HaskellでHTTP/2を実装してみました

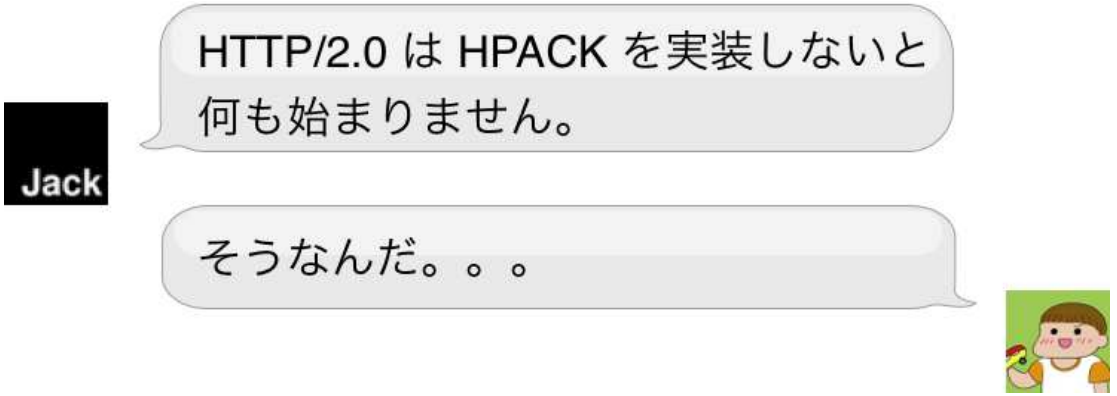
山本和彦
@kazu_yamamoto

お品書き

- 歴史
- TLS との戦い
- 軽量スレッド
- Warp

歴史

2009年　HaskellでHTTPサーバMightyの実装を開始
2011年　MightyをWAIアプリとして再実装
　　WAIのHTTPエンジンのWarpの改良に参加
2013年10月 http/2.0ミニハッカソンに参加



2013年12月 HPACKのみを実装したhttp2 libをリリース
2014年10月 フレームを追加したhttp2 libをリリース
2014年10月 Warpへhttp2を組み込み開始
2014年11月 TLSとの戦い

TLS との戦い

ステップ1

- Haskell の TLS ライブラリに ALPN(Application Layer Protocol Negotiation) を機能追加した
  - NPN(Next Protocol Negotiation) はサポートされていた
  - NPN を参考に ALPN を実装
  - (このパッチは本家にマージされた)

でも Warp が HTTP/2 で
ブラウザと通信できない！

2014年10月24日

Firefox Nightly もそうなんですが、TLS でハンドシェイクして、暗号路で Finished を送った後に、すぐに Alert (close notification) を送ってくるんですが、どうしてでしょうか？

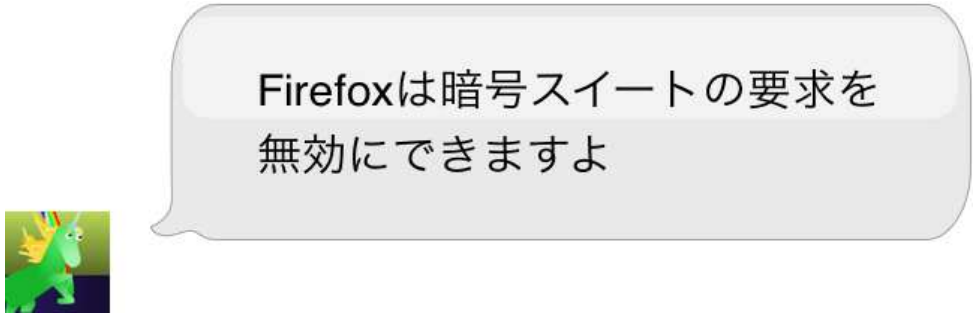
Cipher SuitesがHTTP/2の仕様で要求するものとあってないのではないでしょうか

