Apache Spark

iBigWorld:

Innovations for Big Data in a Real World

Prof. dr. Dragan Stojanovic, **UNI**

Prof. dr. Natalija Stojanovic, **UNI**

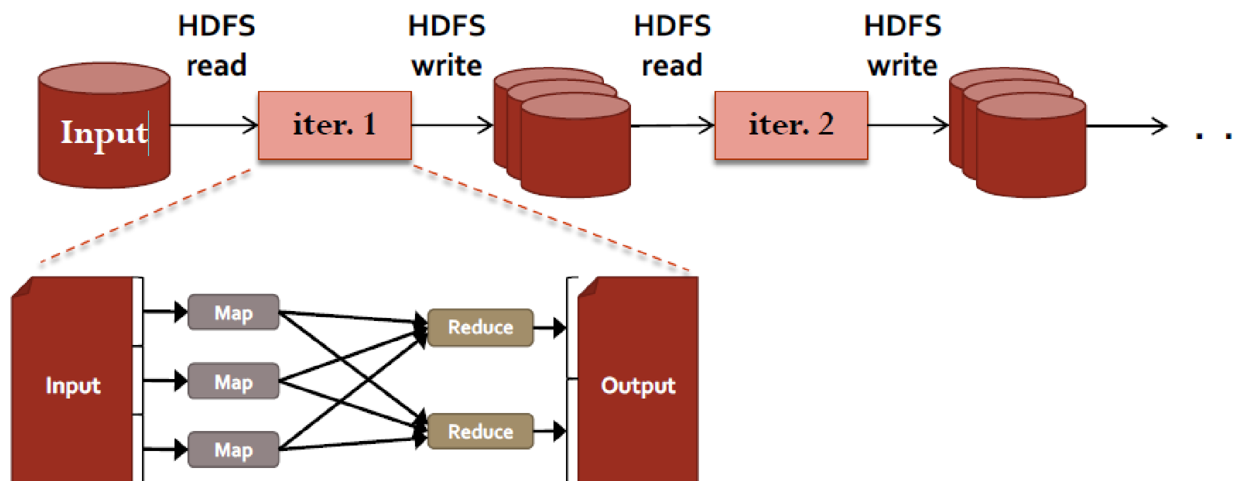
Disclaimer: Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the National Agency (NA). Neither the European Union nor NA can be held responsible for them.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Ograniczenia metody Map Reduce

- Wolny z powodu IO dysku, wysokiej komunikacji i serializacji
- Niewydajne dla:
- Algorytmów iteracyjnych (Machine Learning, Graphs & Network Analysis)
- Interaktywnego wydobywania danych (R, obliczenia podobne do Excelowych, raportowanie ad hoc, wyszukiwanie)



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

MapReduce: słabości i ograniczenia

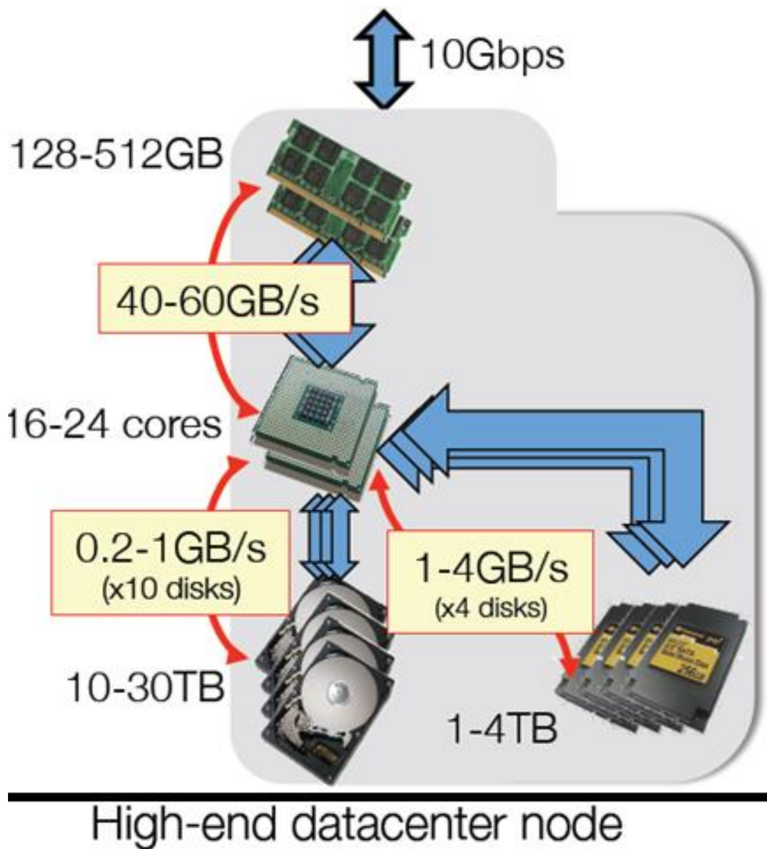
- Wydajność
- Wysoki koszt komunikacji
- Częsty zapis danych wyjściowych na dysk
- Ograniczone wykorzystanie pamięci głównej
- Model programowania
- Trudno zaimplementować wszystko jako program MR
- Wiele kroków MR może być potrzebnych także dla prostych operacji
- Brak struktur kontrolnych i typów danych
- Przetwarzanie w czasie rzeczywistym
- Zadanie MR wymaga przeskanowania całego wejścia
- Przetwarzanie strumieniowe i dostęp losowy niemożliwe

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Rozwiązania?

- Dźwignia na pamięci:
- Wymienić dyski na SSD
- Załaduj dane do pamięci



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Apache Spark

- Apache Spark - https://spark.apache.org/
- Nie jest zmodyfikowaną wersją Hadoopa
- Oddzielny, szybki silnik podobny do MapReduce
- Przechowywanie danych w pamięci dla bardzo szybkich zapytań iteracyjnych
- Ogólne wykresy wykonania i potężne optymalizacje
- Do 100x szybszy niż Hadoop MapReduce
- Zgodność z interfejsami API Hadoop do przechowywania danych
- Możliwość uruchomienia na szczycie klastra Hadoop
- Możliwość odczytu/zapisu do dowolnego
- System wspierany przez Hadoop,
- w tym HDFS, HBase,
- SequenceFiles itp.

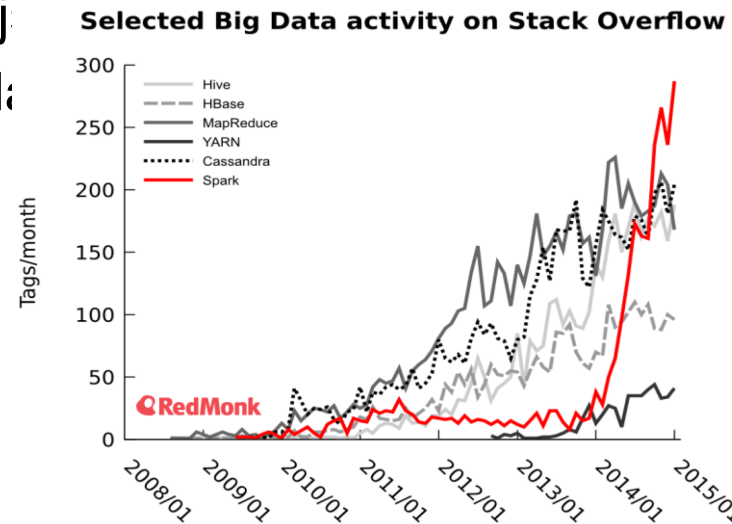
	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc…
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Historia projektu Spark

- Projekt Spark rozpoczął się w 2009 roku
- Opracowany pierwotnie w UC Berkeley's AMPLab przez Matei Zaharia w ramach jego badań doktoranckich
- Open sourced 2010, projekt Apache od 2013
- W 2014 roku Zaharia założył firmę Databricks
- Jest to obecnie najpopularniej
- projekt do analizy dużych d
- Najnowsza wersja: 3.2.1
- 2015 wykres (RedMon



iBigWorld:2020-1-PL01-KA203-082197

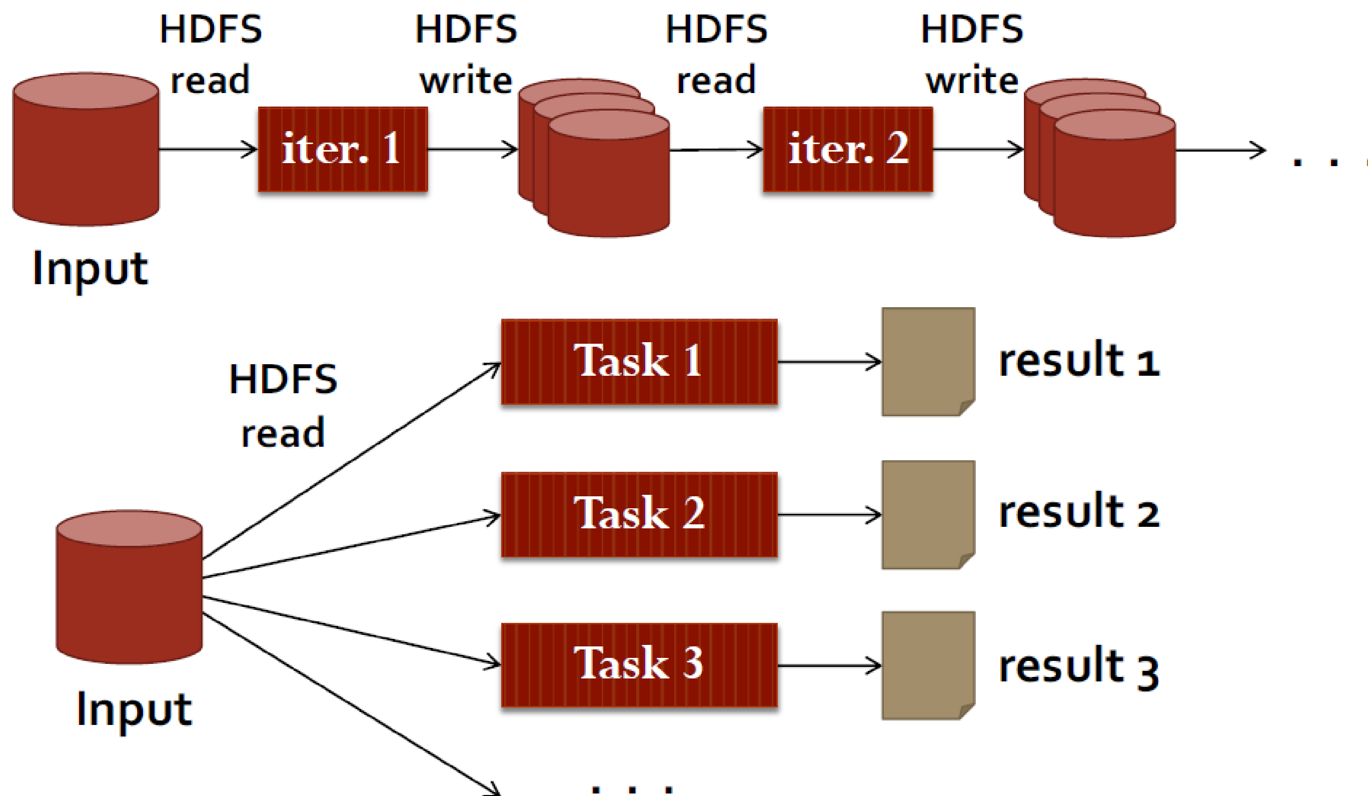
Co-funded by the Erasmus+ Programme of the European Union

Dlaczego nowy model programowania?

- MapReduce znacznie uprościł analizę dużych danych
- Jednak gdy tylko zyskał popularność, użytkownicy chcieli więcej:
- Bardziej złożone, wieloetapowe aplikacje (np. iteracyjne algorytmy grafów i uczenie maszynowe)
- Większej wydajności
- Bardziej interaktywnych zapytań ad-hoc
- Zarówno aplikacje wieloetapowe, jak i interaktywne wymagają szybszego udostępniania danych w ramach zadań równoległych

Co-funded by the Erasmus+ Programme of the European Union

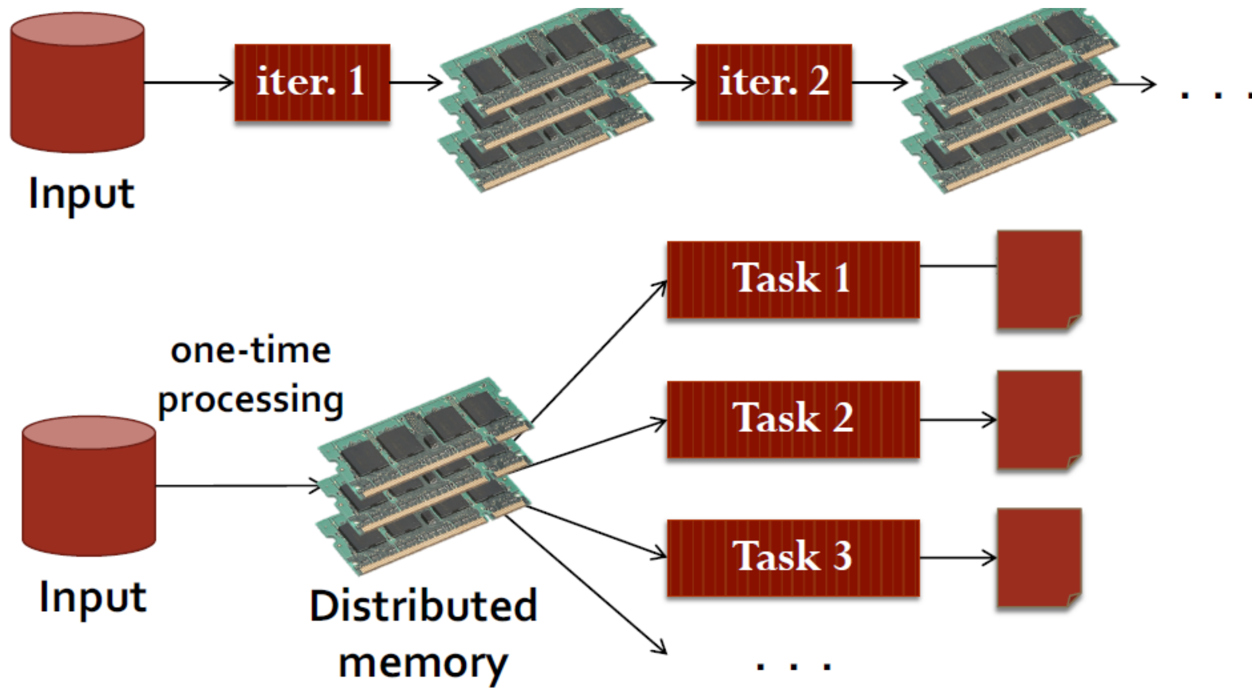
Współdzielenie danych w MapReduce



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Współdzielenie danych w Spark

