스파크 응용 구축

순천향대학교 컴퓨터공학과

이 상 정

학습 내용

1. 스파크 프로그램 수명주기(lifecycle)
2. SparkContext 함수 정의
3. 스파크 응용의 실행
4. 스파크 응용의 빌드

1. 스파크 프로그램 수명주기

스파크 프로그램 수명주기 단계

1. 드라이버 프로그램에서 입력 RDD 생성
   - 드라이버 프로그램에서 외부 데이터를 입력 받거나, 컬렉션을 병렬화(parallelize)하여 입력 RDD를 생성
2. 지연 변환(lazy transformation)을 통해 새로운 RDD 생성
   - filter(), map() 등을 사용하여 변환
3. 재사용되는 RDD는 캐싱
   - 재사용되는 중간 결과의 RDD들은 캐싱
4. 액션을 적용하여 계산을 시작
   - count(), collect() 등의 액션을 호출하여 병렬 계산을 수행

수명주기 단계 1,2

1. 드라이버 프로그램에서 입력 RDD 생성

```
val auctionRDD = sc.textFile("/sparkdata/auction/auctiondata.csv")
      .map(line=>line.split(","))
```

2. 지연 변환(lazy transformation)을 통해 새로운 RDD 생성

```
val auctionRDD = sc.textFile("/sparkdata/auction/auctiondata.csv")
      .map(line=>line.split(","))
val bidsitemRDD = auctionRDD.map(bid=>(bid(item),1))
                                    .reduceByKey((x,y)=>x+y).collect()
```


수명주기 단계 3

3. 재사용되는 RDD는 캐싱

```
val auctionRDD = sc.textFile("/sparkdata/auction/auctiondata.csv")
      .map(line=>line.split(","))
val bidsitemRDD = auctionRDD.map(bid=>(bid(item),1))
                                    .reduceByKey((x,y)=>x+y).collect()
bidsitemRDD.cache()
```

