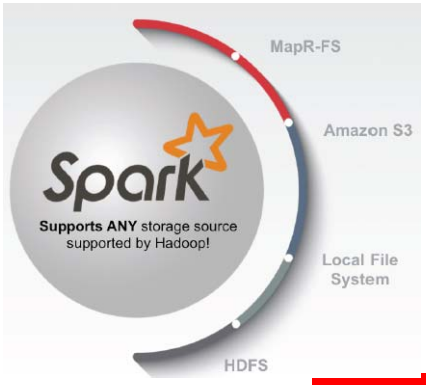


스파크 응용 모니터링

순천향대학교 컴퓨터공학과

이 상 정

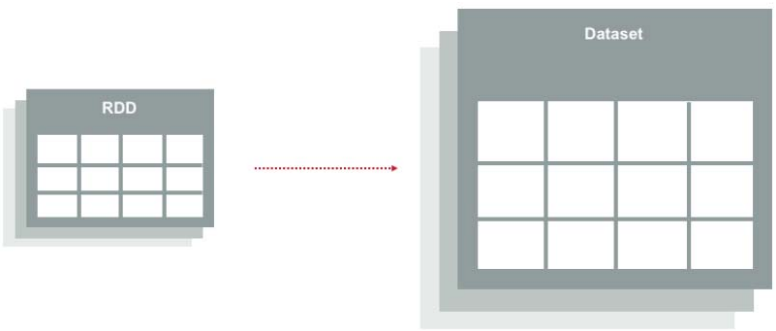
학습 내용

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

1. 스파크 실행 모델

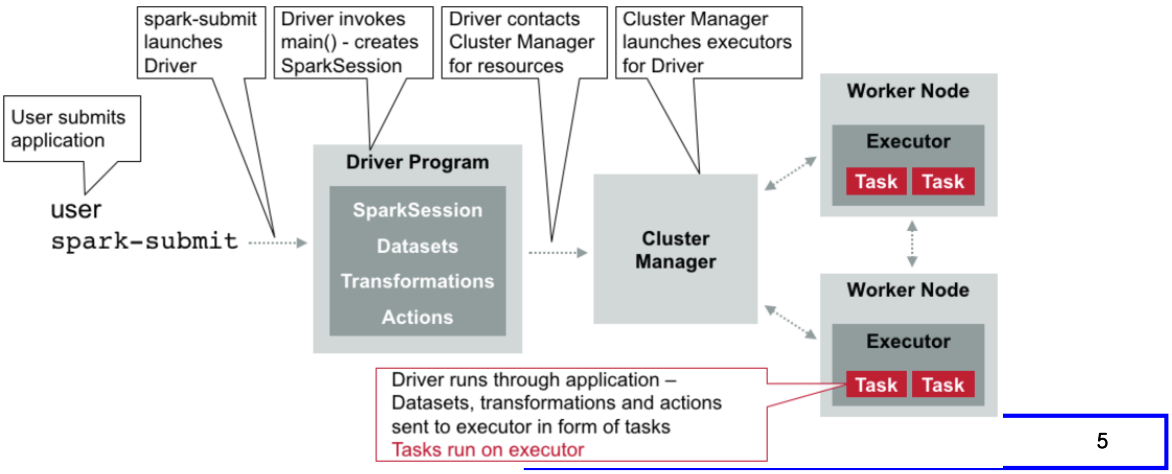
RDD (Resillient Distributed Dataset)

- RDD (Resillient Distributed Dataset)
  - 스파크의 기본 빌딩 블럭
    - 분산 객체 컬렉션, API
    - 병렬로 처리할 수 있는 파티셔닝된 레코드의 컬렉션
      - 레코드는 자바, 스칼라, 파이썬 등의 객체
  - 데이터세트 연산도 하부 구조의 RDD를 사용하여 계산



스파크 실행 컴포넌트 (1)

- 태스크(Task)
  - 분할된 데이터 단위(파티션)에 적용되는 연산
    - 파티션은 클러스터의 코어에서 병렬 실행되는 데이터 단위
  - 실행자(executor)에서 실행할 데이터의 블록과 다수의 변환(transformation)의 조합



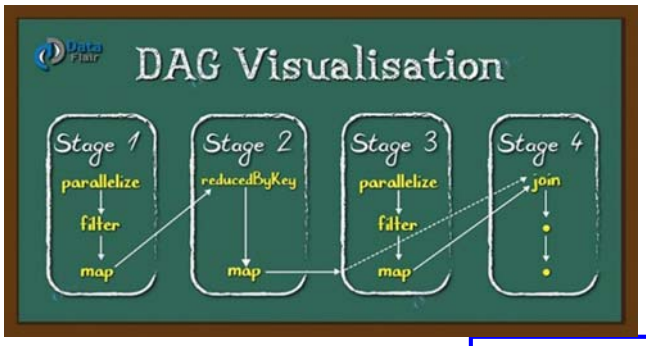
스파크 실행 컴포넌트 (2)

- 스테이지(Stage)
  - 다 수의 노드(머신)에서 병렬로 동일한 연산을 수행하는 태스크들의 그룹
  - 한 스테이지 는 하나의 RDD
    - 파이프라이닝으로 통합되기 전의 RDD
- 잡(Job)
  - RDD를 계산하는데 필요한 작업
  - 특정 액션을 수행하는 스테이지들의 집합
    - 일반적으로 액션 당 하나의 잡이 생성
- 셔플(shuffle)
  - 스테이지들 간의 네트워크을 통한 데이터 전달

스파크 실행 컴포넌트 (3)

- 파이프라이닝(pipelining)
  - 노드 간의 데이터의 이동(셔플) 없이 각 노드가 데이터를 직접 공급할 수 있는 스테이지(RDD)들을 통합하여 단일 스테이지(RDD)로 구성하는 작업

- DAG(Directed Acyclic Graph)
  - RDD 상에서 적용되는 연산을 표현한 논리적인 그래프

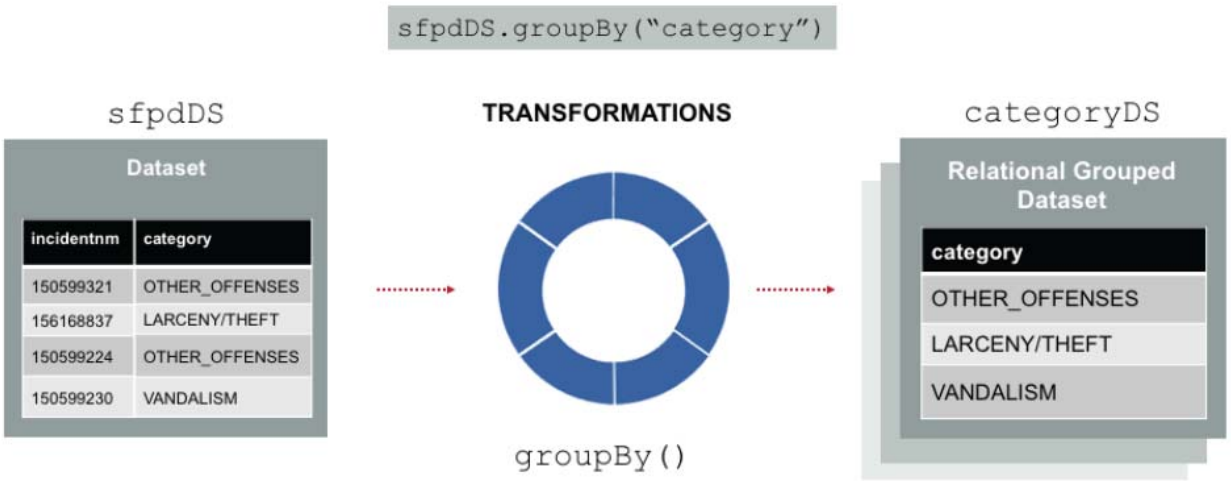


스파크 실행 단계 (Spark Execution Phases)

- 단계 1: 논리적 계획(Logical Plan)
  - 사용자 코드로 부터 DAG를 정의
- 단계 2: 물리적 실행 계획 (Physical Execution Plan)
  - 액션이 호출될 때 DAG는 물리적인 실행 계획으로 변환
  - 물리적 계획에서는 실행을 위한 계산 및 메모리 자원 등을 할당
- 단계 3: 스케줄 (Schedule)
  - 태스크들은 클러스터 상에 스케줄되고 실행
  - 스테이지들은 순서대로 실행
  - 잡(job)의 최종 스테이지가 종료될 때 액션은 완료

PHASE 1	PHASE 2	PHASE 3
Create the Logical Plan: User code defines the DAG	Actions responsible for translating DAG into physical execution plan	Tasks scheduled and executed on cluster

SFPD 데이터세트 연산 예



SFPD 예의 논리적 계획

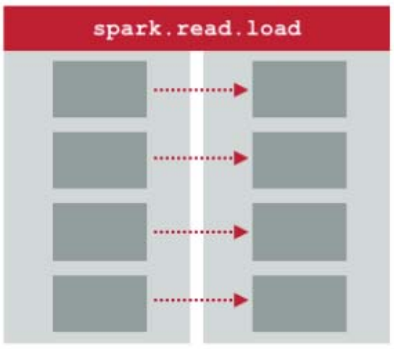
- SFPD 예의 사용자 프로그램이 물리적인 실행 단위로 변환되는 모델을 소개

```
1. val sfpdDF = spark.read.format("csv").option("inferSchema",
   true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category",
   "description", "dayofweek", "date", "time", "pddistrict",
   "resolution", "address", "X", "Y", "pdid")

2. val categoryDS = sfpdDF.groupBy("category")

3. val categoryCountDS = categoryDS.count()

4. categoryCountDS.collect()
```

