Dokumentacja Hadoop oraz Spark w Cyfronet

Uruchomienie klastra do obliczeń bigdata na Zeus

1. Przykład zadania Spark wykorzystującego 36 rdzeni na 3 węzłach w trybie *klastra Spark*

(Run on a Spark Standalone cluster in client deploy mode)

```
srun -N3 --ntasks-per-node=12 --pty /bin/bash
```

Na początek należy uruchomić klaster *Spark* (master+slaves), a po zakończeniu obliczeń zniszczyć:

Załadować moduł Spark:
module load plgrid/apps/spark

Uruchomić klaster Spark:

start_spark_cluster

Wykonać obliczenia:

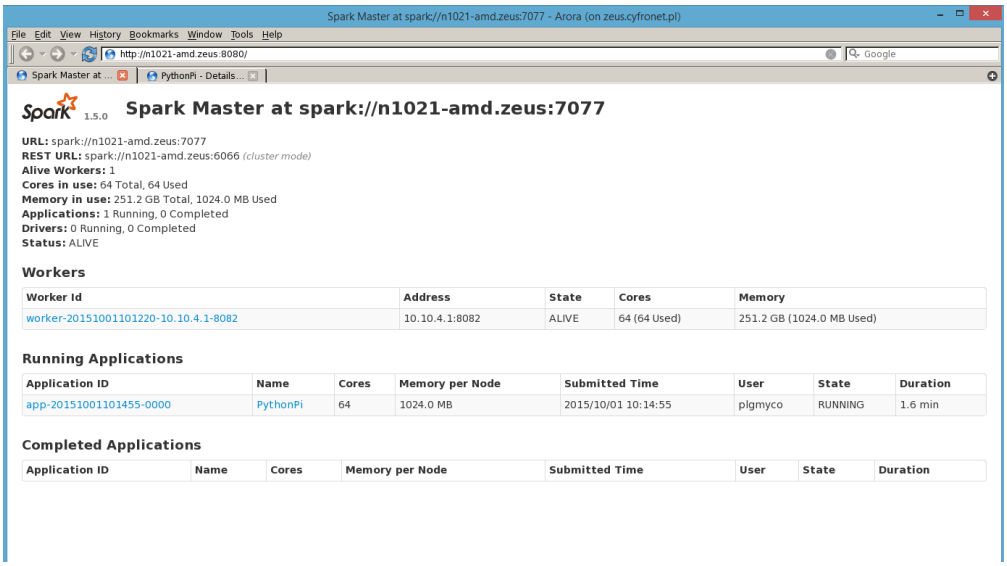
```
$SPARK_HOME /bin/spark-submit $SPARK_HOME /examples/src/main/python/wordcount .py /etc/passwd
$SPARK_HOME /bin/spark-submit --class org.apache.spark.examples.JavaWordCount  $SPARK_HOME /lib/spark-examples
*.jar /etc/passwd
```

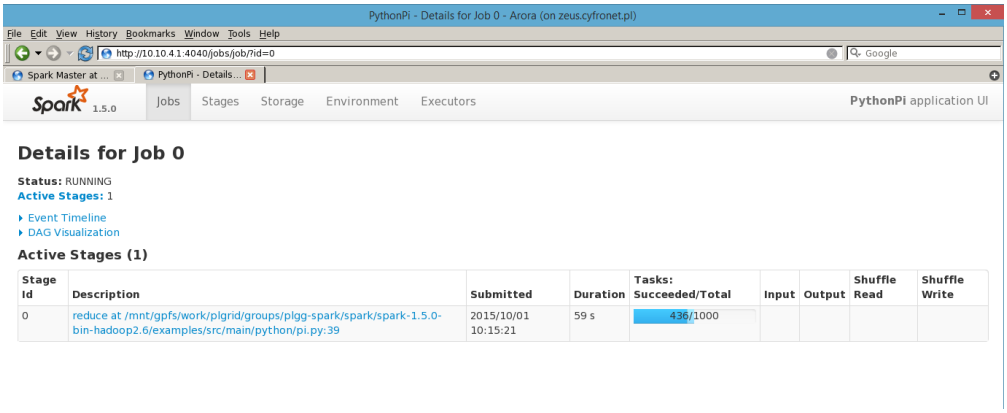
Zatrzymać klaster Spark:

stop_spark_cluster

Podgląd obliczeń prowadzonych na stworzonym klastrze można obserwować w przeglądarce arora:

- module load plgrid/apps/arora && arora
- na porcie 8080 headnoda - Informacje o klastrze Spark.
- na porcie 4040 headnoda - Informacje o aktualnie przetwarzanych zadaniach i środowisku.





