How to build a Distributed System with Real-time Constraints

Using concurrent Functional Programming tools

Armando Santos

December 2, 2022 —

INPUT | OUTPUT

Well-Typed
The Haskell Consultants

Table of Contents

Introduction

What we are doing

How we are doing it

Conclusion

Introduction

Networking Team

Armando Santos
Well-Typed

Marcin Szamotulski
Input Output Global

Duncan Coutts
Well-Typed

Neil Davies
PNSol

Peter Thompson
PNSol

What we are doing

Cardano Node

Ouroboros algorithm paper

- Formal Specification
- Threat models
- Non-realistic assumptions

Refining step

- Real time constraints
- Concurrency
- Operation & Performance

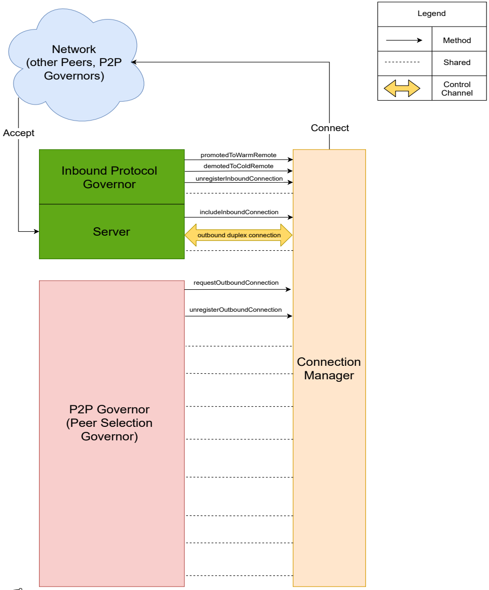
Implementation

- Architecture & Design
- Protocols
- Scale
- Exceptions & Corner cases

Testing

- Properties
- Simulation
- Reliability
- CI

Decentralised Network


Legend
Method
Shared
Control Channel
Network (other Peers, P2P Governors)
Accept
Connect
Inbound Protocol Governor
Server
promotedToWarmRemote
demotedToColdRemote
unregisterInboundConnection
includeInboundConnection
outbound duplex connection
requestOutboundConnection
unregisterOutboundConnection
P2P Governor (Peer Selection Governor)
Connection Manager

- Highly concurrent
- Reliable and Robust
- Predictable
- Manage resource consumption
- 3000 SPOs
- Has to run 24/7

More details

To read more about this, check out our documentation at: https://github.com/input-output-hk/ouroboros-network/

How we are doing it

Functional Programming

Strongly Statically Typed Purely Functional Programming with Haskell!

- Lazyness
- Type Safeness
- Referential Transparency
- STM
- Explicit effects
- More!

Typed Protocols

Internally developed (but open-source) library to specify end-to-end protocols at the type-level!

- Type Safe
- Session Types
- Deadlock free!
- Pure
- Powerful (pipelining out of the box)

QuickCheck

Property based testing framework for Haskell.

- Input random generation
- Shrinking
- Reproducibility
- Coverage checks

IO Simulator

Simulation monad that is a drop-in replacement for IO!

Internally developed (but open source) library to perform all kinds of IO Simulations, in particular:

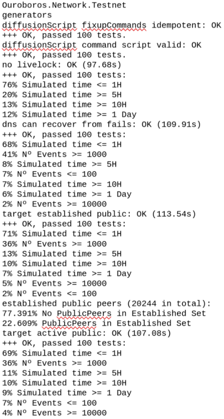
- write network simulations, to verify a complex networking stack
- write disk IO simulations, to verify a database implementation

IO Simulator allows…

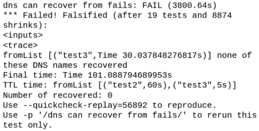
- Early detection of critical races
- Simulation of rare edge cases
- Mocking and error injection
- Simulate time passing
- Looking for different schedules

Most importantly:

- Allows for testing production code and
- Reproducing complex edge-case test failures



```
Ouroboros.Network.Testnet
generators
diffusionScript fixupCommands idempotent: OK
+++ OK, passed 100 tests.
diffusionScript command script valid: OK
+++ OK, passed 100 tests.
no livelock: OK (97.68s)
+++ OK, passed 100 tests:
76% Simulated time <= 1H
20% Simulated time >= 5H
13% Simulated time >= 10H
12% Simulated time >= 1 Day
dns can recover from fails: OK (109.91s)
+++ OK, passed 100 tests:
68% Simulated time <= 1H
41% Nº Events >= 1000
8% Simulated time >= 5H
7% Nº Events <= 100
7% Simulated time >= 10H
6% Simulated time >= 1 Day
2% Nº Events >= 10000
target established public: OK (113.54s)
+++ OK, passed 100 tests:
71% Simulated time <= 1H
36% Nº Events >= 1000
13% Simulated time >= 5H
10% Simulated time >= 10H
7% Simulated time >= 1 Day
5% Nº Events >= 10000
2% Nº Events <= 100
established public peers (20244 in total):
77.391% No PublicPeers in Established Set
22.609% PublicPeers in Established Set
target active public: OK (107.08s)
+++ OK, passed 100 tests:
69% Simulated time <= 1H
36% Nº Events >= 1000
11% Simulated time >= 5H
10% Simulated time >= 10H
9% Simulated time >= 1 Day
7% Nº Events <= 100
4% Nº Events >= 10000
```



```
dns can recover from fails: FAIL (3800.64s)
*** Failed! Falsified (after 19 tests and 8874
shrinks):
<inputs>
<trace>
fromList [("test3",Time 30.037848276817s)] none of
these DNS names recovered
Final time: Time 101.088794689953s
TTL time: fromList [("test2",60s),("test3",5s)]
Number of recovered: 0
Use --quickcheck-replay=56892 to reproduce.
Use -p '/dns can recover from fails/' to rerun this
test only.
```

Conclusion

Progress?

Complex systems spans performance characteristics we can not control.

We do our best in searching through all state space efficiently.

Functional Programming, namely Haskell and its concurrency tools helped us manage complexity.

Code reviewing is very efficient!

Far from perfect

We have had quite a few bugs, and we still do!

- 378 closed bug issues related with networking
- 276 open ones
- 10% of the issues are related with simulation environment
- About a handful of them were due to missplaced logging events

Our CI runs on average beetween 1 and 5 hours of simulated time per test per PR per OS. Which means:

- Assuming around 100 tests in our test suite
- Assuming 3 PRs per day
- Testing on Windows, OSX and Linux
- Results on 11 250 hours of simulation per week.

Far from perfect

We have had quite a few bugs, and we still do!

- 378 closed bug issues related with networking
- 276 open ones
- 10% of the issues are related with simulation environment
- About a handful of them were due to missplaced logging events

Our CI runs on average beetween 1 and 5 hours of simulated time per test per PR per OS. Which means:

- Assuming around 100 tests in our test suite
- Assuming 3 PRs per day
- Testing on Windows, OSX and Linux
- Results on 11 250 hours of simulation per week.
- All with a ~~20 member~~ 5 member team!

Hall of Fame

- Different scheduling found a edge case where state was being blindly overwritten
- Asynchronous exceptions on a blocking `finally` block
- Timeouts not being enforced withing reasonable bounds
- Pruning connections misbehavior in the presence of a TCP Simultaneous Open

Thank you!