UNIVERSIDADE DO ESTADO DE SANTA CATARINA
CURSO DE SISTEMAS DE INFORMACAO

LEONARDO AUGUSTO METZGER

OPTVM: UM SERVIÇO PARA O SUPORTE DA OTIMIZAÇÃO DA MIGRAÇÃO DE VMS

TRABALHO DE CONCLUSÃO DE CURSO

SÃO BENTO DO SUL
2019

LEONARDO AUGUSTO METZGER

OPTVM: UM SERVIÇO PARA O SUPORTE DA OTIMIZAÇÃO DA MIGRAÇÃO DE VMS

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informacao da Universidade do Estado de Santa Catarina, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Mário Ezequiel Augusto

SÃO BENTO DO SUL
2019

Aos meus professores, pais, irmão e amigos que me ajudaram e me motivaram durante o desenvolvimeto.

AGRADECIMENTOS

A ciência é o que nós compreendemos suficientemente bem para explicar a um computador. A arte é tudo mais. - *Donald Knuth*

RESUMO

METZGER, Leonardo. OptVM: Um serviço para o suporte da otimização da migração de VMs. 2019. 32 f. Trabalho de Conclusão de Curso – Curso de Sistemas de Informacao, Universidade do Estado de Santa Catarina. São Bento do Sul, 2019.

@TODO
Palavras-chave: VM. Optimization. Rest.

ABSTRACT

METZGER, Leonardo. Title in English. 2019. 32 f. Trabalho de Conclusão de Curso – Curso de Sistemas de Informacao, Universidade do Estado de Santa Catarina. São Bento do Sul, 2019.

@TODO
Keywords: VM. Optimization. Rest.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
EA	Evolutionary Algorithms
GA	Genetic Algorithms
HATEOAS	Hipertext as the Engine of Application State
JSON	JavaScript Object Notation
MOO	Otimização multiobjetivo
OF	Objective Function
PM	Physical Machine
VM	Virtual Machine
XML	eXtensible Markup Language

LISTA DE ALGORITMOS

1 INTRODUÇÃO

Com a evolução da computação em nuvem, surgiram necessidades cada vez maiores de utilizar ao máximo o poder dos computadores sem sobrecarregar os mesmos, e os componentes que o fazem funcionar, assim como o uso de energia. Estas necessidades surgem para atender requisitos de diminuição de custos, aumento de desempenho, diminuição no consumo de energia, entre outros objetivos que fazem com que usuários de computação em nuvem e empresas que usam este tipo de serviço obtenham vantagem no uso dela.

Para isso, é muito comum que para otimizar o uso dos computadores de um ambiente em nuvem, os provedores utilizem o mecanismo de virtualização. Hoje, os *datacenters* são compostos por máquinas físicas(PMs) e máquinas virtuais(VMs), sendo que, cada PM normalmente possui pelo menos uma ou mais VMs. Essa utilização das VMs permite que seja construído um ambiente flexível em relação a organização e quantidade de VM por Hosts em cada *datacenter*.

Um cenário em que temos a possibilidade de utilizar as PMs como host de múltiplas VMs, é possível que as VMs da nuvem sejam organizadas de diferentes maneiras em relação as PMs. A decisão de organizar a nuvem de uma maneira ou de outra, envolvem os objetivos de quem está gerenciando a nuvem.

Os objetivos podem ser os mais variados. Por exemplo, uma empresa que use o serviço da nuvem pode querer ter um alto desempenho, assim como pode querer ter o menor custo possível. Por esse motivo, existem pesquisas que buscam maneiras de melhorar a alocação e realocação de VMs, e uma das formas de resolver este tipo problema é utilizar a otimização multiobjetivo (MOO).

A migração de uma VM envolve algumas etapas, como, a descoberta da necessidade de migração, a escolha de uma VM a ser migrada e a escolha de um host de destino para essa VM. A etapa em que este trabalho está interessado é a escolha de um host de destino para a VM. Considerando que uma migração seja considerada cara do ponto de vista computacional, o momento da migração deve ser bem escolhido para evitar que a migração da VM feita não gere prejuízos ou problemas maiores do que a própria sobrecarga do host. Assim como o momento da migração é importante, a escolha de um destino também é, pois o host selecionado tem que melhorar a maneira em que os hosts estão organizados naquele determinado momento, para que não haja novas migrações por conta da migração inicial.

Este trabalho tem papel de servir como apoio para a migração de VMs em ambientes de computação em nuvem. Isto é feito através de uma abordagem em que um gerenciador de nuvem, que precise migrar uma VM, possa utilizar um serviço que selecionará as melhores opções de host para fazer a migração de uma VM. O serviço possui uma abordagem que utiliza algoritmos que fazem uma seleção dos melhores hosts baseando-se nos objetivos do consumidor do serviço. Contudo, o serviço é uma solução caixa preta, esta característica traz uma grande vantagem, o usuário não precisa conhecer nada sobre os algoritmos utilizados, precisa apenas utilizar a interface que é definida pelo serviço. A interface do serviço é baseada em *webservices*, utilizando padrões conhecidos para facilitar a integração dos gerenciadores das núvens computacionais.

Este trabalho apresenta uma aplicação prática da construção de um *webservice* utilizando padrões conhecidos na indústria. Além do *webservice*, o trabalho também apresenta uma aplicação de algoritmos de otimização multiobjetivo para um problema que existe hoje. Além disso, são feitos testes para a avaliação dos resultados e gerados métricas através deles.

1.1 METODOLOGIA

Para o desenvolvimento desse trabalho, foi feita pesquisa exploratória em possíveis soluções para resolução do problema e após a etapa de exploração, foram feitas avaliações quantitativas da aplicação da solução.

Existem muitas ferramentas tanto para a construção de *webservices* quanto para a otimização multiobjetivo. Para o desenvolvimento de uma solução, devem ser escolhidas as ferramentas que melhor se adequam para resolver o problema em questão. Na fase de exploração, diversas referências bibliográficas foram utilizadas para a busca e avaliação de ferramentas disponíveis para utilizar no desenvolvimento.

Após as ferramentas escolhidas, foi desenvolvido a solução. A criação do *webservice* foi feito em JAVA, porém, qualquer linguagem de programação que consiga fazer requisições HTTP. Por esse motivo, foi feita uma avalação qualitativa dos resultados obtidos com a solução. Os resultados foram quantificados e avaliados com diferentes quantidades de *clouds/hosts*, utilizando métricas de requisições por segundo (RPS) e tamanho da requisição.

Os resultados obtidos após a avaliá-los foram documentados no capítulo quatro deste trabalho, assim como as variáveis utilizadas para os testes feitos.

1.2 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado em seis capítulos. O primeiro, contextualiza e descreve o problema e a forma de resolvê-lo Já o segundo capítulo, dá embasamento teórico necessário para o entendimento dos demais capítulos. No terceiro serão apresentados aspectos da implementação da solução para o problema encontrado. Os resultados obtidos com a implementação apresentada no capítulo três serão feitas no capítulo quatro. E por último as conclusões que foram obtidas a partir da realização do trabalho.

2 REVISÃO DE LITERATURA

Neste capítulo, são apresentados alguns conceitos e termos utilizados no decorrer do trabalho. Os conceitos apresentados tem relação com a construção do sistema. São apresentadas as técnicas, arquiteturas e algoritmos disponíveis para solução do problema, assim como as utilizadas no desenvolvimento da solução.

As primeiras seções do capítulo abordam os padrões e protocolos disponíveis na literatura que são utilizados na construção de *webservices*. Esses padrões e protocolos são conhecidos por facilitar e flexibilizar a integração de sistemas. Entre eles destacam-se o *Service Oriented Architecture* (SOA), que é uma arquitetura que trata os serviços como componentes e visa utilizar esses componentes para resolver problemas de negócios complexos através de composição de serviços. *Representational State Transfer* (REST), que é um estilo arquitetural que pode ser utilizado, ou não, para a criação de serviços SOA. O *Simple Object Access Protocol* (SOAP), que é um protocolo que muitas vezes é comparado ao REST por também servir para a criação de serviços SOA.

Além disso, nas seções posteriores, o capítulo também apresenta algoritmos que ajudam a resolver o problema da seleção de host para migração de VM através de otimização multi-objetivo (MOO). Existem vários algoritmos que podem ser utilizados para a resolução desse tipo de problema. Neste trabalho, são abordados os algoritmos genéticos (GAs), que fazem parte de um grupo maior de algoritmos, chamado algoritmos evolucionários (EAs). O conceito de EA também será abordado neste capítulo.

Nas primeiras quatro seções, as ferramentas apresentadas estão relacionadas em como é feita a comunicação e disponibilização do serviço. E as outras seções estão relacionados ao funcionamento interno, em como solucionar a seleção dos melhores hosts.

2.1 SERVICE ORIENTED ARCHITECTURE (SOA)

O desenvolvimento de software para um ambiente corporativo é uma tarefa complexa. Conforme Brown [Brown, Johnston e Kelly 2002], no decorrer dos anos, a comunidade de desenvolvimento de software se dedicou em desenvolver novas abordagens, processos e ferramentas para a construção de softwares de grande escala.

Brown considera que uma maneira de descrever um sistema de software é como sendo um composto de uma coleção de serviços. Cada serviço, provém um conjunto de funcionalidades bem definidas. As funcionalidades do serviço sendo bem definidas tornam possível a construção de serviços compostos, ou seja, uma funcionalidade que faça a utilização de outras funcionalidades ou serviços. Esta modularização e coordenação de serviços e funcionalidades caracteriza uma Service Oriented Architecture (SOA).

Segundo [Valipour et al. 2009], SOA pode ser definido como um design de software utilizado para conectar negócios e recursos computacionais sob demanda, e isso possibilita os usuários do serviço (podendo ser outros serviços ou usuários finais)

a alcançarem seus objetivos.

Existem diversas maneiras de implementar uma aplicação baseada em SOA, o importante é que sua interface seja bem definida com as operações que podem ser realizadas. Uma das grandes vantagens do SOA, é a facilidade que ele provém na integração de sistemas. Segundo [Valipour et al. 2009], com as operações bem definidas e disponíveis, o consumidor do SOA, pode se preocupar somente com o que determinado serviço faz e não como é implementado.

