04 JULY 2022

HADOOP & MAPREDUCE

```
class Mapper1BigData extends Mapper<
                    LongWritable, // Input key type
                    Text,         // Input value type
                    Text,         // Output key type
                    IntWritable> {// Output value type

    protected void map(
            LongWritable key,   // Input key type
            Text value,         // Input value type
            Context context) throws IOException, InterruptedException {

            String[] fields = value.toString().split(",");
            String os = fields[2];
            String date = fields[1];

            if(date.compareTo("2021/07/04") >= 0 && date.compareTo("2022/07/03")
<= 0) {
                context.write(new Text(os), new IntWritable(1));
            }
    }
}


class Reducer1BigData extends Reducer<Text, // Input key type
        IntWritable, // Input value type
        Text, // Output key type
        IntWritable> { // Output value type

    private String maxOs;
    private int maxCount;

    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        this.maxOs = "";
        this.maxCount = 0;
    }

    @Override
    protected void reduce(
            Text key, // Input key type
            Iterable<IntWritable> values, // Input value type
```

```
            Context context) throws IOException, InterruptedException {

        String k = key.toString();
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }

        if (this.maxCount == 0 ||
                (this.maxCount == sum && k.compareTo(this.maxOs) < 0) ||
                sum > this.maxCount) {
            this.maxCount = sum;
            this.maxOs = k;
        }
    }

    @Override
    protected void cleanup(Context context) throws IOException,
InterruptedException {
        if (this.maxCount != 0)
            context.write(new Text(this.maxOs), new IntWritable(maxCount));
    }
}


class Mapper2BigData extends Mapper<Text, // Input key type
        Text, // Input value type
        NullWritable, // Output key type
        Text> {// Output value type

    protected void map(Text key, // Input key type
            Text value, // Input value type
            Context context) throws IOException, InterruptedException {

        context.write(NullWritable.get(), new Text(key.toString() + "_" +
value.toString()));
    }
}


class Reducer2BigData extends Reducer<
        NullWritable,           // Input key type
        Text,    // Input value type
        Text,           // Output key type
        IntWritable> {  // Output value type
```

```
    @Override
    protected void reduce(
            NullWritable key, // Input key type
            Iterable<Text> values, // Input value type
            Context context) throws IOException, InterruptedException {

        String maxOs = "";
        int maxCount = 0;

        // this for cycle should be iterate just once
        for(Text i : values) {
            String t = i.toString();
            String[] fields = t.split("_");
            String os = fields[0];
            int count = Integer.parseInt(fields[1]);

            if(maxCount == 0 ||
                    (maxCount == count && os.compareTo(maxOs) < 0) ||
                    count > maxCount) {
                maxCount = count;
                maxOs = os;
            }
        }

        context.write(new Text(maxOs), new IntWritable(maxCount));

    }
}
```

SPARK

```
serverPath = "data/Servers.txt"
patchesPath = "data/Patches.txt"
appliedPatchesPath = "data/AppliedPatches.txt"

outputPath1 = "outPart1/"
outputPath2 = "outPart2/"

# Define the rdds associated with the used input files


# Input format: SID, OS, Model
serverRDD = sc.textFile(serverPath)

# Input format: PID, ReleaseDate, OS
patchesRDD = sc.textFile(patchesPath)

# Input format: PID, SID, ApplicationDate
appliedPatchesRDD = sc.textFile(appliedPatchesPath)

########################################
# PART 1
########################################

# First, select the patches associated with Ubuntu2.
# Then, map the patches RDD into a pair rdd:
# key = PID
# value = ReleaseDate

def mapPidRelDate(l):
    fields = l.split(",")
    pid = fields[0]
    releaseDate = fields[1]
    return (pid, releaseDate)


pidOsUbuntuRDD = patchesRDD.filter(lambda l: l.split(",")[2]=="Ubuntu2")\
                .map(mapPidRelDate)

# from appliedPatches RDD obtain a pair RDD with
# key = PID
# value = AppliedDate

def mapPidApplDate(l):
    fields = l.split(",")
```

```
    pid = fields[0]
    applicationDate = fields[2]
    return (pid, applicationDate)


pidAppliedDateRDD = appliedPatchesRDD.map(mapPidApplDate)

# Join the two RDDs so that (left = appliedPatches, right = patches)
# key = PID
# value = (application date, release date)
# then filter only those lines for which application date == release date
patchesAppliedAtRelease = pidAppliedDateRDD\
                          .join(pidOsUbuntuRDD)\
                          .filter(lambda t: t[1][0]==t[1][1])

# Each element in patchesAppliedAtRelease represents a patch applied on a server
with Ubuntu2
# at the release date
# Now map all the elements into a pairRDD with
# key = PID
# value = 1
# and use a reduceByKey to count for each patch the number of servers on which
# the patches were applied at release date.
# Filter and keep only those patches which were applied to 100 servers or more
res1 = patchesAppliedAtRelease.map(lambda t: (t[0], 1)) \
                   .reduceByKey(lambda i1, i2: i1 + i2)\
                   .filter(lambda s: s[1] >= 100)

# Store the selected PIDs in the first output folder
res1.keys()\
    .saveAsTextFile(outputPath1)

########################################
# PART 2
########################################

# Starting from applied patches rdd, filter only those patches applied in 2021
# and map into a pair RDD with
# key = SID
# value = month
# and perform a distinct operation to keep for each server the distinct months

def filter2021(l):
    fields = l.split(",")
    date = fields[2]
```

```
    return date.startswith("2021")

def sidMonth(l):
    fields = l.split(",")
    sid = fields[1]
    date = fields[2]
    month = int(date.split("/")[1])
    return (sid, month)


serverMonthAppliedPatch = appliedPatchesRDD\
            .filter(filter2021)\
            .map(sidMonth)\
            .distinct()

# Compute the number of distinct months with at least one applied patch for each
server

serverMonths21NumApplPatches = serverMonthAppliedPatch.mapValues(lambda v: 1)\
                                                      .reduceByKey(lambda v1, v2:
v1+v2)

# Calculate the number of distinct months in 2021 without at least one applied
patch
# for each server by applying the formula 12 - number of months with applied
patches
serverMonths21NumNoApplPatches = serverMonths21NumApplPatches.mapValues(lambda v:
12-v)

# serverMonths21NoApplPatches does not contain the servers without applied
patches
# for all the 12 months of 2021

# Use serversRDD to gather information from all servers

# Prepare a pairRDD with
# key = SID
# value = 12
def SID12(l):
    fields = l.split(",")
    sid = fields[0]
    return (sid, 12)
```

```
allServers = serverRDD.map(SID12)

#  Retrieve the servers for which no patches were applied in 2021
serverNoApplPatches2021 = allServers.subtractByKey(serverMonths21NumApplPatches)

# The complete result is the union between serverNoApplPatches2021
# and serverMonths21NumNoApplPatches
res2 = serverNoApplPatches2021.union(serverMonths21NumNoApplPatches)

# Store the result in second output folder
res2.saveAsTextFile(outputPath2)
```