SCALABLE AND RELIABLE VIDEO TRANSCODING IN HASKELL

Alfredo Di Napoli

Haskell Exchange 2015

ABOUT ME

Full story at: **http://goo.gl/qkKwKm**

```
launchMissile :: IO ()
```

iris

discover. develop. share.

IRIS CONNECT

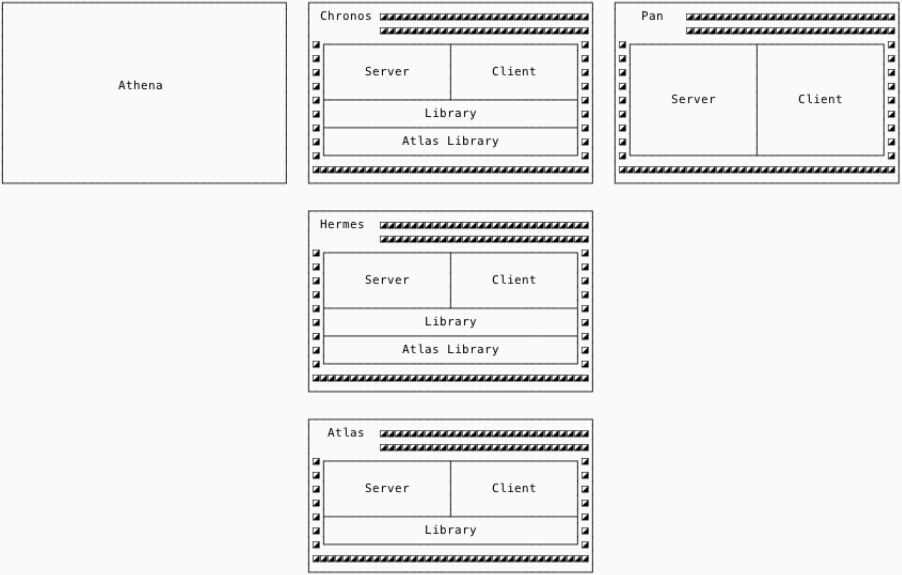
discover. develop. share.

Market leader for CPD solutions

Used by more than 1000 schools across UK, Europe, US & Australia

IRIS CONNECT (CONTD.)

IRIS' GREEK ZOO



Athena

Chronos

Server

Client

Library

Atlas Library

Pan

Server

Client

Hermes

Server

Client

Library

Atlas Library

Atlas

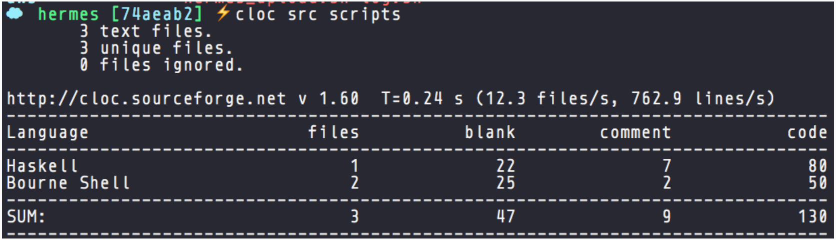
Server

Client

Library

HERMES' EVOLUTION

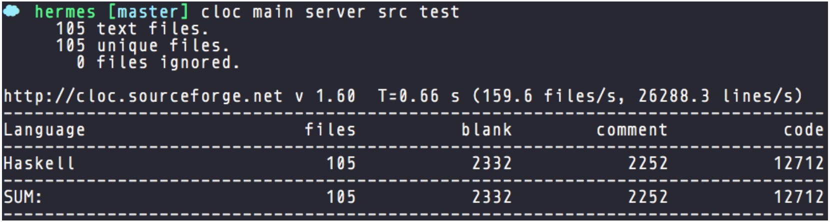
October, 2013



```
hermes [74aeab2] ⚡cloc src scripts
       3 text files.
       3 unique files.
       0 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.24 s (12.3 files/s, 762.9 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Haskell                          1             22              7             80
Bourne Shell                     2             25              2             50
-------------------------------------------------------------------------------
SUM:                             3             47              9            130
-------------------------------------------------------------------------------
```