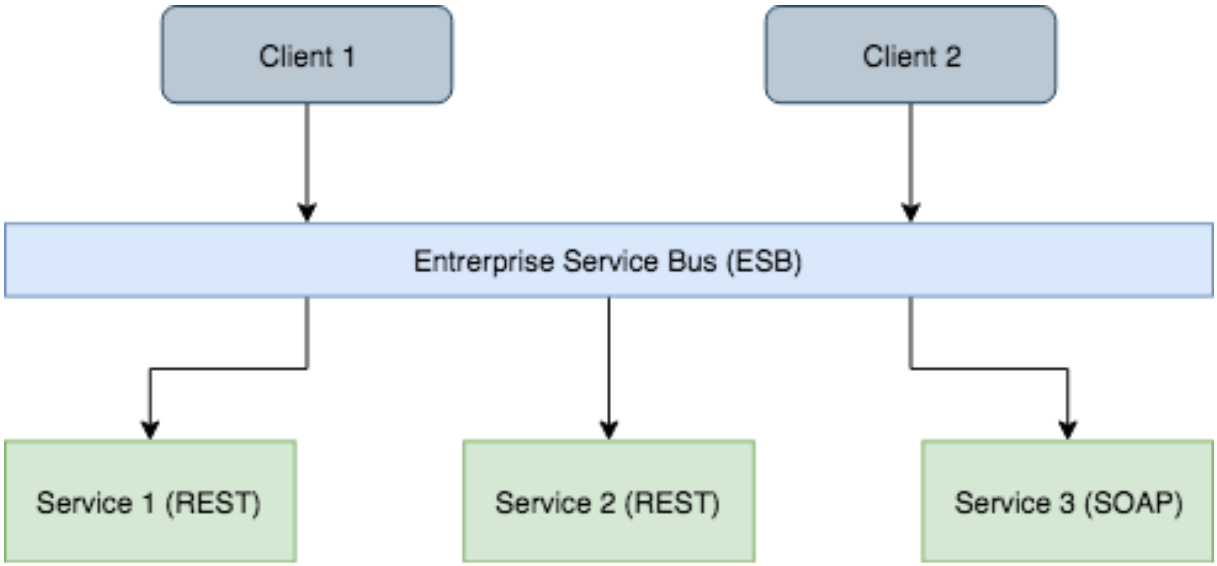
As principais características de um software feito utilizando SOA são que ele é auto contido e modular, interoperável, fracamente acoplado, passível de composição e possui transparência de localização. Como SOA não limita a estratégia utilizada para o desenvolvimento do mesmo, pode-se utilizar qualquer técnica para implementá-lo. No ambiente corporativo, os serviços comumente são implementados utilizando web services SOAP, REST ou chamadas RPCs.

2.1.1 Enterprise Service Bus (ESB)

Para obter essa flexibilidade na construção dos serviços, é comum que com SOA se utilize uma camada que recebe a chamada dos serviços e encaminha para o serviço correto, ela faz isso através de um enfileiramento de mensagens. Essa camada, além de ter responsabilidade enfileirar as mensagens enviadas aos serviços, também pode fazer a tradução da comunicação, caso um serviço trabalhe somente com SOAP e outro somente com REST.

A arquitetura utilizando SOA fica semelhante a da Figura 1.

Figura 1 – SOA com ESB



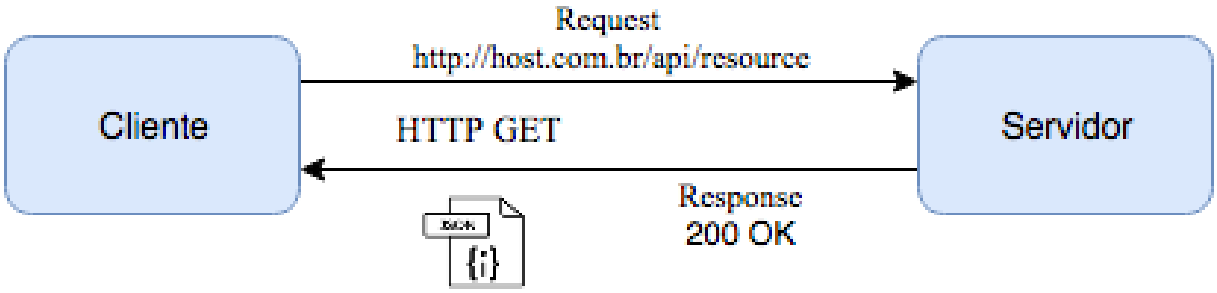
2.2 REPRESENTATIONAL STATE TRANSFER (REST)

REST foi formalizado por Fielding [Fielding e Taylor 2000] em sua tese de doutorado, onde ele tem por objetivo apresentar uma arquitetura para criação de sistemas network-based. Na tese, REST é definido como um estilo arquitetural. Fielding

também define seis diretivas que devem ser respeitadas na criação de um software que é implementado utilizando este estilo arquitetural. As diretivas são maneiras de implementar a API para simplificar o uso e melhorar a arquitetura da API que está sendo construída.

A comunicação no REST, é feita via *Hypertext Trasfer Protocol* (HTTP). Desta meneira a comunicação cliente-servidor é feita utilizando as funcionalidades que o protocolo suporta, como: a utilização de cabeçalhos, *query-strings*, *Universal Resource Identifiers*(URIs), entre outros recursos do protocolo.

Figura 2 – REST sobre HTTP simples



Neste capítulo são apresentadas as seis diretivas que são definidas por Fielding que um sistema deve atender quando REST é utilizado e a maneira de utilização dos verbos HTTP no padrão arquitetural.

As diretivas, podem ser atendidas em sua totalidade ou não, isso depede do nível de maturidade da API. Uma API que não atende todas as diretivas não está deixando de utilizar REST, apenas o utiliza com um nível de maturidade menor. As seis diretivas do REST são:

1. Interface uniforme
2. *Stateless*
3. Cacheável
4. Cliente-servidor
5. Sistema em camadas
6. Código por demanda (Opcional)

2.2.1 Interface Uniforme

Fielding [Fielding e Taylor 2000] destaca que uma das características centrais do REST, e o que difere ele de outros estilos arquiteturais, é a utilização de uma interface uniforme entre os componentes. Esta interface uniforme define a forma com que o cliente e o servidor se comunicam. Segundo [Fredrich 2012] isto desacopla e simplifica a arquitetura.

Segundo [Fredrich 2012] além disso, essa interface, é uniforme quando segue as seguintes características:

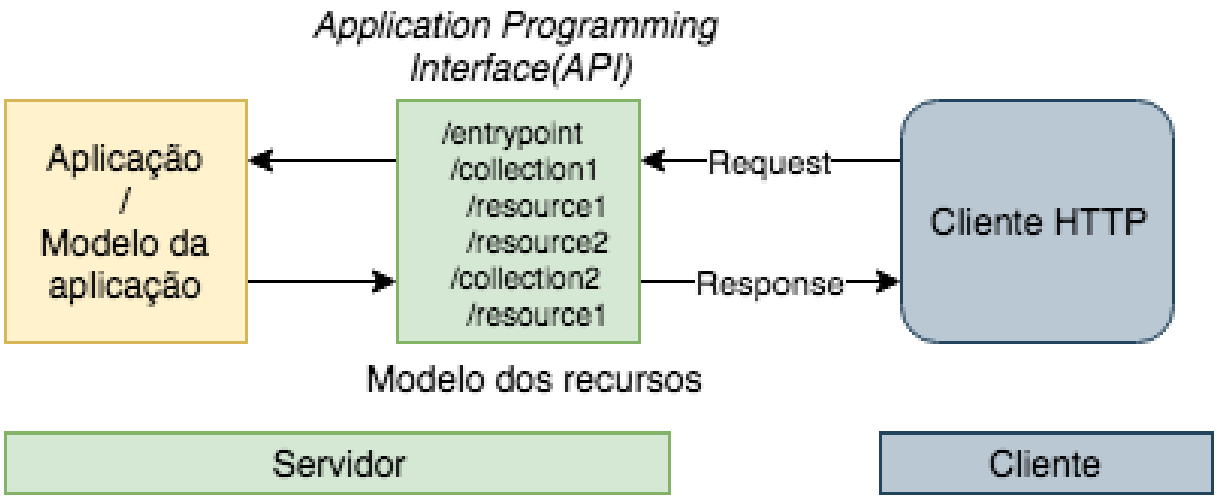
1. Baseada em recurso
2. Manipula os recursos através de representações
3. Possui mensagens autodescritivas
4. *Hypermedia as the Engine of Application State* (HATEOAS)

Ser **baseada em recursos**, significa que a interface deve separar seus recursos através de URIs. Cada recurso possui alguma URI que serve como *endpoint* para interação com o usuário. Essa URI é utilizada para fazer a **manipulação dos recursos utilizando representações**. As representações de um recursos podem ter diversos formatos, como por exemplo *JavaScript Object Notation* (JSON) ou *eXtensible Markup Language*(XML).

Mensagens autodescritivas significa que na requisição feita pelo cliente para o servidor, devem ser incluidas informações sobre o que deve ser feito com o recurso que se está utilizando. Por exemplo, deve ser incluido o formato que está sendo feita a comunicação (XML, JSON), o que está querendo modificar naquele recurso (atualizar, deletar, etc.). Além dessas informações, que são enviadas pelo cliente, as mensagens de resposta também devem ser autodescritivas, contendo informações sobre a resposta, por exemplo, se o recurso pode ser cacheado e por quanto tempo.

Segundo [Fredrich 2012] **HATEOAS** significa clientes entregar estado via conteúdo no corpo da mensagem, *query-strings*, cabeçalhos e URIs. E o servidor retornar estados através de *status-codes*, conteúdo no corpo da mensagem e meta-informações através de cabeçalhos. Seguindo essas regras, a comunicação é considerada comunicação via **hyperteto**. Além disso, também significa enviar a relação entre recursos quando necessário. Isso pode ser feito através de links contidos nos corpos da mensagem ou cabeçalhos da requisição.

Figura 3 – Recursos REST



2.2.2 Cliente-servidor

REST é implementado utilizando o modelo de comunicação cliente-servidor. Isso ajuda com a separação de responsabilidades, e permite que uma portabilidade de clientes do serviço implementado. O REST permite uma reutilização do serviço. Do ponto de vista de arquitetura de software, isto é muito importante, pois permite que componentes fiquem bem modularizados.

