Przetwarzanie strumieni danych w systemach Big Data

część 2 – podstawy

Krzysztof Jankiewicz

Plan

- Architektury i generacje silników – przypomnienie
- Poziomy abstrakcji – wprowadzenie
- Budowa aplikacji
- Źródła i ich typy
- Transformacje i ich typy
- Przetwarzanie stanowe – wprowadzenie
- Ujścia
- Gwarancje
- Czas – kilka pytań

Architektury silników

- Micro-batch
- Pełnostrumieniowe

- *Job-per-cluster* – pojedyncze zadanie alokuje zasoby (tworzy klaster oparty na JVMs) i dokonuje przetwarzania strumieni w odseparowanym środowisku
- *Session-per-cluster* – klaster alokuje zasoby, a następnie obsługuje zadania wchodzące w ramach pojedynczej sesji użytkownika. Zasoby są dzielone pomiędzy zadania jednej sesji
- *Server-per-cluster* – klaster alokuje zasoby, a następnie obsługuje zadania pochodzące z wielu sesji (np. wielu użytkowników). Zasoby są dzielone pomiędzy zadania wielu sesji

Ewolucja systemów przetwarzania strumieni danych

Początki w ramach projektów badawczych, ale także komercyjnych 1990

Generacje rozproszonych systemów przetwarzania strumieni danych:

- Pierwsza generacja (2011)
  - małe opóźnienia
  - **niskopoziomowe API**
  - **brak obsługi etykiet zdarzeń** – brak powtarzalności, spójności i dokładności wyników
  - gwarancje "at-least once"
  - wykorzystanie w architekturze Lambda
- Druga generacja (2013)
  - **API wysokiego poziomu**
  - lepsza obsługa awarii
  - zwiększona przepustowość
  - zwiększone opóźnienia
  - nadal oparcie się na czasie i kolejności przybywania zdarzeń
- Trzecia generacja (2015)
  - **wykorzystanie etykiet zdarzeń**
  - gwarancje "exactly-once"
  - możliwość konfigurowania przepustowości/opóźnienia
  - możliwość obsługi danych bieżących oraz historycznych
  - wyniki powtarzalne, spójne i dokładne
  - możliwe wykorzystanie architektury Kappa

Poziomy abstrakcji – wprowadzenie

- Niskopoziomowe API – "ręczna" obsługa zdarzeń, "ręczne" utrzymywania stanu przetwarzania
- Podstawowe API – *DataStream API*
  - nieskończone kolekcje napływających obiektów
  - przetwarzanie funkcyjne oparte o przetwarzanie kolekcji
  - duże możliwości
- Wysokopoziomowe API
  - *Table API*
    - model relacyjny
    - źródło to "tabela", w której pojawiają z upływem czasu nowe dane
    - metody odpowiadające klauzulom SQL
  - *SQL*
    - model relacyjny
    - źródło jak wyżej
    - wykorzystanie składni SQL
- *Complex Event Processing* – wykorzystanie wzorców (np. MR)

Użyte nazwy API są umowne i w różnych systemach mogą nazywać się odmiennie

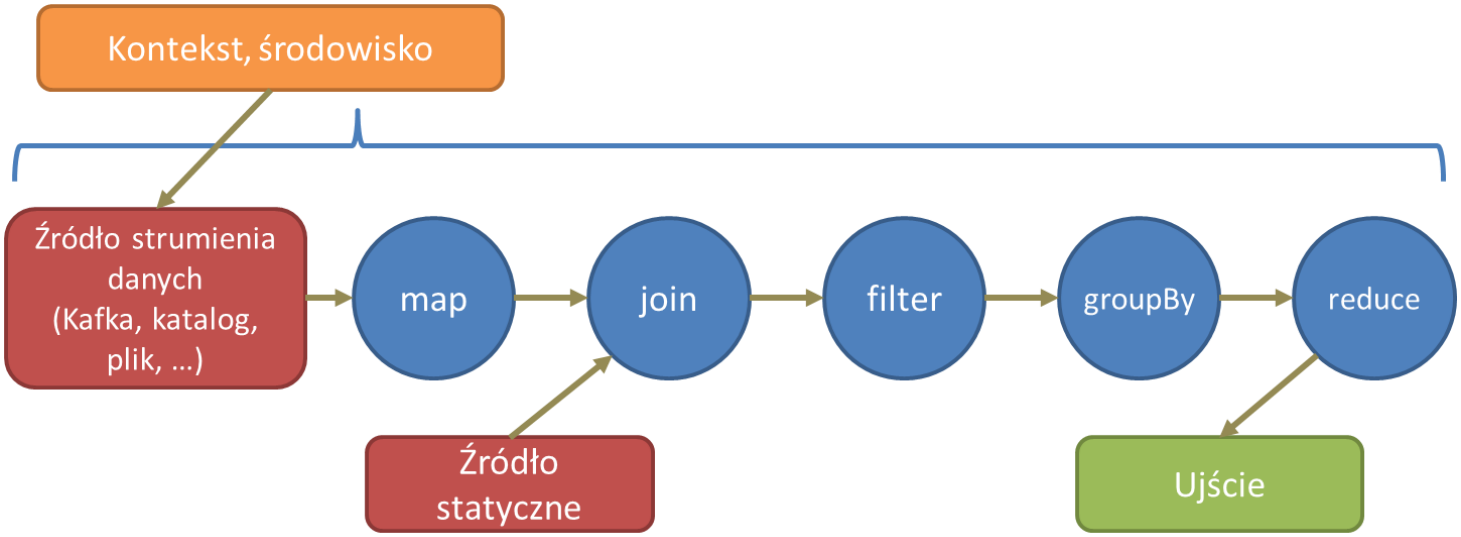
```
val result = orders
        .groupBy($"a")
        .select($"a",
                $"b".count as "cnt")
        .toDataSet[Row]
        .print()
```

```
SELECT a, COUNT(b) as cnt
FROM Orders
GROUP BY a
```

