Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Uso de Elixir para a construção de um middleware baseado em RPC

Mateus Moury Fernandes da Rosa Borges

Proposta de Trabalho de Graduação

Orientador: Nelson Souto Rosa

Recife
Abril 2016

Resumo

De acordo com o que se tem visto nos últimos anos, a capacidade de processamento de um único computador se mostra cada vez mais insuficiente para a construção de aplicações escaláveis e que demandam alto desempenho. Por isso, há uma necessidade cada vez maior de se construírem sistemas distribuídos. O ideal, na construção de um sistema distribuído, é que o programador se preocupe apenas com a sua aplicação, e não com complicações inerentes ao fato de fazer com que sua aplicação seja distribuída, como problemas na rede e de localização. Esse trabalho propõe a construção de um *middleware* baseado em chamada remota de procedimento que visa facilitar o desenvolvimento de aplicações distribuídas. Elixir, uma nova linguagem funcional baseada em Erlang, será usada para o desenvolvimento do *middleware*, já que vem se mostrando uma linguagem que provê facilidades desejáveis no desenvolvimento de um *middleware*, como facilidade para comunicação entre processos e recuperação de falhas, além de ser usada comercialmente em aplicações de grande porte como o *WhatsApp e Pinterest.* O trabalho trará, ainda, comparações quanto ao desempenho do *middleware* desenvolvido nesse trabalho e de outros artigos, bem como vantagens e desvantagens do uso de Elixir para a implementação de middleware.

Palavras-chave: Elixir, *Middleware*, Sistemas Distribuídos*,* Tolerância a Falhas, Programação Funcional.

Abstract

As time goes by, the processing capacity of a single computer is being proved increasingly insufficient to build efficient and scalable applications. Consequently, there is a rising demand for deploying distributed systems. Ideally, when building a distributed system, the developer's concerns should only regard the application's logic, and not complications inherent to distributing the application itself, such as connection problems. Therefore, this work's goal is to build a middleware based in remote procedure calls that aims to facilitate the development of distributed applications for programmers. Elixir, a fairly new functional programming language based in Erlang, is going to be used in the project, since it provides desirable features to construct a middleware, namely failure recover and interprocess communication, among others. Besides, Elixir is used in huge real world applications, like WhatsApp and Messenger. This work will also show the middleware's accomplishments in terms of efficiency and will compare its results with other middleware's, in conjunction with analyzing the benefits and weak points of using the chosen programming language.

Key-words: Elixir, Middleware, Distributed Systems, Failure Recovery, Functional Programming.

Sumário

Introdução

Hoje em dia, as áreas de aplicações para sistemas distribuídos são as mais diversas possíveis. Vários dos maiores e mais complexos sistemas que utilizamos hoje em dia são sistemas distribuídos. O exemplo mais claro de um sistema distribuído que usamos é a Internet. A Internet pode ser vista como um grande sistema distribuído com um imenso número de servidores que se comunicam com um número ainda maior de clientes através de vários protocolos.

Existem várias razões para que sistemas distribuídos sejam usados. Algumas dessas razões são motivadas pela própria natureza do problema que se quer resolver. Por exemplo, na Internet, caso um usuário queira requisitar uma página a um servidor distante, ou bater papo com um amigo, não existe outro caminho a não ser o de transportar dados de um ponto a outro. Outras razões são motivadas pelas propriedades do problema. Se um problema pode ser quebrado em vários subproblemas que podem ser resolvidos independentemente, por exemplo, provavelmente dividir esses subproblemas em várias máquinas pode ser uma boa solução. Adicionalmente, se uma aplicação pode ser distribuída sem muita perda, distribuir tal aplicação evita que ela tenha um ponto único de falha (VÖLTER et al., 2004, p. 1).

