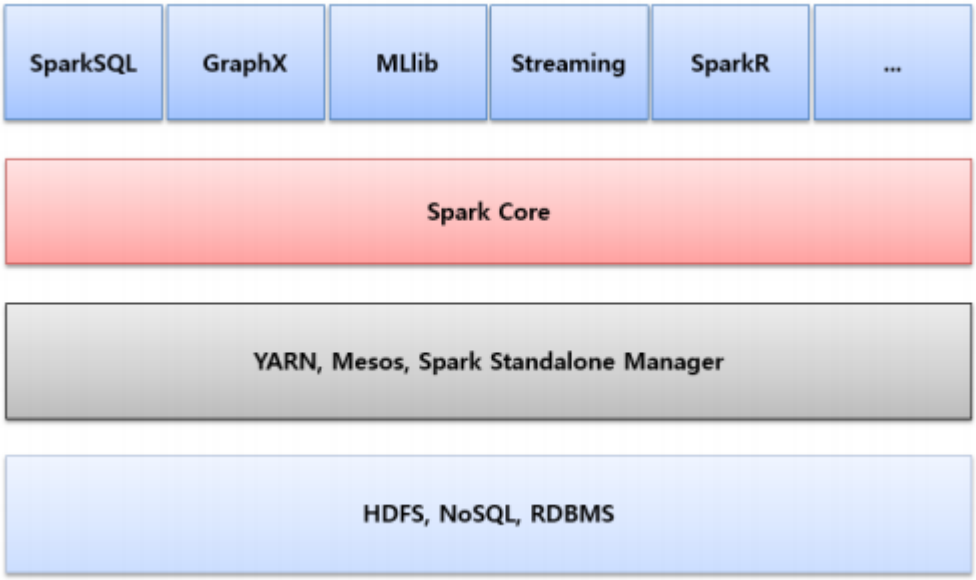
1. 스파크 아키텍처



2. 설치

2.1 설치 전 필요 사항

- virtualbox
- linux(centos)가 설치된 3개의 노드
- jdk 설치

2.2 리눅스 설정

```
# 호스트 이름 정의
vi /etc/hosts
127.0.0.1       localhost
10.10.10.100    add-m01
10.10.10.101    add-s01
10.10.10.102    add-s02

# 비밀번호 없이 로그인
ssh-keygen
cd ~/.ssh
ssh-copy-id demo@add-m01
ssh-copy-id demo@add-s01
ssh-copy-id demo@add-s02

# 환경 변수 세팅
```

```
vi .bashrc
export JAVA_HOME=/home/demo/jdk
export JRE_HOME=/home/demo/jdk/jre
export SPARK_HOME=/home/demo/spark
export PATH=$JAVA_HOME/bin:$SPARK_HOME/bin:$PATH
export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
source .bashrc
```

2.3 스파크 설치

```
# add-m01
curl -O https://archive.apache.org/dist/spark/spark-2.3.1/spark-2.3.1-bin-hadoop2.7.tgz
scp spark-2.3.1-bin-hadoop2.7.tgz demo@add-s01:/home/demo
scp spark-2.3.1-bin-hadoop2.7.tgz demo@add-s02:/home/demo

# add-m01, add-s01, add-s02 모두 설치
tar xvfz spark-2.3.1-bin-hadoop2.7.tgz
ln -s spark-2.3.1-bin-hadoop2.7 spark
```

2.4 클러스터 매니저 설정

스파크 스탠드얼론 클러스터

```
# 클러스터 매니저 워커 설정
cd $SPARK_HOME/conf
cp slaves.template slaves
vi slaves
add-m01
add-s01
add-s02

# 클러스터 매니저 실행, 중지
$SPARK_HOME/sbin/start-all.sh
$SPARK_HOME/sbin/stop-all.sh
```

2.5 확인

2.5.1 콘솔

```
jps
20935 Jps
19016 Master
19085 Worker

ssh add-s01 jps
5876 Jps
5561 Worker

ssh add-s02 jps
5812 Jps
5517 Worker
```

2.5.2 웹 UI

- 스탠드얼론 클러스터 default port 정보
  - 마스터 port : 7077
  - 웹 UI port : 8080

```
# 클러스터 매니저 웹 UI
http://10.10.10.100:8080/

# 앱 실행
spark-submit --master spark://add-m01:7077 $SPARK_HOME/examples/src/main/python/pi.py 10
```

