A double-for-loop (nested loop) in Spark

The following code snippet shows how all values for just some variables of interest within specified timeframes (windows) are extracted from a huge dataframe. Therefore it uses two for-loop. One to iterate through the list of variables that are of interest. Second to iteratre through the all timeframes.

```
    for var_ in variables:
        for incident in incidents:

            var_df = df.filter(
                (df.Variable == var_)
                & (df.Time > incident.startTime)
                & (df.Time < incident.endTime))
```

The problem is, that this code is executed on the driver node (master node) and that each iteration filters against the original huge dataframe. Therefore we will try to do that in a way so that most of the work will be distributed to the worker nodes.

In [104]:

```
input_hdfs_filepath_observations = 'parameters_sample.parquet'
input_local_filepath_incidents = 'incidents.csv'
csv_separator = '\t'

# imports
from pyspark.sql import HiveContext
from pyspark.sql.types import *
from pyspark.sql import functions as F
import pandas as PD
```

Incidents

Load incidents from CSV and parallelize them to Spark

An incident is a timeframe (time window) in which something was went wrong in the factory (the underlying data). Therefore we want to investigage the variables of interest within those timeframes. An incident is specified by a **start timestamp** and an **end timestamp**.

In [105]:

```
schema = StructType([StructField('start', StringType(), True),
                     StructField('end', StringType(), True)])

incidents = sqlCtx.createDataFrame(PD.read_csv(input_local_filepath_incidents, s
ep=csv_separator, header=None), schema)
incidents.show()
```

```
+-------------------+-------------------+
|              start|                end|
+-------------------+-------------------+
|2015-05-09 00:00:12|2015-05-15 23:59:58|
|2015-05-25 00:00:12|2015-05-27 23:59:58|
|2015-06-15 00:00:12|2015-06-19 23:59:58|
|2015-07-01 00:00:12|2015-07-05 23:59:58|
|2015-07-15 00:00:12|2015-07-19 23:59:58|
|2015-08-28 00:00:12|2015-08-30 23:59:58|
+-------------------+-------------------+
```

Add sequential ID to incidents

We need some kind of identifiert that can be tracked through all transformations.

In [106]:

```
from pyspark.sql.functions import monotonicallyIncreasingId
#incidents = incidents.withColumn("id", monotonicallyIncreasingId())

#incidents = incidents.withColumn("id", F.rowNumber().over(Window.partitionBy("s
tart").orderBy("end")))
#incidents.show()
```

Convert timestamp from StringType to TimestampType

The **start timestamp** and the **end timestamp** of an incident are specified as string values in the CSV file. The Pandas CSV read function can not convert them to timestamps automatically. Therefore we will convert them manually.

In [107]:

```
from pyspark.sql.functions import udf
from datetime import datetime
from pyspark.sql.types import TimestampType

def string_to_timestamp(value):
    return datetime.strptime(value, "%Y-%m-%d %H:%M:%S")

str_timestamp = udf(string_to_timestamp, TimestampType())

incidents = incidents.withColumn("start_time",str_timestamp(incidents.start))
incidents = incidents.withColumn("end_time",str_timestamp(incidents.end))
incidents = incidents.drop(incidents.start).drop(incidents.end)
incidents.printSchema()
```

```
root
 |-- start_time: timestamp (nullable = true)
 |-- end_time: timestamp (nullable = true)
```

Observations

Read the observations

The observations contain all variables and all their values specified by a timestamp of the observation and the observed value.

In [108]:

```
#Variable     Time                Value
#852-YF-007   2016-05-10 00:00:00 0
#852-YF-007   2016-05-09 23:59:00 0
#852-YF-007   2016-05-09 23:58:00 0

df = sqlCtx.read.option("mergeSchema", "true").parquet(input_hdfs_filepath_obser
vations).repartition(100).cache()
```

Important variables

Extract top 10 variables from observations

Though this is a coding sample, we don't have a real list of variables of interest. So we simply choose the top 10 variables that have the most observations in the dataframe.