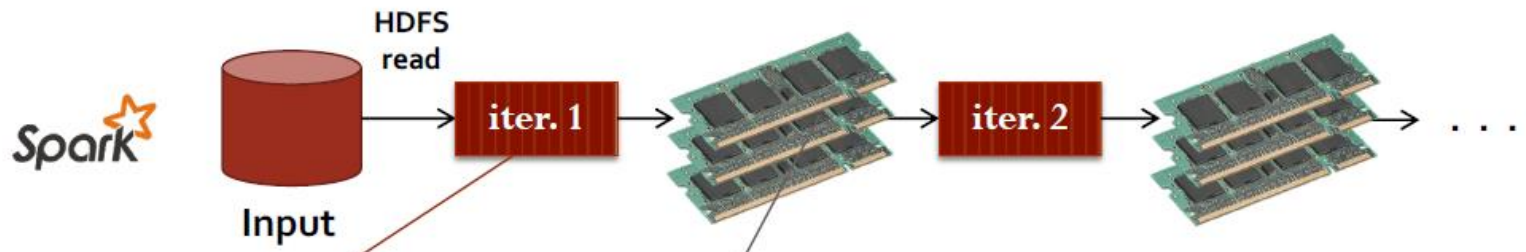


10-100× faster than network and disk

iBigWorld:2020-1-PL01-KA203-082197

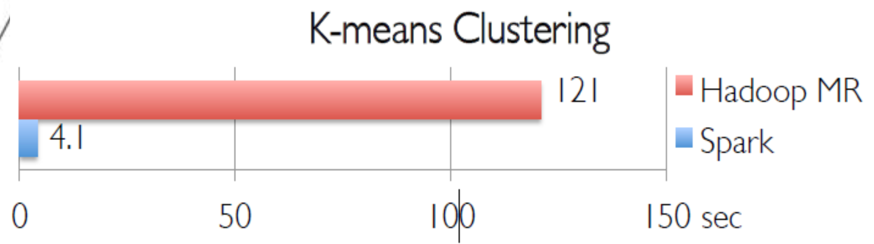
Co-funded by the Erasmus+ Programme of the European Union

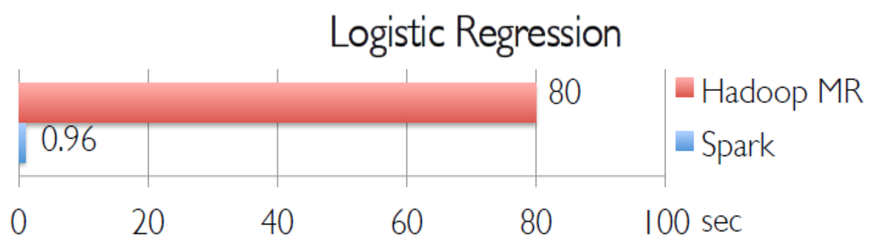
Przepływ danych Spark



Not tied to 2 stage Map Reduce paradigm

1. Extract a working set
2. Cache it
3. Query it repeatedly

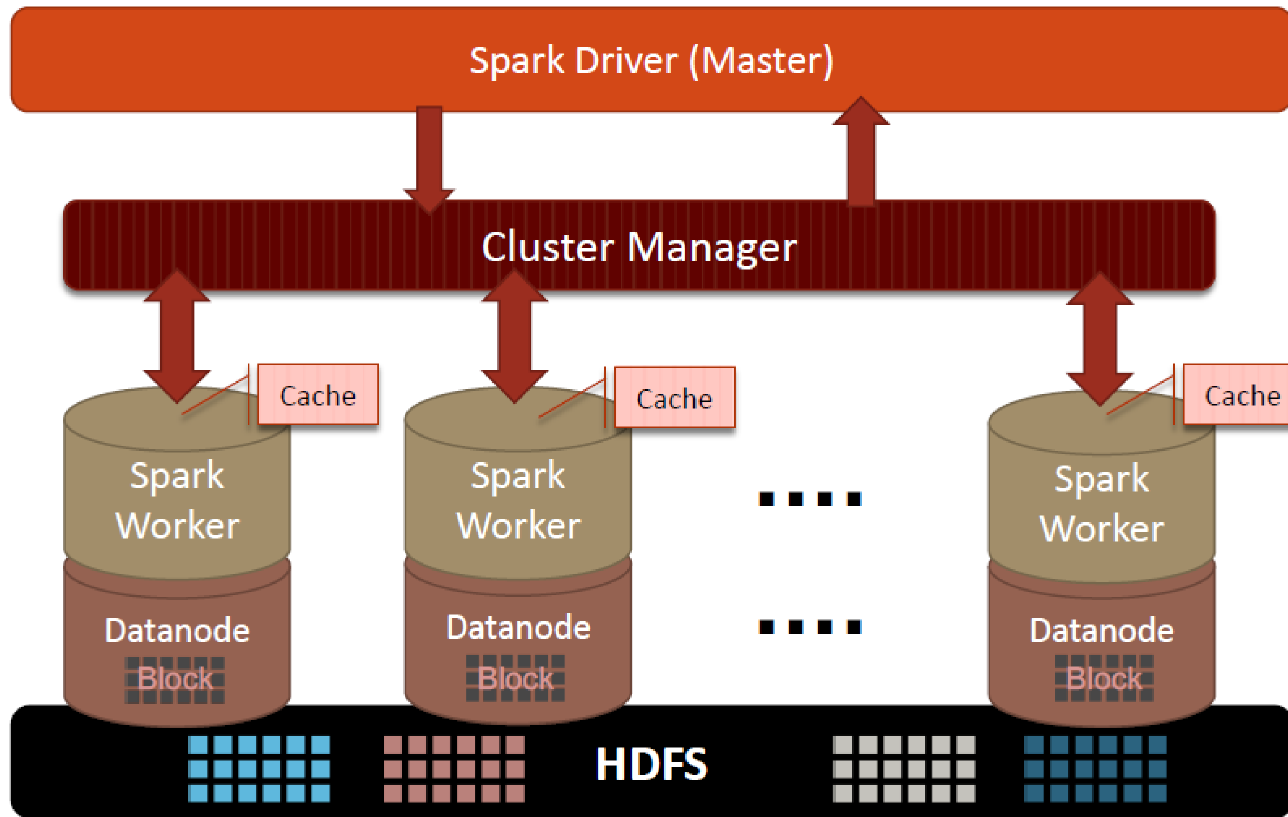




iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

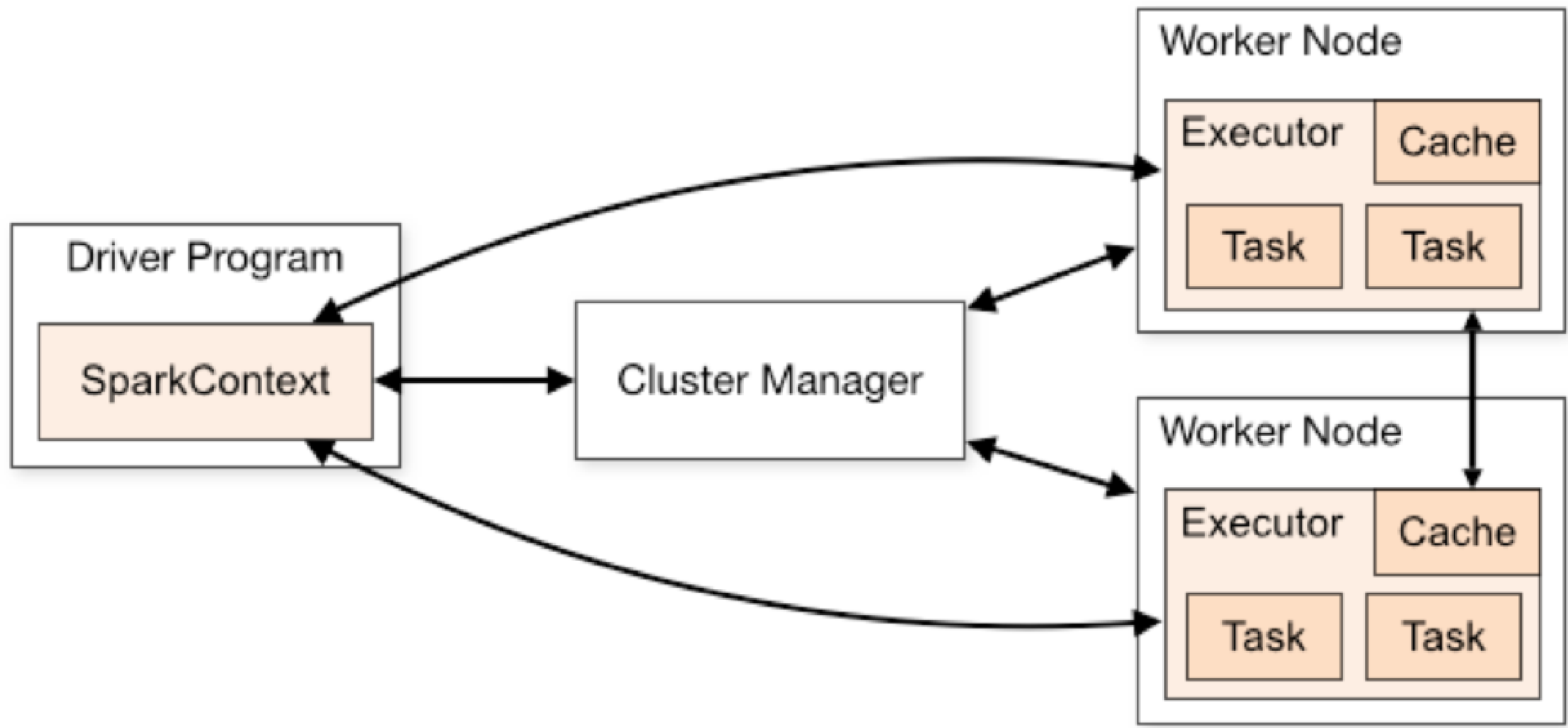
Spark architektura



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Anatomia architektury Sparka



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

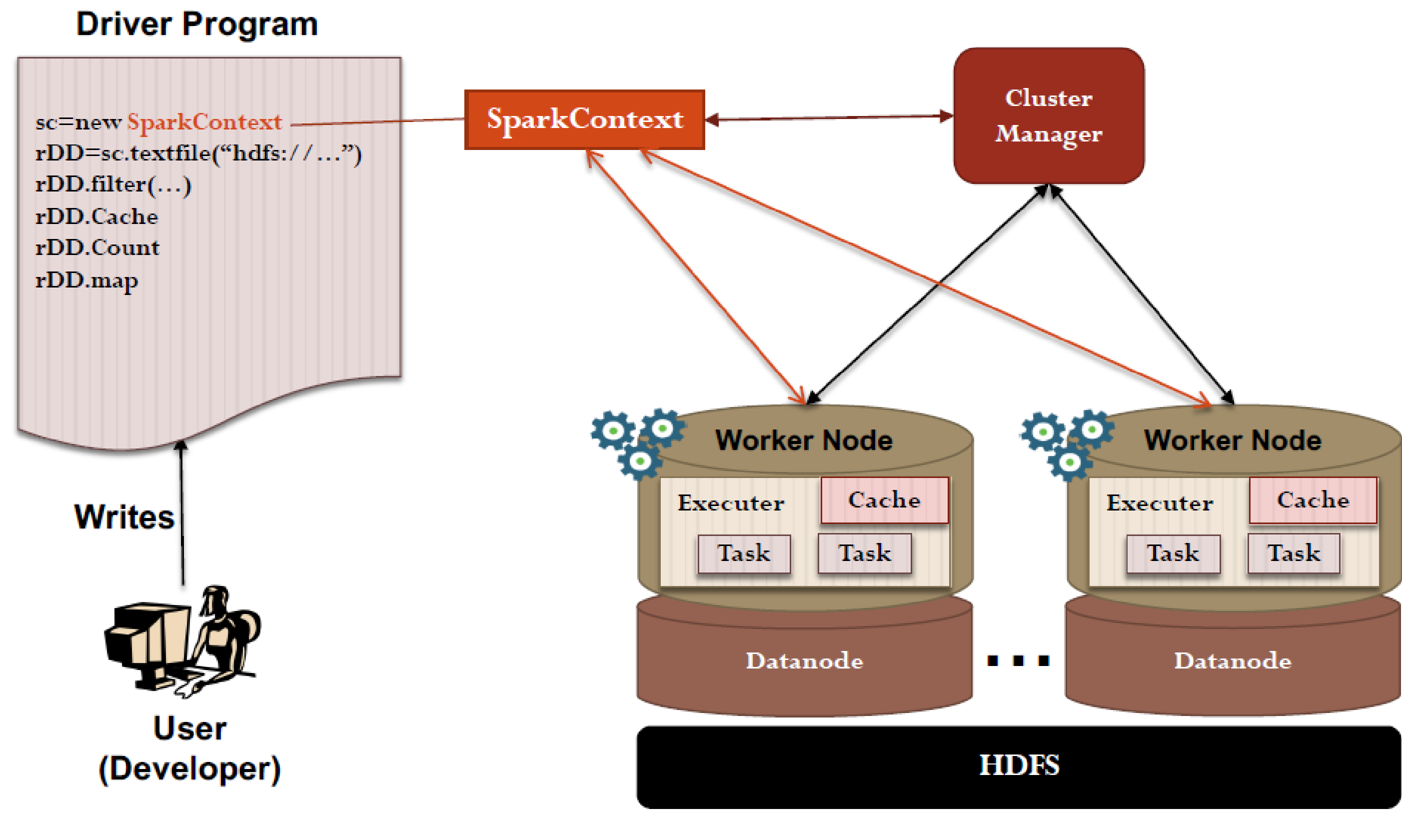
Anatomia architektury Sparka

- Aplikacje uruchamiane są jako niezależne zestawy procesów na klastrze, koordynowane przez obiekt SparkContext w Twoim głównym programie (programie sterownika)
- Każda aplikacja dostaje swoje własne procesy wykonawcze, które pozostają na czas działania całej aplikacji i wykonują zadania w wielu wątkach.
- Aby uruchomić na klastrze, SparkContext może połączyć się z kilkoma rodzajami menedżerów klastrów (Standalone , Mesos lub YARN), które przydzielają zasoby między aplikacjami
- Po podłączeniu, Spark:
- Pozyskuje executory na węzłach w klastrze, które uruchamiają obliczenia i przechowują dane
- Następnie wysyła twój kod aplikacji do executorów.
- Wreszcie, SparkContext wysyła zadania do executorów, aby je uruchomić.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

