Balanceamento de recursos em ambientes virtualizados

Leandro T. Costa, Filipi D. Teixeira, Elder M. Rodrigues, Avelino F. Zorzo

Programa de Pós-Graduação em Ciência da Computação
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal 1.429 – 90.619-900 – Porto Alegre – RS – Brasil

{leandro.teodoro, filipi.teixeira, elder.rodrigues}@acad.pucrs.br,

avelino.zorzo@pucrs.br

Abstract. *This paper describes the improvements on a subsystem to reallocate resources on the Xen Virtual Machine Monitor (VMM). This subsystem, called SRX, balances the resources of a virtualized enviroment in order to meet Service Level Agreements. The main problems on SRX were related to performance degradation due to creation and destruction of processes every time resources had to be reallocated to a VM. Our strategy was to include the SRX features to the Xen Master (XM) tool. In order to show the improvements of our new tool a set of tests is presented at the end of the paper.* [1]

Resumo. *Este trabalho descreve as melhorias realizadas sobre um subsistema de realocação de recursos para o Monitor de Máquinas Virtuais Xen (Virtual Machine Monitor - VMM). Este subsistema, chamado SRX, é capaz de balancear os recursos de um ambiente virtualizado a fim de atender Acordos de Níveis de Serviço. Os principais problemas do SRX estavam relacionados com a degradação de desempenho causada pela criação e destruição de processos que ocorria sempre que recursos eram realocados para uma VM. Nossa estratégia foi a de incluir as características do SRX a ferramenta Xen Master (XM). A fim de mostrar as melhorias de nossa nova ferramenta, um conjunto de testes é apresentado ao final do artigo.*

1. Introdução

Com a evolução do *hardware* computacional os recursos existentes nos equipamentos acabam sendo muitas vezes utilizados de forma ineficiente, pois os computadores possuem um poder computacional muito elevado e a grande maioria das aplicações existentes não consomem, a todo momento, toda capacidade disponibilizada.

Para contornar este problema podem-se utilizar ferramentas de virtualização. O uso da virtualização permite executar diversos Sistemas Operacionais (SOs) em um mesmo *hardware*, maximizando dessa forma a utilização do mesmo. Também apresentam benefícios em relação ao investimento em equipamentos, pois com o auxílio da virtualização é possível ter diversos serviços sendo executados, sendo que cada serviço pode estar executando em diferentes SOs. Cada um desses SOs trabalha de forma isolada entre si, embora compartilhem a mesma máquina física. Como consequência, se

[1] Study developed by the Research Group of the PDTI 001/2010, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91.

evita subutilização dos recursos computacionais, bem como economia de espaço físico e energia.

Por trazer diversas vantagens, esse tipo de tecnologia conquistou uma parte abrangente do mercado. Por isto, foram desenvolvidas diversas ferramentas de virtualização, como por exemplo o Xen [Barham et al. 2003] e o VmWare [Sugerman et al. 2001]. Apesar das grandes vantagens proporcionadas por essas ferramentas, diversos trabalhos têm proposto melhorias e novas funcionalidades.

Em [Rodrigues et al. 2006] é apresentado um subsistema de realocação de recursos, este que foi desenvolvido com o objetivo de otimizar a distribuição de recursos de forma justa para que cada ambiente virtual tenha ajustados os seus níveis de recursos (processador e memória), conforme as necessidades definidas em um acordo de nível de serviço (*Service Level Agreement* - SLA) da aplicação que hospeda [Chen et al. 2008]. Este subsistema foi denominado de Subsistema de Realocação Xen (SRX) [Rodrigues et al. 2006].

Este trabalho propõe um aprimoramento do subsistema de realocação de recursos de Máquinas Virtuais (*Virtual Machines* - VMs), onde o principal objetivo é diminuir o impacto no desempenho causado pelas constantes criações e destruições dos processos responsáveis pela realocação de recursos às VMs.

