



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 – Breve imagem do funcionamento do Web Service proposto. Fonte. . . .	22
Figura 2 – Função compartilhada e Função de invocação do arquivo tasks.py . . .	23
Figura 3 – Produto de matrizes sem concorrência	24
Figura 4 – Chamada de <code>print_matrix_result()</code> e os Workers calculando suas tasks	25
Figura 5 – Cálculo de forma não distribuída de um produto de matrizes 1000x1000	26
Figura 6 – Cálculo de forma não distribuída de um produto de matrizes 250x250 .	27
Figura 7 – Cálculo de um produto de matrizes 250x250 no Web Service	28
Figura 8 – Configuração das VMs criadas para workers	29

Lista de tabelas

Lista de abreviaturas e siglas

VM Virtual Machine ou Máquina Virtual

API Application Programming Interface

123 Isto é outro número

lauro cesar este é o meu nome

Lista de símbolos

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
\in	Pertence

Sumário

1	OBJETIVOS	11
2	INTRODUÇÃO	13
3	DESENVOLVIMENTO	15
3.1	Entendendo Web Services	15
3.2	Entendendo REST	15
3.3	Arquitetura proposta	16
3.4	Problema de Produto de Matrizes	16
3.5	Solução proposta	16
3.6	Message Broker	17
3.7	Ambiente e Configuração	17
3.8	Execução do experimento	17
4	RESULTADOS	31
4.1	Sobre a Solução	31
4.2	Falhas e Tolerâncias	31
4.3	Opinião Pessoal	31
	Referências	33
	ANEXOS	35
	ANEXO A – ROTEIRO	37

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.

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](#)), RPC é um estilo arquitetural de Web Services. Nesse contexto é ainda defendido pelo autor a abordagem REST como uma alternativa a RPC.

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 Transfer) 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.

3.3 Arquitetura proposta

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 tem-se:

- O servidor distribui Pairs, que consistem em chave e valor, tal que o primeiro é um identificador da tarefa e o segundo se refere às informações para execução da tarefa.
- Tem que ser executadas em Bag-Of-Task, ou seja, cada tarefa é independente.
- Espera-se que os processos escravos repetidamente consultem por tarefas disponíveis no repositório (processo mestre).
- A interface do repositório deve conter as seguintes rotinas, tal que seja possível inserir uma pair, remover uma pair retornando seu valor e ler sem remover.

1. pairIn(chave, valor)
2. pairOut(chave)
3. readPair(chave)

3.4 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

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

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.

3.5 Solução proposta

Para a solução desse problema será utilizado a API Celery, que utiliza várias ferramentas em conjunto que satisfazem a arquitetura proposta. A Figura [1](#) ilustra o

funcionamento deste Web Service, tal que a aplicação Django (framework da linguagem Python) oferece uma macro tarefa que é armazenada num Message Broker, similar ao repositório de tarefas, executada em diferentes Celery Workers, que são nossos processos escravos. Ao final as tarefas são armazenadas em banco, mas para nosso experimento obter o resultado de forma não persistível já é suficiente.

3.6 Message Broker

Um Message Broker atua como uma aplicação do nível gateway em um sistema de enfileiramento de mensagens. Sua finalidade principal é converter mensagens recebidas tal que elas podem ser entendidas pela aplicação de destino. [...] No coração de um Message Broker permanece um repositório de regras e programas que transformam uma mensagem do tipo T1 para outro T2 ([TANNENBAUM, 2007](#))

Para a implementação da solução foi utilizado o RabbitMQ, uma solução de código aberto do Message Broker. Entende-se que o Message Broker junto ao Shell do Python irá compor o repositório de tarefas, tal que esses irão atender aos requisitos de implementação das operações de pairIn, pairOut e readPair.

3.7 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. Para simular cada worker server, foi utilizado um ambiente virtualizado no Vagrant, usando Debian Wheezy 7. O código foi escrito na linguagem Python utilizando o interpretador na versão 3.4. O Django foi utilizado apenas como gerente de configuração de dependências e organização de projeto, está sendo utilizado na versão 1.9.8. A versão do Celery utilizada é a 4.0. O repositório que contém os arquivos utilizados no experimento se encontram em: [Link Github](#). Para acessar os arquivos do repositório no computador basta instalar o git e usar o comando `git clone <url do projeto>.git`, que o projeto será baixado para a pasta atual do terminal.