暗号スイート

- HTTP/2 の要求
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- Firefox
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- Chrome
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- Haskell TLS ライブラリに足りない暗号
  - ECDHE(Elliptic Curve Diffie-Hellman Ephemeral)
    - 短命 楕円曲線 DIffie Hellman
  - AES GCM(Galois/Counter Mode)

ステップ2

- HTTP/2 カンファレンス懇親会



- Firefox で TLS の要求を disabled
  - network.http.spdy.enforce-tls-profile
- Firefox と喋れた
  - TLS_RSA_WITH_AES_CBC_SHA

ステップ3

- AES GCM の実装は存在した
  - GCM は暗号化と同時に認証タグを計算
- TLS に AEAD を実装した
  - Authenticated Encryption with Additional Data
  - AES GCM を組み込んだ

- Chrome と喋れた
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- 学んだこと
  - DHE はサーバで暗号パラメータを用意するのが面倒

ステップ4

- ECDHE を実装した
  - これは10分で完成
- TLS でその ECDHE を利用可能にした
- 素の Firefox と喋れた
  - もちろん Chrome とも喋れる

- 学んだこと：楕円曲線のパラメータの設定は不要
  - よいパラメータには名前が付いている
  - TLS では、名前を交換し、決定するだけ
  - HTTP/2 では P256 (素数 256ビット) を使う
    - 128ビットの強度：TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - 楕円の場合は、素数の半分が強度

暗号強度

- デファクトスタンダード暗号技術の大移行(5) より

ビット	80	112	128	192	256
共有鍵暗号	80	112	128	192	256
RSA	1024	2048	3072	7680	15360
DH	1024	2048	3072	7680	15360
楕円曲線暗号	160	224	256	384	512
ハッシュ	160	224	256	384	512