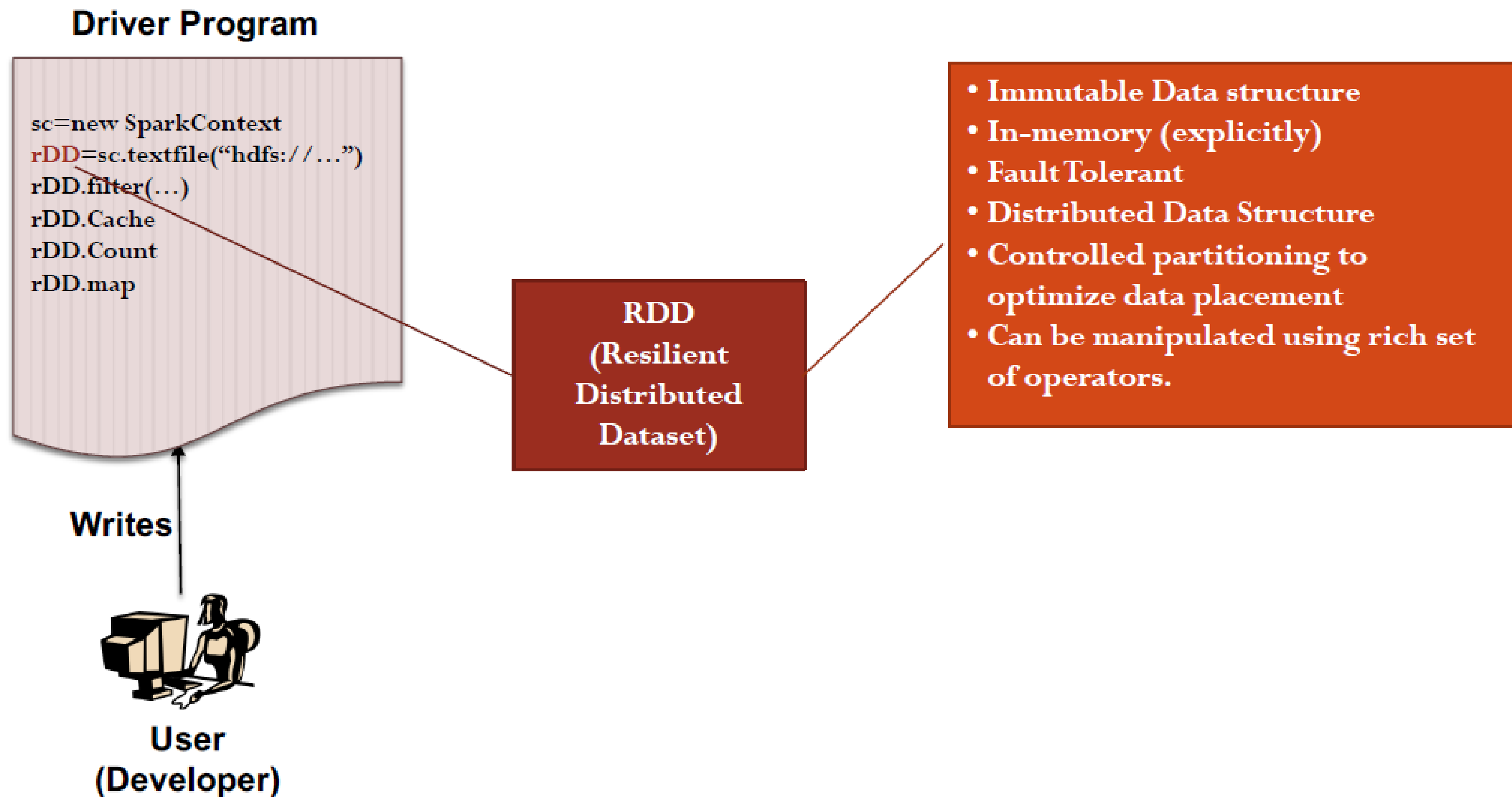
Model programowania Spark



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Model programowania Spark



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Model programowania Spark

- Kluczowa idea: Resilient Distributed Datasets (RDD)
- Rozproszone kolekcje obiektów, które mogą być buforowane w pamięci przez węzły klastra
- Manipulowane przez różne równoległe operatory
- Automatycznie odbudowywane w przypadku awarii
- Mogą być przechowywane w pamięci, na dysku lub w obu tych miejscach
- Możliwość partycjonowania w celu kontroli przetwarzania równoległego
- Interfejs
- Czyste, zintegrowane językowo API dla Scali, Pythona, Javy i R.
- Możliwość interaktywnego użycia z konsoli Scali i powłoki Pythona

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Główna abstrakcja Sparka: RDD

- Resilient Distributed Datasets lub RDD to abstrakcje pamięci rozproszonej, które umożliwiają programiście wykonywanie równoległych obliczeń w pamięci na dużych klastrach w sposób wysoce odporny na błędy.Obecnie 2 rodzaje RDD:Kolekcje równoległe: tworzone przez wykonywanie operatorów na istniejącym zbiorze danych. Deweloper może określić liczbę wycinków, na które ma zostać wycięty zbiór danych. Idealnie 2-3 plasterki na procesor.Zestawy danych Hadoop: tworzone z dowolnego pliku przechowywanego na HDFS lub innych systemach pamięci masowej obsługiwanych przez Hadoop (S3, HBase itp.). Są one tworzone przy użyciu operatora textFile SparkContext. Domyślna liczba plasterków w tym przypadku to 1 plasterek na blok pliku.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Zrównoleglone i rozproszone zbiory danych

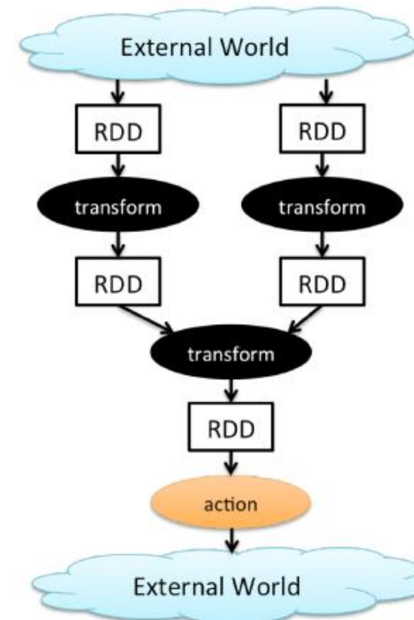
- Zrównoleglone kolekcje są tworzone przez wywołanie metody Parallelize SparkContext w istniejącej kolekcji w programie sterownika. val data = Tablica (1, 2, 3, 4, 5) val distData = sc.parallelize(dane)Po utworzeniu distData może być operowana równolegle. distData.reduce((a, b) => a + b)Rozproszone zestawy danych można tworzyć z dowolnego źródła magazynu obsługiwanego przez Hadoop (HDFS, HBase, S3 itp.) przy użyciu metody textFile SparkContext. Ta metoda pobiera identyfikator URI pliku i odczytuje go jako zbiór wierszy. scala> val distFile = sc.textFile("data.txt")Po utworzeniu distFile może być obsługiwany przez operacje. distFile.map(s => s.length).reduce((a, b) => a + b)

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Koniec operacji RDDs

- Ogólne DAG operacji (np. map-reduce-reduce lub nawet bardziej złożone kombinacje)



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Koniec operacji RDD

Transformations

- Create a new dataset from and existing one.
- Lazy in nature. They are executed only when some action is performed.
- Example :
  - Map(func)
  - Filter(func)
  - Distinct()

Actions

- Returns to the driver program a value or exports data to a storage system after performing a computation.
- Example:
  - Count()
  - Reduce(func)
  - Collect
  - Take()

Persistence

- For caching datasets in-memory for future operations.
- Option to store on disk or RAM or mixed (Storage Level).
- Example:
  - Persist()
  - Cache()

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Transformacje

- Operacje przekształceń na RDD, które zwracają obiekty RDD lub kolekcje RDDnp. mapowanie, filtrowanie, dołączanie, grupowanie itp.Transformacje są leniwe i nie są wykonywane natychmiast, ale tylko wtedy, gdy akcja wymaga zwrócenia wyniku do programu sterownika.Ten projekt umożliwia Sparkowi wydajniejsze działanie.Na przykład możemy zdać sobie sprawę, że zestaw danych utworzony za pomocą mapy zostanie użyty w redukcji i zwróci tylko wynik redukcji do sterownika, a nie większy zmapowany zestaw danych.Załóżmy, że ze unionRDD potrzebujemy tylko pierwszych 5 wierszy.Jeśli jesteśmy leniwi, nie ma nawet potrzeby obliczania całej sumy dwóch RDD, ponieważ gdy znajdziemy pierwsze 5 wierszy, możemy się zatrzymać.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Dwa rodzaje transformacji

- Wąskie przekształcenia. Są wynikiem mapowania, filtrowania itp. i działają na danych tylko z jednej partycji (tj. są samowystarczalne).Wyjściowy RDD ma partycje pochodzące z jednej partycji nadrzędnego RDD.Spark grupuje wąskie transformacje jako etap.
- Szerokie przemiany
- Są wynikiem groupByKey i ReduceByKey. Dane wymagane do obliczenia rekordów w jednej partycji mogą znajdować się w wielu partycjach nadrzędnego RDD.Wszystkie krotki z tym samym kluczem muszą znaleźć się w tej samej partycji, przetwarzanej przez to samo zadanie.Spark musi wykonać tasowanie RDD, które przesyła dane w klastrze i skutkuje nowym etapem z nowym zestawem partycji.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

iBIG World

Transformacja (1)

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(*func*)	Similar to map, but runs separately on each partition of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(*func*)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
sample(*withReplacement*, *fraction*, *seed*)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed.
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numTasks argument to set a different number of tasks.
reduceByKey(*func*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Transformacja (2)

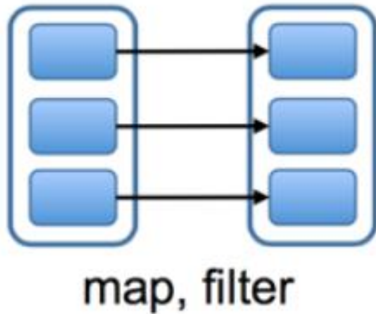
Transformation	Meaning
aggregateByKey(*zeroValue*)(*seqOp*, *combOp*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
sortByKey([*ascending*], [*numTasks*])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean `ascending` argument.
join(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through `leftOuterJoin`, `rightOuterJoin`, and `fullOuterJoin`.
cogroup(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples. This operation is also called `groupWith`.
cartesian(*otherDataset*)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).
pipe(*command*, *[envVars]*)	Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.
coalesce(*numPartitions*)	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
repartition(*numPartitions*)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
repartitionAndSortWithinPartitions(*partitioner*)	Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. This is more efficient than calling `repartition` and then sorting within each partition because it can push the sorting down into the shuffle machinery.

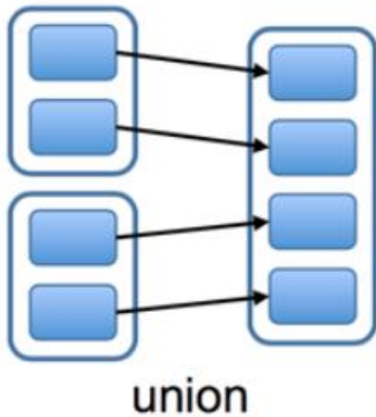
iBigWorld:2020-1-PL01-KA203-082197

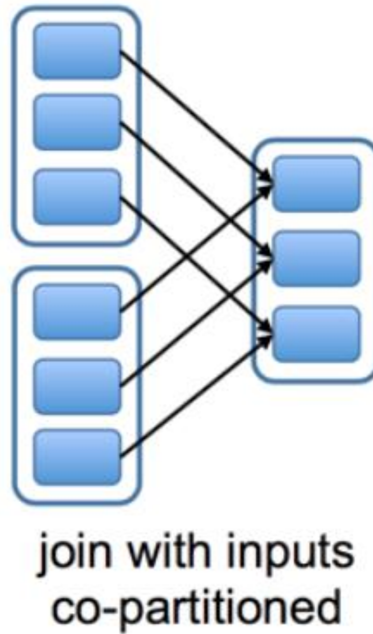
Co-funded by the Erasmus+ Programme of the European Union

Zależności RDD.

