

Sprawozdanie

Piotr Tutak, 286260

Chmury Obliczeniowe – Projekt, Niestacjonarne

1. Cel projektu

Celem projektu było stworzenie rozproszonego systemu do przesyłania danych składającego się z części przesyłającej dane, przechowującej dane oraz z części przetwarzającej dane.

Całość projektu została stworzona w aplikacjach opartych o systemy rozproszone. System jest podstawą do stworzenia pracy magisterskiej, której tematem jest rozproszony system zarządzania danymi dla pojazdów autonomicznych.

2. Użyte technologie

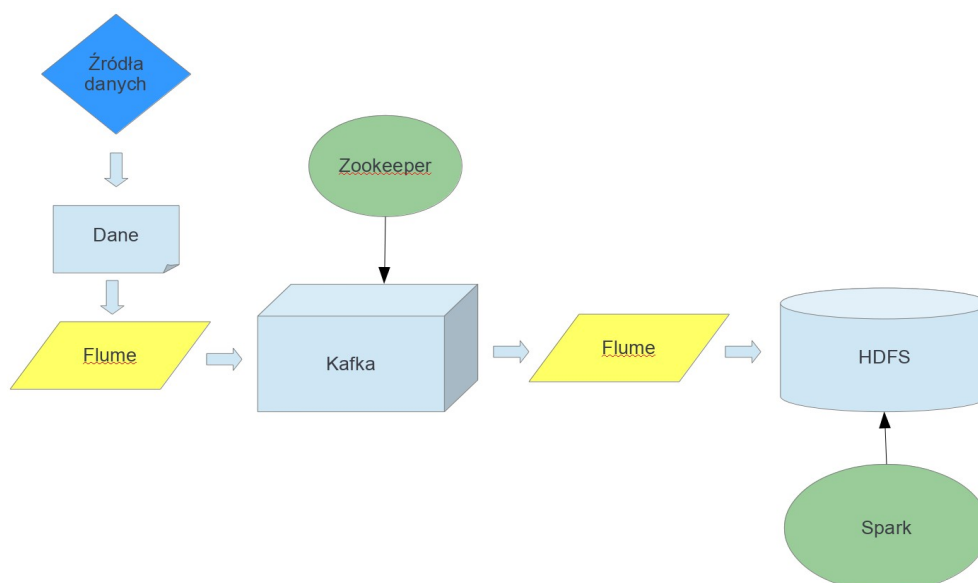
Do realizacji poszczególnych części projektu zostały użyte następujące aplikacje:

- Storage – Hadoop
- Computing – Spark
- Streaming – Kafka + Flume

W celu zapewnienia stabilnego rozwoju systemu i możliwości szybkiego aktualizowania konfiguracji poszczególnych aplikacji, jako system zarządzania konfiguracją został wybrany Ansible.

3. Realizacja

W celu realizacji zadania, dla każdej aplikacji został stworzony i skonfigurowany klaster. Ogólny schemat całego systemu przedstawia się następująco:



Jak widać na powyższym schemacie, dane są transportowane za pomocą agenta Flume'a do klastra Kafki, skąd następnie są przejmowane przez Flume'a i przesyłane do klastra HDFS.

Tam z kolei są przetwarzane przez klaster Spark'a. Wszystkie części systemu zostały uruchomione w wirtualnych maszynach zarządzanych przez narzędzie Ansible.

3.1. Strumieniowanie

a) Flume

Flume to oprogramowanie służące do strumieniowania danych z uprzednio zdefiniowanych źródeł do miejsca docelowego. W każdym agencie Flume'a można zdefiniować 3 podstawowe obiekty, czyli: źródła, kanały i ujścia.

Źródła są, jak sama nazwa wskazuje źródłami danych. Jest kilka typów źródeł danych. W projekcie były używane przede wszystkim 2 źródła danych, czyli *exec-source* i *avro-source*. *Exec-source* to źródło, którym może być dowolny program zdefiniowany przez użytkownika, to źródło zostało użyte do produkcji danych przez program symulujący pojazd autonomiczny, który wysyła dane z kamery. *Avro-source* to źródło które opiera się na uniwersalnym formacie *avro*, który służy do przesyłania dowolnych typów danych wraz z ich schematem. To źródło było używane by połączyć ze sobą różne elementy składowe całego systemu, a więc Fluma z Kafką oraz Fluma z Hadoop'em.

Kanały to zdefiniowane sposoby na tymczasowe przechowywanie danych przez agenta Flume'a, w projekcie były definiowane tylko kanały umieszczone w pamięci.