SFPD 예의 논리적 계획 – 데이터 적재

- sfpd.csv 파일로 부터 sfpdDF 데이터프레임을 생성

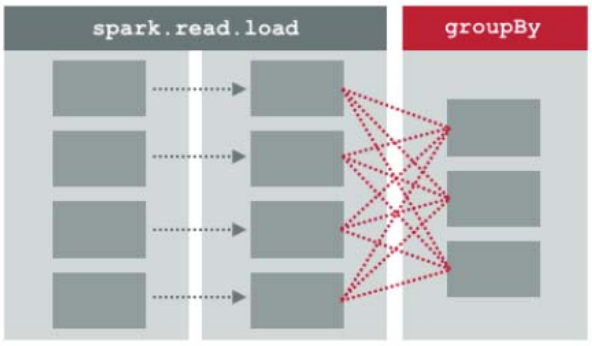
1.
```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict",
"resolution", "address", "X", "Y", "pdid")
```

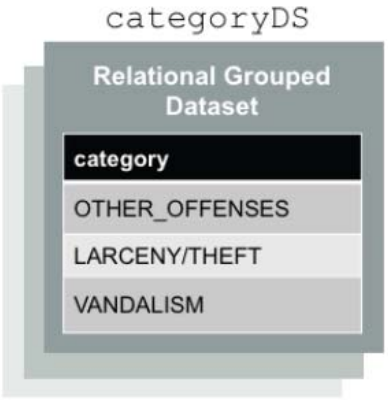


SFPD 예의 논리적 계획 – groupBy 변환

- category 열의 값에 따라 그룹화하여 categoryDS 관계형 그룹 데이터세트(Relational Grouped Dataset) 생성

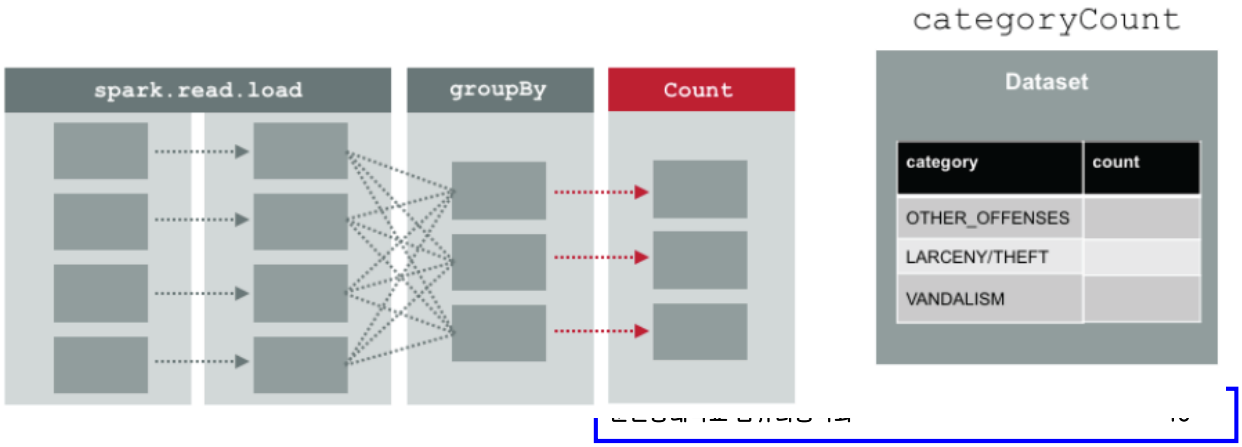
2.
```
val categoryDS = sfpdDF.groupBy("category")
```





SFPD 예의 논리적 계획 – count 변환

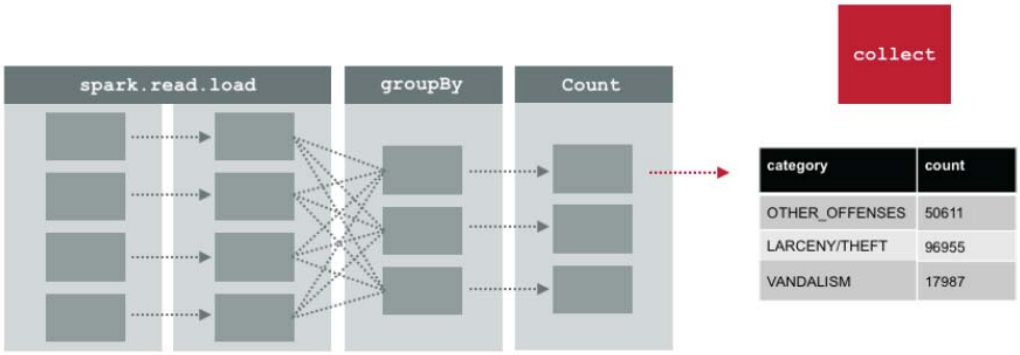
- **count 액션**을 수행
  - count 연산은 변환으로 취급되어 categoryCount 라는 새로운 데이터세트를 생성
    - 따라서 아직 어떤 액션도 수행되지 않음
  - **데이터세트 객체들의 DAG**가 정의
    - 각 데이터세트는 각가 부모의 데이터세트들을 포인터를 유지



SFPD 예의 논리적 계획 – collect 액션

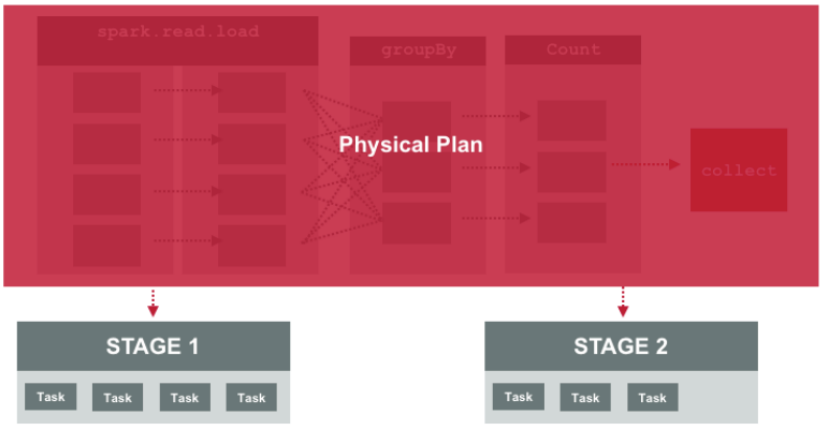
- **collect 액션**이 실행되면 실제 계산을 수행
  - 스파크 스케줄러가 데이터세트를 계산하는 물리적인 실행 계획을 생성
    - 모든 부모 데이터세트들의 물리적 계획도 생성

```
4. categoryCount.collect()
```



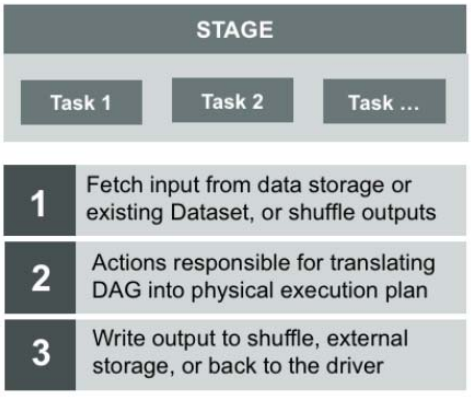
물리적 실행 계획

- 액션이 호출될 때 DAG는 물리적 계획으로 변환
  - 스케줄러는필요한 데이터세트를 계산하는 각 액션 당 잡(job)을 제출
  - 잡(job)은 하나 이상의 스테이지들로 구성되고, 스테이지는 파티션 상에 병렬로 실행되는 태스크들로 구성
  - 각 스테이지들은 순서대로 처리되어 각 태스크가 스케줄되고, 클러스터 상에서 실행



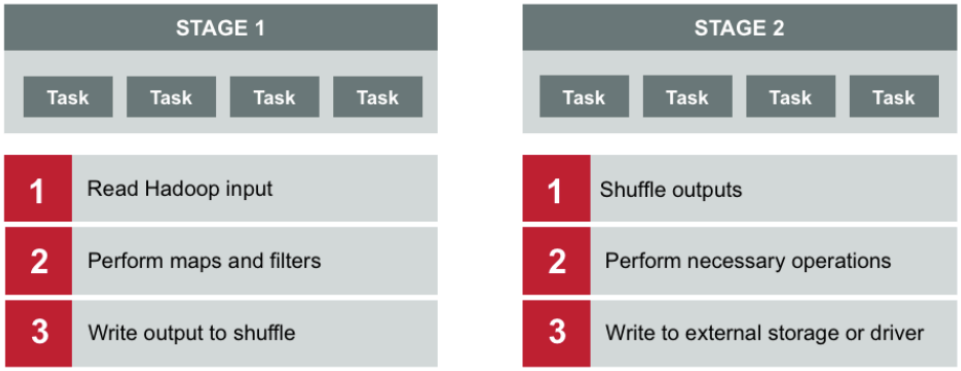
물리적 실행 계획 – 스테이지와 태스크 (1)