수명주기 단계 4

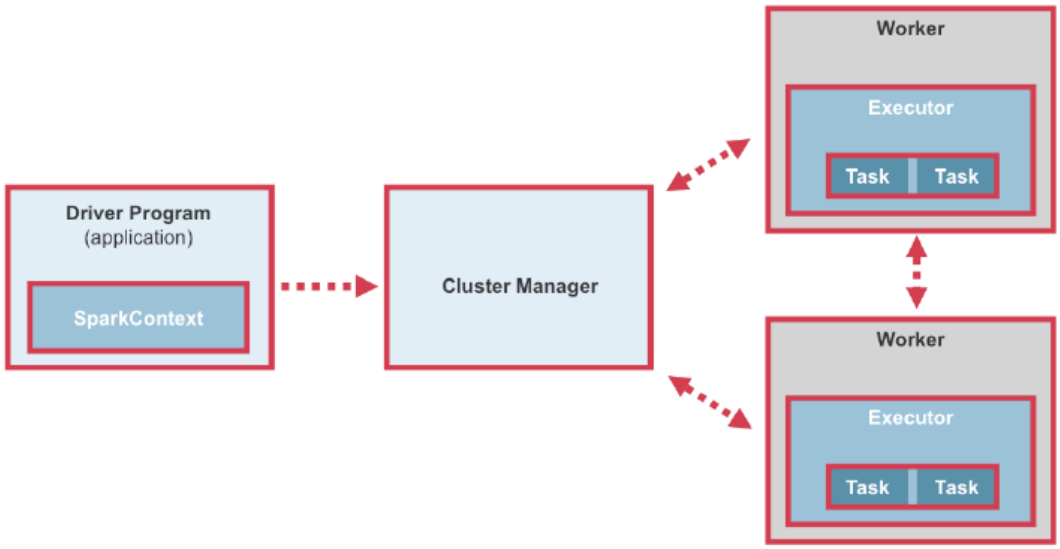
4. 액션을 적용하여 계산을 시작

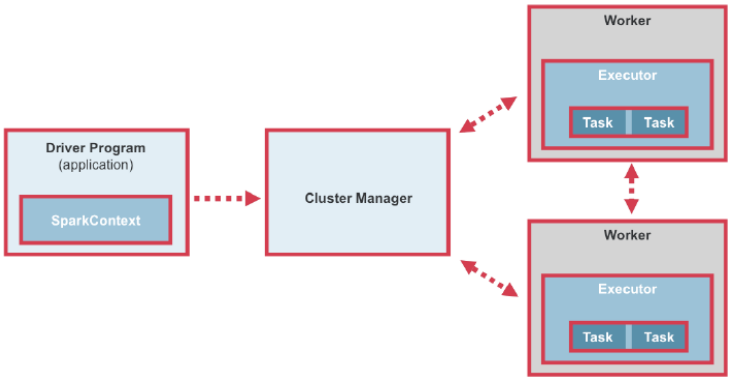
```
val auctionRDD = sc.textFile("/sparkdata/auction/auctiondata.csv")
      .map(line=>line.split(","))
val bidsitemRDD = auctionRDD.map(bid=>(bid(item),1))
                                    .reduceByKey((x,y)=>x+y).collect()
bidsitemRDD.cache()
bidsitemRDD.count()
```


2. SparkContext 함수 정의

스파크 클러스터 구성 – 복습

- 스파크 클러스터는 두 종류의 노드들로 구성
  - 하나의 드라이버 프로그램 노드
  - 다수의 작업자 노드 (worker nodes)
    - 클러스터의 각 노드는 실행자 프로세스 (executor process)를 실행



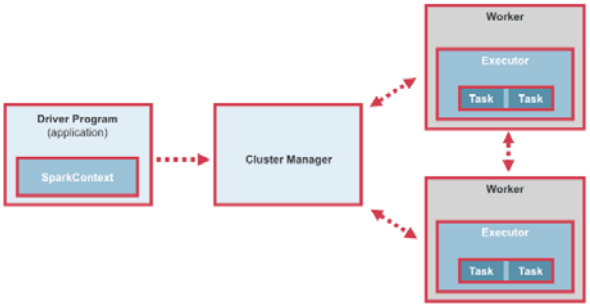


스파크 실행 순서 – 복습

1. 드라이버 프로그램이 SparkContext 객체를 생성
   - SparkContext는 한 응용의 클러스터 연결 접근 방식을 관리
2. SparkContext가 클러스터 관리자(cluster manager)에 연결
   - 클러스터 관리자는 각 응용들에 대한 자원을 할당
3. 클러스터 관리자에 연결되면 스파크는 작업자(worker) 노드들의 실행자(executor)들을 획득
   - 실행자(executor)는 각 응용에 대한 계산과 데이터를 저장하는 프로세스
4. SparkContext에 전달된 Jar 또는 파이썬 파일들이 실행자에게 송신
5. SparkContext가 실행자가 수행할 태스크들을 송신
6. 작업자 노드들은 데이터 저장소를 접근하여 데이터를 수집하고 출력

SparkContext 소개

- 스파크 프로그램은 SparkContext를 생성함으로써 시작
  - 응용 프로그램의 클러스터 연결 접근 방식을 관리
  - RDD를 생성하고 관리하는 메서드 제공
  - 스파크 셸에서는 자동으로 생성되어 sc로 초기화
  - 프로그램 작성 시에는 생성하고 초기화 필요
  - SparkConf 객체를 사용하여 SparkContext 초기화



SparkContext 생성

- SparkContext 생성을 위해 관련 클래스들을 임포트하고, SparkConf 객체를 사용하여 응용의 이름등을 초기화

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

val conf = new SparkConf().setAppName("AuctionsApp")
val sc = new SparkContext(conf)
```

AuctionsApp.scala