Ujścia podobnie są podobne do źródeł, różnią się jedynie tym, że nie produkują danych ale są, jak sama nazwa wskazuje ujściami dla danych. W projekcie były wykorzystane ujścia *avro* (*avro-sink*) oraz ujścia *hdfs*. Źródła, kanały i ujścia można łączyć ze sobą w różne konfiguracje, co daje dużą elastyczność w tworzeniu strumieni danych.

b) Kafka

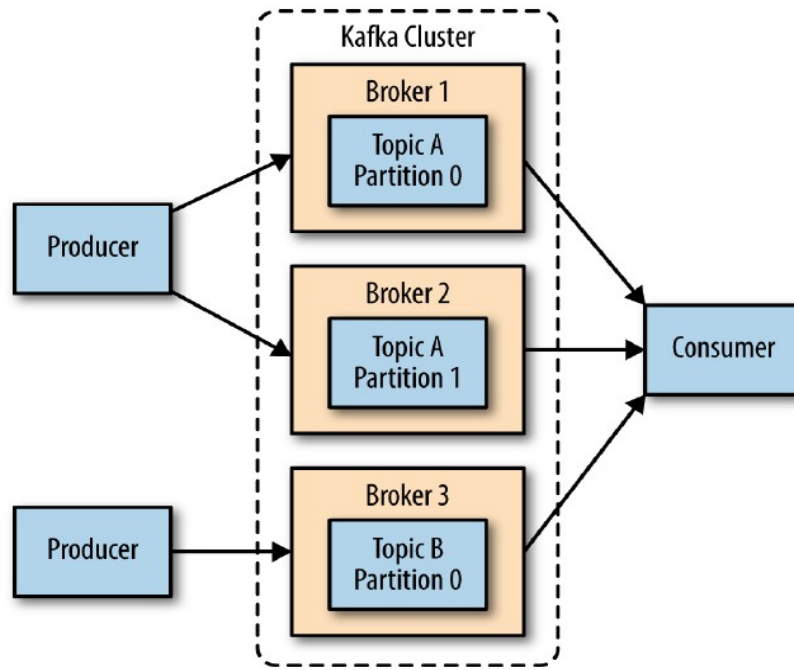
Kafka to klaster w którym dane są przechowywane przez dany czas, dopóki nie zostaną odebrane przez jakiegoś konsumenta.

Kafka składa się z tematów, do których mogą zapisywać się konsumenci, którzy pobierają dane z danego tematu zwane zdarzeniami (*events*), wysyłane tam przez producentów.

Sama Kafka do współdziałania potrzebuje programu Zookeeper, który odpowiada za synchronizację rozproszonych systemów przetwarzania danych.

Każdy temat w klastrze Kafki może dzielić się na partycje, każda partycja umożliwia funkcję automatycznej replikacji. Każda partycja ma przydzielonego brokera, czyli proces zarządzający nią. Dzięki temu Kafka to bardzo odporny na awarie i wysoce dostępny serwis do strumieniowania danych. W zasadzie wszystkie największe rozwiązania, które służą do przesyłania i przetwarzania dużych zbiorów danych opierają się obecnie na Kafce.

Przykładowy poglądowy schemat klastra Kafki jest widoczny na schemacie. Mamy tutaj 2 tematy, z czego temat A posiada 2 partycje, a temat B jedną partycję.



3.2. Storage i Computing

a) HDFS

Storage w projekcie został oparty o rozproszony system przechowywania i przetwarzania danych jakim jest Hadoop. Jednakże Hadoop został w projekcie wykorzystany tylko w części przechowywania danych, czyli HDFS. Za część przetwarzania jest odpowiedzialny Spark.

W systemie HDFS można wyróżnić 2 rodzaje węzłów, tak zwane węzły danych i węzły nazw.

Węzły nazw (*NameNodes*) są odpowiedzialne za zarządzanie węzłami danych i zlecanie im zadań. Na węzłach nazw jest też przechowywana cała hierarchia plików zapisanych w systemie HDFS. Są to też *bramy* wejściowe do całego systemu.

