스파크 응용 모니터링

순천향대학교 컴퓨터공학과

이 상 정

학습 내용

1. 스파크 실행 컴포넌트
2. 스파크 웹 모니터링
3. 스파크 응용 디버그 및 튜닝

1. 스파크 실행 컴포넌트

스파크 실행 모델 – 논리적 실행 계획 (1)

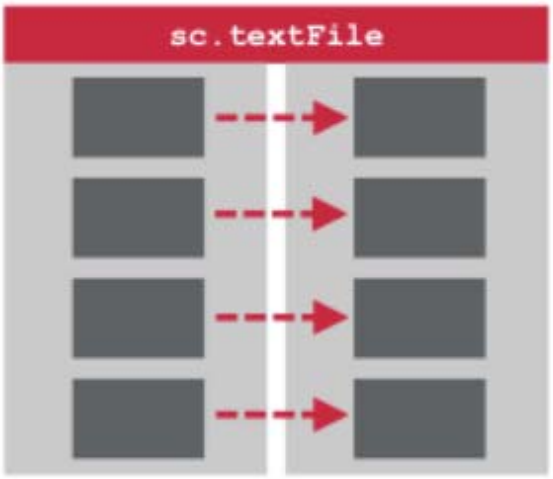
- 사용자 프로그램이 물리적인 실행 단위로 변환되어 실행되는 모델을 고찰
- 먼저, 앞의 SFPD 예에 대한 논리적 실행 계획을 살펴 보면서 스파크 실행 모델의 컴포넌트들을 조사

```
1. val inputRDD = sc.textFile("/sparkdata/sfpd/sfpd.csv")

2. val sfpdRDD = inputRDD.map(line=>line.split(","))

3. val catRDD = sfpdRDD.map(x=>(x(Category),1))
                       .reduceByKey((a,b)=>a+b)
```

스파크 실행 모델 – 논리적 실행 계획 (2)

- sfpd.csv 에서 inputRDD 를 생성

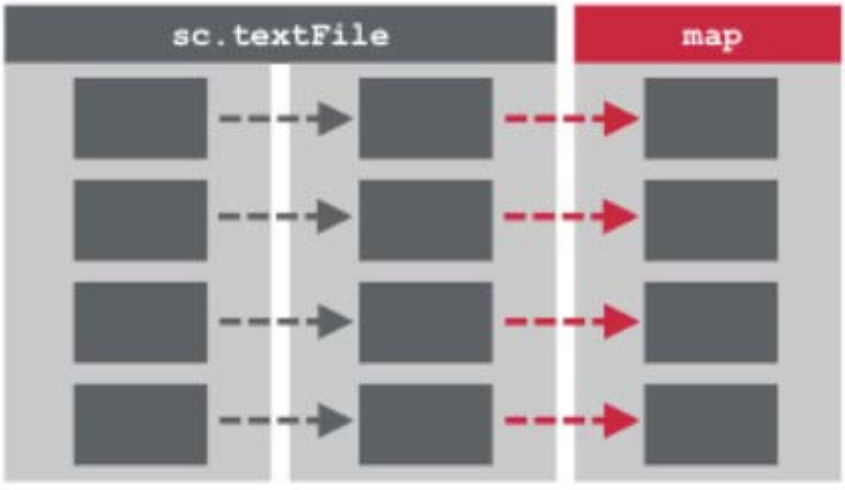
1. val inputRDD = sc.textFile("/sparkdata/sfpd/sfpd.csv")



스파크 실행 모델 – 논리적 실행 계획 (3)

- inputRDD를 ‘,’ 로 기준으로 분리하여 sfpdRDD 를 생성

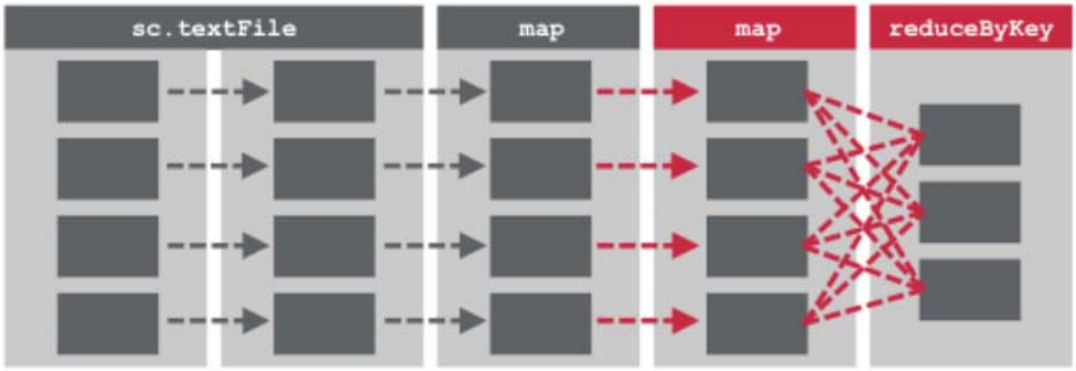
2. val sfpdRDD = inputRDD.map(line=>line.split(","))



스파크 실행 모델 – 논리적 실행 계획 (4)

- Map과 reduceByKey 변환을 적용하여 catRDD 생성
- 지금까지 액션이 실행되지 않았으므로 실행은 지연되고, 이들 RDD 객체들의 DAG(Directed Acyclic Graph)만 정의

3. val catRDD = sfpdRDD.map(x=>(x(Category),1))
   .reduceByKey((a,b)=>a+b)



