RDD

RDD(Resilient Distributed DataSet) 소개와 연산

프로그램 세계에서 모델링이란 현실 세계에 존재하는 사물이나 개념을 프로그래밍 언어로 설명하는 과정

RDD란 스파크에서 사용하는 핵심 데이터 모델로써 다수의 서버에 걸쳐 분산 방식으로 저장된 데이터 요소들의 집합을 의미, 병렬처리가 가능하고 장애가 발생할 경우에도 스스로 복구될 수 있는 내성(rolerance)를 가지고 있다.

which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel

Spark revolves around the concept of a *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel.

파티션: 하나의 RDD에 속한 요소들은 파티션이라고 하는 더 작은 단위로 나눠질 수 있는데, 스파크는 작업을 수행할 때 이 파티션 단위로 나눠서 병렬로 처리를 수행합니다.

셔플링: 파티션은 작업이 진행되는 과정에서 재구성되거나 네트워크를 통해 다른 서버로 이동
셔플링은 전체 작업 성능에 큰 영향을 줌

복구: RDD에 포함된 데이터를 저장해 두는 것이 아니고 RDD를 생성하는데 사용했던 작업 내용을 기억하고 문제가 발생하면 전체 작업을 처음부터 다시 실행하는 대신 문제가 발생한 RDD를 생성했던 작업만 다시 수행해서 복구
- 한번 생성된 RDD가 바뀌지 않아야 한다는 조건

리니지: 스파크에서 RDD 생성 작업을 기록해 두는 것

RDD

spark가 사용하는 기본 데이터 모델

Resilient Distributed Datasets

들어가기 전에 알아둬야 할 사항

- spark cluster
  - 데이터가 여러 서버에 나눠져 병렬 처리
- 분산 데이터로서 RDD
  - Resilient Distributed Datasets : 회복력을 가진 분산 데이터 집합
  - 다수의 "데이터 요소"가 모인 집합
  - 이 요소들이 다시 일정한 단위의 집합으로 나눠져 스파크 클러스터에 흩어져서 저장
- RDD 불변성
  - 한번 만들어진 RDD는 어떤 경우라도 그 내용이 변경되지 않는다.
  - RDD를 만드는 방법만 기억하고 있다면 언제든 똑같은 데이터를 다시 만들 수 있다.
- 파티션
  - 분할된 데이터를 파티션이라는 단위로 관리
  - 하둡파일시스인 HDFS를 사용한다면 하나의 HDFS블록에 하나의 파티션이 구성

들어가기 전에 알아둬야 할 사항

- HDFS
  - 하둡파일 시스템
  - 스파크의 데이터 입력과 출력은 하둡의 InputFoirmat과 OutputFormat을 이용
  - 하둡은 일반 텍스트파일로부터 SequenceFile, Parquet등 다양한 입출력 포맷 지원
    스파크도 동일한 입출력 유형 지원
  - 하둡은 데이터를 읽어들일 때 설정된 InputSplit분할 정책이 따라 전체 데이터를 블록 단위로 분할, 별도 설정이 없을 경우 이 블록이 스파크의 파티션 단위가 됨.
    하지만 스파크에서 제공하는 별도의 매개변수를 이용하면 이를 원하는 값으로 조정할 수 있다.
- Job과 Executor
  - 스파크 프로그램을 실행하는 것을 스파크 잡(Job)을 실행한다고 함. 하나의 Job은 클러스터에서 병렬로 처리되는데 이때 각 서버마다 익스큐터라는 프로세스가 생성됨. 이를 통해 각자 할당된 파티션을 처리하게 됨

