Uniwersytet Warmińsko-Mazurski

Analiza danych z użyciem Apache Spark

Spark Core oraz Spark SQL

Paweł Procaj

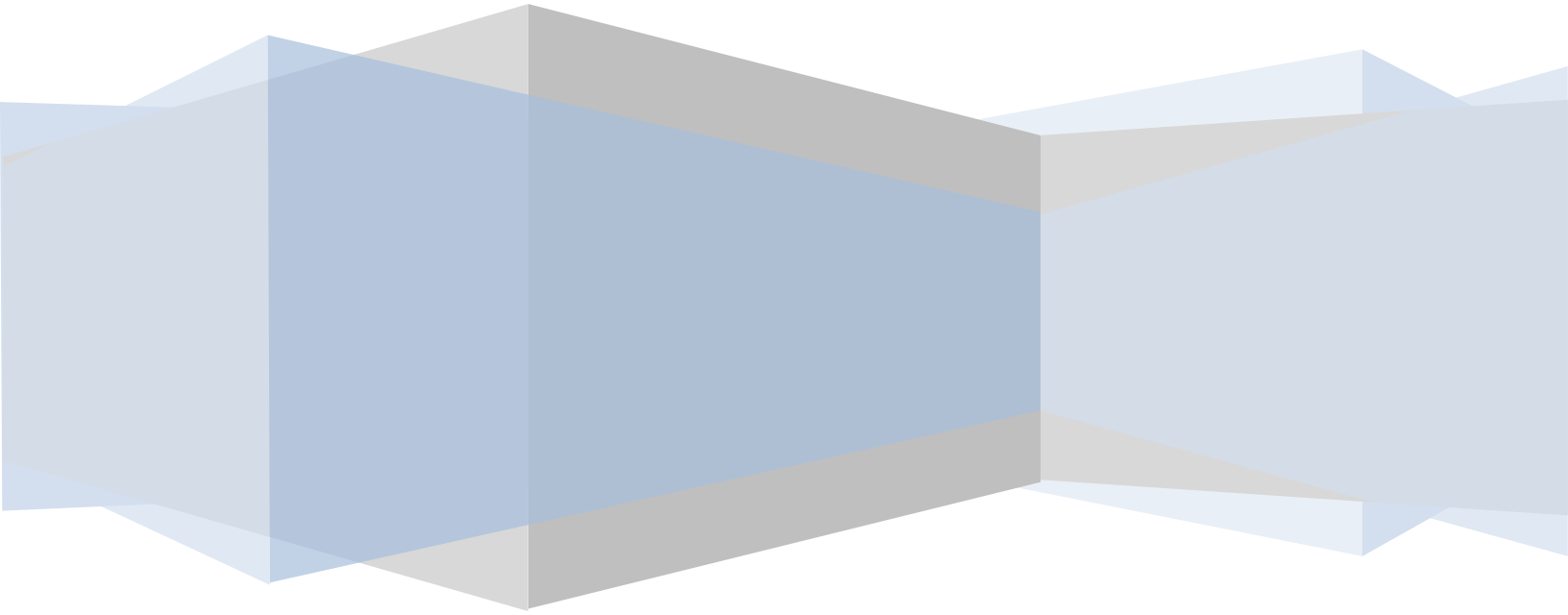
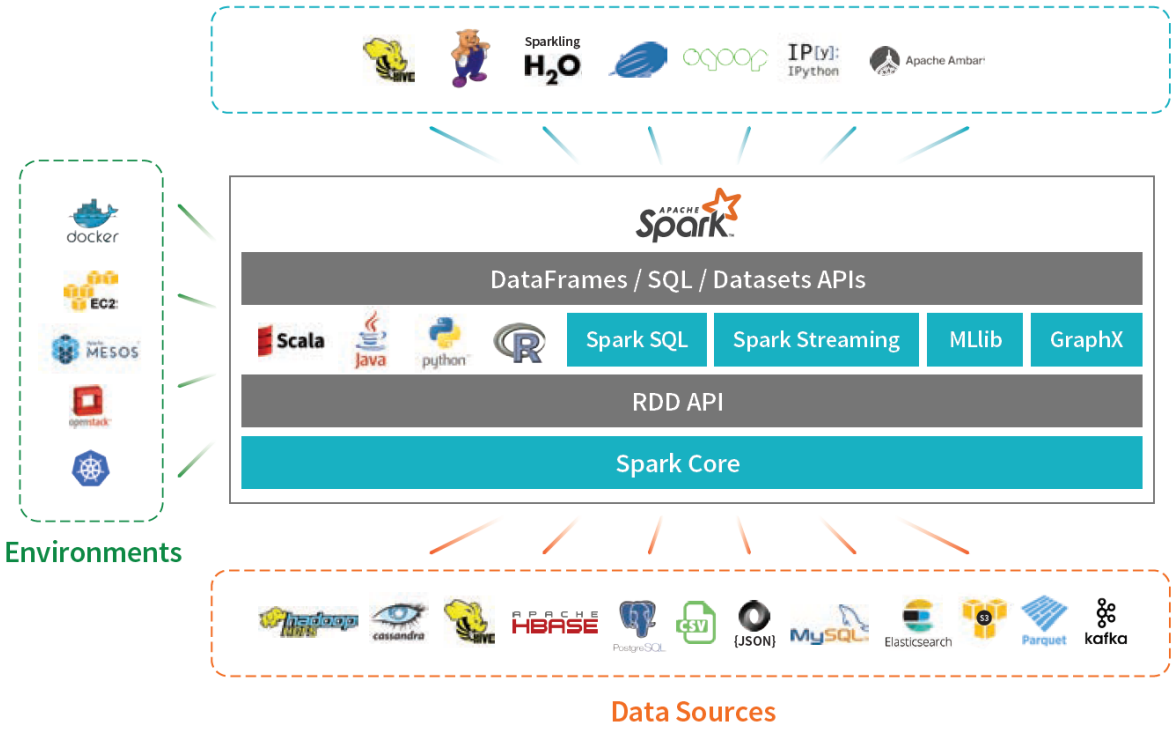


