Petal Data Engineering Problem Set

Write a script to process the *transactions[1-3].csv.gz* files.

These files each contain bank and credit card transactions for 100k users (not real Petal users). Each row has 7 columns and is properly formatted (although you will need to infer the exact csv formatting params yourself).

1. user id
2. account id
3. debit/credit (debit means negative amount. credit means positive amount)
4. amount (this field should always be positive)
5. Random string with tricky characters to parse (where a transaction description would normally go)
6. Date
7. Another random string with tricky characters to parse

The desired output is a csv with the columns:

1. user_id
2. Number of of transactions for user
3. The sum of transaction amounts for the user (use exactly 2 decimal places)
4. The min balance (running sum) for the user at the end of any day (use exactly 2 decimal places)
5. The max balance (running sum) for the user at the end of any day (use exactly 2 decimal places)

And with users in the original input order.
Use the header: “user_id,n,sum,min,max” and don’t use any extra whitespace so we can easily run a diff with our reference solution.

Even though these example files are small, you should pretend they have too much data to fit in memory. But you can assume that the transactions for a single user will fit in memory. So your script should do one of the following:

1. Stream the data (you can use the fact that the files come grouped by user id) so that you don’t run out of memory.
2. Create a barebones in-memory MapReduce framework with a memory efficient mapper and reducer. The framework piece can be as inefficient as you want :)

Parallelism (Bonus):

1. If you chose streaming then add coarse grained parallelism so that multiple files can be processed at once. But a single file is processed in a single threaded way. You should have one output file per input file.
2. If you chose MapReduce then make sure you can run as many mappers as there are input files and a bunch of reducers. You should have one output file per reducer (don’t worry about the output order of the users if you choose MapReduce).

Things to watch out for:

1. Users on the edge of files must be handled somehow. Decide what to do with them and write a comment.
2. Make sure that all balances are output using exactly 2 decimal places.
3. Parallelism in Python is kind of weird. Make sure you’re actually getting good CPU utilization.