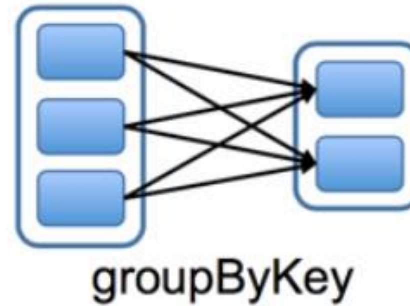
Narrow Dependencies:

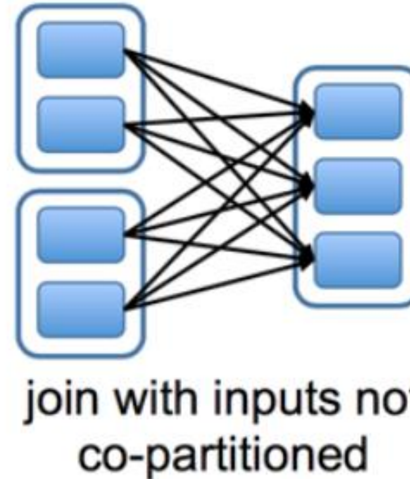






Wide Dependencies:





iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Załóżmy, że nasz RDD zawiera listę{1,2,3}

Podstawowe przekształcenia RDD

- Załóżmy, że nasz RDD zawiera listę{1,2,3}

```
map()           rdd.map(x => x + 2)                {3,4,5}
flatMap()       rdd.flatMap(x => List(x-1,x,x+1))

{0,1,2,1,2,3,2,3,4}
filter()        rdd.filter(x => x>1)               {2,3}
distinct()      rdd.distinct()                     {1,2,3}
sample()        rdd.sample(false,0.2)              non-predictable
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Działania (Actions)

- Akcje to operacje zwracające wartości, tj. każda operacja RDD, która zwraca wartość dowolnego typu poza RDD, jest akcjąnp. Zmniejsz, Policz, Zbierz, Weź, Zapisz jako, …Akcje są synchroniczne. Wyzwalają wykonanie transformacji RDD w celu zwrócenia wartości.Dopóki nie zostanie uruchomiona żadna akcja, dane, które mają zostać przetworzone, nie są nawet dostępneTylko działania mogą zmaterializować cały proces na rzeczywistych danychPowoduje, że dane są zwracane do kierowcy lub zapisywane na wyjściuWywołaj pobieranie danych i wykonywanie wszystkich transformacji na RDD

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Actions

Action	Meaning
reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(n)	Return an array with the first n elements of the dataset.
takeSample(withReplacement, num, [seed])	Return an array with a random sample of num elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(n, [ordering])	Return the first n elements of the RDD using either their natural order or a custom comparator.
saveAsTextFile(path)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.
saveAsSequenceFile(path) (Java and Scala)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
saveAsObjectFile(path) (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded usingSparkContext.objectFile().
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(func)	Run a function func on each element of the dataset. This is usually done for side effects such as updating an Accumulator or interacting with external storage systems. Note: modifying variables other than Accumulators outside of the foreach() may result in undefined behavior. See Understanding closures for more details.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Niektóre przykłady działań

- Załóżmy, że nasz RDD zawiera listę{1,2,3}

collect()	rdd.collect()	{1,2,3}
count()	rdd.count()	3
countByValue()	rdd.countByValue()	{(1,1),(2,1),(3,1)}
take()	rdd.take(2)	{1,2}
top()	rdd.top(2)	{3,2}
reduce()	rdd.reduce((x,y) => x+y)	6
foreach()	rdd.foreach(func)	func applied to each element of rdd

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Trwałość

- Jedną z najważniejszych funkcji platformy Spark jest utrwalanie (lub buforowanie) zestawu danych w pamięci między operacjami.Domyślnie każdy RDD jest przeliczany za każdym razem, gdy uruchamiasz na nim akcję, chyba że utrwalisz RDD.Kiedy utrwalasz RDD x, każdy węzeł przechowuje w pamięci wszystkie partycje x, które oblicza, i wykorzystuje je ponownie w innych akcjach na x. Dzięki temu przyszłe działania będą znacznie szybsze.Możesz oznaczyć RDD do utrwalenia za pomocą metod persist() lub cache(). Gdy jest obliczany po raz pierwszy w akcji, będzie przechowywany w pamięci węzłów.Pamięć podręczna Sparka jest odporna na awarie – jeśli jakakolwiek partycja RDD zostanie utracona, zostanie ona automatycznie przeliczona przy użyciu transformacji, które ją pierwotnie utworzyły.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

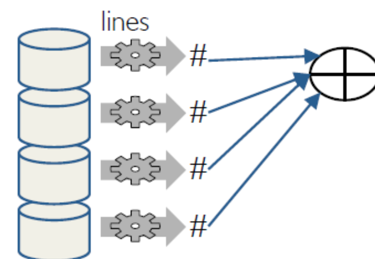
cache()

- Spark model programistyczny

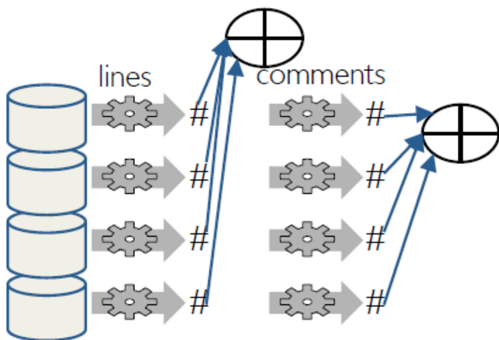
```
lines = sc.textFile("...", 4)

print lines.count()
```

count() causes Spark to:
- read data
- sum within partitions
- combine sums in driver




```
lines = sc.textFile("...", 4)
comments = lines.filter(isComment)
print lines.count(), comments.count()
```



Spark recomputes lines:
- read data (again)
- sum within partitions
- combine sums in driver

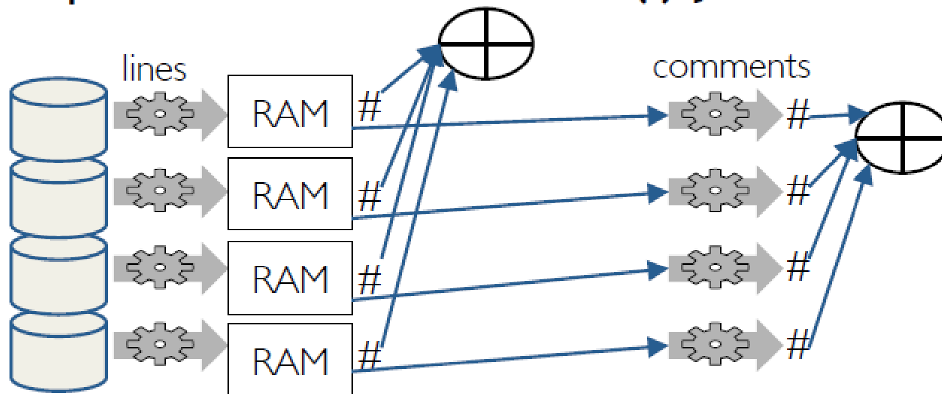
iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

cache()

- Caching RDDs

```
lines = sc.textFile("...", 4)
lines.cache() # save, don't recompute!
comments = lines.filter(isComment)
print lines.count(),comments.count()
```



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

persist() - Poziom przechowywania

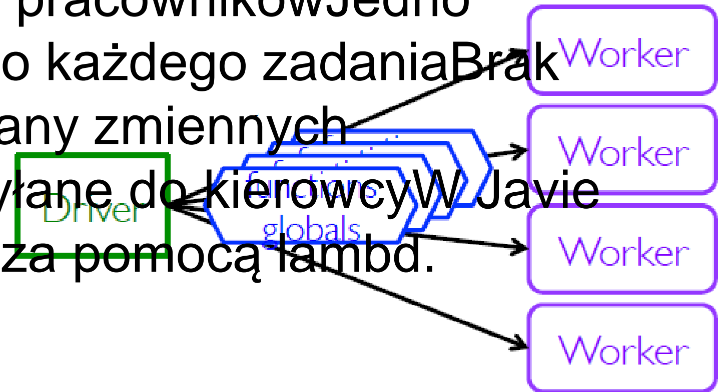
- Używając persist(), określ poziom przechowywania dla utrwalania RDD.Cache() to tylko krótki opis domyślnego poziomu pamięci, którym jest MEMORY_ONLY.Poziom przechowywania dla utrwalania(*):TYLKO PAMIĘĆMEMORY_AND_DISKMEMORY_ONLY_SER, MEMORY_AND_DISK_SERDISK_ONLYMEMORY_ONLY_2, MEMORY_AND_DISK_2 itd.Który poziom przechowywania jest najlepszy? Kilka rzeczy do rozważenia:Staraj się zachować jak najwięcej w pamięciSerializacja sprawia, że obiekty są znacznie bardziej efektywne pod względem przestrzeniStaraj się nie rozlewać na dysk, chyba że funkcje, które obliczają twoje zbiory danych, są drogieUżywaj replikacji tylko wtedy, gdy chcesz mieć odporność na awarie

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Zamknięcia iskier

- „Zamknięcie to szczególny rodzaj obiektu, który łączy dwie rzeczy: funkcję i środowisko, w którym ta funkcja została utworzona.” - MDNSpark automatycznie tworzy zamknięcia dlaFunkcje działające na RDD u pracownikówWszelkie zmienne globalne używane przez tych pracownikówJedno zamknięcie na pracownikaWysyłane do każdego zadaniaBrak komunikacji między pracownikamiZmiany zmiennych globalnych u pracowników nie są wysyłane do kierowcyW Javie domknięcia można zaimplementować za pomocą lambd.



iBigWorld:2020-1-PL01-KA203-082197
Co-funded by the Erasmus+ Programme of the European Union

Rozważ przypadki użycia

- Zadania iteracyjne lub pojedyncze z dużymi zmiennymi globalnymiWysyłanie dużej tabeli przeglądowej tylko do odczytu do pracownikówWysyłanie dużego wektora cech w algorytmie ML do pracownikówZliczanie zdarzeń, które występują podczas wykonywania zadaniaIle wierszy wejściowych było pustych?Ile rekordów wejściowych zostało uszkodzonych?

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Spark współdzielone zmienne

- Zmienne emisji
- Wydajne wysyłanie dużej wartości tylko do odczytu do wszystkich pracowników
- Zapisane u pracowników do wykorzystania w co najmniej jednej operacji Spark.
- Podobnie jak wysyłanie dużej tabeli przeglądowej tylko do odczytu do wszystkich węzłów
- Akumulatory
- Zagregowane wartości od pracowników z powrotem do kierowcy
- Tylko kierowca ma dostęp do wartości akumulatoraW przypadku zadań akumulatory są tylko do zapisu
- Służy do liczenia błędów zauważonych w RDD wśród pracowników

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Zmienne emisji

- Zachowaj zmienną tylko do odczytu w pamięci podręcznej pracowników
- Wysyłaj do każdego pracownika tylko raz zamiast każdego zadania
- Przykład: wydajnie daj każdemu pracownikowi duży zbiór danych
- Zwykle dystrybuowane przy użyciu wydajnych algorytmów rozgłoszeniowych

```
At the driver:
>>> broadcastVar = sc.broadcast([1, 2, 3])

At a worker (in code passed via a closure)
>>> broadcastVar.value
[1, 2, 3]
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Zmienne emisji — przykład

- Wyszukiwanie kodu kraju dla znaków wywoławczych radia HAM

```
# Lookup the locations of the call signs on the
# RDD contactCounts. We load a list of call sign
# prefixes to country code to support this lookup
signPrefixes = loadCallSignTable()

def processSignCount(sign_count, signPrefixes):
    country = lookupCountry(sign_count[0], signPrefixes)
    count = sign_count[1]
    return (country, count)

countryContactCounts = (contactCounts
                        .map(processSignCount)
                        .reduceByKey((lambda x, y: x+ y)))
```

Expensive to send large table
(Re-)sent for every processed file

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Zmienne emisji — przykład

- Wyszukiwanie kodu kraju dla znaków wywoławczych radia HAM

```
# Lookup the locations of the call signs on the
# RDD contactCounts. We load a list of call sign
# prefixes to country code to support this lookup
signPrefixes = sc.broadcast(loadCallSignTable())

def processSignCount(sign_count, signPrefixes):
    country = lookupCountry(sign_count[0], signPrefixes.value)
    count = sign_count[1]
    return (country, count)

countryContactCounts = (contactCounts
                        .map(processSignCount)
                        .reduceByKey((lambda x, y: x+ y)))
```

Efficiently sent once to workers

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Akumulartory

Zmienne, do których można „dodać” tylko operacje asocjacyjneSłuży do efektywnego wdrażania równoległych liczników i sumTylko kierowca może odczytać wartość akumulatora, a nie zadania w executorach

```
>>> accum = sc.accumulator(0)
>>> rdd = sc.parallelize([1, 2, 3, 4])
>>> def f(x):
>>>   global accum
>>>   accum += x

>>> rdd.foreach(f)
>>> accum.value
Value: 10
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Przykłady akumulatorów

- Liczenie pustych linii

```
file = sc.textFile(inputFile)
# Create Accumulator[Int] initialized to 0
blankLines = sc.accumulator(0)

def extractCallSigns(line):
    global blankLines # Make the global variable accessible
    if (line == ""):
        blankLines += 1
    return line.split(" ")

callSigns = file.flatMap(extractCallSigns)
print "Blank lines: %d" % blankLines.value
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Akumulatory

- Zadania u pracowników nie mają dostępu do wartości akumulatora. Zadania widzą akumulatory jako zmienne tylko do zapisu. Akumulatory mogą być używane w akcjach lub transformacjach:
- Akcje: aktualizacja każdego zadania do akumulatora jest stosowana tylko raz
- Transformacje: bez gwarancji (używaj tylko do debugowania)
- Typy: liczby całkowite, podwójne, długie, zmiennoprzecinkoweNiestandardowe typy akumulatorów i operacje agregacji

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Jak działa Spark w czasie wykonywania

Aplikacja użytkownika tworzy RDD, przekształca je i uruchamia akcje.
Daje to DAG (Directed Acyclic Graph) operatorów.
DAG jest kompilowany w etapach.
Każdy etap jest wykonywany jako seria Zadań (jedno Zadanie dla każdej Strefy).
Działania napędzają wykonanie.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Przykład w Scala

```
val textFile = sc.textFile("hdfs://...")

val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)

counts.saveAsTextFile("hdfs://...")
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Ten sam przykład w Python

```
text_file = sc.textFile("hdfs://...”)

counts = text_file.flatMap(lambda line: line.split(" ")) \
            .map(lambda word: (word, 1)) \
            .reduceByKey(lambda a, b: a + b)

output = counts.collect()

output.saveAsTextFile("hdfs://...")
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Ten sam przykład w Java