```
/* Simple App to inspect Auction data */
/* The following import statements are importing SparkContext, all subclasses and SparkConf*/
package solutions
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
//Will use max, min – import java.Lang.Math
import java.lang.Math

object AuctionsApp {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("AuctionsApp")
    val sc = new SparkContext(conf)
    /* Add location of input file */
    val usrhome = System.getenv("/spakdata")
    val aucFile = usrhome.concat("/auction/auctiondata.csv")
    //map input variables
    val auctionid = 0
    val bid = 1
    val bidtime = 2
    val bidder = 3
```



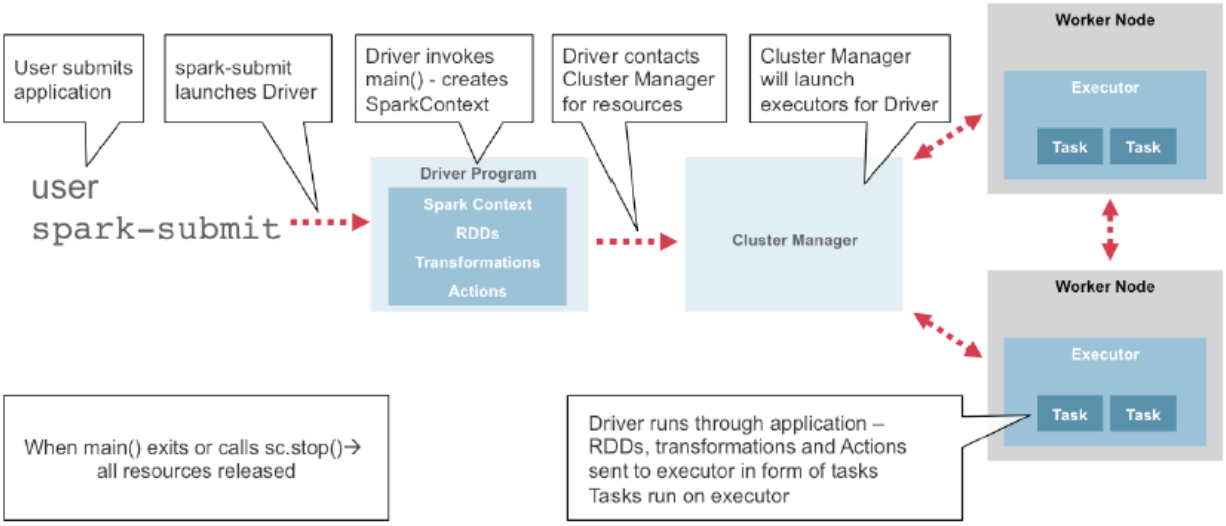
```
    val bidderrate = 4
    val openbid = 5
    val price = 6
    val itemtype = 7
    val daystolive = 8