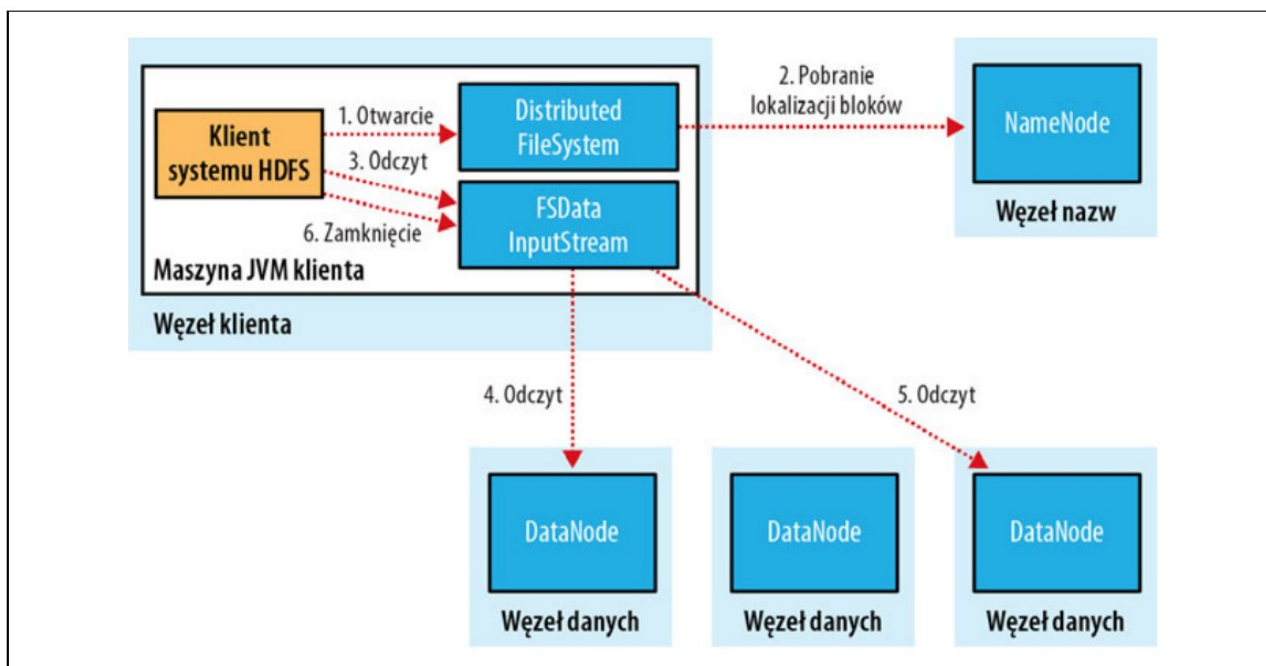
Węzły danych (*DataNodes*) przechowują dane i odpowiadają za replikację danych.

System HDFS umożliwia automatyczną replikację każdej części danych oraz dba o ich odpowiednie rozproszenie na wszystkie węzły systemu tak by zapewnić odpowiednią niezawodność i wysoką dostępność.

Hadoop i HDFS to najczęściej używane obecnie systemy do rozproszonego przechowywania danych, podobnie jak Kafka, jest to system używany w praktycznie każdym poważnym produkcyjnym rozwiązaniu.

W skład systemu Hadoop wchodzi też tak zwany algorytm przetwarzania MapReduce oraz system YARN odpowiedzialne za przetwarzania już raz zachowanych danych w klastrze, jednakże nie były one używane w tym projekcie.

Przykładowy schemat odczytu danych w systemie HDFS

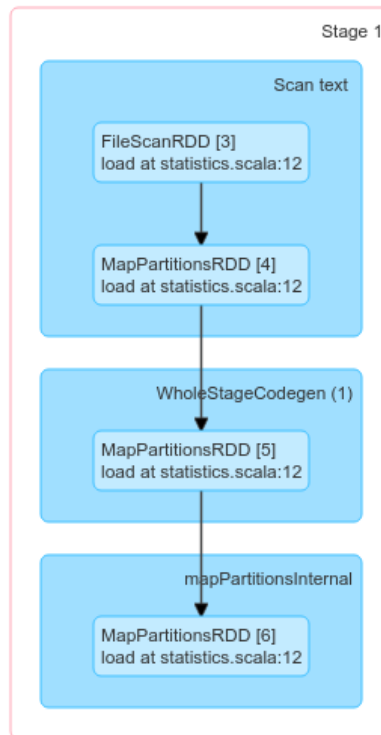


b) Spark

Spark to rozproszony system przetwarzania danych. Działa w integracji z systemem HDFS, można go również włączyć w tryb integracji z systemem YARN, jednakże ta opcja nie była używana w tym projekcie. Spark można też używać w trybie, w którym korzysta on z własnego systemu zarządzania zadaniami, który został w tym projekcie użyty. W systemie Spark można wyróżnić również, podobnie jak w systemie HDFS, dwa rodzaje węzłów. Są to węzły *master* i *worker*. Master jest odpowiedzialny za zlecanie zadań węzłom roboczym, które z kolei przekazują je do odpowiednich wykonawców (*executors*).

By usprawnić i przyspieszyć obliczenia, tworzony jest acykliczny graf skierowany dla każdego zespołu operacji, następnie są one dzielone na takie, które mogą być wykonane lokalnie na każdym węźle, oraz na takie, do których konieczne jest zebranie wszystkich aktualnie przetwarzanych danych (np. grupowanie danych ze względu na jakąś wartość w danej kolumnie). Umożliwia to efektywne wykorzystanie zasobów i zrównoleglenie obliczeń tam gdzie jest to możliwe. Spark wykonuje obliczenia w pamięci w przeciwieństwie do wspomnianego już algorytmu MapReduce, który zapisuje dane tymczasowe na dysku twardym. Dzięki wszystkim tym zaletom Spark jest nawet ponad sto razy szybszy od pierwotnego algorytmu MapReduce.