Obviamente, construir um sistema distribuído é uma tarefa complexa, e existem vários desafios no *design* de um sistema distribuído. Entre outros, um deles é a heterogeneidade de sistemas distribuídos: num sistema em que várias máquinas estão presentes, provavelmente a comunicação será feita por diferentes sistemas operacionais, com diferentes estruturas de comunicação e hardwares variados, então existe um esforço extra para que um sistema heterogêneo funcione. Um outro exemplo de desafio é o de transparência: o sistema tem que ser elaborado de forma que o usuário final não perceba que o sistema é distribuído (MISHRA; TRIPATHI, 2014). Na Internet, por exemplo, o usuário final, que digita uma URL no *browser*, não precisa saber onde está o servidor que traz a informação para seu computador.

Num cenário ideal, é desejável que o programador, ao desenvolver uma aplicação, não se preocupe com os desafios inerentes ao fato dessa aplicação ser distribuída ou não. E para isso serve o *middleware* de distribuição: uma camada extra de infra estrutura que poupa o desenvolvedor de resolver problemas que não são diretamente do escopo da sua aplicação (BERNSTEIN, 1996). Claramente, nenhum *middleware* vai ser capaz de prover uma infraestrutura totalmente transparente para o desenvolvedor. Como um sistema distribuído utiliza comunicação em rede, latência e falhas de conexão são problemas que o usuário frequentemente percebe.

Existem várias abordagens para a implementação de um *middleware* de distribuição. Historicamente, a primeira abordagem foi a de transferência de

arquivos. Quando programas em máquinas diferentes queriam se comunicar, eles faziam o *download* de arquivos que continham informações de interesse, possivelmente modificavam ou criavam novos arquivos, e faziam o *upload* destes. Muito embora essa abordagem tenha resolvido vários problemas, várias outras surgiram para resolver novos problemas.

A abordagem usada para a construção do *middleware* deste trabalho é a de *Remote Procedure Call*s (BIRREL; NELSON, 1984). O objetivo da abordagem é fazer com que invocar um procedimento remoto (isto é, uma função a ser executada em um outro ambiente) pareça o mesmo que invocar uma função local, do ponto de vista do desenvolvedor da aplicação. Desta maneira, o programador escreveria o código sem nenhuma mudança, apesar do procedimento não estar localizado na sua máquina. O RPC funciona como o modelo cliente/servidor: o programa que invoca o procedimento é o cliente, e o que o executa, o servidor. Toda essa operação é síncrona, ou seja, o andamento do cliente é bloqueado até que o procedimento remoto dê a resposta à invocação (ROUSE, 2009).

Todo o software desenvolvido durante este trabalho utiliza Elixir[1] como linguagem de programação. Elixir tem como objetivo prover facilidades para a construção de sistemas distribuídos e tolerantes a falha. Elixir estende Erlang (utiliza a *Erlang Virtual Machine*) para adicionar uma sintaxe mais agradável e maior expressividade.

Além de prover facilidade de comunicação entre processos utilizando mecanismos de transferência de mensagens, Elixir é uma linguagem funcional e engloba vários benefícios desse paradigma, como a imutabilidade de dados, que associada à capacidade da linguagem de criar *threads* leves faz com que sistemas altamente concorrentes e escaláveis possam ser desenvolvidos. A linguagem, apesar de criada recentemente, é usada em sistemas distribuídos de grande porte e com vários usuários, como Pinterest[2] e WhatsApp[3].

Dessa forma, pode-se observar que Elixir parece uma linguagem bastante compatível para preencher os requisitos da construção de um *middleware*. Com cada vez mais aplicações distribuídas, é indiscutível a importância de se tentar construir sistemas de *middleware* cada vez mais robustos e eficientes. Por isso, este trabalho analisa as vantagens e desvantagens de se construir um *middleware* usando a linguagem.

[1] http://elixir-lang.org/
[2] https://www.pinterest.com
[3] https://www.whatsapp.com

Objetivos

O objetivo deste trabalho é, primeiramente, usar Elixir como linguagem de programação para construir um *middleware* de comunicação que comportará chamadas a procedimentos remotos. Além disso, o trabalho objetiva também fazer uma análise detalhada sobre quais as vantagens e desvantagens de ter usado essa linguagem, possivelmente analisando quais seriam as diferenças, perdas e ganhos se fosse usada uma linguagem com o paradigma de orientação a objetos. A eficiência do sistema será mensurada e comparada com a de outros *sistemas de middleware* desenvolvidos no CIn - UFPE.