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");

JavaRDD<String> words = textFile.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String s) { return Arrays.asList(s.split(" ")); }
});

JavaPairRDD<String, Integer> pairs =
        words.mapToPair(new PairFunction<String, String, Integer>() {
                public Tuple2<String, Integer> call(String s) {
                        return new Tuple2<String, Integer>(s, 1); }
        });

JavaPairRDD<String, Integer> counts =
        pairs.reduceByKey(new Function2<Integer, Integer, Integer>() {
                public Integer call(Integer a, Integer b) { return a + b; }
        });
counts.saveAsTextFile("hdfs://...");
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Skończonyprzykład w Java (1)

```
package org.apache.spark.examples;

import scala.Tuple2;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.regex.Pattern;
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Skończony przykład w Java (2)

```
public final class JavaWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");
  public static void main(String[] args) throws Exception {
    if (args.length < 1) {
      System.err.println("Usage: JavaWordCount <file>");
      System.exit(1);
    }
  SparkConf sparkConf = new SparkConf().setAppName("JavaWordCount");
  JavaSparkContext ctx = new JavaSparkContext(sparkConf);
  JavaRDD<String> lines = ctx.textFile(args[0], 1);
  JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String, String>() {
    @Override
    public Iterator<String> call(String s) {
      return Arrays.asList(SPACE.split(s)).iterator();
    }
  });
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Skończony przykład w Java (3)

```
JavaPairRDD<String, Integer> ones = words.mapToPair(
    new PairFunction<String, String, Integer>() {
      @Override
      public Tuple2<String, Integer> call(String s) {
        return new Tuple2<>(s, 1);
      }
    });
  JavaPairRDD<String, Integer> counts = ones.reduceByKey(
    new Function2<Integer, Integer, Integer>() {
      @Override
      public Integer call(Integer i1, Integer i2) {
        return i1 + i2;
      }
    });
  List<Tuple2<String, Integer>> output = counts.collect();
  for (Tuple2<?,?> tuple : output) {
    System.out.println(tuple._1() + ": " + tuple._2());
  }
  ctx.stop();
 }
}
```


iBigWorld:2020-1-PL01-KA203-082197

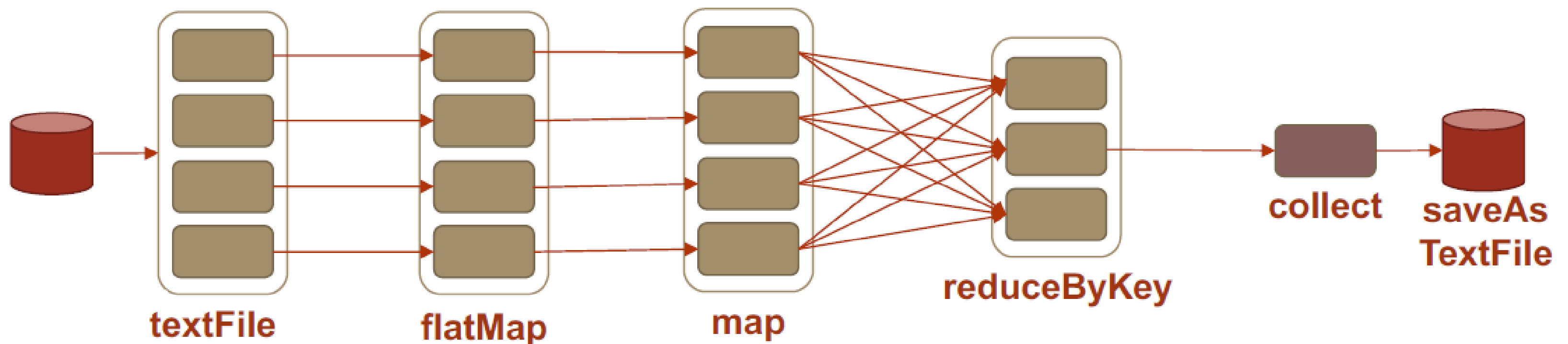
Co-funded by the Erasmus+ Programme of the European Union

Przykład

```
text_file = sc.textFile("hdfs://...")

counts = text_file.flatMap(lambda line: line.split(" ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda a, b: a + b)
output = counts.collect()

output.saveAsTextFile("hdfs://...")
```

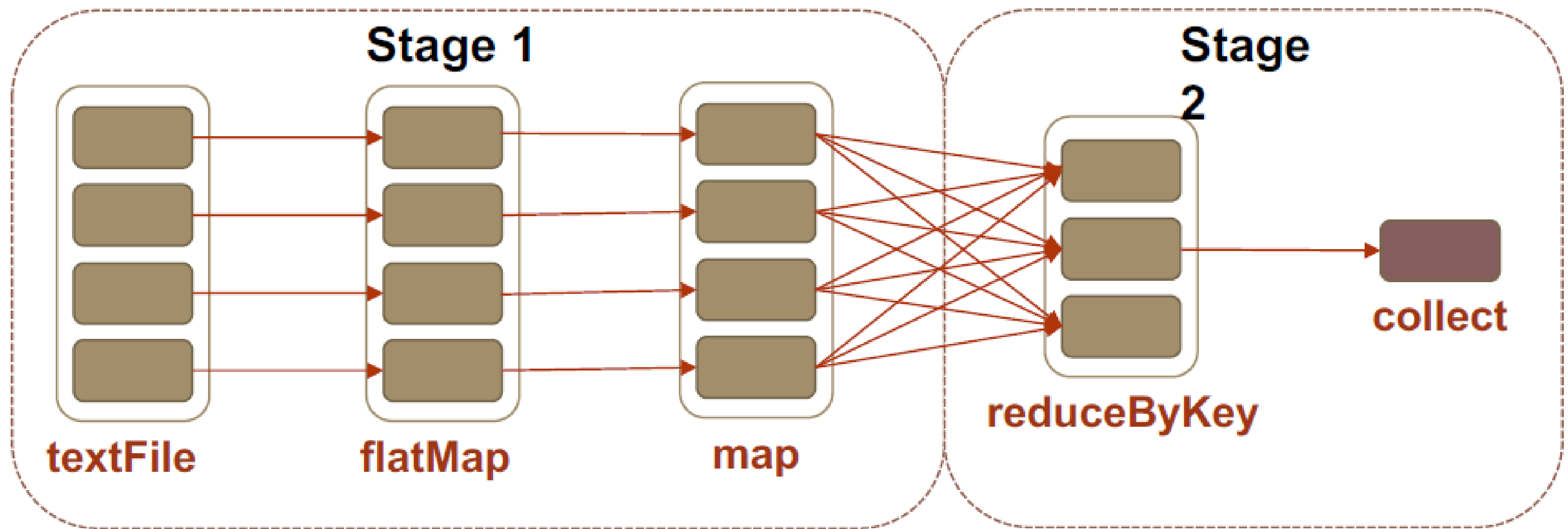


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Plan realizacji

- Etapy to sekwencje RDD, które nie mają pomiędzy nimi Tasowania



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

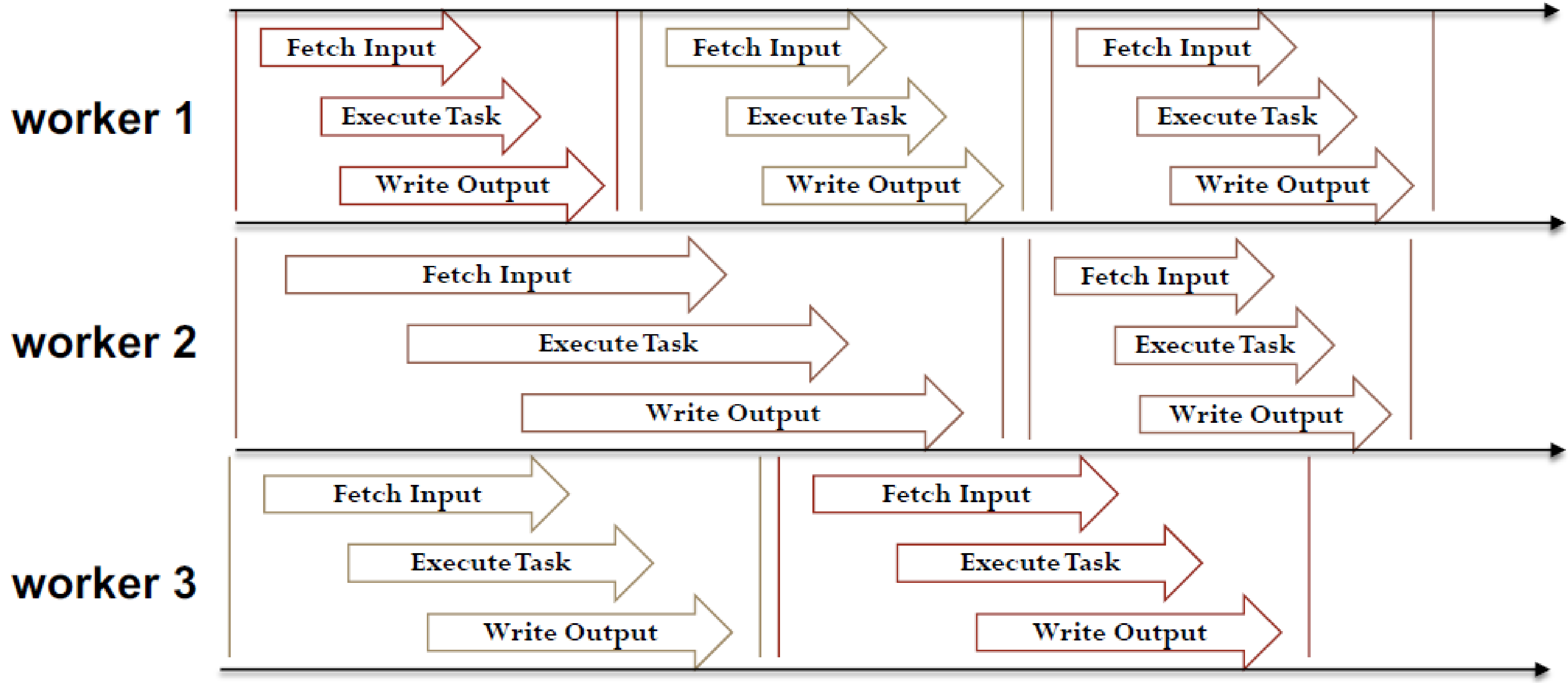
Wykonanie etapu

- Spark:
- Tworzy zadanie dla każdej partycji w nowym RDDPlanuje i przydziela zadania pracownikom (niewolnikom)Wszystko to dzieje się wewnętrznie (nie musisz ni

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

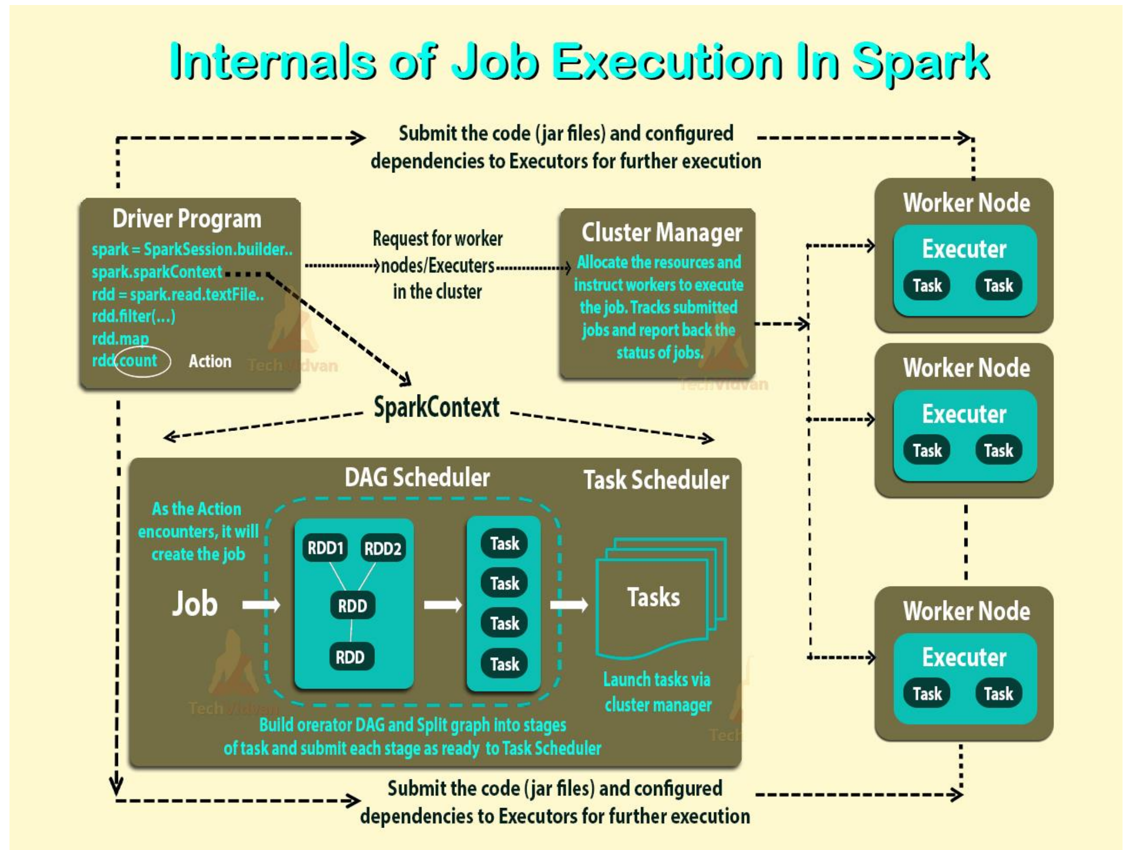
Wykonawca Sparka (pracownicy)



