



Universidade de Brasília – UnB  
Faculdade UnB Gama – FGA  
Engenharia de Software

## **Relatório de Fundamentos de Sistemas Distribuídos**

Autor: Phelipe Wener  
Professor: Fernando W. Cruz

Brasília, DF  
2017





# Lista de ilustrações

Figura 1 – Solução proposta pelo relatório . . . . .	21
Figura 2 – Ligando o repository . . . . .	22
Figura 3 – Ligando dois slaves . . . . .	23
Figura 4 – Obtendo resposta nos masters . . . . .	24
Figura 5 – Obtendo resposta nos masters de matrizes 10x10 . . . . .	25



## Lista de tabelas



# Lista de abreviaturas e siglas

VM            Virtual Machine ou Máquina Virtual

API           Application Programming Interface

SOA           Service-oriented architecture

Mutex        Mutual Exclusion





# Lista de símbolos

$\Sigma$  Utilizada como notação de somatório



# Sumário

<b>1</b>	<b>OBJETIVOS</b>	<b>11</b>
<b>2</b>	<b>INTRODUÇÃO</b>	<b>13</b>
<b>3</b>	<b>DESENVOLVIMENTO</b>	<b>15</b>
3.1	Entendendo Web Services	15
3.2	Entendendo REST	15
3.3	Arquitetura proposta	16
3.3.1	Problema de Produto de Matrizes	16
3.3.2	Requisitos da solução	16
3.3.3	WebService proposto	17
3.3.4	Solução proposta	17
3.4	Ambiente e Configuração	17
3.5	Experimento	18
3.5.1	Desenvolvimento	18
3.5.2	Execução	19
<b>4</b>	<b>RESULTADOS</b>	<b>27</b>
4.1	SOAP vs REST	27
4.2	Sobre a Solução	27
4.3	Falhas e Tolerâncias	27
4.4	Opinião Pessoal	28
	Referências	29
	<b>ANEXOS</b>	<b>31</b>
	<b>ANEXO A – ROTEIRO</b>	<b>33</b>



# 1 Objetivos

O objetivo desse experimento é a compreensão de webservices no contexto de sistemas distribuídos, tal que seja possível por meio de uma aplicação Bag-of-Tasks resolver problemas que demandam alto processamento.



## 2 Introdução

Um sistema distribuído é um conjunto de entidades independentes que cooperam para resolver um problema que não pode ser individualmente resolvido (KSHMKALYANI, 2008) Muitos são os contextos modernos que utilizam desses sistemas para obtenção de alto poder de processamento, afim de se resolver trabalhos difíceis como por exemplo encontrar grandes números primos ou realizar busca de vida extraterrestre [Acesse <https://setiathome.berkeley.edu/> para mais informações].

Existem vários tipos de modelos de aplicações distribuídas, suas características estão intimamente ligadas aos problemas que se propõe a resolver. Neste trabalho o enfoque maior envolve problemas que podem ser resolvidos usando Bag-of-Task. Aplicações Bag-of-Tasks(BoT) são aquelas aplicações paralelas cujas tarefas são independentes uma da outra (CIRNE, 2003)

Para tal foi proposto dois problemas, o primeiro envolve produto de matrizes, operação matemática largamente utilizado na engenharia. A segunda, envolve a construção de fractais, área da geometria não euclidiana, relativamente nova. Serão utilizadas duas abordagens distintas para cada problema, onde fundamentalmente será explorado os conceitos de Web Service, bag-of-tasks e RPC.





## 3 Desenvolvimento

### 3.1 Entendendo Web Services

Os Web Services são aplicativos cliente e servidor que se comunicam através do HTTP (HyperText Transfer Protocol) da World Wide Web (WWW). Conforme descrito pelo World Wide Web Consortium (W3C), os Web Services fornecem um meio padrão de interoperação entre aplicativos de software rodando sobre variedade de plataformas e frameworks ([ORACLE, 2013](#)). Web Service é uma solução que vem sendo amplamente utilizada devido a própria necessidade de se construir aplicações clientes (como browsers) que precisam consumir informações de um servidor. Os aplicativos de smartphones são um exemplo que compõem grande parte dessa demanda atualmente.