In [109]:

```
df_observations = df.groupby(df.Variable).agg(F.count(df.Value).alias('observati
ons')).orderBy(F.desc('observations'))
df_observations = df_observations.filter(df_observations.observations > 100).wit
hColumnRenamed('Variable', 'var')
df_observations.show()
```

```
+---------------+------------+
|            var|observations|
+---------------+------------+
|852-911_01_SPEED|      311683|
|      852-600_11|      298759|
|      852-911_01|      238934|
|      852-600_10|      184953|
|      852-YF-008|      164158|
|      852-YF-007|      164158|
|  852-911_01_TOT|      133981|
|         852-455|       32230|
|      852-YQ-002|         474|
|      852-YF-009|         113|
|      852-YT-001|         113|
+---------------+------------+
```

Convert top 10 observations to a list of variables names

So far we aggregated all observations to find those variables for which the count of value observations is above 100. The resulting information is a dataframe with two columns of which we want to extract just the names of variables.

In [110]:

```
variables =
sc.parallelize(df_observations.drop('observations').collect()).toDF()
variables.show()
```

```
+---------------+
|            var|
+---------------+
|852-911_01_SPEED|
|      852-600_11|
|      852-911_01|
|      852-600_10|
|      852-YF-007|
|      852-YF-008|
|  852-911_01_TOT|
|         852-455|
|      852-YQ-002|
|      852-YF-009|
|      852-YT-001|
+---------------+
```

Coding Playground

In [111]:

```
ref = variables.join(incidents).cache()
ref.show()
```

```
+---------------+-------------------+-------------------+
|            var|         start_time|           end_time|
+---------------+-------------------+-------------------+
|852-911_01_SPEED|2015-05-09 00:00:...|2015-05-15 23:59:...|
|852-911_01_SPEED|2015-05-25 00:00:...|2015-05-27 23:59:...|
|852-911_01_SPEED|2015-06-15 00:00:...|2015-06-19 23:59:...|
|      852-600_11|2015-05-09 00:00:...|2015-05-15 23:59:...|
|      852-600_11|2015-05-25 00:00:...|2015-05-27 23:59:...|
|      852-600_11|2015-06-15 00:00:...|2015-06-19 23:59:...|
|      852-911_01|2015-05-09 00:00:...|2015-05-15 23:59:...|
|      852-911_01|2015-05-25 00:00:...|2015-05-27 23:59:...|
|      852-911_01|2015-06-15 00:00:...|2015-06-19 23:59:...|
|      852-600_10|2015-05-09 00:00:...|2015-05-15 23:59:...|
|      852-600_10|2015-05-25 00:00:...|2015-05-27 23:59:...|
|      852-600_10|2015-06-15 00:00:...|2015-06-19 23:59:...|
|      852-YF-007|2015-05-09 00:00:...|2015-05-15 23:59:...|
|      852-YF-007|2015-05-25 00:00:...|2015-05-27 23:59:...|
|      852-YF-007|2015-06-15 00:00:...|2015-06-19 23:59:...|
|852-911_01_SPEED|2015-07-01 00:00:...|2015-07-05 23:59:...|
|852-911_01_SPEED|2015-07-15 00:00:...|2015-07-19 23:59:...|
|852-911_01_SPEED|2015-08-28 00:00:...|2015-08-30 23:59:...|
|      852-600_11|2015-07-01 00:00:...|2015-07-05 23:59:...|
|      852-600_11|2015-07-15 00:00:...|2015-07-19 23:59:...|
+---------------+-------------------+-------------------+
```

In [112]:

```
from pyspark.sql.functions import *

same_var = col("Variable") == col("var")
same_time = col("Time").between(
    col("start_time"),
    col("end_time")
)

ref2 = ref.join(df.alias("df"), same_var &  same_time)
ref2.show(10)
```

```
+----------+--------------------+--------------------+-------------
------+-----+----------+
|       var|          start_time|            end_time|
  Time|Value|  Variable|
+----------+--------------------+--------------------+-------------
------+-----+----------+
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 22:
35:...|  7.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 20:
55:...|  3.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 19:
15:...|  1.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 17:
35:...|  1.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 15:
55:...|  1.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 14:
15:...|  1.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 12:
35:...|  1.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 10:
55:...|  1.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 09:
15:...|  1.0|852-YF-008|
|852-YF-008|2015-08-28 00:00:...|2015-08-30 23:59:...|2015-08-30 07:
35:...|  1.0|852-YF-008|
+----------+--------------------+--------------------+-------------
------+-----+----------+
```

In [ ]: