High Performance Web Server in Haskell

2011.7.12

IIJ Innovation Institute Inc.
Kazu Yamamoto

My goal

Modular Network Programming
on
Highly Concurrent Environment

Today's talk

Will not talk about

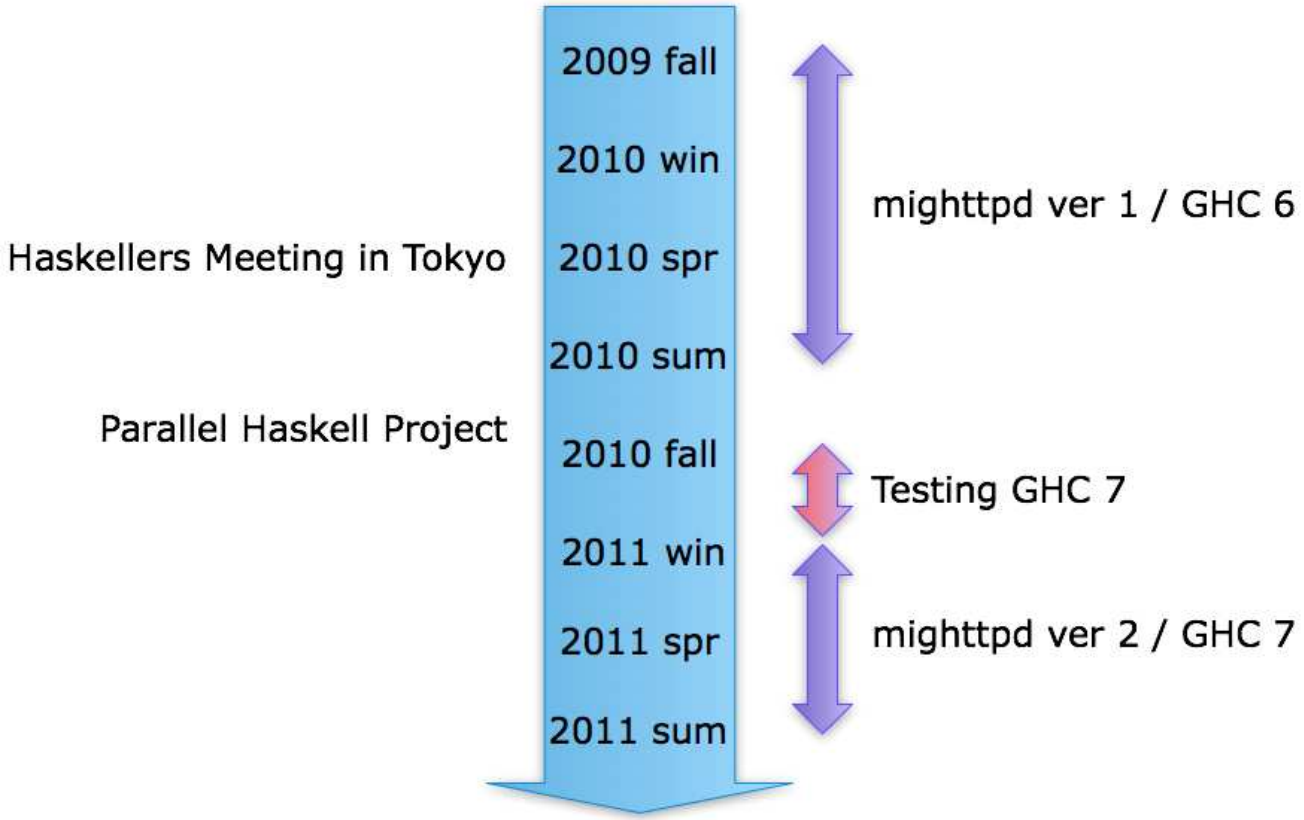
Modular

It's difficult to understand if you don't know Haskell

Will talk about

Highly Concurrent

Timeline


2009 fall
2010 win
Haskellers Meeting in Tokyo
2010 spr
2010 sum
Parallel Haskell Project
2010 fall
2011 win
2011 spr
2011 sum
mighttpd ver 1 / GHC 6
Testing GHC 7
mighttpd ver 2 / GHC 7

Haskellers Meeting 2010 Spring

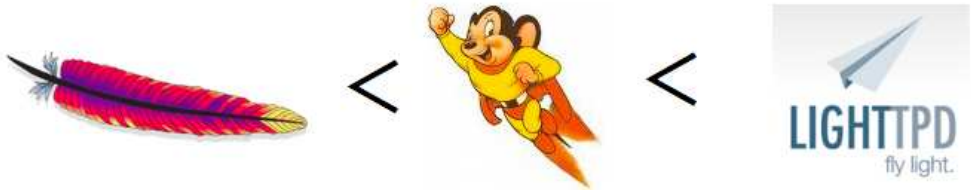
- Simon Peyton Jones came to Tokyo
  - 16 Apr 2010
- I made a presentation
  - Experience on implementing a Web server in Haskell
    - http://www.mew.org/~kazu/material/2010-mighttpd-en.pdf
- The following slides are from the presentation

Three Goals of Mighttpd

Functionality — Mighttpd should provide enough functionality to replace Apache on my domain "Mew.org".