Przykładowy schemat acyklicznego grafu operacji tworzonego przez Spark'a:



4. Wyniki

Dla tworzonego systemu został przygotowany niewielki przypadek użycia, w którym tworzony jest prosty program generujący dane, symulujący pojazd autonomiczny, następnie te dane są przechwytywane przez agenta Flume, który wysyła je do klastra Kafki do tematu *av_camera_data*. Sam klastr składa się z 4 maszyn, na każdej znajduje się jeden broker.

CMAK **default** Cluster ▾ Brokers Topic ▾ Preferred Replica Election Schedule Leader Election Reassign Partitions Consumers

Clusters / default / Topics

Operations

Generate Partition Assignments

Run Partition Assignments

Add Partitions

Topics

Show 10 ▾ entries

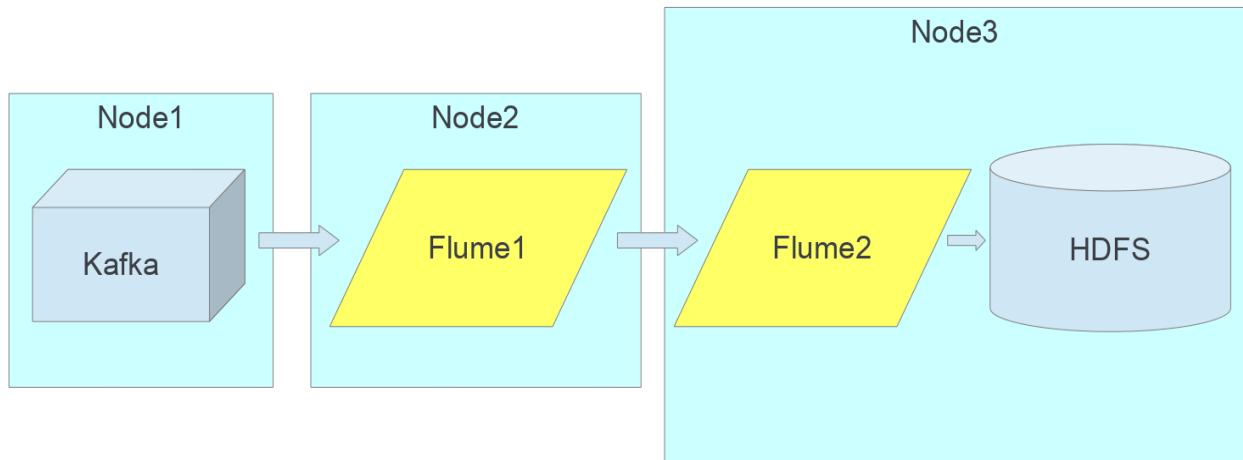
Search:

Topic	↑↓ # Partitions	↑↓ # Brokers	↑↓ Brokers Spread %	↑↓ Brokers Skew %	↑↓ Brokers Leader Skew %	↑↓ # Replicas	↑↓ Under Replicated %	↑↓ Producer Message/Sec	↑↓ Summed Recent Offsets
__consumer_offsets	50	4	100	0	0	3	0	0.00	991
_schemas	1	3	75	0	0	3	0	0.00	327
av_camera_data	4	4	100	0	0	3	0	0.00	85
av_lidar_data	4	4	100	0	0	3	0	0.00	0
av_mobile_data	4	4	100	0	0	3	0	0.00	0
av_radar_data	4	4	100	0	0	3	0	0.00	0
hdfs_topic	4	4	100	0	0	3	0	0.00	383

Showing 1 to 7 of 7 entries

Previous **1** Next

Następnie te dane są znowu przechwytywane przez system agentów Flume, jak na poniższym schemacie:



Jak widać jeden agent Flume'a jest przypisany do tego samego węzła co węzeł nazw w systemie HDFS, dzięki temu może on zapisywać dane w tym systemie po uzyskaniu odpowiednich uprawnień.