2.2.3 Stateless

No REST, é necessário que a aplicação seja *Stateless*, ou seja, as requisições devem ser autocontidas. Isso significa que as requisições de usuários devem conter todas as informações necesárias para o servidor entender e processar a requisição. Não deve-se assumir que o cliente da API utilize-a de uma forma específica para a requisição funcionar.

2.2.4 Cacheável

Essa é uma característica que faz com que aumente o desempenho da aplicação. Deve ser possível fazer cache de informações. Isso faz com que gere menos comunicação entre cliente e servidor. O *cache* pode ser feito de maneira implicita (o cliente define quando cachear) e de maneira explícita (o servidor indica que a requisição pode ser cacheada).

2.2.5 Sistemas em camadas

Deve ser possível adicionar camadas(*middlewares*) na utilização do REST. Essas camadas podem ter diversas finalidades, como, por exemplo, aumento de performance através de *load-balance*, aumento de segurança (autenticação), entre outros.

2.2.6 Código por demanda (Opcional)

Esta restrição é opcional e pouco utilizada. Segundo [Fredrich 2012] é uma maneira de extender a funcionalidade através de envio de código para ser executado pelo cliente. Esta restrição deve ser aplicada apenas quando não viole outras diretivas.

2.2.7 Verbos HTTP

Além dos itens citados nas seções anteriores, outra parte importante no REST, para a uniformidade da interface, são os **verbos http**. Os verbos HTTP representam a ação que o usuário da API está querendo tomar através da requisição enviada. Isso é muito importante, pois atribuí um contexto para cada tipo de requisição da API através do tipo do verbo utilizado.

Tabela 1 – Tabela de verbos HTTP

Verbo	Descrição
GET	busca/le informações na API
POST	cria recursos na API
PUT	atualiza algum recurso
DELETE	deleta algum recurso ou alguns recursos

2.3 SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

SOAP é um protocolo de comunicação baseado em XML(eXtension Markup Language) que foi criado no final dos anos 90. Seu objetivo é fazer a comunicação entre o cliente e o servidor através de informações passadas através de um documento XML. O protocolo utiliza um *schema* XML, que é uma maneira de descrever e validar o formato os dados das requisições e respostas. Esse *schema* é utilizado pelo cliente e pelo servidor para saber como interpretar a resposta, no caso de recebimento de mensagem, e formatar a requisição, no caso de envio. Esse *schema* é chamado de WSDL (Web Services Description Language).

O objetivo do SOAP, é expor regras de negócio de aplicação através de serviços. Por esse motivo, o SOAP é uma opção comumente utilizada na construção de aplicações SOA. Outra característica do SOAP, é que ele não precisa ser implementado sobre um protocolo de transporte específico, e na maioria das vezes é utilizado HTTP, porém é possível implementá-lo utilizando outros protocolos. Além do SOAP permitir utilizar outro protocolo, ele não restringe a implementação em alguma linguagem de programação específica, ou seja, é possível implementá-lo em qualquer linguagem de programação.

O SOAP é comparado ao REST, pois os dois podem ser utilizados para um objetivo semelhante, porém, os dois tem um foco diferente, onde o SOAP tem como objetivo expor regras de negócio como serviço, e o REST visa representar um determinado estado e manipulá-lo através de operações bem definidas.

Segundo [Lecheta 2015] como o SOAP é um grande XML, no contexto dos webservices, ele começou a parder espaço para o REST, o qual é mais simples e permite enviar informações em formatos mais leves, como o JSON por exemplo. Isso é uma grande vantagem, principalmente por ser um problema que tem o potencial de ter quantidade de dados trafegando em larga escala.

Uma comparação do SOAP com o rest, pode ser observado na Tabela 2

Tabela 2 – Tabela comparativa REST vs. SOAP

Comparação	
Design	Protocolo bem definido c
Formato de Mensagens	
Abordagem	Guiado por funções, mais ligadas ao negócio, por exemplo (ge
Protocolo de comunicação	Qualquer protoco
Cache	Não

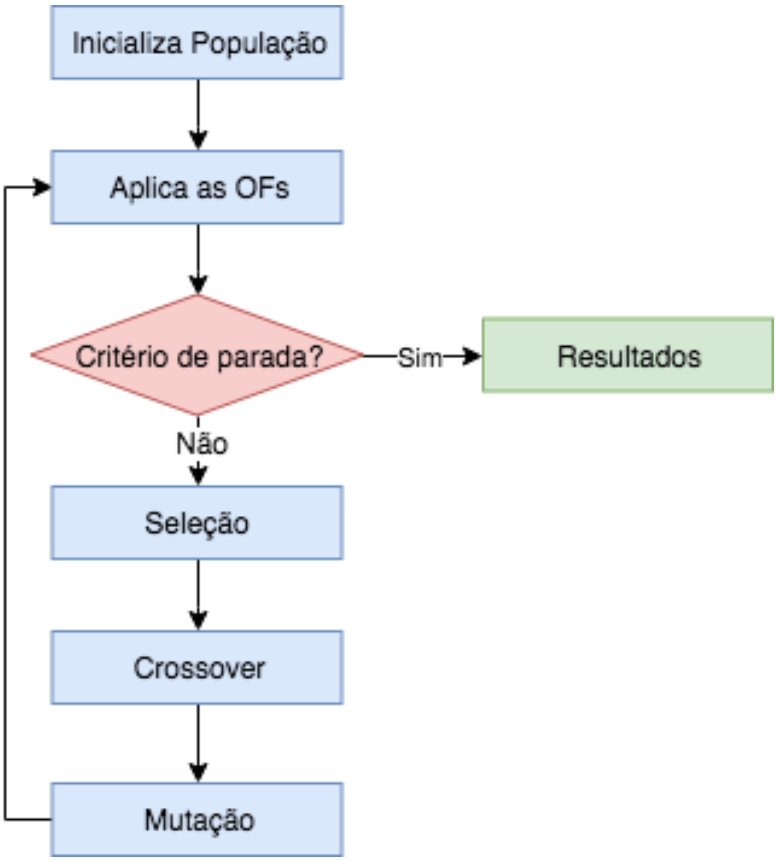
2.4 ALGORITMOS GENÉTICOS (GA)

Algoritmos genéticos (GAs) são uma classe de algoritmos que se baseam na evolução natural de Darwin. Esse processo envolve criar uma população, selecionar os indivíduos mais aptos através de uma função objetivo, aplicar mutação e *crossover* na população. Esse processo resulta em gerações (novas populações) mais adequadas em relação ao objetivo que está sendo avaliado na fase de seleção, assim como ocorre na teoria da seleção natural, os mais "adequados"sobrevivem. O processo

evolutivo (os passos anteriores para geração de novas populações) continua até que haja um critério de parada. Este processo é ilustrado no fluxograma da Figura 4.

De maneira geral, o GA utiliza a função objetivo (OF) para a seleção, ou seja, escolha dos mais adequados, utiliza o *crossover* para a convergência das soluções para melhores soluções, e a mutação como operador de diversidade. Estes operadores são importante para obter tipos diferentes de soluções, inicialmente inexplorados pelo algoritmo.

Figura 4 – Exemplo GA



2.4.1 Seleção

Conforme [Ghosh e Das 2008], o principal objetivo do operador de seleção é manter as soluções boas e eliminar as ruins mantendo o tamanho da população constante.

Existem vários tipos de operadores de seleção. Entre eles, os mais comuns são: **Tournament Selection**, **Proportionate Selection** e **Ranking Selection**. Cada tipo de operador de seleção utiliza uma estratégia diferente para fazer a seleção dos melhores indivíduos. O resultado da seleção feita por esse tipo de operador é chamada de *mating pool*, que representam os indivíduos selecionados para a reprodução.

2.4.2 *Crossover*

O operador de *crossover* é responsável pelo processo onde dois cromossomos são cruzados. Segundo [Ghosh e Das 2008], dois cromossomos são selecionados

da *mating pool* e parte deles são misturados para que seja feita a criação de novos indivíduos. Nos GAs a quantidade de indivíduos que são escolhidos para o cruzamento, dependem da probabilidade de crossover. Após o cruzamento, é gerada uma nova geração (*offspring population*). Depois da fase de *crossover*, o algoritmo segue para a mutação, conforme a Figura 4 ilustra.

2.4.3 Mutação

A mutação é uma etapa que é responsável por manter a diversidade na população. Isso é feito através da mudança do valor de genes de algum cromossomo da população. Geralmente, o operador de mutação é utilizado com uma probabilidade baixa, pois caso contrário, seriam geradas somente populações aleatórias.

A mutação do gene é importante para que as gerações não tenham uma convergência para um único ponto. Tanto o crossover quanto a mutação, fazem com que o espaço de busca de soluções ótimas, seja ampliado. Isso é importante para que nenhuma solução ótima passe despercebido. Sem este operador de diversidade, as soluções podem convergir para um ponto em que todas fiquem muito parecidas, o que não é interessante pois soluções melhores podem estar localizadas em um ponto diferente da busca.

Segundo [Ghosh e Das 2008], estes operadores genéticos do GA, podem ser usados para otimizar tanto problemas mono-objetivo e também multi-objetivo. Neste trabalho, será utilizado o algoritmo para um problema multi-objetivo, por esse motivo, a próxima seção aborda este tipo de problema.

2.5 OTIMIZAÇÃO MULTI-OBJETIVO (MOO)

Segundo [Ghosh e Das 2008] um problema de otimização multi-objetivo, como o próprio nome sugere, possui N objetivos que devem ser otimizados. Na otimização multi-objetivo, existem duas variáveis importantes para fazer a seleção dos indivíduos de uma população, que são o espaço de busca (*search space*) e o espaço dos objetivos (*objective space*).