Modularity — Mighttpd should be able to be modified easily for our research.

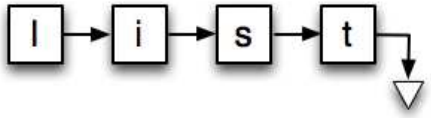
Performance — Mighttpd should exceed Apache on static contents.



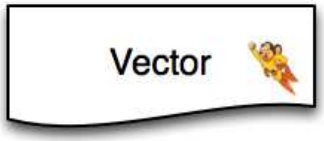
Two Ideas for Performance

ByteString

Traditional **`String`** in Haskell is very slow.



`ByteString` is faster like **`char[]`** in C.



User thread

Kernel thread is heavy.

User thread is light.

HTTP and thread programming

Network protocol	Message oriented	Stream oriented
	DNS	SMTP, HTTP
Network programming	Event driven	Threading
	`select,kqueue,epoll`	`fork,pthread_create`

Event driven programming for stream oriented protocol is messy.

Thread programming for stream oriented protocol is concise.

I want to implement HTTP on threading. Simplicity is a good thing.

User Thread is Real Thread

GHC has an IO manager as a user thread.
It is event-driven.

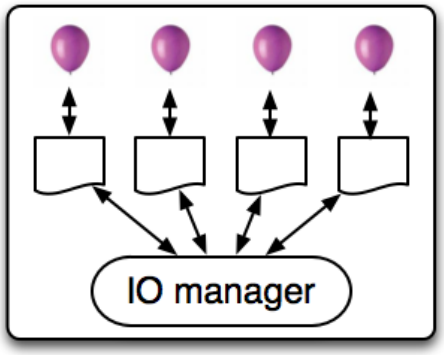
It takes care of buffering and wakes up blocked user threads.

So, using user threads is really thread programming.

`hGet`

`Handle`

`Socket`



User thread

Buffering

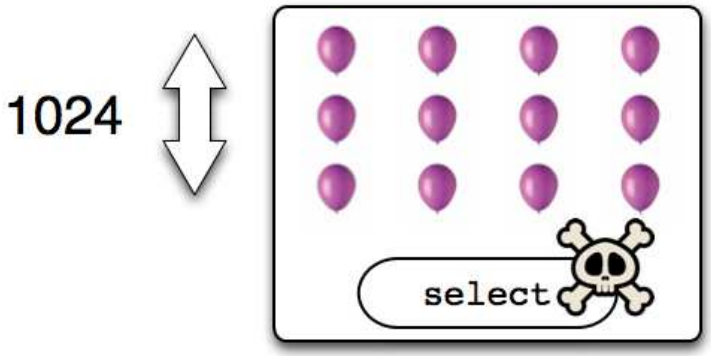
Event driven

The barrier of 1,024 connections

The IO manager is implemented using `select`.

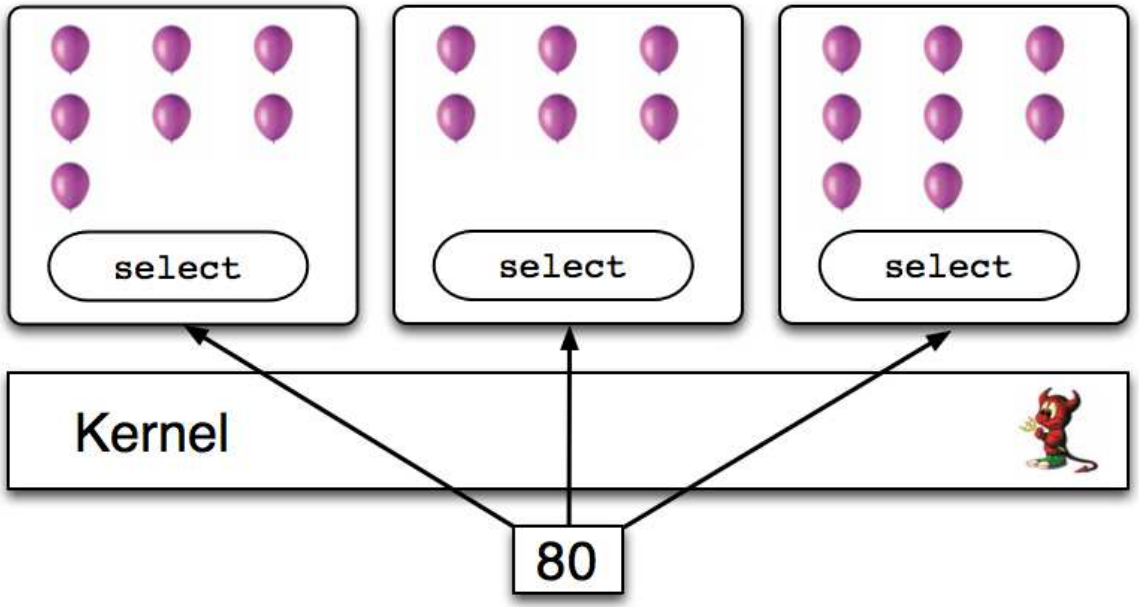
`select` cannot handle over 1,024 files/connections.