3. 동작 이해

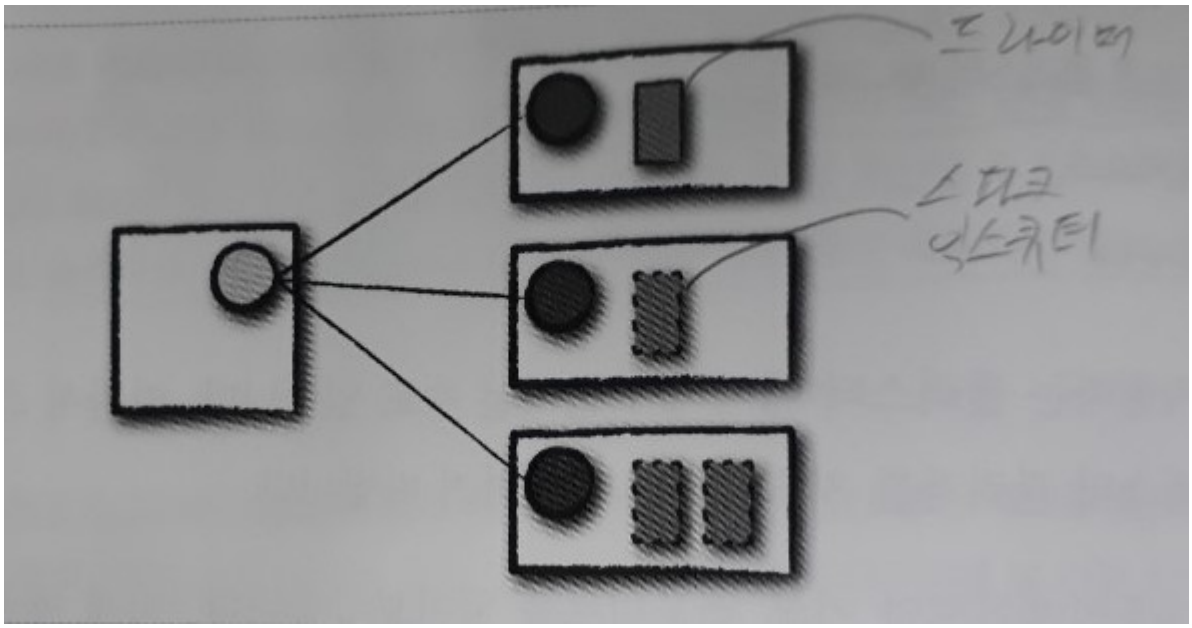
- 스파크 드라이버
  - 스파크 애플리케이션의 실행을 제어
  - 스파크 클러스터의 모든 상태 정보 유지
  - 물리적 컴퓨팅 자원 확보
  - 익스큐터 실행을 위해 클러스터 매니저와 통신
- 스파크 익스큐터
  - 스파크 드라이버가 할당한 태스크를 수행하는 프로세스
  - 태스크를 실행하고 태스크의 상태와 결과를 드라이버에 보고
- 클러스터 매니저
  - 드라이버(마스터)와 워커로 구성
  - 데몬 프로세스

3.1 실행 모드

애플리케이션을 실행할 때 요청한 자원의 물리적인 위치 결정

3.1.1 클러스터 모드

워커 노드에 드라이버와 익스큐터 프로세스를 실행

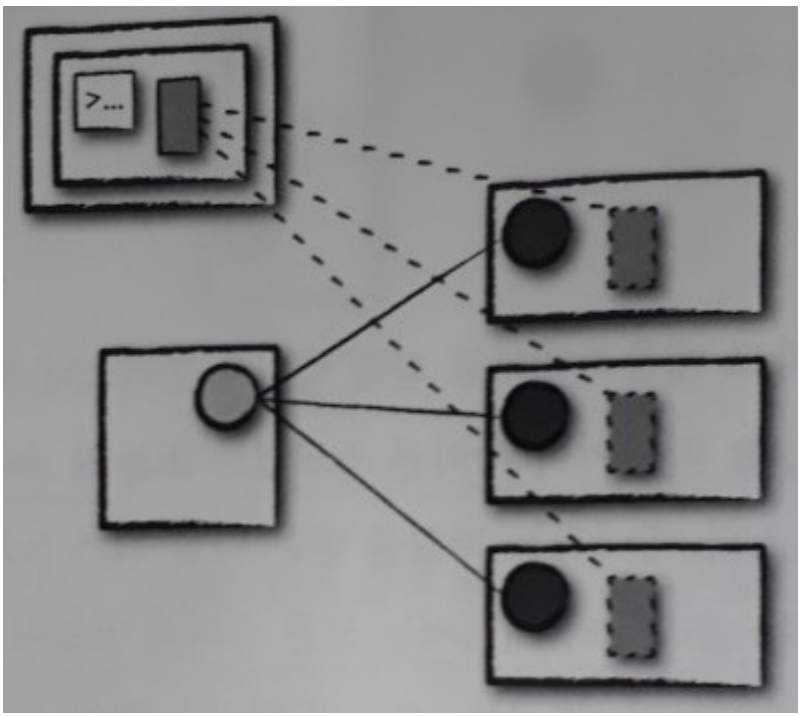


3.1.1.1 사용법

```
pyspark --master yarn --deploy-mode cluster

spark-submit --master yarn --deploy-mode cluster
$SPARK_HOME/examples/src/main/python/pi.py 10
```

3.1.2 클라이언트 모드

독립적인 클라이언트 머신에 스파크 드라이버를 실행하고 워커 노드에 익스큐터 프로세스를 실행



3.1.2.1 사용법

```
pyspark --master spark://add-m01:7077 --deploy-mode client

spark-submit --master spark://add-m01:7077 --deploy-mode client
$SPARK_HOME/examples/src/main/python/pi.py 10
```

3.1.3 로컬 모드

- 스파크 애플리케이션이 단일 머신에서 실행됨
- 병렬 처리를 위해 단일 머신의 스레드 활용

3.1.1.1 사용법

```
pyspark --master local
spark-submit --master local $SPARK_HOME/examples/src/main/python/pi.py 10
```