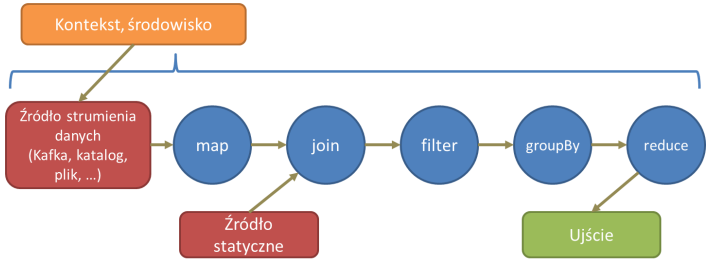
Budowa programu – implementacja

- W wielu systemach, w szczególności korzystających z API na poziomie *DataStream* oraz *Table API,* budowa aplikacji opiera się na pojęciach:
  - kontekstu,
  - źródła,
  - transformacji,
  - ujścia

Kontekst jest obiektem odpowiadającym za połączenie z klastrem silnika przetwarzającego strumienie danych



Przykłady




```
val conf = new

val ssc = new

val lines = ss

val words = li
val pairs = wo
val wordCounts

wordCounts.pri

ssc.start()
ssc.awaitTermi
```

```
val spark = SparkSession.builder
  .appName("StructuredNetworkWordCount")
  .getOrCreate()

import spark.implicits._

val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999).load()

val words = lines.as[String].flatMap(_.split(" "))
val wordCounts = words.groupBy("value").count()

val query = wordCounts.writeStream
  .outputMode("complete").format("console")
  .start()

query.awaitTermination()
```

Źródła i ich typy

- Liczby obsługiwanych źródeł różnią się w zależności od
  - silnika (i jego wersji)
  - typu wykorzystywanego API
- Podstawowe typy źródeł
  - ***testowe***, np.: konsola, sockety TCP
  - ***plikowe***, np.: pliki, katalogi
  - ***systemy kolejkowe***, np.: Apache Kafka, RabbitMQ
  - ***inne systemy przetwarzające strumienie danych***, np.: Apache Storm, Apache Samsa, Amazon Kinesis
  - ***bazy danych***, a w szczególności mechanizmy ***CDC***, np.: Oracle, MongoDB, Neo4j
  - inne…
- W kontekście budowy systemów wiarygodnych źródła dzielimy na
  - **proste** – dostarczające podstawową funkcjonalność, utrudniającą uzyskanie wysokich poziomów gwarancji
  - **wiarygodne** – dostarczające wystarczającej funkcjonalności…

Źródła można także podzielić na:
- wbudowane – dostępne w silniku
- użytkownika – wymagające utworzenia tzw. konektora

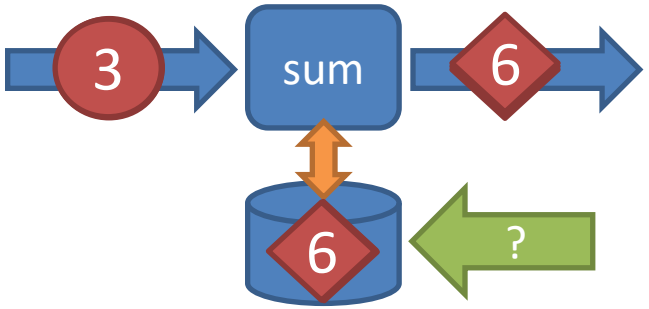
Przykładowo patrz:
`https://nightlies.apache.org/flink/flink-docs-master/docs/connectors/table/overview/`
`https://www.confluent.io/product/connectors/`