3.8 Execução do experimento

- Primeiramente foi criado um projeto no django usando o comando abaixo no terminal.

```
$ django-admin startproject django
```

Com isso é criado um arquivo `manage.py` que irá ajudar na configuração de dependências utilizada.

- Em seguida foi utilizado o mesmo app proj de um [repositório github do projeto celery](#), tal que esse modulo já está configurado uma instância do Celery com configuração para auto descobrir outras tarefas no diretório do projeto django. Para configuração do nosso módulo de tarefas compartilhadas é preciso de um Broker, portanto temos que se certificar antes que ele esteja funcionando. O RabbitMQ é um daemon que fica executando sobre localhost, para saber do status desse serviço basta executar:

```
$ sudo rabbitmqctl status
```

- Agora basta cadastrar um módulo que terá uma tarefa compartilhada no arquivo `proj/settings.py` que o celery irá introduzir ela no repositório de tarefas para execução distribuída nos workers ligados. Para criação do app de produto de matrizes podemos utilizar:

```
$ python manage.py startapp celery_matrix
```

Primeiramente esse comando foi utilizado com o NoConMaMu, mas por ser um nome difícil se reproduzir foi renomeado para `celery_matrix`.

- Em seguida foi criada uma pasta com vários arquivos, alguns deles não são úteis para a solução proposta, portanto podem ser excluídos. Basta permanecer com `apps.py`, `__init__.py`, e a pasta `migrations` para possíveis futuras evoluções em que seria necessário um banco de dados.
- Iterativamente o programa para calcular matrizes foi evoluído, o resultado final é o arquivo `tasks.py`, que contém uma série de funções dentre o qual apenas duas são necessárias para o entendimento (Figura 2), a primeira é a tarefa compartilhada que calcula cada linha da nova matriz, a segunda é uma rotina que lê duas matrizes de um arquivo e repassa para vários que vários workers faça o cálculo.
- Perceba que a função `calculate_line` não é chamada de forma convencional. Em python tudo é objeto, mesmo funções, o que permite ao celery “Decorar” qualquer método, ou seja adicionar um comportamento a um método anotado com `shared_task` para que possa ser utilizado de forma distribuída.
- Dessa forma, em uma interface iremos chamar essa função `print_matrix_result` e ele irá devolver uma matriz calculada de forma distribuída. Para tal, em uma mesma VM que está sendo executando o RabbitMQ como um daemon, podem ser criadas instâncias de worker através do comando:

```
$ celery -A proj worker -l info
```

- Se a tarefa estiver cadastrada corretamente irá aparecer no log do comando algo como:

```
[tasks]
. celery_matrix.tasks.calculate_line
. proj.celery.debug_task
```

- Após instanciado a quantidade de workers que julgar necessário, podemos pedir para se calcular uma matriz via terminal do python, dentro do nosso diretório do projeto django, basta usar o comando:

```
$ python3 ./manage.py shell
```

- Dentro do terminal iterativo do python é necessário importar a função `print_matrix_result`, que irá utilizar das shared tasks:

```
> from celery_matrix.tasks import print_matrix_result
```

- Ao se executar a função `print` importada obtemos na tela o resultado do produto das matrizes A e B, presentes nos arquivos da pasta `input`, veja a (Figura 4). Nos workers ligados (nos terminais de baixo e o em cima a direita) é possível ver também o resultado de cada linha.

- Para testar a solução, foi criado um script dentro de `celery_matrix` que gera matrizes quadradas em um arquivo. O script feito em python se chama `matrix_generator.py`, para executá-lo basta usar o comando:

```
$ python3 matrix_generator.py <nome_do_arquivo> <ordem da matriz>
```

- Foram testadas matrizes de baixa ordem no Web Service, tal que foi possível atestar seu funcionamento mínimo.
- Foram geradas com esse comando duas matrizes na pasta `input` dentro de `celery_matrix`. As duas de ordem 100, o que gerará um número elevado de operações. Nesse ponto foi questionado qual a ordem de matriz é necessário para se testar a eficiência de um Web Service distribuído.

- Dado A e B como matrizes a serem multiplicadas podemos constatar que:
 - Para gerar cada linha da nova matriz, a linha A, multiplica cada elemento da matriz B somando os resultados de cada coluna multiplicada em B para formar um elemento da linha da matriz resultante.
 - Dado n como a ordem de grandeza da matriz, temos para achar um novo elemento da matriz resultante, n operações de multiplicação mais n operações de soma, pois cada multiplicação é somada com outra, logo tem-se 2n operações.