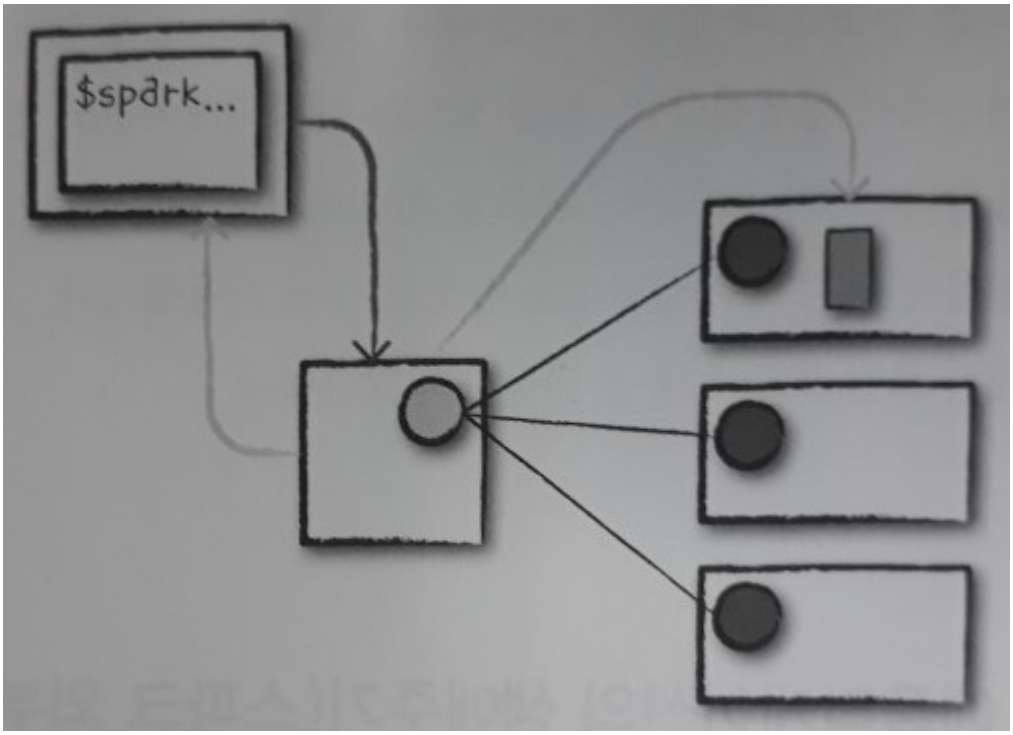
3.2 생애 주기

3.2.1 클러스터 매니저 관점

클러스터 매니저 입장에서 스파크 잡이 할당되고 해제되는 과정

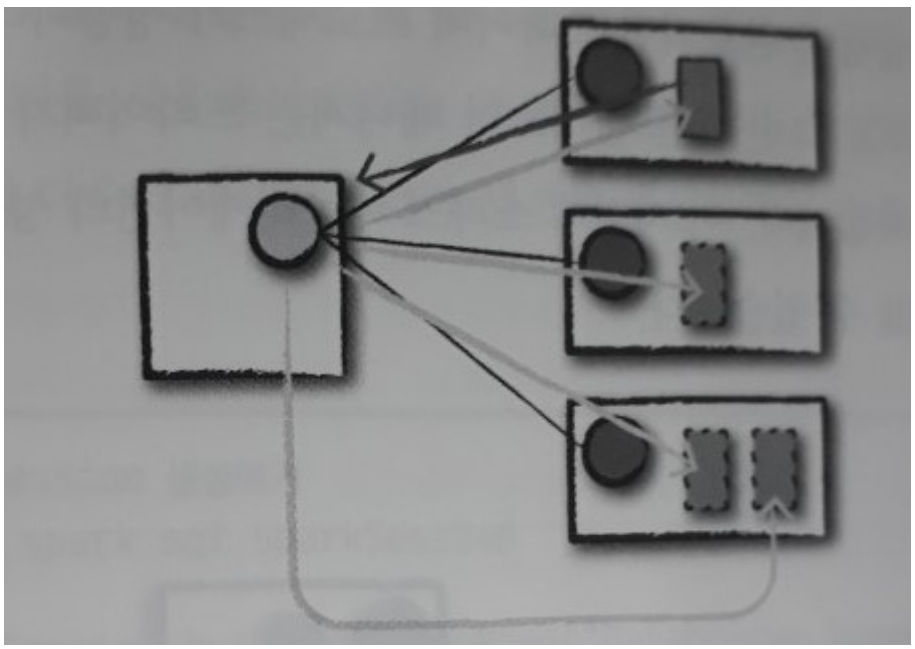
3.2.1.1 요청

- 클라이언트 프로세스가 스파크 애플리케이션 제출
- 스파크 드라이버 프로세스의 자원 요청
- 워커 중 1개에 드라이버 프로세스 실행
- 클라이언트 프로세스 종료