Para melhor compreensão do problema e das melhorias propostas, o trabalho está estruturado da seguinte forma. A Seção 2 apresenta o modelo do processo de realocação do SRX, suas deficiências e uma possível proposta para se contornar este problema. Na Seção 3 é descrita a implementação da proposta desse trabalho, que busca resolver as deficiências apresentadas. Na Seção 4 são descritos os testes efetuados e seus resultados, na Seção seguinte são apresentadas as conclusões e análise dos sistemas comparados.

2. Subsistema de Realocação Xen

A ferramenta SRX é um subsistema desenvolvido para o monitor de máquinas virtuais Xen e foi escrito na linguagem de programação C++ [Stroustrup and Hill 1996]. Ela é capaz de monitorar e realocar, em tempo de execução, os recursos ociosos de um ambiente virtualizado que pode conter um determinado número de VMs. Além disso, esta ferramenta é capaz de manter o nível de qualidade do serviço das aplicações hospedadas nas VMs, através da utilização de SLAs. Desta forma, é provável que as aplicações das VMs tenham seus SLAs respeitados, mesmo quando uma VM necessite de mais recursos do que aqueles que estão disponíveis. Esta funcionalidade é alcançada através da realocação de recursos entre as VMs. Para realizar o processo de realocação descrito, o SRX se utiliza primeiramente, dos resultados do processo de decomposição de SLA [Chen et al. 2007], realizado através de testes de desempenho sobre a estrutura virtualizada.

Este passo consiste na execução de *benchmark* [Saavedra and Smith 1996] sobre a aplicação hospedada nas máquinas virtuais. Os resultados obtidos nos testes são utilizados para mapear as métricas de baixo nível (processador e memória). Esses dados, em conjunto com o SLA das VMs, são utilizados pelo SRX para configurar automaticamente o nível de recursos que será atribuído a cada VM, no momento da sua inicialização ou para se definir um novo nível de recursos para uma VM, caso o seu respectivo SLA seja alterado após a sua inicialização (ver Figura 1).

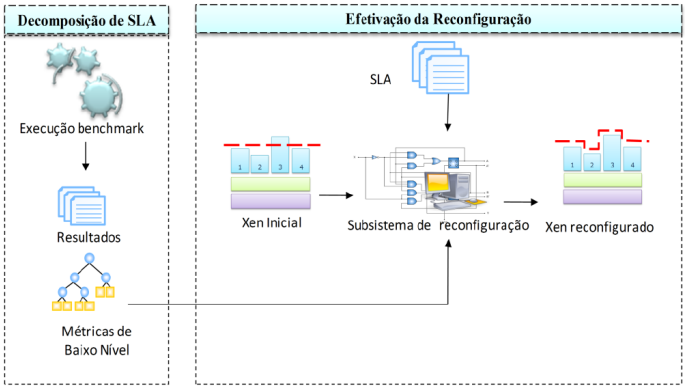


Figura 1. Processo de realocação de recursos [Rodrigues 2009]

2.1. Limitações da ferramenta SRX

O SRX se utiliza do Xen Master (XM) [Matthews et al. 2008] para executar de forma indireta as rotinas de realocação. Este é o gerenciador padrão de recursos do Xen e é o programa que executa diretamente as rotinas de realocação de recursos de memória e CPU para as VMs. Além disso, o XM possui muitas outras funcionalidades, ele é capaz de criar, destruir, aumentar e diminuir as fatias de CPU e memória das VMs, bem como informar as principais configurações de uma VM que esteja ativa. O XM apesar de ter muitas funcionalidades é bastante simples de se utilizar, podendo ser executado através de comando via terminal.

Como apresentado anteriormente, o SRX é escrito em C++ e externo ao XEN, sendo assim, não possui acesso direto às rotinas de gerenciamento de recursos. Portanto quando ele detecta que é necessária a realocação de recursos para uma determinada VM, ele cria um processo através da rotina *fork*, e esse processo, através do comando *exec*, será responsável por executar o programa XM, que contém e executa diretamente as rotinas de realocação. Após a realização da realocação, o processo do XM é destruído.