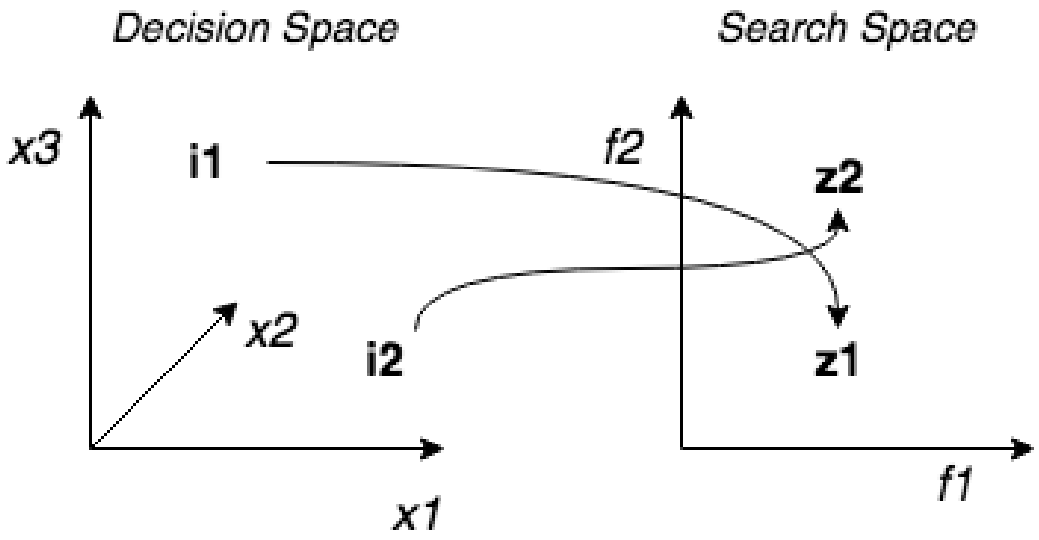
O espaço de busca é representado pelos valores das variávies dos indivíduos de uma população. Através desses valores, os indivíduos são avaliados e possuem um valor correspondente no *objective space*. Este valor é calculado através de seu *fitness*, ou seja, representa o quão apto aquele espaço de busca é em relação aos objetivos.

2.5.1 Dominância de pareto

Após o mapeamento do espaço de busca e espaço de objetivos, é necessário ter uma maneira de reconhecer os melhores indivíduos. Uma possível estratégia para o reconhecimento dos melhores no contexto de MOO é a **Dominância de Pareto**.

Segundo [Ghosh e Das 2008], problemas de MOO, as OFs são conflitantes por natureza, e por consequência, cada OF possui uma solução ótima. Para resolver este problema, as OFs não devem ser consideradas individualmente na avaliação de um indivíduo. Sendo assim, uma solução é dominada caso nenhum objetivo seja melhor do que o qual está sendo comparado. Por exemplo:

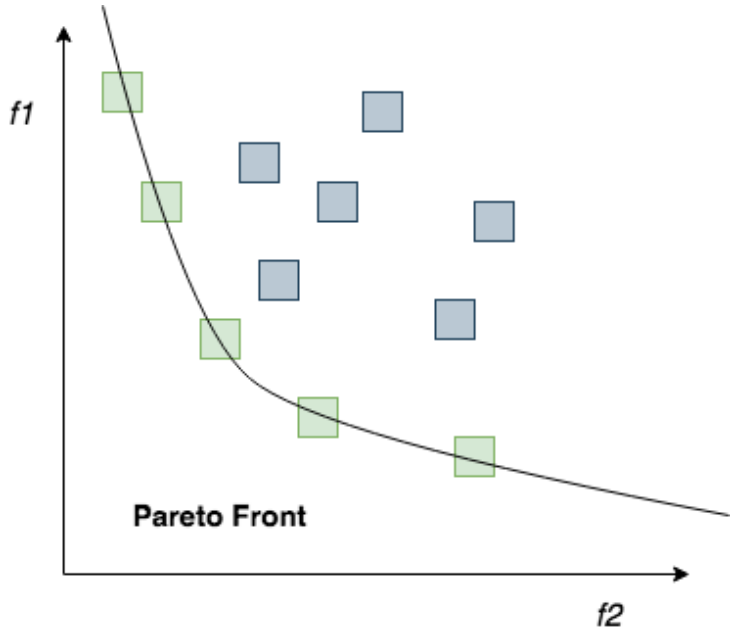
Figura 5 – Espaço de decisáo e espaço dos objetivos



1. **s1 e s2** são duas soluções;
2. **f1 e f2** são duas funções objetivo;
3. **s1** domina **s2** se:
   a) **f1 e f2** da **s1** tem melhores resultados que **s2**.
4. **s1** tenha melhores resultados na OF **f1** e **s2** na OF **f2** ambas são consideradas não dominadas.

O objetivo quando se está fazendo uma otimização multi-objetivo é buscar a chamada **fronteira de pareto** ou *pareto-front*. Onde, apenas as soluções que não são dominadas por nenhuma outra solução fazem parte, ou seja, são as melhores soluções para um problema levando em conta os múltiplos objetivos.

Figura 6 – Pareto-front



2.6 TRABALHOS RELACIONADOS

O OptVM busca oferecer um suporte para a migração de máquinas virtuais. Esse suporte é oferecido através de um serviço que seleciona um subconjunto de hosts dos possíveis alvos de migração. Além disso, oferece uma maneira de restringir a escolha dos hosts que não são elegiveis por questões relacionadas a negócio.

Esta seção apresenta alguns trabalhos relacionados ao problema que o OptVM está se propondo a resolver.

2.6.1 Migração de máquinas virtuais

As nuvens computacionais são utilizadas pela maioria das empresas de software da atualidade. Por esse motivo, as maiores empresas do setor investem muito neste segmento, oferecendo vários tipos de serviços diferenciados para seus consumidores. Estas empresas concorrem em alguns aspectos, como: velocidade, preço, disponibilidade e etc. Para aumentar sua competitividade, é utilizada a virtualização.

Com a virtualização é possível alocar partes de um recurso físico para diferentes consumidores, fazendo com que um recurso físico se torne melhor utilizado. Isso deixa a alocação de recursos muito mais flexível, e torna possível obter uma elasticidade nos serviços oferecidos.

Uma dos objetivos de utilizar a virtualização é obter uma elasticidade dos recursos oferecidos. Ou seja, é possível aumentar sua capacidade de processamento, armazenamento mesmo depois que o já foi alocada uma VM para o usuário. Isso permite que um consumidor do serviço possa escolher o quanto precisa para executar as tarefas que deseja, assim como o provedor também consegue otimizar o uso de seus recursos. Porém, existem casos onde uma máquina virtual não consegue aumentar seu consumo de recursos, por uma limitação do host em que está alocada. Nestes casos, é necessário ser feito uma realocação de VM para outro host.

A realocação de uma VM pode ser feita a nível de nuvem, datacenter(DC) ou host. Quando a realocação é feita em nível de DC ou nuvem, é muito provável que seja necessário migrar uma VM do local em que ela se encontra.

Três momentos podem ser considerados os pontos principais a serem avaliados para uma migração, são eles:

1. A descoberta de uma necessidade de migração
2. Qual máquina virtual deve ser mgirada
3. Para onde deve ocorrer a migração

As migrações das VMs são necessárias em ambientes que envolvem uma infraestruturas grande, onde existem multiplos hosts, datacenters e nuvens. Por esse motivo, não é recomendado que um sistema ou serviço gerencie a infraestrutura inteira sozinho, pois sua escalabilidade poderia se tornar um gargalo. Isso faz com que sejam construidos diferentes serviços e aplicações que se integram e gerenciam a infraesturua.

O OptVM é uma alternativa para lidar com o terceiro momento da migração. Ajuda a escolher o destino da VM através de um *webservice*. Isso remove uma das responsabilidades do serviço de gerencia da infraestrutura da nuvem, diminuindo a possibilidade de um gargalo.

3 OPTVM

O OptVM é um sistema que tem o propósito de dar suporte para a migração de VMs em um ambientes de núvens federadas, isso é feito através *webservices* utilizando o modelo cliente-servidor, mais específicamente o padrão arquitetural Representational State Transfer(REST).

O sistema possui dois principais componentes para atingir seu objetivo: um deles, faz uma filtragem de hosts aplicando restrições. E o outro faz uma otimização na escolha de um subconjunto de possíveis hosts para uma VM migrar.

As restrições que são aplicadas aos possíveis destinos (hosts), elas servem para desconsiderar os destinos que não são passiveis de migração por causa de regras de negócio. As restrições no OptVM são pré-definidas, ou seja, elas devem ser apenas escolhidas e parametrizadas pelo usuário da API, não é possível criar um tipo de restrição. Por exemplo, podem ser definidas regras do tipo: uma VM que está localizada em um país, não pode ser migrada para outro país, onde cada país da regra pode ser parametrizado. Esta regra, pode ser parametrizada com EUA e Israel, por exemplo. Além disso, as restrições são aplicadas antes da otimização, para o sistema desconsiderar hosts que não são realmente alvos de migração e limitar o espaço de busca da otimização.

O outro componente, o qual define um melhor subconjunto de hosts faz isso utilizando MOO. Essa otimização, possui um conjunto de funções objetivo pré-definidas. Esses objetivos também podem ser escolhidos e parametrizados pelo usuário da aplicação conforme sua necessidade. Os objetivos, assim como as restrições, são pré-definidos pelo OptVM e o usuário tem a opção de utilizá-los ou não. Os **objetivos da otimização**, são interesses do usuário, por exemplo, minimização do consumo de energia. No OptVM, os objetivos são traduzidos para funções objetivo, do problema de otimização. Ou seja, internamente são tratados como funções matemáticas que representam o objetivo da otimização.

Como a escolha e parametrização das restrições e funções objetivo da otimização são escolhidos pelo usuário da API, Foi criado um recurso que representa a configuração e parametrização das **restriçoes** e **objetivos de otimização**, e para este recurso foi dado o nome de **política**.

O OptVM busca ser uma solução caixa preta, onde, o usuário não necessita saber nada sobre o funcionamento interno, algoritmos utilizados, etc. Basta utilizar as *Application Programming Interfaces* (APIs) para fazer uso de suas funcionalidades. A intenção é que o usuário não precise entender sobre como as coisas são feitas internamente para obter as vantagens do uso do OptVM.

Neste capítulo, serão apresentados aspectos gerais em relação a implementação e uso do OptVM. No primeiro momento serão mostrados os componentes que integram o sistema. Depois disso, técnicas e ferramentas utilizadas, após isso, será demonstrada a ideia do funcionamento do serviço e como ele é implementado.

3.1 COMUNICAÇÃO

Em termos gerais, uma API é uma interface de software que pode ser chamada e executada [Eizinger 2017].