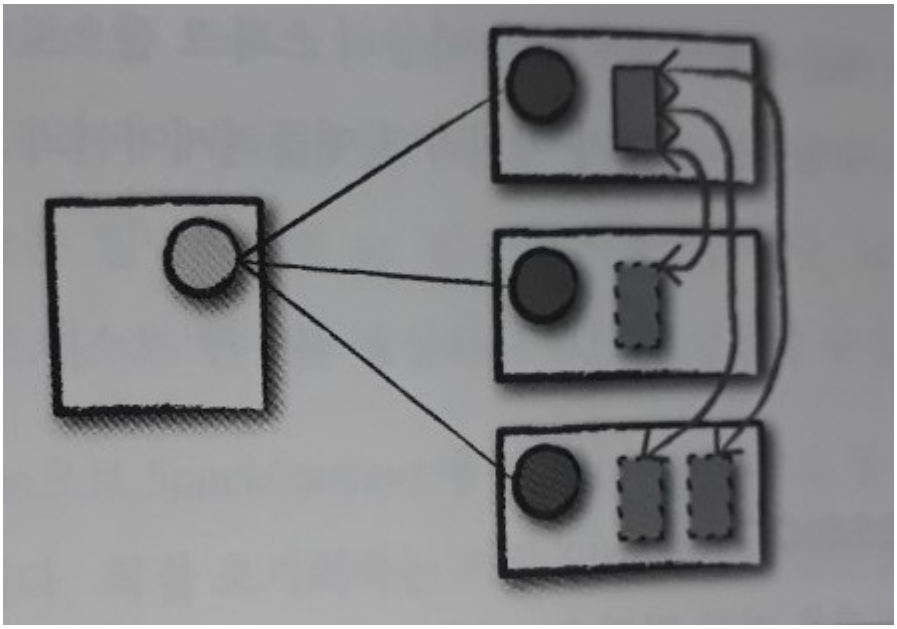
3.2.1.2 시작

- 사용자 코드 실행
- 사용자 코드의 SparkSession이 클러스터 매니저와 통신
- 스파크 익스큐터 프로세스의 실행 요청
- 클러스터 매니저가 스파크 익스큐터 프로세스 시작
- 익스큐터 위치 정보를 드라이버 프로세스로 전송
- 스파크 클러스터 완성