스파크 실행 모델 – 계보(Lineage) 출력

- RDD의 계보는 rdd.toDebugString을 사용
  - 아래 예는 catRDD의 계보로 HadoopRDD, MapPartitionRDD, SuffledRDD 등의 생성 과정 표시

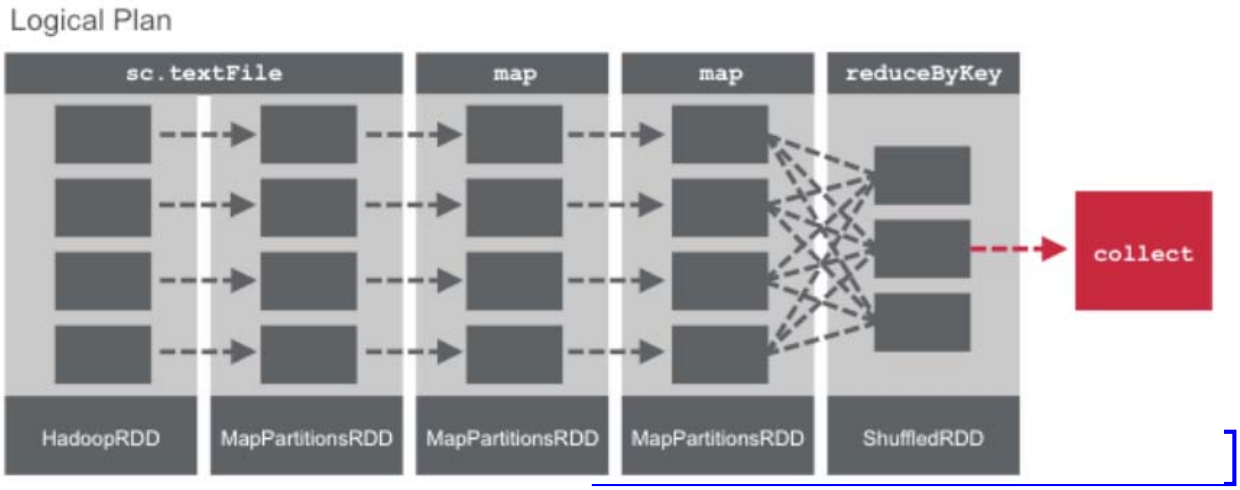
// Display catRDD lineage

catRDD.toDebugString

```
scala> val inputRDD = sc.textFile("/sparkdata/sfpd/sfpd.csv")
inputRDD: org.apache.spark.rdd.RDD[String] = /sparkdata/sfpd/sfpd.csv MapPartitionsRDD
[6] at textFile at <console>:24

scala> val sfpdRDD = inputRDD.map(line=>line.split(","))
sfpdRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[7] at map at <cons
ole>:26

scala> val catRDD = sfpdRDD.map(x=>(x(Category),1)).reduceByKey((a,b)=>a+b)
catRDD: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[9] at reduceByKey at <co
nsole>:30

scala> catRDD.toDebugString
res3: String =
(2) ShuffledRDD[9] at reduceByKey at <console>:30 []
 +-(2) MapPartitionsRDD[8] at map at <console>:30 []
    |  MapPartitionsRDD[7] at map at <console>:26 []
    |  /sparkdata/sfpd/sfpd.csv MapPartitionsRDD[6] at textFile at <console>:24 []
    |  /sparkdata/sfpd/sfpd.csv HadoopRDD[5] at textFile at <console>:24 []
```

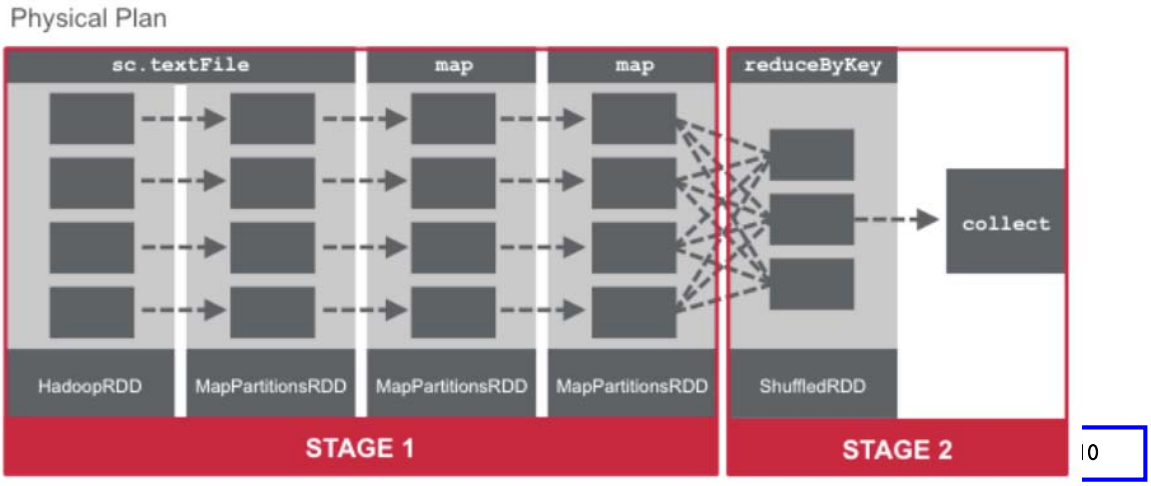
스파크 실행 모델 – 액션의 실행 (1)

