Um Servidor de Máquinas Virtuais Adaptado a Múltiplas Pilhas de Protocolos*

Rafael dos Santos Alves, Miguel Elias Mitre Campista e Luís Henrique Maciel Kosmalski Costa

[1]Grupo de Teleinformática e Automação – PEE/COPPE – DEL/POLI
Universidade Federal do Rio de Janeiro (UFRJ)

{santos,miguel,luish}@gta.ufrj.br

Resumo. *A Internet atual alcançou grande sucesso devido a características como o argumento fim-a-fim e a pilha TCP/IP. Entretanto, essa arquitetura dificulta a adição do suporte à mobilidade, à segurança e à qualidade de serviço. Nesse sentido, este trabalho apresenta um servidor de máquinas virtuais, uma aplicação capaz de criar e controlar roteadores virtuais em diferentes estações físicas. O servidor proposto facilita a implantação de soluções pluralistas baseadas em virtualização de computadores, permitindo a criação de redes virtuais sob demanda e a interação com os usuários dos recursos de rede. O servidor é implementado utilizando o conceito de* Web services *e um protótipo operacional com estações Xen existe atualmente.*

Abstract. *The Internet success is based on characteristics such as the end-to-end argument and the TCP/IP protocol stack. Nevertheless, this architecture hinders the addition of mobility, security and quality of service support. In this scenario, this work, proposes a virtual machine server, an application able to create and control virtual routers in different physical machines. The proposed server facilitates the deployment of pluralist solutions based on computer virtualization, allowing the creation of virtual network on demand and also the interaction between users and network resources. The server is implemented using Web services and a prototype with Xen stations is currently operational.*

1. Introdução

O enorme sucesso da Internet é consequência de decisões fundamentais tomadas em sua origem como a inteligência nas extremidades e a comutação de pacotes. Inicialmente, os requisitos fundamentais eram tornar a rede tolerante a falhas e prover suporte à heterogeneidade dos nós e dos serviços. O protocolo IP (*Internet Protocol*) surgiu com o papel de interconectar toda a rede e prover conectividade fim-a-fim, ao mesmo tempo em que mantinha o núcleo da rede simples, reservando aos nós das extremidades as tarefas mais complexas. Essas escolhas foram tomadas em momentos nos quais não se previa nós tão heterogêneos quanto alguns estáticos e outros móveis. Ainda, que a heterogeneidade fosse tamanha que alguns nós ou serviços não seriam confiáveis, ou que alguns serviços poderiam demandar tratamento diferenciado da rede como, por exemplo, requisitos de qualidade de serviço.

*Este trabalho foi realizado com recursos do FUNTTEL, FINEP, CNPq, CAPES e FAPERJ.

Muitos dos desafios da Internet atual estão relacionados com o atendimento de requisitos como mobilidade, segurança e qualidade de serviço em uma rede criada há décadas, sem prejudicar o seu funcionamento. Nesse sentido, muitos projetos vêm sendo desenvolvidos para propor uma nova Internet, também conhecida como Internet do Futuro. Um dos pioneiros foi o projeto *clean-slate* [Clark et al., 2004] que propôs recriar a Internet com base nas experiências adquiridas ao longo dos anos de operação. Há ainda propostas na direção oposta, que partem do pressuposto de que uma mudança drástica como essa é inviável economicamente e que a Internet deve continuar sendo adaptada à sua constante evolução [Rexford e Dovrolis, 2010]. Essa última abordagem, entretanto, tem que lidar com a multiplicidade de requisitos muitas vezes conflitantes em apenas uma arquitetura e correr o risco de não atender satisfatoriamente nenhum usuário. A solução única pode ser muito complexa e jamais alcançada. Assim, uma possibilidade que vem ganhando atenção é a solução pluralista na qual múltiplas redes virtuais com requisitos diferentes podem ser executadas em paralelo compartilhando o mesmo meio físico virtualizado. Para isso, o conceito de virtualização de máquinas é estendido para redes através de softwares hipervisores que gerenciam o acesso ao hardware de roteadores entre múltiplos roteadores virtuais[1]. Entende-se, portanto, que uma rede virtual é um conjunto de roteadores virtuais e os enlaces entre eles.

Este trabalho propõe a criação de um servidor de máquinas virtuais, uma aplicação capaz de criar, remover, migrar e gerenciar roteadores virtuais em diferentes máquinas físicas. Tal servidor facilita a implantação de soluções pluralistas baseadas em virtualização de computadores. Para isso, o servidor possui um repositório de imagens de máquinas que pode ser utilizado para transferir e instanciar remotamente roteadores virtuais personalizados. Tal característica permite a interação entre usuários confiáveis, administradores ou sistemas inteligentes, e a rede. Uma possibilidade tangível para oferecer interação aos usuários ainda mantendo certo controle é disponibilizar um conjunto de imagens pré-configuradas para que os usuários possam escolher entre elas conforme as suas necessidades. Uma vez que as imagens tenham sido transferidas e instanciadas, o servidor virtual ainda pode gerenciar a migração de roteadores ao vivo e destruir nós selecionados.

No sistema proposto, toda comunicação de usuários com o Servidor de Máquinas Virtuais ocorre através de mensagens do protocolo SOAP (*Simple Object Access Protocol*) [Box et al., 2000] sobre HTTP (*HyperText Transfer Protocol*). O Servidor de Máquinas Virtuais, por sua vez, se comunica com as máquinas físicas que hospedam as máquinas virtuais indicadas nas mensagens SOAP através de uma API de gerenciamento como, por exemplo, a `Libvirt` [Libvirt, 2011] ou a `XenAPI` [Citrix, 2011]. É importante observar que o serviço é totalmente transparente, o que significa que o servidor pode ser atualizado sem alterar a interface com seus clientes. Os resultados obtidos demonstram a operacionalidade do servidor através de provas de conceito realizadas em uma rede de testes no Laboratório do Grupo de Teleinformática e Automação (GTA) da UFRJ.