If GHC 6.12 receives over 1,024 connections, resource exhaustion exception happens.



Prefork library

Prefork is a technique to share a listening port among forked processes.



Now, GHC 6.12 can accept any number of connections!

Mighttpd implementation

Package name			
mighttpd	File base	KVS base	Not released
webserver	HTTP, session, redirect, CGI		
c10k	prefork		

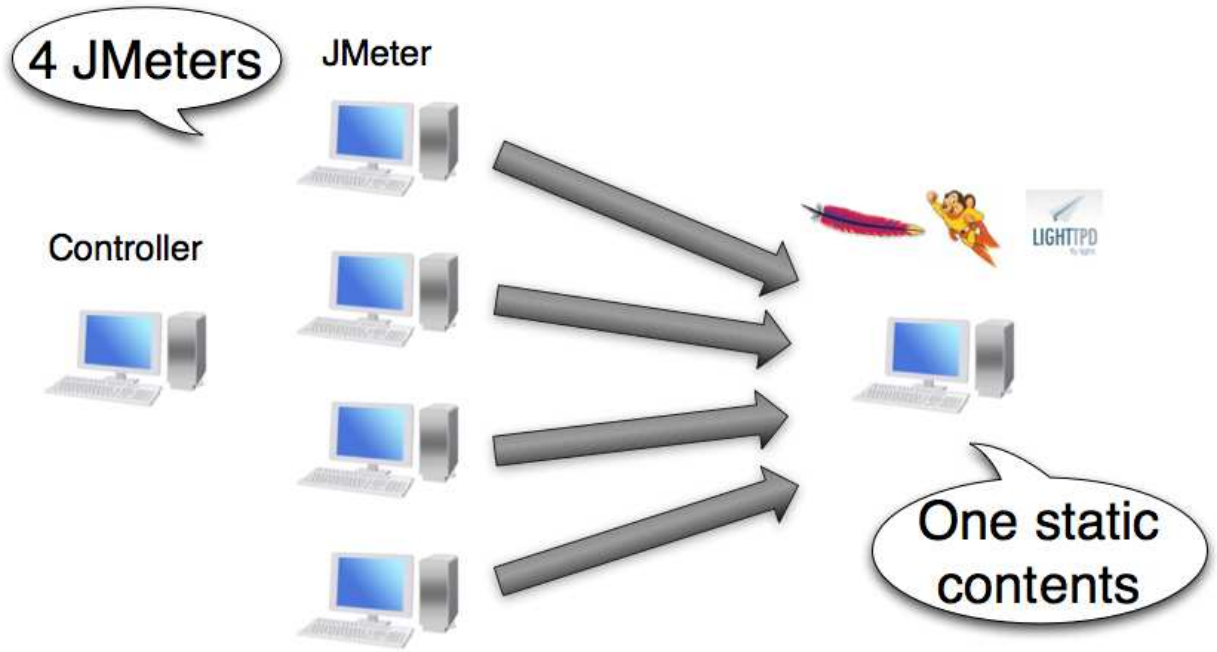
Modularity

"webserver" is designed to handle any storage systems.