- catRDD.collect 액션을 실행하면 계산을 시작
  - 스파크 스케줄러가 RDD를 계산하는 논리적인 실행 계획인 DAG의 계보를 사용하여 물리적인 실행 계획을 생성하고 실행
  - collect 액션이 호출되면 RDD의 모든 파티션들에 대한 계산이 실행되고 결과가 드라이버 프로그램으로 전달



스파크 실행 모델 – 액션의 실행 (2)

- 스케줄러는 그래프의 각 RDD 마다 계산 스테이지(stage)를 갖지만, 부모 스테이지의 데이터의 이동 없이 계산되는 경우 여러 스테이지를 단일 스테이지로 통합(collasping)
  - RDD 통합을 파이프라이닝(pipelining)이라고도 함
  - 아래 예에서는 map 연산들은 데이터 이동이 없어서 스테이지 1로 통합되고, 셔플은 스테이지 2가 됨

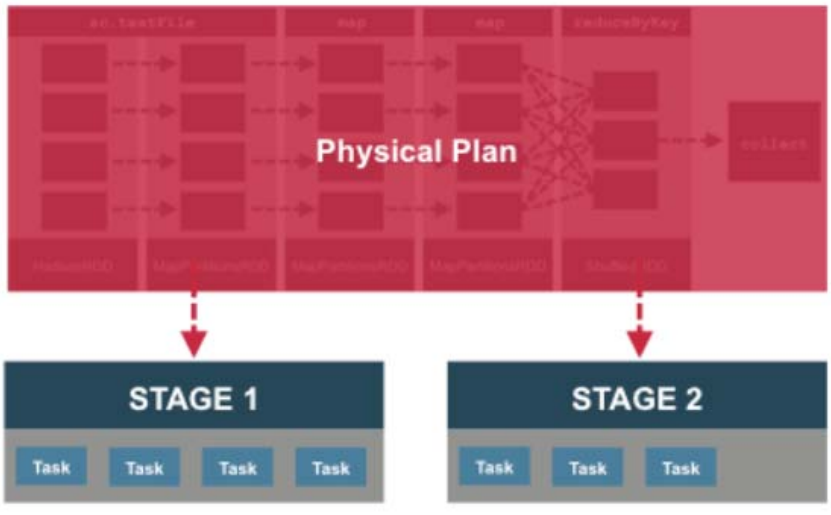


계보의 절단(Lineage Trucation)

- 스케줄러가 RDD의 계보를 절단 또는 통합하는 상황
  - 부모 RDD의 데이터 이동이 없어 여러 개의 RDD들을 단일 RDD로 통합하는 파이프라이닝의 상황
  - RDD가 클러스터의 메모리나 디스크에 지속(persist)되는 경우 스파크 스케줄러는 RDD 그래프를 절단하고, 향 후 지속된 RDD에서부터 계산을 시작
  - RDD가 셔플되어 계산된 경우 셔플의 출력은 디스크에 저장되므로 스케줄러는 계보를 절단

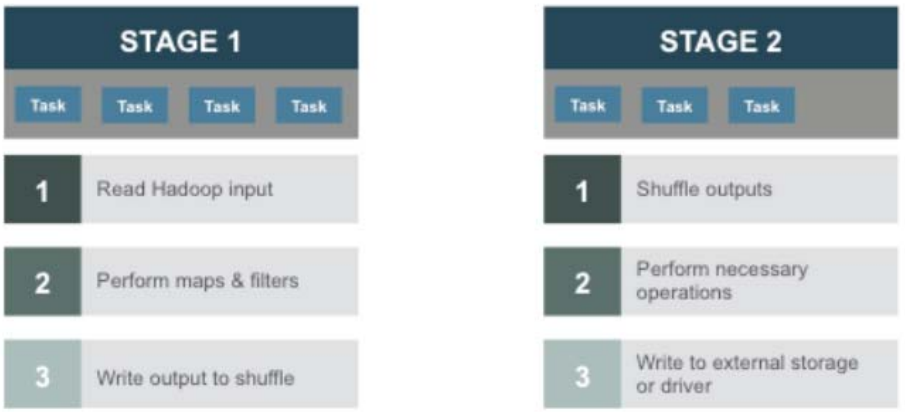
스파크 실행 모델 – 액션의 실행 (3)

- 액션이 호출되면 DAG는 액션의 실행에 필요한 RDD를 계산하는 물리적 실행 계획으로 변환
- 스파크 스케줄러는 모든 RDD들을 계산하는 작업(job)을 제출
  - 각 작업은 하나 이상의 스테이지(stage)를 갖고, 각 스테이지는 태스크(task)들로 구성
  - 각 스테이지는 순서대로 처리되고, 각 태스크들은 스케줄되어 클러스터 상에서 실행



스파크 실행 모델 – 스테이지와 태스크

- 스테이지는 해당 RDD의 각 파티션에 대한 태스크를 가짐
  - 스테이지는 각 파티션의 데이터에 대해 동일 태스크를 수행
- 각 태스크는 아래의 공통 단계를 수행
  1. (데이터 저장소 또는 기존 RDD 또는 셔플 출력 으로 부터) 데이터를 페치
  2. 원하는 RDD 계산을 위해 필요한 연산을 수행
  3. 셔플, 외부 저장소, 드라이버(예를들어 count, collect)로 RDD를 출력



스파크 실행 모델 – 실행 컴포넌트 (1)

- 태스크(task)는 하나의 RDD 파티션에 수행되는 스테이지 내의 작업 단위
- 스테이지(stage)는 병렬로 같은 계산을 수행하는 태스크들의 그룹
- 셔플(shuffle)은 스테이지들 간의 데이터 전달
- 작업(job)은 특정 액션을 수행하는 스테이지들의 집합

