standalone에 붙인 pyspark spark job 실행해보기

About

pyspark application을 standalone에 띄우고 여기서 spark job을 실행해본다. 그 결과를 standalone UI를 통해서 확인한다.

Prerequisite

standalone, spark가 깔려 있어야 한다.

Let's get started !

먼저, application으로 pyspark shell을 띄운다. 참고로 spark shell은 scala 기반, pyspark shell은 파이썬 기반이다.

읽고 싶은 파일을 정하자. 나는 `/kikang/spark3/README.md` 로 정했다. pyspark shell에 들어가있는 상태에서 경로에 맞게 README.md 파일을 불러와보자. (현재 나는 /kikang/spark3에 있다.)

```
$ rdd = sc.textFile('README.md')
```

스파크는 기본적으로 lazy evaluation을 한다. 즉, 위 명령어를 실행한다고 해서 바로 README.md을 읽어오지 않는다. 아래의 action을 발생시키는 코드를 실행해야지 스파크는 그제서야 코드를 실행하기 시작한다.

```
$ rdd.count()
```

만약 위 코드를 실행했는데, 파일을 hdfs://에서 찾을 수 없다는 에러가 뜬다면 두 가지 해결책이 있다.

첫 번째는 hdfs에 README.md을 올리는 것이다. hdfs라는, 마스터와 워커 모두가 공유할 수 있는 저장 시스템에 README.md을 올림으로써 한번에 접근 가능하도록 한다.

두 번째는 파일을 읽는 경로를 달리하는 것이다. 즉, hdfs://가 아니라 file://부터 시작하여 절대 경로를 지정해준다.

```
$ rdd = sc.textFile('file:///kikang/spark3/README.md')
```

즉, 로컬에서 파일을 읽어오겠다는 뜻이다. 근데 만약에 이와 같이 경로를 지정하면 READMD.md라는 파일이 각 워커의 로컬 경로에 모두 존재해야 한다. 만약 파일 크기가 매우 큰 빅데이터라면 디스크 낭비가 발생할 수 밖에 없다. 따라서 hdfs에 파일을 올리는 것을 추천한다.

`rdd.count()` 라는 job을 스파크가 어떻게 수행했는지 UI로 확인해보자.

Spark Master at spark://hadoop-master-01:7177

URL: spark://hadoop-master-01:7177
Alive Workers: 3
Cores in use: 6 Total, 6 Used
Memory in use: 21.0 GiB Total, 3.0 GiB Used
Resources in use:
Applications: 1 Running, 3 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20201130105715-10.128.0.7-36512	10.128.0.7:36512	ALIVE	2 (2 Used)	7.0 GiB (1024.0 MiB Used)	
worker-20201130105715-10.128.0.8-35666	10.128.0.8:35666	ALIVE	2 (2 Used)	7.0 GiB (1024.0 MiB Used)	
worker-20201130105715-10.128.0.9-39881	10.128.0.9:39881	ALIVE	2 (2 Used)	7.0 GiB (1024.0 MiB Used)	

Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20201130122112-0003 (kill)	PySparkShell	6	1024.0 MiB		2020/11/30 12:21:12	hadoop	RUNNING	9.2 min

Completed Applications (3)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20201130114342-0001	Spark shell	6	1024.0 MiB		2020/11/30 11:43:42	hadoop	FINISHED	37 min
app-20201130115212-0002	PySparkShell	0	1024.0 MiB		2020/11/30 11:52:12	hadoop	FINISHED	4.3 min
app-20201130111835-0000	Spark shell	6	1024.0 MiB		2020/11/30 11:18:35	hadoop	FINISHED	13 min

standalone의 UI이다. Running Applications에서 pyspark shell을 클릭해보자.

Jobs | Stages | Storage | Environment | Executors | SQL — PySparkShell application UI

Spark Jobs (?)

User: hadoop
Total Uptime: 26 min
Scheduling Mode: FIFO
Completed Jobs: 1

Event Timeline

Completed Jobs (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	count at <stdin>:1 count at <stdin>:1	2020/11/30 12:22:46	3 s	1/1	2/2

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

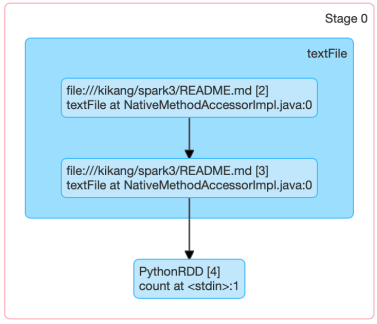
방금 수행된 `rdd.count()` job을 확인할 수 있다. 좀 더 자세히 확인해보자.

Spark 3.0.1 Jobs Stages Storage Environment Executors SQL PySparkShell application

Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 4 s
Locality Level Summary: Process local: 2
Input Size / Records: 6.6 KiB / 108
Associated Job Ids: 0

DAG Visualization



Show Additional Metrics
Event Timeline

Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	2 s	2 s	2 s	2 s
GC Time	92.0 ms	92.0 ms	0.1 s	0.1 s	0.1 s
Input Size / Records	2.2 KiB / 49	2.2 KiB / 49	4.4 KiB / 59	4.4 KiB / 59	4.4 KiB / 59

Showing 1 to 3 of 3 entries

Aggregated Metrics by Executor

Tasks (2)

Show 20 entries Search:

Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	0	0	SUCCESS	PROCESS_LOCAL	0	10.128.0.9	stdout stderr	2020-11-30 21:22:46	2 s	0.1 s	4.4 KiB / 49	
1	1	0	SUCCESS	PROCESS_LOCAL	1	10.128.0.7	stdout stderr	2020-11-30 21:22:46	2 s	92.0 ms	2.2 KiB / 59	

Showing 1 to 2 of 2 entries Previous 1 Next

DAG Visualization에는 이 코드가 어떠한 physical plan을 거쳐서 수행됐는지 보여준다. 처음에 README.md을 읽고, 그 후에 count action을 취한, 아주 간단한 스파크 잡이다.

Tasks을 보자. Executor ID가 0,1인 워커에서 이 잡이 분산되어 실행되었음을 확인할 수 있다.

파이썬이나 R에서 어떤 파일을 읽어오면 변수에 저장하기 마련이다. 그런데 스파크는 애초에 빅데이터를 처리하기 위해 만들어진 프레임워크이다. 따라서 엄청 큰 파일을 읽어오면, 이 읽어온 것을 어디에 저장하는 것을 생각하지 않는다. 따라서 위와 같이 한 번 README.md에 대해서 count job을 수행하고 또 이 job이 필요하다면, 다시 처음부터 불러와서 count을 해야하는 것이다.