Durante a execução da ferramenta se observou que a constante criação e destruição do processo do XM causa uma carga excessiva e desnecessária ao sistema, principalmente em um cenário onde diversas realocações sucessivas são efetuadas. Além do tempo gasto para criação e destruição de processos, outra consequência é a degradação de desempenho causada por *trashing* na *cache* da CPU [Pimentel et al. 2000].

Com o intuito de contornar esse problema, buscou-se adicionar internamente ao XM as funcionalidades de monitoração e realocação providas pelo SRX. Dessa forma, a constante criação e destruição do processo do XM não ocorreria mais e com isso não haveria o problema de *trashing* na *cache* da CPU, ou seja, o principal objetivo é fazer com que as rotinas de realocação sejam executadas diretamente através da criação de um novo módulo para ser integrado ao XM, juntamente com as funcionalidades nativas como criação e destruição de VMs.

Esse módulo seria capaz de prover os recursos de monitoração e realocação providas pelo SRX e com isso a utilização de uma ferramenta externa não seria mais necessária, uma vez que o XM teria um módulo com as mesmas funcionalidades do SRX, com a vantagem que o processo do XM seria criado uma única vez. Através da execução de sua nova funcionalidade, seu novo módulo, seria um *daemon* que estaria constantemente em

execução desde o momento que o administrador do sistema o ativasse, através de comando via terminal, até o momento que o mesmo o desligasse.

3. Implementação de um módulo de monitoração e realocação para o XM

A implementação do novo módulo do XM foi baseada no código do SRX. Ele foi escrito na linguagem de programação Python [Lutz 2006], pois todos os módulos do XM estão escritos nessa linguagem, e foi incluído como uma nova função do XM, sendo simples sua utilização. Esse novo módulo foi nomeado de Realoca Xen (ReX).

Para o ReX nada foi acrescentado em termos de funcionalidade, se comparado com o SRX, sendo assim, o que se fez foi criar o módulo ReX analisando totalmente o código do SRX. Ao final dessa análise pôde-se então, iniciar a codificação do ReX.

A parte principal do código é o chamado módulo de realocação do ReX, este que é responsável por atribuir percentuais de processador e memória, que não estão sendo utilizados pelo sistema, às VMs que estejam precisando de maior quantidade de recursos, em determinado momento.

Para o ReX a criação do processo referente ao XM ocorre uma única vez e, acontece apenas no momento da execução do módulo do ReX por parte do administrador do sistema e não durante o processo de realocação, a exemplo do que ocorre para o SRX.

O processo de realocação para o ReX também ocorre de forma diferente da que ocorre para o SRX, para o ReX não há um processo que necessite ser criado e que em seguida executa o XM. No momento em que uma VM necessita de recursos, a rotina de realocação do ReX é executada sem a necessidade da criação do processo do XM, isso se deve pelo fato do ReX ser totalmente integrado ao XM e com isso, consegue executar diretamente as rotinas de realocação de CPU e memória. Desta forma, o processo referente ao XM estará constantemente em execução e só será "destruído" quando o administrador do sistema interromper a execução do ReX. Na Figura 2 é apresentado como o módulo de realocação do ReX executa as rotinas de realocação de recursos nativas do XM.

