Get Going With Google Go!

Zach Luciano , Tanner Halcumb, Nick Mladineo

April 2020

1 Introduction

With the rise of the internet over the last 30 years, information has never been more accessible. The entirety of mankind's knowledge is quite literally at your fingertips. Despite the availability of information, a large number of people get their news from a single source. These sources are curated by individuals who, whether intentionally or not, have political biases that can creep into the information presented to the reader. *Breitbart* and *NBCNews* are examples of websites with political bias. People often reject information that does not align with their worldview.

Our project, dubbed GoScraper, aims to help consumers of news find the middle ground between two opposing viewpoints. GoScraper works to break down the barrier of politically driven news sources by displaying well known, politically-driven news articles from extremely differing sources side-by-side for the user to interact with. This provides a unique perspective to the discrepancies that surface from the same story being told by individuals seeing the same thing through different lenses. In doing so, we hope to open the eyes of the public toward the political imbalance of the information we are fed everyday.

2 History

Go, or Golang, was built out of dislike for languages like C and C++. A group of three Google engineers spearheaded the development of the language. They liked many aspects of available languages such as the readability of Python, and the speed of C. Though, this was a time where the C family of languages was falling behind the times as computers had become faster, and little was being done to optimize them for faster processing. These languages are complex and often rather heavy on computer resource use. The group's ultimate goal was to make a language that incorporated the best aspects from prior languages. They dreamt of new language which allowed for multi-core use, better resource management, and integrated garbage collection.

Over time, work was started on creating the language Go, which has shown

some of the highest growth since it was released in 2009. Some aspects that have made Go so popular are its speed, readability, package management, and its support for networked and multi-core computing. The compiler written for Go boils the language down to C then Assembly. This allows it to run extremely fast as there are not many steps between Assembly and machine code. The designers of the language loved how Python was easy to read due to it omitting semi-colons and brackets. They found brackets helpful to show the block system they used for scoping, and decided to keep those around but did away with items such as variable-type identifiers and semi-colons. This led to code that was easy to read and maintain.

Go is written under the GCC license, which the developers were rather avid about. They wanted this to be a project the community got involved in and not just something to be used at Google alone.

3 Thoughts About Go

3.1 Why Go?

Go is a very interesting language, it represents an amalgamation of the best aspects from popular languages. It brings about modern programming with speed and efficiency of some of the first high-level programming languages. Go was the perfect choice for the purpose of our web scraper as it is extremely fast, and allows for easy multi-threaded programming. We were able to utilize the service they call GoRoutines (multi-threaded execution) to efficiently grab and store links from web pages at the same time as checking the website for more links. This means there is no time wasted to store information before the program is looking for its next piece of data. This is what makes our project stand out, the fact that there is only a couple of seconds of wait time from initial execution to when it is able to display what it found.

3.2 Great things about Go

Go is an excellent language for those looking for quick execution, multi-threaded systems, or just looking for a modern equivalent of the C language. Go perfectly handles even the hardest of tasks, making it useful for any system. We loved how easy it was to include packages in the code and use methods from different files within the package. Due to Go having built in multi-threaded processing, it was relatively straight forward for us to scrape an entire page without without having to waste time waiting for the links to be stored in the database before reading the rest of the website.

3.3 Quirks

One of the things that is cumbersome about Go is getting it setup initially. The setup of Go is more than just installing the language, the programmer also has

to ensure that their PATH files and GOPATH, GOBIN, etc. are setup correctly. If these are not setup then the system will not run. This became a problem when starting the project, and a lot of time was spent amongst the group trying to get everything to build appropriately. Another hurdle we encountered was in order to use packages the programmer has to get the packages using the "go get" command, if this is not done properly much frustration ensues from trying to figure out why the packages cannot be found.

Another aspect of programming that not everyone thinks about is exception handling. Go allows for exception handling, but in a peculiar way. It is possible for a function call to return more than one value. This additional value will frequently be any resulting exception from the call which should be verified to be null before continuing execution. This is not a traditional way of exception handling, and many programmers would find it verbose and weird to get used to.

4 Language Characteristics

4.1 Similarities and Differences to C++

Go was built to have similar qualities to the popular languages C and C++. The designers of Go favored the speed and reliability of these languages as well as much of the inner workings. Go compiles down to Assembly, just like the family of C languages does, which improves performance and usability of the language. Go does bring modern programming into the mix by including garbage collection, which leads to higher productivity and performance of the language.

Like the family of C languages, Go is a statically typed language. Static typing is a slightly more intuitive in Go than C, as when declaring a variable, the type is set to whatever the value assigned to it is, this will never change for the lifespan of the variable, unlike more modern languages such as Python. This leads to slightly higher compile times as there is less for the compiler to read in terms of syntax checking. Go also does away with semi-colons, many programmers find this useful as there are no more time-consuming hunts for a missing semi-colon and will greatly improve programmer efficiency in doing so.