O restante deste trabalho está dividido da seguinte forma. A Seção 2 apresenta os trabalhos relacionados e descreve o modelo da arquitetura no qual está inserido o Servidor de Máquinas Virtuais proposto neste trabalho. A Seção 3 apresenta o Servidor de

[1]O termo máquinas virtuais e roteadores virtuais são usados neste trabalho de maneira intercalada.

Máquinas Virtuais, sua arquitetura e implementação. A Seção 4 apresenta o protótipo para testes do servidor implantado no laboratório do GTA. A Seção 5 apresenta os experimentos realizados e a Seção 6 conclui este trabalho.

2. Trabalhos Relacionados

Muitos trabalhos na literatura investigam soluções para a Internet do Futuro. Esses trabalhos podem ser divididos em duas categorias: puristas e pluralistas [Moreira et al., 2009]. A abordagem purista procura atender aos diversos requisitos da Internet através da utilização de uma única pilha de protocolos, que deve ser flexível o suficiente para atender as diferentes demandas. Por outro lado, a abordagem pluralista utiliza pilhas de protocolos em paralelo de forma que cada pilha atenda a um conjunto específico de requisitos.

Entre as arquiteturas que utilizam a abordagem purista pode-se citar a arquitetura baseada em papéis [Clark et al., 2004]. Nesse modelo não existe a multiplicidade de camadas evitando, de forma inerente, o problema comum dos protocolos da Internet que é a violação de camadas. Para substituir as camadas, módulos que os autores chamam de papéis são utilizados para permitir a modularização do desenvolvimento dos protocolos. O diferencial importante quando comparado ao modelo em camadas é a inexistência de níveis hierárquicos entre os papéis. É importante notar que os papéis devem ser blocos bem conhecidos e padronizados, permitindo assim a criação de serviços bem definidos.

A arquitetura DONA (*Data-Oriented Network Architecture*) [Koponen et al., 2007] parte do princípio que a Internet passou de centrada em estação para centrada em dados. Portanto, os usuários não estão mais interessados em quem ou onde podem adquirir conteúdo, mas na obtenção desse conteúdo em tempo hábil. Um dos principais representantes do conceito centrado em estação é o sistema de resolução de nomes, o DNS (*Domain Name System*), que responde a solicitações de endereços IP recebendo como parâmetro o nome do servidor de destino. Ao invés do DNS, a arquitetura DONA propõe o uso de primitivas *anycast* baseadas em nomes inseridos acima da camada de rede.

Entre as arquiteturas pluralistas pode-se destacar a arquitetura CABO (*Concurrent Architectures are Better than One*) [Feamster et al., 2007] que propõe utilizar máquinas virtuais de modo que em cada rede virtual uma configuração ou até mesmo uma pilha de protocolos distinta seja utilizada. Dessa forma, múltiplas redes, com diferentes características estão disponíveis em um mesmo substrato físico. A principal motivação por trás da arquitetura CABO é permitir que os ISPs (*Internet Service Providers*) ofereçam serviços diferenciados aos seus clientes, o que atualmente não é possível já que nenhum provedor de serviço possui roteadores em todo o percurso fim-a-fim entre todos os usuários da Internet. Para isso, é proposta a separação entre provedores de serviço e provedores de infraestrutura. Neste caso, os provedores de infraestrutura forneceriam recursos computacionais e de rede para os provedores de serviço, através de acordos comerciais. Um provedor de serviço, por sua vez, pode utilizar recursos de diferentes provedores de infraestrutura e com isso, construir um caminho fim-a-fim de roteadores, possibilitando a oferta de serviços diferenciados aos usuários finais.

O projeto franco-brasileiro Horizon [Horizon Project, 2011] tem por objetivo desenvolver uma arquitetura para a Internet baseada nos conceitos do pluralismo e de inte-

ligência intra e inter-redes virtuais. Nesse projeto, para cada pilha de protocolos existe um conjunto de máquinas virtuais instanciadas ao longo da rede, executando essa pilha. Além disso, para garantir um desempenho mínimo para as redes propõe-se a criação de um plano de pilotagem. Esse plano tem por objetivo sensoriar e atuar na rede. Através de observações colhidas por um conjunto diverso de equipamentos de medidas e do conhecimento acumulado no plano de conhecimento, o plano de pilotagem deve decidir se alguma mudança de configuração deve ser realizada na rede. Em caso positivo, o plano de pilotagem deve acionar os procedimentos de software necessários para que a tarefa seja realizada. O plano de pilotagem é responsável por coordenar os planos de gerenciamento, de controle e de virtualização através, por exemplo, da alteração de parâmetros de configuração de um protocolo de roteamento. Os resultados das alterações implementadas são colhidas como conhecimento a ser disseminado pela rede.

Uma tecnologia importante em muitas das abordagens pluralistas é a virtualização de computadores [Popek e Goldberg, 1974, Egi et al., 2008]. Através da separação dos recursos computacionais, diferentes pilhas de protocolos podem conviver em um mesmo ambiente. Vale notar também que essa tecnologia pode ser utilizada para a construção de redes experimentais para realização de testes de ambas as abordagens, como já é feito em projetos como o PlanetLab [Chun et al., 2003] e GENI [GENI, 2011].