2. Przykład zadania Yarn wykorzystującego 36 rdzeni na 3 węzłach w trybie *klastra Yarn (Hadoop)*

(Prometheus)

srun -N3 --ntasks-per-node=12 --pty /bin/bash

start_yarn_cluster

- Uruchomić wybrany skrypt/aplikacje

```
yarn jar $YARN_HOME /share/hadoop/mapreduce/hadoop-mapreduce-examples *.jar  pi 10 100
```

- Podgląd obliczeń prowadzonych na stworzonym klastrze można obserwować w przeglądarce arora na porcie 8088 headnoda.
- Lista dostępnych węzłów obliczeniowych **yarn node -list**
  - Node-Http-Address można użyć w przeglądarce
- Lista uruchomionych aplikacji **yarn application -list**
- Uruchomienie przeglądarki na węźle:
  - **module load plgrid/apps/arora && arora**
  - Wkleić do przeglądarki uzyskany we wcześniejszym punkcie Node-Http-Address

Po zakończeniu obliczeń zniszczyć węzeł yarn:

stop_yarn_cluster

3. Przykład zadania Spark wykorzystującego 36 rdzeni na 3 węzłach w trybie *klastra Yarn (Hadoop)*

(Prometheus)

srun -N3 --ntasks-per-node=12 --pty /bin/bash

Na początek należy uruchomić klaster Yarn (master+slaves), a po zakończeniu obliczeń zniszczyć:

```
module load plgrid/apps/spark

start_spark_cluster

$SPARK_HOME/bin/spark-submit --master yarn-client --class org.apache.spark.examples.SparkPi $SPARK_HOME/lib/spark-examples*.jar 1000

Zatrzymywanie:
stop_spark_cluster
```

- **Wynik obliczeń aplikacji Spark na klastrze Yarn** dostępny jest w $YARN_APP_LOGS_DIR/userlogs/
- Do wyboru mamy tryb obliczeń yarn-cluster lub yarn-client. Więcej informacji na http://spark.apache.org/docs/latest/running-on-yarn.html
- Podgląd obliczeń prowadzonych na stworzonym klastrze można obserwować w przeglądarce arora na porcie 8088 headnoda.
- module load plgrid/apps/arora && arora

Dostrajanie konfiguracji

Po załadowaniu modułu spark na WN, zmiany w konfiguracji klastrów Spark oraz Yarn możemy wprowadzać w katalogach określonych przez **$SPARK_CONF_DIR** oraz **$YARN_CONF_DIR**

Dostrajanie klastra Spark

- ### Pamięć dla aplikacji

  Częsty problem związany ze zbyt małą ilością pamięci (`java.lang.OutOfMemoryError: Java heap space`) dla bardziej wymagających aplikacji można rozwiązać przez zwiększenie zasobów pamięciowych w pliku $SPARK_CONF_DIR/spark-defaults.conf
  SPARK_WORKER_MEMORY =8G
  SPARK_EXECUTOR_MEMORY=8G
  SPARK_DRIVER_MEMORY=8G
  Inny sposób to ustawienie dodatkowych opcji w samej aplikacji lub parametrów do komendy spark -submit np.:
  --driver- memory 8g
  --executor- memory 8g

- ### Logowanie

  W pliku $SPARK_CONF_DIR/log4j.properties możemy ustawić poziomy logowania w $SPARK_LOG_DIR dla poszczególnych serwisów np.

  ```
  log4j.logger.org.apache.hadoop = DEBUG
  ```

Dostrajanie klastra Yarn

- ### Zasoby dla węzła obliczeniowego

W pliku $YARN_CONF_DIR/yarn-site.xml możemy ustawić ilość pamięci oraz ilość dostępnych rdzeni dla pojedynczego węzła w klastrze.

yarn.nodemanager.resource.cpu-vcores (Number of CPU cores that can be allocated for containers. Default is 8)

yarn.nodemanager.resource.memory-mb (Amount of physical memory, in MB, that can be allocated for containers. Default is 8192)

- ### Service Level Authorization (SLA), Access Control Lists (ACL)

W pliku $YARN_CONF_DIR/hadoop-policy.xml możemy zmieniać ustawienia dostępu do usług klastra dla użytkowników.

https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ServiceLevelAuth.html

Po zmianie konfiguracji (w przypadku gdy już mamy uruchomiony klaster yarn) należy przeładować ustawienia ACL:
yarn rmadmin -refreshServiceAcl

Podstawy tworzenia aplikacji dla Spark

Podstawowe działania na RDD

RDD - distributed dataset - obiekt zawierający zbiór danych gotowy do przetwarzania równoległego przez Akcje i Transformacje bibliotek Spark.