Table of Contents

Apache Spark

Platforma programistyczna do rozproszonego przetwarzania danych.
Rozwijana przez Apache Foundation, UC Berkeley, DataBricks.
Prace rozpoczęte w 2009 roku, publicznie dostępna w 2013 roku.
Obecna stabilna wersja: 2.3.0.
Strona internetowa: http://spark.apache.org

Środowisko działania Apache Spark



Spark Core

Spark Core jest podstawą całej platformy Spark.
Funkcje:

- rozsyła rozproszone zadania,
- planuje wykonywanie zadań,
- zarządza podstawowymi funkcjonalnościami I/O,
- wystawia programistyczny interfejs aplikacji API w języku Java, Python, Scala oraz R,
- opiera się na abstrakcji rozproszonych obiektów RDD (Resilient Distributed Datasets, odporne rozproszone zbiory danych).

Interfejs API spełnia cechy modelu programowania funkcyjnego:

1. Program sterownika wywołuje operacje równoległe takie jak map, filter czy reduce na zbiorach RDD przez przekazanie funkcji do Sparka.
2. Spark następnie planuje wykonanie operacji w trybie równoległym na klastrze obliczeniowym.
3. Operacje typu transformacje tworzą nowe RDD na swoim wyjściu.
4. Obiekty RDD są niemodyfikowalne / niemutowalne (ang. immutable).
5. Operacje są ewaluowane leniwie (ang. lazy evaluation).
6. Odporność na błędy (ang. fault tolerance) jest osiągana przez ponawianie wykonania zadań na RDD.
7. RDD może zawierać obiekty różnych typów danych z dowolnego wspieranego języka.

Spark Core dostarcza również dwa rodzeje współdzielonych zmiennych:

1. Zmienne typu rozgłoszeniowego (ang. broadcast variables)
2. Akumulatory (ang. accumulators).

Spark SQL

Spark SQL jest komponentem bazującym na Spark Core.
Funkcje:

- Dodaje strukturę danych zw. DataFrame, ułatwiającą manipulowanie danymi i pozwalającą na tworzenie danych strukturalnych oraz semi-strukturalnych.
- Dostarcza tzw. język do manipulowania danych / DSL (Domain Specific Language) jako wsparcie dla języków Scala, Java, czy Python.
- Dostarcza wsparcie dla języka SQL.
- Dostarcza dostęp do baz danych poprzez CLI oraz serwer ODBC/JDBC.

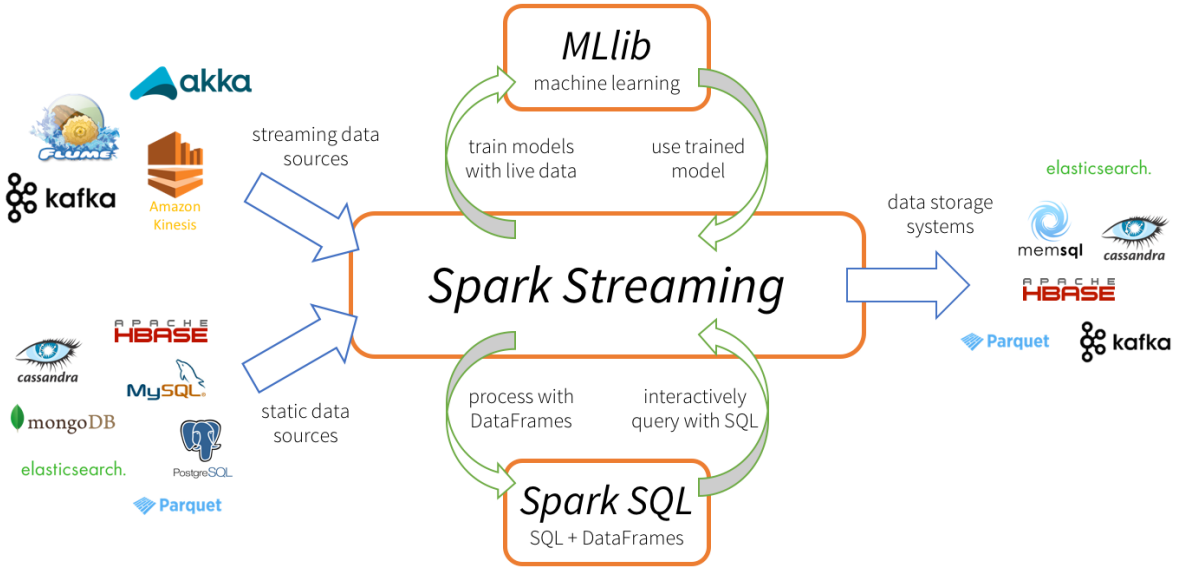
Spark Streaming

Spark Streaming jest modułem Sparka do analityki strumieniowej, wykorzystającym do tego celu możliwości Spark Core do szybkiego planowania zadań.
Cechy:

- Wczytuje dane w formie mini-wsadów (mini-batches) i wykonuje na nich operacje.
- Architektura modułu pozwala na użycie tego samego kodu do analizy wsadowej jak i strumieniowej, dzięki czemu łatwo jest zaimplementować architekturę Lambda.
- Mini-batche wprowadzają jednak opóźnienia równe czasie trwania mini-wsadu. Inne systemy (Storm, Flink, Gearpump) przetwarzają strumień zdarzenie po zdarzeniu.
- Wbudowane wsparcie dla Kafka, Flume, Twitter, ZeroMQ, Kinesis, czy gniazd TCP/IP.
- Nowa wersja (Structured Streaming) posiada interfej kontroli typów danych.

Schemat działania przestawia rysunek:

Środowisko działania przedstawia rysunek:



Spark MLlib - Machine Learning Library

Spark MLlib to rozproszona biblioteka algorytmów uczących się działająca na bazie Spark Core oraz SQL.
Cechy:

- Wiele wbudowanych algorytmów statystycznych oraz uczących się.
- Ułatwia obróbkę danych poprzez łączenie zadań w potoki.
- Ze względu na sposób korzystania z pamięci jest wiele razy szybsza niż oparta na dyskach implementacja Apache Mahout.

Zaimplementowane algorytmy:

- Statystyka sumaryczna, korelacje, generacja danych losowych.

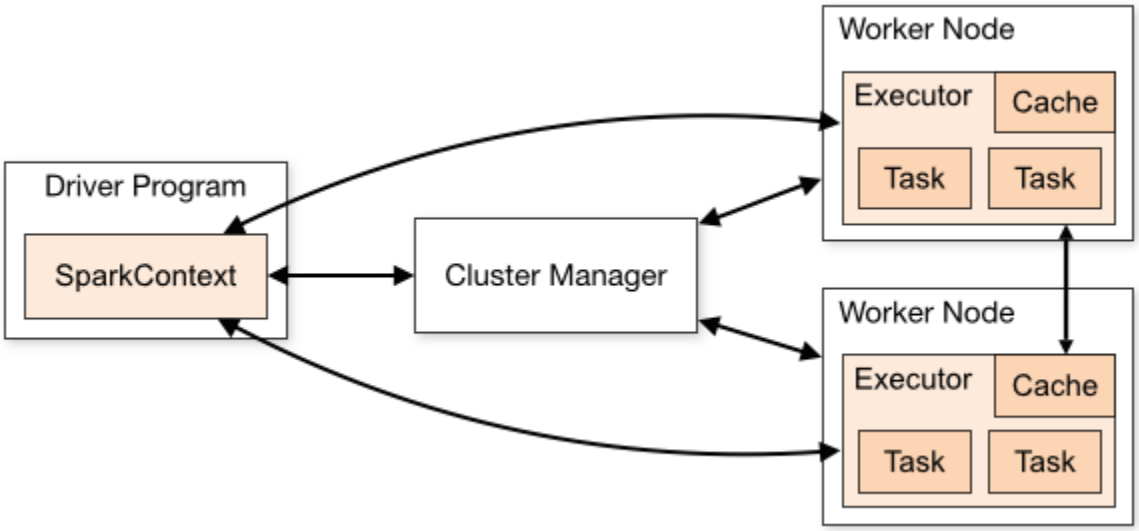
- Klasyfikacja i regresja: rzadkie i gęste wektory oraz macierze, regresja logistyczna, regresja liniowa, drzewa decyzyjne, klasyfikacja metodą naiwnego Bayesa.
- Filtrowanie kolaboratywne: ALS (metoda naprzemiennych najmniejszych kwadratów, ang. alternating least squeares).
- Klastrowanie: k-means, alokacja metodą Dirichleta (ang. latent Dirichlet allocation, LDA).
- Techniki redukcji wymiarów: dekompozycja wartości osobliwej (ang. singular value decomposition, SVD) oraz analiza głównych składowych (ang. principal component analysis, PCA).
- Ekstrakcja cech oraz funkcje transformujące.
- Algorytmy optymalizacyjne: stochastic gradient descent, limited-memory BFGS (L-BFGS).

Spark GraphX

Spark GraphX to rozproszona biblioteka do przetwarzania danych grafowych na bazie Spark Core oraz Spark SQL. Służy do analizy danych z użyciem koncepcji grafów skierowanych, gdzie do oznaczania obiektów używa się wierzchołków grafów, a jako relacje traktowane są krawędzie między wierzchołkami. Zawiera kilka popularnych algorytmów, m.in. PageRank, zliczanie trójkątów.

Architektura klastra Spark

Spark może być uruchomiony w trybie klastrowym, czyli w grupie maszyn współpracujących ze sobą. Aplikacje Sparka działają wtedy jako niezależne zestawy procesów w klastrze, koordynowane przez obiekt SparkContext w programie głównym (sterowniku).



