Vargha Hokmran

**A Short Review to Get Started with “Go” Language**

This paper is based on Alphabet Company’s official website “golang.org”, and its objective is to introduce their new language “Go” or “GoLang”. It will explain what new features it has, and how they can be beneficial for programmers, and companies. There are few main headlines and questions that this paper is trying to address. It starts with a little history of the project, and the brains behind that. Then it explains the necessity of this new language and the goals that Go tries to achieve. Finally, it would be spoken that how Go is the right answer to these necessities.

***History of the project***

On September of 2007, Robert Griesemer, Rob Pike and Ken Thompson, started the idea of Go, by sketching the goals for a new language. Quickly they made a plan, and started to design as a part time project. After about four months Ken started it on compiler, and after a few months the language became a full-time project. In the same year, Ian Taylor came to the group to do the GCC front end, and Russ Cox joined, and helped to make the language and libraries ready.

Alphabet publicly introduces this open source project on November 10, 2009. However, it took a couple more years of design and improvements until “Go 1” the stabilized version of Go was released on March 28, 2012 and it officially became reliable for different applications, products, and projects. The purpose of Go 1 is to provide a long-term stability, and the improvements would be focused on performance, reliability, portability, and new features like internationalization. The experience of Go 1 is also going to be used as an infrastructure for Go 2, which may come out after a few years.

***Disadvantages:***

Even though, there are some discussions around possible disadvantages of Go, the advantages look big enough to show them insignificant. For example, there is no support for generics in Go, which means you cannot write algorithms in terms of types to be specified later, and be instantiated when needed. However, there are some alternate ways to achieve the goals that other languages are using generics to solve them. Some of them these methods are well explained on “Applied Go” (<https://appliedgo.net/generics/>).

***Advantages***

It is not possible to fit all of the advantages in this short article. The following lines are going to focus on the most important factors by this paper’s writer:

**Speed, and Efficiency:** After many years of enhancements on computer hardware, there haven’t been much speed improvement on programming language structures. Unlike older programming languages, compiling a large Go code on a single machine takes only a few seconds. Go's type system doesn’t follow a hierarchy, and therefore, it doesn’t spend any time to define the relationships between different types. “Also, although Go has static types the language attempts to make types feel lighter weight than in typical OO languages.”

**Effect on Memory**

Let’s compare two codes, the first one in Java, and the second one an equivalent in Go:

Java: **integer myNum = new Integer (2014);**

Go: **var myNum int32 2014**

If we compare the Java code above for the variable *myNum*, which requires 16 bytes on a 32 bit JVM, and 32 bytes on a 64 bit JVM, for myNum, to the same variable in go Go, which takes exactly 4 bytes, we will see Go is very more efficient in memory.

You may argue that we can code it in Java as

**int myNum = 2014;**

,and it similarly takes only 4 bytes, but if you use it in a collection like list or map, the compiler would change it to an Integer object.

**Effect on CPU Cash**

Go is also more efficient for processors. It lets you create a compact data structures, which avoids unnecessary indirection, which leads to a better utilization on cash. Obviously, a better cache utilization would increase the performance of the program.

**Faster inlining**

Inlining is not unique to Go, but what is important is the way that Go uses inlining at the time of compiling codes. In older programs, the compiler make a call to the compiled function, but Go treats the called function as it is part of the caller function.

The argue against that might be that this kind of inlining is expensive because it increases the size of the binary file, but Go only does it for simple function candidates.

Let’s look at an example code:

**package maxPack**

**fun Max (a, b, int) int {**

**if a > b { return a }**

**return b**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package main**

**import “macPack”**

**fun Double (a, b int) int {return 2 \* maxPack.Max (a,b) }**

**Garbage Collection**

The mandatory system of garbage collection in Go, makes it more reliable, faster, and safer.

**Safety and Ease of Programming:** Programmers still have to pick between compilation efficiency, execution efficiency, or ease of programming. The purpose of Go is to combine the ease of programming of a dynamically typed interpreted language like Python and JavaScript, with the efficiency and safety of a compiled and statically typed language like C++ or Java.

**Garbage Collection:** Go has a full garbage collection system, which provides a fundamental support for concurrent execution and communication, while by the current popular languages garbage collection and parallel computation are not well supported.

**Multiprocessing:** Multiprocessing on multicore machines has a lot of worry and confusion. “Go proposes an approach for the construction of system software.”

**Cleaner Code:** A big part of software development is dependency management, and the “header files” of the C tradition don’t follow a clean dependency analysis, and they also reduce the compilation speed. Go’s has an easy software construction model, which makes dependency analysis much quicker and avoids the overhead of C-style include files and libraries.

**Unicode Identifier:** Go extended the space of identifiers from ASCII to Unicode. Unicode includes 17 planes of each 216, which makes it possible to directly use 1,114,112 characters, while ASCII only contains 256 characters. The code below is an example of using this feature:

**package main**

**import "fmt"**

**func main() {**

**fmt.Println("Hello, 世界")**

**}**

These are just a few benefits of this new language, and this young programming language looks to be a good choice of system programming. The idea behind Go is to combine the ease of programming with the efficiency and safety, so it can be a good replacement for interpreted and compiled languages. The power of Go make it a fast and safe solution for small and big networking and multicore computing projects.

Links Cited:

"Frequently Asked Questions (FAQ)." Frequently Asked Questions (FAQ) - The Go Programming Language. Google, n.d. Web. 19 Apr. 2017<<https://golang.org/doc/faq#What_is_the_purpose_of_the_project>>.

Pike, Rob. "1. Abstract." Go at Google: Language Design in the Service of Software Engineering. Google Inc, n.d. Web. 19 Apr. 2017<<https://talks.golang.org/2012/splash.article>>.

"Who Needs Generics? Use ... Instead!" Applied Go. Applied Go, 13 July 2016. Web. 19 Apr. 2017. <https://appliedgo.net/generics>.

Dave Cheney. "Five Things That Make Go Fast." Dave Cheney. N.p., n.d. Web. 19 Apr. 2017. <https://dave.cheney.net/2014/06/07/five-things-that-make-go-fast>.