들어가기 전에 알아둬야 할 사항

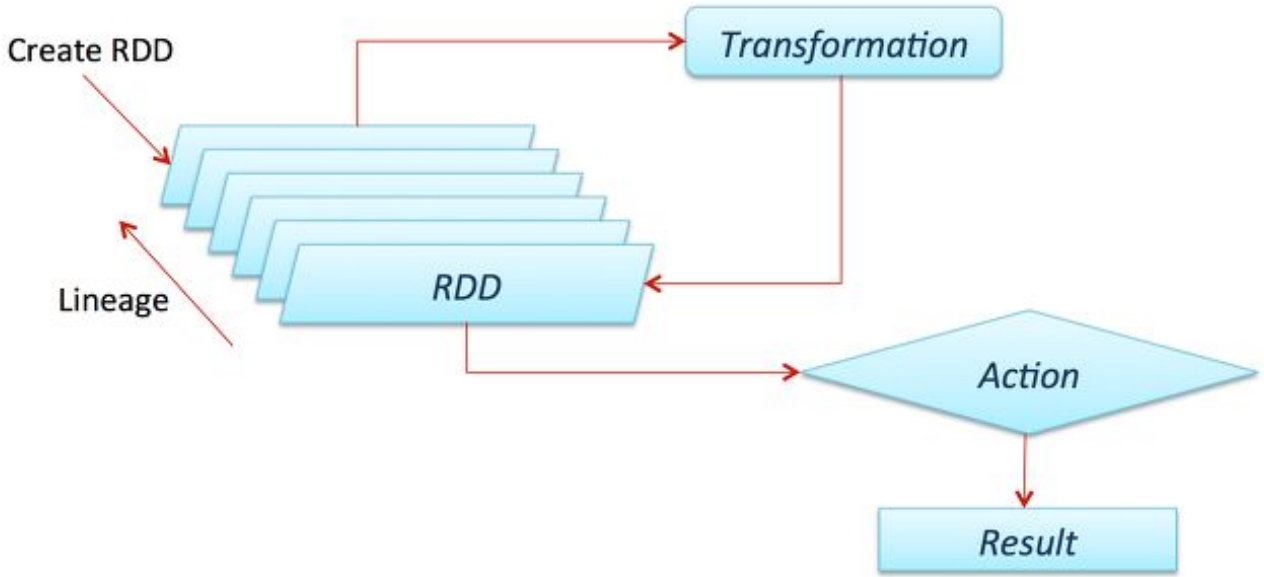
- 드라이버 프로그램
  - 메인 함수를 가지고 있는 프로그램을 가리켜 드라이버
  - spark context를 생성하고 그 instance를 포함하고 있는 프로그램
  - 자신을 실행하는 서버에서 동작하면서 스파크컨텍스트를 생성해 클러스터의 각 워커 노드들에게 작업을 지시하고 결과를 취합하는 역할을 수행한다.
  - 별도의 작업용 서버를 사용하는 경우 많음
- 트랜스포메이션과 액션
  - transformation
    - RDD의 형태를 변형하는 연산
  - action
    - 어떤 동작을 수행해 그 결과로서 RDD가 아닌 다른 타입의 결과를 반환하는 연산
  - 리턴타입이 RDD인 경우는 트랜스포메이션, 그 외에는 액션