     //build the inputRDD
    val auctionRDD = sc.textFile(aucFile).map(line => line.split(",")).cache()
    //total number of bids across all auctions
    val totalbids = auctionRDD.count()
    //total number of items (auctions)
    val totalitems = auctionRDD.map(line => line(auctionid)).distinct().count()
    //RDD containing ordered pairs of auctionid,number
    val bids_auctionRDD = auctionRDD.map(x => (x(auctionid), 1)).reduceByKey((x, y) => x + y)
    //max, min and avg number of bids
    val maxbids = bids_auctionRDD.map(x => x._2).reduce((x, y) => Math.max(x, y))
    val minbids = bids_auctionRDD.map(x => x._2).reduce((x, y) => Math.min(x, y))
    val avgbids = totalbids / totalitems
    println("total bids across all auctions: %s ".format(totalbids))
    println("total number of distinct items: %s ".format(totalitems))
    println("Max bids across all auctions: %s ".format(maxbids))
    println("Min bids across all auctions: %s ".format(minbids))
    println("Avg bids across all auctions: %s ".format(avgbids))
  }
}
```

3. 스파크 응용의 실행

스파크 응용의 실행 단계 (1)

- 스파크는 하나의 중앙 조정자인 하나의 드라이버와 여러 분산 작업자인 실행자(executor)로 구성된 마스터-슬레이브 아키텍처로 실행
  1. 먼저, 사용자는 spark-submit 명령으로 응용을 제출
  2. spark-submit 명령은 main() 메서드를 호출하는 드라이버 프로그램을 실행
     - main() 메서드에서는 드라이버에게 클러스터 관리자 (cluster manager) 위치를 알려주는 SparkContext를 생성
  3. 드라이버는 필요한 자원 요청 및 실행자의 실행을 위해 클러스터 관리자를 접촉
  4. 클러스터 관리자는 드라이버가 요청한 실행자를 실행
  5. 드라이버는 태스크 형식으로 각 실행자에게 전송된 응용(RDD, 변환 및 액션)을 통해 작업을 수행

스파크 응용의 실행 단계 (2)



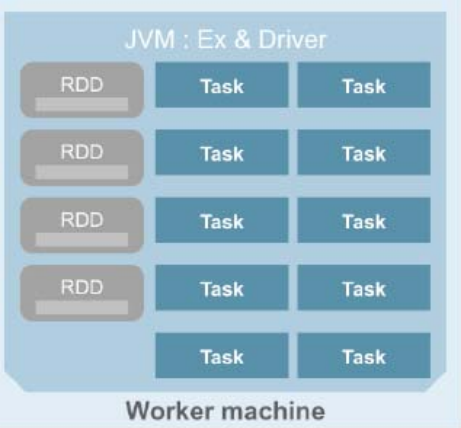
스파크 응용 실행 옵션

- 스파크는 클러스터 관리자를 통해 실행자를 실행
  - 클러스터 관리자는 스파크의 플러그가 가능한 컴포넌트(pluggable component)이어서 Hadoop YARN, Apache Mesos 등과 같은 다양한 자원들의 관리가 가능
  - 자체의 내장 클러스터 클러스터 관리자도 제공

- 스파크 실행 모드
  - 로컬 모드 (local mode)
  - 독립형 모드 (standalone mode)
  - 하둡 YARN
  - 아파치 Mesos

로컬 모드 (Local Mode)

- 로컬 모드는 하나의 JVM에서 드라이버와 작업자(worker)가 실행
  - 모든 RDD와 변수들이 같은 메모리 공간에 생성
  - 중앙의 마스터가 없고, 사용자가 실행을 시작
  - 개발 단계에서 프로토타이핑, 디버깅, 테스팅 dydeeh에 적합



독립형 모드 (Standalone Mode) (1)

- 스파크는 독립형 배치 모드(standalone deploy mode)를 제공
  - 스파크의 독립적인 클러스터 모드
  - 마스터와 작업자들을 수작업으로(manually) 실행하거나,스파크에서 제공하는 스크립트를 사용하여 실행
  - 테스트를 위해 단일 머신에서 마스터와 작업자들의 데몬들을 실행할 수도 있음
  - 클러스터의 각 노드에 스파크의 컴파일된 버전을 배치
  - 마스터의 spark://IP:PORT URL을 SparkContext 생성자에 전달하여 스파크 클러스터에서의 응용을 실행
- 두 가지 동작 모드
  - 클러스터 모드 (cluster mode)
  - 클라이언트 모드 (client mode)

독립형 모드 (2)

- 클러스터 모드에서는 드라이버가 클러스터 내의 하나의 작업자 프로세스로 실행
  - 클라이언트 프로세스는 응용을 제출하자마자 응용이 완료될 때를 기다리지 않고 바로 종료
- 클라이언트 모드에서는 드라이버가 클라이언트와 같은 프로세스로 실행
- spark-submit으로 실행을 시작하면 응용의 jar 파일들이 자동적으로 모든 작업자 노드(worker node)에 분배

Cluster Mode	Client Mode
Driver launched from worker process inside cluster	Driver launches in the client process that submitted the job
Can quit without waiting for job results (async)	Need to wait for result when job finishes (sync)

하둡 YARN

- 하둡 클러스터의 YARN에서 실행
- 클러스터 모드에서는 비동기 프로세스(async process)로 실행되어 작업의 종료를 기다리지 않음
  - YARN 클러스터의 응용 마스터(alpplication master) 프로세스에서 드라이버 실행
- 클라이언트 모드에서는 작업의 종료를 기다림
  - 상호작용의 셸이나 디버깅 등에 적합
  - 드라이버가 클라이언트 프로세스로 실행되고, 응용 마스터 프로세스는 YARN으로부터의 자원 요청에 사용

Cluster Mode	Client Mode
Driver launched in Application Master in cluster or worker	Driver launched in the client process that submitted the job
Can quit without waiting for job results (async)	Need to wait for result when job finishes (sync)
Suitable for production deployments	Useful for Spark interactive shell or debugging

spark-submit