- 스테이지는 데이터세트의 RDD의 각 파티션에 대한 태스크를 가짐
  - 스테이지는 특정 파티션의 데이터 상에서 같은 일을 하는 태스크를 수행
  - 일반적으로 스테이지의 수는 DAG의 RDD에 기반한 데이터세트의 수와 일치
- 각 태스크는 아래의 공통 단계를 수행
  1. 데이터 저장소 또는 기존 데이터세트로부터 입력과 셔플의 결과를 읽음
  2. 필요한 데이터세트를 계산하기 위해 필요한 변환과 액션을 수행
  3. 셔플, 외부 저장소, 드라이버(예를들어 count, collect) 등으로 출력



물리적 실행 계획 – 스테이지와 태스크 (2)

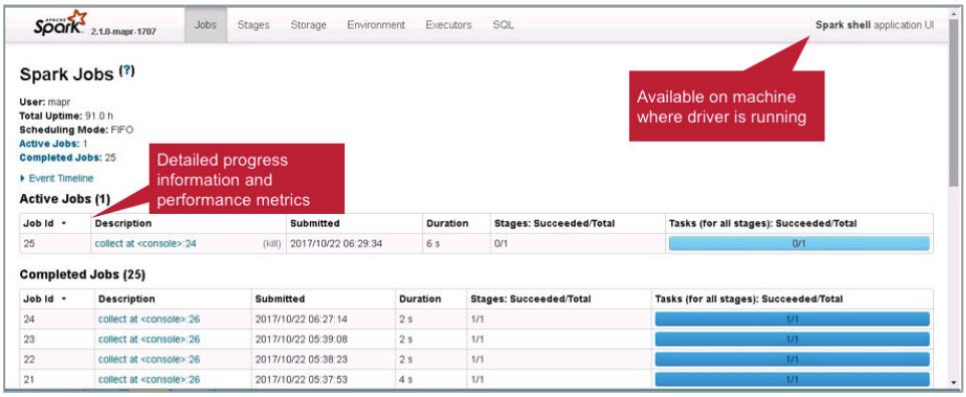
- 예를 들어 스테이지 1에서는 데이터를 적재하고, 베이스 데이터세트를 생성
  - 스테이지 내의 모든 태스크들은 같은 일을 수행
  - 4개의 태스크들이 각각 3개의 스텝을 병렬 실행
    - 스텝 1은 데이터를 읽고, 스텝 2에서 map과 filter 변환을 수행하고, 스텝 3에서 셔플하기 위해 출력
- 스테이지 2에서도 각 태스크들이 셔플 후 연산하고 출력



2. 스파크 웹 모니터링

스파크 웹 UI

- 스파크 웹 UI는 스파크 작업(job)의 진행과 성능에 관한 정보를 제공
  - 디폴트로 드라이버의 4040 포트를 통해 실행 중인 응용의 정보를 표시
    - 같은 호스트에 여러 개의 SparkSession이 실행 중이면 4040, 4041, 4042 등의 연속적인 포트로 표시
  - 하둡 YARN 클 러스터인 경우 YARN 자원 관리자(Resource Manager, http://master:8088) 를 통해 접근

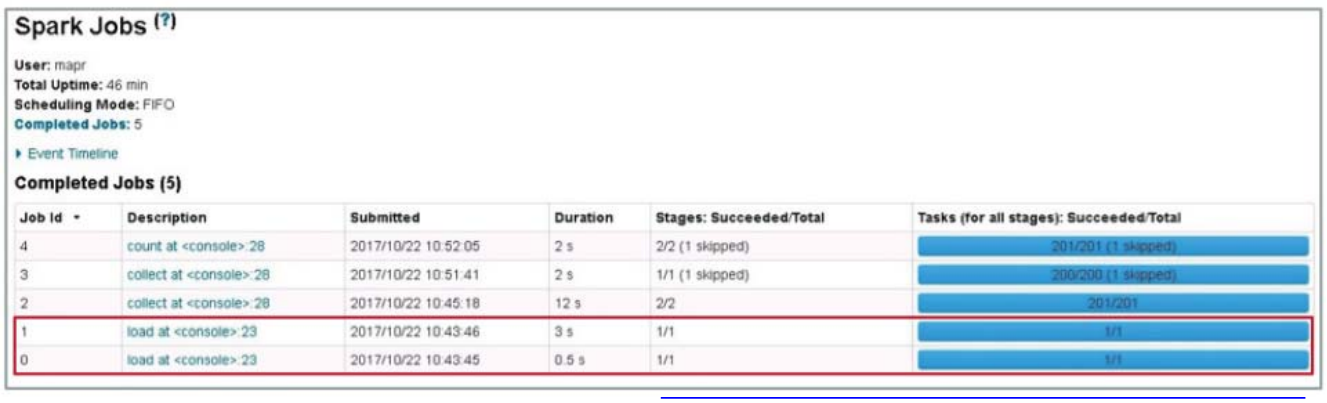


http://<driver-node>:4040

스파크 웹 UI – 잡 페이지 (Jobs Page) (1)

- 잡 페이지는 최근에 완료 된 스파크 잡에 대한 상세 실행 정보를 표시
  - 아래 예에서 Job Id 0는 처음 실행된 잡으로 첫번째 코드의 spark.read.load 메서드에 해당
    - 단일 태스크로 구성

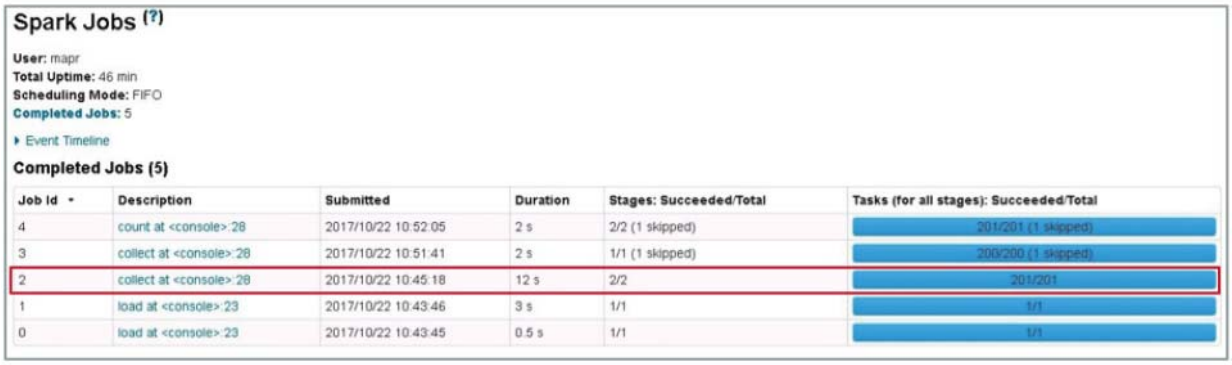
```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
```



스파크 웹 UI – 잡 페이지 (2)

- Job Id 1은 IncidentCountDF 생성에 해당
- Job Id 2는 collect 액션에 해당
  - 2개의 스테이지, 201개의 태스크로 구성
  - 실행 시간은 12초

```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
incidentCountDF.cache()
incidentCountDF.collect
```



스파크 웹 UI – 잡 페이지 (3)

- Job Id 3는 두번째 collect 액션에 해당
  - 1개의 스테이지, 200개의 태스크들로 구성
  - 실행 시간은 2초
  - 캐싱된 데이터프레임을 사용하여 스테이지와 태스크가 줄고(skip), Id 2 보다도 실행시간도 빠름

```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
incidentCountDF.cache()
incidentCountDF.collect
incidentCountDF.collect
```

스파크 웹 UI – 잡 페이지 (4)

- Job Id 4는 count 액션에 해당
  - 2개의 스테이지, 201개의 태스크들로 구성
  - 실행 시간은 2초
  - 캐싱된 데이터프레임을 사용

```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
incidentCountDF.cache()
incidentCountDF.collect
incidentCountDF.collect
incidentCountDF.count
```

Spark Jobs (?)

