(http://34cross.in)

From Zero to HIPster (Haskell In Production)

(TL,DR)
We're building a micro-service platform christened Hasura.io (http://hasura.io) (alpha release scheduled in summer 2015), and we used Haskell as the core programming language to build it.

This is a post for people who're not very sure about using Haskell in production, to convince them otherwise. We're not proselytising for conversion of huge enterprise codebase into complete Haskell. But a little bit of Haskell here and there is probably safe.

The post is bereft of any actual code samples and quantitative benchmarks (we'll keep adding them here as we go along).

Our Background

We're a team of developers with fairly diverse backgrounds. Most of our production experience had been with the mainstream languages like C, Java, Python, C++, C#, Javascript. When we set about the task of building a PaaS/BaaS, we started evaluating what sort of language and toolkit would be best suited for building the core platform.

What Attracted Us To Haskell:

To be really honest, we're a bunch of young people, and seeing something like this gets us really fired up:

> Haskell is an advanced purely-functional programming language. An open-source product of more than twenty years of cutting-edge research, it allows rapid development of robust, concise, correct software. With strong support for integration with other languages, built-in concurrency and parallelism, debuggers, profilers, rich libraries and an active community, Haskell makes it easier to produce flexible, maintainable, high-quality software.
> Source: wiki.haskell.org (https://wiki.haskell.org/Haskell)

Purely functional (whatever that meant at the time)! Twenty years of cutting edge research! Robust, concise and correct software! We'd already bumped into some conversations about how tight and helpful the Haskell community was supposed to be.

This was the stuff of engineering dreams. But Haskell was notorious for having a steep learning curve, was it worth putting business requirements at risk? Is it really as hard as people claim it to be? What is up with the category theory discussions that come up at the drop of a hat?

And then, we came across this: Beating the averages (http://www.paulgraham.com/avg.html)

The promise of something that's hard to use, and then works really well and increases productivity by a few orders of magnitude, is a pretty solid and sensible promise.

We all do use vim after all.

First steps with Haskell

After slightly more rigorous evaluation with some sample programs, we thus started working on projects (webapps in production) with some parts entirely written in Haskell. (Yay, microservices!). Some of the first real-world things we wrote included a ZeroMQ broker, automatically generating a typesafe CRUD and query API from a database schema, a redis-caching layer and more.

Some of the libraries we used during development were written by stalwarts in the Haskell and programming languages community. Haskell library authors tend to be spectacularly experienced, qualified and/or savvy people. Coupled with the nature of the language, this sort of means the code you write is on good foundations, safety and performance wise. Check the JSON parsing speed improvements (https://hackage.haskell.org/package/aeson) out as the library version numbers go up!

After each project, we felt we were getting better superlinearly. Our experiences carried forward elsewhere too. Our Python code became more functional, concise and easier to maintain.

A Real World Problem: Expose a subset of SQL in JSON with fine grained permissions

We needed performant libraries for setting up an HTTP server, JSON parsing and interfacing with Postgres. Easy AST manipulation, the ability to reason about the correctness, and easily adding new features were the prime requirements during development. Above all, we needed the final program to scale up well, and be efficient with the tons of IO it would have to do.

Requirements: Flexible codebase, Scale up, Handle IO well

Decisions decisions.

Language	Programming ease/desire	Memory footprint	Concurrency
Java	None	High	Mature
Node	Ok, but programming with callbacks	Ok	Very Immature
Haskell	Yes. Program 'synchronously' and GHC handles the rest.	Low	Mature

Python	Yes	High	Not easy

Objectively too, Haskell was the choice here.

The three performant libraries that we ended up with were:

- Warp (http://www.aosabook.org/en/posa/warp.html)- http server
- Aeson (https://hackage.haskell.org/package/aeson) - JSON parsing
- Hasql (http://nikita-volkov.github.io/hasql-benchmarks/) - Postgres connections

It was a breeze to rapidly prototype and test individual components before composing them into a whole. Midway through the project, we had to switch to a different Postgres library. The changes that we had to make were so localized and easy to reason about, that the entire transition took hardly a few programmer days.

Very surprisingly we got done really quickly. When we think back, we realize that the majority of the time we spent on the project was only on the problem we had set out to solve. The code in the project relates very closely to the actual JSON to SQL transformation rules we've formulated on paper, and Haskell quickly got everything else out of the way.

A Conclusion to the Ramble

To conclude this slightly rambly post, let's revisit some of the initial claims made by the maintainers of the language, and how it was useful to us in practice. There are so many discussions about how and why haskell. In fact, here's a good place to see it being thrashed out: What is haskell actually useful for? (http://stackoverflow.com/questions/1604790/what-is-haskell-actually-useful-for)

1. **Purely functional:** We could reason about our entire program, and the code reflected our reasoning almost exactly.
2. **Concise code:** Write code now, refactor soon. Smaller code lets you refactor larger portions easily (and hence frequently).