```
/spark-home/bin/spark-submit – used to run in any mode

./bin/spark-submit \
  --class <main-class>
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

- Spark-submit은 응용을 실행하는 스크립트
  - --class 는 응용의 시작점인 main class 기술
  - --master 는 클러스터의 마스터 URL 또는 실행 모드
  - --deploy-mode는 동작 모드
  - --conf 는 스파크 설정 프로퍼티
  - <application-jar>는 응용의 jar 파일
  - [application-arguments]는 main() 메서드로 전달되는 인수

spark-submit 예

- 실행 모드 별 예
  - 로컬 모드
    - $SPARK_HOME/bin/spark-submit –class <class path>
      --master local[n] <application-jar>
  - 독립형 모드
    - $SPARK_HOME/bin/spark-submit –class <class path>
      --master spark:<master url> <application-jar>
  - YARN 클러스터
    - $SPARK_HOME/bin/spark-submit –class <class path>
      --master yarn --deploy-mode cluster <application-jar>
  - YARN 클라이언트
    - $SPARK_HOME/bin/spark-submit –class <class path>
      --master yarn --deploy-mode client <application-jar>

4. 스파크 응용의 빌드

Scala 설치

- 우분투 14.04에 루트 권한으로 스칼라 설치
  - Scala 버전 2.11.11
    - 최근 버전은 2.12.2(2017년 6월 기준)이나 메이븐 저장소 스파크 코어 호환성을 고려하여 2.11.11 설치

```
# cd /usr/local/src
# wget https://downloads.lightbend.com/scala/2.11.11/scala-2.11.11.tgz
# mkdir scala
# tar -xvf scala-2.11.11.tgz -C scala
```

  - 사용자의 ~/.bashrc 의 PATH 환경변수에 스칼라 경로 추가

```
$ vi ~/.bashrc
    .......
    export SCALA_HOME=/usr/local/src/scala/scala-2.11.11
    export PATH=$SCALA_HOME/bin:$PATH
$ source ~/.bashrc
```

Scala 설치 예

```
root@MASTER:~# cd /usr/local/src
root@MASTER:/usr/local/src# wget https://downloads.lightbend.com/scala/2.11.11/scala-2.11.11.tgz
--2017-06-27 09:52:40--  https://downloads.lightbend.com/scala/2.11.11/scala-2.11.11.tgz
Resolving downloads.lightbend.com (downloads.lightbend.com)... 52.84.186.208, 52.84.186.112, 52.84.186.239, ...
Connecting to downloads.lightbend.com (downloads.lightbend.com)|52.84.186.208|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 29086055 (28M) [application/octet-stream]
Saving to: 'scala-2.11.11.tgz.2'

100%[======================================================================>] 29,086,055  11.2MB/s   in 2.5s

2017-06-27 09:52:47 (11.2 MB/s) - 'scala-2.11.11.tgz.2' saved [29086055/29086055]

root@MASTER:/usr/local/src# mkdir scala

root@MASTER:/usr/local/src# tar -xvf scala-2.11.11.tgz -C scala
scala-2.11.11/
scala-2.11.11/lib/

root@MASTER:/usr/local/src# ls
scala  scala-2.11.11.tgz  scala-2.11.11.tgz.1  scala-2.11.11.tgz.2  scala-2.12.2  scala-2.12.2.tgz
root@MASTER:/usr/local/src# ls scala
scala-2.11.11
root@MASTER:/usr/local/src#
cssjlee@MASTER:~$ vi .bashrc
cssjlee@MASTER:~$ source .bashrc
cssjlee@MASTER:~$
cssjlee@MASTER:~$ scala
Welcome to Scala 2.11.11 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121).
Type in expressions for evaluation. Or try :help.

scala>

scala> :quit
cssjlee@MASTER:~$
```