4.2 Inner Workings of Go

One thing that every programmer runs into is where they are able to call different variables. This is the problem of how scope is interpreted for each language. Go follows a traditional block method for scoping. That means that when a variable is declared it is only available within the block that it was declared in. This extends to other blocks within the declaring block and any function calls made within its scope. This is a very important thing to know if the programmer is trying to declare a variable and use it in a higher block than it was

originally declared in, the compiler will not allow this as it will not know what to do with the information. This also has to do with the data management in Go. Go manages data using pointers, and if a variable is used out-of-scope, the garbage collector has already thrown away the pointer to that variable and it can no longer be found. It is also important to note that Go variable must be declared before use, if a variable is used before it is declared the system will throw an undefined error as there is no pointer to the variable yet.

5 Breakdown of GoScraper

5.1 Go Web Scraper

When we set out with our language selection, we considered the strengths of Go. With its efficient approach to multi-threaded processes, it is clear that we had a tool worthy of concurrently scraping web pages. Once we found a package suitable for web-crawling, GoColly, we began experimenting with the capabilities. Essentially, the package allows a collector to be instantiated. Once we have a collector, we provide it GoRoutines to carry out upon execution. These included gathering HTML tags, attributes, or even beginning a spider to crawl the hyperlinks on a page. We initially realized how broad we could make the scope of this collector, as we could provide a depth parameter. This depth parameter would determine the number of layers to traverse. As a depth of one would remain on the start URL, a depth of two would traverse the first page as well as all the links existing on this page. This functionality is especially powerful, and the possibilities are endless. We experimented more with counting words, but ultimately, decided to narrow down our goal. We had been testing on news sites and began to discuss the problems with mainstream media presenting their differing perspectives. This gave us the idea to try and match articles from opposing sides of the political spectrum to demonstrate the discrepancies. Next we looked to build GoRoutines to gather article details. This started with mapping article headlines that existed at depth one, or the home page of various news resources. We would tell our collector to run with a given URL ("foxnews.com"), then upon seeing specific HTML tags, we would store the text contents of this attribute. The HTML tag to search depended on the structure of the news page. For one of our original examples, Fox News, the parent attribute we desired was the "H2" tag, and the details existed in the child "a" tag. Once we could gather headlines from various news sources, with their links mapped, we began the matching debacle.

5.2 GoScraper Headline Matching

The headline matching portion of the app works by isolating the headlines from the rest of the URL for each article. The algorithm is meant to test every single article against another article, leaving no stone unturned. The first step of this process is to find an article based on the news source, so it first tries to find an

article from one source, then will go through the list again looking for a link from another source. Once the two opposing articles are chosen, it then picks apart the URL for each to isolate the headlines. When we have just the headlines, the article tests each word in one headline to every word in another headline, adding to a sum every time a word is matched. This data is then stored in a map to keep the links and sums together. The data is then picked through, looking for the key-pair that had the highest number of matching words. Once this key-pair is found, the links are then sent to a method that builds the "index.html" file for the dashboard to later read when it is built.

5.3 Scrapi

Running alongside our GoScraper is a RESTful API nicknamed Scrapi. Scrapi takes advantage of the built in JSON functionality within GoLang as well as the powerful HTML router package gorilla/mux. The API stores and retrieves headline and URL information from a postgres database running in Amazon Web Service. A snippet of code is below, demonstrating how straight forward setting up an API can be.

```
func main() {
    r := mux.NewRouter()
    r.HandleFunc("/", HomePage)
    r.HandleFunc("/articles", CreateArticle).Methods("POST")
    r.HandleFunc("/articles", GetArticle).Methods("GET")
    http.Handle("/", r)
}
```

In the sample code above, we have created a NewRouter which will handle all incoming requests, and have told it to perform the specific functions CreateArticle and GetArticle when the /articles endpoint is hit with either a POST or GET method respectively. All of this functionality is backed up by the incredible JSON support that GoLang has built in to the language. The JSON package has "decode" and "encode" functions which work alongside struct definitions to package information up in a a reliable manner. This simple, yet intuitive, way of interacting with HTML calls allowed us to implement a completely functional API with around half of the lines of code required for a java equivalent. Additionally, GoLang's aim for brevity when it comes to writing code makes the finished product much easier to read.

6 Conclusion

At the end of the day, our goal was to build a web scraper and match article headlines to give the user a more neutral perspective on a given news story. We accomplished this task and even added an API on top, Scrapi. Writing this project in Go played to its strengths in the sense that we wanted to process lots of information efficiently. By sifting through all the links and articles on a

news home page, there is plenty of weight to process. However, by providing GoRoutines to occur when reaching a specific HTML tag, we are able to tame endless data retrieval. Even though the language can handle it, for our purposes it was helpful to have so many parameterized options to direct web scraping duties. For these sorts of benefits, we could see ourselves using Go for future projects when dealing with heavy loads of data.

7 References

"Effective Go." *Go*, golang.org/doc/effective_go.html.

Scully, Ethan. "Go vs C++: A Comparison." *Career Karma*, 11 Jan. 2020, careerkarma.com/blog/go-vs-c-plus-plus/.

Łowicki, Michał. "Scopes in Go." *Medium*, Golangspec, 11 Aug. 2016, medium.com/golangspec/scopes-in-go-a6042bb4298c.