- 파이프라인(pipeline)은 부모 RDD의 데이터의 이동 없이 RDD가 계산될 때, 스케줄러는 여러 개의 RDD를 단일 스테이지로 통합
- DAG(Directed Acyclic Graph)는 RDD의 연산을 표현한 논리적인 그래프
- RDD(Resilient Distributed Dataset)는 여러 파티션으로 구성된 병렬 데이터 세트

스파크 실행 모델 – 실행 컴포넌트 (2)

Component	Description
Tasks	Unit of work within a stage corresponds to one RDD partition
Stages	Group of tasks which perform the same computation in parallel
Shuffle	Transfer of data between stages
Jobs	Work required to compute RDD; has one or more stages
Pipelining	Collapsing of RDDs into a single stage when RDD transformations can be computed without data movement
Directed Acyclic Graph (DAG)	Logical graph of RDD operations
Resilient Distributed Dataset (RDD)	Parallel dataset with partitions

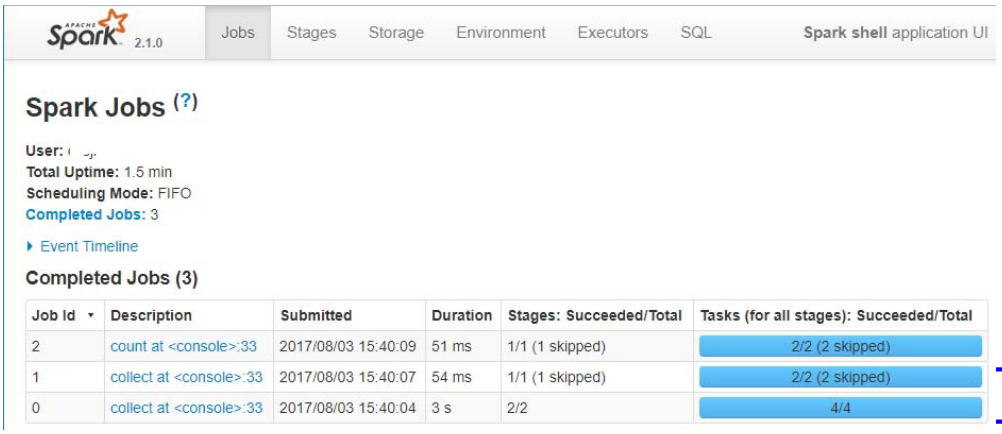
스파크 실행 단계 (Spark Execution Phases)

- 스파크의 실행 단계
  - 단계 1: 사용자 코드로 부터 DAG와 RDD를 정의
    - RDD에 변환(transformation) 적용 시 새로운 RDD가 생성되고, DAG에서 원래 RDD를 부모로 가리킴
  - 단계 2: 액션이 호출될 때 DAG는 물리적인 실행 계획으로 변환
    - 스케줄러는 액션 당 모든 필요한 RDD를 계산하는 작업(job)을 제출
    - 작업은 하나 이상의 스테이지로 구성되고, 스테이지는 파티션 상에 병렬로 실행되는 태스크들로 구성
    - 파이프라이닝으로 절단되지 않으면 하나의 스테이지는 하나의 RDD에 해당
  - 단계 3: 태스크들은 클러스터 상에 스케줄되고 실행
    - 스테이지들은 순서대로 실행
    - 작업의 최종 스테이지 종료될 때 액션은 완료

2. 스파크 웹 모니터링

스파크 웹 UI

- 스파크 웹 UI는 스파크 작업(job)의 진행과 성능에 관한 정보를 제공
  - 디폴트로 드라이버의 4040 포트를 통해 응용의 수행 중 정보를 표시
    - 같은 호스트에 여러 개의 SparkContext로 실행 중이면 4040, 4041, 4042 등의 연속적인 포트로 표시
  - 응용 시작 전에 spark.eventLog.enabled 를 true로 세팅하고, spark.eventLog.dir 디렉토리 지정하면 이벤트 후에도 웹 UI를 표시
  - 하둡 YARN 클 러스터인 경우 YARN 자원 관리자(Resource Manager, http://master:8088) 를 통해 접근



참고 – 스파크 응용 설정 방법

- Spark-shell/spark-submit의 명령행으로 설정

```
$SPARK_HOME/bin/spark-shell --master yarn
--conf "spark.eventLog.enabled=true"
--conf "spark.eventLog.dir=hdfs:///sparkdata/eventlog"
```

- 스파크의 spark-defaults.conf 에 지정

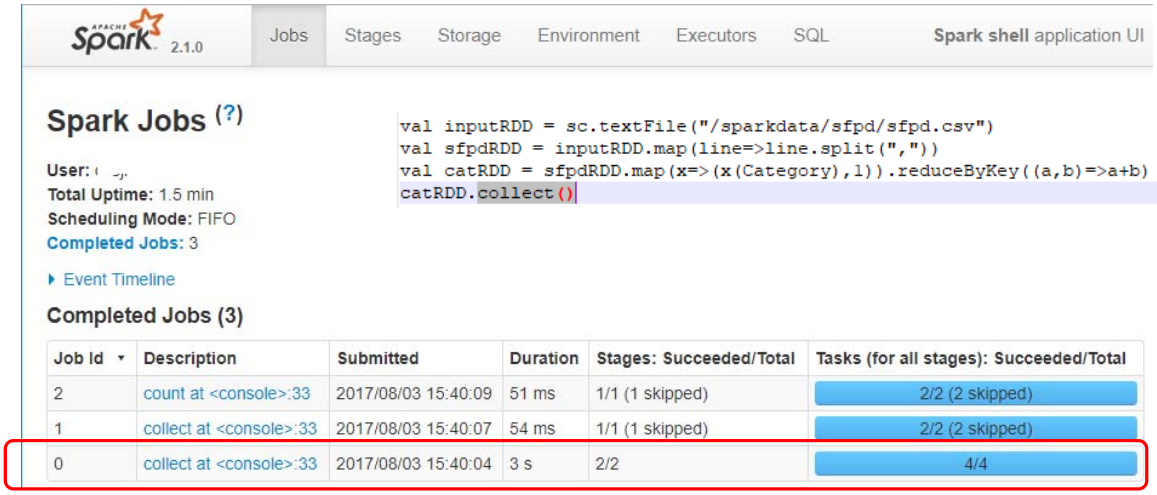
```
spark.eventLog.enabled      true
spark.eventLog.dir          hdfs:///sparkdata/eventlog
```

