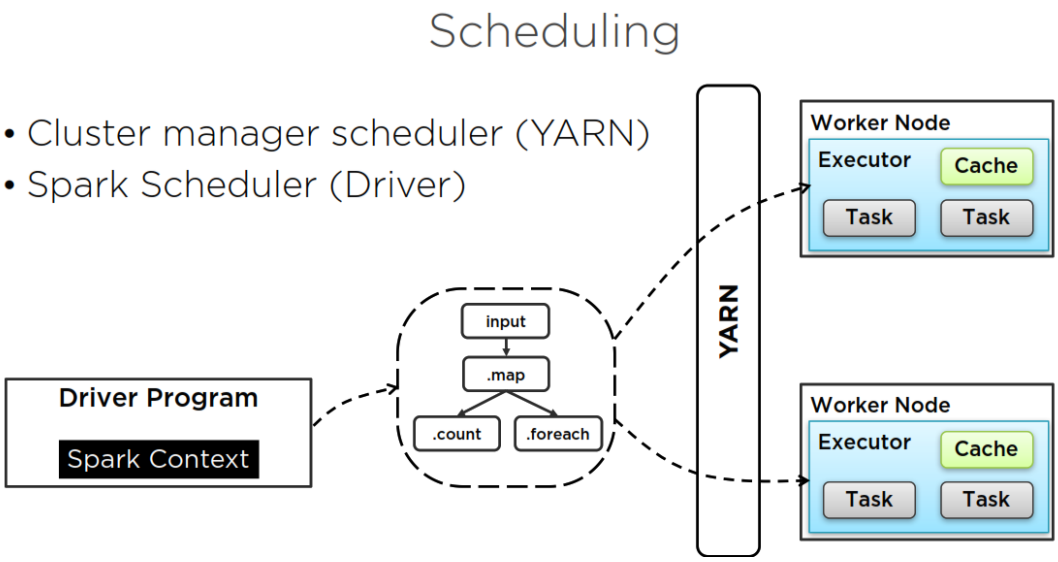



Rys 3. Zasada działania warstwy Speed Layer

### 3.3 Batch layer

BatchJob jest niejako dopełnieniem StreamJob- wykonuje on te same zadania, natomiast jest o wiele dokładniejszy ze względu na to, że aby zmniejszyć użycie zasobów w warstwie przetwarzającej dane w czasie rzeczywistym korzytamy z algorytmów aproksymujących niektóre agregacje. Warstwa batch odczytuje większe porcje danych zapisanych juz na HDFS.

Jego wadą jest to, że infromacje zawarte w pozyskanych danych otrzymamy z opóźnieniem




Rys. 4 Zasada działania warstwy Batch Layer

## Wnioski

Stworzona aplikacja pozwa na rozwiązanie problemu optymalizacji przetwarzania danych w czasie rzeczywistym dzięki zastosowaniu architektury lambda. Zastosowanie dwóch przepływów danych pozwala zarówbno na dostęp do wyników w czasie rzeczywistym jak i przeprowadzenie obliczeń, transformacji itd. Na całym zbiorze danych co z kolei gwarantuje pełną historię oraz wysoką jakoś otrzymanych wyników.

## Instrukcja

Do uruchomienia programu potrzebne jest następujące oprogramowanie:
-VirtualBox
-Vagrant
-IntelliJ IDEA Community Edition
-chocolatey (menadżer pakietów dla Windows), openssh
-cygwin/cmdr (lub jakiś inny emulator terminala)
Kod programu znajduje się w repozytorium
https://github.com/mateuszpierzchala/lambda_arch
Do uruchomienia projektu użyjemy przygotowanej wcześniej maszyny wirtualnej z zainstalowanymi: Apache Spark, Kafka, Hadoop, Cassandra i Zeppelin.
Na początku musimy probrać repozytorium git:

```
git clone https://github.com/aalkilani/spark-kafka-cassandra-applying-lambda-architecture.git
```

po pobraniu należy „postawić” maszynę wirtualną poleceniem

```
vagrant up
```

a następnie połączyć się z nią przez ssh

```
vagrant ssh
```

Powinniśmy się znajdować w linii poleceń jako użytkownik lambda-pluralsight
W IntelliJ należy spakować naszprogram do pliku JAR- klikamy na menu Maven (w prawym górnym rogu) i wybieramy opcję „package” a następnie przenosimy plik `spark-lambda-1.0-SNAPSHOT-shaded.jar` do folderu `...\lambda_arch\vagrant`

Zanim uruchomimy SteramingJob należy utworzyć odpowiednie tabele w CassandraDB. Zrobimy to przez Apache Zeppelin- w folderze `...\ lambda_arch\zeppelin` znajduje się notatnik „cassandra”. Aby go otworzyc w oknie przeglądarki wpisujemy `localhost:8988` i przechodzimy do interfejsu zeppelina. Następnie importujemy ww. notatnik i uruchamiamy okienka aż do momentu pierwszego selecta.
Gdy mamy już utworzone tabele do których będą zapisywane dane możemy uruchomić LogProducer. W IntelliJ klikamy PPM na nazwę LogProducer w drzewie plików i wybieramy opcję „Run”.
Teraz możemy w końcu uruchomić nasz program. W linii poleń maszyny wirtualnej przechodzimy do katalogu `~/pluralsight/spark` i wpisujemy polecenie `./bin/spark-submit --master local[*] --deploy-mode client --driver-memory 2g --executor-memory 1g --executor-cores 1 --class streaming.StreamingJob /vagrant/spark-lambda-1.0-SNAPSHOT-shaded.jar`
Nasza aplikacja powinna zacząć działać i po chwili w zeppelinie możemy uzyskać pierwsze wyniki pisząć proste zapytanie CQL `select * from lambda.activity_by_product;`
Powinniśmy zauważyć pojawiające się co jakiś czas kolejne wpisy.
Aby uruchomić zadanie batchowe należy zaimporsowac z folderu zeppelin notatnik BatchtoCassandra i uruchomić go podobnie jak wcześniejszy.