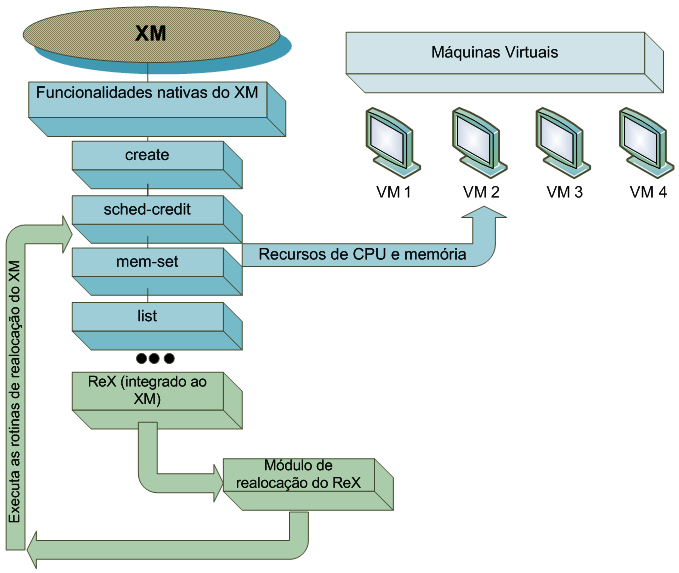


Figura 2. Processo de realocação para o ReX

Outra vantagem do ReX é que ele não precisa ser compilado, uma vez que ele é escrito em Python, uma linguagem totalmente interpretada. Outra funcionalidade é que ele pode ser executado de qualquer diretório, sendo que para executá-lo basta abrir um terminal e executar o comando: "xm ReX", com isso o XM é criado e começa a executar o módulo de monitoração e realocação do ReX, exibindo todas as configurações das VMs, percentuais de recursos utilizados por elas e avisos que indicam que para determinada VM estão sendo alocados recursos de CPU e memória.

4. Etapa de Testes

Sabendo que o sistema ReX é uma reimplementação com intuito de melhorar o desempenho do SRX, porém integrando as funcionalidades deste último ao XM, foram realizados testes para comparar o desempenho desses dois sistemas.

No decorrer da execução dos testes se observou que no quesito usabilidade o ReX mostrou-se indiscutivelmente superior ao SRX, pelo fato de ser uma chamada do XM, sendo assim transparente o seu uso, sem necessidade de compilação e importação de bibliotecas como no caso do seu antecessor.

Para executar os testes foi definido o SO Ubuntu 8.04 por fornecer maior compatibilidade e suporte com a versão atual do Xen, (versão 3.2).

4.1. Ambiente de Testes

Para executar os testes foram utilizadas duas estações de trabalho com processador *Pentium 4 HT* com frequência de *3.2 GHz* e *1.0 GB* de memória *RAM* padrão *DIMM 133 MHz*. A primeira estação de trabalho foi alocada para executar 4 VMs.

Pelo fato das estações de trabalho terem um número bastante limitado de memória, foi alocado para cada VM apenas 64 MB de memória RAM, deixando 64 MB reservados para as realocações de memória (memória livre) efetuadas pelo escalonador. O restante da memória foi reservado para o Domínio 0. Já o processador disponível que utiliza tecnologia HT [Bulpin and Pratt 2005] é reconhecido como dois processadores físicos pelo SO e, portanto foi configurado da seguinte forma, um processador para o Domínio 0 e um processador para os demais domínios, ou seja, demais VMs. Dessa forma, foi definido que inicialmente cada VM disponibilizaria de 20% de fatia de processamento da CPU, os 20% restantes ficaram reservados para o processo de realocação de CPU às VMs.

Em cada VM foi instalado o *software* Apache Tomcat [Tomcat 2010], este que entre suas funcionalidades é um servidor de páginas *Web*, contendo alguns *templates* de página que executam variadas operações, dentre elas Java script [Flanagan 2006], applets [Harold 2000] ou até mesmo HTML simples [Willard 2009].

A segunda estação de trabalho foi alocada como cliente, para isto foi instalado o *software* para teste de desempenho Jmeter [JMeter 2010].

O Jmeter é um *software* que tem a capacidade de gerar requisições *web* automaticamente, simulando o comportamento de um número de usuários que pode ser determinado. Assim ele cria um conjunto de *threads*, cada uma simulando um usuário que faz requisições aos endereços de páginas que foram configurados previamente. A Figura 3 exemplifica o ambiente utilizado.