- 프로그램 상에서 SparkContext를 생성 할 때 SparkConf를 사용하여 설정

```
val conf = new SparkConf().set("spark.eventLog.enabled", "true")
val sc = new SparkContext(conf)
```

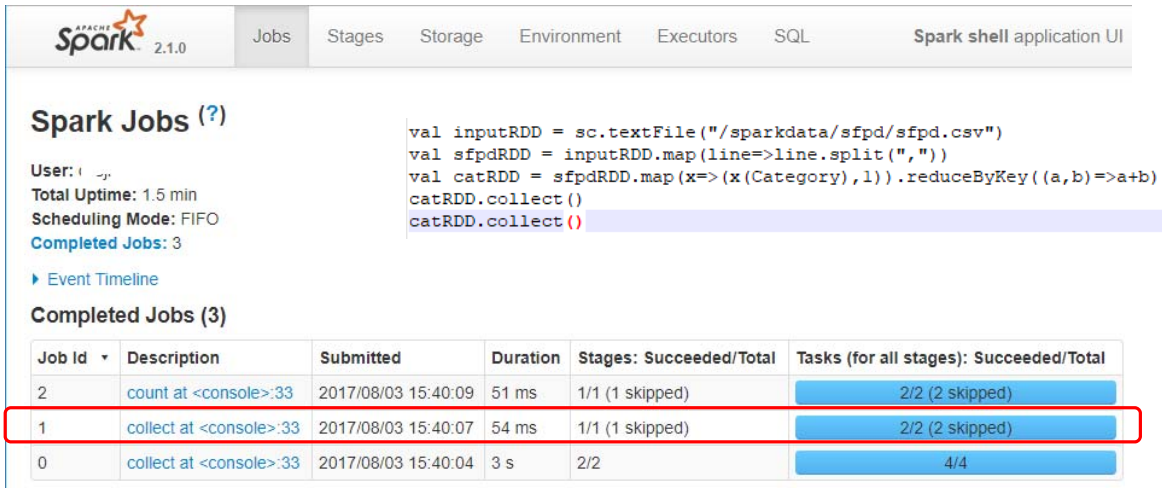

스파크 웹 UI – 작업들 (Jobs) (1)

- Jobs 페이지는 진행 중이거나 최근에 완료된 스파크 작업들의 상세한 실행 정보를 제공
  - 작업의 성능, 실행 중인 작업, 스테이지, 태스크의 진행을 표시
- 아래 예에서 job 0가 collect() 액션에 해당
  - 2개의 스테이지와 4개의 태스크로 구성



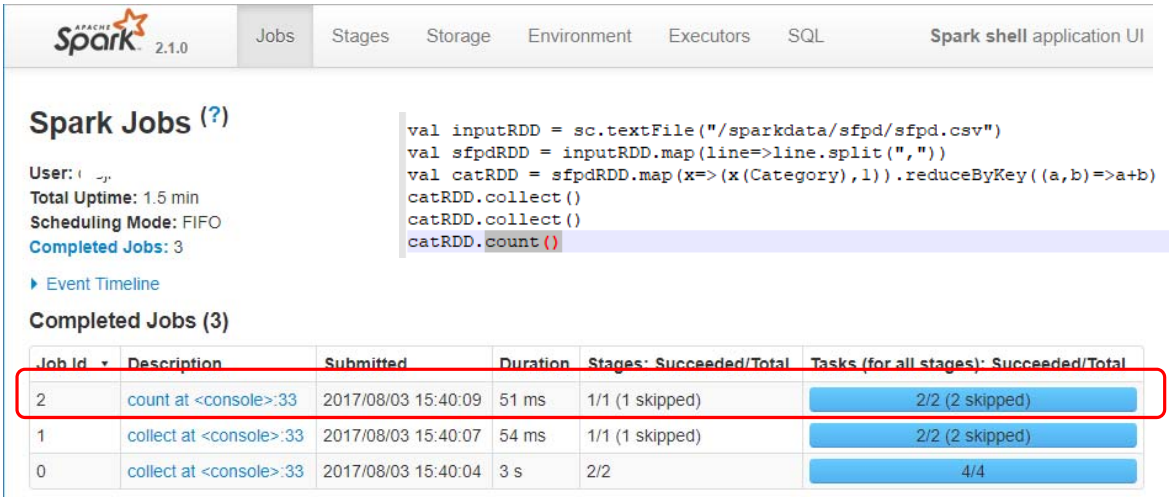
스파크 웹 UI – 작업들 (Jobs) (2)