Estrutura do Trabalho

O trabalho será composto pelos seguintes capítulos:

1. **Introdução**: este capítulo detalhará o tema, explicando os termos utilizados e mostrando a motivação para a execução do trabalho, bem como o objetivo que deve ser atingido ao final dele.
2. **Sistemas Distribuídos e *Middleware* de Comunicação**: o capítulo discutirá mais a fundo o que é um sistema distribuído e que tipo de problemas ele resolve, dando alguns exemplos de famosos sistemas distribuídos. Além disso, introduzirá conceitos básicos do que é um *middleware*, e mostrará exemplos de um *middleware* bastante usado em funcionamento: o RMI.
3. **Arquitetura e implementação do *middleware***: este capítulo irá dispor a arquitetura do *middleware*, explicando qual será a função de cada módulo do seu *design*, que será baseado nos *Remoting Patterns*, bem como trechos da implementação dessa arquitetura, utilizando Elixir como linguagem de programação.
4. **Avaliação do sistema**: aqui, serão dispostas diferentes análises sobre a implementação realizada, como a eficiência do sistema e os benefícios e malefícios de ter usado Elixir para implementá-lo.
5. **Conclusão**: Será mostrado um breve resumo do estudo realizado e da arquitetura apresentada. Em seguida, quais foram os desafios encontrados, as limitações do sistema e sugestões de trabalhos futuros.
6. **Referências Bibliográficas**: a lista de referências utilizadas no trabalho.

Cronograma

As atividades serão desenvolvidas conforme o cronograma abaixo.

Atividades	Abril				Maio				Junho				Julho	
Revisão Bibliográfica														
Estudo sobre sistemas distribuídos e investigação do middleware RMI														
Pesquisa aprofundada em Elixir e estudo dos *Remoting Patterns*														
Desenvolvimento do *middleware* seguindo a arquitetura estudada														
Elaboração do relatório final														
Preparação da defesa														
Defesa														

Possíveis Avaliadores

Um possível avaliador do trabalho a ser produzido é o professor Ricardo Massa Ferreira Lima (CIn/UFPE).

Assinaturas

_____

Nelson Souto Rosa
Orientador

_____

Mateus Moury Fernandes da Rosa Borges
Aluno

Referências Bibliográficas

[1] VÖLTER, Markus; KIRCHER, Michael; ZDUN, Uwe. **Remoting Patterns:** Foundations of Enterprise, Internet and Realtime Distributed Object Middleware. Chichester: Wiley, 2004.

[2] MISHRA, Kamal Sheel; TRIPATHI, Anil Kumar. Some Issues, Challenges and Problems of Distributed Software System. **International Journal of Computer Science and Information Technologies.** Varanasi, India, 07/2014. Caderno p. 3. Disponível em: http://www.ijcsit.com/docs/Volume%205/vol5issue04/ijcsit2014050420.pdf. Acesso em: 11/04/2016

[3] ROUSE, Margaret. **Remote Procedure Call (RPC).** 2009. Disponível em: http://searchsoa.techtarget.com/definition/Remote-Procedure-Call. Acesso em: 10/04/2016

[4] BIRRELL, ANDREW D.; NELSON, BRUCE JAY. Implementing Remote Procedure Calls. **ACM Transactions on Computer Systems.** Palo Alto, CA, 02/1984. Caderno p. 39-59. Disponível em: http://www.cs.virginia.edu/~zaher/classes/CS656/birrel.pdf. Acesso em: 14/04/2016

[5] BERNSTEIN, Philip A.. **Middleware:** A model for distributed system services. **Communications of the ACM**, New York, NY, v. 39, n. 2, p. 86-98. 02/1996.