Zarządca klastra (Cluster Manager)

Istnieje kilka używanych typów zarządców klastra (managerów), które przydzielają zasoby aplikacjom:

- Tryb samodzielny – zarządca dołączany standardowo do Sparka, pozwala łatwo zbudować klaster.
- YARN – podstawowy zarządca zasobów w systemie Hadoop2.
- Mesos – system zarządzania zasobami, potrafiący uruchamiać aplikacje Spark i MapReduce.
- Kubernetes – system open-source do automatyzacji instalacji, skalowania i zarządzania aplikacjami kontenerowymi.

Zarządca klastra kontroluje fizyczne maszyny i przydziela zasoby dla aplikacji Sparka (pamięć RAM, ilość procesorów, miejsce na dyskach).
Po połączeniu z managerem klastra, posiadając dane o lokalizacji wolnych zasobów, Spark tworzy wykonawców (ang. executors) na węzłach w klastrze, a dokładniej tworzy na nich procesy wykonujące obliczenia i trzymają dane aplikacji. Następnie Spark rozsyła kod aplikacji do wykonawców, a SparkContext wysyła im zadania do wykonania.

SparkSession grupuje w sobie:

- SparkContext
- SQLContext
- HiveContext
- StreamingContext
- SparkConf

Monitorowanie

Każdy program sterownika ma swój interfejs graficzny w postaci aplikacji WWW, typowo uruchamianej na porcie 4040.

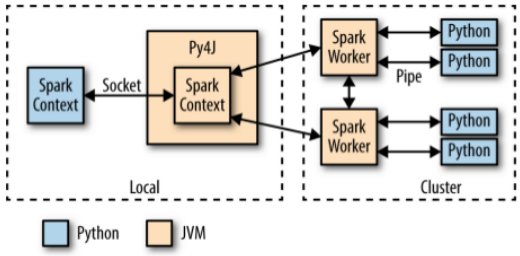
Cechy aplikacji monitorującej:

- wyświetla informacje na temat uruchamianych zadań, wykonawców oraz zużycia magazynu danych.
- Aplikacja dostępna jest w przeglądarce internetowej pod adresem:
    - http://<driver-node>:4040 (lub 4041, 4042, ...)

Interpreter PySpark

Działanie

- Kiedy wystartuje interpreter Pythona w PySparku uruchamia się również maszyna JVM, z którą się on komunikuje przez gniazdo sieciowe, czyli wewnętrzny mechanizm system operacyjnego.
- By obsłużyć komunikację PySpark używa do tego projektu Py4J.
- JVM funkcjonuje jako właściwy sterownik Spark Driver.
- JVM ładuje kontekst JavaSparkContext, który komunikuje się z wykonawcami (executorami) Sparka w klastrze.



- Interfejs API Pythona woła obiekt SparkContext, który tłumaczy te wołania na wołania interfejsu Java API modułu JavaSparkContext.
- Na przykład implementacja metody sc.textFile() rozsyła wołanie do metody .textFile() obiektu JavaSparkContext, który ostatecznie komunikuje się z maszyną JVM egzekutora, by ten załadował tekst z HDFS.
- Wykonawcy Sparka na klastrze startują swój własny interpreter I komunikują się z nim przez potok, kiedy muszą wykonać kod użytkownika.
- Obiekt Python RDD na lokalnym kliencie PySpark koresponduje z obiektem klasy PythonRDD w lokalnym JVM.
- Dane związane z RDD żyją właściwie w maszynie SparkJVM jako obiekty Java.
- Na przykład, uruchomienie sc.textFile() w interpreterze Pythona będzie wołało metodę JavaSparkContext.textFile(), która załaduje dane jako obiekty Java String w klastrze.
- Podobnie, ładowanie pliku Parquet/Avro używając metody newAPIHadoopFile będzie ładowało obiekty jako obiekty Java Avro.
- Kiedy wołania API są tworzone w obiekcie Python RDD, każdy związany kod (np. funkcja lambda pythona) jest serializowany jako tzw. moduł *cloudpickle* (format PiCloud) i jest rozsyłany do wykonawców.
- Dane jest konwertowana z obiektów Java na reprezentację w Python and strumieniowana do skojarzonego z wykonawcą interpretera Pythona poprzez utworzony.

Powłoka pyspark

Konsola / powłoka Sparka pozwala analizować dane interaktywnie. Istnieją dwie wersje powłoki:

- domyślna w języku Scala, uruchamiana poprzez ./bin/spark-shell
- dla języka Python, uruchamiana poprzez ./bin/pyspark

W Sparku głównym typem danych jest rozproszona kolekcja danych. Każdy obiekt zbioru danych w PySparku jest formalnie typu Dataset[Row], zwanego inaczej DataFrame.