- Esse ciclo acima se repete n vezes, pois cada linha de A gera uma nova linha da matriz resultante, logo temos que o número de operações é dado pela formula:

$$operacoes(n) = 2n * n \quad (3.1)$$

- Para matrizes de ordem 100 temos 20.000 operações, não é um número suficientemente alto. Calculando o produto entre duas dessas matrizes em um programa convencional que não usa de paralelismo, nota-se um tempo de execução de 0.41 segundos. Para tal mensuramento foi utilizado o comando `time` do SO, com o script `matrix_calculator.py` que está dentro de `celery_matrix` e não usa de nenhuma técnica de concorrência. Para executar o script foi usado:

```
$ time python3 matrix_calculator.py
```

- No nosso Web Service foi utilizado a biblioteca `time` do python para medir esse tempo de execução. Foi atestado um tempo muito superior a aplicação sem concorrência: Em média 83.52 seg com dois workers executando(Figura 4)
- Uma explicação possível é que o tempo de comunicação é muito alto, não compensando para a ordem de 100 elementos. Para tal se iniciou um terceiro worker. Se o trabalho executar em maior tempo, nossa hipótese está comprovada. Se o tempo diminuir significa que as tarefas estão alocadas de forma errada, ou o gerenciamento da API não é eficiente rodando tudo em uma única VM.
- Conforme a figura Figura 3, o tempo de processamento foi menor, o que significa que se utilizado uma quantidade maior de processos escravos, menor será o tempo de execução até um ponto em que o tempo de comunicação será um gargalo.
- Uma outra hipótese é que o processamento utilizado em uma matriz de ordem 100 não é gargalo nenhum para uma aplicação não concorrente. Tal que a concorrência só irá ser lucrativa quando o número de elementos aumentar significativamente.
- Para tal foram geradas duas matrizes A e B de ordem 1000. O que implica em 2.000.000 de operações segundo a formula 3.1.
- Ao ser executado o script `matrix_calculator.py` da mesma forma dos passos anteriores, obtemos um tempo de 10 minutos e 13.979 segundos, conforme mostra a Figura 5
- Com o Web Service não foi possível calcular, em mais de 1 hora de execução não houve retorno de resultados.
- Decidiu-se reduzir a ordem para 250, o resultado para o script sem concorrência foi 6.5 segundos (Figura 6) e do Web Service 207.6 segundos, Figura 7

- Como todo o web service estava rodando em uma única VM, abriu-se a hipótese que o Celery não estava conseguindo trabalhar com as tarefas de forma paralela. Para comprovar isso foram criadas 2 novas VMS. Na antiga permaneceria a VM com o RabbitMQ e o Cliente Python, as outras duas pretendia-se utilizar apenas na execução dos workers.
- A configuração realizada para usar worker remoto foi seguindo esse [tutorial](#).
- As VMs para execução do worker foram configuradas seguindo a configuração da Figura 8, porém não se conseguiu comunicação externa em nenhuma delas, nem mesmo uma com a outra. O que tornou impossível continuar o experimento utilizando VMs.
- A hipótese de que as tarefas não estão bem divididas é uma possibilidade que fica apenas no campo da reflexão, pode-se ver que se existisse um host rodando com um worker para cada tarefa, teríamos um tempo de execução muito bom, uma vez que se observado no terminal das figuras de execução do Web Service, percebe-se que cada task executa em um tempo muito baixo.
- Os experimentos foram finalizados.

