Using the Go Programming Language in Practice

Erik Westrup and Fredrik Pettersson
Department of Computer Science, Lund University

June 5, 2014

Introduction

When developing software today we often use programming languages and tools that has been around for a very long time or we have a lot of experience with. This especially applies for companies where a change would mean huge investments in education of developers. However, it could be profitable to switch to a modern language in the long run. Using a modern language has the potential of making the development faster and more flexible, team work easier, bug finding less complicated, build times shorter and maintenance more viable.

New programming languages are invented all the time but there are a few languages that are more distinct than others. The Go Programming Language is one of these. Go was created and designed at Google and it became a public open source project in 2009. There are several reasons that make this language stand out. Go has built-in concurrency mechanisms. Because of this the problem of organizing many tasks running at the same time becomes simple. Other things that makes Go stand out is e.g. fast compilation, light syntax and constructs and garbage collection.

Go at Axis

This thesis was carried out at Axis Communications. Today Axis mostly develop their software using the C language but they are starting to glance at other, more modern, languages. The main reason being that they feel that their development process would benefit from it. The purpose of this thesis was to investigate the challenges of introducing Go as the main language for Axis in development of their software. This means not only investigating the language itself but investigating the whole software development process using Go. Furthermore, there are two different compilers for Go: The standard Go compiler and a front-end to GCC. Because the standard compiler does not support the architecture in many Axis products, another purpose for this thesis was to investigate different tools for cross-compilation, including the Go front-end to GCC.

Go and Its Tools

The first thing you notice about a language is its syntax. The syntax of Go will be familiar to programmers that are used to languages from the C-syntax family but it's clean like Python's. There is some differences though, for example type declarations in Go are done in same order as you read it from left-to-right, unlike C.

As described in the introduction, a big aspect of Go is its concurrency model. Built into Go there are constructs used to achieve concurrency including the so called goroutines which are lightweight threads, channels used for communication and the select statements for handling channel events. Compared to other languages like Java and C, with these constructs Go-developers can create concurrent programs in a natural way which clearly communicates what is going on.

The fast compilation for Go programs is achieved by using a smart dependency resolution method. When compiling a Go program, no dependency is ever resolved twice for example. Because of this, large Go programs are compiled in a few seconds on a single computer. On our desktop computer, the whole Go distribution with standard library is compiled in about 30 seconds.

A feature that could be very advantageous for companies with their codebase written in C is the possibility for C integration in Go. This feature makes it possible to import and use existing C-libraries. This means that when introducing Go, there is no need to rewrite entire applications. Instead, code replacement can be done in an iterated fashion.

Conclusions

After having used Go for a few months we have found some good things about it. We found Go to be really fun to use and easy to learn. Writing concurrent software feels very easy and clean when using goroutines and channels for communication. The language makes it easy to write object-oriented software using composition over inheritance. The build tools are simple to use (no need for any Makefiles or Autotools) and the builds are fast. The built-in package manager makes usage of third party libraries a lot easier. We found the Go community to be helpful and active when we took part in the discussions on the mailing lists and proposed patches to some bugs we found in the build tools.

During our work, the part we had the most trouble with was to get cross-compiling to work with up-to-date tools. Building a cross-compiling toolchain from scratch, we found, is still hard. To get it to work with the Go tools is even harder.

Go includes some basic data structures but not nearly as many as e.g. Java's collection framework. This could offend programmers that are used to have everything at hand. Another drawback of Go is the fact that debugging is currently not well supported.

Considering the good things and the bad things, we feel that Go is ready for personal use and for some corporate usage. However we feel that Go needs some more time for the tools to become more stable until it can be adopted in the embedded field of programming.