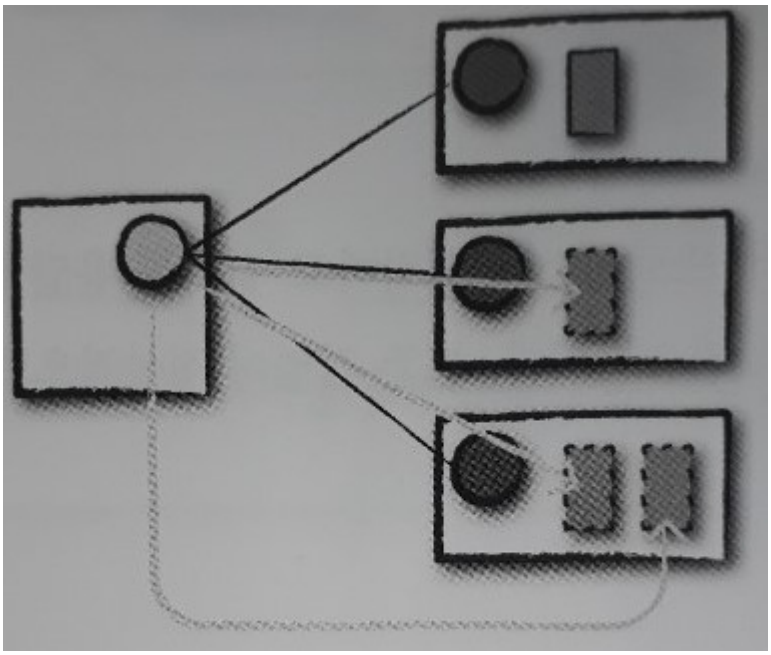
3.2.1.3 실행

- 코드 실행
- 드라이버가 각 워커에 태스크 할당
- 태스크의 상태, 성공/실패 여부를 드라이버에 전송

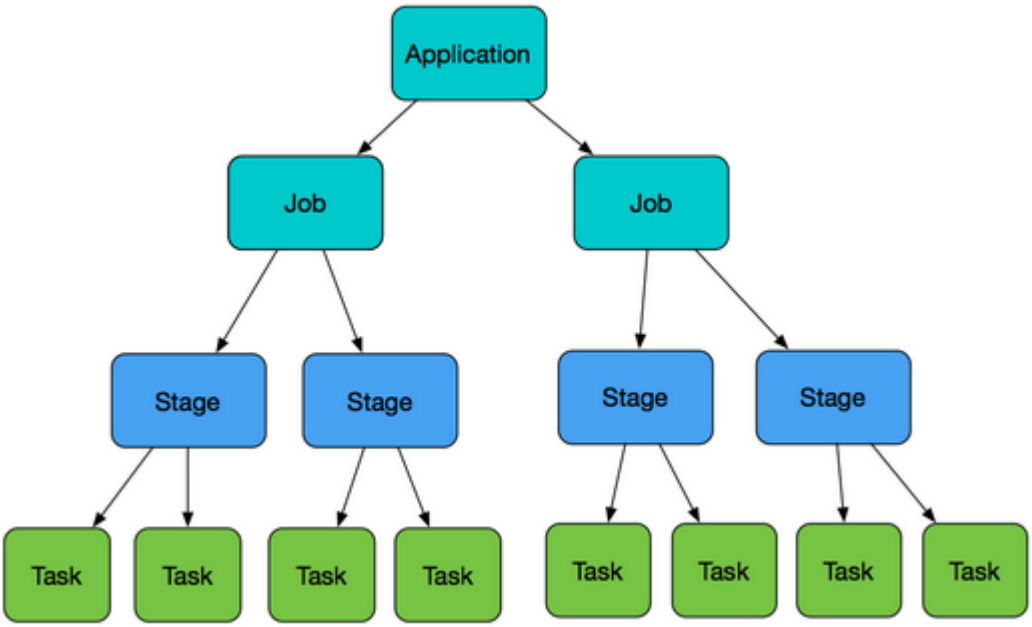


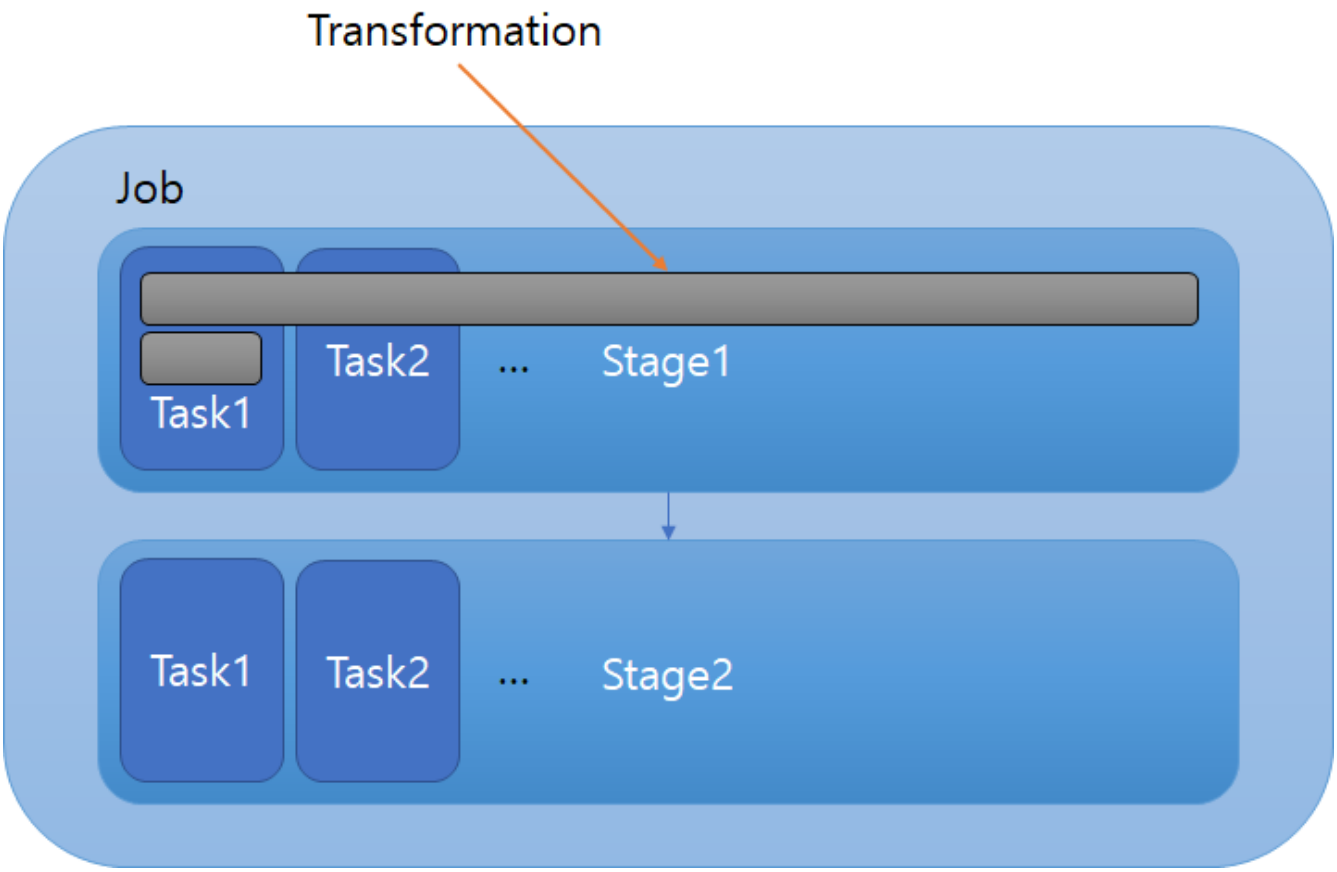
3.2.1.4 완료

- 드라이버 프로세스가 성공/실패 중 하나의 상태로 종료됨
- 클러스터 매니저가 드라이버가 속한 스파크 클러스터의 모든 익스큐터 종료



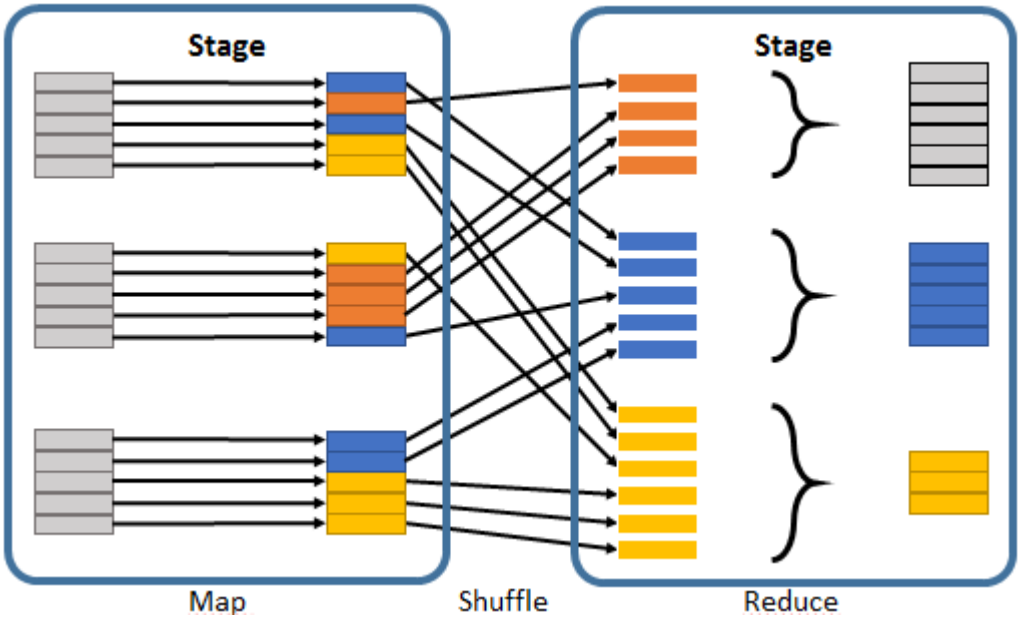
3.2.2 스파크 내부 관점





- 잡
  - 일련의 스테이지로 나뉨
  - Action으로 하나의 스파크 잡이 실행되고 완료됨
- 스테이지
  - 스테이지의 수는 셔플 작업이 얼마나 많이 발생하는지에 달림
  - 다수의 머신에서 동일한 연산을 수행하는 태스크의 그룹
    - 가능하면 많은 태스크(트랜스포메이션)을 동일한 스테이지로 묶으려 노력

    - 셔플이 끝난 다음 새로운 스테이지를 시작
- 태스크
    - 태스크는 단일 익스큐터에서 실행할 데이터의 블록과 다수의 트랜스포메이션 조합
    - 태스크는 데이터 단위에 적용되는 연산 단위를 의미

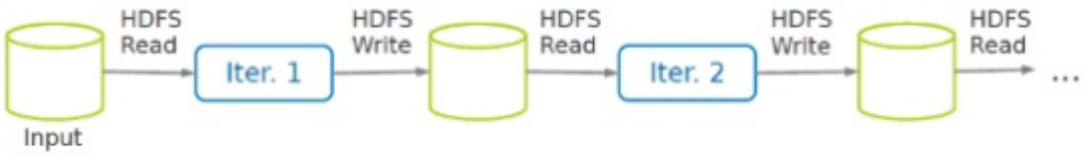


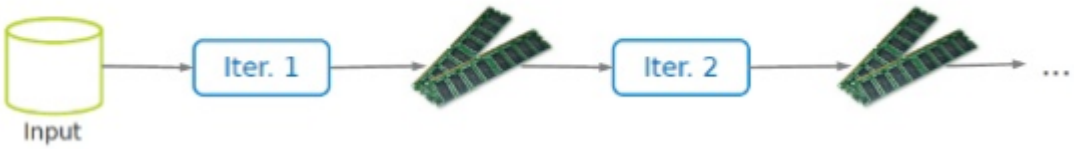
- 셔플
    - 데이터의 물리적 재분배 과정
    - DataFrame의 정렬(sort)이나 키별로 적재된 파일 데이터를 그룹화(groupbykey) 하는 작업
    - 스파크 잡이 실행되는 도중에 셔플을 수행하면 `spark.sql.shuffle.partitions`개의 셔플 파티션 생성
        - `spark.sql.shuffle.partitions` : 클러스터의 코어 수에 맞추어 설정
        - 참고 사이트 : https://ourcstory.tistory.com/147
- 예제

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .master("local") \
    .appName("Word Count") \
    .getOrCreate()

df1 = spark.range(2, 10000000, 2) # 2,4,6,...
df2 = spark.range(2, 10000000, 4) # 2,6,10,...
step1 = df1.repartition(5) # 셔플링
step12 = df2.repartition(6) # 셔플링