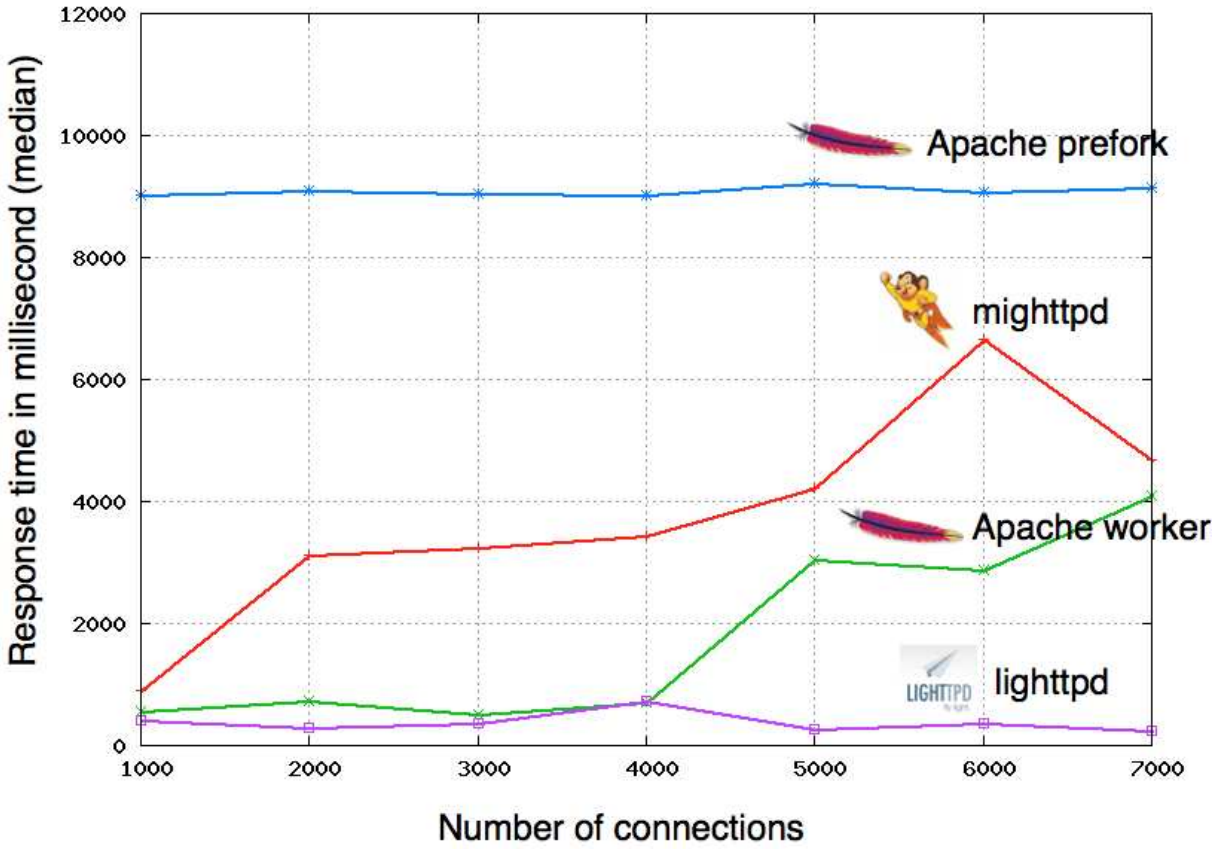
Functionality

"mighttpd" works on Mew.org now!

Benchmark Environment



Benchmark Result



Profiling

File IO is dominant.
Why, Mighttpd slower than Apache?

```
% ab -n 2000 -c 200 -k http://localhost/

COST CENTRE    MODULE    %time %alloc

fileGet        File      73.3   37.4
mighty         File      20.0   57.9
fileInfo       File       6.7    2.9
fileMapper     File       0.0    1.1
```

Ah, it's overhead of `select`!

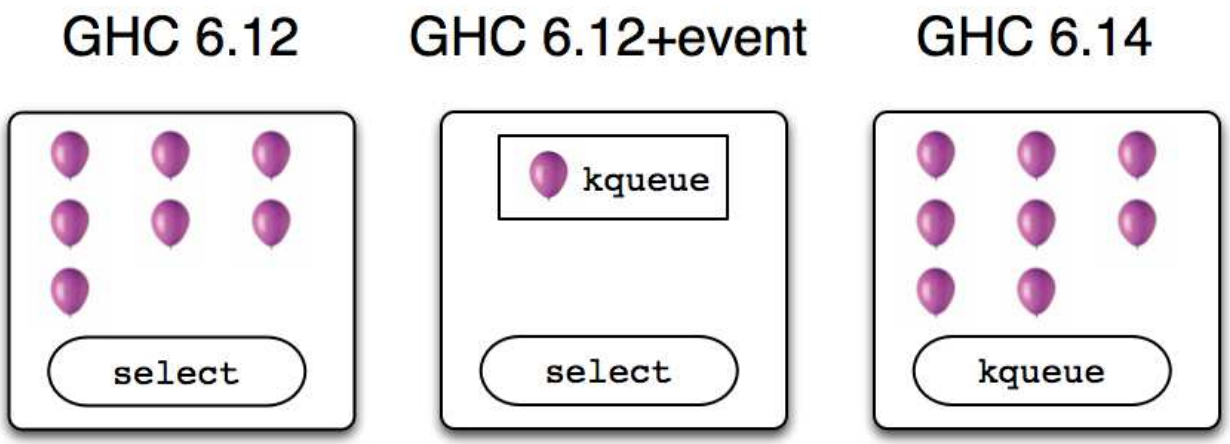
Any hopes?