- job 1은 두번째 collect() 액션에 해당
  - 2개의 태스크를 갖는 1개의 스테이지로 구성
  - 첫 번째 collect() 액션은 3초 수행에 반해, 두번째 collect() 액션은 54ms 수행



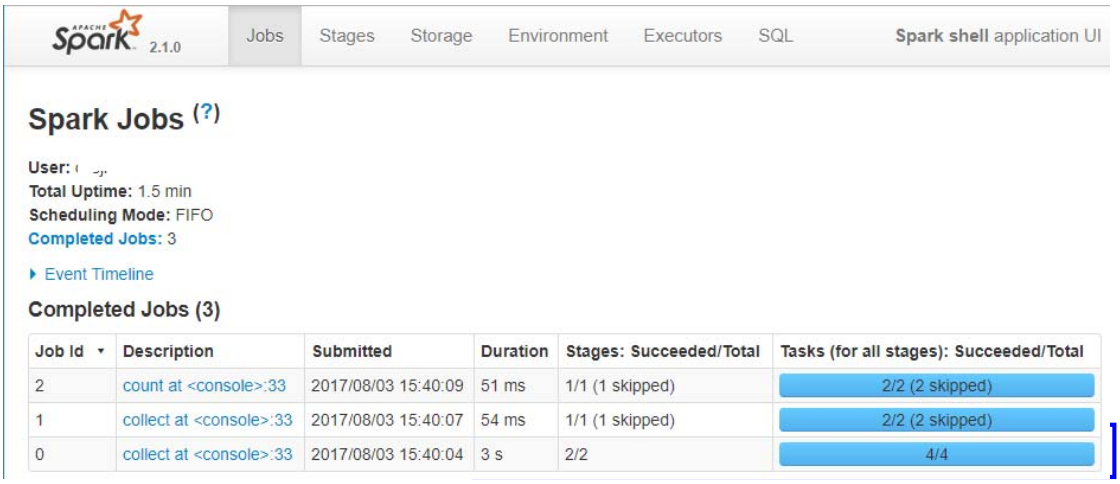
스파크 웹 UI – 작업들 (Jobs) (3)

- job 2는 두번째 collect() 액션에 해당
  - 2개의 태스크를 갖는 1개의 스테이지로 구성



스파크 웹 UI – 생략된 스테이지 (Skipped Stages)

- 처음 collect() 액션은 모든 RDD를 계산한 후 catRDD를 캐싱하여 두 개의 스테이지로 구성
- 두 번째 collect()와 count() 액션은 캐싱된 RDD를 사용하여 스케줄러가 계보를 절단하여(truncate lineage) 스테이지를 생략(skip)하여 job1(54ms)이 job 0 (3s) 보다 빠름

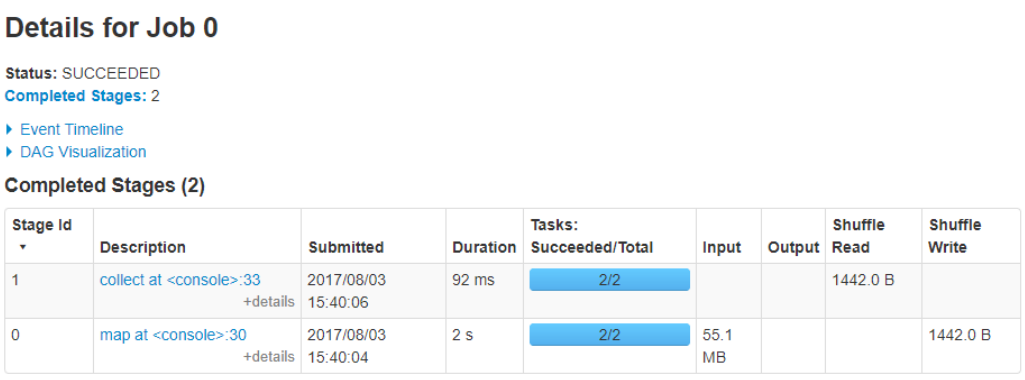


스파크 웹 UI – Job 0 작업의 상세

- 작업의 Description 열을 클릭하면 작업의 상세 페이지 표시

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	count at <console>:33	2017/08/03 15:40:09	51 ms	1/1 (1 skipped)	2/2 (2 skipped)
1	collect at <console>:33	2017/08/03 15:40:07	54 ms	1/1 (1 skipped)	2/2 (2 skipped)
0	collect at <console>:33	2017/08/03 15:40:04	3 s	2/2	4/4

- 실행 중인 작업/스테이지/태스크의 진행 상황 표시
  - Map 변환 2초, collect 액션 92 ms 수행 시간



스파크 웹 UI – Job 1 작업의 상세

- Collect() 액션이 49ms 수행 시간

Details for Job 1

Status: SUCCEEDED
Completed Stages: 1
Skipped Stages: 1

- Event Timeline
- DAG Visualization

Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	collect at <console>:33 +details	2017/08/03 15:40:07	49 ms	2/2			1442.0 B	

Skipped Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	map at <console>:30 +details	Unknown	Unknown	0/2				

스파크 웹 UI – Job 1 작업의 스테이지 상세

- 작업 상세에서 Description을 클릭하면 스테이지 상세 표시

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	collect at <console>:33 +details	2017/08/03 15:40:07	49 ms	2/2			1442.0 B	

- 태스크의 측정값과 진행 상황 상세 표시

Details for Stage 3 (Attempt 0)

Total Time Across All Tasks: 16 ms
Locality Level Summary: Node local: 2
Shuffle Read: 1442.0 B / 77