3. **Robust code:** Static typing with type inference, and a smart compiler (The Glorious Glasgow Haskell Compilation System).
4. **Testing:** As waxed poetic by everyone, the type system dramatically reduces a huge class of unit tests. Quickcheck makes another class of tests quite a breeze, by automatically generating test cases for assertions that you make about your functions. Most of what remains, is best tested manually anyway (eg: IO related failures).
5. **Concurrency with IO:**
    - GHC's threading abstractions and IO manager, makes it easy to think about concurrently executing computations and gives you all the benefit of event-based asynchronous runtimes like node
    - GHC has non-blocking IO (like node), supports real multiple threads and multicores naturally (unlike node) and works using events underneath (like node). It's not even a fair war.
6. **Rich libraries:** Haskell libraries are extremely easy to integrate and very very easy to just dig in and modify safely for your own use (and maintain forked versions).
7. **Deployment:** Private cloud deployments for Haskell code becomes really easy, because all library dependencies can be statically-linked. This makes version control and deployment a one step process.

Some common roadbumps

1. **Type errors** could be bizarre for a beginner, and parse errors are not helpful at all.
2. **Documentation:** Types are great for documentation when they are simple and concise. This is mostly the case, but in some libraries (ahem..lens), the type signature offers little to no documentation. In these cases, it takes a significant effort to use the library, but types finally make sense and are often what one returns to.
3. **Language extensions:** More often than not, when you're looking at a library in Haskell, you'll see use of GHC's language extensions. Some are very intuitive (OverloadedStrings, ScopedTypeVariables). Some are not (RankNTypes). This makes it harder to 鈥渆arn from the source鈥.

The Clincher

No matter what you work on, there is a joy in programming with Haskell. You feel better after you write progressively harder code. And progressively harder code keeps coming your way. There's so much working Haskell code out there, that you can just stare at days for and not really understand (but always use!).

The satisfaction we feel after a good day of Haskell is unparalleled, and at the end of the day, that's what it's really all about isn't it.

Andre Fernandes • 5 months ago

I love this language, it's very clean and powerfull. I used to program in C++, but now I love my functional language.
Good to know there's more people using it, here in Brazil (where I live) is so difficult to find Haskell programmers that sometimes I believe I am the only one.

Dulguun Otgon ↗ Andre Fernandes • 3 months ago

Yes, I even feel like I'm the only one who knows about this language in my country. Feels really isolated and hopeless sometimes. Finding job with Haskell is impossible in 10 years, I fear.

34crossblog Mod ↗ Andre Fernandes • 5 months ago

Yep. I think we're the only Haskell devs in the entire city sometimes. Programming in Haskell is a huge increase in programmer productivity for us and I think the entire ecosystem is getting better at a rapid rate. It's a great experience to be a part of the Haskell community at this stage and see these advancements happen.

msx • 5 months ago

"We all do use vim after all."
Right, now everything becomes clear :P

Now, back on track: why didn't you consider Go? Is there anything specially bad about it? (I mean for you)

34crossblog Mod ↗ msx • 5 months ago

:) We wanted a language that would increase developer productivity as much as possible. It seemed that a functional programming language, with a solid type system to ensure correctness was the way to go. Also, GHC supports green threads and uses multiple cores well, and supports powerful parallel programming constructs.
So Haskell seemed like clear choice.

However, we've not really used Go, so we can't really draw a comparison. What are your thoughts on Go? In what circumstances would you clearly choose Go over something else?

1

msx ↗ 34crossblog • 5 months ago

Hi, wonderful answer, thank you.

Well, it was more curiosity than anything else as Go is outright

designed for multi-threading operations -- and also has a more 'familiar' approach to programming than Haskell.

Said that, I like Haskell a lot and I think it's a good choice too. Best of luck guys.

• Reply • Share ›

shahidhk • 6 months ago

Looking forward for a great product :P And thank you for bringing me to the world of Haskell :P

• Reply • Share ›

34crossblog Mod ➦ shahidhk • 5 months ago

Thanks Shahidh! We're trying to get some interesting stuff going with Haskell and some web-dev in general too in Chennai. Are you based here?

--
Tanmai

• Reply • Share ›

Shanthakumar ➦ 34crossblog • 4 months ago

Seems like you folks have already started a Chennai Haskellers group. And its quite long for west-chennai-ties to reach out for meetups. Anyway good luck with your product :)

• Reply • Share ›

kp ➦ 34crossblog • 5 months ago

It's heartening to see real world services written in Haskell from Chennai! Hope you start a Haskellers group and share your real world experience developing and maintaining software written in Haskell