11-792 Project Report

Nicholas Gekakis, Boyue Li

March 1, 2018

1 Overview

In this project, we are building a distributed question answering pipeline framework, which allows users to easily configure multiple modules, create complex pipelines and tune parameters automatically.

2 Requirements

2.1 Easy to configure and deploy

The framework should be easy to configure and deploy.

2.2 Save and resume

The framework should be save intermediate results so that it can be interrupted resume running at a later time.

2.3 Automatical parameter tuning

The framework should be able to automatically tune some parameters that have been exposed to the system by the pipeline developers.

2.4 Easy to develop users' modules

The framwork should support a easy way for users to develop their own modules.

2.5 Automatical parameter tuning

The framework should support distributed parallel processing to handle large datasets and complicated pipelines (e.g. pipelines with several components).

2.6 Automatical load balancing

The framework should automatically handle load balancing since different modules need different excution time.

3 Design

3.1 Overview

As shown in Fig. 3.1, a pipeline is constructed from several independent modules. A module reads data from data server, processes data according to all possible parameters, then save results for each configuration to the data server.

The number of instances of each module is managed by the load balancing server.

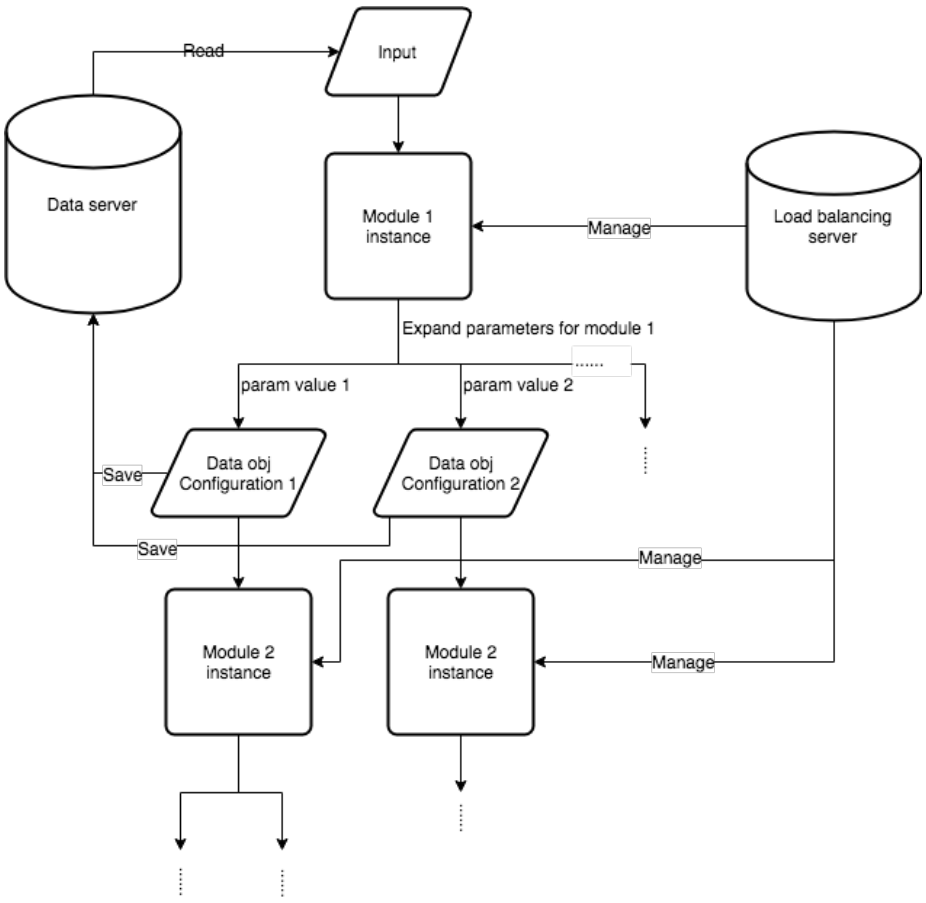

Figure 1: Control flowchart.