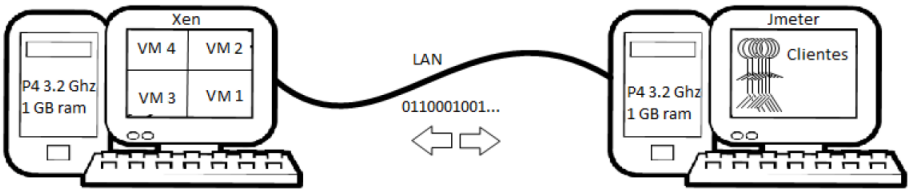


Figura 3. Estrutura de testes

4.2. Experimento 1

A estratégia de testes consistiu em definir páginas do servidor Tomcat que disponibilizassem de aplicações distintas entre si, ou seja, procurou-se por páginas com tecnologias diferentes (HTML, javascript, applets). Com a lista de páginas a serem acessadas configurou-se quatro instâncias do Jmeter para solicitar requisições a estas páginas. Desta forma tendo uma instância para cada VM da estação de trabalho servidora. Neste ponto foram iniciados testes preliminares com o objetivo de indicar quantos usuários cada instância do Jmeter deveria conter de forma a não deixar o sistema no limite de carga. Pelo fato da máquina servidora possuir um *hardware* limitado, foram necessários apenas 3 usuários em cada instância para ocupar os recursos do sistema de forma mediana.

O próximo passo foi fazer a decomposição de um SLA referente ao tempo de respostas aos clientes, o qual se mediu 0,9 segundos para uma carga de 40 clientes para uma única VM (VM 2). Com estes dados foi possível iniciar os testes do escalonador ReX, onde foi aumentado o número de clientes da VM 2, a fim de fazê-la necessitar de recursos extras do sistema para o cumprimento do SLA. Neste ponto o Jmeter simulava 3 clientes para cada VM, exceto para a VM 2, onde foram simulados 80 clientes. A Figura 4 exemplifica a configuração descrita.

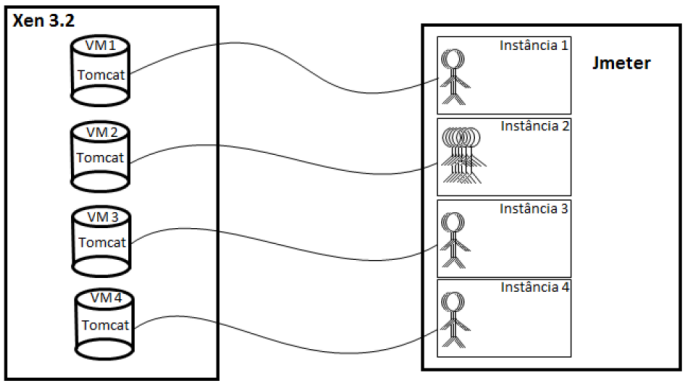


Figura 4. Estrutura de testes detalhada

Para proporcionar confiabilidade nos resultados cada teste foi repetido 28 vezes para cada sistema (ReX e SRX). Desta forma foi possível a realização de cálculos estatísticos que oferecem maior relevância aos resultados obtidos. Este processo foi bastante oneroso em relação ao tempo, pois cada teste precisou ser iniciado manualmente e por vezes o Jmeter travou, necessitando que o teste fosse reiniciado.

Após concluir os testes com o escalonador ReX, foram feitos os testes com o escalonador SRX. Nestes testes foram utilizadas as mesmas configurações de ambiente para garantir a validade dos resultados a serem comparados.

É importante salientar que cada vez que um teste era executado, tanto o ReX quanto o SRX executavam o processo de realocação não mais do que duas vezes até o final de cada teste.