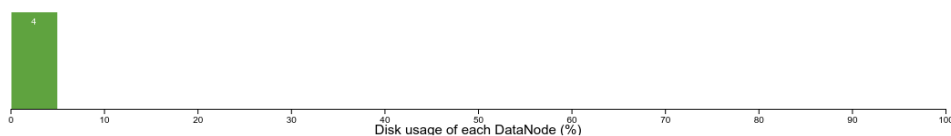
System HDFS składa się z 4 węzłów roboczych i jednego węzła nazw:



Datanode Information

✓ In service ⬇ Down ⏳ Decommissioning ⚡ Decommissioned ⚡ Decommissioned & dead
➡ Entering Maintenance ⚡ In Maintenance ⚡ In Maintenance & dead

Datanode usage histogram



In operation

DataNode State: All Show 25 entries Search:									
Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Version
✓ hadoop5:9866 (192.168.2.205:9866)	http://hadoop5:9864	0s	67m	24.5 MB	4.53 GB	9.78 GB	77	24.5 MB (0.24%)	3.3.0
✓ hadoop3:9866 (192.168.2.203:9866)	http://hadoop3:9864	2s	67m	24.75 MB	4.53 GB	9.78 GB	82	24.75 MB (0.25%)	3.3.0
✓ hadoop2:9866 (192.168.2.202:9866)	http://hadoop2:9864	0s	67m	22.08 MB	4.53 GB	9.78 GB	77	22.08 MB (0.22%)	3.3.0
✓ hadoop4:9866 (192.168.2.204:9866)	http://hadoop4:9864	0s	67m	15.32 MB	4.53 GB	9.78 GB	71	15.32 MB (0.15%)	3.3.0

Showing 1 to 4 of 4 entries

Previous 1 Next

Entering Maintenance

No nodes are entering maintenance.



























Dane trafiają do katalogu `/user/flume/av_mobile_data/camera_data`:

Browse Directory


   

Show 25 entries

Search:

<input type="checkbox"/>	 Permission	 Owner	 Group	 Size	 Last Modified	 Replication	 Block Size	 Name	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	153.35 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569107	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	333.57 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569108	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	356.23 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569109	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	171.89 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569110	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	150.71 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569111	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	160.74 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569112	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	136.6 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569113	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	184.43 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569114	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	169.92 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569115	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	135.58 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569116	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	354.2 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569117	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	314.63 KB	Jan 26 20:49	3	128 MB	"AvCameraData".1611690569118	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	177.82 KB	Jan 26 20:50	3	128 MB	"AvCameraData".1611690569119	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	325.62 KB	Jan 26 20:50	3	128 MB	"AvCameraData".1611690569120	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	323.64 KB	Jan 26 20:50	3	128 MB	"AvCameraData".1611690569121	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	159.37 KB	Jan 26 20:50	3	128 MB	"AvCameraData".1611690569122	
<input type="checkbox"/>	-rw-r--r--	flume	supergroup	179.8 KB	Jan 26 20:50	3	128 MB	"AvCameraData".1611690569123	

Następnie w systemie Spark, który jest zainstalowany na tych samych węzłach co węzły systemu HDFS zlecane jest zadanie przetwarzania tych danych:

 3.0.1 Spark Master at spark://192.168.2.201:7077

URL: spark://192.168.2.201:7077
Alive Workers: 4
Cores in use: 4 Total, 0 Used
Memory in use: 2.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 6 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (4)

Worker Id	Address	State	Cores	Memory	Resources
worker-20210127205632-192.168.2.203-12001	192.168.2.203:12001	ALIVE	1 (0 Used)	512.0 MiB (0.0 B Used)	
worker-20210127205638-192.168.2.204-12001	192.168.2.204:12001	ALIVE	1 (0 Used)	512.0 MiB (0.0 B Used)	
worker-20210127205737-192.168.2.202-12001	192.168.2.202:12001	ALIVE	1 (0 Used)	512.0 MiB (0.0 B Used)	
worker-20210127205744-192.168.2.205-12001	192.168.2.205:12001	ALIVE	1 (0 Used)	512.0 MiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (6)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20210127220323-0005	CarStatistics	4	512.0 MiB		2021/01/27 22:03:23	spark	FINISHED	16 s
app-20210127220243-0004	CarStatistics	4	512.0 MiB		2021/01/27 22:02:43	spark	FINISHED	16 s
app-20210127220217-0003	CarStatistics	4	512.0 MiB		2021/01/27 22:02:17	spark	FINISHED	15 s
app-20210127220104-0002	CarStatistics	4	512.0 MiB		2021/01/27 22:01:04	spark	FINISHED	15 s

System Spark posiada możliwość przeglądania zadań i ich wyników.

