Haskell in the corporate environment

Jeff Polakow

October 17, 2008

Talk Overview

- Haskell and functional programming
- System description
- Haskell in the corporate environment

Functional Programming in Industry

- FP languages spreading into industry
  - Haskell, OCaml, F#, Erlang
- FP ideas spreading into mainstream languages
  - garbage-collection, generics, iterators
- FP already widely used
  - Javascript, Spreadsheets

Haskell in Industry

(haskell.org/haskellwiki/Haskell_in_industry)

- Finance
  - Credit Suisse, Standard Chartered
- Science & Engineering
  - Amgen, Eaton
- Contractors
  - Galois, Aetion

General Haskell Description

(from haskell.org)

- 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.

Haskell

- Language specification
  - Haskell 98 report
  - Common extensions
- Language implementation
  - Several compilers
  - GHC is only industrial strength one

haskell.org

Functional Programming

- Program is a function

  i.e. takes an input value and produces an output value

- Value can be many things
  - basic things, e.g. numbers, strings, etc...
  - pairs of values
  - functions
  - user defined, i.e. datatypes

- Recursion

Higher-order Functions

- Function which takes a function as input
- Modular design
- Code reuse
- Custom control structures

Purity

- No observable effects from execution of function
  - no mutable memory (i.e. can't change a variable's value)
  - no exceptions
  - no IO
- Effects added back in a controlled manner
  - can put pure code into effectful code
  - encourages modular design

  separate program logic from interaction with outside world
- Allows for interesting possibilities
  - possibility of heavy compiler optimization
  - simpler user interface for STM

Lazy Evaluation

- Only evaluate expressions which are used
  - call-by-need evaluation
  - requires different mode of thinking from most languages
- Infinite data structures (i.e. streams)
- Allows declarative style

```
enumerate xs = zip xs [1 ..]
enumerate ['a','b','c'] == [('a',1), ('b',2), ('c',3)]
```

Strong Static Types

- Guarantee some errors can't happen
- Help with refactoring
- Help structure program
  - Algebraic datatypes
  - Know possible structure of all values of a given type

```
data BinTree a = Leaf a
               | Branch a (BinTree a) (BinTree a)
```

Parametric Polymorphism

- Function works the same on all input types
  - fst :: (a , b) → a
  - length :: [a] → Int
- Code reuse

Ad-hoc Polymorphism

- Function overloading
- Function acts differently on different types
- Specify how function acts on each type
- Static type error to use function at unspecified type

Type Inference

- Most types can be inferred
- Type discipline is unobtrusive
- Can also explicitly specify type
  - good documentation
  - type checker will complain if inferred type conflicts

Haskell is a High Level Language

- Very expressive type system
- Powerful abstraction mechanisms
- Small gap between description and implementation

Syntax

- Can be very concise
- Optionally whitespace dependent
  - visually specify scope
  - removes need for lots of parentheses
- Can be addictive