Przykład użycia:
Najpierw stwórzmy DataFrame z tekstu pliku README w katalogu źródłowym Sparka:
>>> textFile = spark.read.text("README.md")
Na stworzonym obiekcie możemy teraz wykonać dowolne akcje bądź transformacje:
>>> textFile.count()
103
>>> textFile.first()
Row(value=u'# Apache Spark')

Możemy też przekształcić obiekt textFile typu DataFrame na inny, przykładowo wywołując funkcję filter, która zwraca nowy obiekt DataFrame z podzbiorem linii w pliku:
>>> linesWithSpark = textFile.filter(textFile.value.contains("Spark"))
Wszystkie akcje można wywołać w jednym ciągu:
>>> textFile.filter(textFile.value.contains("Spark")).count()
20

Budowa aplikacji PySpark

Aplikacja PySpark to aplikacja wykorzystująca Python API programistycznie. Poniżej znajduje się przykład.

```
"""SimpleApp.py"""
from pyspark.sql import SparkSession

logFile = "YOUR_SPARK_HOME/README.md"  # plik w systemie lokalnym
spark = SparkSession.builder.appName("SimpleApp").getOrCreate()
logData = spark.read.text(logFile).cache()

numAs = logData.filter(logData.value.contains('a')).count()
numBs = logData.filter(logData.value.contains('b')).count()
print("Lines with a: %i, lines with b: %i" % (numAs, numBs))
spark.stop()
```

Program liczy w tekście liczbę linii zawierających literę ‘a’ oraz ‘b’. Do stworzenia kolekcji został użyty obiekt SparkSession.
Do jego wywołania służy skrypt **spark-submit**. Przykład wywołania:

- YOUR_SPARK_HOME/bin/spark-submit --master local[4] --py-files my-lib.zip SimpleApp.py

...
Lines with a: 46, Lines with b: 23

Jeżeli PySpark jest zainstalowany jako biblioteka środowiska Python (przez pip install pyspark) to można uruchomić standardowy interpreter języka Python:

- python SimpleApp.py

...
Lines with a: 46, Lines with b: 23

Przykładowe aplikacje

Będąc w katalogu domowym Sparka:
$ ./bin/spark-submit --master local ./examples/src/main/python/pi.py 10

Spark Core

RDD (Resilient Distributed Datasets)

Na rozproszonych obiektach RDD można wywołać dwie akcje: transformacje oraz akcje.

Problemy z RDD:

- brak schematu danych
- brak sprawdzania poprawności typów danych
- są zwykłymi obiektami javy, kosztowne serializacja oraz odśmiecacanie pamięci
- brak złożonych optymalizacji przekształceń (catalyst optimizer, Tungsten execution engine).

Transformacje

Transformacja obiektu RDD daje w wyniku inny obiekt RDD i tworzy tzw. historię transformacji obiektu RDD.
Przykłady: map, filter, sample, mapValues, union, itp.
Dwie kategorie transformacji:

- Wąskie transformacje, operujące na danych z pojedynczej partycji RDD.
- Szerokie transformacja wymaga przenoszenia dnych między partycjami przed wykonaniem obliczeń w klastrze.

Akcje

Akcje zwracają konkretną wartość na podstawie wykonanych obliczeń na obiektach RDD.
Przykłady: reduce, count, first, foreach.
Wywołanie akcji uruchamia wykonanie pracy na klastrze Sparka.

Cache

Spark dostarcza mechanizm cache'owania zbioru danych w pamięci. Jest to funkcja cache() dostępna dla obiektu RDD.
Cache przydaje się, gdy wykonywana na Sparku praca wymaga kilku akcji wykonujących transformacje na konkretnym zestawie danych.

Leniwa ewaluacja

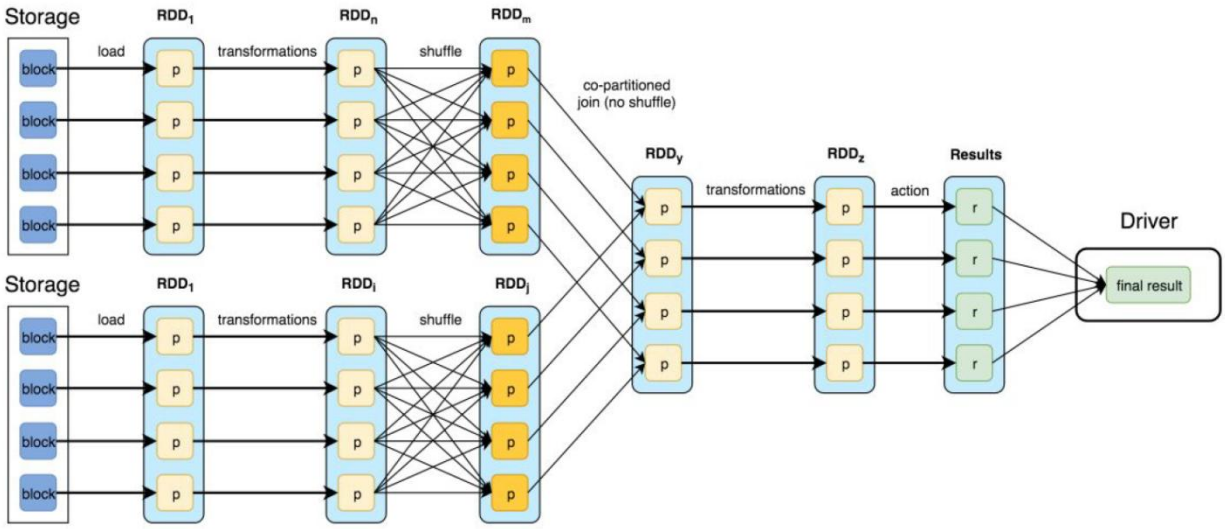
Przekształcenia zbiorów RDD (takie jak map, filter, czy nawet textFile) są ewaluowane w sposób leniwy, co oznacza, że Spark nie zacznie ich wykonywania, dopóki nie zobaczy końcowej akcji
Poprzez leniwą ewaluację oraz grupowanie działań Spark optymalizuje liczbę operacji na danych.