A idéia básica é que algum aplicativo cliente pode chamar os serviços fornecidos por um aplicativo servidor ([TANNENBAUM, 2007](#)). Dessa forma é possível criar softwares modularizados utilizando os conceitos de Front e Backend, arquiteturas de micro serviços ([ALMEIDA, 2015](#)), bem como consumidores automáticos como se pretende realizar nos experimentos abaixo. Esse conceito está abrangido no termo SOA(Service-oriented architecture), que prega que as funcionalidades de uma aplicação sejam disponibilizadas como serviços, de forma que se evite redundância de dados e replicação de sistemas ou funcionalidades desenvolvidas para o mesmo fim.

Além disso, precisamos garantir que a chamada de serviço prosseguirá ao longo das regras definidas pela aplicação do servidor. Note que este princípio não é diferente do que é necessário para realizar uma chamada de procedimento remoto ([TANNENBAUM, 2007](#)). Portanto, nota-se que facilmente pode-se confundir os conceitos de Web Service com Remotes Procedure Calls (RPC), uma vez que em ambos existe servidores, clientes e um modo padronizado de comunicação, o que os difere? Para ([XINYANG; SHEN; FAN, 2009](#)), Web Services adotam arquiteturas RPC como seu desenho arquitetural. Nesse contexto é ainda defendido pelo autor a abordagem REST como uma alternativa a RPC. Tal que essa abordagem costuma ser mais simplista que outras formas de Webservice que implementam SOAP, WS-Addressing, WS-Security, etc.

### 3.2 Entendendo REST

REST é um estilo arquitetônico que Roy T. Fielding definiu pela primeira vez em sua tese de doutorado. [...] Ele viu o REST como uma forma de ajudar a comunicar os conceitos básicos subjacentes a Web. Para entender REST é necessário entender a

definição de recurso, representação e estado. (XINYANG; SHEN; FAN, 2009)

Basicamente, o REST(Representational State Transferer) se comunica via solicitações baseadas em quatro operações HTTP: GET, POST, PUT e DELETE. Cada uma dessas operações se aplica a recursos, informações relevantes mantidas pelas aplicações, o qual são representados via notação comum: XML ou JSON são os mais utilizados. Nessa troca de representações de recursos não há nenhuma troca de estado.

(KSHEMKALYANI, 2008), enfatiza algumas vantagens em se usar REST em relação a RPC. A mais notável diferença é facilidade de uso uma vez que diferente das outras abordagens, o REST não é uma série de padronizações, ele apenas usa HTTP para fazer comunicação, onde isso pode ser configurado de maneira que for conveniente, desde que os desenvolvedores sigam uma convenção de representação.

### 3.3 Arquitetura proposta

#### 3.3.1 Problema de Produto de Matrizes

Multiplicação de Matrizes: Sejam  $A = [a_{ij}]_{m \times n}$  e  $B = [b_{rs}]_{n \times p}$ .

Definimos  $AB = [c_{uv}]_{m \times p}$  onde

$$c_{uv} = \sum_{k=1}^n a_{uk}b_{kv} = a_{u1}b_{1v} + \dots + a_{un}b_{nv} \quad (\text{BOLDRINI, 2009})$$

Para ser necessário a resolução de matrizes usando computação distribuída, é preciso que se tenha entradas que resultem em uma matriz de ordem muito alta. Portanto, serão feitos testes com matrizes em várias ordens de tamanho.

#### 3.3.2 Requisitos da solução

No presente roteiro Anexo A, é proposto a implementação de um repositório de tarefas o qual processos escravos vão frequentemente ler, executar e enviar resultados. Dentre os requisitos dessa arquitetura, junto ao contexto de multiplicação de matrizes, entende-se que:

- Um ou mais masters podem ser executados colocando via *pairIn* uma chave, com uma linha ou coluna de uma matriz. Por exemplo: *pairIn*('A2', '[1, 2, 3]').
- O repositório recebe e guarda essa *pair* deixando-a disponível tanto em *pairOut* quanto em *readPair*.
- Um ou mais slaves irá procurar iterativamente via *pairOut* e executar o produto entre duas *pairs* retornando o resultado através de *pairIn* com uma chave no formato 'ixj' por exemplo, tal que i é a linha do elemento em A e j é da coluna de B.

- Após multiplicada toda a matriz, o master tem que consultar o repositório para obter o resultado.

A figura 1 ilustra como se dará a comunicação.