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	7 ms	7 ms	9 ms	9 ms	9 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	684.0 B / 36	684.0 B / 36	758.0 B / 41	758.0 B / 41	758.0 B / 41

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records
1 stdout stderr	SECOND:41755	50 ms	2	0	0	2	1442.0 B / 77

Tasks (2)

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	4	0	SUCCESS	NODE_LOCAL	1 / SECOND stdout stderr	2017/08/03 15:40:07	7 ms		758.0 B / 41	
1	5	0	SUCCESS	NODE_LOCAL	1 / SECOND stdout stderr	2017/08/03 15:40:07	9 ms		684.0 B / 36	

스파크 웹 UI – 기타

- Storage 페이지는 지속적인 RDD(persisted RDD)에 관한 정보 표시
  - RDD 상에서 persist() 또는 cache() 를 호출하면 RDD는 지속됨
- Environment 페이지는 스파크 응용의 환경 설정 값을 표시
- Executor 페이지는 응용의 실행자 정보 표시

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	192.168.0.110:39677	Active	1	23.9 KB / 2.7 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	SECOND:41755	Active	1	23.9 KB / 384.1 MB	0.0 B	1	0	0	8	8	3 s (75 ms)	57.8 MB	0.0 B	1.4 KB	stdout stderr	Thread Dump
2	DATANODE1:49479	Active	0	0.0 B / 384.1 MB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 3 of 3 entries Previous 1 Next

- SQL 페이지는 SQL 정보 표시

3. 스파크 응용 디버그 및 튜닝

성능 저하 문제 감지

- 다른 태스크들 보다도 수행 시간이 길어서 데이터-병렬 처리의 성능에 문제가 되는 소스 태스크를 스큐(왜곡,skew)라 함

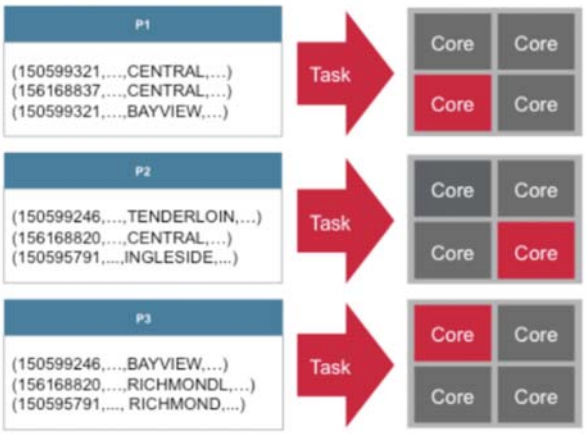
- 스파크 웹 UI를 사용하여 스큐를 모니터링
  - 스테이지 상세 페이지에서 다른 태스크 보다 긴 수행 시간을 갖는 태스크 조사
  - 다른 태스크들 보다 더 많은 데이터를 읽거나 쓰는 태스크 조사
  - 특정 노드들 상에서 태스크 수행이 지연되는가를 조사
  - 읽기, 계산, 쓰기의 각 단계(phase)에서의 수행 시간 조사

성능 저하 이슈

- 성능 저하는 다음과 같은 공통적인 이슈에 기인
  - 병렬성의 수준
  - 셔플 동작 시 사용되는 직렬화 형식
  - 응용의 최적화하는 메모리 관리

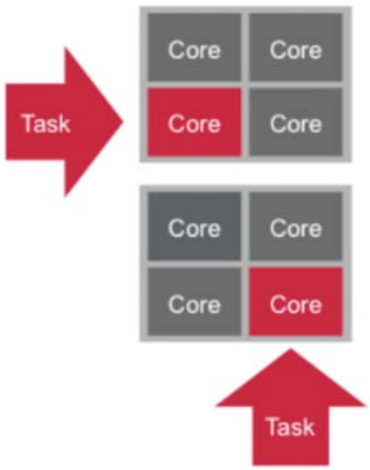
공통적인 성능 이슈 – RDD 파티셔닝 (RDD Partitioning) 복습

- RDD는 데이터의 부분들로 구성된 파티션으로 분할되고, 스케줄러는 각 파티션의 태스크를 생성
  - 각 태스크는 클러스터의 단일 코어에서 실행
  - 스파크는 최상의 병렬성을 기반으로 파티션을 분할



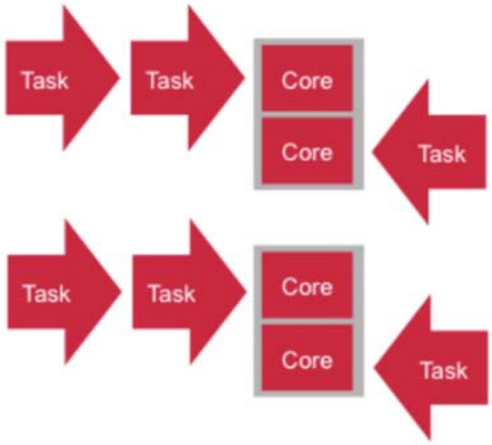
공통적인 성능 이슈 – 병렬성의 수준 (1)

- 병렬성의 수준(level of parallelism)이 성능에 영향
- 너무 적은 병렬성을 가지면 자원이 쉬게 됨



공통적인 성능 이슈 – 병렬성의 수준 (2)

- 너무 큰 병렬성을 가지면 각 파티션과 관련된 오버헤드가 커짐

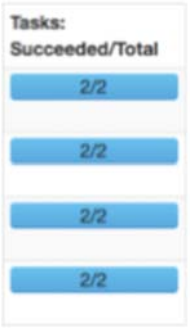


공통적인 성능 이슈 – 병렬성의 수준 튜닝 (1) (Tuning Level of Parallelism)