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

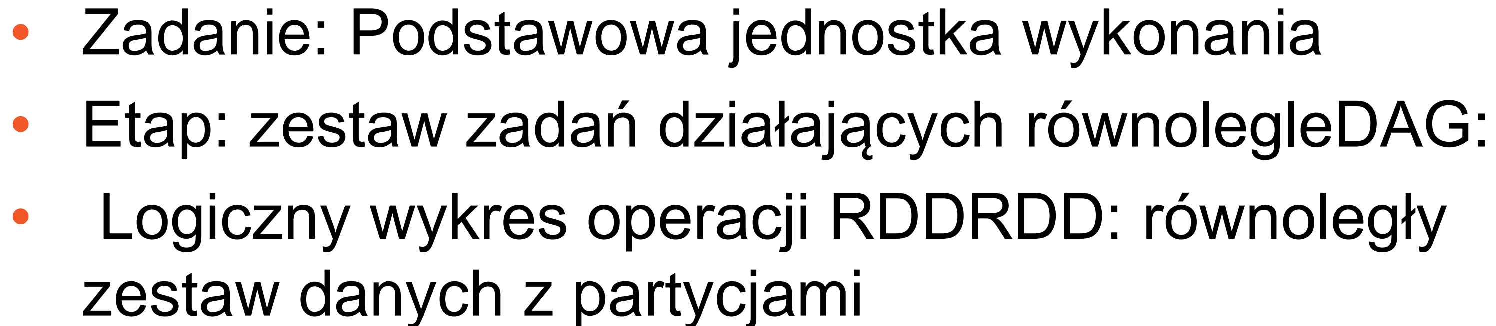
Wykonywanie zleceń w Spark



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Podsumowanie komponentów Spark



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Reprezentacja koncepcyjna

Transformations

RDD

Action

Value

Action

Value

1

```
text_file = sc.textFile("hdfs://...")
```

2

```
counts = text_file.flatMap(lambda line: line.split(“ ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda a, b: a + b)
```

3

```
output = counts.collect()
```

4

```
output.saveAsTextFile("hdfs://...")
```


iBigWorld:2020-1-PL01-KA203-082197

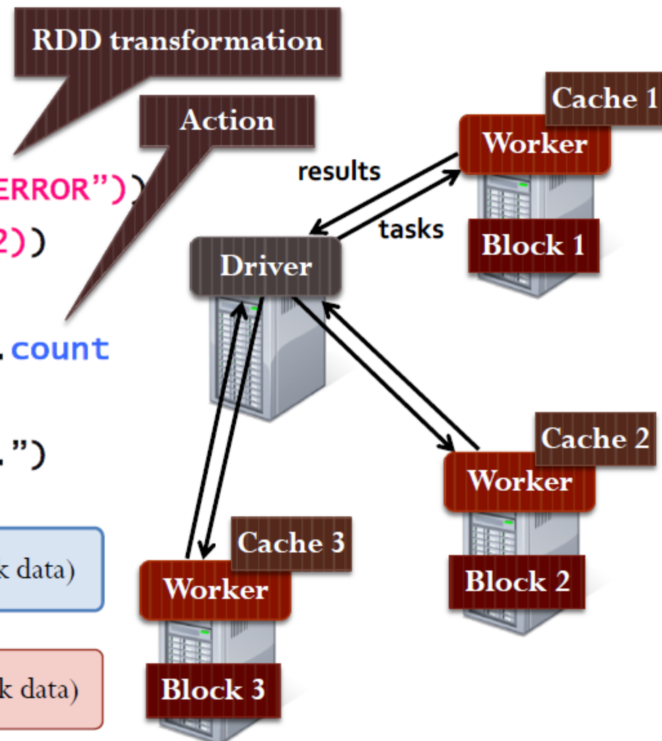
Co-funded by the Erasmus+ Programme of the European Union

Wykonywanie w czasie wykonywania (eksploracja dziennika)

- Załaduj komunikaty o błędach z dziennika do pamięci, a następnie interaktywnie wyszukuj różne wzorce

RDD creation

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
...
cachedMsgs.saveAsTextFile("hdfs://...")
```



Full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

Scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

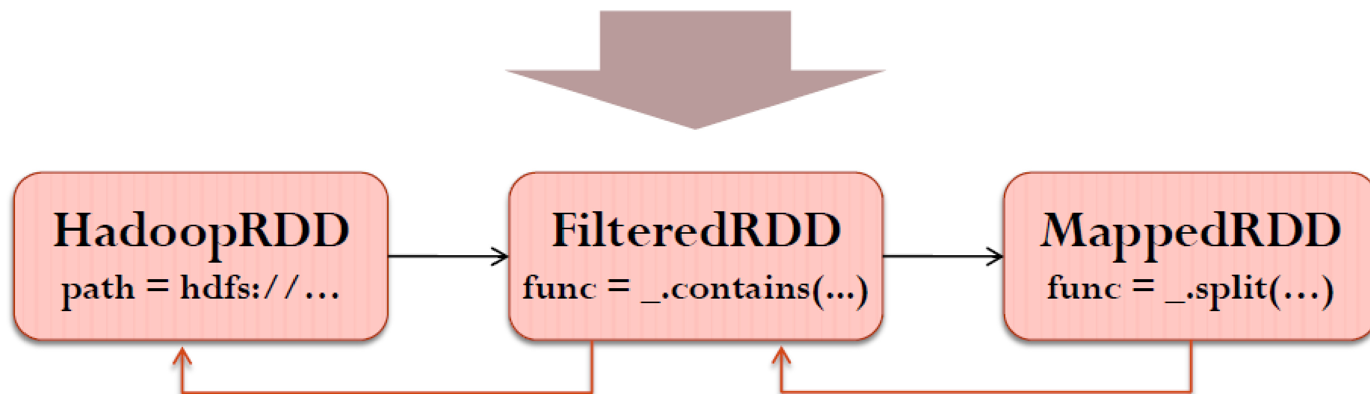
iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Tolerancja błędów

- RDD śledzą serię transformacji użytych do ich zbudowania (ich rodowód)
- Informacje o rodowodzie są wykorzystywane do ponownego obliczania utraconych danych
- Na przykład:

```
messages = textFile(...).filter(_.contains("error"))
                         .map(_.split('\t')(2))
```



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Przykład: regresja logistyczna:

```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Load data in memory once

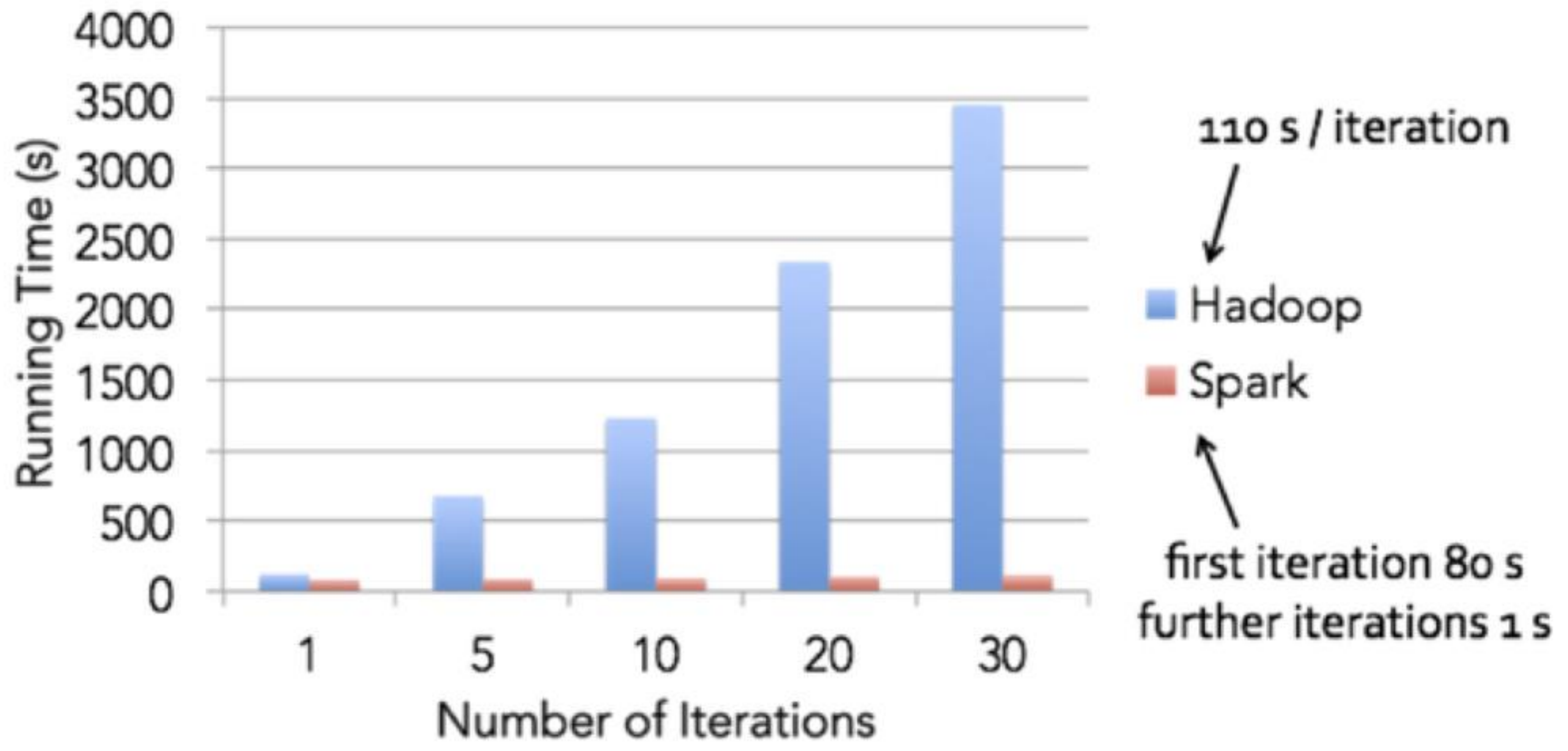
Initial parameter vector

Repeated MapReduce steps to do gradient descent

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Wydajność regresji logicznej

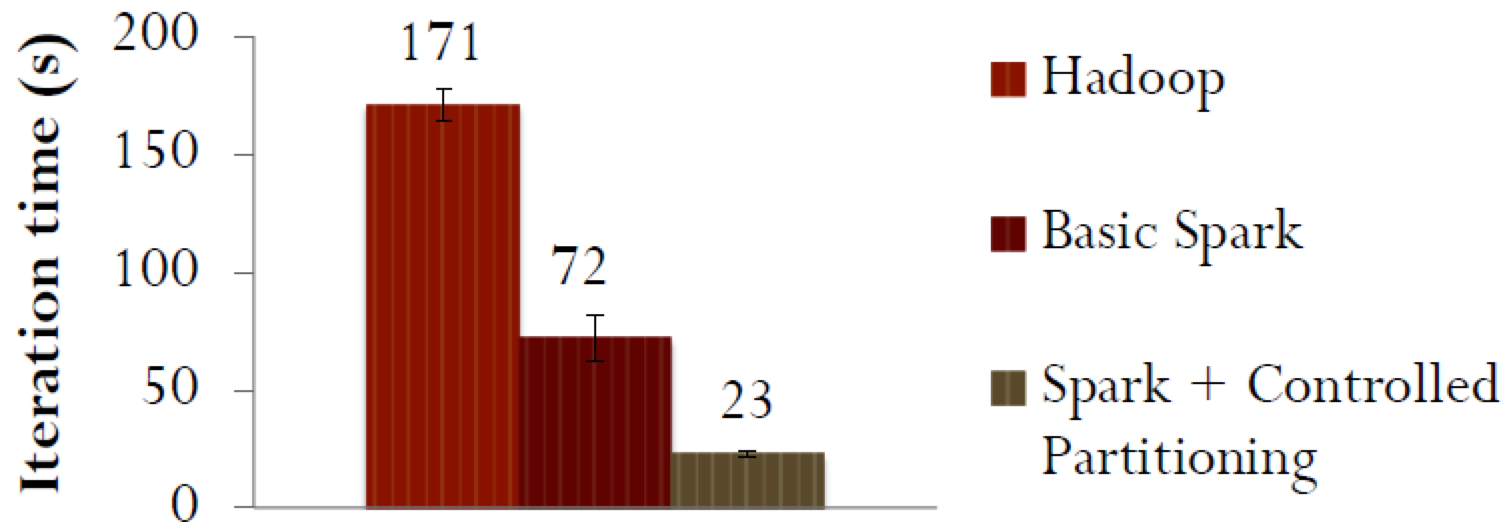


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Inne funkcje Spark

- Redukcja na podstawie skrótu (szybciej niż sortowanie MapReduce). Kontrolowane partycjonowanie danych w celu zmniejszenia komunikacji



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Aplikacje użytkownika

- Analityka w pamięci i wykrywanie anomalii (Conviva)
- Interaktywne zapytania dotyczące strumieni danych (Quantifind)
- Eksploracyjna analiza logów (Foursquare)
- Szacowanie ruchu z danymi GPS (Mobile Millennium)
- Klasyfikacja spamu na Twitterze (Monarch)
- https://spark.apache.org/powered-by.html