Akcje i Transformacje jako argument przyjmują wyrażenie lambda lub nazwę funkcji.

Uruchomienie przykładu w wersji interaktywnej na węźle:

```
$SPARK_HOME/bin/pyspark
```

Przykład 1:

```
rdd = sc.parallelize([1, 2, 3, 4])
rdd.map(lambda x: 2 * x).cache().map(lambda x: 2 * x).collect()
>>> [4, 8, 12, 16]
```

Opis:

parallelize() - tworzy obiekt RDD ze zbioru

map() - transformacja RDD, operuje funkcją lambda na wszystkich elementach RDD.

cache() - zachowuje w pamięci wybrany obiekt RDD. Metodę cache() wykorzystuje się, gdy chcemy użyć wybranego RDD więcej niż raz np. w algorytmach iteratywnych jak PageRank. Jeśli w takim wypadku nie zrobimy cache() to RDD będzie wyliczone za każdym razem od nowa. Możemy również użyć podobnej metody persist() .

collect() - akcja RDD zwracająca pythonową listę elementów RDD

Przykład 2:

Uruchomienie przykładu w wersji interaktywnej:

```
$SPARK_HOME/bin/pyspark
from operator import add
rdd.map(lambda x: 2 * x).reduce(add)
>>> 20
rdd.flatMap(lambda x: [x, x]).reduce(add)
>>> 20
```

Opis:

reduce() - akcja na RDD agregująca elementy obiektu RDD za pomocą funkcji lambda która ma przyjmować dwa argumenty i zwracać jeden wynik. re duce() zwróci ostatecznie jedną wartość. Działanie reduce() dobrze demeonstruje przykład wyszukiwania największej liczby w całej kolekcji: reduce (lambda a, b: a if (a > b) else b)

flatMap() dokonuje Transformacji na RDD. Stosuje funkcję lambda do wszystkich elementów RDD i zwraca ‘płaski’ wynik typu lista RDDs (bez zagnieżdżonych struktur), czyli elementów wyjściowych może być więcej niż wejściowych, co odróżnia działanie od map() .

Zliczanie słów w pliku tekstowym

Przykład 3:

Uruchomienie przykładu w wersji interaktywnej:

```
$SPARK_HOME/bin/pyspark
```

```
from pyspark import SparkContext
from operator import add
sc = SparkContext(master="local[*]", appName="PythonWordCount")

lines   = sc.textFile("/etc/passwd")
words   = lines.flatMap(lambda x: x.split(' '))
marked_words = words.map(lambda x: (x, 1))
counted_words = marked_words.reduceByKey(add)
print counted_words.collect()

sc.stop()
```

Opis:

Stworzenie głównego obiektu aplikacji i nadanie mu kontekstu

np. appName, master (tryb obliczeń: endpoint klastra lub local)

textFile() wczytuje plik tekstowy, który musi być dostępny na wszystkich węzłach.

Zwraca “RDD of Strings”, czyli obiekt gotowy do zrównoleglonych operacji Spark.

flatMap() dokonuje Transformacji na RDD. Stosuje funkcję lambda do wszystkich elementów RDD i zwraca ‘płaski’ wynik typu lista RDDs (bez zagnieżdżonych struktur).

map() dokonuje Transformacji na RDD. Stosuje funkcję lambda do wszystkich elementów RDD i zwraca ‘listę RDD’ (tyle samo elementów listy co na wejściu ale przetransfromowanych).

reduceByKey() Transformacja RDD, która agreguje wartości dla każdego powtarzającego się

w krotkach (K,V) klucza. Na wartościach zostanie wykonana operacja określona funkcją lambda.

metoda **collect()** tworzy listę łańcuchów z obiektu RDD, czyli jest to Akcja.

blocked URL

Sortowanie słów

Aby wczytać wiele plików do jednego obiektu RDD możemy użyć maski: textFile("/my/directory/*.txt")

Przykład 4:

Uruchomienie przykładu w wersji interaktywnej:

```
$SPARK_HOME/bin/pyspark
```

```
lines   = sc.textFile("/etc/passwd")
words   = lines.flatMap(lambda x: x.split(' '))
sortedCount = words.map(lambda x: (x, 1)).sortByKey(lambda x: x)
output  = sortedCount.collect()
```