DAG (Directed Acyclic Graph)

W Sparku przetwarzanie zadań modelowane jest z użyciem koncepcji grafu DAG.
Definicja:

DAG to skierowany graf acykliczny, czyli bez cykli (pętli) między węzłami grafu. Tworzony jest jako kolekcja wierzchołków oraz krawędzi skierowanych, gdzie każda krawędź łączy jeden wierzchołek z innym.

Graf DAG budowany jest dla łańcucha transformacji i akcji. Przykład:



Budową grafu DAG zajmuje się moduł Sparka DAGScheduler (planista DAG), który dzieli go również na etapy.

Zadania planisty DAGScheduler:

1. Tworzy graf DAG transformacji i akcji operacji RDD.
2. Optymalizuje graf, by zwiększyć wydajność wykonania pracy (przykładowo map, a później filter zostaną zamienione miejscami w kolejności uruchomienia).
3. Śledzi pochodzenie obiektów RDD.
4. Określa preferowane miejsce uruchomienia, biorąc pod uwagę status cache.
5. Uruchamia transformacje.
6. Wysyła zbiory zadań (traktowane jako etapy) do niskopoziomowego Planisty Zadań (Task Scheduler).
7. Obsługuje błędne sytuacje.

Zadania planisty TaskScheduler:

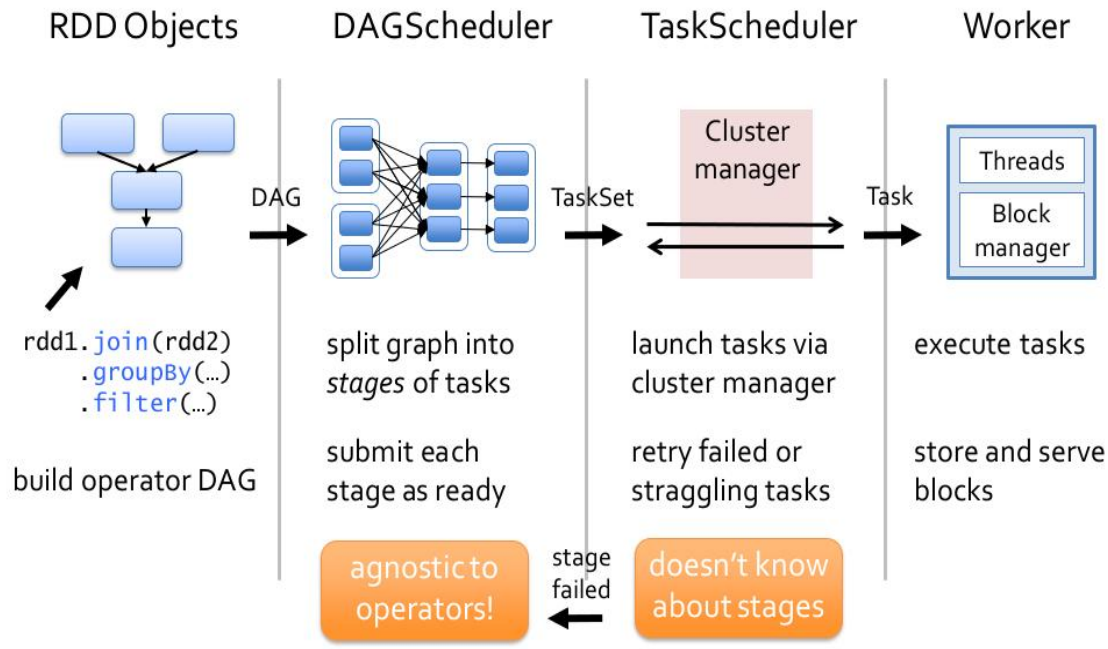
1. wysyłanie zadań do klastra,
2. uruchamianie zadań,
3. obsługę błędów (w tym ponawianie zadań).
4. Wysyła komunikaty do nadrzędnego DAG Scheduler.

Liczbę etapów można wyliczyć następująco:

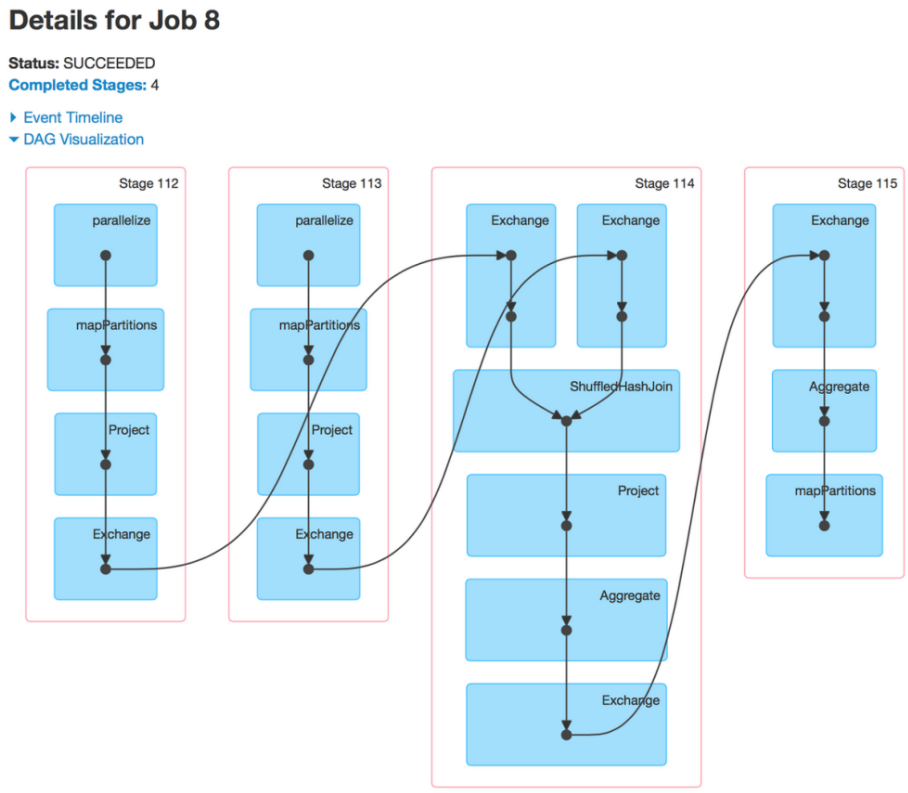
liczba etapów = liczba szerokich transformacji + 1

W ogólności etapy są wykonywane sekwencyjnie oprócz etapu join, dla którego dwa etapy poprzedzające złączenie mogą być uruchamiane równolegle. Zadania (taski) zgrupowane w etapie uruchamiane są równolegle. Liczba tasków zależy od liczby partycji RDD.

Powyższy schemat przetwarzania w Sparku można więc zobrazować w ten sposób:



W aplikacji WWW sterownika Sparka można podejrzeć szczegółowy obraz zbudowanego grafu:



Obsługa różnych formatów plików

Spark potrafi obsługiwać wiele typów plików, a że powstał z myślą o zastosowaniu m.in. w systemie Hadoop to potrafi obsługiwać również dane w formatach tej platformy.

Przykładowe formaty plików obsługiwane przez Sparka:

- SequenceFile
- CSV
- Avro
- ORC
- Parquet
- JSON
- XML
- txt

Tabela poniżej dla tych samych danych pliku pokazuje rozmari pliku w różnym formacie:

Data	Size
Csv	365 MB
avro	157 MB
Orc	45 MB
parquet	49.4 MB
Json	632 MB

Xml 1.1 GB

Duża efektywność przechowywania danych mają pliki orc oraz parquet.

Spark SQL

SparkSQL to moduł Sparka do przetwarzania danych strukturalnych. Interfejsy pakietu SQL operuje już strukturami danych o pewnych schematach.
Istnieje kilka sposobów pracy ze SparkSQL:

- czysty język SQL
- interfejs API DataFrame
- interfejs API DataSets

SQL

- SparkSQL pozwala wywoływać zapytania SQL o podstawowej składni bądź składni HiveQL.
- Może podłączać się do bazy Hive.
- Dane zwracane są jako DataFrame.
- Istnieje powłoka spark-sql (komunikacja JDBC/ODBC).

DataFrames

Podstawowym typem danych w SparkSQL jest DataFrame, który jest rozproszoną kolekcją danych z nazwanymi kolumnami. Jest jakby tabelą w relacyjnej bazie danych lub ramką danych w R/Python.
Sposoby tworzenia DataFrame:

- Structured data files
- Tables in Hive
- External databases
- RDD

Programowe tworzenie schemata danych
Trzy kroki do stworzenia DF:

1. Stwórz krotkę lub listę RDD z oryginalnego RDD.
2. Stwórz schemat reprezentowany przez obiekt StructType pasujący do struktury krotek (list) z punktu 1.
3. Zastosuj schemat do RDD przez metodę createDataFrame sesji Sparka.

Datasets

Dataset to experimental interface added in Spark 1.6 that tries to provide the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.).
The unified Dataset API can be used both in Scala and Java. Python does not yet have support for the Dataset API, but due to its dynamic nature many of the benefits are already available (i.e. you can access the field of a row by name naturally row.columnName). Full python support will be added in a future release.

Widoki tymczasowe

- Spark pozwala tworzyć tymczasowe widoki na bazie danych w DataFrame, do których następnie można odwoływać się w zapytaniu SQL.
- Widoki tymczasowe są definiowane na czas trwania sesji I znikną, jeśli zakończy się tworząca je sesja.
- Jest możliwość stworzenia globalnego widoku, który będzie istniał aż do zakończenia całej aplikacji Sparka. Tworzy się ona w ramach bazy danych „global_temp".

Data Sources

Czytanie i zapisywanie danych

Domyślnym formatem źródła danych jest parquet, co można zmienić ustawieniem spark.sql.sources.default.

Czytanie I zapis:

```
df = spark.read.load("examples/src/main/resources/users.parquet")
df.select("name", "favorite_color").write.save("namesAndFavColors.parquet")
```

Łatwo jednak wczytać inny rodzaj danych, bo wystarczy wyspecyfikować jego nazwę pełną (np. org.apache.spark.sql.parquet) lub skróconą (json, parquet, jdbc, orc, libsvm, csv, text). Podobnie wygląda zapis.

