Building Correct Distributed Systems with the PGo Compiler

Finn Hackett, Shayan Hosseini
Renato Costa, Matthew Do, Ruchit Palrecha, Yennis Ye, Ivan Beschastnikh
University of British Columbia

Presentation abstract:
Distributed systems are difficult to design and implement correctly. In response, both research and industry are exploring applications of formal methods to distributed systems. For example, Amazon has reported using TLA+ and PlusCal to verify its web services. A key challenge in this domain is the missing link between the formal design of a system and its implementation. Today, this link is bridged through manual and error-prone developer effort.

In this talk we will overview Modular PlusCal (MPCal) language and the PGo[1] compiler toolchain. PGo translates MPCal models to PlusCal for model checking and also compiles MPCal models to executable Go code. This allows system designers to use MPCal to model and verify their distributed system designs and then use PGo to extract working implementations of these designs.

We have constructed several larger systems with PGo. For example, we built a distributed key-value store based on the Raft protocol, which is 5x faster than other similar verified Raft systems. Additionally, we built several eventually consistent systems based on conflict-free data types (CRDT). PGo makes verified distributed systems easier to build, providing at least a 3x reduction in development time as compared to prior work.

We first presented PGo at the TLA+ conference in 2019. Since that time PGo has grown in capabilities and has become more mature. In particular, PGo has a new Scala-based design, a more advanced runtime system, and a special focus on modeling and compiling fault-tolerant systems. PGo can now also verify and compile MPCal models separately and then compose these to form larger systems. We also use language-based fuzzing and model-based trace checking to make sure the PGo compiler itself is correct. In this talk we will include an overview of these newer developments.

Presentation outline:

- Briefly motivate formal verification and the complexity of debugging concurrent code
- Explain our project's goal: push developers to specify systems *before* writing code; have developers use PGo to derive an initial code prototype for free
- Motivate the challenges of translating PlusCal code into running implementations
- Explain key MPCal language features and how they allow implementation compilation
- Outline how PGo works at the high level
- Demonstrate a simple example in PGo
    - E.g., A proxy server that proxies the incoming requests to back-end servers
- Describe PGo's runtime architecture and how it provide the same atomicity semantics as models in MPCal
- Explain our experience with building a Raft-based key-value store with PGo
- Demo the Raft-based key-value store

[1] https://github.com/DistCompiler/pgo

    - Show PGo workflow in action
    - Briefly show our MPCal VSCode extension
- Explain the modularity features of MPCal specs
- Talk about our efforts to make sure that the PGo output is correct
    - Fuzz tests for the PGo compiler
    - Model-based trace checking
- Discuss outstanding limitations and ongoing work
    - Make MPCal easier to use, for example:
        - Allowing archetypes to be multithreaded
        - Separating the outputted TLA+ code from the MPCal code
    - Improve PGo compiler to output faster Go code, for example:
        - Implementing constant propagation
        - Implementing a reactive programming approach to avoid busy-waiting on `await` statements
    - Stronger verification of PGo output
    - Compiling MPCal models directly to TLA+ for a more efficient TLA+ representation
    - Building and verifying Byzantine Fault Tolerant systems with PGo