Como o OptVM é um serviço que deve ser disponibilizado para uma arquitetura de cliente-servidor de maneira distribuída, haviam três possíveis maneiras de implementá-lo, que eram REST, SOAP e via chamadas RPC. Para o desenvolvimento do OptVM foi escolhida a implementação utilizando o modelo REST. A escolha desta opção se deu principalmente pelos seguintes motivos:

1. É um padrão arquitetural maduro;
2. É agnóstico em relação a liguangens de programação;
3. É flexivel em relação ao modelo de comunicação.

Como o padrão arquitetural REST é agnostico em relação ao formato utilizado para fazer a comunicação dos dados. O *econding* dos dados pode ser feito da maneira que for mais conveniente para o usuário. No caso do OptVM é possível fazer a comunicação tanto no formato XML como no formato JSON.

Isso é controlado pelo próprio cliente da aplicação. É controlado através do cabeçalho *Accept* da requisição HTTP.

3.2 REPRESENTAÇÃO DO SERVIÇO

O padrão REST, definido por Fielding [Fielding e Taylor 2000], sugere que se deve criar uma interface para interação com o sistema. Essa interface é representada através de recursos, e a interação com esses recursos é feita através de requisições HTTP, as quais contém verbos, o corpo da mensagem, cabeçalhos, entre outras informações. Além disso, o padrão arquitetural também sugere que haja links para verificar outras informações e tomar ações sobre os recursos, complementando o HATEOAS.

3.2.1 Recursos

O OptVM trabalha em cima de 2 recursos, chamados políticas(*policies*) e otimizações(*optimizations*). Ambos os recursos, trabalham juntos, porém, conseguem trabalhar de maneira independente.

O recurso de políticas é responsável por gerenciar (criar, atualizar, deletar), as políticas cadastradas. As políticas são compostas por objetivos e restrições.

Já o recurso das otimizações, é responsável por gerenciar as otimizações executadas pelo sistema. Este recurso, além de fazer a otimização, também guarda um histórico, com informações mais detalhadas, que podem ser consultadas após feitas as otimizações. Além do histórico, o recurso também guarda métricas e informações sobre a otimização em si, como tempo de execução, número de hosts de entrada/saída, por exemplo.

Conforme REST propõe, são utilizados os verbos para realizar as operações correspondentes ao que eles se propõe a fazer. Então o verbo POST é utilizado para a criação de recursos, e o GET para a busca de recurso(s), DELETE para deleção e PUT para atualização.

No geral, as APIs do OptVM estão organizadas da seguinte maneira:

Tabela 3 – Tabela recurso otimização

Verbo	URI	Operação
GET	*/optimizations*	Todas as otimizações
POST	*/optimizations*	Cria uma otimização
GET	*/optimizations/:id*	Otimização específica
GET	*/optimizations/:id/details*	Detalhes da otimização
GET	*/optimizations/:id/metrics*	Métricas da otimização

Tabela 4 – Tabela recurso policy

Verbo	URI	Operação
GET	*/policies*	Todas as políticas
POST	*/policies*	Cria uma política
PUT	*/policies*	Atualiza uma política
GET	*/policies/:id*	Política específica
DELETE	*/policies/:id*	Exclui específica

Para cada API, existem campos que fazem parte do corpo da requisição e da resposta ao criar, buscar e atualizar algum recurso. Alguns campos são representados por entidades e podem ter o mesmo formato, tanto na requisição, quanto na resposta, mudando apenas a obrigatoriedade de alguns atributos da entidade. Isso acontece com o recurso de *policies* que é representado por uma entidade chamada *Policy*. Esta entidade tem o seguinte formato:

Tabela 5 – Política

Nome	Tipo	Descrição
objectives	Array	Objetivos da política
restrictions	Array	Restrições da política

Como os campos são arrays, eles possuem um conjunto de itens dentro deles. O tipo de objeto que está dentro do array não é primitivo, e também tem sua representação. A representação dos tipos de objetos, que são as restrições e os objetivos, são demonstrados pela Tabela 6 e Tabela 7 respectivamente.

Tabela 6 – Tabela de propriedades da restrição da política

Nome	Tipo	Descrição
type	Enumeration	Tipo da restrição
parameters	Object	Parâmetros da restrição

A representação da *otimizations* é maior e possui mais subobjetos aninhados do que a representação do recurso *policies*. Além disso, a otimização utiliza um formato em que a requisição e a resposta não possuem o mesmo formato. A requisição é composta pelo formato das Tabelas 8, 9, 10, 11 e 12:

Tabela 7 – Tabela das propriedades do objetivo da política

Nome	Tipo	Descrição
id	*enumeration*	Nome do objetivo
params	*object*	Parâmetros do objetivo (Opcional)

Tabela 8 – Tabela de propriedades da restrição da política

Nome	Tipo	Descrição
policyId	*int*	Identificador da política que deve ser usada
clouds	*array*	Lista de núvens da federação

Tabela 9 – Tabela de propriedades da Nuvem

Nome	Tipo	Descrição
id	*int*	Identificador da nuvem ()
datacenters	*array*	Lista de datacenters que a nuvem gerencia

Tabela 10 – Tabela de propriedades do Datacenter

Nome	Tipo	Descrição
id	*int*	Identificador do DC
name	*string*	Nome do DC
architecture	*string*	Arquitetura utilizada no DC
hypervisor	*string*	Hypervisor utilizado pelo DC
os	*string*	Sistema operacional utilizado pelo DC
localization	*string*	Localização do DC
costInfo	*object*	Informações de custo
hosts	*array*	Hosts alocados no DC

Tabela 11 – Tabela de propriedades do Host

Nome	Tipo	Descrição
id	*int*	Identificador do Host
bandwidth	*int*	Banda larga do Host
hops	*int*	Saltos de distância
requiredMemory	*int*	Memória requerida por VMs
hardware	*object*	Informações de hardware (RAM, *Storage*)
migratingVMs	*int*	VMs alocadas que estão migrando
vms	*array*	VMs alocadas no Host

A resposta, utiliza somente os identificadores das entidades envolvidas. Não é utilizado a representação completa dos hosts, sendo assim, é possível diminuir o corpo da mensagem. Porém, caso o usuário da API queira obter o corpo completo dos itens selecionados, basta que ele utilize a chamada de detalhes do recurso *optimizations*. Para a resposta o formato das tabelas **??** e **??**:

Tabela 12 – Tabela de propriedades da VM

Nome	Tipo	Descrição
id	*int*	Identificador da VM
ram	*int*	Memória RAM
storage	*int*	Armazenamento
size	*int*	Tamanho da imagem
bandwidth	*int*	Banda larga
currentAllocatedMemory	*int*	Memória RAM alocada
dirtyPages	*int*	Páginas sujas

Tabela 13 – Tabela das propriedades da resposta

Nome	Tipo	Descrição
id	*int*	Identificador da resposta
hosts	*array*	Melhores hosts
metric	*string*	Link para as métricas da otimização
details	*string*	Link para informações mais detalhadas da otimização

Tabela 14 – Tabela das propriedades da resposta

Nome	Tipo	Descrição
cloud	*int*	Identificador da nuvem
datacenter	*int*	Identificador do datacenter
host	*int*	Identificador do host

3.3 COMPONENTES

O OptVM é dividido em dois principais componentes: O aplicador de constraints (*constraint applyier*) e o otimizador(*optmizer*). Os dois componentes estão relacionados, porém são representados e aplicados separadamente.

3.4 APLICADOR DE CONSTRAINTS (CONSTRAINT APPLYIER)

Como citado na introdução do capítulo, o OptVM utiliza restrições pré-definidddas. Para o entendimento de como funciona as o aplicador de restrições, é importante conhecer os tipos de restrições e como elas funcionam.

3.4.1 Tipos de restrições (constraints)

As restrições pré definidas, tem um *alias* para que o usuário do serviço faça a identificação no uso delas. Os tipos de restrições disponibilizadas pelo OptVM operam em 3 níveis, são eles: **cloud**, **datacenter** (DC) e **hosts**.

Cada restrição tem uma responsabilidade específica, uma regra de negócio que não deve ser infringida. As restrições disponibilizadas e seus respectivos níveis são as seguintes:

1. Cloud:

a) *CloudName*: Nome de uma nuvem em que o host não pode ser alocado.

2. DC:
   a) *Conflito*: Pode conflitar uma localização, por exemplo, uma VM dos USA, não pode habitar em um host de ISRAEL;
   b) *Custo*: Os custos que irão gerar ultrapassam os parâmetros.
3. Host
   a) *Dependência*: Depende que o host tenha um sistema operacional(OS) ou hypervisor específico (baseado em seus parâmetros);
   b) *Coabitação*: Dependendo dos parâmetros da constraints, exige que o host seja vizinho (1 hop de distância).

3.4.2 Algoritmo

O *constraint applyier* tem a responsabilidade de remover os hosts que não atendem à uma ou mais restrições estabelecidas pelo usuário. Como as restrições operam nos 3 níveis, é importante que algoritmo tenha acesso a um contexto onde obtenha informações sobre as núvens, datacenters e hosts disponíveis.

O algoritmo que aplica as constraints, consiste em iterar sobre as constraints ordenadas da mais significativa (operam em nível de cloud), para a menos significativa (operam em nível de host), e remover os itens que não atendem as restrições, após cada iteração.

