DocuTrace

F20FC: Industrial Programming
Coursework 2

Sam Fay-Hunt — sf52@hw.ac.uk

December 5, 2020

Contents

The report should have between 10–15 pages and use the following format (if you need space for additionalscreenshots, put them into an appendix, not counting against the page limit, but don't rely on the screenshotsin your discussion)

1 Introduction

The DocuTrace application is a moderate size, data-intensive application, its purpose is to analyse and display document tracking data from the website issuu.com. The website hosts a substantial number of documents, and provides anonymised usage statistics, the data is provided in the form of a sequence of individual JSON entries seperated by new lines.

It is assumed the users of an application like DocuTrace would be someone with a high enough degree of technical competency to use simple Linux command line applications, the user would likely be a researcher (data science), or a business. The prior assumption leads to the assumption that the hardware running this application would be closer to server class than standard consumer hardware with higher CPU core counts and alot more RAM. To try and compensate for the potential scale of the data a signifantly large amount of RAM is not mandatory to run this application, but a pool of aproximately 8GB of RAM should be installed on the system at a minimum when processing 3 million lines otherwise a significant performance penalty may be incurred.

DocuTrace was written in Python 3, it is intended to be run on Ubuntu 20.04, and has not been tested on other operating systems.

Comprehensive documentation has been generated, a list of dependencies, installation and run instructions are provided, note the recommended entrypoint of the application is via the "docutrace" shell script, this script will automatically configure an environment variable to specify the output directory of graph files generated.

2 Requirements Checklist

Here you should clearly show which requirements you have delivered andwhich you haven't.

3 Design Considerations

Here you should clearly state what you have done to your application tomake it more usable and accessible.

Extensive documentation has been provided for other developers wishing to extend this codebase, additionally the `PEP8` sytle guide has been utilised to aid readability and help ensure consistent coding style.

In addition to the requireed parameters specified in the CLI requirement some secondary parameters have been included, this includes controls to log verbosity, an argument to limit the volume of data displayed in the GUI and command line, and a parameter to exit the application early (used when only a single task is needed).

The potentially long loading times when procesing large datafiles motivated the inclusion of an animated loading bar to provide some feedback to the user.

A logger is used to handle debug, info, warning and exception messages. The logger sends messages to stdout so they can be managed by journald, this includes exceptions and traceback messages this is a common practice and lets system admins handle the log output as they see fit. The verbosity of the log message can be set with the -v parameter, by default all messages of logging level WARNING and above are displayed.

4 User Guide

Use screen shots of the running application along with text descriptions to help youdescribe how to operate the application.

5 Developer Guide

Describe your application design and main areas of code in order to help another developer understand your work and how they might develop it. You may find it useful to supplementthe text with code fragments.

The provided documentation includes a definition of expected parameter types for all functions and methods. Additionally primative type stubs have been added to the function and method signatures to improve readability. More comprehensive type information could be added in the future by definig custom type stubs for all the classes, and union types.

5.1 Summary of application flow

This application can be broken down into 4 parts: Analysis, Gui, Utils packages and the main.py script; main.py is the primary entrypoint of the application, Analysis module handles all the data processing needs of the application, Util provides convienience functions like logging, exceptions, validation and also a task selection script, and finally the Gui module handles all functionality related to the graphical user interface.

The main entrypoint (DocuTrace/main.py) is a script to define the arguments, checks the validity of the filepath (a thread is started to begin processing the data right away) and task identifier from the command line, finally it starts the logic that parses the task and runs the given task once the data has finished processing.

Once processing is finished the task is launched, this will open the GUI. An instance of the ComputeData class is instantiated useing the data obtained by the DataCollector class, it is then passed to the gui, this class handles all iteractions between the gui and the data.

Once the gui opens the selected task should be visible on the screen, with the data visible, at this point new parameters for the various tasks can be provided to view other datapoints related to this task, or the number of instances displayed can be modified, depending on the task.