Este trabalho propõe um servidor de máquinas virtuais voltado para a abordagem pluralista que tem como principal objetivo a automatização do processo de gerenciamento de máquinas e, em última instância, de redes virtuais. Vale notar que este trabalho encontra-se no escopo do projeto Horizon, embora, sob o ponto de vista funcional, pudesse ser utilizado por outras propostas, como por exemplo, o CABO, que depende de uma infraestrutura muito semelhante a necessária pelo projeto Horizon, formada por máquinas com suporte à virtualização que oferecem suporte a diferentes redes virtuais.

3. O Servidor de Máquinas Virtuais

O Servidor de Máquinas Virtuais provê um conjunto de serviços Web. Podem figurar como clientes do Servidor de Máquinas Virtuais, por exemplo, um administrador da rede, ou um sistema autônomo de gerenciamento, em última instância, qualquer agente interessado em monitorar ou alterar recursos da rede. Além das tarefas de criação de máquinas virtuais e de redes virtuais, serviços básicos do servidor, um conjunto de serviços extras foi adicionado ao servidor tornando-o um controlador de redes virtuais. Dessa forma, esse servidor pode ser utilizado como ferramenta para a realização das tarefas definidas pelo administrador. Por exemplo, caso o administrador perceba um gargalo de desempenho de uma rede virtual em um determinado nó, uma requisição para o aumento de recursos (memória, CPU etc.) pode ser feita através de um serviço provido pelo servidor.

De acordo com a proposta deste trabalho, cada um dos roteadores físicos é equipado com alguma tecnologia, como o Xen [Barham et al., 2003] ou VMWare [VMware, 2011], que os torna capazes de hospedar sistemas virtualizados. A função básica do Servidor de Máquinas Virtuais é criar sob demanda máquinas virtuais nos nós físicos da rede e configurar seus enlaces de forma a garantir que a rede esteja ativa e conectada após o término de toda operação.

3.1. Arquitetura do Servidor

A Figura 1 apresenta o cenário no qual o Servidor de Máquinas Virtuais é utilizado no contexto do Projeto Horizon. Na mesma figura, também podem ser observados o plano de pilotagem proposto no projeto Horizon e uma máquina com suporte a virtualização. Sempre que o plano de pilotagem requisitar um dos serviços providos pelo Servidor de Máquinas Virtuais, uma mensagem deve ser enviada utilizando o protocolo SOAP sobre HTTP para o servidor. O Servidor de Máquinas Virtuais, por sua vez, comunica-se com as máquinas físicas que hospedam as máquinas virtuais indicadas na mensagem SOAP e que serão acessadas através de uma API de gerenciamento como, por exemplo, a `Libvirt` ou a `XenAPI`. É importante observar que, para o plano de pilotagem, a forma como o serviço vai ser realizado é totalmente transparente, permitindo que o Servidor de Máquinas Virtuais seja atualizado, mudando a implementação de seus serviços, sem alterar a interface com seus clientes.

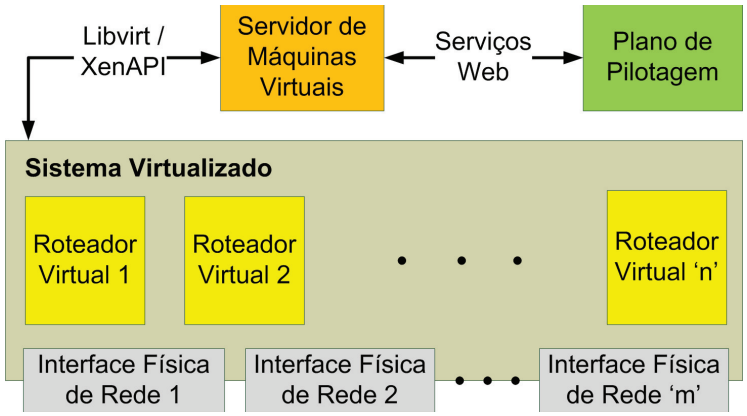

Figura 1. Arquitetura do Servidor de Máquinas Virtuais proposto.

Na maioria dos serviços oferecidos, o cliente deve esperar uma mensagem de retorno informando o resultado da operação. Em caso de falha, um relatório com os motivos da falha é enviado ainda por serviço Web ao cliente.

É importante notar que não existe qualquer restrição conceitual ao tipo de protocolo de comunicação instalado na máquina virtual. A limitação que pode existir é do ponto de vista da implementação. Um determinado protocolo pode não estar disponível para o sistema operacional desejado, por exemplo.

A Figura 2 mostra como o sistema evolui ao longo do tempo. Nesse exemplo, assume-se que o serviço requisitado é o de criação de uma máquina virtual (`createVirtualMachine`). Na primeira interação do sistema (Mensagem 1), o plano de pilotagem - o cliente - envia uma mensagem SOAP contendo uma requisição do serviço. Ao receber a mensagem, o Servidor de Máquinas Virtuais identifica o sistema Xen indicado na mensagem e envia (Mensagem 2), através de uma biblioteca de gerenciamento de sistema virtualizado, uma requisição para a criação de uma máquina virtual com as características definidas na mensagem que o plano de pilotagem enviou.

Em seguida, o sistema Xen tenta criar a máquina virtual solicitada e envia o resultado da operação ao Servidor de Máquinas Virtuais (Mensagem 3). O termo "criar máquina virtual" aqui utilizado refere-se ao processo de definição de um arquivo que será

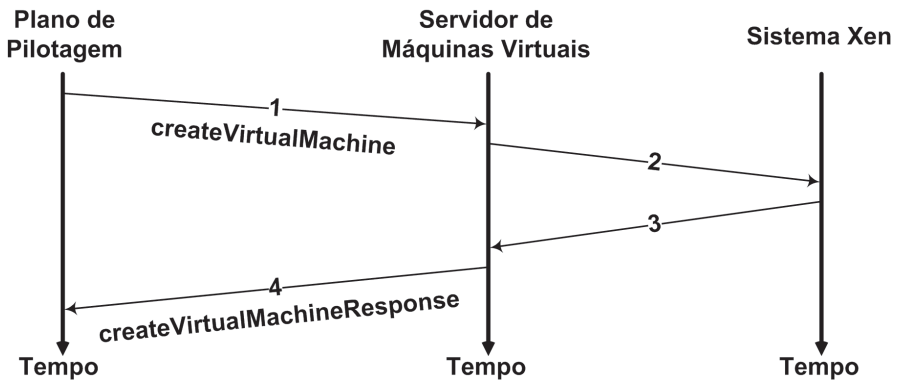

Figura 2. Evolução do sistema no tempo.

utilizado como disco virtual para a máquina hóspede e outros parâmetros, como tamanho da memória, número de interfaces de rede etc. Finalmente, o servidor cria uma mensagem XML com o resultado da operação e a envia ao plano de pilotagem (Mensagem 4).

Um ponto importante a ser ressaltado é a localização do arquivo que será utilizado como disco virtual. Esse arquivo deve conter o sistema operacional desejado, com as funções requeridas pelo serviço. Por exemplo, caso a máquina virtual requisitada seja um roteador IPv4, o arquivo deve conter um sistema operacional Linux, com suporte a IPV4 e o protocolo de roteamento RIP (*Routing Information Protocol*). Esse arquivo pode ser armazenado em locais diferentes. O Servidor de Máquinas Virtuais pode também armazenar diferentes imagens de sistemas operacionais e transferí-las para as estações físicas no momento em que máquinas virtuais forem criadas.

3.2. Implementação

Um protótipo do Servidor de Máquinas Virtuais foi implementado e implantado dentro do laboratório do GTA com o objetivo de realizar análises de prova de conceito. Os serviços atualmente disponíveis estão descritos na Tabela 1.

O serviço de criação de máquinas virtuais (`createVirtualMachine`) utiliza os parâmetros passados pela requisição do serviço para criar um nó virtual no nó físico da rede especificado. Vale destacar que esse serviço tem duas variações principais. Na primeira, o Servidor de Máquinas Virtuais envia o arquivo utilizado como disco virtual para o servidor físico que irá abrigar a nova máquina virtual. Na segunda, o disco deve existir em algum local acessível pelo servidor físico que abriga o roteador virtual. O serviço de criação de redes virtuais (`createVirtualNetwork`) deve criar um conjunto de nós virtuais em máquinas físicas da rede. Além disso, para que os nós criados formem uma rede, deve-se realizar o mapeamento entre a interface física indicada e a interface virtual criada, além da configuração dos endereços de rede.

Quando o serviço de criação de máquinas virtuais é utilizado transferindo o disco virtual, os detalhes referentes ao tipo de máquina virtual a ser utilizada devem ser definidos. Alguns serviços foram criados com o objetivo de mostrar aos clientes os tipos de sistema operacional disponíveis no Servidor de Máquinas Virtuais. O serviço `getAvailableOSes` apresenta os sistemas operacionais disponíveis (Linux, Windows etc.), já o serviço `getAvailableArch` apresenta as arquiteturas disponíveis e final-

Tabela 1. Serviços oferecidos pelo Servidor de Máquinas Virtuais.

Serviço	Descrição
createVirtualMachine	criação de máquinas virtuais
createVirtualNetwork	criação de redes virtuais
destroyVirtualMachine	exclusão de máquinas virtuais
getAvailableArch	arquiteturas disponíveis no servidor
getAvailableKernelVersions	versões de kernel disponíveis no servidor
getAvailableOSes	sistemas operacionais disponíveis no servidor
getPhysicalServerStatus	informações sobre um servidor físico
getRegisteredNodes	lista de nós registrados
getVirtualMachineSchedulerParameters	consulta parâmetros de escalonamento de máquina virtual
getVirtualMachineSchedulerType	consulta de tipo de escalonador de máquina virtual
getVirtualMachineStatus	informações sobre determinado domínio virtual
migrateVirtualMachine	migração de máquinas virtuais
registerNodes	registro de novo nó físico na rede
sanityTest	teste de sanidade sobre funcionamento do servidor
setVirtualMachineSchedulerParameters	ajuste de parâmetros de escalonamento de máquina virtual
shutdownVirtualMachine	desligamento de máquina virtual

mente o serviço `getAvailableKernelVersions` apresenta as versões de kernel disponíveis.

Além das tarefas básicas do Servidor de Máquinas Virtuais, tarefas adicionais foram implementadas. O serviço `destroyVirtualMachine` pode ser utilizado para destruir uma máquina virtual, por exemplo, quando a rede para a qual ela foi criada não é mais necessária. Em alguns casos a máquina virtual deverá ser desligada para ser religada depois de um tempo, por exemplo para economia de energia. O serviço `shutdownVirtualMachine` pode ser utilizado com esse intuito. Os serviços `getPhysicalServerStatus` e `getVirtualMachineStatus` têm por objetivo obter algumas informações gerais, como memória e número de processadores, acerca de uma máquina física e de uma virtual, respectivamente. Essas informações podem ser utilizadas pelo plano de pilotagem no processo de tomada de decisões. O serviço de migração (`migrateVirtualMachine`) pode ser utilizado para mover uma máquina virtual de um nó físico para outro, por exemplo, quando um determinado nó físico encontra-se sobrecarregado. Esse serviço utiliza o mecanismo de migração padrão do Xen [Clark et al., 2005], porém outros mecanismos podem ser utilizados, como por exemplo o mecanismo de migração com separação de planos em [Pisa et al., 2010].

Nos sistemas operacionais multitarefa, escalonadores de CPU são utilizados para dividir recursos de processamentos entre os processos. De forma semelhante, o Xen utiliza um escalonador de CPU para compartilhar recursos entre as máquinas virtuais. Entre os principais escalonadores utilizados no Xen pode-se citar o *Credit Scheduler* [Citrix, 2007], o SEDF (*Simple Earliest Deadline First*) [Leslie et al., 1996] e o BVT (*Borrowed Virtual Time*) [Duda e Cheriton, 1999]. Atualmente, o escalonador padrão do Xen é o *Credit Scheduler*. O tipo de escalonador utilizado pela máquina virtual pode ser consultado pelo serviço `getVirtualMachineSchedulerType`. Além disso, os parâmetros de escalonamento de uma máquina virtual podem ser consultados e alterados pelos serviços `getVirtualMachineSchedulerParameters` e `setVirtualMachineSchedulerParameters`, respectivamente.

Inicialmente, o Servidor de Máquinas Virtuais não tem conhecimento sobre as máquinas físicas às quais tem acesso. Para garantir que o Servidor de Máquinas Virtuais tenha conhecimento desses nós, o serviço `registerNodes` permite que um determinado nó seja registrado no servidor para futura administração, ou seja, o nome, a chave pública e os endereços IP são enviados ao servidor que armazena essas informações localmente. Os nós registrados podem ser consultados pelo serviço `getRegisteredNodes`.

O protótipo foi implementado utilizando a linguagem de programação Java. A biblioteca `Libvirt` [Libvirt, 2011], versão 0.7.5 foi utilizada para a realização de tarefas administrativas e a biblioteca `Axis2` [Perera et al., 2006] em sua versão 1.5.1 para a construção dos serviços Web. O servidor Web utilizado foi o `Tomcat` [Apache, 2011], versão 6. Completando o ambiente de desenvolvimento, a IDE (*Integrated Development Environment*) utilizada foi o `NetBeans` [Oracle, 2011a] versão 6.7.1. Adotou-se como plataforma de virtualização o Xen. Essa plataforma de virtualização possui grande apoio da comunidade acadêmica, o que a torna mais confiável, já que um grande número de pesquisadores e usuários comuns tem utilizado essa tecnologia ao redor do mundo [Egi et al., 2007, Clark et al., 2005, Cherkasova et al., 2007]. Além disso, o Xen é distribuído sob uma licença de código livre, permitindo que alterações em seu funcionamento sejam propostas.

A linguagem de programação Java [Oracle, 2011b] é uma linguagem orientada a objetos com suporte aos principais sistemas operacionais disponíveis. Além disso, conta com uma grande quantidade de bibliotecas publicamente disponíveis que foi um dos fatores determinantes para a escolha dessa linguagem para o protótipo. Além disso, o suporte a serviços Web em Java contaram pontos a favor nessa decisão. Finalmente, um programa Java é, em princípio, multiplataforma o que permite que ele seja executado em qualquer estação desde que esta possua uma máquina virtual Java.

A `Libvirt` é uma biblioteca para gerenciamento de sistemas virtualizados de código livre. A biblioteca foi originalmente desenvolvida em C e atualmente provê suporte para um grande conjunto de linguagens, a destacar Java e Python. Os recursos de gerenciamento da `Libvirt` são genéricos, ou seja, funções comuns à maioria dos sistemas de virtualização, por exemplo, Xen, VMware, OpenVZ, QEMU etc., são providas. Essa generalidade da biblioteca permite que mesmo no caso de alteração da plataforma de virtualização, a maior parte do código do Servidor de Máquinas Virtuais possa ser reaproveitada. Essa propriedade é altamente desejável para tornar o Servidor de Máquinas Virtuais útil em cenários mais amplos do que seria se comparado ao uso de uma biblioteca de administração específica para sistemas Xen.

A biblioteca `Axis2` desenvolvida pela *Apache Foundation* implementa o protocolo SOAP. É importante observar que essa biblioteca também é publicada sob uma licença de código livre. Atualmente, a Axis2 está implementada em C e em Java, sendo a última a implementação utilizada neste protótipo.

Caso alguma tarefa atualmente não disponível seja necessária, a adição de um novo serviço é simples e não afeta o funcionamento dos outros serviços previamente disponíveis. Cada serviço no Servidor de Máquinas Virtuais é implementado como um método em sua classe principal (`VirtualMachineServer`). Todo serviço deve ser

implementado como um método público que recebe um objeto da classe `OMElement` (*Object Model Element*) e retorna outro objeto da classe `OMElement`. Essa classe é oferecida pela biblioteca Axis2 e tem por objetivo armazenar um elemento XML, ou seja, depois de transformado em uma *string*, um objeto `OMElement` torna-se uma *tag* de uma mensagem XML. Neste caso, o elemento recebido como parâmetro é o conteúdo de uma mensagem SOAP, e o `OMElement` retornado será também o conteúdo da mensagem SOAP enviada como resposta pelo Servidor de Máquinas Virtuais.

3.3. Acesso ao Servidor de Máquinas Virtuais

Com o objetivo de facilitar a criação de sistemas de software que acessem o Servidor de Máquinas Virtuais, uma classe foi desenvolvida. Para cada serviço oferecido, a classe `HorizonXenClient` possui um método para a criação do conteúdo da mensagem. A Listagem 1 apresenta a API oferecida por essa classe.

Listagem 1. API oferecida pela classe `HorizonXenClient`.

```
public OMElement createVirtualMachinePayload(String phyServer, String
    vmName, String vmIP, String vmRAM);
public OMElement createVirtualNetworkPayload(Vector<String> phyServers,
     Vector<String> VMNames, Vector<String> IPs, Vector<String> RAMs,
    Vector<String> netInterface);
public OMElement destroyVirtualMachinePayload(String phyServer, String
    vmName);
public OMElement getAvailableArch();
public OMElement getAvailableKernelVersions();
public OMElement getAvailableOSes();
public OMElement getPhysicalServerStatusPayload(String phyServer);
public OMElement getRegisteredNodesPayload();
public OMElement getVirtualMachineSchedulerParametersPayload(String
    phyServer, String VMName);
public OMElement getVirtualMachineSchedTyplerePayload(String phyServer,
     String VMName);
public OMElement getVirtualMachineStatusPayload(String phyServer,
    String vmName);
public OMElement migrateVirtualMachinePayload(String sourcePhyServer,
    String destPhyServer, String vmName, String live);
public OMElement registerNodesPayload(Vector<PhysicalServer> phyServers
    );
public OMElement sanityTestPayload(String testString);
public OMElement setVirtualMachineSchedulerParametersPayload(String
    phyServer, String VMName, String Weight, String Cap);
public OMElement shutdownVirtualMachinePayload(String phyServer, String
     vmName);
```

Observa-se que todos os métodos retornam um objeto da classe `OMElement` que será utilizado como conteúdo de mensagens SOAP. As ocorrências `vmName` referem-se ao nome esperado para a máquina virtual. Esse nome será o nome acessível através dos recursos de administração de sistemas virtualizados e para interações futuras com o Servidor de Máquinas Virtuais. O parâmetro `phyServer` refere-se ao nome DNS externamente acessível do nó físico ou o endereço IP desse nó. Os parâmetros `vmIP` e `vmRAM` apontam, respectivamente, o endereço IP e o tamanho da memória RAM desejados para o novo domínio virtual.

Existem ainda parâmetros específicos para a função de migração de máquinas virtuais: `sourcePhyServer` e `destPhyServer` definem, respectivamente, os nós físicos de origem e de destino do domínio virtual a ser migrado; o parâmetro `live`, que pode receber os valores `true` ou `false`, define se a migração deve ser realizada ao vivo, ou seja, sem interrupção do funcionamento do domínio virtual. Para isso, as páginas de memória utilizadas pela execução da máquina virtual são armazenadas e transferidas para a máquina de destino para que a execução retorne no mesmo estado que estava antes da transferência.

Alguns serviços podem atuar sobre um conjunto de máquinas físicas e virtuais. Nesses casos os parâmetros esperados são vetores e a semântica é similar aos casos já apontados. Existe ainda o parâmetro `testString` do método de teste de sanidade. Esse parâmetro define a *string* que formará o corpo da mensagem de teste e que será retornada pelo servidor, caso o servidor esteja funcionando corretamente. Com relação ao serviço de alteração de parâmetros do escalonador, os parâmetros `Weight` e `Cap` são referentes aos parâmetros do *Credit Scheduler* [Citrix, 2007] utilizado pelo Xen.

É importante observar que a utilização da classe `HorizonXenClient` não é obrigatória. O cliente pode ser construído sem fazer uso dessa biblioteca. Não existe sequer limitação quanto à linguagem de programação, já que a utilização de um serviço Web permite essa flexibilidade. O único requisito é a necessidade de utilização do protocolo SOAP e de que o conteúdo da mensagem seja um XML válido com os campos esperados pelo servidor.

4. Protótipo

Para a realização de testes, um protótipo foi implantado no laboratório do Grupo de Teleinformática e Automação da UFRJ. Esse protótipo serve como prova de conceito para o Servidor de Máquinas Virtuais. A Tabela 2 apresenta os computadores utilizados e suas respectivas funções.

Tabela 2. Computadores no protótipo e suas configurações.

Computador	Arquitetura	Kernel	Processador	Memória
vms	i386	2.6.30-2	Core 2 2.13 GHz	2 GB
xen1	i386	2.6.32-5-xen	Celeron 2.8 GHz	3 GB
xen2	amd64	2.6.32-5-xen	Core 2 Duo 2.53 GHz	4 GB
xen3	i386	2.6.32-5-xen	Pentium 4 HT 3.4 GHz	2 GB

A Figura 3 representa a topologia da rede de testes. Todas as máquinas podem ser acessadas através do roteador que as conecta à Internet. A comunicação entre o Servidor de Máquinas Virtuais, hospedado na máquina `vms`, e as outras máquinas, ocorre através desse roteador. As estações `xen1`, `xen2` e `xen3` são máquinas que executam o Xen como software de suporte à virtualização e são operadas pelo Servidor de Máquinas Virtuais. No estado atual, o mecanismo de migração de máquinas virtuais do Xen é utilizado. Nesse mecanismo é necessário que o disco virtual esteja em algum local acessível para as duas máquinas físicas que participam do processo de migração. Essa limitação vem sendo estudada por alguns trabalhos na literatura. Por exemplo, o trabalho de [Mattos et al., 2011] apresenta uma solução para essa limitação

através de uma técnica de virtualização híbrida combinando as plataformas Xen e Openflow [McKeown et al., 2008].

