Scaling Database Systems
Winter Term 2025/26
Chair of Scalable Database Systems
Prof. Dr.-Ing. S. Scherzinger

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

Milestone 3: MapReduce Translation

Compiling Relational Algebra to MapReduce Jobs

The goal of the coding project is to build a mini-version of Apache Hive, called *miniHive*. In the third milestone, we translate 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" (version 3, available online for free at `http://www.mmds.org/#ver30`). The Python module `luigi` is such a workflow system that can execute MapReduce jobs (among many other things). In the miniHive Docker container, `luigi` is already pre-installed.

   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 develop your code on our personal computer, outside the miniHive Docker container, make sure that you have `luigi` installed.

2. 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 does not 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, encoding one tuple. The pizza files are provided in Stud.IP.

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

   You find the input files (`*.json`) and the skeleton code for `ra2mr.py` in Stud.IP.

   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 (in the container). This assumes that all input files already 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). This assumes that all input files reside in the same directory as the `.py` files.

  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 mode 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 described here: https://luigi.readthedocs.io/en/stable/running_luigi.html. This is useful for development and manual testing.

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

```
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 refuse to recompute them.

Similarly, to evaluate a projection query while writing to the local file system, write

```
PYTHONPATH=. luigi --module ra2mr ProjectTask \
--querystring "\project_{name} Person;" \
--exec-environment LOCAL --local-scheduler
```

To run 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.

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

Remarks: For Milestone 3, focus on *correctness* rather than efficiency.

All you are required to provide is a correct and clean implementation that

- passes all Praktomat unit tests, and
- can execute the queries from the same unit tests in HDFS mode on the miniHive Docker container.

Praktomat will run the unit tests in `test_ra2mr.py` and `test_e2e.py` upon submission. Solutions that rely on hard-coding or other non-general methods will not be accepted by us. Secret tests in Praktomat are designed to detect such cases.

Upload `sql2ra.py`, `raopt.py` (from the previous milestones) and `ra2mr.py` as one submission to Praktomat: `https://praktomat.sdbs.fim.uni-passau.de` (accessible only within the Uni Passau network or via the Uni Passau VPN). As `ra2mr.py` relies on the output of `sql2ra.py` and `raopt.py`, you may encounter some problems introduced by your previous milestones. In this case, you may have to revise your previous milestone implementations accordingly.

After the Praktomat deadline has passed, your implementation will also be tested on the miniHive Docker container. Your implementation will be executed on Hadoop and data from HDFS will be processed: We recommend that you try this out beforehand (`--exec-environment HDFS`). Just because the unit test pass locally does not mean that the code runs on MapReduce, e.g., when your code relies on global variables.

Deadline: January 9, 2026, 12:00 (noon, daylight). A valid submission must pass all public tests in the `LOCAL` and `HDFS` execution environment, the plagiarism check, and must not be hard-coded.

Note: From January 8, 2026 - just a day before the Milestone 3 deadline - you can no longer use OpenVPN to access Praktomat and upload your submissions. Instead, you need to use eduVPN. It is your responsibility to ensure that you can hand in your submission timely through Praktomat, either by setting up eduVPN or by accessing Praktomat directly from the campus network (e.g., via eduroam) without a VPN.