O pseudocódigo do algoritmo é representado da seguinte maneira:

Algoritmo 1: Constraint Applyier

Input: as $constraints$ restrições, $context(Clouds, DCs, Hosts)$ contexto com todas as opções disponíveis
Output: $context'(Clouds, DCs, Hosts)$ somente com Clouds/DCs/Hosts que atendem as restrições

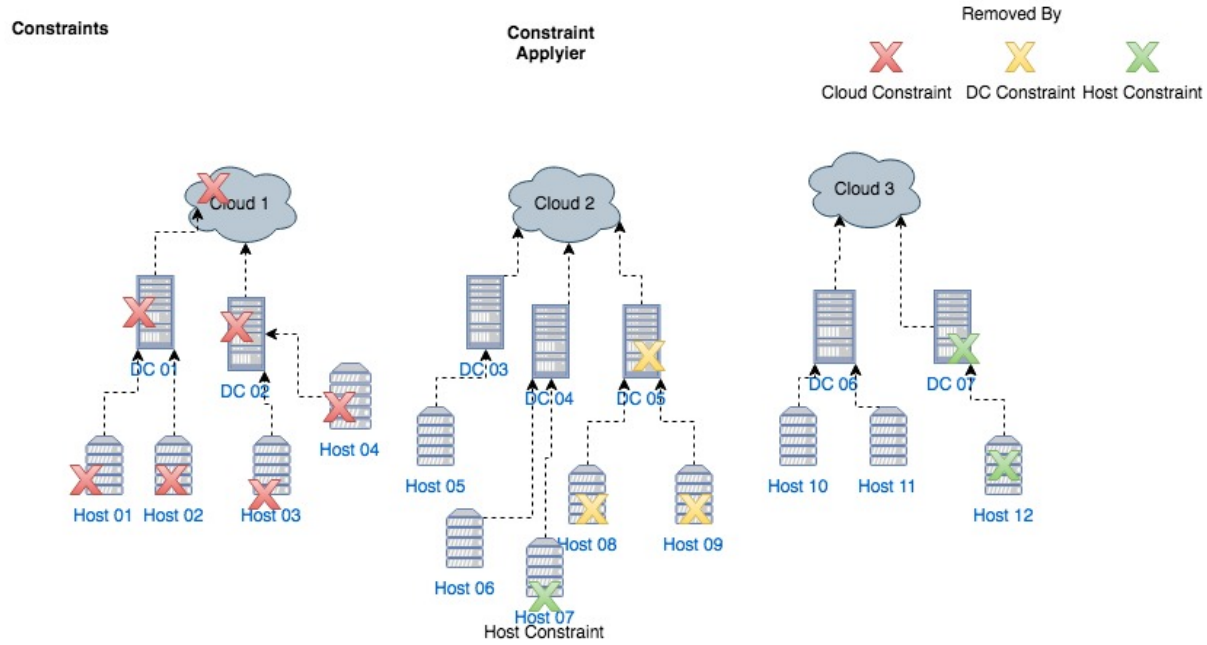
```
context' ← context
for constraints ∈ constraints do
    if c é do tipo CLOUD then
        aplica a constraints nas núvens do contexto
        context' ← nuvensAtualizadas
    end
    else if constraints é do tipo DC then
        aplica a constraints nas DCs do contexto
        context' ← datacentersAtualizados
    end
    else if constraints é do tipo HOST then
        aplica a constraints nas Hosts do contexto
        context' ← hostsAtualizados
    end
end
```

A intenção de aplicar as restrições ordenadas das mais significativas, que são as entidades que eliminarão mais destinos de uma vez (*Clouds* e *Datacenters*), para as menos significativas (*Hosts*), para que diminua o número de iterações. Pois quando, por exemplo, um *datacenter* não atende uma restrição, nenhum host daquele

datacenter são elegíveis como um possível destino, então, as restrições em nível de host para os hosts desse determinado datacenter nem devem ser executadas.

Uma representação visual, de como o algoritmo funcionaria, pode ser vista na Figura 7. A ideia é que funcione semelhante a uma estrutura de árvore com três níveis, e que quando um nó pai não atende a uma restrição, consequentemente a sub-árvore também não atende. E a avaliação é feita top-down, começando pela raiz e indo em direção aos nós folhas.

Figura 7 – Exemplo de aplicação das constraints



A Figura 7 mostra como as constraints aplicadas em seus respectivos níveis, eliminariam a possíbilidade da VM migrar para determinado host. Na Figura 7, foram representados os seguintes casos:

1. A **Cloud 1** não atende à alguma restrição escolhida em nível de nuvem, então nenhum de seus DCs e por consequência hosts dos DCs, serão alvos válidos;
2. O **Datacenter 05** não atende alguma restrição em nível de DC, então nenhum de seus hosts estarão disponíveis como alvo de migração;
3. E os **Host 07 e Host 12** não atendem alguma restrição em nível de host;
4. Um caso mais específico é quando o **Host 12** não atende a alguma restrição, logo o **DC 07** não terá nenhum host válido, então também deve ser desconsiderado.

3.5 OTIMIZADOR (OPTIMIZER)

Considerando que o cliente da API é uma federação de nuvens computacionais. O *optimizer* é responsável por otimizar a seleção dos hosts disponíveis nas nuvens da federação, para a VM que necessita migrar. O objetivo é alcançar um melhor subconjunto para a alocação da VM. Além disso, a seleção desse subconjunto é selecionado atendendo os objetivos e restrições da **política** selecionada para a otimização, escolhida pelo usuário.

A otimização feita no OptVM faz uso de GAs. Os GAs fazem iterações sobre uma população que é iniciada aleatoriamente no algoritmo. Para cada iteração, o algoritmo avalia os indivíduos ou soluções da população pela *fitness function*, que também é chamada de função objetivo. Após a avaliação dos indivíduos, é feita a seleção dos melhores avaliados e também é feito o crossover, gerando uma nova população, que é mais evoluida e mais adequada ao problema. O número de iterações feitas é escolhido pelo usuário do algoritmo. Quanto mais iterações houver, mais evoluida estará a população, porém, maior será o consumo computacional. Caso o número de iterações for muito alto, pode inviabilizar o uso do algoritmo por causa do tempo de resposta.

No OptVM, os objetivos são dinâmicos, ou seja, é possível que o usuário da API escolha por alguns objetivos e não por outros. Os objetivos disponíveis no OptVM são três, dos quais uma combinação de dois ou os três podem ser escolhidos, são eles:

1. Minimização do consumo de energia;
2. Minimização do tempo de instalação;
3. Minimização da sobrecarga da migração.

Como a otimização possui um conjunto de indivíduos, que formam uma população, cada indivíduo da população representa uma possível solução para o problema. Nos algoritmos de otimização o indivíduo possui uma representação (*encoding*), que pode ser feita de diversas maneiras, de maneira binária, por inteiros, entre outros tipos de representação.

Como uma representação representa uma solução para o problema, no OptVM, a representação escolhida foi uma representação de inteiros, ou seja, uma solução para o problema é um array de inteiros com tamanho do subconjunto escolhido pelo usuário onde cada número representa um host. Os números contidos no subconjunto representam o indice de um host de todos os disponíveis.

$$Solution = \left[\begin{array}{ccc} 36 & 5 & 120 \end{array} \right]$$

No exemplo, o tamanho escolhido para o subconjunto é de 3 hosts, ou seja, a otimização busca achar o melhor subconjunto de 3 hosts dentre os N disponíveis. No exemplo, os 3 hosts escolhidos na solução, são os hosts 36, 5 e 120 do conjunto de N hosts.

3.6 FUNÇÕES OBJETIVO (OFs)

Como citado na seção anterior, são disponibilizadas três OFs e que internamente, o OptVM utiliza funções matemáticas para a representação das OFs. Nesta seção será demonstradas e explicadas as funções utilizadas pelo OptVM.

As funções objetivo, assim como as restrições, utilizam a representação das entidades (host, vm) e seus atributos para fazer o processamento necessário na etapa em que é responsável.

3.6.1 Minimização do consumo de energia

O consumo de energia é um aspecto importante a ser minimizado quando se trata da migração de uma VM. No contexto da otimização da seleção de hosts para

fazer a migração de uma VM, o host a ser buscado para receber a VM, deve ser o que impacte menos em relação ao consumo de energia no processo de migração da VM.

Segundo [Beloglazov et al. 2011], o consumo de energia dos *hosts* está relacionado ao uso de recursos de CPU. Apesar de estar relacionado, outros aspectos devem ser levados em conta, como a capacidade do CPU. A mesma utilização de CPU em processadores de capacidades diferentes, implica que o processador de menos capacidade gaste mais energia que o de maior capacidade.

Conforme [Beloglazov et al. 2011] existe uma relação entre o consumo de CPU e o gasto de energia. Sendo assim, uma importante variável para esta OF é a utilização de CPU. O valor atribuído à utilização de CPU na OF é uma percentagem, e é representada por U_{cpu_i} na Equação 1.

Além disso, duas outras variáveis são utilizadas para calcular a energia gasta pelo *host*, que são: O máximo de energia suportado pelo host e o mínimo (quando ele se encontra em estado inativo). Estas outras duas medidas são representadas por $host_i$ (p_{max_i}) e $host_i$ (p_{min_i}) respectivamente.

Com isso, podemos definir que a Equação que representa o consumo de energia é a seguinte:

$$energ(h_i) = ((p_{\text{max}_i} - p_{\text{min}_i}) * U_{\text{cpu}_i} + p_{\text{min}_i}) \tag{1}$$

3.6.2 Minimização do tempo de instalação

Para a migração de uma VM occorer, algumas etapas devem ser concluídas, são elas: a detecção de uma migração, a transferência da VM para o *host* de destino e a instalação no mesmo. Esta OF está interessada em escolher um *host* que sobrecarregue o mínimo possível a última etapa da migração.

cap:

$$cap_{\text{h}i} = \sum_{k=1}^{Npes} (pe_k * mips_k) \tag{2}$$

$$t_{\text{fix(vm}_i\text{)}} = \sum_{j=1, i \neq j}^{N} M_j / larg_{\text{ban}}(h_{\text{i}}) \tag{3}$$

$$t_{\text{reconf(vm}_i\text{)}} = \frac{(v_{\text{dr}}(vm_i) * iter) + M_i}{larg_{\text{ban}}} + t_{\text{fix(vm}_i\text{)}} \quad 0 < i < N, N <= Cap_{\text{hosts}} \tag{4}$$

3.6.3 Minimização da sobrecarga da migração

$$sob_{\text{mig}}(h_i) = \sum_{i=1}^{m} r_i * w_i \tag{5}$$

$$sob_{\text{mig}} = \sum_{i=1}^{N} sob_{\text{mig}}(h_i) * P \quad N <= Cap_{\text{hosts}} \tag{6}$$