User: mapr
Total Uptime: 46 min
Scheduling Mode: FIFO
Completed Jobs: 5

▸ Event Timeline

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at <console>:28	2017/10/22 10:52:05	2 s	2/2 (1 skipped)	201/201 (1 skipped)
3	collect at <console>:28	2017/10/22 10:51:41	2 s	1/1 (1 skipped)	200/200 (1 skipped)
2	collect at <console>:28	2017/10/22 10:45:18	12 s	2/2	201/201
1	load at <console>:23	2017/10/22 10:43:46	3 s	1/1	1/1
0	load at <console>:23	2017/10/22 10:43:45	0.5 s	1/1	1/1

스파크 웹 UI – 잡 상세 페이지 (1)

- 잡 페이지에서 Description 열의 링크를 누르면 잡 상세 페이지 (Job Details)가 표시
  - 잡의 스테이지 별로 진행 상황등을 표시
  - 아래는 Job Id 2의 상세 페이지

Details for Job 2

Status: SUCCEEDED
Completed Stages: 2

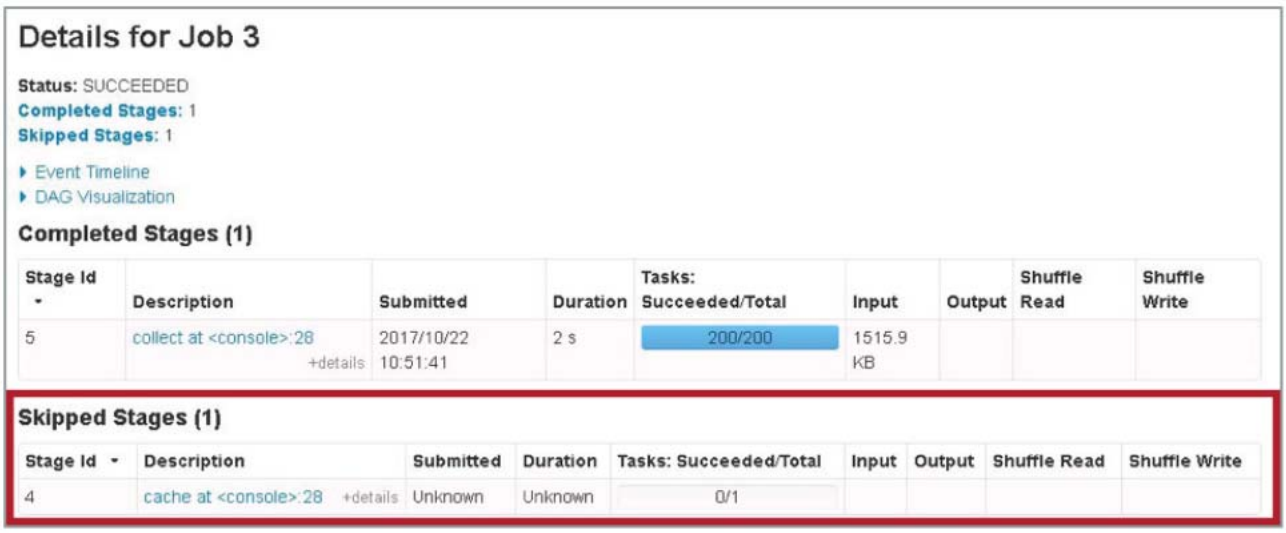
▸ Event Timeline
▸ DAG Visualization

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	nput	Output	Shuffle Read	Shuffle Write
3	collect at <console>:28 +details	2017/10/22 10:45:24	6 s	200/200			2.0 MB	
2	cache at <console>:28 +details	2017/10/22 10:45:18	6 s	1/1	55.1 MB			2.0 MB

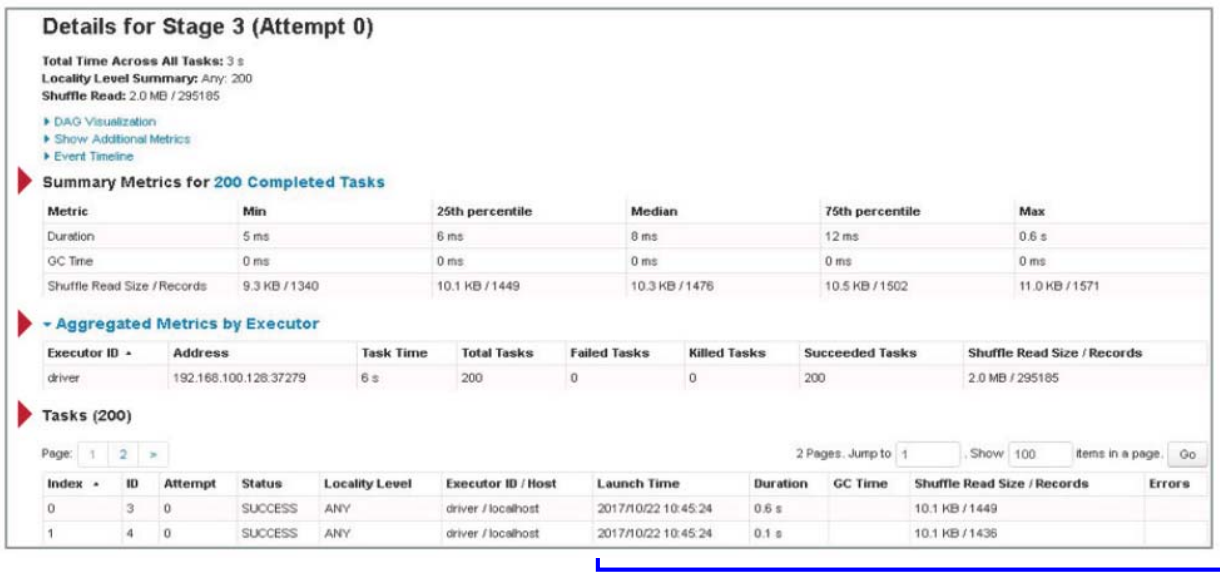
스파크 웹 UI – 잡 상세 페이지 (2)

- Job Id 3의 상세 페이지
  - 캐싱된 데이터프레임을 사용하여 스킵된 스테이지가 표시



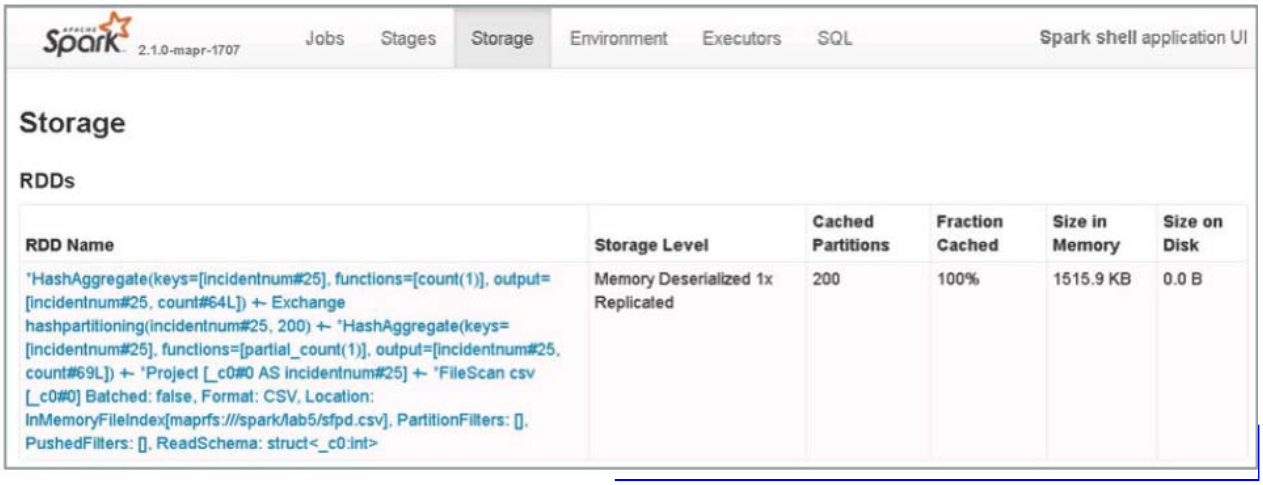
스파크 웹 UI – 스테이지 상세 페이지

- 잡 상세페이지에서 Description 열의 링크를 누르면 스테이지 상세 페이지 (Stage Details)가 표시
  - 또는 메인 페이지에서 Stages 메뉴로도 표시
  - 스테이지의 완료된 태스크의 요약 통계와 태스크 별로 진행 상황 표시



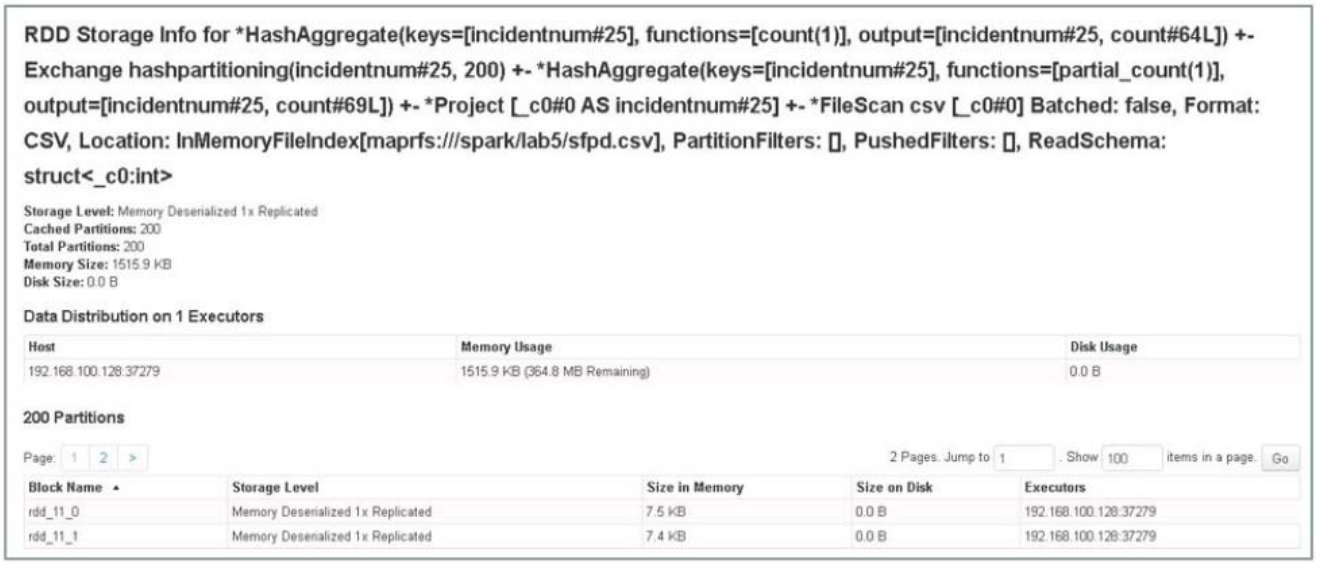
스파크 웹 UI – 스토리지 페이지 (1)