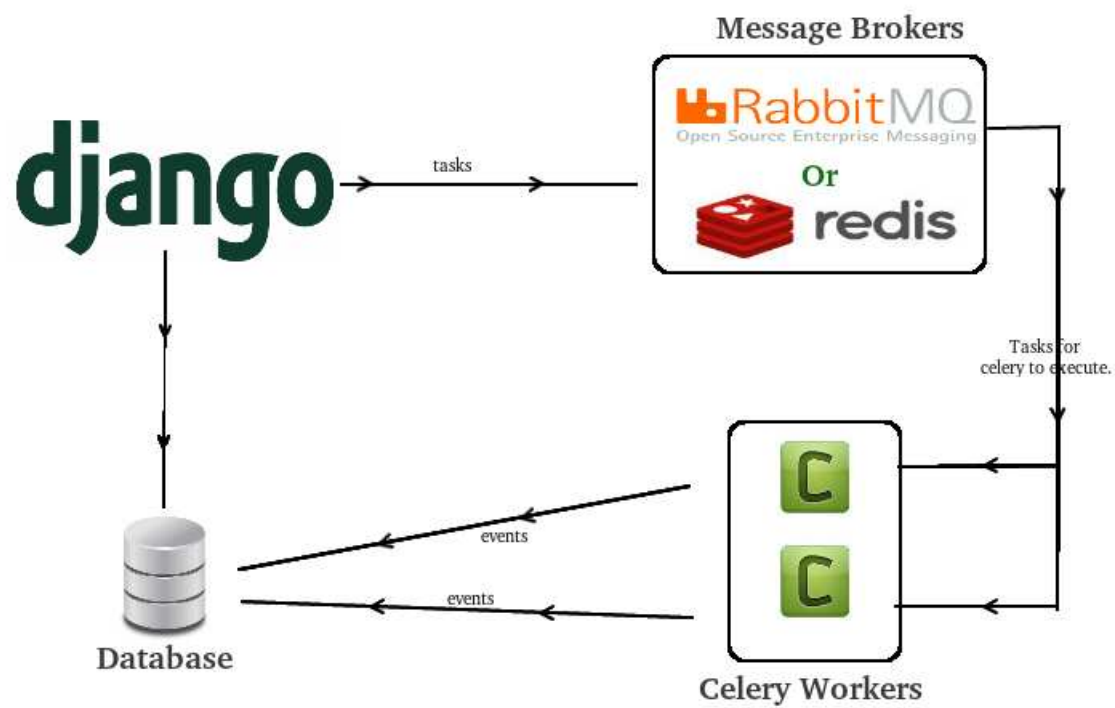


Figura 1 – Breve imagem do funcionamento do Web Service proposto. [Fonte](#).


```
17
18 # Given a line of matrix A, multiplied by another matrix B,
19 # we have a new row of matrix.
20 @shared_task
21 def calculate_line(A_line, B):
22     new_line = []
23
24     column_number = len(B[0])
25
26     for column in range(column_number):
27         element = calculate_element(A_line, [c[column] for c in B])
28         new_line.append(element)
29
30     return new_line
31
```

```
42
43 def print_matrix_result():
44     # Sample 3x3
45     A = read_matrix('celery_matrix/input/A.matrix')
46
47     # Sample 3x4
48     B = read_matrix('celery_matrix/input/B.matrix')
49
50     R = []
51
52     for A_line in A:
53         result = calculate_line.delay(A_line, B)
54         while not result.ready():
55             pass
56         R.append(result.get())
57
58     print_matrix(R)
```