step2 = step1.selectExpr("id * 5 as id") # 10, 20, 30,...
step3 = step2.join(step12, ["id"]) # 셔플링
step4 = step3.selectExpr("sum(id)")
step4.collect() # Action, 결과 전송을 위한 셔플링
step4.explain() # 실행계획 출력

== Physical Plan ==
*(7) HashAggregate(keys=[], functions=[sum(id#6L)])
+- Exchange SinglePartition
```

```
+- *(6) HashAggregate(keys=[], functions=[partial_sum(id#6L)])
   +- *(6) Project [id#6L]
      +- *(6) SortMergeJoin [id#6L], [id#2L], Inner
         :- *(3) Sort [id#6L ASC NULLS FIRST], false, 0
         :  +- Exchange hashpartitioning(id#6L, 200)
         :     +- *(2) Project [(id#0L * 5) AS id#6L]
         :        +- Exchange RoundRobinPartitioning(5)
         :           +- *(1) Range (2, 10000000, step=2, splits=3)
         +- *(5) Sort [id#2L ASC NULLS FIRST], false, 0
            +- Exchange hashpartitioning(id#2L, 200)
               +- Exchange RoundRobinPartitioning(6)
                  +- *(4) Range (2, 10000000, step=4, splits=3)
```

3.2.3 세부 실행 과정





3.2.3.1 파이프라이닝

- map -> filter -> map을 1개의 스테이지로 구성 후, 첫 번째 map 결과를 중간 저장하지 않고 개별 레코드를 읽어 map -> filter -> map 수행 후 저장한다.
- 메모리나 디스크에 중간 결과를 기록하는 방식보다 훨씬 빠르다.

3.2.3.2 셔플 결과 저장

- 셔플 파일을 로컬디스크에 기록
- 잡이 실패한 경우 셔플 파일을 참조하여 해당 스테이지부터 처리
- 이미 셔플된 데이터를 이용해 새로운 잡을 실행하면 셔플 파일을 사용하여 이전스테이지를 처리하지 않음

4. 프로그래밍 방법

4.1 대화 쉘

4.1.1 pyspark