spark

3.0.1

History Server

Event log directory: hdfs://192.168.2.201:9000/user/spark/event-log

Last updated: 2021-01-27 22:29:51

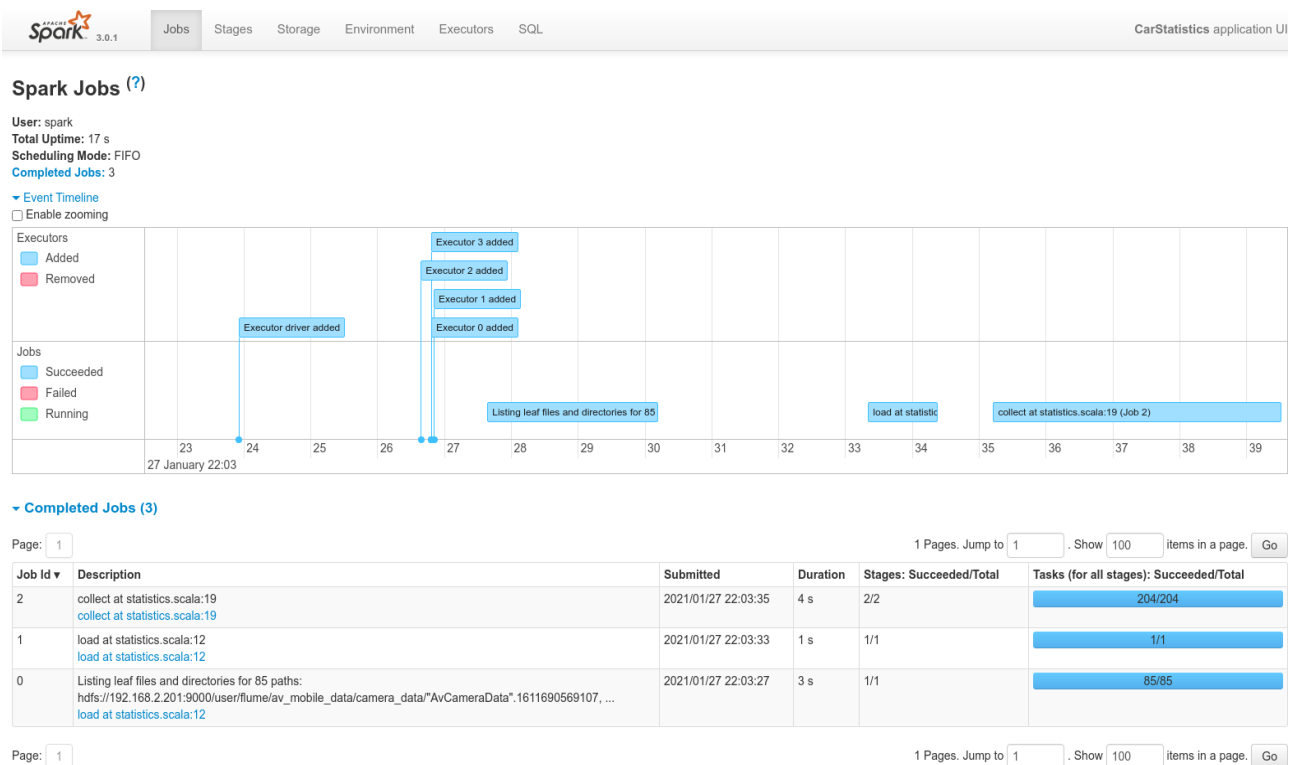
Client local time zone: Europe/Warsaw

Show 20 entries

Search:

Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.0.1	app-20210127220323-0005	CarStatistics	2021-01-27 22:03:22	2021-01-27 22:03:39	17 s	spark	2021-01-27 22:03:39	Download
3.0.1	app-20210127220243-0004	CarStatistics	2021-01-27 22:02:42	2021-01-27 22:02:59	17 s	spark	2021-01-27 22:03:00	Download
3.0.1	app-20210127220217-0003	CarStatistics	2021-01-27 22:02:15	2021-01-27 22:02:32	16 s	spark	2021-01-27 22:02:32	Download
3.0.1	app-20210127220104-0002	CarStatistics	2021-01-27 22:01:02	2021-01-27 22:01:19	16 s	spark	2021-01-27 22:01:20	Download
3.0.1	app-20210127215328-0001	CarStatistics	2021-01-27 21:53:26	2021-01-27 21:54:08	42 s	spark	2021-01-27 21:54:09	Download
3.0.1	app-20210127212608-0000	Spark shell	2021-01-27 21:26:04	2021-01-27 21:44:08	18 min	spark	2021-01-27 21:44:09	Download
3.0.1	app-20210126223808-0002	CarStatistics	2021-01-26 22:38:05	2021-01-26 22:38:31	26 s	spark	2021-01-26 22:38:31	Download
3.0.1	app-20210126223210-0001	Spark shell	2021-01-26 22:32:06	2021-01-26 22:37:35	5.5 min	spark	2021-01-26 22:37:36	Download
3.0.1	app-20210126223018-0000	CarStatistics	2021-01-26 22:30:16	2021-01-26 22:31:01	45 s	spark	2021-01-26 22:31:01	Download
3.0.1	app-20210126220606-0000	CarStatistics	2021-01-26 22:06:05	2021-01-26 22:07:56	1.9 min	spark	2021-01-26 22:07:57	Download
3.0.1	app-20210126215423-0002	CarStatistics	2021-01-26 21:54:20	2021-01-26 21:55:00	40 s	spark	2021-01-26 21:55:00	Download
3.0.1	app-20210126211833-0001	Spark shell	2021-01-26 21:18:30	2021-01-26 21:42:14	24 min	spark	2021-01-26 21:42:15	Download
3.0.1	app-20210126204753-0000	Spark shell	2021-01-26 20:47:51	2021-01-26 21:18:04	30 min	spark	2021-01-26 21:18:05	Download
3.0.1	app-20210126201605-0000	Spark shell	2021-01-26 20:16:02	2021-01-26 20:26:31	10 min	spark	2021-01-26 20:26:32	Download
3.0.1	app-20210126024746-0000	Spark shell	2021-01-26 02:47:43	2021-01-26 02:56:08	8.4 min	spark	2021-01-26 02:56:09	Download
3.0.1	app-20210126000038-0005	Spark shell	2021-01-26 00:00:34	2021-01-26 00:54:40	54 min	spark	2021-01-26 00:54:41	Download