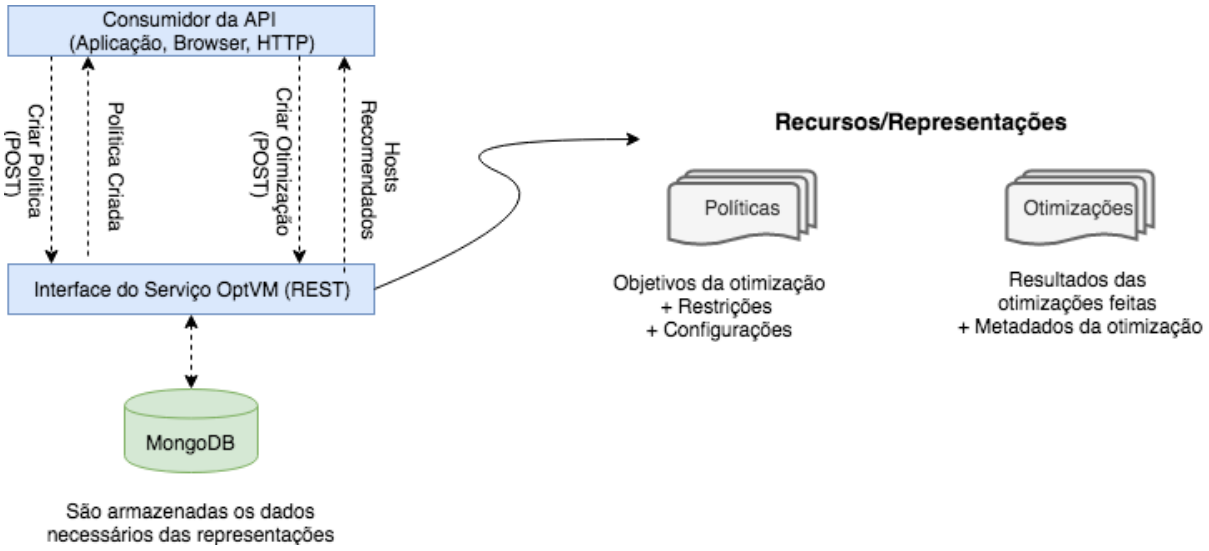
3.7 FUNCIONAMENTO DO SERVIÇO

A utilização do serviço, sugere que seja seguido um fluxo. Não é obrigatório utilizar este fluxo. Porém o OptVM é utilizado em sua totalidade se o seguir. O próprio serviço ajuda o usuário seguir o fluxo, através do HATEOAS, indicando links e próximas ações e consultas que podem ser feitas pelo consumidor da aplicação.

A ideia é que no caso de uso mais simples de utilização do serviço, para fazer uma otimização, é seguido o seguinte fluxo:

1. Criação de uma política, com seus objetivos e restrições de negócios;
2. Envio do conjunto de núvens/DCs/Hosts para ser feita a otimização;
3. Obtenção do subconjunto de hosts otimização.

Figura 8 – Exemplo de aplicação das constraints



Apesar do caso de uso básico, é possível utilizar a API de maneiras diferentes, e fazer as combinações que forem mais úteis para o usuário. Por exemplo, é possível criar uma otimização sem uma política, para este caso, serão adotados valores padrões para os parâmetros obrigatórios da política e todos os objetivos na otimização.

A criação de uma política, é feita enviando os objetivos e as restrições. Após criada a política, a mesma pode ser utilizada em uma otimização. Para a criação da política, pode ser utilizado o seguinte formato:

```
{
  "objectives": [{"id": "MIN_ENERGY_CONSUMPTION"}, {"id": "
     MIN_INSTALLATION_TIME"}],
  "constraints": [{
    "type": "Cost",
    "params": {
      "max_per_memory": 0.35
    }
  }]
}
```

Após a criação de um política, é possível usá-la para fazer otimizações. A mesma política, pode ser utilizada por múltiplas otimizações.

A criação de uma otimização, retorna um objeto com um resumo dos melhores hosts e sua identificação, para melhorar o destino da VM. Além disso, é possível obter detalhes de execução, através de */metrics*, assim como detalhes da otimização através da URI */details*.

No exemplo, a criação de um recurso de otimização deve-se ser utilizado o seguinte formato, para o corpo da requisição. Está sendo utilizado o formato JSON, porém, o mesmo se aplica também para o XML:

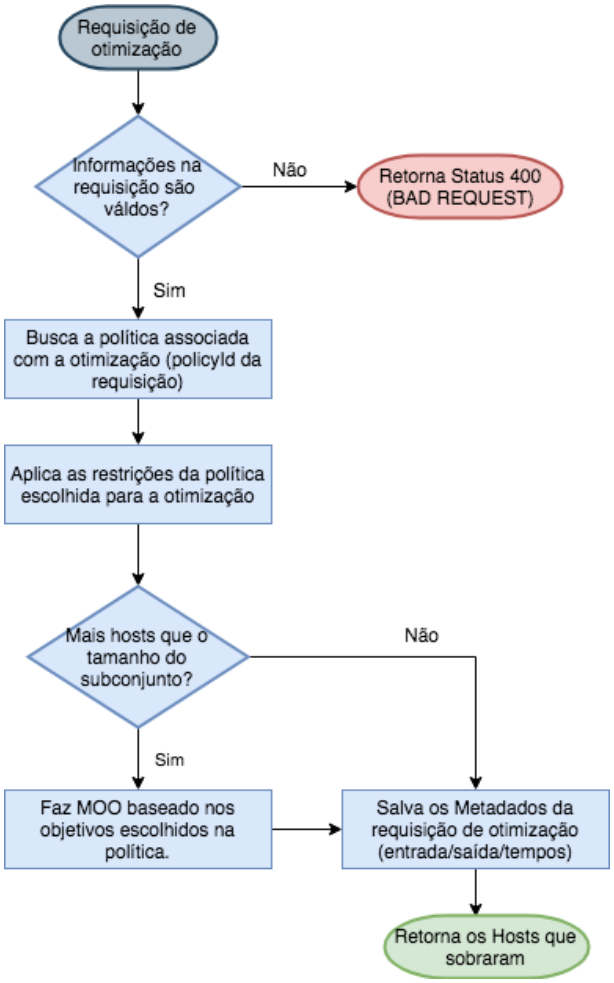
```
{
  "id": 1,
  "policyId": 2,
  "clouds": [
    {
      "id": 1,
      "name": "Test",
      "datacenters": [
        {
          "id": 1,
          "hosts": [
            {
              "id": 1,
              "vms": [{"id": 10}]
            },
            ...
          ]
        },
        ...
      ]
      ...
    }
  ]
}
```

Quando o OptVM recebe uma requisição de otimização, o seguinte fluxo é seguido para processar a requisição e retornar as melhores opções de seleção de host para o usuário:

E a resposta tem o seguinte formato:

```
{
  "id": 1,
  "policyId": 2,
  "hosts": [
    {
      "cloud": 1,
      "datacenter": 2,
      "host": 1
```

Figura 9 – Exemplo de aplicação das constraints




```
    },
    ...
  ],
  "policy": "http://host.domain/api/policies/2",
  "details": "http://host.domain/optimizations/1/details",
  "metrics": "http://host.domain/optimizations/1/metrics"
}
```

Apesar do exemplo estar mostrando arrays com apenas um item, tanto para a requisição como para a resposta, muito provavelmente haverá outros itens.

3.8 TECNOLOGIAS UTILIZADAS

O desenvolvimento de uma API pode envolver ferramentas como bibliotecas, frameworks e bancos de dados. Nesta seção, são apresentadas as principais ferramentas que foram utilizadas para fazer o desenvolvimento da API.

3.8.1 MOEA Framework

Segundo [Hadka 2014], o *Moea Framework* é um framework é uma biblioteca gratuita que serve para experiementar e desenvolver algoritmos evolucionários baseados em múltiplos objetivos. Nesta biblioteca existem diversos algoritmos, incluindo vários tipos de GAs, que foram utilizados no desenvolvimento deste trabalho.

3.8.2 MongoDB

MongoDB é um banco de dados orientado a documentos. A escolha dele foi feita por dois principais motivos:

1. Facilidade no desenvolvimento;
2. Facilidade de armazenamento de dados não estruturados.

O mongoDB faz o armazenamento dos documentos, que, basicamente são um JSON armazenado de maneira binária e eficaz. O banco separa esses JSONs em coleções, que podem ser separadas da maneira que o usuário achar melhor.

Como o MongoDB é orientado a documentos, é possível utilizá-lo para armazenar dados que não necessáriamente sigam a mesma estrutura sempre. Isso é importante no contexto do OptVM por causa da flexibilidade que o próprio OptVM permite. Por exemplo, no caso das restrições, o OptVM permite que sejam criadas restrições de diversos tipos, tendo parâmetros também de tipos diferentes, sendo assim, utilizar um banco de dados não estruturado, é uma alternativa melhor, pois permite flexibilidade no armazenamento dos itens.

4 RESULTADOS

Este capítulo é dedicado aos resultados obtidos com testes feitos utilizando a solução desenvolvida no trabalho. Os testes utilizaram um ambiente de VM para que os testes possam ser reproduzidos mais fácilmente. Foram utilizadas diferentes métricas para avaliar o comportamento da API. Além das diferentes métricas, os dados inseridos também possuem diferentes formatos e valores. Isso foi feito para que seja percebido o comportamento da API em diferentes cenários, assim como avaliar se o comportamento é alterado drásticamente ou mantem seu comportamento mesmo em casos extremos.

É importante definir bem como a API se comporta em cenários diferentes, pois, dependendo aonde a mesma estiver instalada, cada requisição gera um custo e este custo deve ser conhecido. Por exemplo, se a API estiver hospedada em nuvem, cada requisição tem um custo, seja ele por tráfego de rede, armazenamento dos dados, processamento, etc. As métricas utilizadas buscam auxiliar na decisão de utilização da API.

Os testes realizados foram aplicados somente no recurso de otimizações, que é a seleção de *hosts*. Apesar da API ter outros recursos, como o de políticas, este recurso é apenas um suporte para flexibilizar o uso das migrações. O problema que o trabalho se propõe a resolver envolve somente a etapa de criação de uma otimização. Sendo assim, este é o cenário que é avaliado.

4.1 AMBIENTE

O ambiente utilizado foi uma máquina virtual utilizando **Ubuntu Server 18.04** com memória de **4.0GB** utilizando um processador **2,3 GHz Intel Core i5**. As requisições são feitas através de uma rede *bridge* da máquina física para a virtual.