October, 2015



```
hermes [master] cloc main server src test
     105 text files.
     105 unique files.
       0 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.66 s (159.6 files/s, 26288.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Haskell                        105           2332           2252          12712
-------------------------------------------------------------------------------
SUM:                           105           2332           2252          12712
-------------------------------------------------------------------------------
```

hermes' challenges

Upon taking the lead on Hermes, I was asked for a couple of requirements to be fullfilled, the most important one being that the system needed to be deployed in a cluster, capable of scaling according to demand.

hermes' challenges

More specifically, we wanted a system with these desirable properties:

- ∼ Scalable
- ∼ Fault tolerant
- ∼ Distributed

“OUT OF THE TAR PIT” DOCET

> The classical ways to approach the difficulty of state include OOP programming which tightly couples state together with related behaviour, and functional programming which — in its pure form — eschews state and side-effects all together. [..] We argue that it is possible to take useful ideas from both and that this approach offers significant potential for simplifying the construction of large-scale software systems.

In the same fashion, we have 2 different worlds colliding:

- ∼ We need to transcode videos, which is a very stateful operation
- ∼ As good Haskell programmers, we want to have components in our system to be as stateless as possible.

A SHARED NOTHING ARCHITECTURE (SN)

A shared nothing architecture (SN) is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system.

”All problems in computer science can be solved by another level of indirection.” - Butler Lampson

RABBITMQ

1. RabbitMQ was just the right tool for the job at hand:
    - ~ Easy to setup
    - ~ Can be configured to operate in a federation of nodes
    - ~ Extremely reliable
    - ~ Good Haskell bindings for it (*AMQP*)
2. A question genuinely arise: it seems extremely costly to shuffle video as binary blobs over the queues. Can we avoid that?

ABSTRACTION IS THE (MEDIA KEY)

```
root__m-stg-main-2014_10_29_13_27_26-videos-1-2333-vid-smc-oxz8dmdi1lx7fong
    ^    ^   ^        ^                 ^   ^   ^   ^   ^        ^
    |    |   |        |                 |   |   |   |   |        |
comment  |   |        |                 |   |   |   |   |        |
         |   |        |                 |   |   |   |   |        |
host ----+   |        |                 |   |   |   |   |        |
             |        |                 |   |   |   |   |        |
database --- +        |                 |   |   |   |   |        |
                      |                 |   |   |   |   |        |
dataset version ------+                 |   |   |   |   |        |
                                        |   |   |   |   |        |
resource (video or image) --------------+   |   |   |   |        |
                                            |   |   |   |        |
user ID ------------------------------------+   |   |   |        |
                                                |   |   |        |
video ID -------------------------------------- +   |   |        |
                                                    |   |        |
channel type ---------------------------------------+   |        |
                                                        |        |
video products -----------------------------------------+        |
                                                                 |
MAC (avoids submission of bogus keys) ---------------------------+
```

To be fair, the media key abstraction was already present in Atlas when I choose RabbitMQ, but it was the perfect fit for it!

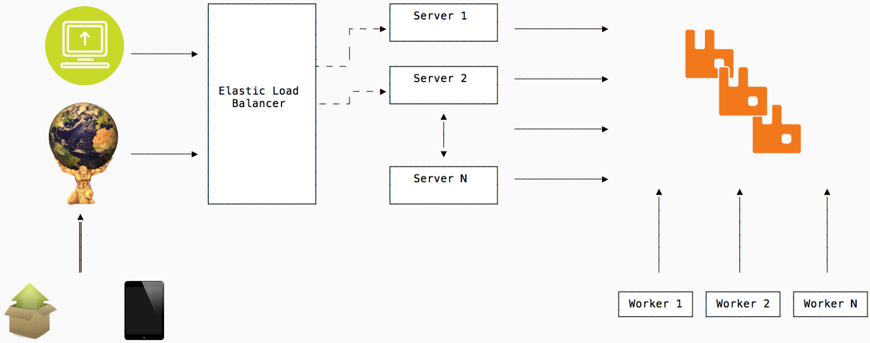
WHAT ABOUT SCALABILITY?

