Relatório Parcial - Implementação do Protocol HTTP2 na linguagem Scala

Daniel Q. Miranda

Sumário

Proposta

Me proponho a implementar uma biblioteca e/ou servidor HTTP/2 na linguagem Scala. Preferencialmente buscarei trabalhar para que ela seja integrada ao projeto Akka, muito popular framework de concorrência e computação distribuída na JVM, que recentemente iniciou um projeto de implementação de HTTP combinando esforços de diversos outros projetos e frameworks de aplicações Web. Caso isso não seja possível, a desenvolverei de maneira independente.

Cronograma previsto inicialmente

- Mar-Abr: Estudo preliminar, estudo do projeto Akka e conversa com desenvolvedores para verificar viabilidade de integração
- Abr-Maio: Design inicial de arquitetura, planejamento, estudo do protocolo
- Maio-Jul: Design de APIs, detalhamento da arquitetura, prototipação inicial
- Jul-Out: Implementação de biblioteca já funcional, incluindo maior partes das features possível:
  - Múltiplos streams (multiplexing)
  - Compressão de Headers
  - *Server Push*
  - HTTP2c (TLS)
- Out-Nov: Otimizações de performances, testes e comparações com outras implementações, elaboração de estudos de arquitetura,

Progresso atual

Até o presente momento a maior parte do progresso constituiu em compreender e avaliar:

- O protocolo HTTP/2
- Implementações existentes do HTTP/2 em diversas linguagens
- Implementações de outros servidores HTTP em Java

- O ecossistema Scala e aplicação possível do resultado pela comunidade

Plataforma Akka

O plano inicial era trabalhar dentro da plataforma Akka, já muito adotada para concorrência em Java e Scala, e que parece estar, com sucesso, iniciando um processo de unificação de bibliotecas de comunicação na JVM.

O contato inicial do projeto foi recebido até com interesse, mas os desenvolvedores manifestaram falta de recursos e/ou tempo para *** a iniciativa. O Akka, assim como diversos outros grandes projetos em Scala, vem sendo financiado pela empresa Typesafe, e a maioria dos contribuidores de grande-escala são seus empregados, com objetivos e planos pré-existentes e nem sempre flexíveis.

Decidi então trabalhar por contra própria, tomando a biblioteca HTTP do Akka apenas como inspiração. Porém, ainda cogito utilizar a biblioteca homônima ao projeto, que lida principalmente com concorrência através do modelo de atores (mais sobre isso segue nas próximas seções), entrada e saída a nível de arquivos e sockets, dentre outros.

O protocolo HTTP/2

A nova versão do protocolo HTTP (HTTP/2) adiciona um certo nível de complexidade ao original para alcançar diversos objetivos, visando modernizar o protocolo para que ele continue eficiente nas décadas que seguirão.

Alguns exemplos destes objetivos, com seus correspondentes mecanismos, são:

Utilizar eficientemente um número pequeno (menor possível) de conexões

O protocolo HTTP/2 permite a existência de múltiplos *streams* de dados, tanto cliente-servidor como na direção oposta, para aproveitar ao máximo uma única conexão TCP. Isso permite que, por exemplo, diversos recursos sejam carregadas simultaneamente, que ações de um usuário em um aplicativo Web sejam enviadas sem espera, ou que um modo de uso com troca de informações contínua seja possível (que hoje, por exemplo, é comummente implementado em browsers por um segundo protocolo, o WebSocket).

Diminuir o uso de banda em geral

Mecanismos de compressão são suportados universalmente pelo HTTP/1 há mais de uma década, mas em múltiplos casos fraquezas de segurança (CRIME (Kelsey 2002), BREACH (Prado, Harris, and Gluck 2013)) foram descobertas durante seu uso, causadas pelo gradual “vazamento” de informações sobre padrões dos dados ou meta-dados

transmitidos. O HTTP/2 visa melhorar significativamente esta situação introduzindo um mecanismo próprio de compressão de headers, eficiente e resistente a esses vazamentos, chamado HPACK.

