miniHive

Milestone 3

Compiling Relational Algebra to MapReduce Jobs

In the third milestone, we compile relational algebra queries into a physical query plan of MapReduce jobs. The MapReduce jobs can then be executed directly on Hadoop.

1. Read the chapter on "Workflow Systems" for MapReduce engines in chapter 2.4.1 of the book "Mining Massive Datasets". The Python module `luigi` is such a workflow engine that can execute MapReduce jobs (among many other things). `luigi` **is already installed in the course VM**.

   A good starting point are the `luigi` examples on GitHub: `https://github.com/spotify/luigi/tree/master/examples`, e.g. the `hello_world.py` file.

   If you are interested in details, our setup is inspired by `luigi/examples/wordcount_hadoop.py`. However, **you are not expected to dig deep into** `luigi`. Understanding (and appreciating) *what* it does for you should be enough.

   If you want to code outside of the VM, on our personal computing device, make sure you have `luigi` installed.

2. In the VM, you need a file `luigi.cfg` in the folder where you are running your luigi tasks, with these contents:

   ```
   [hadoop]
   streaming-jar=/usr/lib/hadoop-mapreduce/hadoop-streaming.jar
   python-executable=/usr/local/bin/python3.6
   ```

3. Add code to the provided Python module `ra2mr` which compiles a relational algebra query into a MapReduce workflow. We make the following simplifying assumptions:

   - We assume that the queries are the output of milestone 2 and therefore use the operators σ, π, ρ, and $\bowtie_p$ (Equi-join) only. (We do not implement the cross product as a MapReduce job, it doesn't make much sense for *big* data.)
   - We assume that our input consists of "flat" JSON documents (no arrays or nested objects). Here is the data for relation `Person` from the *pizza* example. Each line is a key-value pair, separated by a tab. The key is the relation name, the value is a flat JSON document.

     ```
     Person      {"Person.name": "Amy", "Person.age": 16, "Person.gender": "female"}
     Person      {"Person.name": "Ben", "Person.age": 21, "Person.gender": "male"}
     ...
     ```

In Moodle, you find the skeleton code for `ra2mr.py`. You only need to flesh out the code for Mapper and Reducer functions. The boilerplate code for building workflows with `luigi` is already provided!

A task parameter of type `ra2mr.ExecEnv` controls the execution environment:

- Set the task parameter `exec_environment` to `HDFS` to run tasks on Hadoop (the VM). This assumes that all input files reside in HDFS.
  This is our *production mode*. Unless you are willing to endure long waits, this mode is unsuitable for development. Use `LOCAL` for development, as described next.
- Set the task parameter `exec_environment` to `LOCAL` to run tasks without HDFS or Hadoop involved (or even installed) The pizza files are provided in Moodle.
  This is intended as the *development mode*: You will have quick turnarounds and can easily inspect any intermediate data written to temporary files.
- The unit tests set the task parameter `exec_environment` to `MOCK`. All files are then kept in main memory only. This is intended for *unit testing*.

This is how you would interact with `ra2mr` from within Python code:

```
import luigi
import radb
import ra2mr

# Take a relational algebra query...
raquery = radb.parse.one_statement_from_string("\project_{name} Person;")

# ... translate it into a luigi task encoding a MapReduce workflow...
task = ra2mr.task_factory(raquery, env=ra2mr.ExecEnv.HDFS)

# ... and run the task on Hadoop, using HDFS for input and output:
# (for now, we are happy working with luigi's local scheduler).
luigi.build([task], local_scheduler=True)
```

You can also execute `luigi` tasks from the command-line, as also described here `https://luigi.readthedocs.io/en/stable/running_luigi.html`. This is great for development and manual testing.

For instance, to evaluate a selection query locally on the VM, you can write

```
python3.6 ra2mr.py SelectTask \
--querystring "\select_{gender='female'} Person;" \
--exec-environment LOCAL --local-scheduler
```

or alternatively

```
PYTHONPATH=. luigi --module ra2mr SelectTask \
--querystring "\select_{gender='female'} Person;" \
--exec-environment LOCAL --local-scheduler
```

Inspect the `*.tmp`-files for intermediate results and the final output. Remember to clear any output files before starting the next task, since `luigi` will not recompute them.

Similarly, to evaluate a projection query locally on the VM, you can write

```
python3.6 ra2mr.py ProjectTask \
--querystring "\project_{name} Person;" \
--exec-environment LOCAL --local-scheduler
```

To run the queries on Hadoop, simply switch to the `HDFS` environment. Make sure that all required input files have been loaded into HDFS, and that any previous output has been cleared. (If one way of calling the tasks on Hadoop doesn't work, try the other one. Currently, I cannot explain why there are sometimes issues.)

4. Combine your code of all three milestones, to execute SQL queries in *miniHive*. The unit tests in `test_e2e.py` check this for you.

We will use special unit test modules called `pytest` and `pytest-repeat`. This allows us to repeat unit tests, to check for "flaky" tests. Tests may fail sporadically, often due to changes in the execution order, most likely when you compute the join. To check for flakiness, the unit tests in `test_ra2mr.py` will be run several times. (This is no guarantee, of course, for the absence of flakiness.) If you want to check this yourself, install `pytest` and `pytest-repeat` on your VM using `pip`.

1 Material in Moodle

- The skeleton code for `ra2mr.py`.
- The file `test_e2e.py` contains unit tests. Make sure your implementation passes these tests.
- The file `test_ra2mr.py` contains unit tests. Make sure your implementation passes these tests.

2 What to Submit

- Submit `ra2mr.py` with your extensions/implementation,
- and your implementation of the previous milestones.

Remarks: For Milestone 3, you are not asked to find any particular optimizations to make your implementation more efficient. All you are required to provide is a correct and clean implementation.