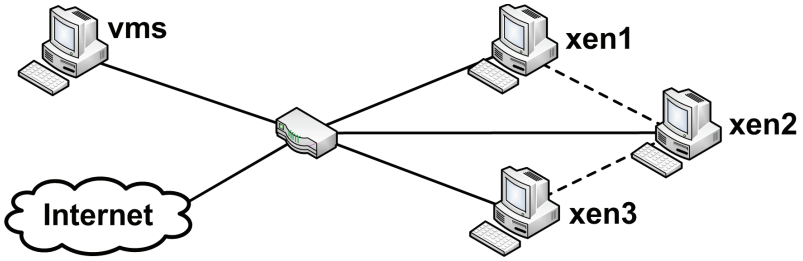

Figura 3. Topologia do *testbed*.

Todas as estações executam o sistema operacional Linux Debian. As versões de kernel e de arquiteturas utilizadas são descritas na Tabela 2. Com o objetivo de facilitar a autenticação do Servidor de Máquinas Virtuais, um mecanismo de SSH (*Secure Shell*) sem senha foi configurado entre o Servidor de Máquinas Virtuais e as máquinas. Mais especificamente, a autenticação é realizada através da utilização de chaves públicas configuradas previamente [Corp., 2003].

5. Experimentos e Observações

Experimentos foram realizados para avaliar algumas métricas de desempenho do Servidor de Máquinas Virtuais proposto. A partir da classe `HorizonXenClient`, dois clientes foram desenvolvidos para o Servidor de Máquinas Virtuais. O primeiro, um cliente JAR (*Java ARchive*) permite a requisição de serviços a partir da linha de comandos. O segundo um cliente JSP (*JavaServer Pages*) permite a requisição de serviços a partir de uma página Web.

O protótipo apresentado na Seção 4 foi utilizado para os experimentos, que têm por objetivo avaliar o consumo de recursos por alguns serviços do Servidor de Máquinas Virtuais. O cliente JAR foi utilizado nos experimentos sempre a partir de uma máquina externa ao protótipo, ou seja, além das quatro máquinas apresentadas no protótipo, uma quinta máquina assumiu o papel de cliente nos experimentos. Foram realizadas trinta rodadas experimentais para cada serviço e valores médios e de desvio padrão associados são apresentados.

Na Tabela 3 encontram-se os dados referentes à criação de máquinas virtuais. Para isso, duas possibilidades existem: transferência da imagem do servidor para as máquinas Xen e posterior inicialização da máquina virtual ou inicialização de máquina virtual já na máquina Xen. Nos experimentos, as duas variações do serviço (com ou sem transferência de disco virtual) foram avaliadas. Para os experimentos de criação com transferência de disco, duas métricas foram avaliadas: o tempo de execução do serviço para o cliente, que inclui o tempo de comunicação entre o cliente e o Servidor de Máquinas Virtuais, a transferência da imagem entre o Servidor de Máquinas Virtuais e a máquina Xen e o tempo de inicialização da máquina virtual; e o tempo de transferência de disco isoladamente.

Como visto, a transferência de disco tem impacto relevante no tempo total para a criação da máquina virtual. Entretanto, com ela não existe como pré-requisito a presença da imagem nos elementos de rede. Essa possibilidade oferece maior liberdade para que o

cliente possa criar sob demanda a sua máquina virtual personalizada. Além disso, ao comparar os tempos entre as máquinas `xen1` e `xen3` observa-se que a melhor configuração (memória e processamento maiores) da máquina `xen1` leva a um melhor desempenho desta máquina. Esse comportamento repete-se na comparação entre as máquinas `xen1` e `xen2`, onde a última possui menores tempos para o cliente, nas duas variações do serviço de criação de máquinas virtuais, e para a transferência de discos.

Tabela 3. Tempos na criação de máquinas virtuais (em segundos).

	Com transferência de disco		Sem transferência de disco
Computador	**Cliente**	**Transferência de disco**	**Cliente**
xen1	43,77 $\pm$ 1,35	35,47 $\pm$ 0,41	6,86 $\pm$ 0,25
xen2	42,20 $\pm$ 1,62	35,21 $\pm$ 0,13	5.72 $\pm$ 0,37
xen3	46,62 $\pm$ 5,80	36,52 $\pm$ 4,99	7,05 $\pm$ 1,74

A Tabela 4 apresenta o consumo de recursos de processamento no cliente e no Servidor de Máquinas Virtuais para as duas variações do serviço de criação de máquinas virtuais. Pode-se observar que os recursos de CPU exigidos no cliente são pequenos. A semelhança do processamento utilizado nos diferentes cenários corrobora o baixo consumo de recursos. Isso possibilita a utilização de clientes em dispositivos com baixa capacidade de processamento, como celulares, ou ainda a utilização de um sistema autônomo, no qual os clientes podem ser agentes móveis, por exemplo.

Tabela 4. Processamento utilizado (em segundos).

Computador	Com transferência de disco	Sem transferência de disco
xen1	0,82 $\pm$ 0,03	0,79 $\pm$ 0,01
xen2	0,79 $\pm$ 0,01	0,79 $\pm$ 0,01
xen3	0,81 $\pm$ 0,02	0,79 $\pm$ 0,01

Tanto o servidor como os clientes implementados, além da documentação de uso e instalação do Servidor de Máquinas Virtuais podem ser encontrados no seguinte sítio Web `http://www.gta.ufrj.br/~santos/vms`.

6. Conclusão e Trabalhos Futuros

Neste trabalho foi desenvolvido um servidor de máquinas virtuais adaptado a diferentes pilhas de protocolos. O servidor aqui apresentado é uma importante ferramenta no desenvolvimento de novas propostas de protocolos de comunicação para lidar com a complexidade e a multiplicidade de requisitos que ora se apresentam para a Internet.

