> *"Apache Spark is an open-source cluster computing framework originally developed in the AMPLab at UC Berkeley. In contrast to Hadoop's two-stage disk-based MapReduce paradigm, Spark's in-memory primitives provide performance up to 100 times faster for certain applications. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is well suited to machine learning algorithms."*

Have a look at the official documentation (`https://spark.apache.org/documentation.html`) and examples (`https://spark.apache.org/examples.html`). Again, you're encouraged to use Google!

We will use the WDC tick data from `http://www.kibot.com/Buy.aspx#free_historical_data`. You will find the 1.1GB `WDC_tickbidask.txt` file in `/cs/bigdata/datasets`. You will also find a 4.6MB file named `WDC_tickbidask_short.txt` that only contains data for the first few days; working with this file is much faster. Both files contain comma-separated values:

```
09/28/2009,09:10:37,35.6,35.29,35.75,150
09/28/2009,09:30:06,35.48,35.34,35.64,14900
09/28/2009,09:30:06,35.49,35.37,35.61,100
09/28/2009,09:30:10,35.4,35.4,35.57,100
09/28/2009,09:30:10,35.48,35.43,35.53,200
[...]
```

According to `http://www.kibot.com/Buy.aspx#free_historical_data`:

> *"The order of fields in the tick files is: Date,Time,Price,Bid,Ask,Size. Bid and Ask values are omitted in regular tick files and are aggregated from multiple exchanges and ECNs. For each trade, current best bid/ask values are recorded together with the transaction price and volume. Trade records are not aggregated and all transactions are included."*

In `assignment4.tar.gz`, you will find a file named `stocktick.py` that contains the declaration of a Python class named `StockTick` that will be used to represent a record:

Contents of `stocktick.py`:

```python
class StockTick:
   def __init__(self, text_line=None, date="", time="", price=0.0, bid=0.0, ask=0.0, units=0):
      if text_line != None:
         tokens = text_line.split(",")
         self.date = tokens[0]
         self.time = tokens[1]
         try:
            self.price = float(tokens[2])
            self.bid = float(tokens[3])
            self.ask = float(tokens[4])
            self.units = int(tokens[5])
         except:
            self.price = 0.0
            self.bid = 0.0
            self.ask = 0.0
            self.units = 0
      else:
         self.date = date
         self.time = time
```

```
        self.price = price
        self.bid = bid
        self.ask = ask
        self.units = units
```

Question You will also find a file named `stock.py` that you are asked to complete. The objective is to compute various statistics on the tick file. Read all comments in the file carefully:

Contents of `stock.py`:

```
import sys
from stocktick import StockTick
from pyspark import SparkContext

def maxValuesReduce(a, b):
   ### TODO: Return a StockTick object with the maximum value between a and b for each one of its
   ### four fields (price, bid, ask, units)

def minValuesReduce(a, b):
   ### TODO: Return a StockTick object with the minimum value between a and b for each one of its
   ### four fields (price, bid, ask, units)

def generateSpreadsDailyKeys(tick):  ### TODO: Write Me (see below)
def generateSpreadsHourlyKeys(tick): ### TODO: Write Me (see below)
def spreadsSumReduce(a, b):          ### TODO: Write Me (see below)


if __name__ == "__main__":
   """
   Usage: stock
   """
   sc = SparkContext(appName="StockTick")

   # rawTickData is a Resilient Distributed Dataset (RDD)
   rawTickData = sc.textFile("/cs/bigdata/datasets/WDC_tickbidask.txt")

   tickData =  ### TODO: use map to convert each line into a StockTick object
   goodTicks = ### TODO: use filter to only keep records for which all fields are > 0

   ### TODO: store goodTicks in the in-memory cache

   numTicks =  ### TODO: count the number of lines in this RDD

   sumValues = ### TODO: use goodTicks.reduce(lambda a,b: StockTick(???)) to sum the price, bid,
               ### ask and unit fields
   maxValuesReduce = goodTicks.reduce(maxValuesReduce) ### TODO: write the maxValuesReduce function
   minValuesReduce = goodTicks.reduce(minValuesReduce) ### TODO: write the minValuesReduce function
```

```
avgUnits = sumValues.units / float(numTicks)
avgPrice = sumValues.price / float(numTicks)

print "Max units %i, avg units %f\n" % (maxValuesReduce.units, avgUnits)
print "Max price %f, min price %f, avg price %f\n"
       % (maxValuesReduce.price, minValuesReduce.price, avgPrice)

### Daily and hourly spreads
# Here is how the daily spread is computed. For each data point, the spread can be calculated
# using the following formula : (ask - bid) / 2 * (ask + bid)
# 1) We have a MapReduce phase that uses the generateSpreadsDailyKeys() function as an argument
#    to map(), and the spreadsSumReduce() function as an argument to reduce()
#    - The keys will be a unique date in the ISO 8601 format (so that sorting dates
#      alphabetically will sort them chronologically)
#    - The values will be tuples that contain adequates values to (1) only take one value into
#      account per second (which value is picked doesn't matter), (2) sum the spreads for the
#      day, and (3) count the number of spread values that have been added.
# 2) We have a Map phase that computes thee average spread using (b) and (c)
# 3) A final Map phase formats the output by producing a string with the following format:
#    "<key (date)>, <average_spread>"
# 4) The output is written using .saveAsTextFile("WDC_daily")

avgDailySpreads = goodTicks.map(generateSpreadsDailyKeys).reduceByKey(spreadsSumReduce);   # (1)
#avgDailySpreads = avgDailySpreads.map(lambda a: ???)                                      # (2)
#avgDailySpreads = avgDailySpreads.sortByKey().map(lambda a: ???)                          # (3)
avgDailySpreads = avgDailySpreads.saveAsTextFile("WDC_daily")                              # (4)


# For the hourly spread you only need to change the key. How?

avgHourlySpreads = goodTicks.map(generateSpreadsHourlyKeys).reduceByKey(spreadsSumReduce); # (1)
#avgHourlySpreads = avgHourlySpreads.map(lambda a: ???)                                    # (2)
#avgHourlySpreads = avgHourlySpreads.sortByKey().map(lambda a: ???)                        # (3)
avgHourlySpreads = avgHourlySpreads.saveAsTextFile("WDC_hourly")                           # (4)

sc.stop()
```

When you are done editing the file, you can try running your program. To this end, you must first load Spark:

```
$ module load natlang
$ module load NL/HADOOP/SPARK/1.0.0
```

If you get an error message saying that `load` was not found, you can try to run `source /etc/profile` and run the command again. You can then run your script using:

```
$ spark-submit --py-files stocktick.py stock.py
```

With the `WDC_tickbidask_short.txt`, the output should be:

```
[...]
Max units 394500, avg units 189.727741
Max price 37.120000, min price 34.520000, avg price 36.110827
[...]
```

With the following contents for the files that contain the daily and hourly spread:

```
$ cat WDC_daily/part-00000
2009-09-28, 8.7461956441e-05
2009-09-29, 9.62263125455e-05
2009-09-30, 0.000103149255577
2009-10-01, 9.09907962315e-05
2009-10-02, 0.000109917689058

$ head WDC_hourly/part-00000
2009-09-28:09, 0.000133175829341
2009-09-28:10, 9.21066196523e-05
2009-09-28:11, 7.37038962707e-05
2009-09-28:12, 7.80151688106e-05
2009-09-28:13, 6.903254691e-05
2009-09-28:14, 7.01609887022e-05
2009-09-28:15, 6.81955681382e-05
2009-09-28:16, 0.00329163416378
2009-09-29:09, 0.000168793690811
2009-09-29:10, 0.000124472420975
```

What results do you get for `WDC_tickbidask.txt`?