FAST DATA STRUCTURES IN NON-C SYSTEMS LANGUAGES

Jeyhun Abbasov, **Amirabbas Sherafatian**

Institute of Computer Science, University of Tartu

UNIVERSITY OF TARTU 1632

The Problem

It is a common and proven idea to build data structures (or some area of an application) using another programming language performing better and then connect those parts together. In this project, it has been tried to implement the Trie data structure in Go to be consumed in Python.
The main challenge is to discover and bring it to the practice how to share codes as libraries written in Go to Python. It means, together with implementing concepts in both language, we need to go further to publish the code from Go to Python.
Behind this, we analyze time complexity and space complexity for this specific Trie data structure implemented in both Go and Python to appreciate if it is reasonable to do implementation in Go or it makes sense to keep up with the pure Python.

Method

The method is to implement a scenario (finding patterns in a text with the use of a Trie) in two approaches. Then we will be able to make a comparison between those two with the respect of time and space complexity.
In the first approach, the scenario is wholly implemented in python, and then we calculate the time and space complexity.
In the second approach, we try to implement the Trie in Go as a separated library. Then we import it into python client application to consume the Trie implemented in Go. Since, the Python already has a library to import libraries written in C, we use a package CGO in Go - which make us able to export the code in Go to C, to establish a C based bridge between Go and Python. Ultimately the client app in python would take a benefit of the already made Trie in Go to store the keywords and afterwards finding them in a text.

Results

We are looking at a text that length is 599179. It is clear that pure Python implementation of trie is faster than go implementation of the same trie. However, when it comes to space complexity go implementation is less memory consuming than python implementation. (Fig. 1)

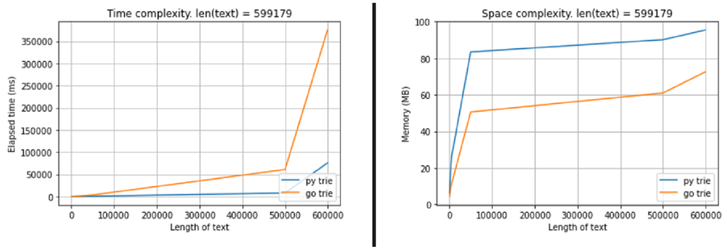

Fig. 1: Trie in Go vs Python. Time and Space complexity,

Remarks

Based on our results, it can be seen that the second approach (a mix of Go and Python) helps to have a better memory usage and we got better space complexity; However, a rise in time complexity in this approach is noticeable. As a suggestion for further work, Goroutines can increase the speed of the execution time.
Behind this, since we are using a C based bridge, considerable number of data types conversions from Go to C and C to Python or vice versa are happening. As a result, the use case does matter to which approach to choose – we should try to keep the data types conversions as less as possible. Truly, if we try to take the second approach for the applications with many request to Go side, it will increase the number of data types conversions and execution time consequently.

References

[1] Miki Tebeka. Python and Go, extending python With go, packaging python code, using python in memory. June 2020. https://www.ardanlabs.com/blog/2020/06/python-go-grpc.html
[2] Andrea Stagi. Extending Python with Go. Jan 2020. https://dev.to/astagi/extending-python-with-go-1deb
[3] Rene Manqueros. Running Go code from Python. Apr 2021. https://medium.com/analytics-vidhya/running-go-code-from-python-a65b3ae34a2d