O servidor foi implementado utilizando a linguagem de programação Java. Um conjunto de serviços está atualmente disponível e pode ser acessado com ajuda da classe de apoio desenvolvida. Essa classe, de uso opcional, foi desenvolvida também em Java e pode ser utilizada como forma de redução do tempo de desenvolvimento de clientes para o Servidor de Máquinas Virtuais.

Como trabalhos futuros, espera-se ampliar o número de serviços oferecidos pelo servidor, tais como a oferta de estatísticas mais detalhadas sobre os estados das máquinas físicas e virtuais da rede, tornando-o, na prática, um controlador de máquinas virtuais.

Além disso, o protótipo deve ser ampliado permitindo a exploração de um cenário maior do que o experimentado até o momento.

Referências

Apache (2011). Apache tomcat. http://tomcat.apache.org/. Acessado em março de 2011.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. e Warfield, A. (2003). Xen and the art of virtualization. Em *ACM Symposium on Operating Systems Principles (SOSP)*, pp. 164–177.

Box, D., Ehnebuske, D., Kakivaya, G., Mendelsohn, A. L. N., Nielsen, H. F., Thatte, S. e Winer, D. (2000). Simple object access protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Acessado em março de 2011.

Cherkasova, L., Gupta, D. e Vahdat, A. (2007). Comparison of the three CPU schedulers in Xen. *SIGMETRICS Performance Evaluation Review*, 35(2):42–51.

Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M. e Bowman, M. (2003). Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12.

Citrix (2007). Credit-based CPU scheduler. http://wiki.xensource.com/xenwiki/Credit-Scheduler. Acessado em abril de 2011.

Citrix (2011). Xen management API project. http://wiki.xensource.com/xenwiki/XenApi. Acessado em abril de 2011.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. e Warfield, A. (2005). Live migration of virtual machines. Em *Symposium on Networked Systems Design & Implementation (NSDI)*, pp. 273–286.

Clark, D., Braden, R., Sollins, K., Wroclawski, J., Katabi, D., Kulik, J., Yang, X., Faber, T., Falk, A., Pingali, V., Handley, M. e Chiappa, N. (2004). New Arch: Future generation Internet architecture. Relatório técnico, MIT Laboratory for Computer Science and International Computer Science Institute (ICSI).

Corp., S. C. S. (2003). *SSH Secure Shell for Servers Version 3.2.9 - Administrator's Guide*. SSH Communications Security.

Duda, K. J. e Cheriton, D. R. (1999). Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. Em *Proceedings of the ACM symposium on Operating systems principles (SOSP)*, pp. 261–276.

Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F. e Mathy, L. (2008). Towards high performance virtual routers on commodity hardware. Em *ACM CoNEXT*.

Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Mathy, L. e Schooley, T. (2007). Evaluating xen for router virtualization. Em *International Conference on Computer Communications and Networks (ICCCN)*, pp. 1256–1261.

Feamster, N., Gao, L. e Rexford, J. (2007). How to lease the Internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37(1):61–64.

GENI (2011). Exploring networks of the future. http://www.geni.net. Acessado em março de 2011.

Horizon Project (2011). A new Horizon to the Internet. http://www.gta.ufrj.br/horizon. Acessado em abril de 2011.

Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S. e Stoica, I. (2007). A data-oriented (and beyond) network architecture. Em *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 181–192.

Leslie, I., McAuley, D., Black, R., Roscoe, T., Barham, P., Evers, D., Fairbairns, R. e Hyden, E. (1996). The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1280–1297.

Libvirt (2011). The virtualization API. http://libvirt.org/. Acessado em março de 2011.

Mattos, D., Fernandes, N. C. e Duarte, O. C. M. B. (2011). Xenflow: Um sistema de processamento de fluxos robusto e eficiente para migração em redes virtuais. Em *Simpósio Brasileiro de Redes de Computadores (SBRC)*. Aceito para publicação.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. e Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM Computer Communications Review*, 38(2):69–74.

Moreira, M. D. D., Fernandes, N. C., Costa, L. H. M. K. e Duarte, O. C. M. B. (2009). *Minicursos do Simpósio Brasileiro de Redes de Computadores (SBRC)*, capítulo Internet do Futuro: Um Novo Horizonte, pp. 1–59. SBC, Recife, PE.

Oracle (2011a). NetBeans. http://netbeans.org/. Acessado em março de 2011.

Oracle (2011b). Oracle technology network for java developers. http://www.oracle.com/technetwork/java/index.html. Acessado em abril de 2011.

Perera, S., Herath, C., Ekanayake, J., Chinthaka, E., Ranabahu, A., Jayasinghe, D., Weerawarana, S. e Daniels, G. (2006). Axis2, middleware for next generation web services. Em *International Conference on Web Services (ICWS)*, pp. 833–840.

Pisa, P., Fernandes, N., Carvalho, H., Moreira, M., Campista, M., Costa, L. e Duarte, O. (2010). OpenFlow and Xen-based virtual network migration. Em Pont, A., Pujolle, G. e Raghavan, S., editors, *Communications: Wireless in Developing Countries and Networks of the Future*, volume 327 of *IFIP Advances in Information and Communication Technology*, pp. 170–181. Springer Boston.

Popek, G. J. e Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421.

Rexford, J. e Dovrolis, C. (2010). Future Internet architecture: clean-slate versus evolutionary research. *Communications of the ACM*, 53(9):36–40.

VMware (2011). VMware virtualization software for desktops, servers and virtual machines for public and private cloud solutions. http://www.vmware.com/. Acessado em março de 2011.