```
pyspark
Python 2.7.5 (default, Jul 13 2018, 13:06:57)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
2019-08-19 19:48:19 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
2019-08-19 19:48:21 WARN  Utils:66 - Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Python version 2.7.5 (default, Jul 13 2018 13:06:57)
SparkSession available as 'spark'.
>>>
```

4.1.2 jupyter notebook

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .master("spark://10.10.10.100:7077") \
    .appName("Word Count") \
    .getOrCreate()
spark
```

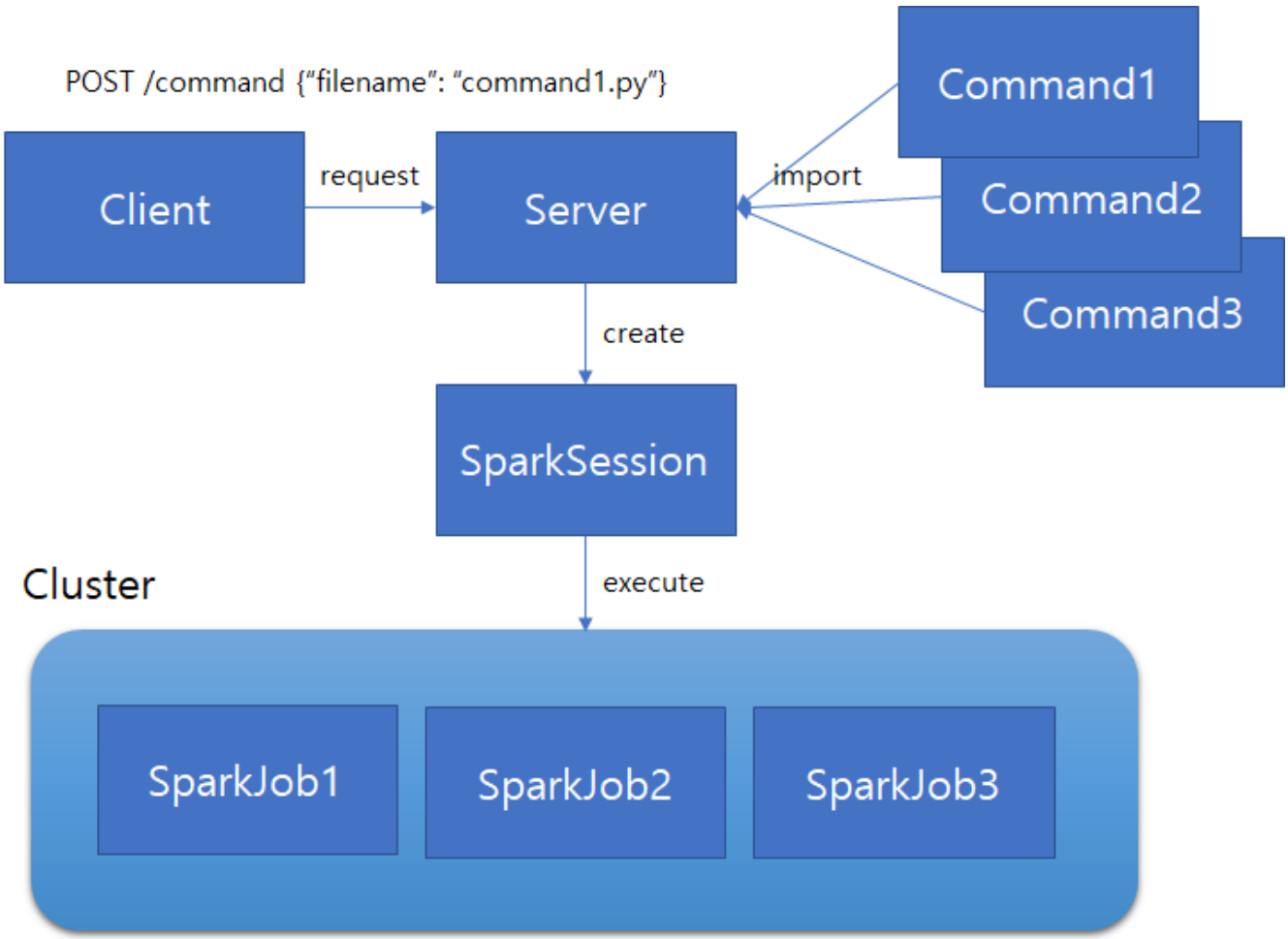
4.2 애플리케이션 제출

4.2.1 spark-submit

```
spark-submit --master spark://add-m01:7077 --deploy-mode client
$SPARK_HOME/examples/src/main/python/pi.py 10
```