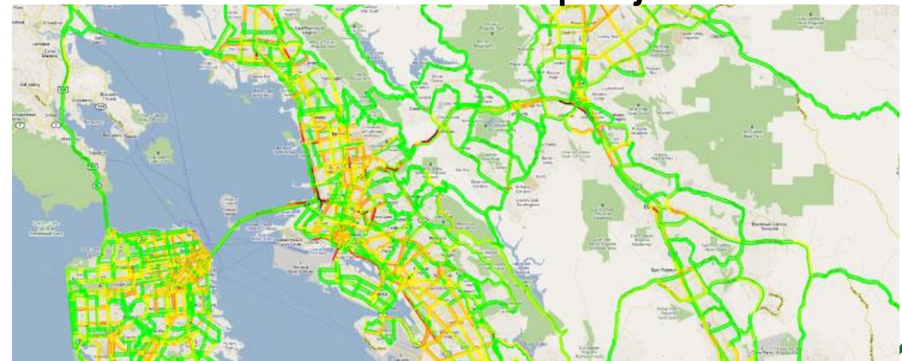
CONVIVA

quantiFind

foursquare

KLOUT

Mobile Millennium project

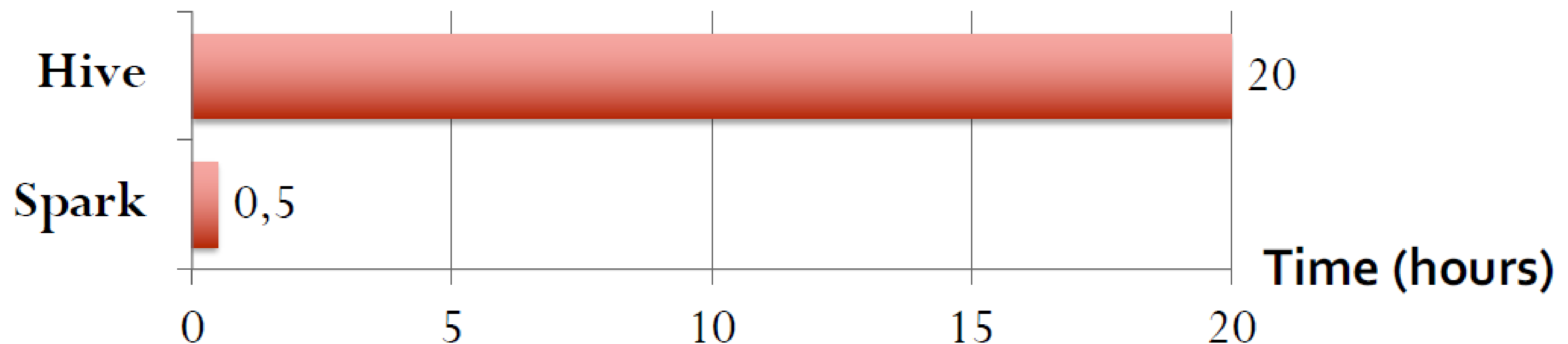


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Conviva GeoReport

- Grupuj agregacje na wielu kluczach z tym samym filtrem40-krotny zysk w stosunku do Hive dzięki unikaniu wielokrotnego odczytu, deserializacji i filtrowania

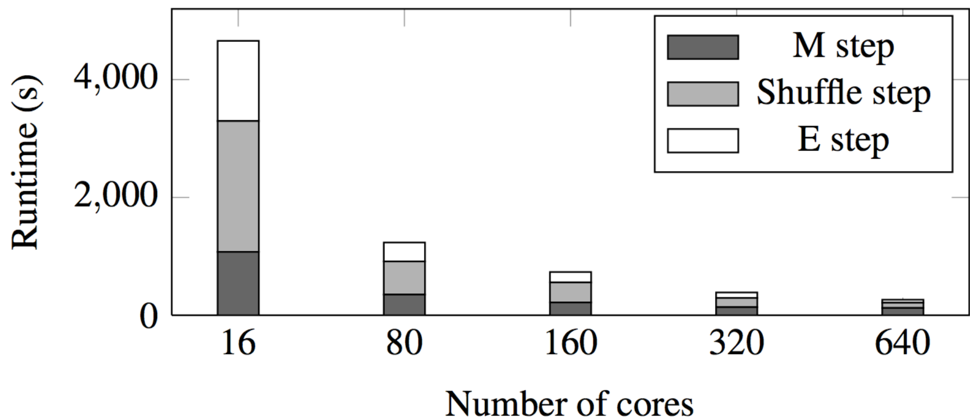


iBigWorld:2020-1-PL01-KA203-082197
Co-funded by the Erasmus+ Programme of the European Union

Mobilny projekt Millennium

- Oszacuj ruch w mieście na podstawie danych GPS pochodzących z crowdsourcingu

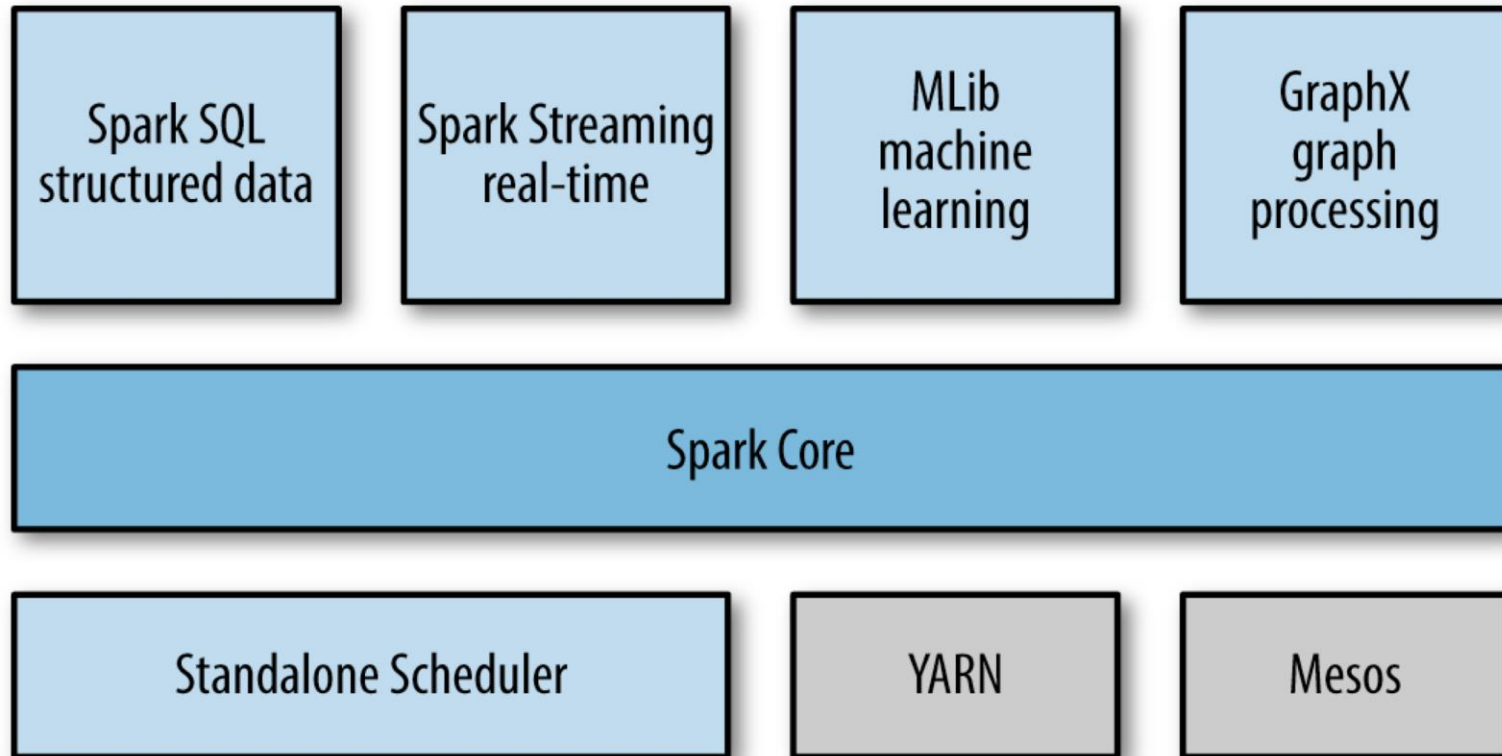
- Skalowanie iteracyjnego algorytmu EM do 160 węzłówAlgorytm maksymalizacji oczekiwań (EM), który szacuje warunki ruchu na drogach miejskich



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Stos oprogramowania Spark



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

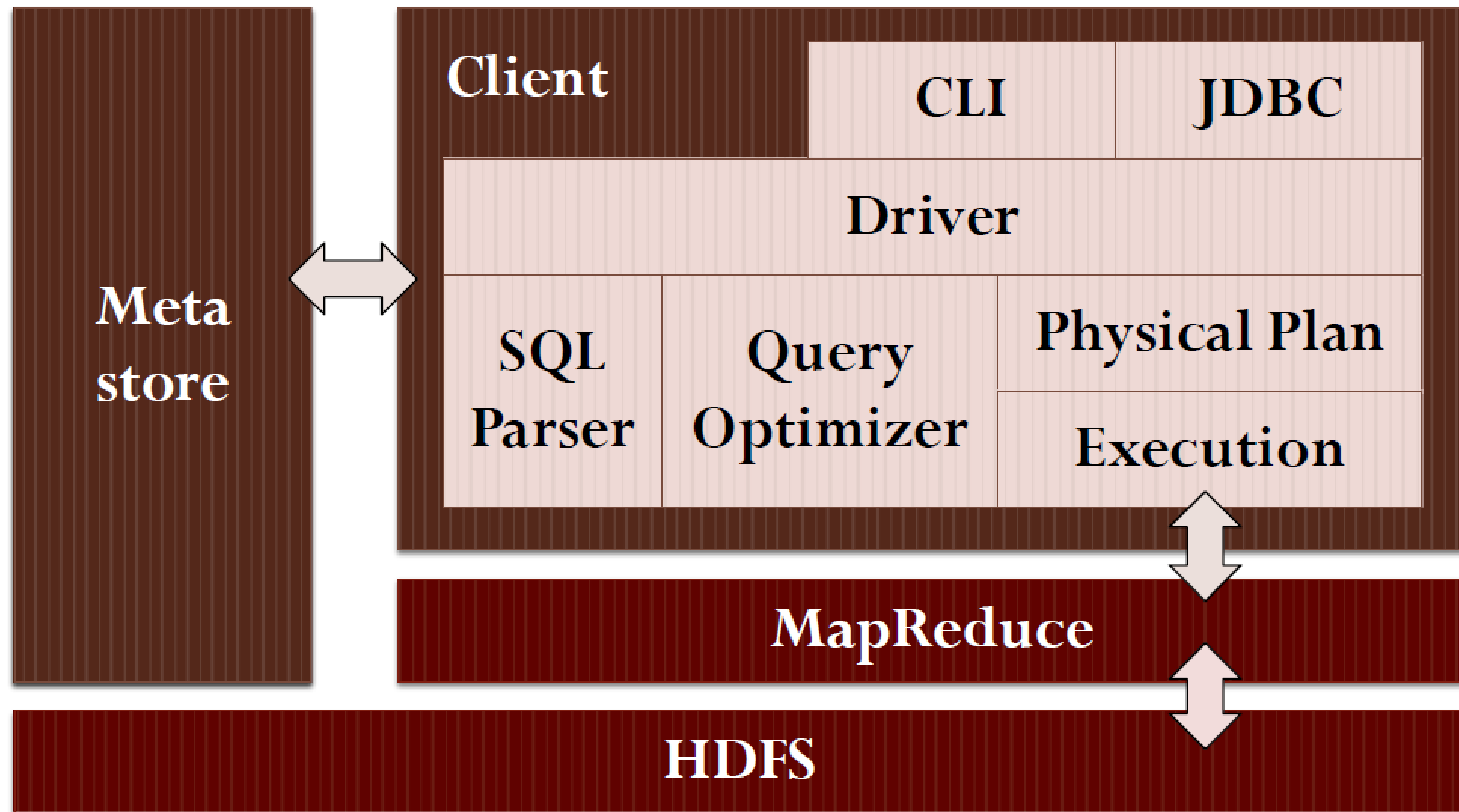
Spark SQL

- Moduł Spark do przetwarzania danych strukturalnychPort Apache Hive do uruchomienia na SparkZgodny z istniejącymi danymi Hive (możesz uruchamiać niezmodyfikowane zapytania Hive na istniejących danych)Przyspieszenie do 40xMotywacja:Hive jest świetny, ale silnik wykonawczy Hadoop sprawia, że nawet najmniejsze zapytania zajmują kilka minutWielu użytkowników danych zna SQLCzy możemy rozszerzyć Hive, aby działał na Spark?Początkowo „Shark”, teraz „Spark SQL”

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

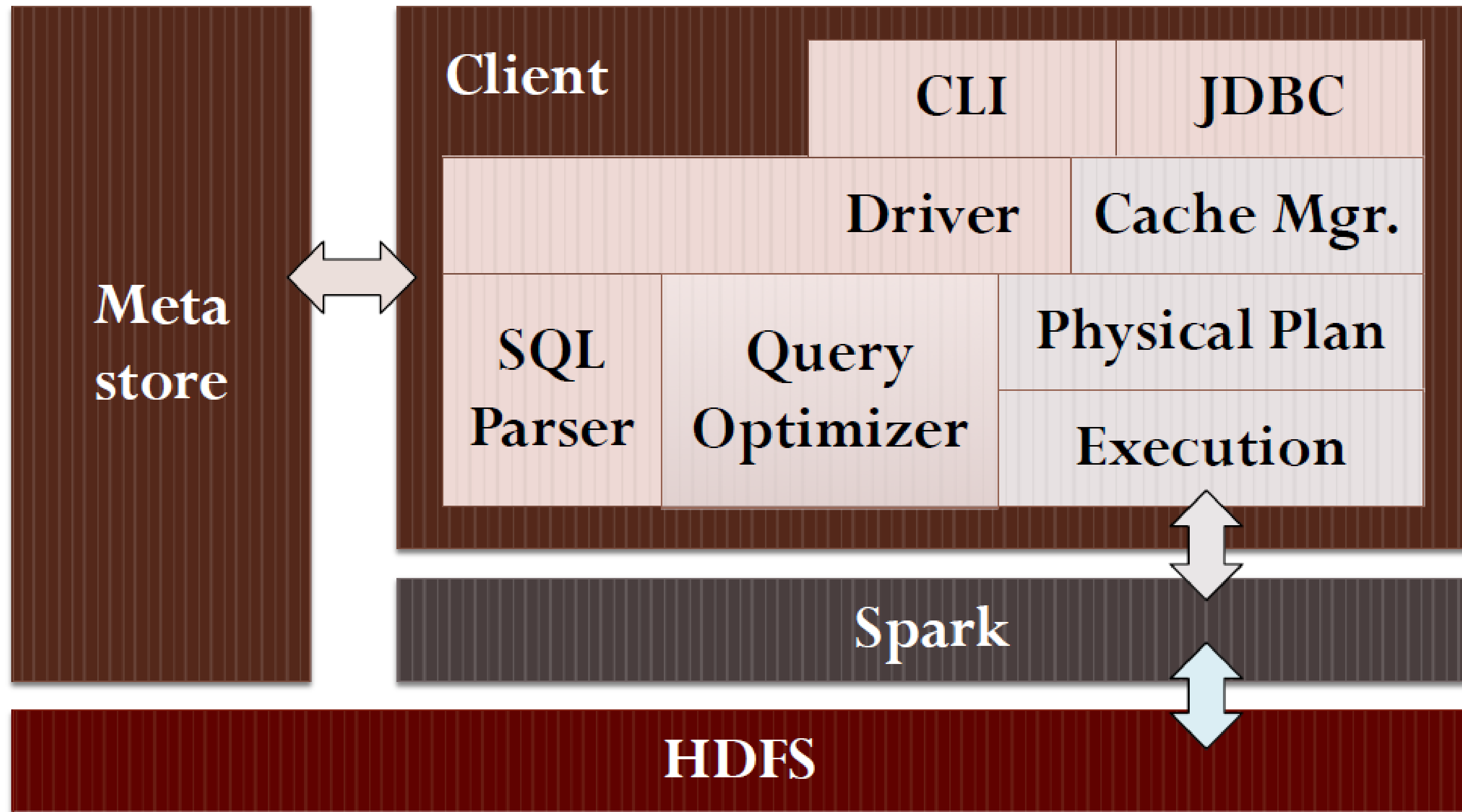
Hive Architektura



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Spark SQL Architektura



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Wydajne przechowywanie w pamięci

