Assignment 3: Task 3 and 4

Task 3 and 4

- Task 3
  - Counting the number of commands in every 15 min interval
- Overall idea:
  - Extract all timestamp
  - Map: (time_interval = timestamp/(15 min), 1)
  - Reduce: (time_interval, count)

```
JavaRDD<Long> timestamp = lines.flatMap(new FlatMapFunction<String, Long>() {
        @Override
        public Iterable<Long> call(String inputLine){
                if (inputLine.contains("<Command ")) {
                        Pattern p = Pattern.compile("timestamp=\"(\\d*?)\"");
                        Matcher m = p.matcher(inputLine);
                        if (m.find()) {
                           return Arrays.asList(Long.parseLong(m.group(1)));
                        }
                }
                return Arrays.asList();
        }
});
```

```
JavaPairRDD<Integer, Integer> timestamp_one = timestamp.mapToPair(
        new PairFunction<Long, Integer, Integer>() {
        @Override
        public Tuple2<Integer, Integer> call(Long timeStamp) {
                int interval = (int)(timeStamp/1000/(15*60));
                return new Tuple2<Integer, Integer>(interval, 1);
        }
});

JavaPairRDD<Integer, Integer> timestamp_count = timestamp_one.reduceByKey(
        new Function2<Integer,Integer,Integer>(){
        @Override
        public Integer call(Integer a, Integer b){
                return (a+b);
        }
});
```

Task 3 and 4

- Task 4
  - Counting and ordering all distinct commands.
- Overall idea:
  - Extract all distinct commands
  - Map: (command, 1)
  - Reduce: (command, count)
  - Swap, sort by key, swap back
    - (command, count) -> (count, command)
    - Sort({(count, command), …, (count, command)})
    - (count, command) -> (command, count)

```
JavaRDD<String> cmdType = lines.flatMap(new FlatMapFunction<String, String>() {
    @Override
    public Iterable<String> call(String s){
        String cmdTypeString = "";
        if (s.contains("<Command ")) {
            Pattern p1 = Pattern.compile("_type=\"(.*?)\"");
            Matcher m1 = p1.matcher(s);
            if (m1.find()) {cmdTypeString = m1.group(1);}
            if (!cmdTypeString.equals("EclipseCommand")) {
                return Arrays.asList(cmdTypeString);
            } else {
                Pattern p2 = Pattern.compile("commandID=\"(.*?)\"");
                Matcher m2 = p2.matcher(s);
                if (m2.find()) {
                    cmdTypeString = "EclipseCommand:"+m2.group(1);
                    return Arrays.asList(cmdTypeString);
                }
            }
        }
        return Arrays.asList();
    }
});
```

```
JavaPairRDD<String, Integer> cmdOne = cmd_type.mapToPair(
        new PairFunction<String, String, Integer>() {
        @Override
        public Tuple2<String, Integer> call(String a) {
                return new Tuple2<String, Integer>(a, 1);
        }
});

JavaPairRDD<String, Integer> cmdCount = cmdOne.reduceByKey(
        new Function2<Integer,Integer,Integer>(){
        @Override
        public Integer call(Integer a, Integer b){
                return (a+b);
        }
});
```

```
JavaPairRDD<Integer, String> swappedPair = cmdCount.mapToPair(
        new PairFunction<Tuple2<String, Integer>, Integer, String>() {
        @Override
        public Tuple2<Integer, String> call(Tuple2<String, Integer> item) throws Exception {
                return item.swap();
        }
});

JavaPairRDD<Integer, String> swappedPairSorted = swappedPair.sortByKey(false);

JavaPairRDD<String, Integer> swappedPairSortedSwapBack = swappedPairSorted.mapToPair(
        new PairFunction<Tuple2<Integer, String>, String, Integer>() {
        @Override
        public Tuple2<String, Integer> call(Tuple2<Integer, String> item) throws Exception {
                return item.swap();
        }
});

List<Tuple2<String, Integer>> output = swappedPairSortedSwapBack.collect();
```

DS assignments review

Socket programming

- System calls for TCP/IP sockets
  - Bind
  - Listen
  - Accept
  - Read
  - Write
  - ….
- Server/Client communication

Map/Reduce

- Map/Reduce paradigm:
  - Key/Value pair
- Common tasks:
  - Counting
  - TopK
  - Join

Apache Spark

- Apache Spark VS. Hadoop
- RDD
- Common tasks:
  - Counting
  - Sorting