A escolha de um ambiente com VM é permitir o isolamento de recursos. Além disso, o ambiente é mantido de forma mais controlada, sendo assim, há uma melhor consistencia dos testes feitos. Além do maior controle sobre o ambiente, é importante salientar que o ambiente simulado está disponível nas nuvens comerciais na indústria, como AWS, AZURE e etc. Sendo assim, caso uma máquina com as características da VM testada for utilizada, o desempenho da API tende a ser o mesmo.

4.2 DADOS UTILIZADOS

A API do OptVM permite que seja feitas várias combinações de dados. As combinações podem variar em relação à restrições utilizadas, objetivos da otimização, quantidade de hosts, nuvens e datacenters envolvidos, entre outros.

Na avaliação dos resultados, foram utilizados dados que simulam casos simples (utilizando poucos hosts, restrições e objetivos) e casos mais complexos (utilizando mais hosts, restrições e objetivos). O formato dos cenários utilizados é demonstrado na Tabela 15

Os dados utilizados foram gerados através de um código java. O código construído gerou em todos os casos 5 nuvens, sendo que, a quantidade de datacen-

Tabela 15 – Tabela de formato dos dados de teste

Ambiente	Restrições	Objetivos	Hosts
Super Simples	0	2	100
Simples	1	2	500
Médio	2	3	1000
Difícil	3	3	2500

ters/hosts varia para ficar compatível com cada cenário. Para cada host, atribuiu-se 5 VMs.

Para os testes conter os dados mais próximos da realidade quanto possível, foram pré-definidos alguns valores para atributos que são fixos, por exemplo, sistema operacional, memória RAM total, armazenamento total. Nestes casos, um dos valores pré-definidos foi selecionado aleatoriamente como valor para o atributo da entidade. Nos casos onde o atributo é completamente variável, foi atribuido valores aleatórios para os mesmos mantendo um intervalo condizente com a realidade, por exemplo, preço por armazenamento entre 0.1 e 1 (centavos).

Como o OptVM permite escolher o tamanho do subconjunto de *hosts* que quer obter, além dos dados da nuvem, DCs e hosts, foi utilizado como padrão a busca de um subconjunto de 3 hosts como sendo uma possível solução.

4.3 MÉTRICAS UTILIZADAS

É muito comum uma API ser medida através de uma medida quantitativa de requisições por segundo (RPS) que ela consegue atender. O número de requisições varia de aplicação para aplicação, e isso depende da tecnologia utilizada, arquitetura do sistema, se há consulta em banco de dados ou não, algoritmos utilizado, etc. Esta métrica define se a utilização de um serviço é viável em relação a quantidade de consumidores ou quantidade de utilização, ou seja, se atende as necessidades de uso do consumidor do serviço. Através desta métrica é possível identificar gargalos na utilização.

Além da métrica de requisições por segundo, outro aspecto importante na utilização de API é o tamanho do *payload*. O payload, muitas vezes é atribuido um custo por tráfego da máquina, e por processamento e armazenamento, sendo assim, quanto maior o payload, mais tráfego, processamento e armazenamento ele gerará, consequentemente, aumentará o custo. Esta métrica pode identificar possíveis erros em relação ao modelamento da API, ou seja, a API modelada de uma maneira mais otimizada, utilizando os dados de maneira diferente, pode obter um *payload* menor.

4.3.1 WRK

O WRK é uma ferramenta construída em C utilizada para fazer *benchmarking* de aplicações HTTP. Com ela é possível fazer requisições concorrentes utilizando múltiplos ou um único core. Além disso, o WRK permite utilizar uma duração do teste, threads e conexões HTTP.

O resultado de um teste utilizando o WRK consegue nos trazer algumas

informações sobre a API, como RPS, transferência por segundo, entre outras. Das informações que o WRK nos traz, as escolhidas e o seu respectivo motivo foram:

1. RPS: Medir a velocidade do serviço;
2. Respostas com status de insucesso: Medir a confiabilidade.

Para o teste, foram utilizados os seguintes valores para os parâmetros do WRK:

1. Conexões: 25
2. Threads: 1
3. Tempo de teste: 3 minutos

Os parâmetros escolhidos para a realização dos testes, significam que uma thread está disparando requisições através de 25 conexões para o serviço ou seja, o serviço recebe as requisições através de 25 conexões concorrentes. Já o tempo escolhido, foi de três minutos, para que não haja um viés no número de requisições por conta de alguma lentidão inicial, por exemplo, pela abertura de conexão com banco de dados.

Além das métricas do WRK e o tamanho do payload, o OptVM salva o tempo de execução de cada etapa da criação de uma otimização, que são, a aplicação das constraints e MOO. Por ser a fase mais significativa, na avaliação dos resultados quantitativos, foi considerada a média de tempo de execução da MOO. A esta métrica foi dado o nome de *optimization execution time*(OET). Esta métrica permite identificarmos o impacto da fase de MOO na requisição.

4.4 COMPARAÇÕES

Nesta seção são apresentados os comparativos entre os cenários testados, utilizando os dados, métricas e ambiente descritos nas seções anteriores.

Para as métricas, os seguintes resultados foram obtidos:

Tabela 16 – Tabela de tamanho do Payload

Cenário	Tamanho Payload(KB)	RPS	Erros	OET
Super Simples	85	11,25	0	
Simples	419	8,11	0	
Médio	836	5,95	23	
Difícil	2000	4,17	12	

A partir destes resultados algumas conclusões podem ser tiradas:

1. Quando a requisição fica maior, eventuais erros começam a ocorrer;
2. A quantidade de RPS diminui significativamente por causa do tamanho do *payload*;
3. A fase de MOO não é afetada drasticamente, apesar do significativo aumento de *hosts*.

Quando comparado ao número de requisições de outros *webservices*, a quantidade de RPS é relativamente baixa. Porém, para o propósito deste *webservice* a quantidade pode ser suficiente, pois, como a migração é um evento custoso computacionalmente, o ideal é evitá-la.

Um dos diferenciais do OptVM é a flexibilidade que ele dá ao usuário da API para configurar a otimização da maneira que melhor se adequar ao seu problema, através das políticas e de parâmetros. Tanto um usuário do *webservice* que gerencie um ambiente complexo, quanto um que gerencie um ambiente simples, podem obter vantagens o utilizando.

5 CONCLUSÃO

A otimização da seleção de hosts na migração de máquinas virtuais foi o objetivo da solução deste trabalho. Com o crescimento da utilização de computação em nuvem, esse é um tipo de problema em que a indústria começa a se deparar. A otimização na utilização de recursos computacionais tem uma importância por diminuir o custo para empresas, melhorar o consumo de energia, entre outros fatores que fazem com que o trabalho seja útil.

Neste trabalho a solução foi desenvolvida através de um serviço utilizando REST como estilo arquitetural e algoritmos genéticos para a otimização. O trabalho fornece uma solução genérica, onde, outros trabalhos que envolvam migração de máquinas virtuais e que envolvam uma ou mais núvens, possam fazer uso da API construída.

@TODO Apresentar resultados @TODO Falar sobre resultados

5.1 TRABALHOS FUTUROS

Este trabalho permitiu ampliar o campo de visão de soluções para o problema de otimização de migração de VMs, especialmente na seleção de hosts. É possível fazer melhores versões ou evoluir a partir deste trabalho.

5.1.1 Pesquisas em restrições para o ambiente de núvens federadas

Neste trabalho, foram apresentadas restrições em nível de *cloud*, *datacenter* e *host*. Futuramente, podem ser exploradas alternativas para esses níveis. Além disso, também é possível fazer uma busca de outras restrições, que façam parte de algum destes três grupos.

5.1.2 Comparação de algoritmos MOO para migração de máquinas virtuais

Algoritmos Genéticos são utilizados neste trabalho para fazer a parte de MOO. Porém, é possível utilizar vários outros tipos de algoritmos como *Particle Swarm* (PS), *Variable Neghborhood Search* (VNS) e *Ant Colony Optimization* (ACO). Diferentes algoritmos podem ser utilizados para efeito de comparação para este problema.

5.1.3 Generalização do problema

O problema abordado neste trabalho é bastante específico e pode ser generalizado. Por exemplo, o trabalho não leva em consideração o "efeito colateral"em razão da migração da VM.

O serviço implementado no trabalho busca responder a pergunta: **"Qual é o melhor subconjunto de hosts para esta VM migrar, respeitando as restrições e objetivos da política?"**. A partir desse questionamento, uma possível pergunta para o problema generalizado, levando em considerações os efeitos colaterais, seria: **"Qual é a melhor disposição que a federação pode ter considerando determinado cenário?"**.

5.2 CONSIDERAÇÕES FINAIS

@TODO

Referências

BELOGLAZOV, A. et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. In: **Advances in computers**. [S.l.]: Elsevier, 2011. v. 82, p. 47–111. Citado na página 21.

BROWN, A.; JOHNSTON, S.; KELLY, K. Using service-oriented architecture and component-based development to build web service applications. **Rational Software Corporation**, v. 6, p. 1–16, 2002. Citado na página 3.

EIZINGER, T. **API Design in Distributed Systems: A Comparison between GraphQL and REST**. [S.l.]: Wien, 2017. Citado na página 14.

FIELDING, R. T.; TAYLOR, R. N. **Architectural styles and the design of network-based software architectures**. [S.l.]: University of California, Irvine Doctoral dissertation, 2000. v. 7. Citado 3 vezes nas páginas 4, 5 e 14.

FREDRICH, T. Restful service best practices. **Recommendations for Creating Web Services**, p. 1–34, 2012. Citado 3 vezes nas páginas 5, 6 e 7.

GHOSH, A.; DAS, M. K. Non-dominated rank based sorting genetic algorithms. **Fundamenta Informaticae**, IOS Press, v. 83, n. 3, p. 231–252, 2008. Citado 2 vezes nas páginas 9 e 10.

HADKA, D. Moea framework user guide. 2014. Citado na página 25.

LECHETA, R. R. **Web Services RESTful: Aprenda a criar web services RESTful em Java na nuvem do Google**. [S.l.]: Novatec Editora, 2015. Citado na página 8.

VALIPOUR, M. H. et al. A brief survey of software architecture concepts and service oriented architecture. In: IEEE. **2009 2nd IEEE International Conference on Computer Science and Information Technology**. [S.l.], 2009. p. 34–38. Citado 2 vezes nas páginas 3 e 4.