- 스토리지 페이지 (Storage Page)는 저장된 데이터세트 (persisted Dataset)에 대한 정보 제공
  - Persist 또는 Cache 된 경우 저장된 데이터세트이라고 함
  - 아래 페이지는 캐싱된 데이터세트의 RDD의 부분을 표시
    - 캐싱된 데이터의 크기 및 저장 매체 등이 표시



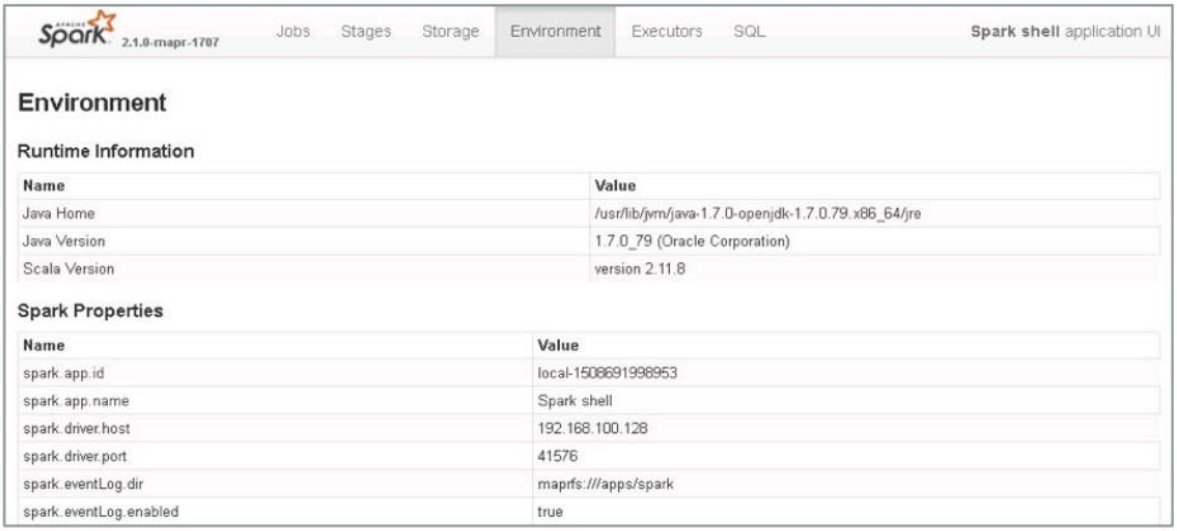
스파크 웹 UI – 스토리지 페이지 (2)

- 스토리지 페이지에서 RDD Name 열을 클릭하면 데이터세트에 대한 상세 정보 표시
  - 데이터세트의 RDD 파티션들에 대한 상세 정보 표시



스파크 웹 UI – 환경 페이지

- 환경 페이지(Environment Page)는 스파크 응용 환경에서 활성화된 프로퍼티를 표시
  - spark-defaults.conf, SparkSession, 명령행 라인 등에서 설정된 속성 등을 표시



참고 – 스파크 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://master:9000/sparkdata/history"
```

- 스파크의 spark-defaults.conf 에 지정

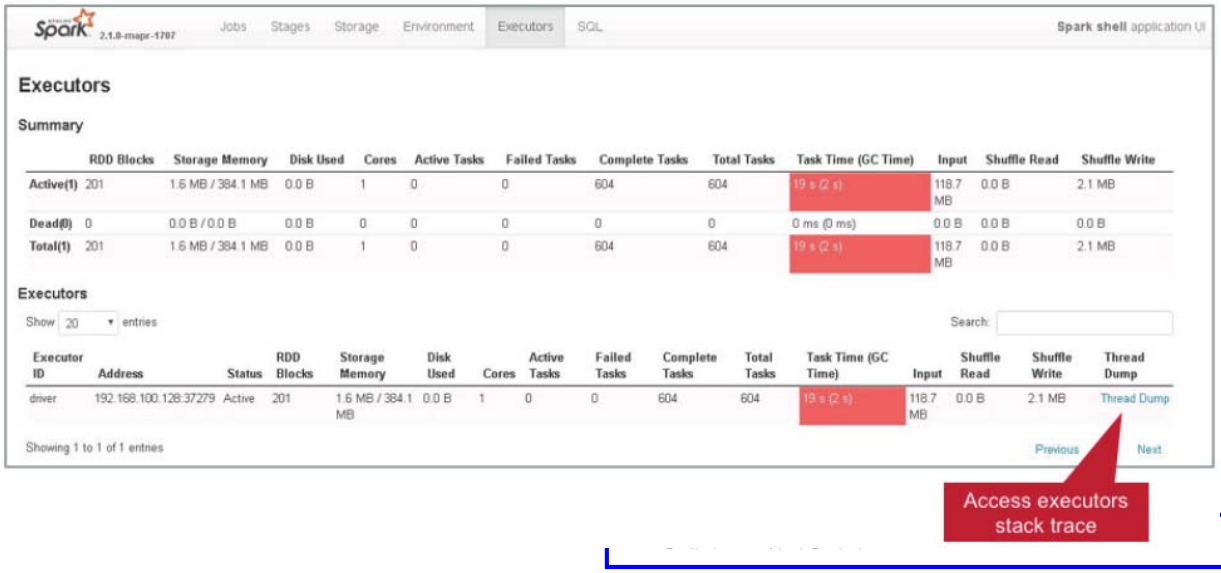
```
spark.eventLog.enabled      true
spark.eventLog.dir          hdfs://master:9000/sparkdata/history
```

- 프로그램 상에서 SparkSession를 생성 지정

```
val spark = SparkSession.builder.appName("sfpdApp")
                .config(" spark.eventLog.enabled", "true")
                .getOrCreate()
```

스파크 웹 UI – 실행자 페이지

- 실행자 페이지(Executor Page)는 활성화된 실행자들을 표시
  - 각 실행자의 실행 및 저장 등에 관한 성능 측정값 표시

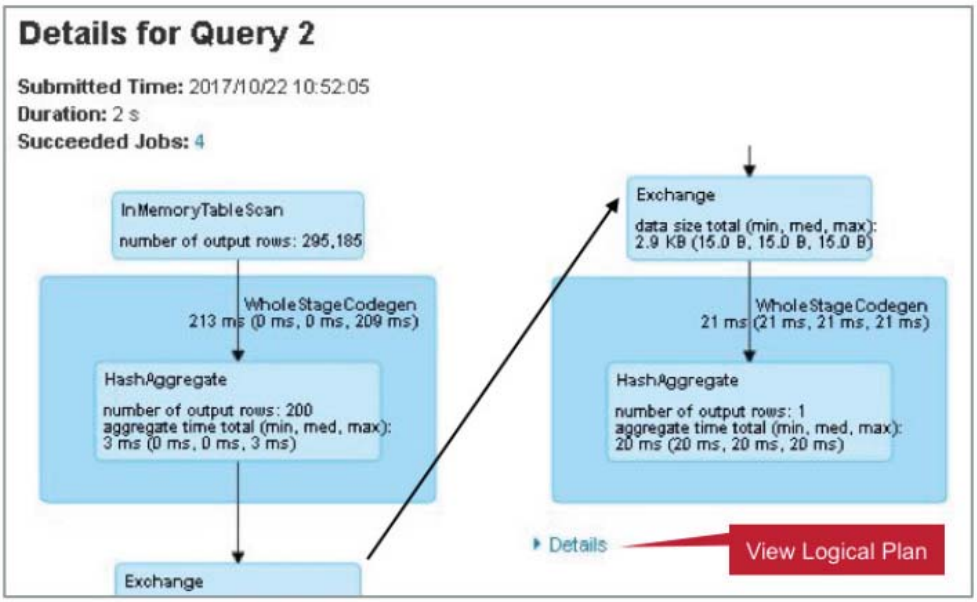


스파크 웹 UI – SQL 페이지 (1)

- SQL 페이지는 완료된 질의에 대한 정보 표시
  - Description 열을 링크를 누르면 실행된 질의의 상세 정보와 논리적 계획 등이 표시

스파크 웹 UI – SQL 페이지 (2)

- 질의 상세 페이지
  - Details 링크를 누르면 상세한 논리적 계획이 표시



스파크 웹 UI 예

```
// 클래스 임포트
import spark.implicits._

// 데이터프레임 정의
val sfpdDF = spark.read.format("csv")
    .option("inferSchema",true).load("/sparkdata/sfpd/sfpd.csv").toDF("incident
    num", "category", "description", "dayofweek", "date", "time", "pddistrict",
    "resolution", "address", "X", "Y", "pdid")

// 베이스 데이터프레임에 변환 적용하여 사건해결 데이터프레임 생성
val resolutionDF = sfpdDF.select("resolution").distinct
// 사건해결 물리적 실행 계획 출력
resolutionDF.explain()
// 캐싱 및 카운트 액션
resolutionDF.cache()
resolutionDF.count     // 총 사건해결 건 수
resolutionDF.count     // 총 사건해결 건 수, 캐싱된 데이터프레임 사용
```

스파크 웹 UI 실행 예 (1)

```
// 클래스 임포트
import spark.implicits._

// 데이터프레임 정의
val sfpdDF = spark.read.format("csv") .option("inferSchema",true).load("/sparkdata/sfpd/sfpd.csv").toDF("incidentnum", "category",
    "description", "dayofweek", "date", "time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
```

FINISHED

```
import spark.implicits._
sfpdDF: org.apache.spark.sql.DataFrame = [incidentnum: int, category: string ... 10 more fields]
```

Took 54 sec. Last updated by admin at June 13 2020, 1:16:25 PM.