- HTTP/2 の要求
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    P256

軽量スレッド

お題

Webサーバの実装を考える

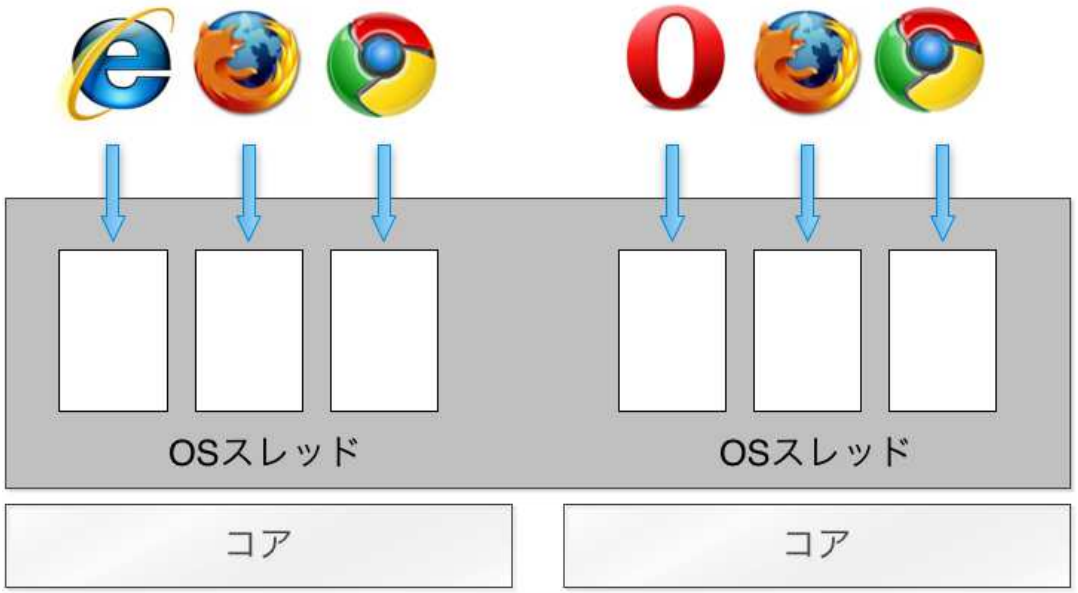
クライアントが1つの場合

- 見通しのよい一直線のコードが書ける
- 一つのコアしか活用していない

"On the duality of operating system structures", 1978

イベント(メッセージパッシング)と
スレッド(プロセスベース)は双対である

OSスレッドを使って複数の場合に対応



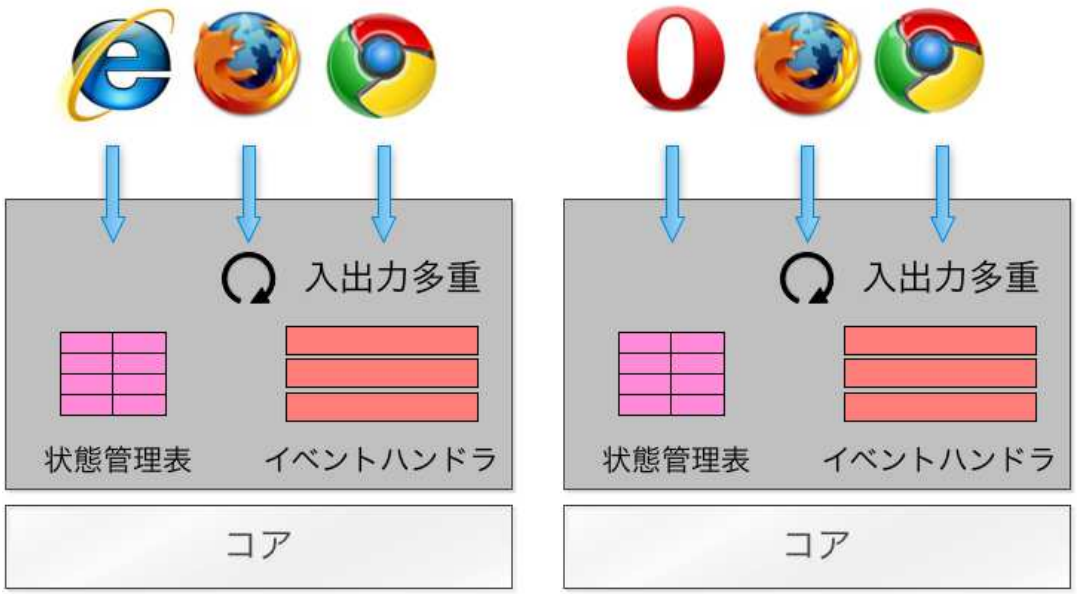
- 利点：見通しのよいコードを再利用できる
  複数のコアを活用できる
- 欠点：OSスレッドの切り替えが多発し遅い