Figura 2 – Função compartilhada e Função de invocação do arquivo tasks.py





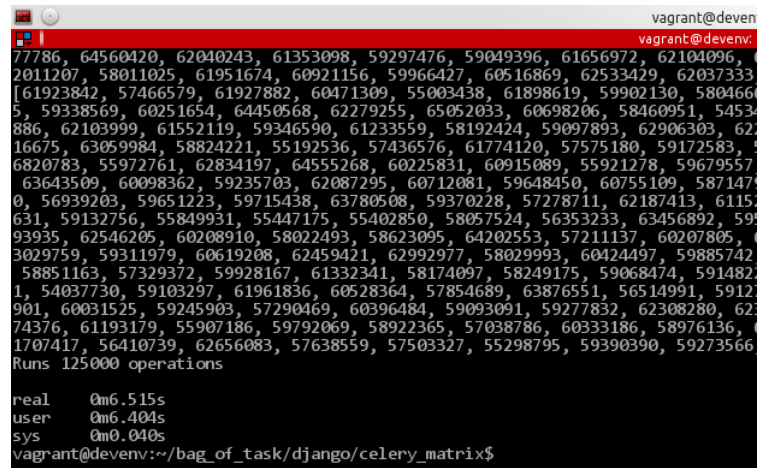

```

vagrant@devenv: ~/bag_of_task/django/celery/matrix
567280, 25130206, 258448628, 254216167, 256091850, 252021575, 244037186, 240784472, 254354126, 256388378, 24897662, 247176743, 2465
6264, 255321586, 260897174, 258651416, 245832732, 248983724, 245996843, 257119531, 240677919, 235420062, 246529295, 241209845, 251405
40, 247067934, 256337893, 249175920, 244264208, 244287762, 250699707, 243045360, 245644422, 254980456, 251330028, 256101175, 25452787
, 251769508, 252606449, 250384753, 244781767, 239778430, 259295658, 254868172, 258940726, 248778600, 246595035, 249480805, 255163635,
252031280, 247990564, 257685198, 246572771, 247689290, 248050645, 249679457, 2526686812, 250815777, 256334460, 252202478, 244139180, 2
1136891, 244385386, 252108202, 248853012, 249875143, 260886240, 251144069, 250433349, 244288483, 236357644, 244561850, 244414702, 250
27601, 247545680, 251630879, 256119982, 254025775, 250048581, 260137785, 247603323, 243186071, 249860366, 242415931, 244044775, 25547
299, 25526606, 246117468, 250948331, 244844447, 249276006, 252808374, 248593227, 259320321, 248452125, 242822199, 256094405, 2504105
8, 255442744, 257286767, 241406083, 241432641, 246978684, 250055496, 244736132, 244730932, 245767049, 238785681, 249648383, 253217870
250516420, 245427512, 239699072, 248092318, 253296416, 248022408, 249205372, 253001722, 252356900, 245910778, 240585781, 249893491,
48822954, 251583471, 243504221, 240358998, 251285289, 252049421, 260174863, 254351169, 246039888, 252269355, 248240763, 242621695, 24
823153, 247885551, 260089684, 255234583, 249219391, 249024970, 237500139, 239779172, 238704877, 259560895, 241940237, 250845911, 2533
9172, 250931399, 240554244, 245099000, 254233391, 252084204, 249645703, 246755596, 248211537, 246107110, 249673893, 248299820, 247863
83, 245782319, 243855206, 251958260, 244838847, 254839797, 241228319, 250572942, 247709372, 252720580, 246732567, 249415189, 25337506
, 245193233, 248762002, 252874646, 244998856, 238638982, 254885509, 242846935, 247886354, 248006525, 232167366, 245160533, 241597279,
252964204, 247463679, 251871349, 244404162, 257937801, 249970341, 250042020, 256515521, 243450036, 254875520, 249339252, 243644966, 2
3694415, 247368801, 252787399, 253331753, 244541849, 252528354, 250373343, 248662599, 245951736, 249464760, 246124563, 237705322, 251
00305, 243735825, 243174088, 252357354, 246440937, 237646877, 249499846, 248802577, 261107371, 243307946, 250375438, 243628875, 25311
691, 240808378, 248952125, 250164184, 247388228, 245355167, 249882858, 249454603, 256641047, 242705465, 249719370, 263104977, 2452895
4, 253498679, 247709015, 248537605, 248592148, 256144256, 244873041, 248891114, 241308635, 248570063, 253082750, 246352863, 242592365
251326227, 248671276, 244080184, 250517278, 246016099, 251666641, 245192405, 24187755, 244749826, 239416739, 241819350, 245403955,
40117482, 246976519, 250092560, 237909882, 243561987, 246352863, 249486714, 248182583, 250020681, 250201518, 244729006, 248432627, 24
798286, 247355700, 251947410, 250413116, 252094151, 250877955, 240025448, 245846261, 247409272, 242327902, 245754984, 245098431, 2431
4723, 245121423, 238913751, 250468436, 236977676, 248071896, 252810958, 250567386, 240447889, 246076901, 244627537, 241810363, 241477
96, 241397363, 240479425, 234934339, 249939303, 256806267, 244495830, 249692050, 242166882, 247847151, 245444558, 249534278, 25268599
, 249697483, 257144941, 244405548, 258797119, 245503952, 238814624, 25322107, 244212536, 247724269, 238175866, 253719346, 254050756,
253065340, 244586740, 256603906, 244188744, 253677547, 240888455, 250770282, 250132841, 248844024, 251032795, 250070705, 252563703, 248
0469246, 240219172, 248423117, 249587743, 253629062, 255351587, 248507683, 250007833, 245441365, 246945675, 248760758, 252563703, 248
54849, 244189477, 248110292, 249064928, 246459190, 237475526, 253740810, 251126649, 251623222, 242979239, 247096146, 249540630, 24672
103, 247690397, 250267313, 246820117, 249198321, 240985898, 256997922, 254188939, 251659566, 250521890, 257019118, 249122263, 2423763
7, 236591056, 249546053, 242037840, 245897656, 236259812, 243354065, 249529448, 249192004, 242704948, 251257630, 252976492, 248127692
Runs 200000 operations

real    10m13.978s
user    9m40.216s
sys     0m0.792s
vagrant@devenv:~/bag_of_task/django/celery/matrix$