Zlecone zadanie to prosta statystyka, która wyświetla ilość przesłanych danych przez dany samochód reprezentowany przez unikalny identyfikator. Tu przykładowy listing wykonanego zadania przedstawiony przez serwer Spark History:

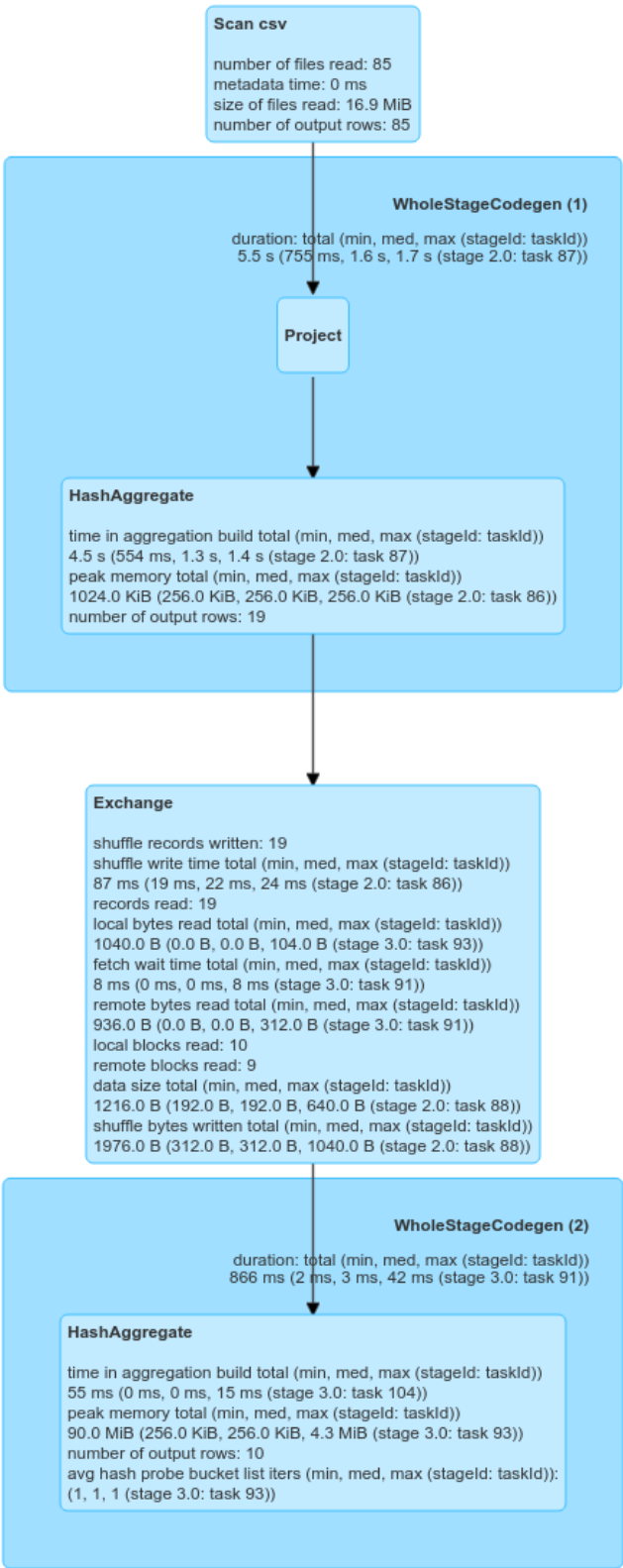


Dla każdego zadania można podglądać jego schemat, wraz ze statystykami

Duration: 5 s

Succeeded Jobs: 2

☒ Show the Stage ID and Task ID that corresponds to the max metric



Kod źródłowy zadania:

```
package statistics

import org.apache.spark.sql.SparkSession

object CarStatistics {
  def main(args: Array[String]) {
    val spark = SparkSession.builder.appName("CarStatistics").getOrCreate()
    val df = (
      spark.read.format("csv")
        .option("header", "false")
        .option("delimiter", ",")
        .load("hdfs://192.168.2.201:9000/user/flume/av_mobile_data/camera_data")
        .toDF("car_id", "data_type", "data_uuid", "date", "data")
    )

    val result = df.groupBy("car_id").count()

    println("Car data statistics results")
    result.collect.foreach(println)

    spark.stop()
  }
}
```

Oraz wyniki zaprezentowane w konsoli

```
2021-01-27 22:03:39,514 INFO scheduler.DAGScheduler: Job 2 finished: collect at statistics.scala:19, took 4.312599 s
2021-01-27 22:03:39,547 INFO codegen.CodeGenerator: Code generated in 23.799332 ms
[360bab8b-81f4-49a3-bd72-d1246d58d52c,1]
[a450d116-d66f-4ab8-a363-6fee2046b2dd,1]
[2d1e9b98-f075-48d0-b9aa-ff91a7cdae1f,1]
[50e30c32-6a6a-4459-a7e2-d970e12e0952,1]
[0de991e3-612b-42e9-9521-b1925d496b72,26]
[396406c8-d25a-4082-8265-ac3736880acd,1]
[f264ca48-40f4-4604-8f47-dc76fdda4667,34]
[1205563f-8950-4146-975c-106628809d84,1]
[f14db780-af0c-41c2-95a0-14ee4e78728e,1]
[2b94cdf2-4542-474c-abf6-aaec8d0844d0,18]
2021-01-27 22:03:39,573 INFO server.AbstractConnector: Stopped Spark@6f3f0fae[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
2021-01-27 22:03:39,586 INFO ui.SparkUI: Stopped Spark web UI at http://192.168.2.201:4040
2021-01-27 22:03:39,588 INFO cluster.StandaloneSchedulerBackend: Shutting down all executors