One Hope

Tibbe and Bos are developing "event" library for `kqueue` and `epoll`.

Now we can use it for event-driven network programming.

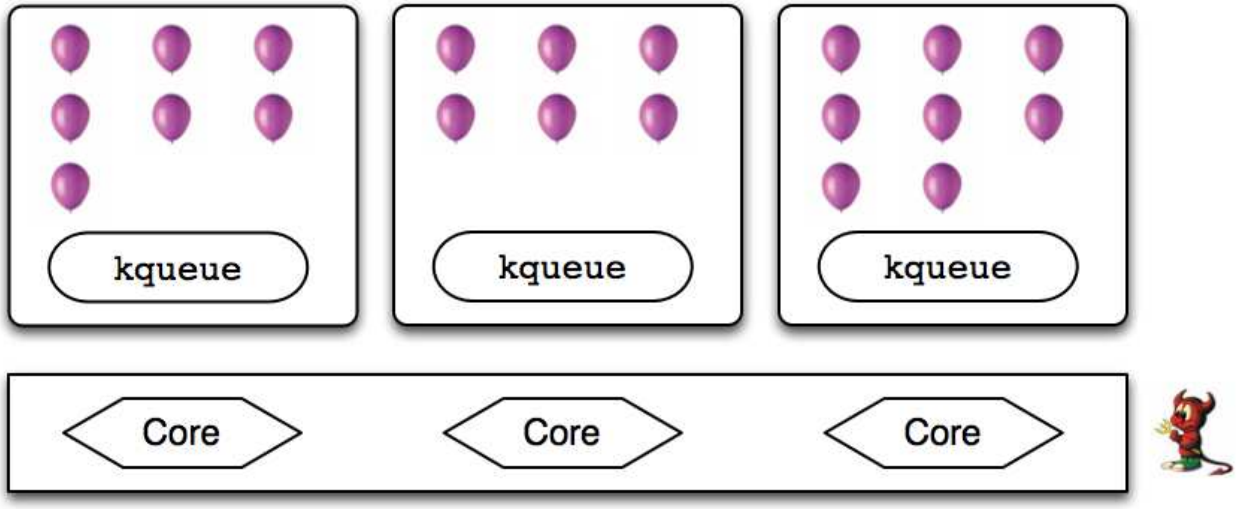
They are planning to integrate it into the IO manager in GHC 6.14.



Future architecture

Since there is only one IO manager,
GHC 6.14 would not balance on multi-core.

But the prefork technique could be used
to balance on multi-core.



Between Mighttpd 1 and 2

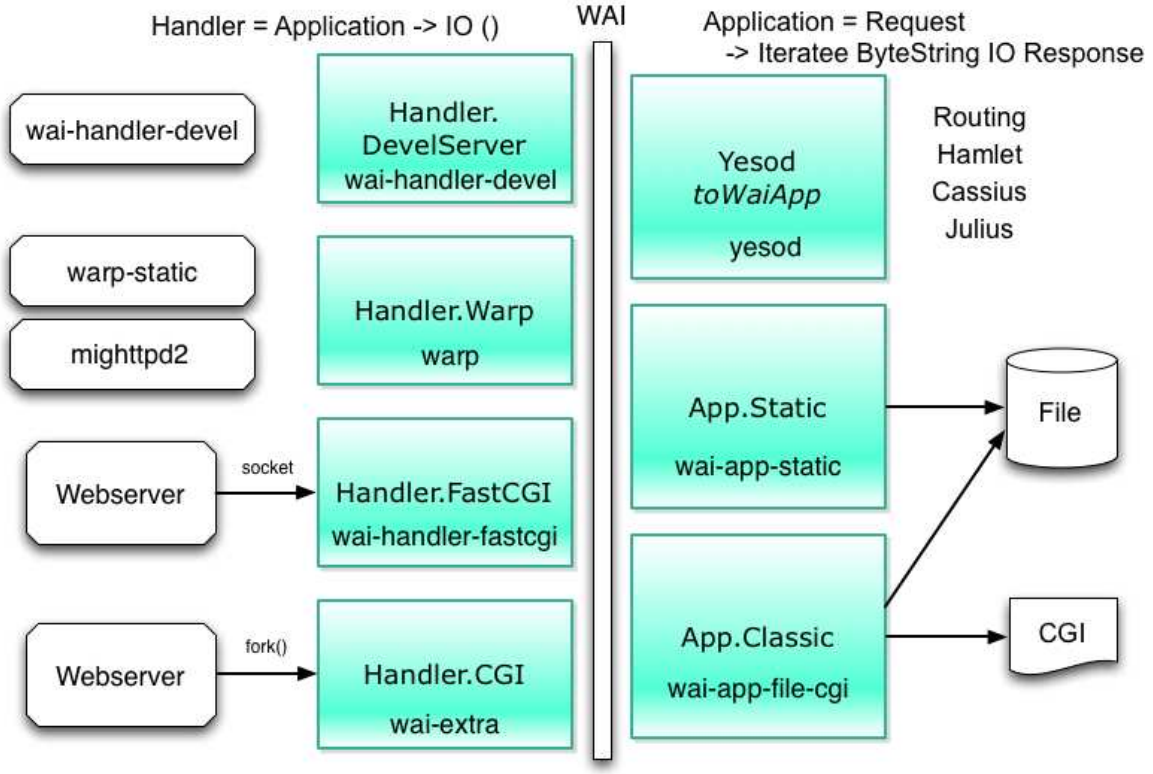
- Parallel Haskell Project
  - Budget from MS Research
  - Steering by well-typed
  - IIJ-II was chosen as a partner
  - well-typed and IIJ-II have skype meeting every other week
- GHC 7 (aka GHC 6.14)
  - New IO manager based on `epoll()` and `kqueue()`
- Web application framework boom
  - Snap
  - HappStack
  - Yesod
    - WAI (Web Application Interface)

