Revelio Labs Data Caching Assignment

Background

The Revelio Labs seniority model is used to assign standardized seniority levels to positions. The model takes in a company name and job title and assigns one of the following seven seniority levels to the position:

1. Entry Level (Ex. Accounting Intern, Software Engineer Trainee, Paralegal)
2. Junior Level (Ex. Account Receivable Bookkeeper, Junior Software QA Engineer, Legal Adviser)
3. Associate Level (Ex. Senior Tax Accountant; Lead Electrical Engineer; Attorney)
4. Manager Level (Ex. Account Manager; Superintendent Engineer; Lead Lawyer)
5. Director Level (Ex. Chief of Accountants; VP Network Engineering; Head of Legal)
6. Executive Level (Ex. Managing Director, Treasury; Director of Engineering, Backend Systems; Attorney, Partner)
7. Senior Executive Level (Ex. CFO; COO; CEO)

For the purpose of this assignment, you can assume that the seniority model exists behind a gRPC endpoint with the following specification:

```
service SeniorityModel {
        rpc InferSeniority (SeniorityRequestBatch) returns (SeniorityResponseBatch);
}

message SeniorityRequestBatch {
        repeated SeniorityRequest batch = 1;
}

message SeniorityRequest {
        int32 uuid = 1;
        string company = 2;
        string title = 3;
}

message SeniorityResponseBatch {
        repeated SeniorityResponse batch = 1;
}

message SeniorityResponse {
        int32 uuid = 1;
        int32 seniority = 2;
}
```

The uuid in the SeniorityResponse message will match the uuid in the corresponding SeniorityRequest message.

The API can sustainably process about one batch of 1000 seniority requests per second. Beyond that, it will start queuing requests and may take longer to respond.

Challenge

The purpose of this assignment is to infer seniority levels for a dataset of job postings.

There's a process that you don't control which scrapes job posting data and uploads a JSONL file to the following S3 folder once every minute: s3://rl-data/job-postings-raw/. The size of the individual JSONL files vary, but on average there are about 8M new lines produced per day.

The files are labeled by their unix timestamp in seconds, e.g. "1721183934.jsonl". Each line of the JSONL file has the following format:

{"url": "https://www.reveliolabs.com/job/97513c3d-b4a4-4bfb-b71e-f95d5482ed14/", "company": "Revelio Labs", "title": "Senior Data Engineer - Data Flow", "location": "New York, NY", "scraped_on": 1721183915}

Your job is to build a data augmentation service that reads JSONL files from s3://rl-data/job-postings-raw/, infers a seniority value for every job posting, stores the value in a new "seniority" field for each line, and outputs the data to another folder s3://rl-data/job-postings-mod/. Each line of the output JSONL should have the following format:

{"url": "https://www.reveliolabs.com/job/97513c3d-b4a4-4bfb-b71e-f95d5482ed14/", "company": "Revelio Labs", "title": "Senior Data Engineer - Data Flow", "location": "New York, NY", "scraped_on": 1721183915, "seniority": 3}

Considerations

A naive solution would involve hitting the gRPC endpoint once for every single line of every JSONL file. However, every call to the gRPC endpoint runs a costly model inference operation on the input data. Therefore we want to minimize calls to the gRPC endpoint to only run model inference when it's strictly necessary.

For instance, imagine we have the following file "1721183934.jsonl":

{"url": "https://www.reveliolabs.com/job/97513c3d-b4a4-4bfb-b71e-f95d5482ed14/", "company": "Revelio Labs", "title": "Senior Data Engineer - Data Flow", "location": "New York, NY", "scraped_on": 1721183915}

{"url": "https://www.reveliolabs.com/job/97513c3d-b4a4-4bfb-b71e-f95d5482ed15/", "company": "Revelio Labs", "title": "Senior Data Engineer - Data Flow", "location": "San Diego, CA", "scraped_on": 1721183915}

It would be wasteful to call the gRPC endpoint twice, since both rows in the file have the same exact data for "company" and "title".

Furthermore, imagine that several hours (or even several months) later we receive an additional file with the following rows:

{"url": "https://www.reveliolabs.com/job/97513c3d-b4a4-4bfb-b71e-f95d5482ed16/", "company": "Revelio Labs", "title": "Senior Data Scientist", "location": "New York, NY", "scraped_on": 1721184876}
{"url": "https://www.reveliolabs.com/job/97513c3d-b4a4-4bfb-b71e-f95d5482ed17/", "company": "Revelio Labs", "title": "Senior Data Engineer - Data Flow", "location": "Boston, MA", "scraped_on": 1721184876}

It would again be wasteful to call the gRPC endpoint twice, since we've already run the model inference operation on {"company": "Revelio Labs", "title": "Senior Data Engineer - Data Flow"}.

The crux of this assignment is to build an optimized data caching service so as to minimize calls to the gRPC endpoint. You have full freedom over how you build both the caching service and the data augmentation service. Your submission will be graded primarily on the efficiency of the caching service.

For additional context, we have processed about 1.8B scraped job postings in the past year. During that time we have observed 20M distinct (company, title) pairs.

Final Notes

- The S3 paths referenced in the assignment are not real.
- You can make use of 3rd party tools (open or closed source) for the caching service if you'd like, but you need to thoroughly justify why you've chosen them.
- The following guiding questions will probably be useful for evaluating the efficiency of your solution:
    - How long would it take to pass a single JSONL file through the full data flow?
    - How long would it take to pass a day's worth of JSONL files through the full data flow?
    - How much of the above runtime is spent on reading from the caching layer?
    - How much of the above runtime is spent on writing to the caching layer?
    - What is the CPU footprint of both services?
    - What is the memory footprint of both services?