### 3.3.3 WebService proposto

Para os experimentos a seguir é proposto um WebService RESTful, não só por uma curiosidade do aluno em implementar essa abordagem que vem ganhando cada vez mais o mercado, mas por que em questão de complexidade o REST parece mais fácil que o SOAP de se implementar. A decisão foi com base nesse [link](#).

Ao final, no capítulo de Resultados será analisado se essa escolha.

### 3.3.4 Solução proposta

Para a solução desse problema foi utilizada a arquitetura [Rails 5](#) para se implementar o serviço Restful do repository. Para o master e slave, foram criados dois scripts em python que farão as requisições REST através do módulo [Request](#), que trabalha com requisições HTTP. Os pairs serão armazenados em memória, numa estrutura de dados chamada Array, que irá guardar cada pair como uma hash com chave e valor.

De fato a arquitetura rails é um tanto complexa para o problema, foi escolhida pela facilidade no uso de seus componentes bem como sua configuração. O repositório irá guardar os dados em memória, o que torna grande parte do suporte do rails inútil para o contexto.

## 3.4 Ambiente e Configuração

Todos os passos abaixo foram desenvolvidos numa distribuição Linux chamada kubuntu, que difere fundamentalmente em termos de interface com o Ubuntu. Portanto, deve ser possível executar os mesmos passos em qualquer versão do Ubuntu 14.04, bem como em um Debian 7. O repository foi usado dentro de um host externo, uma máquina gratuita dos planos da plataforma [Heroku](#) que apesar de ter baixo recurso, será suficiente para nosso experimento.

## 3.5 Experimento

### 3.5.1 Desenvolvimento

- Primeiramente foi implementado um projeto matrix-repository utilizando [Ruby on Rails](#), através do comando: `rails new matrix-repository -api -O`, tal que `-api` sinaliza uma arquitetura Restful com dado representado em JSON e `-O` que o projeto não irá precisar de banco, nosso contexto.
- Em seguida foi implementado o controlador `repository_controller.rb` no diretório `app/repository/controllers`.
- Nesse controller foi implementado os métodos `pair_in`, `pair_out` e `read_pair` conforme requisitado, e para fins de suporte outros dois métodos `read_all` e `reset`.
- O controller usa um *design pattern* [singleton](#) chamado de `task_stack` que é responsável por gerenciar a lista de pairs que o repository mantém. Ele está localizado também em `app/repository/controllers`.
- Esse objeto singleton é mantido durante toda a vida da aplicação, diferente dos outros componentes do Rails, como os controllers, que são descartados a cada requisição de usuário. Logo, se a lista que armazenasse as pairs estivessem em um controller, seria reiniciada a cada execução.
- O singleton tem 3 métodos que atendem aos requisitos: `put`, `get(key)`, `pop(key)` onde apenas o `put` e o `pop` alteram o estado do array mantido. Logo precisam ser gerenciados, tal que a modificação seja feita de forma síncrona.
- Para tal, foi utilizado um mutex(mutual exclusion), que permite que apenas um processo altere o array por vez, prevenindo de possíveis inconsistências, uma vez que vários masters e slaves irão requisitar alterações simultaneamente. O Ruby prevê um objeto nativamente para fazer isso.
- Enquanto se implementava o singleton e o controller, foi preciso configurar as rotas, o resultado final se encontra em `config/routes.rb`
- Finalizada a implementação do repository, foi focado agora no master, para isso foi simplesmente criada uma pasta matrix-master e depois um arquivo `master.py`.
- Basicamente o master lê um arquivo de matriz, como os exemplos na pasta matrix-master/input, e envia eles via método `repo_request` que requisita pela url no formato `http://localhost:3000/repository/pair_in/Ax/{a11, a12, a13}`, caso esteja sendo usado o repository no localhost.