5.2 Analysis package

5.2.1 FileRead

The FileRead module handles all interaction with the filesystem with reguards to opening and closing the file. The core function used to read from the file is the `stream_file_chunks()` function, this function creates an iterator and laziliy reads the file on demand. There are 3 classes within this module, ParseFile, JsonProcessContextManager, and JsonParseProcess. A file is actually read by the `parse_file()` method of the ParseFile class, when concurrency is enabled it uses the JsonProcess-ContextManager to add chunks of the JSON data file to a JoinableQueue [1] managed by the context manager, see Fig. 1. The context manager starts all the processes, and maintains 2 queues: the aformentioned queue that recieves JSON chunks, and a queue to recieve processed data back from each process know as the feedback_queue.

```
1         with JsonProcessContextManager(data_collector, max_workers) as jtcm:
2             for chunk in self.file_iter:
3                 jtcm.enqueue(chunk)
4             logger.debug('Finished queueing chunks')
```

Figure 1: Context manager to start Processes and enqueue chunks from the JSON file, inside the ParseFile class.

The JsonParseProcess subclasses pythons Process [1] class to get around the global interpreter lock, it takes chunks from the JSON queue, processes the chunk and then adds a DataCollector class to the feedback queue until all items have been processed. There is substantial room for optimisaion in this class, deepcopy is utilised liberally as a workaround to prevent race conditions, however this adds substantal overhead to the processing time (almost double) for collection of data.

When the context manager main thread is finished enqueueing JSON chunks it starts a thread to deque and merge the feedback queue back into a single instance of the DataCollector class.

5.2.2 DataCollector

The DataCollector module features 3 convienience classes (ReadingData, BrowserData, and DocLocation) to help with internal data representation in the DataCollector class. The convienience classes all overload the `__add__()` method which is used when the merge_dict() function is used, additionally ReadingData and BrowserData overload `__eq__()` and `__lt__()` methods which when combined with the `@total_ordering` decorator from functools [2] allows all comparison operations to be performed.

The constructor of the DataCollector class itself has parameters to disable collection of a specific metric if desired, this can be useful to speed up testing, the path paramter is important; it is passed to an instace of ParseFile (see Section 5.2.1). During instantiation the constructor builds a list of methods that will be used by the ParseFile class to process the JSON data once the `gather_data()` method is called.

5.2.3 ComputeData

The ComputeData class contains several helper functions indluding some sorting functions and funcstions to get country and continent names from an Alpha2 country code.

5.2.4 Plots

5.3 Gui

5.3.1 main

5.3.2 Tab

5.3.3 Tasks

5.4 Utils

5.4.1 Exceptions

5.4.2 Logging

5.4.3 Tasks

5.4.4 Validation

6 Testing

Show the results for testing all cases and prove that the outputs are what are expected.Preferably, use unit testing to test core functionality of the implementation. If certain conditionscause erroneous results or the application to crash then report these honestly.

7 Personal Development

A short discussion on lessons learnt from the feedback given on CW1and a discussion how you integrated this feedback into CW2. Cover both coding and report writing,possibly more (project management, preparing for interview style questions etc). Lessons learnt from the experience of CW1: Started out using test driven development

Feedback from CW1: — code — Less code duplication too much global state limited input validation no custom exceptions not restrictive enough access modifiers

— report — intro: should cover short spec cover goals cover env

dev section: should discuss class dependencies should discuss method interfaces := params/return, assumptions of args

conclusion: should discuss adv lang features

8 Conclusions

Reflect on what you are most proud of in the application and what you'd have likedto have done differently. You should reflect on the produced software, and compare software devel-opment in scripting vs. systems languages. Most proud of: - Concurrency during file reading - Structure of the code, good decoupling front and back end

Do differently: - Leave more time to work on the gui - Use a different library for the gui - subclass the datacollector class to break it into smaller tasks

A References

References

[1] (). "Multiprocessing — Process-based parallelism — Python 3.8.6 documentation," [Online]. Available: https://docs.python.org/3.8/library/multiprocessing.html?highlight=process#module-multiprocessing (visited on 12/05/2020).

[2] (). "Functools — Higher-order functions and operations on callable objects — Python 3.9.1rc1 documentation," [Online]. Available: https://docs.python.org/3/library/functools.html (visited on 12/05/2020).