"Why Threads Are A Bad Idea", 1996

OS スレッドはデッドロックに
陥りやすく性能も低い

代替品はイベントである

イベント駆動を使って複数の場合に対応



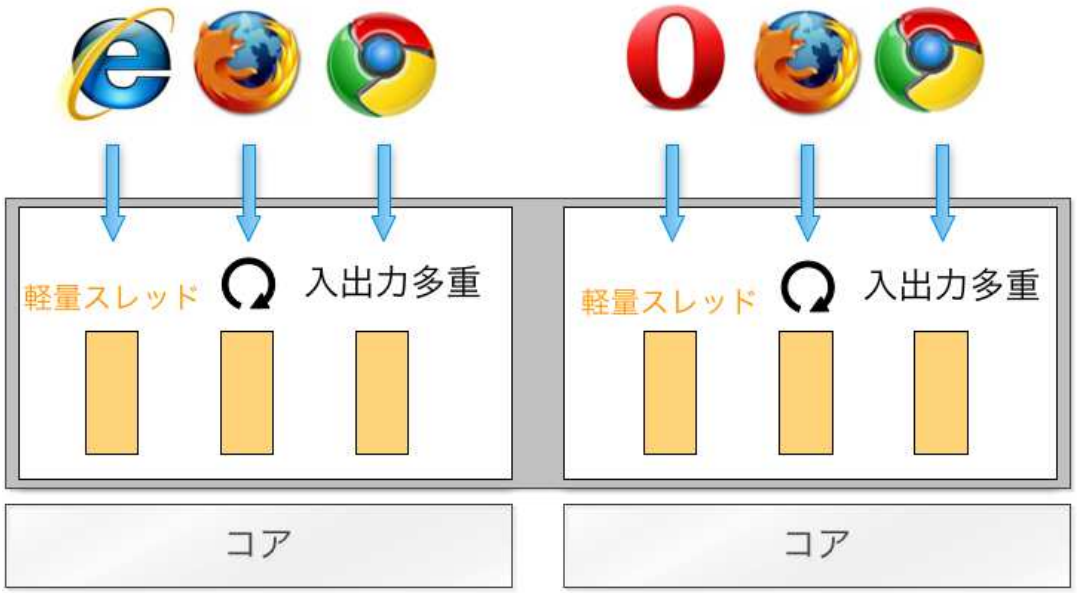
- 利点：コアの性能を引き出せる
  - 複数のコアを利用するには prefork してポートを共有
- 欠点：見通しの悪いコードへ作り直し
  - preforkの欠点：共有メモリがない
    (レート制御などの実現が難しい)