- Para execução do master, deve-se executar no terminal o seguinte comando:  
`$ python3 master.py <nome do arquivo matrix> <tipo>`  
tal que o tipo deve ser A ou B, afim de que seja identificada qual matrix ele deve mandar linhas e qual colunas. Vale lembrar também que multiplicação de matrizes não é comutativa.
- Após isso o master.py espera por resposta na função `get_result(order)` onde iterativamente, a cada 1 segundo(delay colocado para não buscar muito rápido) ele recupera os resultados, denotados por chave no formato 'ixj'. Quando ele tem todos resultados, recupera no repository via `read_all`, uma adaptação de `read_pair` e imprime o resultado.
- Nosso master portanto fica esperando o resultado. Teoricamente pode-se executar muitos masters, mas isso daria problemas uma vez que pares de incompatibilidade na multiplicação seriam processadas pelos slaves. Portanto 2 masters é o ideal.
- Foi criado um diretório matrix-slave com um arquivo python slave.py.
- Os slaves basicamente executam infinitamente, procurando por tarefas sucessivamente no repositório, tal que se não existir uma chave Bj que multiplica com Ai ele deve procurar um proximo elemento i+1 ou j+1.
- Após executar uma tarefa, ele devolve o resultado via método `repo_pair_in`, que faz um POST no repositório com `pair_in`.

### 3.5.2 Execução

- Primeiramente é preciso ligar o matrix-repository, a figura 2 ilustra esse processo.
- Podemos conferir ele acessando o link [http://localhost:3000/repository/read\\_all](http://localhost:3000/repository/read_all), que deve estar com um array vazio.
- Logo precisamos ligar nossos slaves, veja a Figura 3
- Perceba que o argumento passado é igual a ordem da matrix a ser calculada, os slaves precisam saber disso.
- Após isso é preciso disparar os masters passando nome do arquivo com a matriz e o seu tipo.
- Será obtido um resultado conforme a Figura 4.
- Se uma outra operação for selecionada é preciso resetar os pairs, para isso basta chamar por <http://localhost:3000/repository/reset>. O master não usa `pairOut` pois

removeria o pair resultante antes que o outro master conseguisse acessar. O que seria um erro.

- Para fins de teste, executou-se usando matrizes de 10x10, resultados estão nas Figuras [5](#)
- O experimento foi finalizado. Resultados no próximo capítulo.