Fine, but RabbitMQ doesn't give you scalability...

1. We stood on the shoulder of giants - namely AWS' Auto Scaling Groups
2. Our very first native scaling algorith looked like:
    - ∼ Scaling up: Based on CPU% over time
    - ∼ Scaling down: Based on CPU% over time

It kept us going for a while...

THE ARCHITECTURE



1. Scaling up was too conservative and slow
    - $\sim$ It could take up to 15 mins to spawn a new worker
2. Scaling down suffered similar problems
3. The result was unoptimal customer experience (due to the slow turnaround time) and unoptimal for us (due to the additional costs incurring from poor scaling down)

What's the elephant in the room?

Why not use Cloud Haskell?

WHY NOT CLOUD HASKELL

1. CH encourages Erlang-style (i.e. actor based) communication, so nodes should know each other

   We do not want that!

2. Peer discovery would have been tricky in a dynamic environment where new machines born and die frequently
3. It wasn't mature enough in 2013, if not for a handful of companies using it

IRIS CONNECT'S STORY

A sharing and collaboration CPD platform for teachers via video recording, feedback and introspection.

1. Initially build with RoR, it was rewritten from scratch in Haskell (backend) and RoR + Angular.js (frontend)
   1.1 Effort initially started by my colleage Chris Dornan and Well Typed

2. The backend is composed by two main projects:
   2.1 The frontend-facing API server which holds the model and the business logic (**Atlas**)
   2.2 The video transcoding system (**Hermes**), a highly distributed and fault tolerant system, built on top of RabbitMQ.

WHY HASKELL?

Because software development is a marathon, not a sprint.

”It took me more time writing the specs that implementing the feature itself.”

Because we are like Shlemiel the painter.

"I can't help it," says Shlemiel. "Every day I get farther and farther away from the paint can!"

1. The more time it pass, the farther we get from our “paint can”, the mental model we built of the system.
2. In large scale systems, you can have parts that won’t be touched for *years*!
   - 2.1 How do you defend yourself when the refactoring or feature time comes?
3. A rich, strong and expressive type system can be your ultimate ally against complexity
   - 3.1 Things like `newtype`s and `ADT`s can help you cure common “diseases” like *Boolean Blindness*

As universe expands, so does the entropy in your software: use types to keep it at bay!

SOME “PROS” OF WORKING IN HASKELL

1. Refactoring is a dream
2. EDSLs are a piece of cake
3. Makes impossible states unrepresentable

REFACTORING IS A DREAM

1. The type system naturally guides you
2. In Haskell we tend to write small and generic functions
   - 2.1 Cfr. Bob Martin's "Clean Code"
   - 2.2 Most of the time they don't even break as they are written to work on polimorphic types
   - 2.3 Code reuse = profit!

So ultimately is not just about the strong type system, is about Haskell's (and Haskellers) natural tendency towards **composition** and **parametricity**.

EDSLS ARE A PIECE OF CAKE

```
fromPreset :: MediaFile -> MediaFile
           -> Maybe Atlas.VideoFilter
           -> VideoPreset -> Maybe VideoRotation
           -> LogLevel -> [T.Text]
fromPreset filename outFilePath flt vpres vi ll =
  let cli = ffmpegCLI $ mconcat [
              i $ toTextIgnore filename
            , loglevel ll
            , fromVideoPreset vpres
            , isVideoRotated vi <?> resetRotateMetadata
            , yuv420p
            , vf [rotateMb vi]
            , isJust flt <?> vf_technicolor
            , o_y_ext (toTextIgnore outFilePath) (Left vpres)
            ]
  in T.words cli
```

MAKES IMPOSSIBLE STATES UNREPRESENTABLE

Real world scenario:

```haskell
-- | Creates a new Supervisor.
-- Maintains a map <ThreadId, ChildSpec>
newSupervisor :: IO Supervisor

-- | Start an async thread to supervise its children
supervise :: Supervisor -> IO ()

-- | forkIO-inspired function
forkSupervised :: Supervisor
               -> RestartStrategy
               -> IO ()
               -> IO ThreadId
```

Example usage:

```haskell
main = do
  sup <- newSupervisor
  supervise sup
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

Can you spot a potential bug?

Nothing in the types is forcing us to call `supervise` before actually supervising some thread!