```
// 베이스 데이터프레임에 변환 적용하여 사건해결 데이터프레임 생성
val resolutionDF = sfpdDF.select("resolution").distinct
// 사건해결 물리적 실행 계획 출력
//resolutionDF.rdd.toDebugString
resolutionDF.explain()
```

FINISHED

```
== Physical Plan ==
*(2) HashAggregate(keys=[resolution#41], functions=[])
+- Exchange hashpartitioning(resolution#41, 200)
   +- *(1) HashAggregate(keys=[resolution#41], functions=[])
      +- *(1) Project [_c7#17 AS resolution#41]
         +- *(1) FileScan csv [_c7#17] Batched: false, Format: CSV, Location: InMemoryFileIndex[hdfs://master:9000/sparkdata/sfpd/sfpd.cs
v], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c7:string>
resolutionDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [resolution: string]
```

Took 1 sec. Last updated by admin at June 13 2020, 1:16:26 PM.

스파크 웹 UI 실행 예 (2)

```
// 캐싱 및 카운트 액션
resolutionDF.cache()
resolutionDF.count      // 총 사건해결 건 수
```

FINISHED

```
res2: Long = 17
```

Took 9 sec. Last updated by admin at June 13 2020, 1:16:36 PM.

```
resolutionDF.count      // 총 사건해결 건 수, 캐싱된 데이터프레임 사용
```

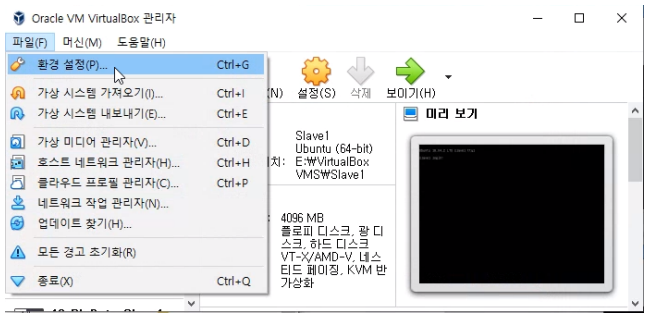
FINISHED

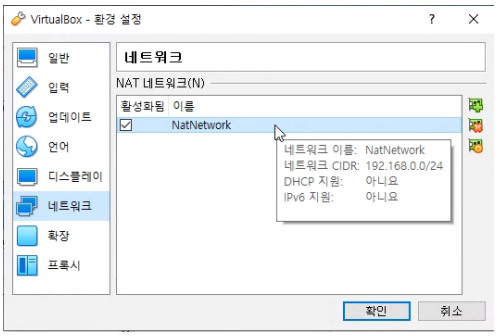
```
res3: Long = 17
```

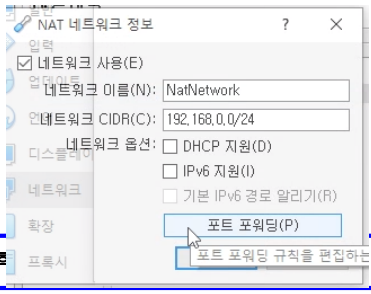
Took 3 sec. Last updated by admin at June 13 2020, 1:16:39 PM.

스파크 웹 UI – NAT 설정 (1)

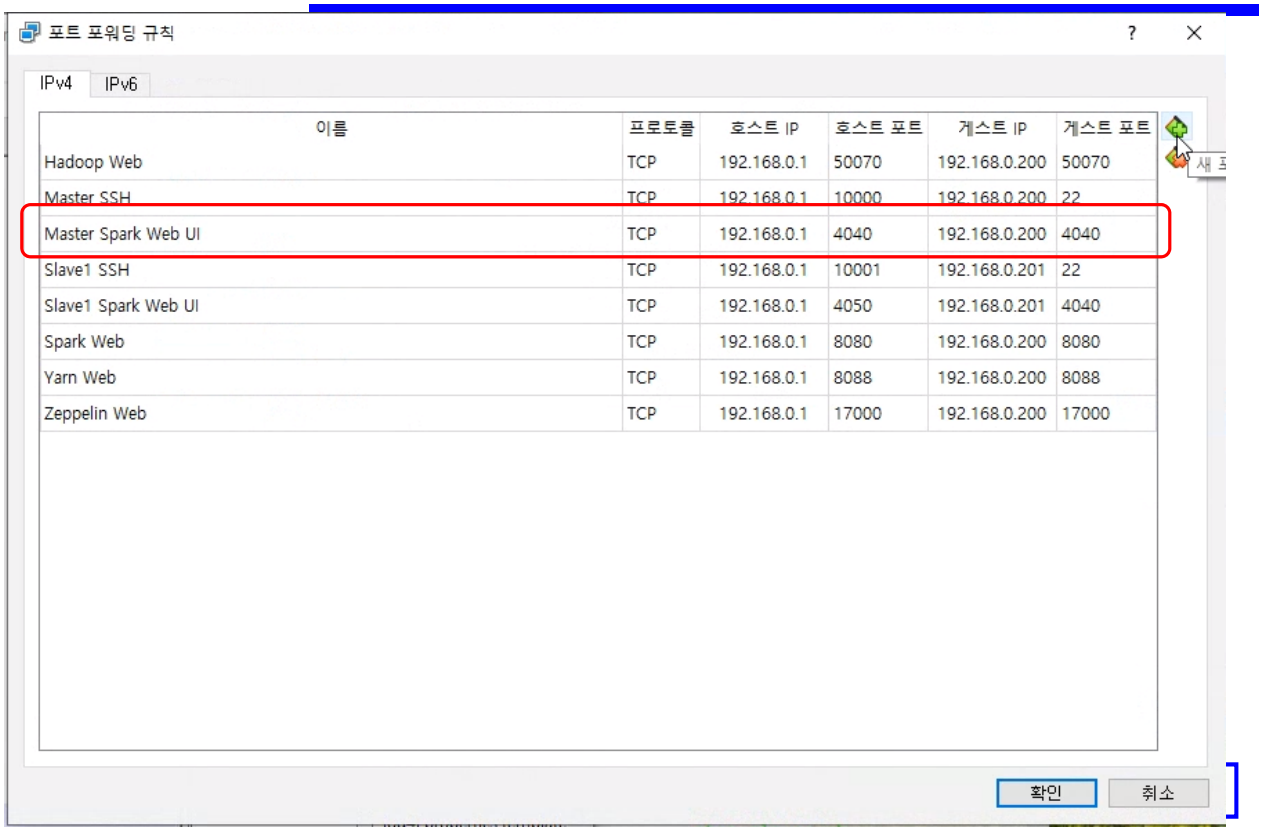
- NAT 내부의 스파크 드라이버의 웹 UI포트 4040포트 접근하기 위해서 게이트웨이 포트포워딩(port forwarding) 설정
  - 제플린은 마스터(192.168.0.200)에 설치되어 실행







스파크 웹 UI – NAT 설정 (2)



제플린 스파크 설정 수정

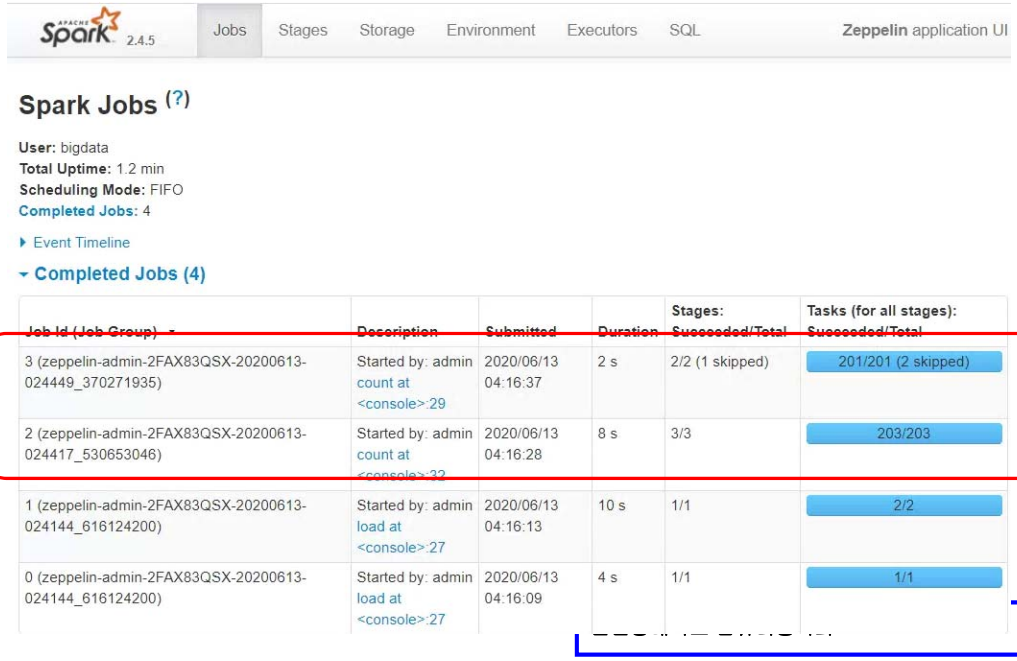
- 제플린에서 스파크 실행 시에는 스파크 인터프리터 master 설정 라애와 같이 수정해야 스파크 웹 UI 모니터링 가능
  - Interpretes에서 spark 검색하여 master 속성을 yarn-cluster에서 spark://master:7077 로 수정
    - 수정하면 yarn 웹 8088 포트에서는 스파크 잡 모니터링 안됨
  - 제플린에서 스파크 실행 후 스파크 웹 UI 4040 포트 모니터링

Properties

name	value
args	
master	spark://master:7077
spark.app.name	Zeppelin

스파크 웹 UI – 잡 페이지 예

- 4개의 잡 수행
  - Job ID 2, 3이 두 개의 count 액션 수행 잡



스파크 웹 UI – 첫번째 count

❑ Job ID 2의 잡 상세 페이지

- 3개의 스테이지로 구성
- 총 8.3초 수행

Details for Job 2

Status: SUCCEEDED
Job Group: zeppelin-admin-2FAX83QSX-20200613-024417_530653046
Completed Stages: 3

▸ Event Timeline
▸ DAG Visualization

▾ **Completed Stages (3)**

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	Started by: admin count at <console>:32 +details	2020/06/13 04:16:36	0.3 s	1/1			11.0 KB	