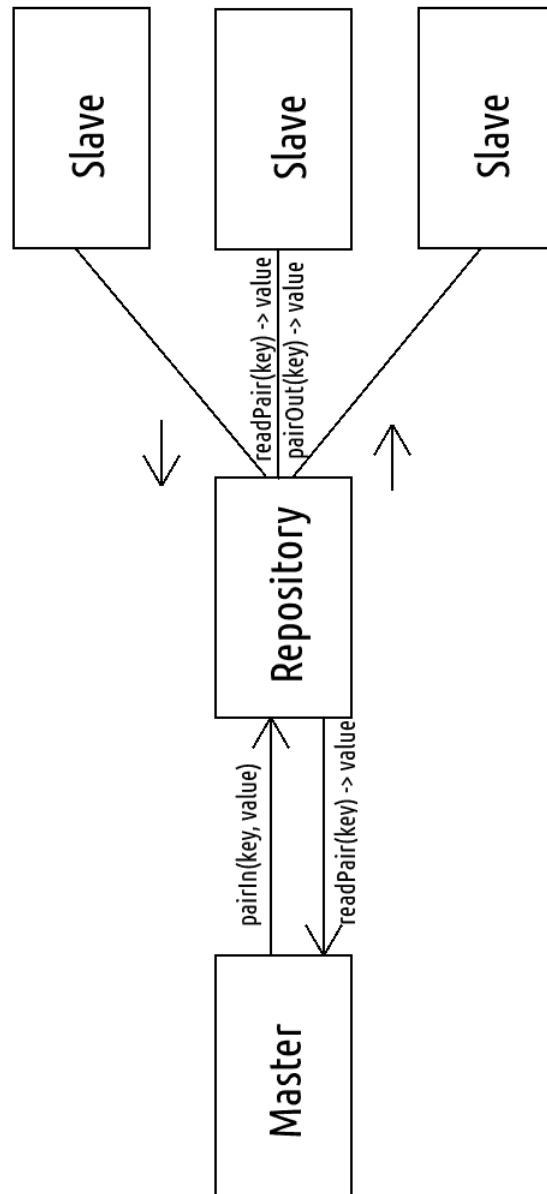



Figura 1 – Solução proposta pelo relatório



```
kuwener@wenerianus: ~/workspaces/ruby/matrix_calculator/matrix-repository
kuwener@wenerianus:~/workspaces/ruby/matrix_calculator/matrix-repository 168x48
matrix-master: matrix-repository: matrix-slave: README.md from https://git.berndkell.com
kuwener@wenerianus:~/workspaces/ruby/matrix_calculator/matrix-repository$ cd matrix-repository/
kuwener@wenerianus:~/workspaces/ruby/matrix_calculator/matrix-repository$ rails s
=> Booting Puma
=> Rails 5.0.3 application starting in development on http://localhost:3000
=> Run rails server -h for more startup options
Puma starting in single mode...
* Version 3.8.2 (ruby 2.2.3-p173), codename: Sassy Salamander
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

Figura 2 – Ligando o repository





Figura 3 – Ligando dois slaves







## 4 Resultados

### 4.1 SOAP vs REST

A decisão de fazer a solução em REST não pareceu favorável em contraponto ao SOAP. Isso foi percebível ainda quando o experimento estava em execução, um dos principais pontos onde o REST dificultou e o SOAP ajudaria foi na padronização de dados.

No REST, na padronização das mensagens, foi preciso convencionar tudo para que o envio e recebimento de dados ocorresse de forma consistente entre os componentes. No SOAP o suporte para isso é melhor, uma vez que um XML template é necessário.

Claro que em outros contextos, outras características divergentes entre os dois deveriam ser consideradas, mas para esse contexto a diferença fundamental foi essa.

### 4.2 Sobre a Solução

O Web Service proposto é uma arquitetura robusta e personalizável que pode lidar com os mais diferenciados contextos. Tentou-se seguir ao máximo a interface proposta no relatório, o que tornou a solução em alguns pontos não muito otimizada. Por exemplo, ao invés de *pairOut* precisar de uma chave para execução, poderia simplesmente ser um método pop que retornasse dois pairs disponíveis.

Na parte de sincronização, foi possível ver o mutex funcionando em matrizes na ordem de 100 pra cima, onde o *pairOut* espera um master colocar para que o outro faça isso, sendo possível visualizar no terminal esse delay. O desempenho para matrizes grandes não foi muito bom, a de 100x100 nem foi finalizada, a de 10x10 executando em 2 slaves rodou um tempo consideravelmente alto.

A solução é evidentemente mais funcional para vários hosts, onde se pode ter diversos slaves procurando pelas mesmas chaves, para tal basta mudar a URL de acesso do master e slave, de localhost, para um endereço com um repositório ativo.

### 4.3 Falhas e Tolerâncias

Uma falha do sistema é não suportar multiplicações de mais de 2 matrizes de forma consistente, já que se cadastrado mais de uma matriz A ou B, os slaves não saberão qual é o elemento de A ou B que precisa ser multiplicado.

Não foram identificados limites de execução, mas para algumas matrizes a solução é inviável, mesmo se bem distribuída.

## 4.4 Opinião Pessoal

Foi um trabalho de grande aprendizado que me mostrou as diversas formas de se trabalhar com micro-serviços, sob uma arquitetura distribuída e modularizada, tal que seria possível criar em outras linguagens, de outras maneiras, masters e slaves de forma a continuar utilizando o repository sem problemas.

SOA é um caminho interessante para modelagem de grandes sistemas, apesar de seus problemas já citados, suas vantagens pode render um projeto inteligente, desde que seja bem analisado a necessidade dessa arquitetura no contexto.

## Referências

- ALMEIDA, A. *Arquitetura de Serviços ou Monolítica?* 2015. Disponível em: <<http://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica>>. Citado na página 15.
- BOLDRINI, J. L. Álgebra linear. Depto. de Matemática da Universidade Estadual de Campinas, 2009. Citado na página 16.
- CIRNE, W. Running bag-of-tasks applications on computational grids: The mygrid approach. Parallel Processing, 2003. Citado na página 13.
- KSHEMKALYANI, A. D. Distributed computing. principles, algorithms, and systems. Cambridge University Press, 2008. Citado 2 vezes nas páginas 13 e 16.
- ORACLE. *The Java EE6 Tutorial: What Are Web Services?* 2013. Disponível em: <<http://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html>>. Citado na página 15.
- TANNENBAUM, A. S. Distributed computing. principles, algorithms, and systems. Pearson Education, Inc, 2007. Citado na página 15.
- XINYANG, F.; SHEN, J.; FAN, Y. Rest: Uma alternativa a rpc para arquiteturas web services. Future Information Networks, 2009. Citado 2 vezes nas páginas 15 e 16.





# Anexos



## ANEXO A – Roteiro

