VM^2T - Um Sistema para Auxílio na Migração de VMs em Nuvens OpenStack

Felipe Aparecido dos Santos Novais[1], **Lúcio Agostinho Rocha**[1], **Fábio Luciano Verdi**[1]

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Sorocaba – SP

novais@ufscar.br, verdi@ufscar.br, larocha@ufscar.br

Abstract. *This article presents a tool whose purpose is to ease the customization of the logic for virtual machines (VMs) scheduling in OpenStack computing clouds. It is vital for flexibility in cloud computing that virtual environments can be elastic, with the capacity of migrating a virtual environment from a physical host to another. Although the platform is robust, the VM migration based on the OpenStack scheduler does not take into account the current use of hardware resources such as CPU, RAM memory, disk space and network. Due to that, a given VM can be migrated to a place which there are no enough resources to run it. We present VM^2T, a free and open source tool that is responsible for collecting the current usage of physical resources in the cloud and then provide the best physical host to receive a certain VM, bringing the possibility of creating a scheduling logic through a portable and user-friendly Web tool.*

Resumo. *Este artigo apresenta uma ferramenta com o propósito de facilitar a customização de lógicas para escalonamento de máquinas virtuais (MVs) em nuvens de computação OpenStack. É vital para a flexibilidade na computação em nuvem que ambientes virtualizados possam ser elásticos, tendo a capacidade de migrar um ambiente virtual de um* host *físico para outro. Apesar da robustez da plataforma, a migração de MVs tendo como base o escalonador do OpenStack não leva em conta o uso atual dos recursos de hardware como CPU, memória RAM, espaço em disco e rede. Devido a isso, há situações onde uma determinada MV é migrada para locais onde não há recursos adequados para sua execução. Neste artigo apresentamos o Virtual Machine Migration Tool (VM^2T), uma ferramenta gratuita e de código-fonte aberto que é responsável por coletar dados do uso atual de recursos físicos na nuvem e então oferecer o melhor* host *físico para receber uma determinada MV, dando a possibilidade para criar uma lógica de escalonamento através de uma portátil ferramenta Web amigável ao usuário.*

1. Introdução

A migração de MVs dentro de *datacenters* é uma atividade frequente e que requer atenção. Os algoritmos de escalonamento do OpenStack [Openstack 2015] não consideram o uso atual dos recursos físicos usados como: CPU, memória RAM, espaço em disco, redes e etc. Estes valores são avaliados de maneira estática observando apenas quantos núcleos de CPU ainda estão disponíveis em um *host*, a quantidade de memória RAM ainda não alocada para as MVs e a quantidade de espaço em disco que ainda pode ser usada. Porém,

é importante observarmos o quanto os valores já alocados estão sendo usados efetivamente e fisicamente. Para isso, o Projeto Ceilometer [OpenStack Wiki 2015] foi criado dentro do OpenStack a fim de que dados atualizados sobre o uso dos recursos sejam coletados.

Uma nuvem de computação tem de ser flexível para movimentação de ambientes virtuais. Se houver necessidade de algum ambiente virtualizado consumir mais recursos, tais como CPU, espaço em disco ou memória, é importante que a disponibilização destes recursos seja feita de forma rápida e eficiente. Ao otimizar migrações de MVs é necessário a escolha de uma lógica de escalonamento, que decide qual é o *host* mais elegível para que uma MV seja migrada. O OpenStack oferece algumas lógicas de escalomentos pré-definidas e a configuração dessas lógicas pode chegar a um alto nível complexidade.

No último lançamento do OpenStack (Juno), o Ceilometer passou a oferecer a capacidade de coleta dos dados dos *hosts* físicos, o que não era possível na versão anterior (Icehouse). A ferramenta apresentada neste artigo analisa os dados coletados pelo Ceilometer e então escolhe o melhor *host* físico para migração de MVs.

O VM^2T é uma ferramenta Web integrada ao OpenStack que torna acessível a elaboração de lógicas de escalonamento para migrações de MVs. São coletados dados sobre o uso de recursos da nuvem e, a partir desses dados, são eleitos os melhores *hosts* para que uma certa MV seja migrada. A precisão dos dados procura garantir que não haverá sobrecarga de recursos na nuvem. O uso da ferramenta ocorre através de uma interface Web, tornando a ferramenta portátil e podendo ser acessada de qualquer navegador e em qualquer dispositivo.

2. Trabalhos Relacionados

Na literatura, migração e escalonamento de recursos em *datacenters* são assuntos recorrentes, principalmente direcionadas à *greenit*. Há vários trabalhos que tratam de escalonamento, tais como [Lago et al. 2012] e [Cohen et al. 2014], que apresentam soluções de escalonamento visando a redução de consumo energético em *datacenters*. Em [Stage and Setzer 2009] os autores evidenciam a observação de recursos físicos para evitar sobrecarga de recursos em nuvens de computação.

Tratando especificamente do OpenStack, em [Ledyayev and Richter 2014] é descrita a configuração do escalonador para computação de alta perfomance. [Litvinski and Gherbi 2013] disseca totalmente o módulo de escalonamento do OpenStack, com observações importantes para a Seção 5.1 deste trabalho.

3. VM^2T

O *Virtual Machine Migration Tool* (VM^2T) é uma aplicação Web que oferece ao usuário a capacidade de migrar MVs de forma aprimorada considerando precisamente o uso atual de recursos na nuvem. A ferramenta permite ao usuário a escolha em tempo real de uma lógica de escalonamento que é utilizada para a eleger o melhor *host* que receberá uma determinada MV.

3.1. Funcionamento do VM^2T

O VM^2T possui uma interface Web que, inicialmente, solicita informações de usuário e senha para acesso ao sistema. A partir disso, o sistema autentica as credenciais e oferece

Figura 1. Seleção de MVs (1); Seleção de Hosts (2); Seleção de Critérios (3); Resultados da busca/Migrar (4); Notificação de migração (5).

acesso a uma tela com duas seções de monitoramento (Fig. 1, item 1 e item 2): uma para as MVs e outra para os *hosts* físicos. Dentro da aplicação, o usuário pode optar por migrar uma MV para um *host* físico de maneira manual, ou seja, sem considerar nenhum aspecto de uso dos recursos do ambiente. Porém, o propósito da ferramenta é que o usuário administrador da nuvem possa selecionar critérios que o VM^2T utilizará para indicar o melhor *host* para qual uma determinada MV deverá migrar. Avaliamos as opções de critérios com 4 algoritmos que ilustram a prova de conceito da escolha do melhor *host* físico de destino para a migração. Essa escolha é baseada em requisitos comumente necessários em ambientes de nuvem:

- **Por CPU:** melhor *host* físico em porcentagem de utilização de CPU;
- **Por memória RAM:** melhor *host* físico em porcentagem de utilização de memória RAM;
- **Por Disco:** melhor *host* físico em porcentagem de utilização de espaço em disco;
- **Por Peso:** procura pela melhor máquina por meio de pesos baseados em características como CPU, RAM e espaço em disco. O usuário informa o nível de importância de cada característica variando de 0 a 10, sendo 0 sem importância e 10 o mais importante. Por exemplo, os pesos dados poderiam ser 10 para CPU (muito importante), 5 para RAM (razoavelmente importante) e 0 para disco (sem

importância).

Observe que, conforme comentado anteriormente, os dados dos recursos acima citados são coletados sob demanda usando o Ceilometer e se referem ao estado atual dos recursos dos *hosts* físicos.

Após a escolha de um desses critérios, a ferramenta define o melhor *host* conforme o critério desejado, apresentando este resultado na tela e permitindo que o administrador migre a MV selecionada para o *host*.

Além disso, a interface Web do VM^2T exibe uma descrição precisa dos recursos utilizados na nuvem, tanto dos *hosts* físicos quanto das MVs, tornando a ferramenta adequada para o monitoramento de recursos, uma vez que o administrador possui uma visão geral dos recursos de *hardware* sendo utilizados no seu ambiente.

4. Detalhes de implementação do VM^2T

O VM^2T é totalmente baseado em tecnologias livres. A aplicação foi dividida em dois níveis: *back-end* (lado servidor) e *front-end* (lado cliente).

4.1. Back-end

- **OpenStack**: Principal tecnologia utilizada, em essência o módulo OpenStack REST *API*, que fornece uma interface de comunicação entre a ferramenta e o OpenStack *controller*. Com a REST API, coletamos dados, autenticamos credenciais e migramos MVs, através do envio de comandos em formato de mensagens HTTP ao *controller*. A versão do OpenStack utilizada durante o desenvolvimento da ferramenta é o OpenStack Juno, porém, o VM^2T se mostrou compatível com versões anteriores como a IceHouse e Havana.
- **Java:** Linguagem de programação pela qual é projetado todo o lado servidor da aplicação, com as tecnologias Java Servlet e Java Server Pages (JSP). É responsável pelo transporte de informação entre a REST API do OpenStack e o *front-end*. A partir da entrada do usuário no navegador existe um código Java que interpreta essa ação e a formata em uma mensagem HTTP que é enviada ao *controller* para que seja executado um comando de gerência da nuvem. Dentre os papéis do código Java, podemos citar a coleta de recursos em uso nos *hosts* e MVs e os algoritmos que buscam os melhores *hosts* através de um certo critério.

4.2. Front-end

- **Linguagens de Marcação:** Para editar a disposição visual da aplicação Web as linguagens HTML5 e CSS3 foram utilizadas. Elas são responsáveies por formatar e apresentar as informações recebidas do servidor ou enviar requisições para que informações sejam processadas pelo *back-end*.
- **JavaScript:** Linguagem de programação utilizada para atualizar informações dinamicamente na página Web, sem que seja necessária diversas requisições ao servidor, diminuindo o tempo de carregamento dos elementos visuais da aplicação.

4.3. Arquitetura do VM^2T

A arquitetura do VM^2T é apresentada na Fig. 2. Através da OpenStack REST API (Fig 2. item 2) a ferramenta é integrada ao OpenStack. A REST API realiza a comunicação

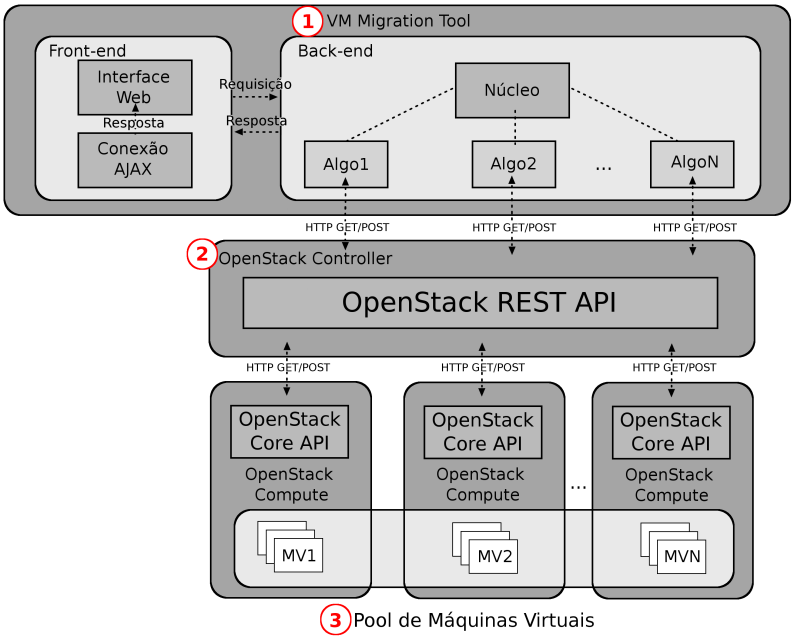

Figura 2. Arquitetura do VM^2T.

necessária para o acesso de comandos no *controller*, onde se pode requisitar dados dos recursos de *hardware* da nuvem ou a migração de MVs. Como explicado na Seção 3.1, tendo acesso ao *controller*, obtemos a gerência total da nuvem. Esse controle é intermediado através de mensagens HTTP enviadas da ferramenta (Fig 2. item 1) ao OpenStack REST API. Assim, o VM^2T pode realizar operações como o *live-migrate* parar migrar uma MV a um específico *host*, fazendo com que a ferramenta gerencie máquinas *compute* (Fig 2. item 3).

Interior à sub-arquitetura da ferramenta (Fig 2. item 1) existe uma divisão entre *front-end* e *back-end*. O *front-end* é a camada visual apresentada ao usuário, onde ele interage e **requisita** ações. O *back-end* processa as requisições do usuário e **responde** a essas ações. Por exemplo, no *front-end*/página Web o usuário seleciona uma opção para migrar uma MV. Ao receber essa solicitação, o *back-end*/servidor interpreta e, através da REST API, envia um comando para que o *controller* execute a ação de migração.

Anexo ao *back-end*, está o núcleo da aplicação, onde se localizam as principais bibliotecas e classes do VM^2T. Cada algoritmo executado no *back-end* corresponde a uma ação realizada pelo cliente tal como: atualizar recursos, escolher o melhor *host* por RAM e migrar MVs. Adequando cada ação representada por um algoritmo, diferentes mensagens são formatadas e enviadas para a REST API resultando em uma resposta XML/JSON que é interpretada e devolvida para o *front-end*.

5. Avaliações e testes

Os cenários estão apresentados nas Tabelas de 2 a 5 e testam as configurações padrões do escalonador OpenStack. A cada cenário de teste o ambiente é refabricado desde o início. Cada *host* mantém as mesmas configurações durante todos os cenários de teste. As seguintes configurações foram usadas para os *hosts* físicos (ver Tabela 1):

O seguinte roteiro foi realizado para avaliar o escalonador do OpenStack:

Tabela 1. Configurações do cenário de testes

Nome:	Host1	Host2	Host3
CPU:	2 núcleos	2 núcleos	3 núcleos
Ram:	2048 MB	2048 MB	4096 MB

1. Consideramos que cada MV consome 1 núcleo de CPU e 1024 MB de RAM;
2. Executamos um programa no Host3 para elevar o consumo de recursos do *host*, mantendo o processador no limite de uso;
3. Instanciamos 4 MVs.
4. As tabelas abaixo mostram a ordem em que as MVs foram instanciadas e em qual *host* físico. Após isso, escolhemos um MV para migrar.

Tabela 2. Cenário 1

Passo	**Descrição**
1.	Instanciada MV em Host1
2.	Instanciada MV em Host2
3.	Instanciada MV em Host2
4.	Instanciada MV em Host3
–Iniciado consumo de recursos em Host3–	
5.	VM em Host2 migrou para Host1

No Cenário 1, escolhemos uma MV do Host2 para migrar. Tal MV migrou para o Host1, o que é razoável considerando que o Host3 está com sobrecarga de uso.

Tabela 3. Cenário 2

Passo	**Descrição**
1.	Instanciada MV em Host1
2.	Instanciada MV em Host1
3.	Instanciada MV em Host2
4.	Instanciada MV em Host3
–Iniciado consumo de recursos em Host3–	
5.	VM em Host1 migrou para Host2

No Cenário 2, escolhemos um MV do Host1 para ser migrada. Esta MV migrou para o Host2, o que também resulta em um bom balanceamento de carga pois evitou o Host3.

No Cenário 3, escolhemos um MV do Host2 como candidata à migração. Neste caso, o escalonador escolheu o Host3 como destino, ignorando o fato de que tal *host* está com a CPU sobrecarregada.

No Cenário 4 ocorre exatamente a mesma situação do Cenário 3, ou seja, escolhe-se o Host3 como candidato.

Tabela 4. Cenário 3

Passo	Descrição
1.	Instanciada MV em Host2
2.	Instanciada MV em Host3
3.	Instanciada MV em Host2
4.	Instanciada MV em Host1
–Iniciado consumo de recursos em Host3–	
5.	VM em Host2 migrou para **Host3**

Tabela 5. Cenário 4

Passo	Descrição
1.	Instanciada MV em Host1
2.	Instanciada MV em Host2
3.	Instanciada MV em Host1
4.	Instanciada MV em Host3
–Iniciado consumo de recursos em Host3–	
5.	VM em Host1 migrou para **Host3**

Desta forma, notamos que em dois cenários testados o escalonador do OpenStack não considerou o uso atual dos recursos do Host3 e o escolheu como destino para receber a MV.

Realizamos um teste com a nossa ferramenta conforme descrito a seguir. Usamos as mesmas configurações dos *hosts* conforme Tabela 1. Instanciamos 4 MVs que foram alocadas conforme a Figura 3.

Na parte superior da Fig. 3 (*Virtual Machines*) estão dispostas as MVs instanciadas na nuvem: 2 MVs estão no Host1, 1 MV está no Host2 e 1 MV está no Host3. No meio da tela aparecem os *hosts* físicos do ambiente. Na parte inferior estão os critérios de seleção de *hosts* físicos. Selecionamos uma das MVs (VMa) do Host1 como candidata à migração. Neste exemplo, escolhemos o critério CPU para busca. O VM^2T elege o Host2 como o melhor levando em conta que o Host3, apesar de possuir 2 núcleos de CPU disponíveis, está sobrecarregado com nossa aplicação de consumo de CPU.

5.1. Análise e interpretação de resultados

Percebemos um comportamento aleatório do escalonador do OpenStack durante as iterações do teste. Raramente o arranjo de instâncias ocorre da mesma forma como também observado por [Litvinski and Gherbi 2013]. O uso da CPU no Host3 (obtido através do comando Unix top) era de 96% para os 4 cenários. Nos Cenários 1 e 2, obtemos um balanceamento que pode ser considerado adequado. Porém, ao observarmos os Cenários 3 e 4, notamos que a alocação de MVs não considerou o uso efetivo da CPU do Host3. Apesar do Host3 possuir mais núcleos disponíveis, o uso da CPU do *host* físico está alto e, portanto, alocar novas MVs em tal *host* não é adequado.

Figura 3. Teste com VM^2T.

6. Descrição da demonstração

Devido à dificuldade para executar uma nuvem OpenStack fora do ambiente de servidores, um vídeo explicativo sobre o VM^2T será apresentado no Salão de Ferramentas 2015.

6.1. Documentação e download

A aplicação VM^2T, seu manual de uso e manual de instalação estão disponíveis para download em: `https://github.com/FelipeNovais/VM-T`.

Referências

Cohen, J., Cordeiro, D., and Raphael, P. L. F. (2014). Energy-aware multi-organization scheduling problem. Euro-Par 2014.

Lago, D. G., Madeira, E. R. M., and Bittencourt, L. F. (2012). Escalonamento com prioridade na alocação ciente de energia de máquinas virtuais em nuvens. XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos.

Ledyayev, R. and Richter, H. (2014). High performance computing in a cloud using openstack. The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization.

Litvinski, O. and Gherbi, A. (2013). Experimental evaluation of openstack compute scheduler. Fourth International Conference on Ambient Systems, Networks and Technologies.

Openstack (2015). official site. `http://www.openstack.org/`. Último acesso em : 26/02/2015.

OpenStack Wiki (2015). Ceilometer. `https://wiki.openstack.org/wiki/Ceilometer`. Último acesso em: 26/02/2015.

Stage, A. and Setzer, T. (2009). Network-aware migration control and scheduling of differentiated virtual machine workloads. International Conference on Software Engineering 2009.