```

Figura 5 – Cálculo de forma não distribuída de um produto de matrizes 1000x1000



```
vagrant@devenv:~$ python bag_of_task/django/celery_matrix.py
77786, 64560420, 62040243, 61353098, 59297476, 59049396, 61656972, 62104096,
2011207, 58011025, 61951674, 60921156, 59966427, 60616869, 62533429, 62037333
[61923842, 57466579, 61927882, 60471309, 55003438, 61898619, 59902130, 580466
5, 59338569, 60251654, 64450568, 62279255, 65052033, 60698206, 58460951, 5453
886, 62103999, 61552119, 59346590, 61233559, 58192424, 59097893, 62906303, 62
16675, 63059984, 58824221, 55192536, 57436576, 61774120, 57575180, 59172583,
6820783, 55972761, 62834197, 64555268, 60225831, 60915089, 55921278, 59679557
63643509, 60098362, 59235703, 62087295, 60712081, 59648450, 60755109, 587147
0, 56939203, 59651223, 59715438, 63780508, 59370228, 57278711, 62187413, 6115
631, 59132756, 55849931, 55447175, 55402850, 58057524, 56353233, 63456892, 59
93935, 62546205, 60208910, 58022493, 58623095, 64202553, 57211137, 60207805,
3029759, 59311979, 60619208, 62459421, 62992977, 58029993, 60424497, 59885742
58851163, 57329372, 59928167, 61332341, 58174097, 58249175, 59068474, 591482
1, 54037730, 59103297, 61961836, 60528364, 57854689, 63876551, 56514991, 5912
901, 60031525, 59245903, 57290469, 60396484, 59093091, 59277832, 62308280, 62
74376, 61193179, 55907186, 59792069, 58922365, 57038786, 60333186, 58976136,
1707417, 56410739, 62656083, 57638559, 57503327, 55298795, 59390390, 59273566
Runs 125000 operations
real    0m6.515s
user    0m6.404s
sys      0m0.040s
vagrant@devenv:~/bag_of_task/django/celery_matrix$
```

