Exam project Word Finder

Concurrent Programming

*DM*519

Mark Jervelund - mjerv15 - mark@jervelund.com

IMADA

April 25, 2016

Specification

In the WordFinder project the task was to design and implement a program that looks for strings in .txt files using concurrency.
The program has 3 main functions.

Findall:
Find all occurrences of words and return a list with the result interface containing where they where found.
FindAny:
Find the first occurrence of a word and return its location and line using the result interface
Stats:
The stats function returns as object stats this object has 6 sub-functions
occurrences: returns how often a word occurs
foundIn: returns findall for a word
mostFrequent: returns the most occurring word in all the files scanned
leastFrequent: Returns the least occurring word in all the files scanned
words: Returns a list of all words found in the files:
wordsByOccurrences: Returns a list of all words sorted by how often the occur, with the least occurring first, and the most occurring last.

Short description of program

I designed two different functions to handle the two main functions in this project, One for findAll, Findany, and Stats.Foundin, and one for the rest of the stats object.
Findall, FindAny, and Foundin.
This function crawls directories using a DirectoryStream and passes the path it gets into a logic loop. This either calls itself if its a directory, or one of the two functions for handling small or large files.
These two functions then split the files into lines which are checked for a word using Regular expression. and then adds the results to a Arraylist.
Findall and foundin returns this Arraylist when the problem is done, and FindAny stops the executer as soon as a result is found and returns that.

Stats is mainly being handled by a hashmap.
Its designed almost the same way as the other functions except its not looking for a pattern but its just adding all the words to a hashmap container the word and a integer counting how many occurrences of the word there is.
The stats needed for the stats object are simply generated on the fly after the hashmap is made by simply looping over the hashmap to get occurrences, mostFrequent, leastFrequent, words, and wordsBy-Occurrences.

Concurrency implementation

The concurrent implementation of the program was made using a fixed pool executor with a pool og cores + 1. as it made very scalable to n cores, while performances scaling linear or close to from my testing.
The way i made the program Thread safe was by using a Concurrent hashmap for the map, and synchronizing on the Arraylist when adding results to it.
How ever this design does also have some problem, as its very RAM heavy peaking at around 4.3 GB doing my testing on a 3.4 Gigibyte dataset. but also being prone to problems with the garbage collector when handling very large datasets.
An other problem with my design is that the only why i have of figuring out when the execution of all tasks have been finished is by using a AtomicInteger to count how many tasks i start and then decrement it as the tasks finish which gives me some overhead.

Problems with implementation

One of the major problems with this implementation is that its very hungry ram wise, and will ingest data as fast as its allowed disk wise, and this can cause some problems if you have a limited around of ram if you do not have the processing power to back-up the data ingestion.

Testing

Runtime:
The runtime of my program was pretty good compared to what i have expected, it processes around 200 megabytes of data per second.
Accuracy:
For testing the accuracy of the program i shared my test dataset with other students on the course to figure out if my program got the results that it should. and it did. Stability:
For stability testing i tested how the program would preform when trowing large datasets at it, in the region of 3 Gigabytes ad it preformed as it should most of the time, it did how ever have some problems with garbage collection and Heap size.
Scalability:
From the testing i have done, the program benefits from an increasing amount of thread. however, this does vary a lot depending on how big the target files are and how they are structured.

Conclusion

From what i have tested program i have designed is far faster than a single threaded implementation but far faster.