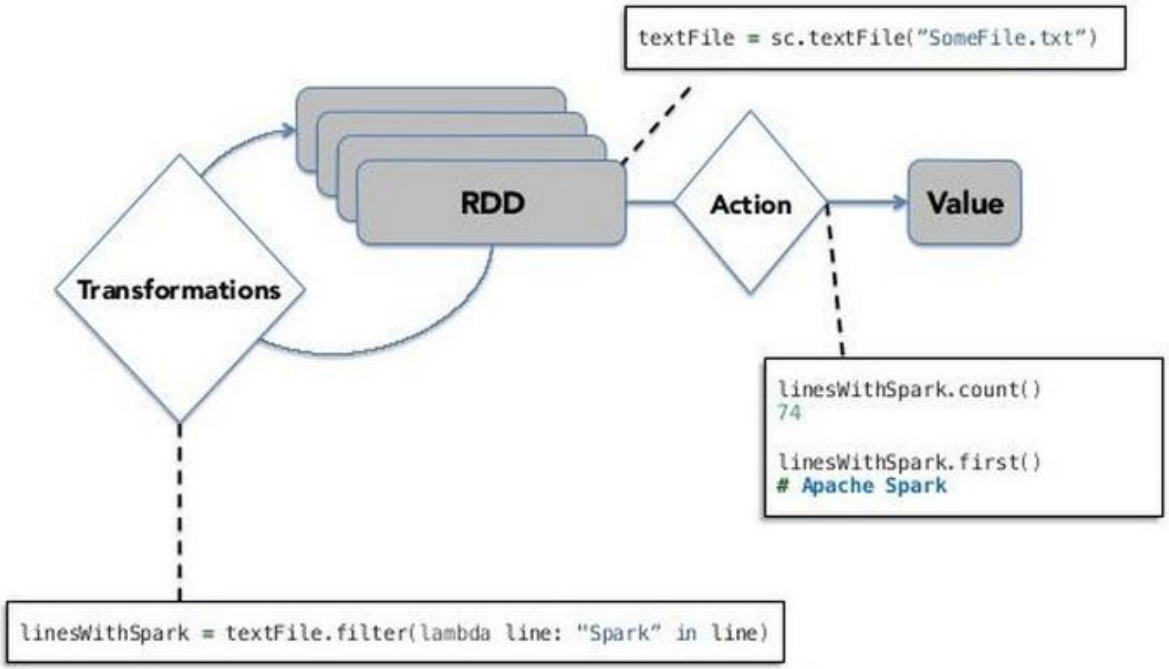
들어가기 전에 알아둬야 할 사항

- 지연(lazy)동적과 최적화
  - Transformation에 해당하는 연산인 경우 해당 RDD를 사용하는 다른 Action 연산이 호출될 때까지는 실제 Transformation을 수행하지 않는 방식
  - 이 방식을 장점은 실행 계획의 최적화가 가능: 사용자가 입력한 변환연산을 즉시 수행하지 않고 모아뒀다가 가장 최적의 수행방법을 찾아 처리할 수 있기 때문에
- 함수의 전달
  - val rdd2 = rdd1.map(_ + 1) // rdd1의 각 원소에 1을 더한다.
  - RDD의 map() 메서드에 전달된 함수는 클러스터를 구성하는 각 서버에서 동작할 수 있도록 클러스터이에 속한 모든 워커 서버에 전달되어야 하기 때문에
    - Serializable 인터페이스 구현 필요
    - scala: object class
    - java: Function interface
    - python: 모듈 최상위 위치 똔느 지역함수를 선언해 사용

들어가기 전에 알아둬야 할 사항

- 데이터 타입에따른 RDD 연산
  - RDD가 제공하는 메서드는 데이터의 타입과 밀접한 관계를 맺고 있습니다.
  - PairRDDFunctions
    - 키와 값의 형태로 구성된 데이터에 대한 연산을 제공
    - groupByKey, reduceByKey, mapValues
  - OrderedRDDFunctions
    - 키와 값으로 구성된 데이터 대상
    - 키에 대한 정렬 가능한 경우를 위한 것
  - DoubleRDDFunctions
    - double 유형의 데이터를 위한 연산 제공
  - SequenceFileRDDFunctions
    - 하둡의 시퀀스 파일을 다루기 위한 연산 제공


Create RDD
Transformation
RDD
Lineage
Action
Result


textFile = sc.textFile("SomeFile.txt")
RDD
Action
Value
Transformations
linesWithSpark.count()
74
linesWithSpark.first()
# Apache Spark
linesWithSpark = textFile.filter(lambda line: "Spark" in line)

2.1.2 스파크 컨테스트 생성

spark context

- spark application과 cluster의 연결을 관리하는 객체

```
val conf = new SparkConf().setMaster("local").setAppName("RDDSample")

val sc = new SparkContext(conf)
```

2.1.3 RDD 생성

```
val rdd1 = sc.parallelise(List("a", "b", "c", "d", "e"))

val rdd1 = sc.textFile("<spark_home_dir>/README.md")
```