Figura 6 – Cálculo de forma não distribuída de um produto de matrizes 250x250



```

vagrant@devenv:~/bag_of_tasks/django<-3>
vagrant@devenv:~/bag_of_tasks/django/celery_matrix/83x24
68, 63300844, 60644970, 57951327, 54103833, 64106647, 58482814, 58306864, 56080563,
63420604, 62218198, 57323671, 59987158, 60910812, 55103848, 61272737, 57205286, 58
030935, 55027454, 65417946, 58028523, 58559007, 59892818, 57099218, 61960805, 61976
242, 57733095, 64142173, 53910995, 62605205, 65319078, 65546811, 57904579, 61606345,
59661762, 59873012, 58718348, 62748435, 60933396, 59441240, 61168429, 60021208, 6
0095651, 59308524, 58607207, 56043470, 59646565, 59086932, 57980473, 56783798, 5734
9285, 56621494, 64113127, 63149144, 56954418, 60919319, 57155434, 55083692, 6269903
2, 57639353, 57016522, 64684244, 56394489, 61381426, 59597014, 62710371, 64933343,
57359160, 60321655, 60712200, 58824280, 61782524, 56138213, 58233089, 60087575, 607
66650, 57445943, 61152042, 61173324, 60156976, 58665432, 61406105, 60308190, 588091
04, 56543094, 63924528, 60411556, 59650542, 59821813, 59845376, 58253735, 55255489,
58578838, 57477309, 63715420, 59538748, 6062204, 60723783, 59129431, 57797250, 59
140746, 59591289, 59053231, 58174247, 58243658, 61620732, 62024843, 56416074, 61342
151, 61705284, 61370812, 55552858, 60217897, 64527595, 63511925, 59403617, 62071880
58944752, 61062641, 59762308, 62422436, 60040704, 56518048, 56979323, 62146503, 5
7866321, 61796192, 59832421, 56758364, 57692367, 56012100, 59781093, 59088192, 6063
4685, 57039394, 58967872, 56719651, 56971869, 55493209, 60802574, 64365240, 5510228
7, 59977007, 58672529, 57437884, 61276366, 61364944, 59459397, 58947056, 60721021,
645519, 60744261, 59124177, 61144528, 61400959, 59219109, 58861333, 5852839, 657622
64, 63013493, 51529702, 58264918, 56555382, 57874706, 59018159, 56737441, 56213761,
59610343]
Executed in %.3f sec 207.6463050842852
>>>

vagrant@devenv:~/bag_of_tasks/django<-2>
vagrant@devenv:~/bag_of_tasks/django/82x21
210, 71537407, 71347459, 65941177, 67147992, 71389255, 65424304, 71392939, 6591929
4, 66236891, 63038201, 75960823, 67874522, 71868278, 69050009, 66978881, 72122310,
72813429, 66971011, 71594411, 646...
[2017-05-23 23:14:15.947: INFO/MainProcess] Received task: celery_matrix.tasks.cal
culate_line [69acd74d-10db-48f4-8f8c-c3ae0a8bd045]
[2017-05-23 23:14:16.339: INFO/PoolWorker-1] Task celery_matrix.tasks.calculate_l
ine [69acd74d-10db-48f4-8f8c-c3ae0a8bd045] succeeded in 0.040025599999999999s: [60803
537, 61565226, 62757567, 60936799, 57901277, 59763125, 54908972, 60808061, 5961603
6, 61794275, 57207652, 61473997, 61168379, 61173228, 59281409, 52955255, 65723896, 5
61459572, 57207652, 61473997, 61168379, 61173228, 59281409, 52955255, 65723896, 5
9659268, 56100002, 63004828, 57404122, 62554420, 61360249, 62314633, 58781165, 578
23546, 61238612, 58859437, 63442246, 60943373, 58807076, 58742165, 58743710, 62480
572, 60324863, 57354376, 58844081, 63542070, 58943295, 62098338, 59816591, 5823882
1, 54357536, 64779542, 55132292, 61484070, 60859830, 57605895, 59187232, 6048365
1, 57575532, 55986204, 57844579, 58611525, 61990482, 57716948, 59684771, 57957541, 6
1393943, 62161233, 60400869, 60645584, 59763885, 59497879, 63460765, 57794566, 607
25868, 63300844, 60644970, 57951327, 54103833, 64106647, 58482814, 58306864, 56080
563, 63420604, 62218198, 57323671, 59987158, 60910812, 55103848, 61272737, 5720528
6, 58030933, 55027454, 65417946, 58028523, 58559007, 59892818, 57099218, 61960805,
61976242, 57733095, 64142173, 53910995, ...]

vagrant@devenv:~/bag_of_tasks/django<-3>
vagrant@devenv:~/bag_of_tasks/django/80x24
61687421, 56904552, 58853226, 61467782, 61923172, 62682950, 61125537, 61950638,
61465388, 60437749, 60706650, 62046493, 64168372, 64160507, 63539422, 61851247,
62384149, 59150068, 64927900, 63478485, 64565730, 59184581, 56600749, 63593591,
59927751, 59808380, 60083366, 65822080, 64986491, 61350440, 62822932, 63591845,
59373611, 63791234, 59490445, 60924482, 56785375, 66508452, 61033722, 62974081,
60585454, 57384885, 65496835, 62753220, 59374572, 63851325, 537...
[2017-05-23 23:14:13.941: INFO/MainProcess] Received task: celery_matrix.tasks.c
alculate_line [3d34d766-72c7-4a79-87dc-1a80a22dfdb0]
[2017-05-23 23:14:14.319: INFO/PoolWorker-1] Task celery_matrix.tasks.calculate_l
ine [3d34d766-72c7-4a79-87dc-1a80a22dfdb0] succeeded in 0.03188528600003375s: [
62907040, 64933466, 63576294, 64329376, 58865057, 61925146, 61499630, 61789419,
64074286, 65458226, 58888321, 63131070, 6280401, 66341073, 63743321, 61156628, 58525650,
61600344, 63458226, 62365120, 6280401, 66341073, 63743321, 61156628, 58525650,
67090219, 64047729, 59371613, 65861722, 61544214, 65660745, 64771891, 64125718,
62925601, 62562759, 65588014, 57808572, 68003569, 64685381, 62040308, 66236687,
63681448, 63823099, 65016332, 62580291, 62613707, 68486920, 62767988, 66179675,
62135533, 61755513, 59682689, 66255396, 59257627, 65872022, 63183949, 61434990,
60674501, 64351004, 61710526, 60816616, 62687939, 61584525, 59465992,
65665144, 60647674, 61232718, 61635519, 63712318, 63709453, 62998016, 64711393,
63657489, 61306745, 68098710, 62587254, 65477920, 58900013, 58726387, 67200416,
64519845, 61036160, 61000946, 67584550, 64466298, 58161312, 62858160, 67009447,
61574495, 67326235, 62023114, 61223116, 57004490, 67265247, 62786311, 62415235,
62530805, 61726424, 65254261, 67743908, 62591558, 65008004, 543...