"Why Events Are A Bad Idea", 2003

イベント駆動は制御構造と状態管理が困難

軽量スレッドは言語のサポートが鍵

軽量スレッドを使って複数の場合に対応



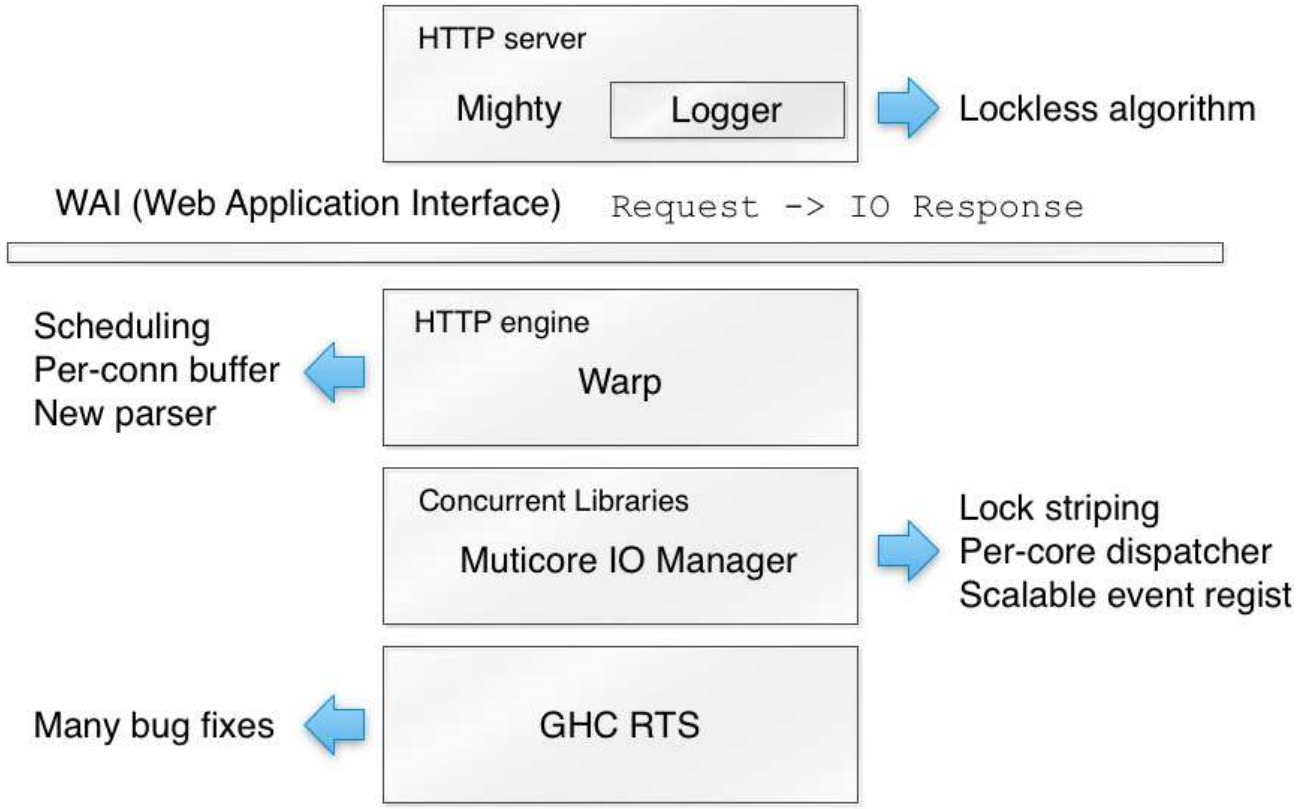
- 利点：見通しのよいコードを再利用できる
- 利点：コアの性能を引き出せる

複数のクライアントを扱う技術のまとめ

OS スレッド	クライアント一つ用と複数用のコードが同じ 一直線のコードを書ける 低速
イベント駆動	クライアント一つ用と複数用のコードが異なる コードを分断し状態を管理 高速
軽量スレッド	クライアント一つ用と複数用のコードが同じ 一直線のコードを書ける 高速

Warp

WAI と Warp


HTTP server
Mighty
Logger
Lockless algorithm
WAI (Web Application Interface)
Request -> IO Response
Scheduling
Per-conn buffer
New parser
HTTP engine
Warp
Concurrent Libraries
Muticore IO Manager
Lock striping
Per-core dispatcher
Scalable event regist
Many bug fixes
GHC RTS

Mio: A High-Performance Multicore IO Manager for GHC

Andreas Voellmy Junchang Wang
Paul Hudak
Yale University, Department of Computer Science
andreas.voellmy@yale.edu
junchang.wang@yale.edu
paul.hudak@yale.edu

Kazuhiko Yamamoto
IIJ Innovation Institute Inc.
kazu@iij.ad.jp

Abstract

Haskell threads provide a key, lightweight concurrency abstraction to simplify the programming of important network applications such as web servers and software-defined network (SDN) controllers. The flagship Glasgow Haskell Compiler (GHC) introduces a run-time system (RTS) to achieve a high-performance multicore implementation of Haskell threads, by introducing effective components such as a multicore scheduler, a parallel garbage collector, an IO manager, and efficient multicore memory allocation. Evaluations of the GHC RTS, however, show that it does not scale well on multicore processors, leading to poor performance of many network applications that try to use lightweight Haskell threads. In this paper, we show that the GHC *IO manager*, which is a crucial component of the GHC RTS, is the scaling bottleneck. Through a

1. Introduction

Haskell threads (also known as green threads or user threads) provide a key abstraction for writing high-performance, concurrent programs [16] in Haskell [12]. For example, consider a network server such as a web server. Serving a web request may involve disk IOs to fetch the requested web page, which can be slow. To achieve high performance, web servers process other requests, if any are available, while waiting for slow disk operations to complete. During high load, a large number of requests may arrive during the IOs, which gives rise to many requests in progress concurrently. A naive implementation, using one *native thread* (i.e. OS thread) per request would lead to the use of a large number of native threads, which would substantially degrade performance due to the relatively high cost of OS context switches [22]. In con-