- https://spark.apache.org/docs/latest/submitting-applications.html

4.3 미니 프로젝트

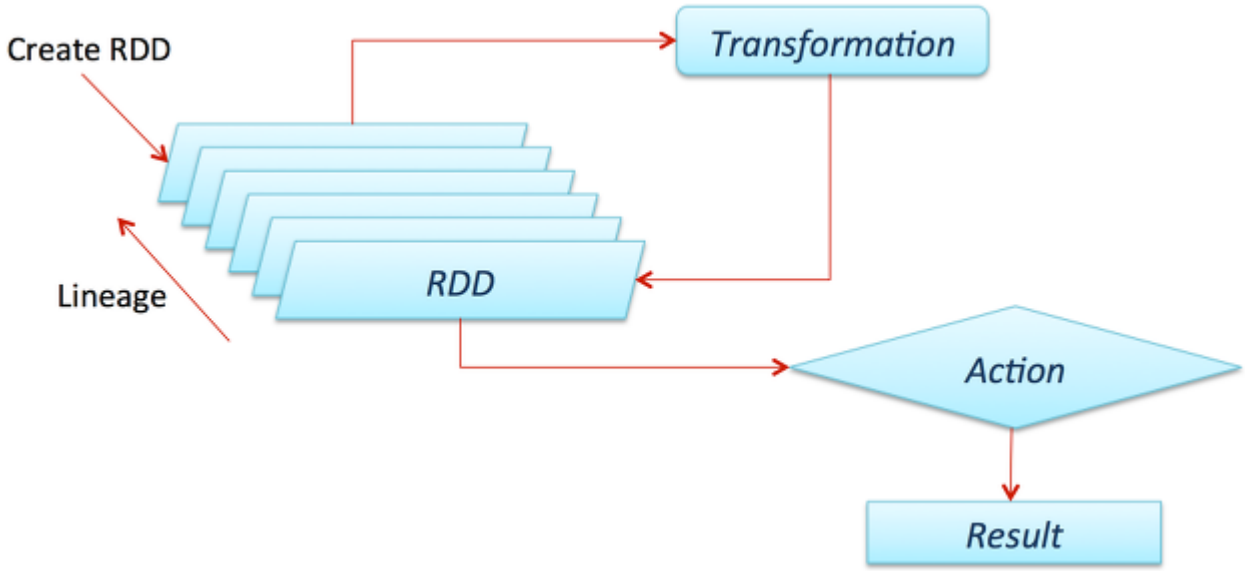


5. RDD

5.1 RDD 란

탄력적인 분산 데이터셋(Resilient Distributed Dataset)

불변성을 가지며 병렬로 처리할 수 있는 파티셔닝된 레코드의 모음



파이썬을 사용해 RDD를 다룰때는 높은 오버헤드가 발생, 꼭 필요한 경우가 아니라면, 파이썬에서는 구조적 API(DataFrame)를 사용하는 것이 좋음

5.2 예제

5.2.1 RDD 생성

- DataFrame을 rdd로 변환

```
spark.range(10).rdd
```

- 파이썬 객체로 RDD 생성

```
myCollection = "Spark The Definitive Guide : Big Data Processing Made Simple"\
  .split(" ")
words = spark.sparkContext.parallelize(myCollection, 2)
```

- 데이터 소스로 생성

```
words = spark.sparkContext.textFile("path")
```

5.2.2 트랜스포메이션

지연 처리 방식

- distinct

```
words.distinct().count()
words.distinct().collect()
```

- filter

```
def startsWithS(individual):
  return individual.startswith("S")

words.filter(lambda word: startsWithS(word)).collect()
words.filter(lambda word: word.startswith("S")).collect()
```

- map

```
words2 = words.map(lambda word: (word, word[0], word.startswith("S")))
words2.filter(lambda record: record[2]).take(5)
```

- flatMap

```
# 단어를 문자 집합으로 변환
words.flatMap(lambda word: word).take(15)
```

- sortBy

```
words.sortBy(lambda word: len(word) * -1).take(2)
```

- randomSplit

```
randomSplit = words.randomSplit([0.1, 0.9])
print randomSplit[0].collect()
print randomSplit[1].collect()
```

- https://spark.apache.org/docs/2.3.1/rdd-programming-guide.html#transformations

5.2.3 액션

즉시 실행 방식, 데이터를 드라이버로 모으거나 외부 데이터 소스로 보냄

- reduce

```
spark.sparkContext.parallelize(range(1, 21)).reduce(lambda x, y: x + y) # 210

def wordLengthReducer(leftWord, rightWord):
  if len(leftWord) > len(rightWord):
    return leftWord
  else:
    return rightWord

words.reduce(wordLengthReducer)
```

- count

```
words.count()
```

- first

```
words.first()
```

- max, min

```
spark.sparkContext.parallelize(range(1, 21)).max()
spark.sparkContext.parallelize(range(1, 21)).min()
```

- take
  - 먼저 하나의 파티션을 스캔, 그 다음 해당 파티션의 결과 수를 이용해 파라미터로 지정된 값을 만족하는데 필요한 추가 파티션 수를 예측

```
words.take(5)
words.takeOrdered(5)
```

- https://spark.apache.org/docs/2.3.1/rdd-programming-guide.html#actions

6. DataFrame

7. DataSource