```

Figura 7 – Cálculo de um produto de matrizes 250x250 no Web Service

A screenshot of a VIM editor window showing a Vagrantfile configuration. The window title is 'Vagrantfile (/workspaces/bag_of_tasks) - VIM'. The file content is a Vagrantfile for two VMs, 'alpha' and 'beta'. It includes comments about Vagrant configuration options and network settings. The VM 'alpha' is defined with hostname 'worker01', IP '10.0.0.11', and network 'private_network'. The VM 'beta' is defined with hostname 'beta', IP '10.0.0.10', and network 'private_network'. The configuration is enclosed in a 'config.vm.define' block for each VM.

```
7 you're doing...
8 Vagrant.configure("2") do |config|
9   # The most common configuration options are documented and commented below.
10  # For a complete reference, please see the online documentation at:
11  # https://docs.vagrantup.com~
12
13  # Every Vagrant development environment requires a box. You can search for
14  # boxes at https://atlas.hashicorp.com/search~
15  config.vm.box = "debian/wheezy64"~
16
17  config.vm.define :alpha do |alpha|
18    alpha.vm.hostname = "worker01"~
19    alpha.vm.network "private_network", ip: "10.0.0.11"~
20  end~
21
22  config.vm.define :beta do |beta|
23    beta.vm.box = "hashicorp/precise64"~
24    beta.vm.network :private_network, ip: "10.0.0.10"~
25    beta.vm.hostname = "beta"~
26  end~
27
28  config.vm.define "worker02"~
29
```

Figura 8 – Configuração das VMs criadas para workers

4 Resultados

4.1 Sobre a Solução

O Web Service proposto é uma arquitetura robusta e personalizável que pode lidar com os mais diferenciados contextos. O Celery conta com uma boa documentação e uma comunidade bem ativa. O RabbitMQ já é conceitado como Message Broker, também é uma ferramenta robusta, tal que seria necessário demasiados experimentos para conhecê-lo melhor.

4.2 Falhas e Tolerâncias

A solução é evidentemente mais funcional para vários hosts, onde se pode ter um processamento assíncrono de um ou mais workers em grande escala, tal que irão executar a tarefa que em um processo de forma sequencial, seria inviável. Para tal seria também necessário senso crítico para distribuir bem a aplicação, tal que ligar uma série de workers em apenas um host irá apresentar recorrentes problemas de concorrência, nesse caso, rede e clock principalmente.

O Web service pode ser configurado ainda pra rodar de forma Virtualizada, apesar de o experimento não ter atingido com sucesso o uso de múltiplas VMs, isso é uma possibilidade. Tal que se consiga utilizar de um equipamento o máximo de seu recurso.

4.3 Opinião Pessoal

O trabalho se revelou com tema muito aberto e de prazo muito curto, apesar de ter sido de grande aprendizado. No início foi pensado em duas soluções, o Celery com produto de matrizes e um Web Service Rest que calculasse fractais. Porém pelo prazo não foi possível entregar esse segundo, e o primeiro foi sempre a solução que pareceu mais segura.

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 2 vezes nas páginas 15 e 17.
- XINYANG, F.; SHEN, J.; FAN, Y. Rest: Uma alternativa a rpc para arquiteturas web services. Future Information Networks, 2009. Citado na página 15.

Anexos

ANEXO A – Roteiro