- Proste buforowanie rekordów jako obiektów Java jest nieefektywne ze względu na wysoki narzut na obiektZamiast tego Spark SQL wykorzystuje magazyn zorientowany na kolumny przy użyciu tablic typów pierwotnych.Ten format nazywa się Parkiet

Benefit: similarly compact size to serialized data, but >5x faster to access

Row Storage

1	john	4.1
2	mike	3.5
3	sally	6.4

Column Storage

1	2	3
john	mike	sally
4.1	3.5	6.4

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Zbiory danych (DataSet) i ramki danych (DataFrame)

- Zestaw danych to rozproszony zbiór danych, zwykle zorganizowany w rekordy.Nowy interfejs dodany w Spark 1.6 hat zapewnia korzyści z RDD z korzyściami zoptymalizowanego silnika wykonywania Spark SQLDataset API jest dostępny w Scali i Javie. Python nie, ale wiele zalet interfejsu Dataset API jest już dostępnych.

- DataFrame to zestaw danych zorganizowany w nazwane kolumny.Jest koncepcyjnie odpowiednikiem tabeli w relacyjnej bazie danych lub ramki danych w R/Pythonie.Ramki DataFrames mogą być tworzone z: plików danych strukturalnych, tabel w programie Hive, zewnętrznych baz danych lub istniejących RDD.Interfejs API DataFrame jest dostępny w językach Scala, Java, Python i R.W Scali i Javie ramka DataFrame jest reprezentowana przez zestaw danych wierszy.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Punkt wyjścia: SparkSession

- Punktem wejścia do wszystkich funkcji w Spark SQL jest klasa SparkSession.Aby utworzyć podstawową SparkSession, po prostu użyj SparkSession.builder()z pyspark.sql importuj SparkSessioniskra = SparkSession \.konstruktor \.appName("Podstawowy przykład Pythona Spark SQL") \.config("spark.jakaś.config.opcja", "jakaś-wartość") \.getOrCreate()

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Tworzenie DataFrames

W ramach SparkSession aplikacje mogą tworzyć ramki DataFrames z istniejącego RDD, z tabeli Hive lub ze źródeł danych Spark (json, parkiet, csv, text, jdbc, orc, libsvm).Spark SQL może automatycznie wywnioskować schemat zestawu danych JSON i załadować go jako DataFrame.

```
df=spark.read.json("examples/src/main/Resources/people.json")
df.show()
# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

DataFrame -operacje

Ramki danych zapewniają specyficzny dla domeny język do manipulacji danymi strukturalnymi w Scala, Java, Python i R.

```
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)

df.select("name").show()
# +-------+
# |   name|
# +-------+
# |Michael|
# |   Andy|
# +-------+
```

```
df.filter(df['age’]>21).show()
# +---+----+
# |age|name|
# +---+----+
# | 30|Andy|
# +---+----+

df.groupBy("age").count().show()
# +----+-----+
# | age|count|
# +----+-----+
# |  19|    1|
# |  30|    1|
# +----+-----+
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Programowe uruchamianie zapytań SQL

- Funkcja sql w SparkSession umożliwia aplikacjom programowe uruchamianie zapytań SQL i zwraca wynik jako DataFrame.

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")
sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Funkcje ładowania/zapisywania

W najprostszej formie do wszystkich operacji będzie używane domyślne źródło danych (Parquet, chyba że skonfigurowano inaczej przez spark.sql.sources.default).df=spark.read.load("przykłady/źródło/główne/zasoby/użytkownicy.parkiet")df.select("imię","wiek").write.save("out.parkiet") Możesz również ręcznie określić źródło danych (json, parkiet, csv, text , jdbc, orc, libsvm). Ramki DataFrames ładowane z dowolnego typu źródła danych można przekonwertować na inne typy przy użyciu tej składni.df=spark.read.load("examples/src/main/resources/people.json", format="json")df.select("imię","wiek").write.save("out.parkiet", format="parkiet")

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Spark SQL vs Hive

3.6x efficient

CPU Time
(CPU days)

0 100 200 300

15x efficient

CPU reservation time
(CPU days)

0 400 800 1200 1600 2000

2.6x faster

Latency
(hours)

0 2 4 6 8

Spark
Hive

Co-funded by the Erasmus+ Programme of the European Union

Co dalej?

- Model Sparka był motywowany dwoma pojawiającymi się zastosowaniami, aplikacjami interaktywnymi i wieloetapowymi (iteracyjnymi)Innym pojawiającym się przypadkiem użycia, który wymaga szybkiego udostępniania danych, jest przetwarzanie strumienioweŚledź i aktualizuj stan w pamięci w miarę nadejścia zdarzeń.Raportowanie na dużą skalę, analiza kliknięć, filtrowanie spamu itp.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

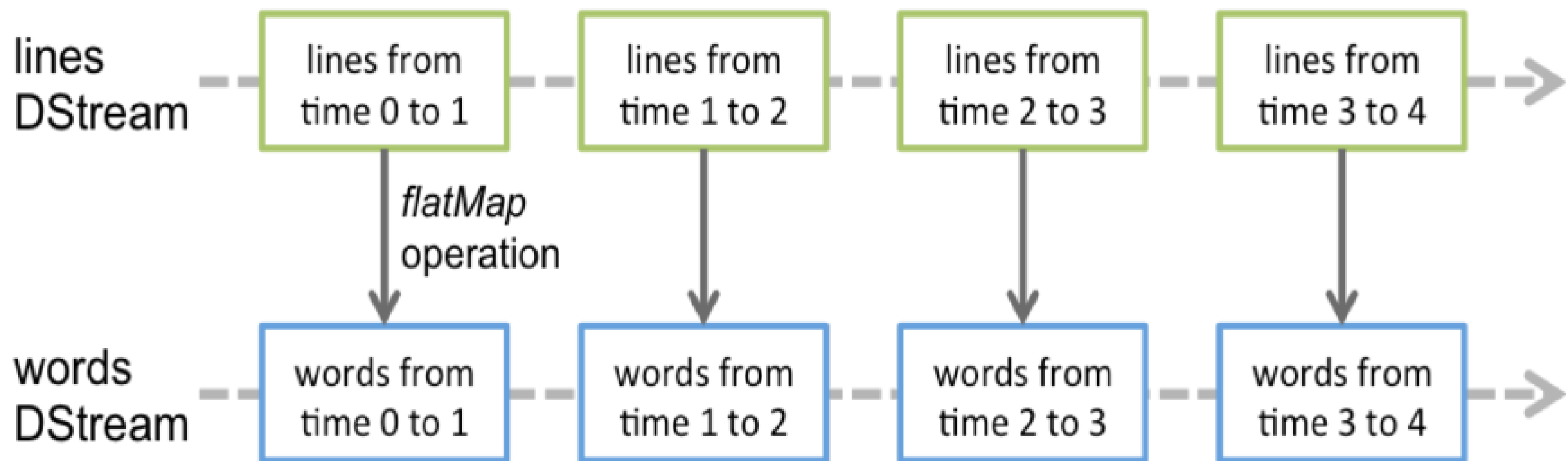
Spark Streaming

- Spark Streaming to rozszerzenie podstawowego interfejsu Spark API, które umożliwia skalowalne, wysokoprzepustowe i odporne na błędy przetwarzanie strumieni danych na żywo.Spark Streaming odbiera strumienie danych wejściowych na żywo i dzieli je na (mikro)partie, które są następnie przetwarzane przez aparat Spark w celu wygenerowania końcowego strumienia wyników w partiach.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Operacje Spark na partiach

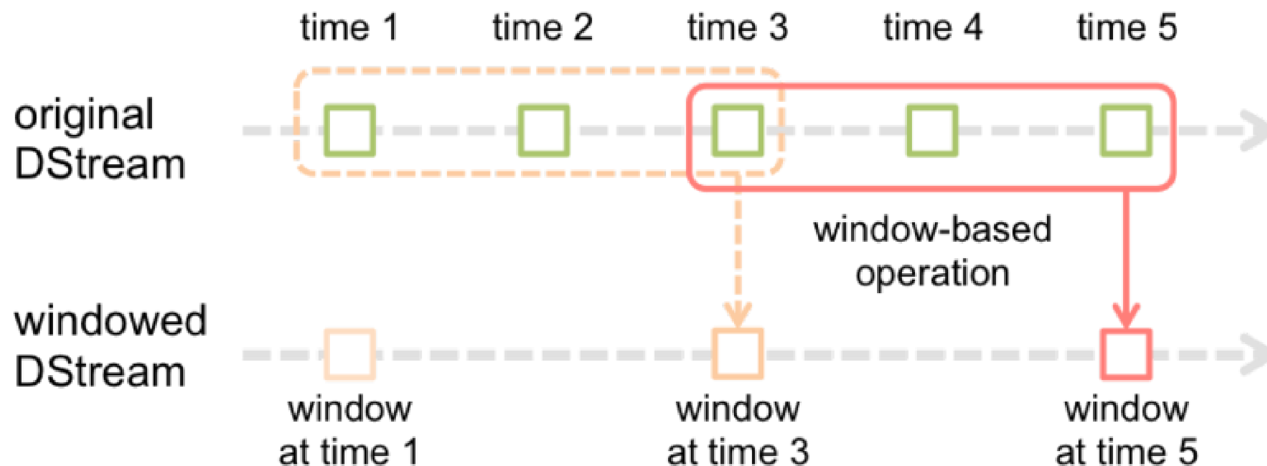


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Operacje na oknach

- Operacje okien są stosowane tylko do tych RDD, które mieszczą się w „przesuwnym” oknie czasowym



- Odporność na awarie jest obsługiwana za pomocą replikacji danych między wieloma węzłami roboczymi Spark.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Przykład Spark Streaming

```
#StreamingContext is the main entry point streaming
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
# two working thread and batch interval of 1 second
sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
# Create a lines DStream connected to a hostname:port
lines = ssc.socketTextStream("localhost", 9999)
# Split each line into words
words = lines.flatMap(lambda line: line.split(" "))
# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)
# Print the first ten elements of each RDD generated in the stream
wordCounts.pprint()
ssc.start()             # Start the computation
ssc.awaitTermination()  # Wait for the computation to terminate
```

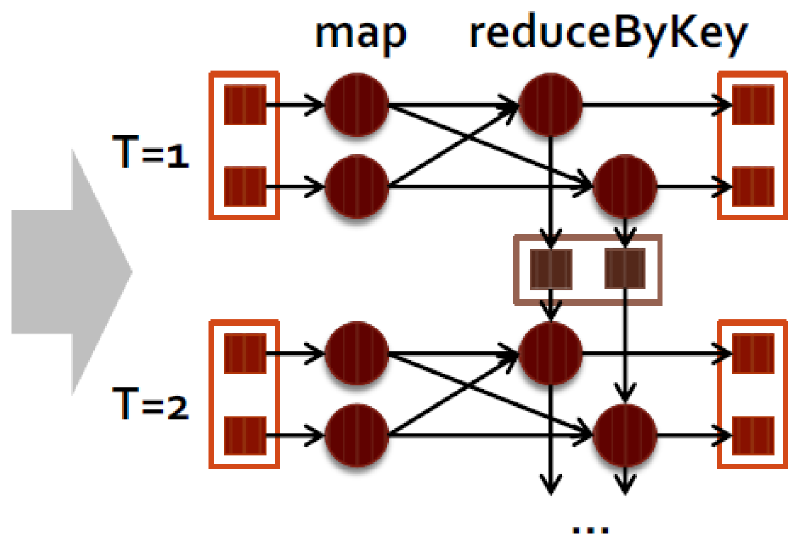

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Model czasu wykonywania

- Działa jako seria małych (~1 s) zadań wsadowych, zachowując stan w pamięci jako odporne na awarie dyski RDD
- Bezproblemowo mieszaj z partią i ad-hoc

```
wordCounts =
  scc.flatMap(lambda line:
              line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a,b: a+b)
```



Can process 42 million records/second (4 GB/s) on 100 nodes at sub-second latency

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Spark MLlib & GraphX

- Spark MLlib: Biblioteka uczenia maszynowego dołączona do interfejsu Spark API.Algorytmy uczenia maszynowego do 100 razy szybsze niż implementacje w MapReduce (Mahout)Wiele algorytmów i narzędzi: regresja liniowa, klastrowanie, częste eksplorowanie zestawów przedmiotów, dekompozycja macierzy,...

- Spark GraphX: API do przetwarzania wykresów i równoległych obliczeń wykresów zintegrowanych z platformą Spark.W tym m.in. PageRank, silnie powiązane komponenty, liczba trójkątów itp.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Referencje

- Apache Spark
  - https://spark.apache.org/
- Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia, Learning Spark, O'Reilly Media, 2015.
  - https://github.com/databricks/learning-spark
- Jules S. Damji, Brooke Wenig, Tathagata Das, and Denny Lee, Learning Spark: Lightning-Fast Data Analytics, 2nd edition, O'Reilly Media, 2020
  - https://github.com/databricks/LearningSparkV2

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union