Warp

Kazu Yamamoto,
Michael Snoyman, and Andreas Voellmy

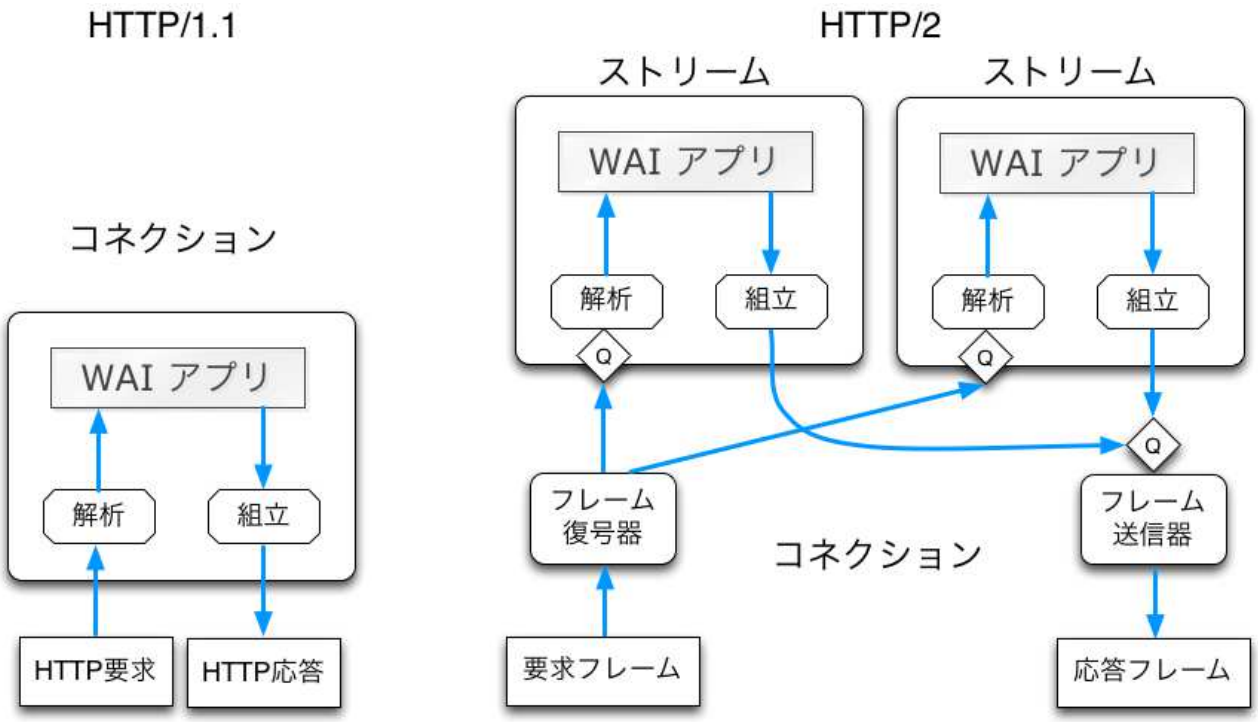
The Performance of Open Source Applications
Speed, Precision, and a Bit of Serendipity

Warp is a high-performance HTTP server library written in Haskell, a purely functional programming language. Both Yesod, a web application framework, and `mighty`, an HTTP server, are implemented over Warp. According to our throughput benchmark, `mighty` provides performance on a par with `nginx`. This article will explain the architecture of Warp and how we achieved its performance. Warp can run on many platforms, including Linux, BSD variants, Mac OS, and Windows. To simplify our explanation, however, we will only talk about Linux for the remainder of this article.

Network Programming in Haskell

Some people believe that functional programming languages are slow or impractical. However, to the best of our knowledge, Haskell provides a nearly ideal approach for network programming. This is because the Glasgow Haskell Compiler (GHC), the flagship compiler for Haskell, provides lightweight and robust user threads (sometimes called green threads). In this section, we briefly review some

Warp での HTTP/1.1 と HTTP/2



今後の課題

- TLS の高速化
- 4種類のストーリミング型へ対応
- タイマー
- HTTP/2 のプライオリティ

宣伝

- IIJ Technical WEEK 2014 で講演します
  - 2014年11月26日 14:40 〜 15:30
  - マルチコア時代の最新並列並行技術 Haskellから見える世界