Obsługa JSON

Czytanie i zapis:

```
df = spark.read.load("examples/src/main/resources/people.json", format="json")

df.select("name", "age").write.save("namesAndAges.parquet", format="parquet")
```

Można podać więcej parametrów:

```
df = spark.read.load("examples/src/main/resources/people.csv",

                     format="csv", sep=":", inferSchema="true", header="true")
```

Uruchamianie SQL bezpośrednio

Zamiast wczytywać pliku można również bezpośrednio go użyć tworząc odpowiednie zapytanie SQL:

```
df = spark.sql("SELECT * FROM parquet.`examples/src/main/resources/users.parquet`")
```

Pliki Parquet

Parquet to format kolumnowy, wspierany przez wiele systemów przetwarzania danych. Spark SQL potrafi czytać I zapisywać pliki Parquet. Zachowują one schemat oryginalnych danych data.

Optymalizacja wykonania przekształceń SQL

Dwa rodzaje trybów optymalizacji:

- Optymalizator regułowy: optymalizator używa zestawu reguł do wykonania zapytania.
- Optymalizator kosztowy: optymalizator szacuje koszty wykonania zapytania i tworzy wiele planów wykonania dla różnych zestawów reguł.

Fazy optymalizacji:

- Wstępna analiza zapytania.
- Drzewo AST (abstract syntax tree) uzyskane od parsera SQL na bazie zapytania.
- Obiekt DataFrame.
- Brakujące / nieznane typy atrybutów, relacji.
- Logiczna optymalizacja
- Zwijanie stałych (constant folding).
- Spychanie predykatów (predicate pushdown).
- Rzutowanie typów.
- Fizyczne planowanie.
- Tworzenie kilku fizycznych planów wykonania.
- Wybór planu wg modelu kosztowego.
- Generacja kodu wykonania planu.
- Java bytecode.

Funkcje statystyczne i matematyczne

W pakiecie pyspark.sql.functions dostępnych jest kilka ciekawych i ważnych funkcji statystyczne.

Generowanie danych losowych

Generacja danych losowych przydaje się do testów istniejących już algorytmów. W pakiecie pyspark.sql.functions istnieją funkcje do generowania kolumn, które mogą zawierać przykładowo wartości z rozkładu normalnego (funkcja randn) lub ciągłego (funkcja rand).

Statystyki sumaryczne

Funkcja df.describe() pozwala poznać szereg wartości statystycznych dla zbioru danych. W jej wyniku dostajemy obiekt DataFrame zawierający następujące informacje dla każdej z kolumn:

- Liczba nie-pustych (nie-nullowych) wpisów w zbiorze.
- Wartość średnia (mean).
- Standardowe odchylenie (standard deviation).
- Wartości min i max dla każdej kolumny liczbowej.

Jeśli mamy DataFrame z dużą liczbą kolumn, można wyświetlić tylko ich podzbiór za pomocą metody select().

Kowariancja i korelacja

Kowariancja to miara jak dwie zmienne zmieniają się względem siebie. Liczba dodatnia oznacza, że istnieje tendencja, że jak jedna zmienna wzrośnie, to druga też wzrośnie. Natomiast wartość ujemna oznaczałaby, że jeśli jedna zmienna wzrośnie, to druga zmaleje.

Przykłady w ćwiczeniach.
Jak widać kowariancja dwóch losowo generowanych kolumn jest bliska 0, natomiast kowariancja wartości kolumny id z sobą samą jest bardzo wysoka.
Kowariancja jest trudna w interpretacji i lepiej do tego nadaje się korelacja. Korelacja jest znormalizowaną miarą kowariancji, która jest łątwiejsza do zrozumienia, ponieważ dostarcza ilościowe pomiary statystycznej zależności między dwiema zmiennymi losowymi.
W przykładzie kolumna id koreluje doskonale z samą sobą, podczas gdy dwie losowo wygenerowane kolumny mają niską wartość korelacji.

Tabulacja krzyżowa/ Cross Tabulation (Contingency Table)

Można utworzyć tabele krzyżowe między dwoma kolumnami I wyliczać częstości ich wystąpień w zbiorze.

Częstość rzeczy

Dostępny jest algorytm określający częstość wystąpienia rzeczy w zbiorze / dokumencie.

Funkcje matematyczne

Dostępne są funckje matematyczn typu jednoargumentowego (sin, cos, floor, ceil) czy dwuargumentowae jak pow (power).

Konwersja na bibliotekę Pandas

Łatwo uzyzskać konwersję z DataFrame na obiekty Pandas. Wystarczy wywołać metodę df.toPandas().
Łatwo odwrócić wtedy tabelę – transpose().
Można utworzyć DataFrame z obiektów Pandas DataFrame używając metody createDataFrame(pandas_df).

```
import numpy as np
import pandas as pd


# Generate a Pandas DataFrame
pdf = pd.DataFrame(np.random.rand(100, 3))

# Create a Spark DataFrame from a Pandas DataFrame using Arrow
df = spark.createDataFrame(pdf)

# Convert the Spark DataFrame back to a Pandas DataFrame using Arrow
result_pdf = df.select("*").toPandas()
```