3	Started by: admin count at <console>:32 +details	2020/06/13 04:16:31	5 s	200/200			2.7 KB	11.0 KB
2	Started by: admin count at <console>:32 +details	2020/06/13 04:16:28	3 s	2/2	54.3 MB			2.7 KB

스파크 잡이 실행하는 도중에 셔플을 수행하면 기본적으로 200개의 셔플 파티션을 생성

spark.sql.shuffle.partitions 속성을 지정하여 원하는 값으로 변경

스파크 웹 UI – 두번째 count

❑ Job ID 3의 잡 상세 페이지

- 3개의 스테이지 중 1개의 스테이지 스킵
  - 캐싱된 데이터프레임 사용
- 총 2.2초 수행

Details for Job 3

Status: SUCCEEDED
Job Group: zeppelin-admin-2FAX83QSX-20200613-024449_370271935
Completed Stages: 2
Skipped Stages: 1

▸ Event Timeline
▸ DAG Visualization

▾ **Completed Stages (2)**

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	Started by: admin count at <console>:29 +details	2020/06/13 04:16:39	0.2 s	1/1			11.0 KB	
6	Started by: admin count at <console>:29 +details	2020/06/13 04:16:37	2 s	200/200	9.1 KB			11.0 KB

▾ **Skipped Stages (1)**

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	count at <console>:32 +details	Unknown	Unknown	0/2				

스파크 히스토리 서비스

- 스파크 응용의 실행 후에도 웹 UI 표시하려면 스파크 히스토리 서비스(history service)를 설정하여 사용
  - 스파크의 spark-defaults.conf 에 지정

    spark.eventLog.enabled true

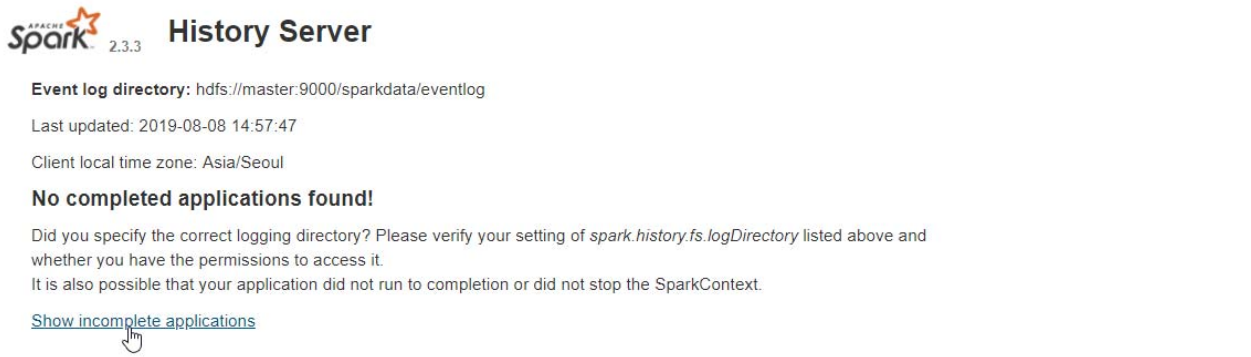
    spark.eventLog.dir hdfs://master:9000/sparkdata/eventlog

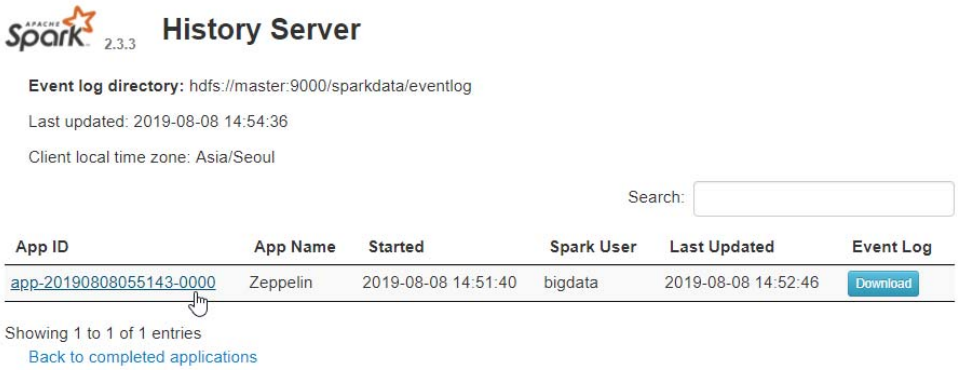
    spark.history.fs.logDirectory hdfs://master:9000/sparkdata/eventlog
  - 설정한 이벤트 로그 디렉토리 생성

    $ hadoop fs -mkdir /sparkdata/eventlog
  - 스파크 재시작
  - 히스토리 서버 실행

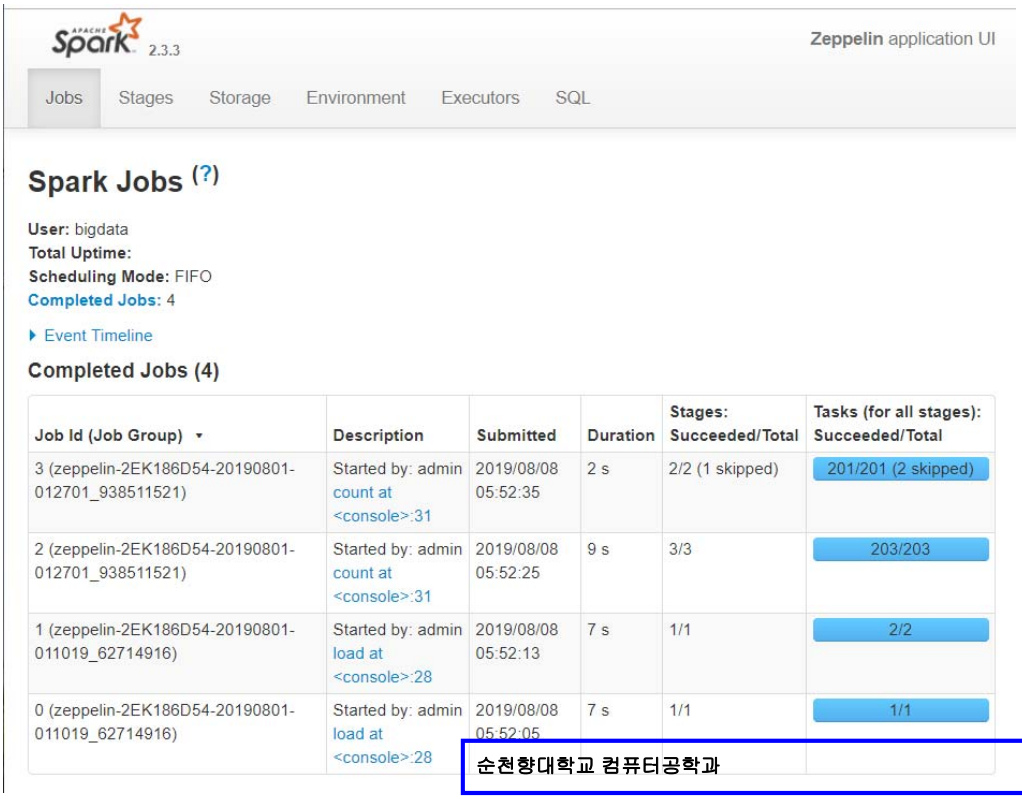
    $SPARK_HOME/sbin/start-history-server.sh
    - 중지: $SPARK_HOME/sbin/stop-history-server.sh
  - 스파크 응용을 실행하면 이벤트 로그가 설정한 디렉토리에 저장
  - 스파크 응용 실행 후 18080 포트로 웹 UI 접속
    - 가상머신 포트포워딩 설정해야 함

히스토리 웹 UI (1)





히스토리 웹 UI (2)



3. 스파크 응용 디버그 및 튜닝

성능 저하 문제 감지

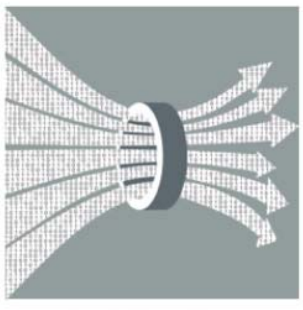
- 다른 태스크들 보다도 수행 시간이 길어서 데이터-병렬 처리의 성능에 문제가 되는 소스 태스크를 스큐(왜곡,skew)라 함

- 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

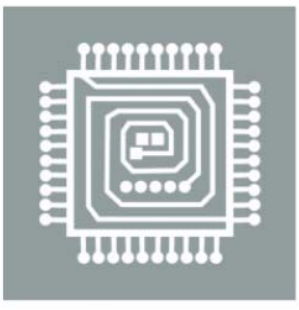
성능 저하 이슈

- 성능 저하는 다음과 같은 공통적인 이슈에 기인
  - 병렬성의 수준 (Level of Parallelism)
  - 셔플 동작 시 사용되는 직렬화 형식 (Serialization Format)
  - 최적화된 메모리 관리 (Memory Management)

Level of Parallelism



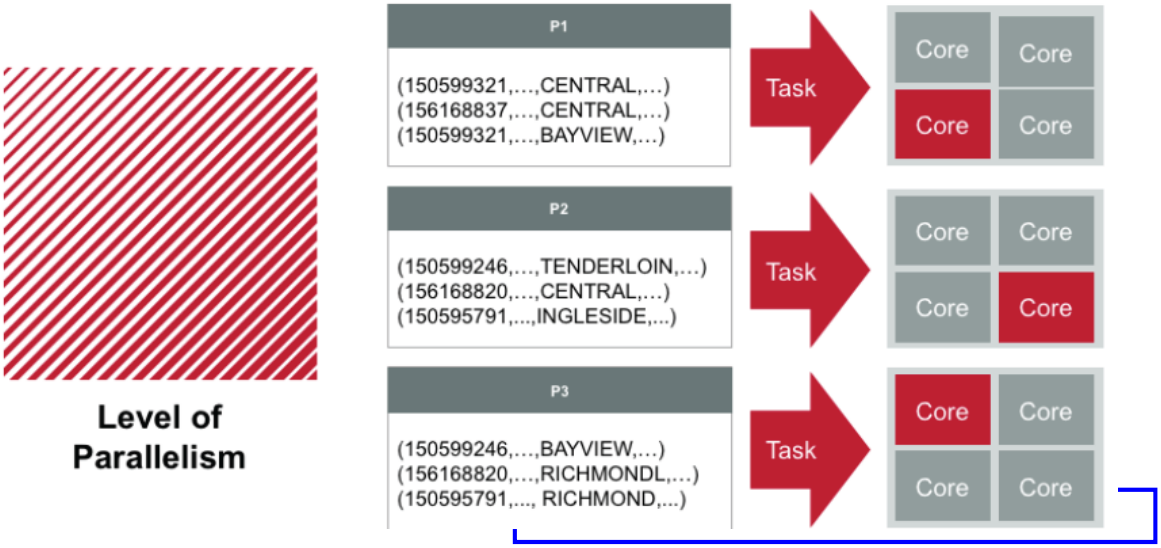
Serialization Format



Memory Management

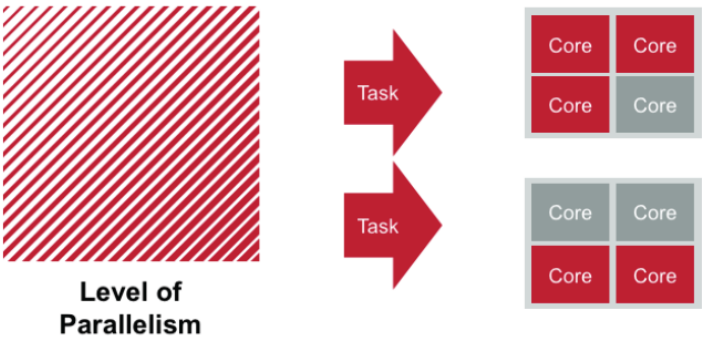