```haskell
main = do
  sup <- newSupervisor
  -- Wrong! We forgot to start the supervisor...
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

As Haskellers, we can certainly do better!

PHANTOM TYPES TO THE RESCUE!

Phantom Types allow us to “embed” constrain on our types, together with smart constructors.

```
data Uninitialised
data Initialised

data Supervisor_ a = Supervisor_ {
      -- record fields (omitted)
      }

type SupervisorSpec = Supervisor_ Uninitialised
type Supervisor = Supervisor_ Initialised
```

Let’s now slightly change our API to be this:

```haskell
-- | Creates a new Supervisor.
newSupervisor :: IO SupervisorSpec

-- | Start an async thread to supervise its children
supervise :: SupervisorSpec -> IO Supervisor
```

What did we get? Let’s try to run the “wrong” snippet again...

```
main = do
  sup <- newSupervisor
  _ <- forkSupervised sup OneForOne $ do
        threadDelay 1000000
        print "Done"
```

What did we get? Let’s try to run the “wrong” snippet again...

```
main = do
  sup <- newSupervisor
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

GHC will complain:

```
Couldn't match type Control.Concurrent.Supervisor.Uninitialised
         with Control.Concurrent.Supervisor.Initialised
Expected type: Supervisor
Actual type: Control.Concurrent.Supervisor.SupervisorSpec
```

1. This is because now we require a `Supervisor` to be initialised first
2. The type system prevented us making a silly mistake
   2.1 Failed with a very useful error message
3. Profit!

This is just a small example (this is only one of the possible solutions), but the benefits are real.

`https://github.com/adinapoli/threads-supervisor`

SNAGS OF WORKING IN HASKELL

1. Slow(ish) Compilation
2. Cabal Hell

slow(ish) compilation

SLOW COMPILATION: THE CAVEAT

1. It's a problem all non-interpreted languages have to deal with
2. GHC indeed does incremental compilation, building only what's changed
3. It's even slower if..
   3.1 You have TH (Template Haskell) in your code
   3.2 You are building with profiling enabled

If you want faster feedback loop, consider using ghci

It's the aggregate of more than one problem, which most of the time results in "I couldn't install package X (easily)"

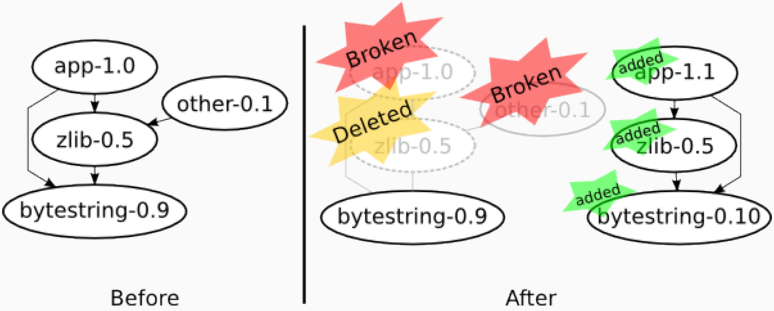


Figure 1: Image courtesy of Well Typed Ltd

CABAL HELL – THE SILVER LININGS

1. Sandboxes mitigate the issue

```
cabal sandbox init
cabal install
```

2. “Package aggregates” can help

   Stackage

   HaskellLTS

   Nix and NixOS

3. Broader solutions are in the pipeline

   Edward Z. Yang’s “Backpack”

Thank you.

Questions?

EXTERNAL REFERENCES

My road to Haskell

http://www.alfredodinapoli.com/posts/2014-04-27-my-road-to-haskell.html

Don Stewart - Haskell in the large

http://code.haskell.org/~dons/talks/dons-google-2015-01-27.pdf

Joel Spolsky - Back to Basics

http://www.joelonsoftware.com/articles/fog0000000319.html