Testing GHC 7

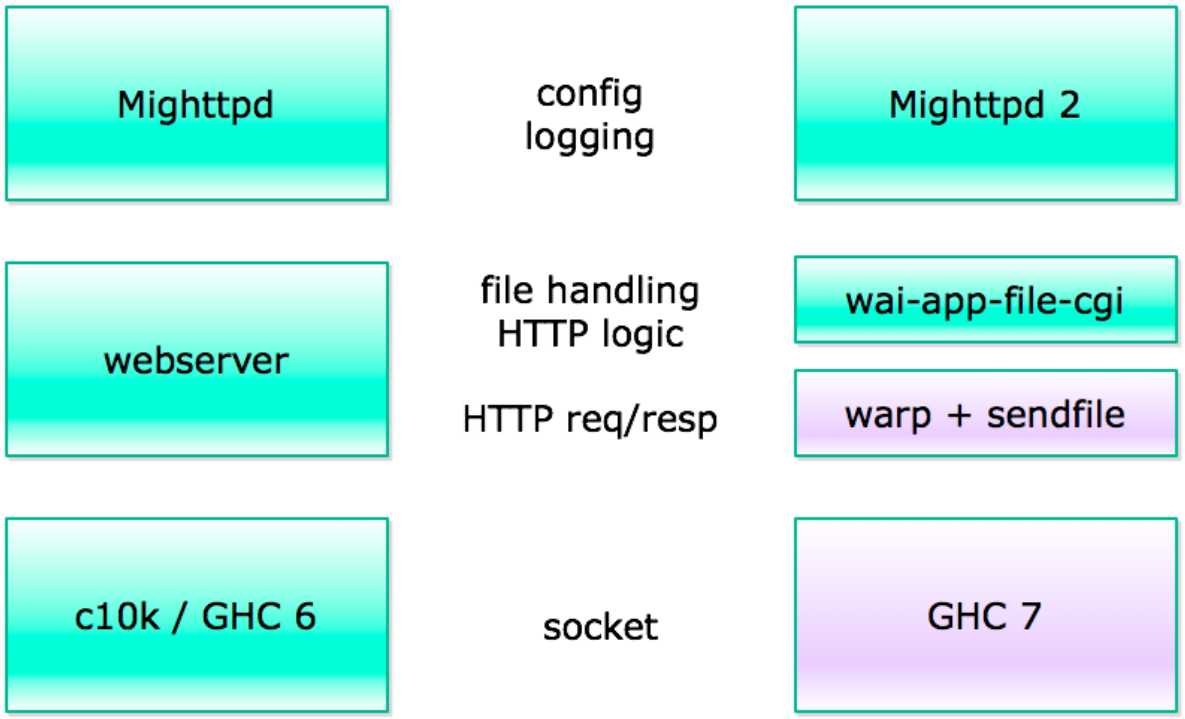
- New IO manager of GHC 7.0.1 was unstable
  - I found 6 bugs
  - GHC HQ and well-type fixed them
- Bugs
  - kqueue socket disappears on Mac if demonized
    - http://hackage.haskell.org/trac/ghc/ticket/4449
  - Cannot wait signals
    - http://hackage.haskell.org/trac/ghc/ticket/4504
  - Event logs are strange
    - http://hackage.haskell.org/trac/ghc/ticket/4512
  - IO manager would be dead-locked
    - http://hackage.haskell.org/trac/ghc/ticket/4514
  - Behavior of getContents is strange
    - http://hackage.haskell.org/trac/ghc/ticket/4895
  - hsc2hs cannot work on Mac
    - http://hackage.haskell.org/trac/ghc/ticket/4852
- New IO manager of GHC 7.0.2 is now stable