Transformacje i ich typy

- Transformacje służące do implementacji przetwarzania strumieni danych z wykorzystaniem *DataStream* i *Table API* można podzielić na wiele kryteriów
  - Utrzymywanie stanu
    - bezstanowe (*stateless*)
      - DataStream API, np.: `filter`, `map`
      - Table API, np.: `where`, `select`
    - stanowe (*stateful*)
      - wbudowane
      - własne
  - Liczba przetwarzanych strumieni
    - pojedynczy
    - wiele, np.: połączenie, split, union
  - Typu
    - transformujące
    - agregujące
    - funkcje okna
    - połączenia
      - strumień – tabela
      - strumień – strumień
    - inne

Przetwarzanie stanowe – wprowadzenie

- Co to jest stan?
- Przetwarzanie bezstanowe
  - Cechy
  - Przykłady
- Przetwarzanie stanowe
  - Cechy
  - Przykłady
  - Interaktywne zapytania
  - Jaki strumień (co) jest wynikiem przetwarzania stanu?
    - complete
    - append
    - update



key	value
A	1
B	2

key	value
A	1
B	2
C	**2**

key	value
A	1
B	**5**

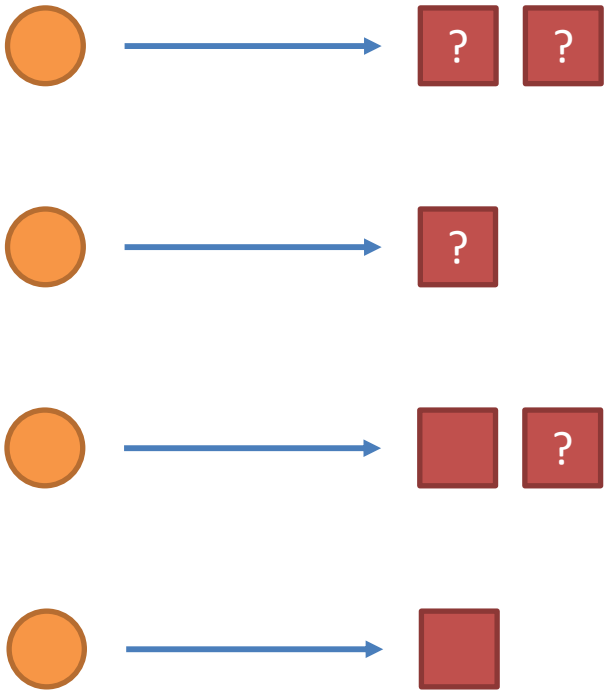
Ujścia

- Liczby obsługiwanych ujść różnią się w zależności od
  - silnika (i jego wersji)
  - typu wykorzystywanego API
- Podstawowe typy ujść
  - ***testowe***, np.: konsola, sockety TCP
  - ***plikowe (lokalne, rozproszone)***, np.: pliki, katalogi
  - ***systemy kolejkowe,*** np.: Apache Kafka, RabbitMQ
  - ***inne systemy przetwarzające strumienie danych,*** np.: Apache Storm, Apache Samsa, Amazon Kinesis
  - ***bazy danych***, w szczególności bazy danych NoSQL, np.: Oracle, MongoDB, Neo4j, Elasticsearch
  - inne…
- Kluczowe są możliwości funkcjonalne ujść.
  W zależności od przypadku wymagana może być
  - możliwość **modyfikacji** wprowadzonych uprzednio danych
  - skalowalność – obsługa "gęstego" i "nieskończonego" (teoretycznie) strumienia danych wyjściowych

Przykładowo patrz:
`https://nightlies.apache.org/flink/flink-docs-master/docs/connectors/table/overview/`
`https://www.confluent.io/product/connectors/`

Gwarancje

- Gwarancje
- Typy gwarancji
  - brak gwarancji (*no guarantee*)
  - co najwyżej raz (*at most once*)
  - przynajmniej raz (*at least once*)
  - dokładnie raz (*exactly once*)



- System jest tak dobry jak jego najgorszy element

Czas – kilka pytań

- Czy czas podczas przetwarzania strumieni danych ma znaczenie?
- O jakim czasie mówimy?
- W jaki sposób mierzymy czas?
- Czy czas biegnie tylko do przodu?
- Czy dane pojawiają się "na czas"?

Pytania?