- 파티션의 개수 확인 방법

1. 스파크 웹 UI에서 스테이지의 수 확인

   Total tasks = number of partitions



2. rdd.patitions.size() 로 확인

공통적인 성능 이슈 – 병렬성의 수준 튜닝 (2)

- 병렬성 수준 튜닝 방법

1. 데이터의 셔플 시 파티션의 수 지정

   예를들어, reduceByKey(func, numPartitions)

2. RDD의 파티션 수를 변경

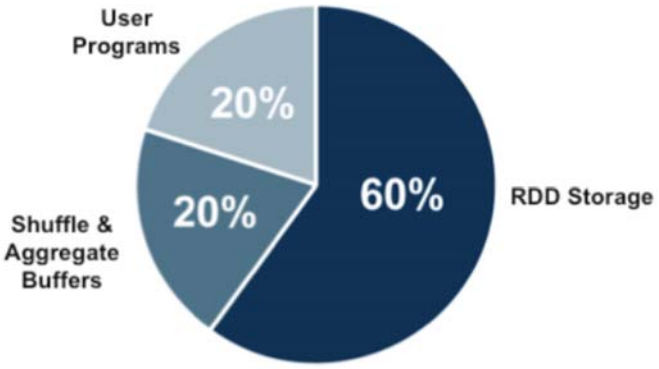
   - repartition()
   - coalesce()

공통적인 성능 이슈 – 직렬화 형식 (Serialization Format)

- 셔플 동작 시 많은 양의 데이터가 네트워크 상에서 전송
- 스파크는 객체들을 이진 형식으로 직렬화하여 전송
  - 종종 직렬화 전송이 성능의 병목이 됨
- 스파크는 디폴트로 Java에 내장된 직렬화 방식(Java built-in serializer) 사용
- Kryo 직렬화 방식이 더 효율적임
  - 빠르고 효율적인 그래프 직렬화 자바 프레임워크
  - https://github.com/EsotericSoftware/kryo

공통적인 성능 이슈 – 메모리 관리

- 스파크의 메모리 사용 내용을 파악하면 메모리 관리 최적화에 도움
- 스파크의 메모리 사용 (디폴트)
  - RDD 저장소 60%
  - 셔플 20%
  - 사용자 프로그램 20%



공통적인 성능 이슈 – 메모리 사용 튜닝

- 메모리 사용 튜닝
  - RDD 저장소, 셔플, 사용자 프로그램의 메모리 영역을 조정
- RDD 저장소
  - cache(), persist() 를 사용하면 RDD 파티션을 메모리 버퍼에 저장
  - persist()는 다양한 옵션을 가짐
    - 디폴트는 persist(MEMORY_ONLY)이며 cache()와 같음
      - 캐시할 메모리가 부족하면 새 RDD는 캐싱하고, 이전 RDD는 제거하고 필요하면 다시 계산
    - persist(MEMORY_AND_DISK)
      - 데이터를 디스크에 저장하고, 필요한 경우 메모리에 적재하여 계산을 줄임
    - persit(MEMORY_ONLY_SER)
      - 직렬화된 형식으로 객체를 캐싱
      - 원본 객체 캐싱보다 더 느릴 수 있으나 가비지 컬렉션(garbage collection)을 없애 시간을 줄임

로그 (Log)

- 스파크의 로깅 시스템은 log4j에 기반
  - Log4j 는 자바 기반 로깅 유틸리티
    - https://logging.apache.org/log4j
  - 로깅 레벨이나 출력을 조정 가능
  - log4j 설정 예는 스파크 conf 디렉토리에 제공
  - 로그 파일들의 위치는 배포 모드에 따라 다름

Deployment Mode	Location
Spark Standalone	work/ directory of distribution on each worker
Mesos	work/ directory of Mesos slave
YARN	Use YARN log collection tool

성능 튜닝 팁

- 많은 데이터의 셔플을 피함
  - 집계(aggregation)가 가능하면 aggregateByKey를 사용
  - 큰 데이터 상의 groupByKey는 많은 셔플을 유발하므로, 가능하면 reduceByKey를 사용
    - combineByKey, foldByKey 도 사용할 수 있음
- RDD의 모든 요소를 드라이버에 복사하지 않음
  - collect() 액션은 RDD의 모든 요소들을 단일 드라이버 프로그램으로 복사하므로, RDD가 아주 크면 드라이버가 고장나 멈출 수 있음
    - countByKey, countByValue, collectAsMap도 마찬가지임
- 가능하면 filter 변환을 하여 작은 데이터 세트를 유지
- 쉬고 있는 태스크가 많으면 (~10k) coalesce() 사용하여 통합
- 클러스터의 모든 슬롯을 사용하고 있지 않다면 repartition

과제

- 자신만의 데이터를 사용하여 앞에서 배운 스파크를 적용하고 실행
  - 텀 프로젝트와 관련하여 연관성이 있는 데이터

참고 자료

- MapR Academy DEV 361 Build and Monitor Apache Spark Applications
  - http://learn.mapr.com/dev-360-apache-spark-essentials
  - Lesson 6: Monitor Apache Spark Applications
- Spark Monitoring and Instrumentation
  - https://spark.apache.org/docs/latest/monitoring.html
- Configuring Spark Applications
  - https://www.cloudera.com/documentation/enterprise/5-7-x/topics/spark_applications_configuring.html