2021-01-27 22:03:39,589 INFO cluster.CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
2021-01-27 22:03:39,974 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2021-01-27 22:03:39,982 INFO memory.MemoryStore: MemoryStore cleared
2021-01-27 22:03:39,982 INFO storage.BlockManager: BlockManager stopped
2021-01-27 22:03:39,986 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2021-01-27 22:03:39,988 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped
2021-01-27 22:03:39,992 INFO spark.SparkContext: Successfully stopped SparkContext
2021-01-27 22:03:39,995 INFO util.ShutdownHookManager: Shutdown hook called
2021-01-27 22:03:39,995 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-54ed2ca6-d322-4740-8470-e752b2887286
2021-01-27 22:03:39,996 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-1e057356-8819-4c7c-a36d-c9d60c52021b
[ptutak@hadoop1 ~]$ cat output.txt
Car data statistics results
[360bab8b-81f4-49a3-bd72-d1246d58d52c,1]
[a450d116-d66f-4ab8-a363-6fee2046b2dd,1]
[2d1e9b98-f075-48d0-b9aa-ff91a7cdae1f,1]
[50e30c32-6a6a-4459-a7e2-d970e12e0952,1]
[0de991e3-612b-42e9-9521-b1925d496b72,26]
[396406c8-d25a-4082-8265-ac3736880acd,1]
[f264ca48-40f4-4604-8f47-dc76fdda4667,34]
[1205563f-8950-4146-975c-106628809d84,1]
[f14db780-af0c-41c2-95a0-14ee4e78728e,1]
[2b94cdf2-4542-474c-abf6-aaec8d0844d0,18]
[ptutak@hadoop1 ~]$
```

5. Zastosowanie i podsumowanie

Przedstawiony system przetwarzania ma uniwersalne zastosowanie we wszelkich możliwych dużych systemach przetwarzania danych i jako taki jest stosowany przy zastosowaniu trochę innych narzędzi. Każdy problem gromadzenia a później przetwarzania dużych zbiorów danych będzie miał coś wspólnego z przedstawioną koncepcją na rozwiązanie tego problemu.

Stworzenie i skuteczne zarządzanie tak rozbudowanym systemem nie jest praktycznie możliwe bez oprogramowania służącego do zarządzania konfiguracją.

System będzie dalej rozbudowywany by ostatecznie stworzyć kompleksowy system do zarządzania danymi dla pojazdów autonomicznych jako temat pracy magisterskiej