Web Application Interface

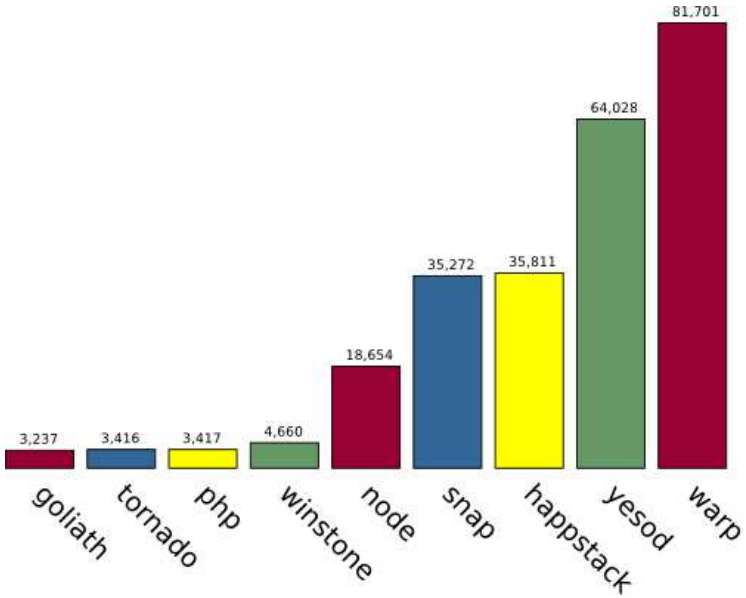
- API for Yesod and HappStack



Adopting Web Application Interface



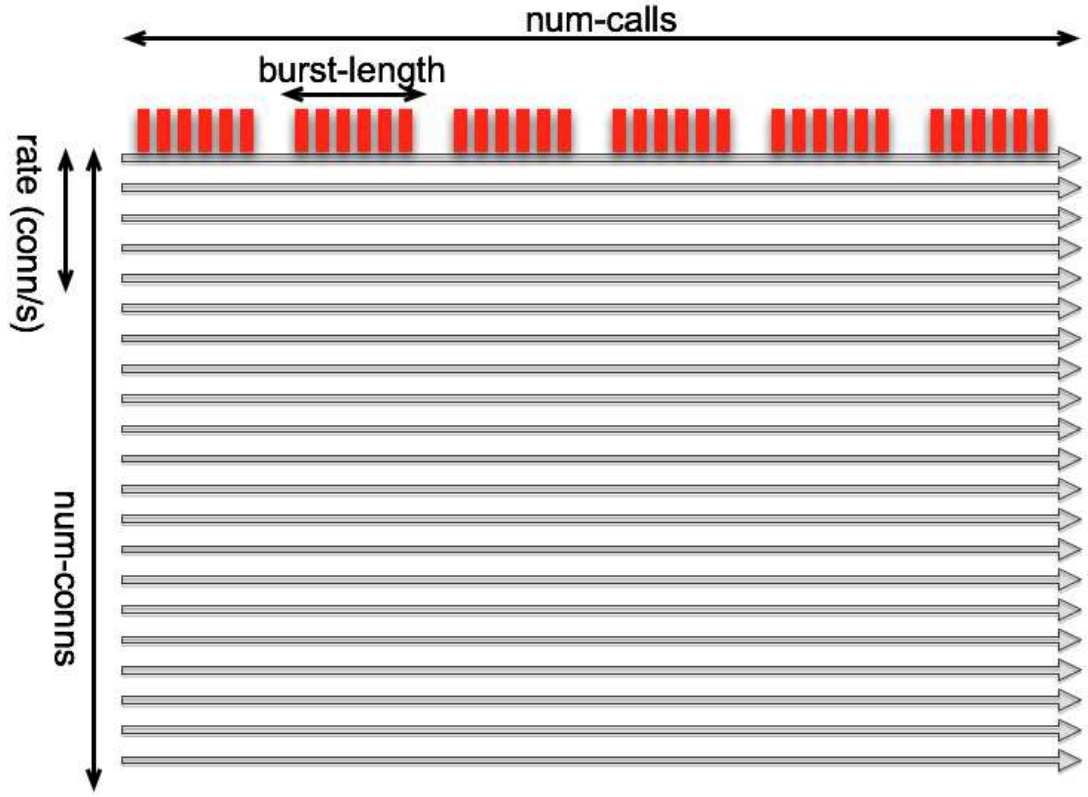
Warp performance



http://www.yesodweb.com/blog/2011/03/preliminary-warp-cross-language-benchmarks

- Warp
  - No HTTP logic
  - Just parses HTTP req and composes HTTP resp
  - Does not handle Last-Modified:
  - Does not touch a file

httperf Ping-Pong benchmark




```
httperf --hog --num-conns 1000 --num-calls 1000
 --burst-length 20 --rate 1000 --server localhost
 --port 3000 --uri /
```