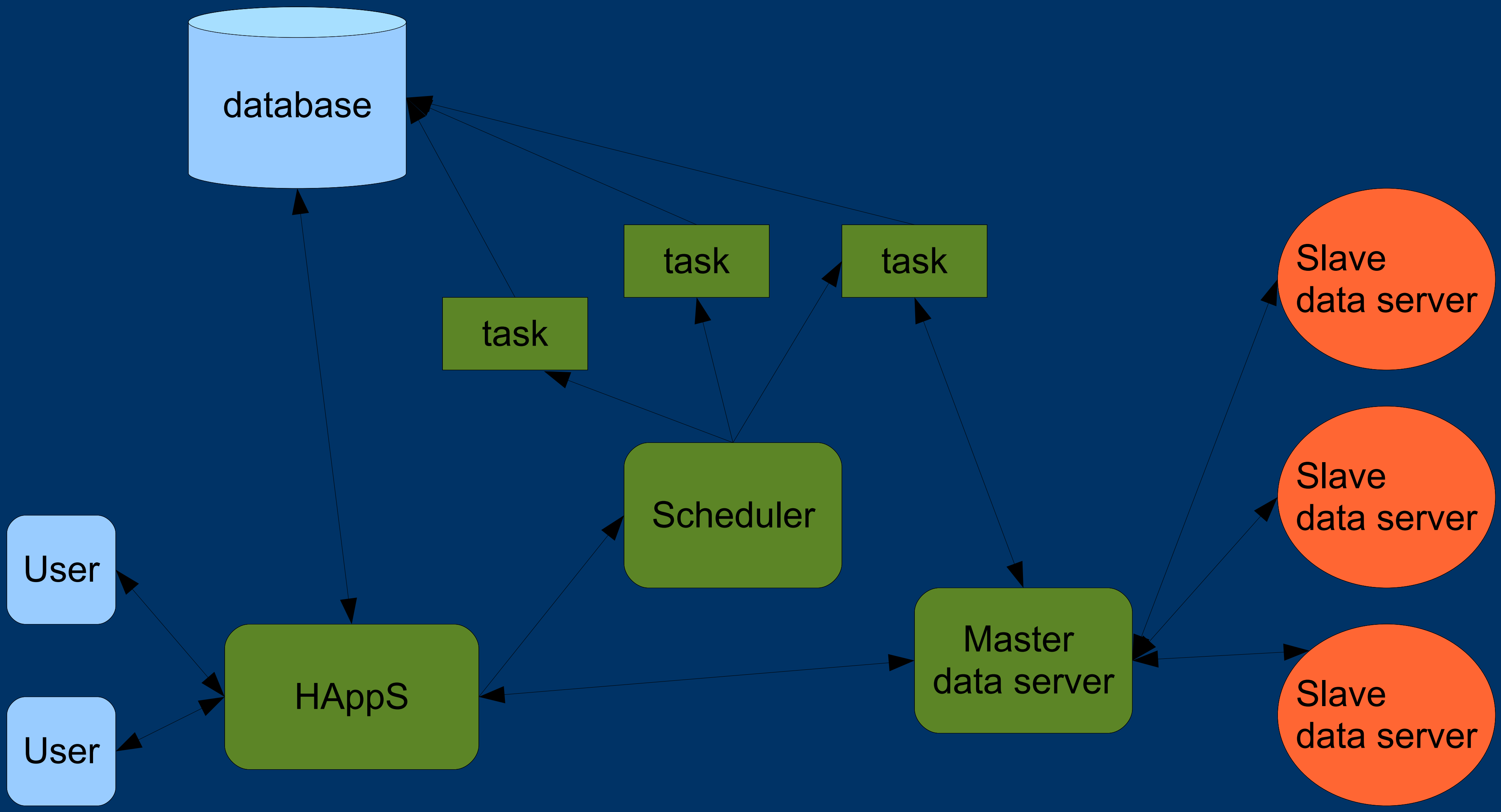
Real World Haskell System

- Small credit trading group
- Credit markets are opaque
- Information management is main task
- Quantitative analysis less important

System Overview

- Database and Web system
- Scheduler to spawn autonomous tasks
- Several communicating pieces
- Distributed over several computers

System Architecture



Novelties

- Statically typed tables with mini SQL DSL
  - Manipulate tables in memory
  - Generates SQL queries to create a table in memory
- Automatic generation of RPC wrappers
- Proc monad for logical process machinery
- Abstract (socket-based) server machinery

The Good

- Usual stuff
  - Types & type classes for static guarantees
  - First class (higher-order) functions for code reuse
- Purity
  - Able to upgrade old (poorly documented) code with relative ease
- Performance not an issue (for our purposes)

The Bad

- Upgrading to new Haskell implementation was painful
  - Some libraries don't like XP
  - Some libraries don't like cabal-install
- Errors / inadequacies of some libraries
- Most library documentation is poor

Useful Haskell Tools

- Database access tools
  - HDBC, Takusen, etc...
- Web tools
  - HAppS, powerful but difficult to install and learn
  - HSP, WASH, etc...
  - Curl bindings, FTP lib work pretty well
- Ability to write stable server-like programs
  - Great lightweight threads support
  - Good socket interface

Useful Haskell Tools

- Scripting
  - ghci as a shell, HSH
  - Good string processing machinery

- Foreign library interaction
  - FFI, plus helper tools, are good
  - No easy way to use .NET or Java libs

- Development Environment
  - GHC is easy to install & low maintenance
  - Libraries are not always easy to install
  - Available IDEs not adequate for everyone

Useful Haskell Tools

- Testing tools
  - Quickcheck
  - Smallcheck
  - HUnit
- Step Debugger
- Memory use Profiler
- Haskell community
  - haskell irc
  - haskell-cafe

Is Haskell ready for use in the corporate environment?

Yes

- But it helps if you are
  - free to try drastically new things
  - capable of functioning without IT dept support
  - a seasoned Haskell programmer
  - comfortable with laziness/strictness trade offs
  - comfortable reading library source code
  - capable of understanding and fixing linker errors

For more info: haskell.org