Every module runs on an independent process, and communicates through RabbitMQ using data objects which contains parameters, current excuetion status and the path to input file. Fig. 3.1 describes the information flow. Load balancing server distributes jobs to different modules' instances. Once the job is finished, the instance send a confirmation to the load balancing server.

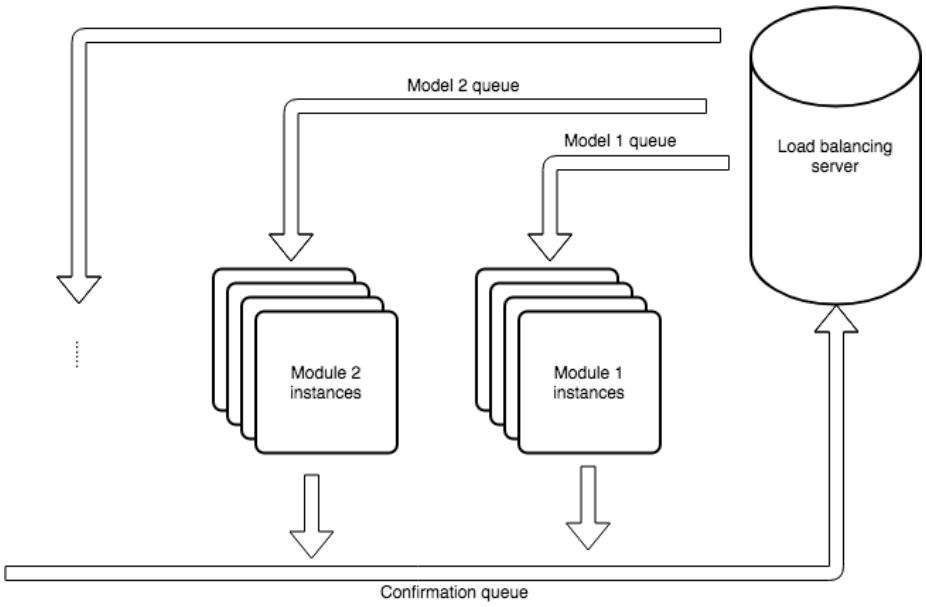

Figure 2: Information flowchart.

Users only need to specify

1. The connections between modules
2. The parameters every module needs

The framework will automatically handle execution, load balancing and parameter tuning.

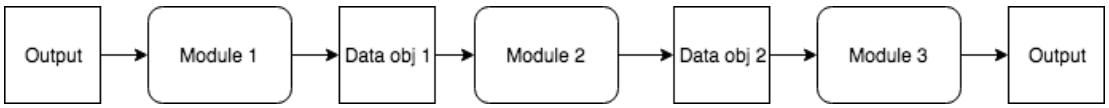

Figure 3: A sample pipeline

Figure 3.1 shows a sample pipeline which passes the input information object through some modules and produces the output information object.

3.2 Pipeline

The pipeline class manages modules and parameters. It reads configuration file, creates the pipeline and controls load balancing when running.

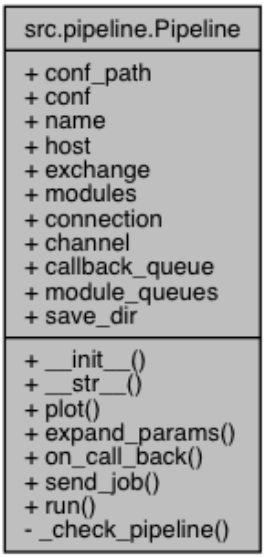

Figure 4: UML diagram for the pipeline class.

3.3 Module

A module is the basic computation unit which takes an input and produces an output. Every input and output is an information object defined in sec. 3.5.

A module needs to maintain the following fileds:

- Name of the module.
- Input module.
- Output module.
- Parameters.
- Configuration of the pipeline.

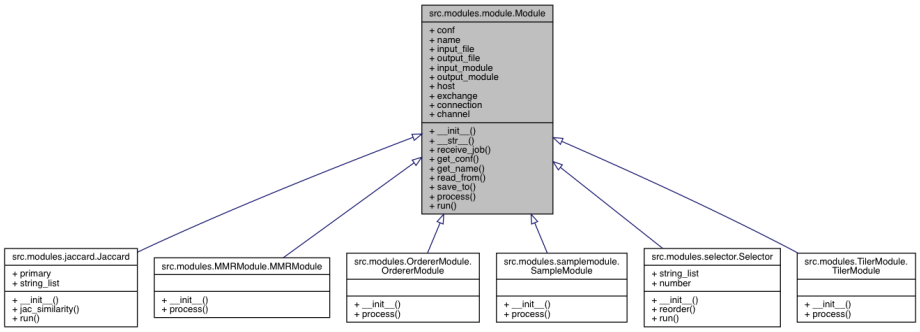

Figure 5: UML diagrams for the abstract module class and derived classes.

3.4 Parameter

The parameter class manages one parameter. It should handle all operations related the parameter, including updating the parameter value according to its step size, set and reset the value.

It also needs to save maintain the following fileds:

- Name of the parameter.
- Default values of the parameter.
- Tuning interval of the parameter.
- Step size of the parameter.

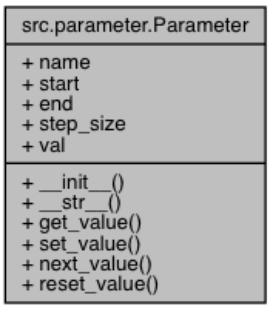

Figure 6: UML diagram for parameter class.

3.5 Data

The data class is the information object used to pass data between modules. It maintains the following fileds:

- Producing module: the module that produced this information object.
- Consuming module: the module that this information object to be passed to.
- Data path: the path to actual data file.
- Configuration: the configuration that produced the data object.

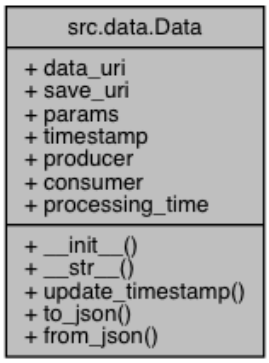

Figure 7: UML diagram for data class.

3.6 File format

A data file has a description contains the following fileds:

- Data: the actual data.
- Configuration: the configuration produced the file it.
- Timestamp: the timestamp when it is created.
- Producing module: the module that produced this file.

3.7 Configuration file

We use YAML files to configure the framework.

4 Example pipeline

5 Toy example

This pipeline consists of three sample modules, which do nothing but add it own configuration and parameters to the data to show this pipeline is working.

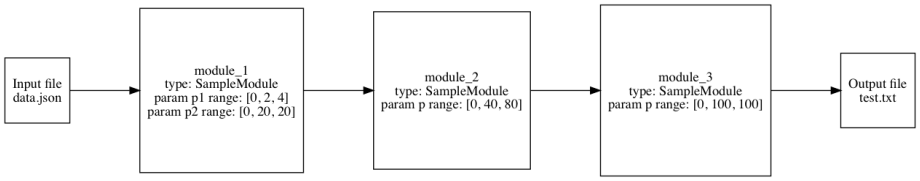

Figure 8: The structure for toy pipeline

6 BioAsq example

This pipeline is a fully operational BioAsq pipeline.

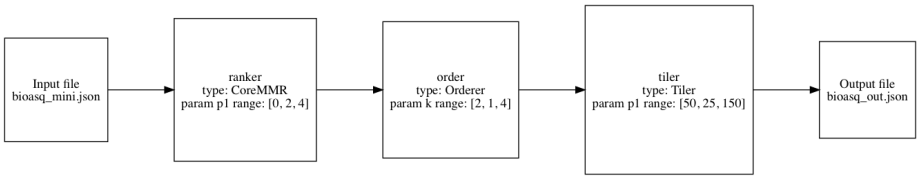

Figure 9: The structure for BioAsq pipeline

7 Experiments

7.1 Dataset1

7.2 Dataset2

7.3 Dataset3

8 Conclusion

Acknowledgements