Bold Challenge

Raul Barth

1. Solution Modelling

The first observation to be done is that the number of the rooms is fixed (6 rooms), while the number of departments is variable (maximum 10).

My first thought was to use Spark for dealing with the data and use DataFrame to structure the input csv file. Doing that, it would be possible to query the data using SparkSQL and make some aggregation to extract the needed output. However, following the environment I would have, I chose for not using it, and for developing the solution using pure Java.

Considering I am using Java core and that the output of the code is CSV files saved in Department x Room pairs, the solution is to read the input file and, for each line, spread the processing into as much threas as rooms we have (5 in our case). That is the work-around I found for not using neither Spark nor MapReduce (as I do not know precisely how your environment is).

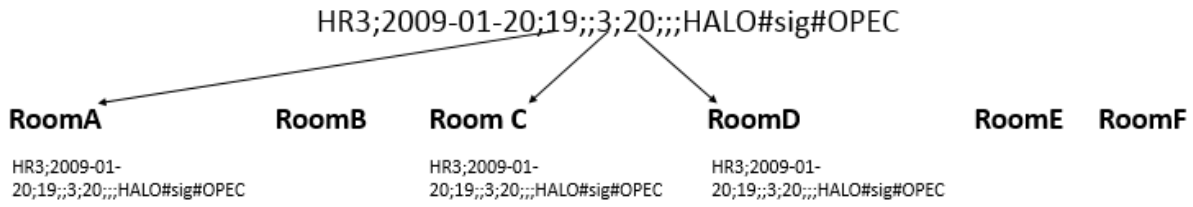
The main component of the solution is the use of *MultiValueMap* that comes with apache-commons and is a multiple values HashMap. Therefore, the *key* is the pair Department x Room and the values are the input file lines that fit that relation.

2. Algorithm

The algorithm is as follows:

- Read input file
- Start threads, one for each Room (6 in our case)
- Read file lines
    - For each line
        - Build the pairs (departments x rooms) using the threads
        - Insert the pairs in the MultiValueMap
            - Key: “department - room” (ex: HR1-roomA)
            - Values: all lines which fits has not null at the room position
- Create the files based on the (key,values) pairs

For each line of the input file, the code spread it between the 6 rooms. Above, a line containing department HR3 is included in HR3xRoomA, HR3-RoomC and HR3-RoomD:



3. Source Code

The development was done using git and the source code is available at GitHub in:

https://github.com/barthraul/challenge

4. **Generate .JAR:**

The .JAR can be generated using a simple Maven command:

```
mvn clean install
mvn package
```

It will be generated at Challenge/target/

5. **Run .JAR file:**

The executable .jar file is in /Challenge/target/boldChallenge.jar

java –jar boldChallenge.jar <inputFile> <outputPath> <rooms> <departments>

Please, consider the following:

- <inputFile>: the input csv file to be considered
- <outputPath>: the output path where the list of csv will be saved
- <rooms>: list of rooms to be considered, separated by ",". If you want all the rooms to be considered, please insert "all". It is case sensitive, so follow the format of "room" plus the room letter in uppercase (roomA, roomB, etc)
- <departments>: list of departments to be considered, separated by ",". If you want all the departments to be considered, please insert "all". It is case sensitive.

Ex:

Java –classpath challenge.jar bold.challenge.MainProcess D:/dataset.csv D:/output roomA,roomB,roomC HR1,HR2

Java –classpath challenge.jar bold.challenge.MainProcess D:/dataset.csv D:/output all HR1,HR2

6. **Considerations**

- It is a Maven project, build using JDK 1.8, which must be consider when using Lambda.
- Use of Thread
- The program is composed by two classes, one containing the initial process, with set ups and threads callings, and the other implementing the threads.
- A JUnit class was built for unit tests.
- Tests were performed using JUnit and some performance tests was done comparing the processing time using different size of input files.

7. **Performance Tests**

Dataset Size	**Processing Time (millisec)**	**Output Generated Files**
6.000 rows (0.245Mb)	255 milliseconds	48
50.000 rows (2.5Mb)	756 milliseconds	48
100.000 rows (4.09Mb)	1304 milliseconds	48
200.000 rows (8.17Mb)	2513 milliseconds	48
400.000 rows (8.17Mb)	4747 milliseconds	48

8. **Time Complexity**

Regarding that the main part of the code is composed by a *while*, which should be equivalent to a *for loop,* I would say that this code complexity is O(n). It can be realized from the table above. That is not the best efficiency an algorithm can have.