SBT 설치 (1)

❑ SBT (Simple Build Tool)

- Scala 기반 빌드 툴로 Java, Scala 소스 코드 빌드
  - 아파치 Ivy 사용하여 의존성(dependency) 관리
  - Apache Maven 보다는 단순하고 쉽게 사용
  - 로컬 저장 소 디렉토리: ~/ivy2/cache
  - http://www.scala-sbt.org/

- 설치
  - org.scala-sbt sbt 0.13.15 (2017년 6월)

```
# echo "deb https://dl.bintray.com/sbt/debian /" |
  tee -a /etc/apt/sources.list.d/sbt.list
# apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
  2EE0EA64E40A89B84B2DF73499E82A75642AC823
# apt-get update
# apt-get install sbt
```

SBT 설치 예

❑ 처음 실행 시 저장소 라이브러리 적재

```
@MASTER:~$ sbt
Getting org.scala-sbt sbt 0.13.15  (this may take some time)...
downloading https://repo.typesafe.com/typesafe/ivy-releases/org.scala-sbt/sbt/0.13.15/jars/sbt.jar ...
        [SUCCESSFUL ] org.scala-sbt#sbt;0.13.15!sbt.jar (4487ms)
downloading https://repo1.maven.org/maven2/org/scala-lang/scala-library/2.10.6/scala-library-2.10.6.jar ...
        [SUCCESSFUL ] org.scala-lang#scala-library;2.10.6!scala-library.jar (3419ms)
downloading https://repo.typesafe.com/typesafe/ivy-releases/org.scala-sbt/main/0.13.15/jars/main.jar ...
        [SUCCESSFUL ] org.scala-sbt#main;0.13.15!main.jar (5153ms)
downloading https://repo.typesafe.com/typesafe/ivy-releases/org.scala-sbt/compiler-interface/0.13.15/jars/compil
er-interface.jar ...
        [SUCCESSFUL ] org.scala-sbt#compiler-interface;0.13.15!compiler-interface.jar (4509ms)
downloading https://repo.typesafe.com/typesafe/ivy-releases/org.scala-sbt/actions/0.13.15/jars/actions.jar ...
        [SUCCESSFUL ] org.scala-sbt#actions;0.13.15!actions.jar (4430ms)
downloading https://repo.typesafe.com/typesafe/ivy-releases/org.scala-sbt/main-settings/0.13.15/jars/main-settin
gs.jar ...
        [SUCCESSFUL ] org.scala-sbt#main-settings;0.13.15!main-settings.jar (4599ms)
downloading https://repo.typesafe.com/typesafe/ivy-releases/org.scala-sbt/interface/0.13.15/jars/interface.jar .
..
        [SUCCESSFUL ] org.scala-sbt#interface;0.13.15!interface.jar (4680ms)
Updated file /home/cssjlee/project/build.properties setting sbt.version to: 0.13.15
[info] Loading project definition from /home/cssjlee/project
[info] Updating {file:/home/cssjlee/project/}cssjlee-build...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Set current project to cssjlee (in build file:/home/cssjlee/)
>
> exit
@MASTER:~$
```


AuctionsApp 응용 프로그램 작성

❑ SBT 프로젝트 스크립트 작성

- 소스 프로그램: src/main/scala/AuctionsApp.scala

```
cssjlee@MASTER:~/spark/auction/sbt$ find .
.
./src
./src/main
./src/main/scala
./src/main/scala/AuctionsApp.scala
./auctions.sbt
```

- 프로젝트 스크립트
  - 메이븐 저장소(Maven Repository)의 Spark Project Core 2.1.1 사용
    - https://mvnrepository.com/artifact/org.apache.spark/spark-core_2.10/2.1.1
  - auctions.sbt

```
name := "Auctions Project"
version := "1.0"
scalaVersion := "2.11.11"
libraryDependencies += "org.apache.spark" % "spark-core_2.10" % "2.1.1"
```