Warp and mighttpd 2

- Benchmarking in my environment
  - Host
    - Intel(R) Xeon(R) CPU L5520 @ 2.27GHz x 8, 4 cores for each (32 cores)
    - 24G memory
    - Ubuntu 10.04, KVM 0.12.3
  - Guest
    - 4 cores
    - 1G memory
    - Ubuntu 10.10
- Warp (memory only)
  - 23928.1 req/s, 1 core, w/o logging
- Mighttpd 2 (with static files)
  - 4229.7 req/s, 1 core, w/o logging

Show-stoppers

- Tree based dictionary for Content-Type: O(log n)
  → Array-based immutable hash O(1)
- Date.Time
  - To parse and format HTTP Date (e.g. Last-Modified:)
  - Too slow. Consuming 30-40% of CPU time
  - Many division on type transforms
  - Inefficient list programming

  → Creating simple ByteString based library
- System.Posix.Files.getFileStatus
  - Getting size and modification time of files (stat())

  → Caching in memory
    - Removing all cached information every 10 seconds
- System calls
  - Context switches are evil for user threads

sendfile

- The sendfile library
  - Unnecessary seek() and stat()
- Creating simple-sendfile library
  - Calling sendfile() only
  - No standard exits
    - Linux
    - FreeBSD
    - Mac
    - Fallback
- System calls in the current code
  - HTTP requests
    - recv()
  - HTTP response -- header
    - writev()
  - HTTP response -- body
    - open()
    - sendfile() -- Note that stat() information is cached
    - close()
    - File descriptor could be cached but the logic would be very complex

Benchmark on a single core

- nginx
  - 22713.3 req/s, 1 core, w/o logging
- Warp (memory only)
  - 23928.1 req/s, 1 core, w/o logging
- mighttpd2
  - 21601.6 req/s, 1 core, w/o logging
  - 4229.7 req/s, 1 core, w/o logging, not tuned

Scaling on multi cores

- New IO manager is a single kernel thread
  - +RTS -Nx does not help to scale on multi cores
  - +RTS -Nx is not friendly to forkProcess
  - Introducing the prefork technique again
- nginx with 3 workers
  - 30471.2 req/s, 3 cores, w/o logging
  - 22713.3 req/s, 1 core, w/o logging
- mighttpd2 with 3 prefork processes
  - 61309.0 req/s, 3 cores, w/o logging
  - 21601.6 req/s, 1 core, w/o logging

Logging is the biggest show-stoppers

```
128.141.242.20 - - [08/Jul/2011:17:05:14 +0900]
"GET /favicon.ico" 404 11
```

- Data.Time again
  - → Caching formatted string
    - Calling gettimeofday() every second
    - Formatting with Data.Time due to time zone
- getnameinfo() in C
  - → Simply implement in Haskell

Various logging schemes

- Serialization
  - Haskell channel (atomic queue)
    - Buffering in memory
  - Appending a file
- Writing a file
  - truncate() and mmap()
  - Blocking write()
  - Non-blocking write()
- File IO dedicated process with shared memory
- Implemented many combinations...
- Appeared that the simplest one is best
  - Non-blocking write() with Handle on each process
  - Handle is automatically locked by MVar.
  - Multi line buffering with BlockBuffering
    - hPut flushes the buffer before buffering if there is not enough space
    - So, hPut never split a line

Benchmark with logging

- nginx with 3 workers
  - 25035.2 req/s, 3 cores, w/ logging
  - 30471.2 req/s, 3 cores, w/o logging
- mighttpd2 with 3 prefork processes
  - 31101.5 req/s, 3 cores, w/ logging
  - 61309.0 req/s, 3 cores, w/o logging
- Room for improvement in logging?

Conclusions so far

- Mighttpd 2 is fast enough
  - To one httperf Ping-Pong benchmark in one env, Mighttpd 2 is faster than nginx
- Haskell user thread is good for C10K
  - System calls are evils
  - Blocking IO is also evil
- Room for improvement in logging?
- Todo
  - Reverse proxy
  - Tackling multi-thread IO manager?
    - It would be hard. Worth trying?
  - Enhancing httperf
    - epoll() / kqueue()
    - IPv6