파티션 (Partition)

- 데이터세트의 RDD는 데이터의 부분들인 파티션으로 분할
  - 스케줄러는 각 파티션에 대해 하나의 태스크를 생성
  - 각 태스크는 클러스터의 단일 코어 상에서 실행



병렬성의 수준 (Level of Parallelism)

- 병렬성의 수준이 성능에 영향
  - 너무 큰 병렬성을 가지면 각 파티션과 관련된 오버헤드가 커져서 잡의 실행 시간 증가
  - 너무 적은 병렬성을 가지면 자원이 쉬게 됨
  - 자원의 너무 많이 또는 너무 적게 이용하지 않도록 파티션의 수를 조정할 필요가 있음



병렬성의 수준 튜닝 (Tuning Level of Parallelism)

- 파티션 수를 확인하고 조정
- 현재 파티션의 수 확인
  - 다음 메서드 중 하나를 사용하여 데이터세트의 RDD의 파티션 확인
    ds.rdd.partitions.size()
    ds.rdd.getNumPartitions()
  - 또는 스파크 웹 UI의 스테이지 페이지에서 태스크의 수 확인
    - 태스크마다 하나의 파티션을 가짐
- 파티션 수 조정
  - 파티션 수 축소
    ds.rdd.coalesce()
    - 너무 많은 태스크들이 쉬고 있을 때 축소
  - 파티션 수 증가
    ds.rdd.repartition()
    - 클러스터의 모든 코아를 사용하고 있지 않을 때 증가

Level of Parallelism

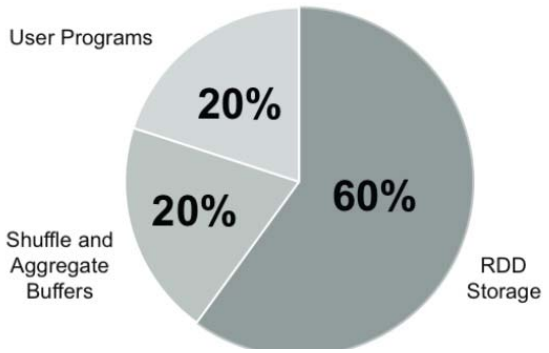
직렬화 형식 (Serialization Format)

- 셔플 동작 시 많은 양의 데이터가 네트워크 상에서 전송
- 스파크는 인코더가 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 자바 직렬화 프레임워크
  - https://github.com/EsotericSoftware/kryo

Serialization Format

메모리 관리 (Memory Management)

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



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
    - persist(MEMORY_ONLY_SER)
      - 직렬화된 형식으로 객체를 캐싱
      - 원본 객체 캐싱보다 더 느릴 수 있으나 가비지 컬렉션(garbage collection)을 없애 시간을 줄임

성능 저하 감지 (1)

- 스파크 웹 UI의 잡, 스테이지를 모니터링하여 성능 저하 요인을 감지
  - 느린 태스크
  - 읽기/쓰기 이슈
  - 스큐(skew)

성능 저하 감지 (2)

- 느린 태스크
  - 스테이지 상세 페이지에서 특정 노드에서 아주 느리게 수행하는 태스크를 파악
  - 일반적으로 셔플 문제로 인해 느리게 수행됨

- 읽기/쓰기 이슈
  - 많은 읽기/쓰기를 수행하는 태스크 파악
  - 캐싱을 적용하여 읽기를 최소화

- 스큐
  - 클러스터에 데이터와 태스크가 고르게 배분되는지를 확인
  - 스테이지 상세 페이지에서 느리게 수행되는 태스크들과 태스크들의 수를 확인
  - 노드들 상의 데이터의 파티션과 잡의 스케줄링도 확인

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 node
Apache Mesos	`work/` directory of Mesos slave node
Hadoop YARN	Use YARN log collection tool

과제

- 강의 시간의 실습 내용을 정리하여 제출
- 텀 프로젝트 과제
  - 텀 프로젝트 데이터를 사용하여 앞에서 배운 스파크 응용 모니터링을 적용하고 실행

참고 자료

- Build a Simple Apache Spark Application
  - https://learn.mapr.com/series/sparkv2/dev-361-build-and-monitor-apache-spark-applications-spark-v21
    - Lesson 5: Monitor Apache Spark Applications
- Spark Monitoring and Instrumentation
  - https://spark.apache.org/docs/latest/monitoring.html
- Enabling the Spark history service
  - https://www.ibm.com/support/knowledgecenter/en/SS3H8V_1.1.0/com.ibm.izoda.v1r1.azka100/topics/azkic_t_histsrv.htm