AuctionsApp.scala

```
/* Simple App to inspect Auction data */
/* The following import statements are importing SparkContext, all subclasses and SparkConf*/
package solutions
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
//Will use max, min – import java.Lang.Math
import java.lang.Math

object AuctionsApp {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("AuctionsApp")
    val sc = new SparkContext(conf)
    /* Add location of input file */
    val usrhome = System.getenv("/spakdata")
    val aucFile = usrhome.concat("/auction/auctiondata.csv")
    //map input variables
    val auctionid = 0
    val bid = 1
    val bidtime = 2
    val bidder = 3
```



```
    val bidderrate = 4
    val openbid = 5
    val price = 6
    val itemtype = 7
    val daystolive = 8

     //build the inputRDD
    val auctionRDD = sc.textFile(aucFile).map(line => line.split(",")).cache()
    //total number of bids across all auctions
    val totalbids = auctionRDD.count()
    //total number of items (auctions)
    val totalitems = auctionRDD.map(line => line(auctionid)).distinct().count()
    //RDD containing ordered pairs of auctionid,number
    val bids_auctionRDD = auctionRDD.map(x => (x(auctionid), 1)).reduceByKey((x, y) => x + y)
    //max, min and avg number of bids
    val maxbids = bids_auctionRDD.map(x => x._2).reduce((x, y) => Math.max(x, y))
    val minbids = bids_auctionRDD.map(x => x._2).reduce((x, y) => Math.min(x, y))
    val avgbids = totalbids / totalitems
    println("total bids across all auctions: %s ".format(totalbids))
    println("total number of distinct items: %s ".format(totalitems))
    println("Max bids across all auctions: %s ".format(maxbids))
    println("Min bids across all auctions: %s ".format(minbids))
    println("Avg bids across all auctions: %s ".format(avgbids))
  }
}
```

AuctionsApp 응용의 빌드

- 응용 빌드

  $ sbt package

  - target/scala-2.11/auctions-project_2.11-1.0.jar 파일 생성

```
@MASTER:~/spark/auction/sbt$ sbt package
[warn] Executing in batch mode.
[warn]   For better performance, hit [ENTER] to switch to interactive mode, or
[warn]   consider launching sbt without any commands, or explicitly passing 'shell'
[info] Loading project definition from /home/cssjlee/spark/auction/sbt/project
[info] Set current project to Auctions Project (in build file:/home/c   e/spark/auction/sbt/)
[info] Updating {file:/home/   /spark/auction/sbt/}sbt...
[info] Resolving jline#jline;2.14.3 ...
[info] Done updating.
[info] Compiling 1 Scala source to /home/   /spark/auction/sbt/target/scala-2.11/classes...
[info] Packaging /home/   /spark/auction/sbt/target/scala-2.11/auctions-project_2.11-1.0.jar ...
[info] Done packaging.
[success] Total time: 16 s, completed Jun 28, 2017 9:15:03 PM
```


AuctionsApp 응용 실행

- spark-submit 명령을 사용하여 실행

  $ $SPARK_HOME/bin/spark-submit --class AuctionsApp
  --master yarn target/scala-2.11/auctions-project_2.11-1.0.jar

```
@MASTER:~/spark/auction/sbt$ $SPARK_HOME/bin/spark-submit --class AuctionsApp --master yarn
 target/scala-2.11/auctions-project_2.11-1.0.jar
total bids across all auctions: 10654
total number of distinct items: 627
Max bids across all auctions: 75
Min bids across all auctions: 1
Avg bids across all auctions: 16
   @MASTER:~/spark/auction/sbt$
```

과제

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

참고 자료

- MapR Academy DEV 360 Apache Spark Essentials
  - http://learn.mapr.com/dev-360-apache-spark-essentials
  - Lesson 3: Build a Simple Apache Spark Application