Melhorar eficiência de aplicações que usam o HTTP, como a Web

O mecanismo de "Server Push", onde um servidor pode preventivamente recomendar que um cliente faça uma requisição pode acelerar significativamente a visualização de páginas Web em geral. Também tem potencial benefício para o uso do HTTP como API (através de padrões como REST e SOAP), permitindo o paginamento de recursos de maneira limpa e eficiente.

Manter compatibilidade "bidirecional" entre HTTP/1 e HTTP/2

Múltiplos mecanismos de negociação foram especificados, para permitir que:

- Clientes pré-existentes se mantenham funcionando sem perda de funcionalidade
- Clientes novos possam se conectar de maneira genérica a um servidor, e utilizar o HTTP/2 se possível
- Seja possível identificar se um servidor é compatível com HTTP/2 sem overhead.

Os principais escolhidos e especificados no protocolo são:

- Negociação de upgrade: uma conexão HTTP/1 é iniciada, com um conjunto de headers informando a compatibilidade com HTTP/2. Isso permite conexão universal, aceita tanto por HTTP/1 e HTTP/2
- Negociação de protocolo de aplicação através do TLS, quando em uso: o protocolo Transport Layer Security tem um mecanismo interno de negociação, que pode ser usado para selecionar HTTP/2 caso possível
- Conexão direta através da versão 2, que tem um preâmbulo de conexão próprio, facilmente distinguível do HTTP/1

Implementações existentes do HTTP/2 e outros servidores em Java

As principais implementações no ecossistema Java vem de dois projetos já muito estabelecidos: Jetty (do projeto Eclipse) e Netty. Ambas tem alto foco em performance e qualidade em geral, e tem grandes gamas de contribuidores e suporte.

Após o início do projeto, descobri a existência do projeto http4s-blaze em Scala, com objetivos similares aos meus, que implementa um servidor HTTP/2 já na linguagem Scala. Teria sido interessante saber de sua existência de antemão, mas não vejo um

problema: o objetivo principal do trabalho é o aprendizado e a experiência. Minha implementação pode seguir escolhas diferentes de arquitetura, prioridades, etc.

Decisões de design e escolhas tomadas

Embora infelizmente ainda tenha produzido pouco código, não suficiente para um protótipo, já tenho decididas diversas escolhas de modo de trabalho, arquitetura, objetivos, etc.

- Sistema de build: SBT
- Biblioteca de I/O: Akka I/O
- Buscar ao máximo implementar, sempre que possível, todo o servidor utilizando estruturas de dados imutáveis, programação funcional e composição de componentes independentes. Este paradigma é fortemente suportado pela linguagem Scala, assim como a biblioteca Akka I/O, e não necessariamente incorre em perda de eficiência com o devido cuidado.
- Focar em simplicidade e elegância arquitetural e correção em vez de performance. Devido ao tempo de trabalho e recursos, é incrivelmente difícil competir com projetos com vários contribuidores, empresas em seu suporte, poder computacional, etc. no quesito performance
- Utilizar um design baseado em "pipeline", com os diversos estágios de um request sendo tratados independentemente por componentes distintos

Cronograma

O plano do cronograma inicial segue mantido: o trabalho até outubro/novembro consistirá em implementar o máximo possível do protocolo em um servidor simples, mas funcional. Para alcançar esse objetivo, serão necessárias algumas adaptações, como:

- Se limitar a um ou o menor número possível de métodos de negociação

- Focar inicialmente na comunicação básica, codificação, decodificação e tratamento de headers, assim como funcionamento com streams únicos (mas bidirecionais)
- Focar inicialmente na implementação sem TLS (em *plain-text*)

Kelsey, John. 2002. "Compression and Information Leakage of Plaintext." In *Fast Software Encryption*, edited by Joan Daemen and Vincent Rijmen, 2365:263–76. Lecture Notes in Computer Science. Springer Berlin Heidelberg. doi:10.1007/3-540-45661-9_21.

Prado, Angelo, Neal Harris, and Yoel Gluck. 2013. "SSL, Gone in 30 Seconds." In *Black Hat USA*. http://breachattack.com/resources/BREACH%20-%20BH%202013%20-%20PRESENTATION.pdf.