4.3. Experimento 2

Nesta etapa os testes foram feitos de maneira mais simples, tendo o objetivo de apontar quanto tempo cada um dos sistemas comparados utiliza para efetuar uma realocação de recursos. Para isto inserimos *patches* no ReX e no SRX, com a finalidade de calcular o tempo das rotinas de realocação de cada um.

A implementação dos *patches* foi bastante simples, ou seja, apenas recuperou-se o tempo do sistema imediatamente antes e imediatamente depois do procedimento de realocação. Obtendo a diferença entre esses dois tempos geraram-se os resultados. É importante salientar que para o experimento 2, também foram executados 28 testes para cada sistema (ReX e SRX). Ao final se obteve um total de 112 execuções de testes, sendo que 56 testes foram executados para o experimento 1 e os outros 56 para o experimento 2.

4.4. Resultados dos testes para o experimento 1

Os resultados obtidos no experimento 1 mostraram que o ReX obteve um pequeno ganho em relação ao SRX. Entretanto os dois sistemas respeitaram o SLA de 0,9 segundos proposto inicialmente. A Figura 5 apresenta a média do tempo de respostas em cada VM para o ReX. A Figura 6 apresenta os mesmos resultados para o SRX.

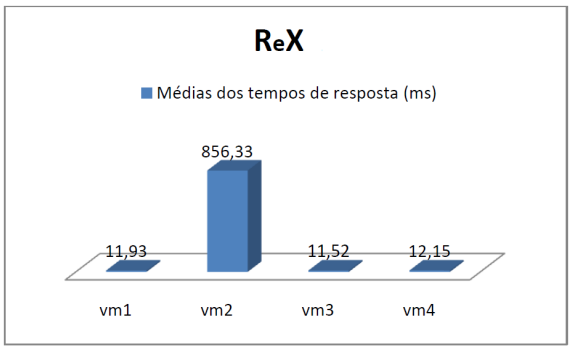


Figura 5. Carga no sistema ReX

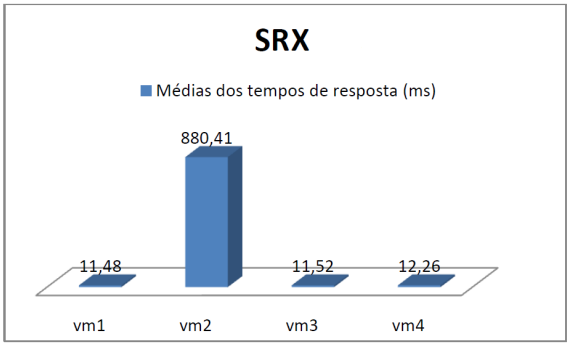


Figura 6. Carga no sistema SRX

As Figuras 5 e 6 apresentam claramente a diferença de carga utilizada nos testes entre a VM 2 e as demais. Pode-se perceber também, que a média do SRX foi um pouco maior, ela foi superior em 24,08 ms se comparado com a média de ReX. Esta diferença deve-se à criação do processo XM para cada realocação, processo este, que não ocorre no ReX. A Tabela 1 apresenta os dados estatísticos referentes à VM 2, esta que foi estressada durante os testes.

Tabela 1. Dados Estatísticos para o experimento 1

Sistema avaliado	VM 2 média (ms)	VM 2 mediana (ms)	VM 2 desvio padrão (ms)	Número de amostras
ReX	856,33	869	105,23	28
SRX	880,41	915	96,49	28

Utilizando os dados da Tabela 1 foram construídos os gráficos de distribuição de *Gauss* [Kim et al. 2008] para análise dos resultados, onde foi definido um intervalo de confiança de 95%. A Figura 7 apresenta os resultados para o ReX, apresentando um intervalo de confiança entre 815,6 ms e 897,06 ms. Enquanto que a Figura 8 apresenta os resultados para o SRX com intervalo de confiança entre 843,06 ms e 917,75 ms.

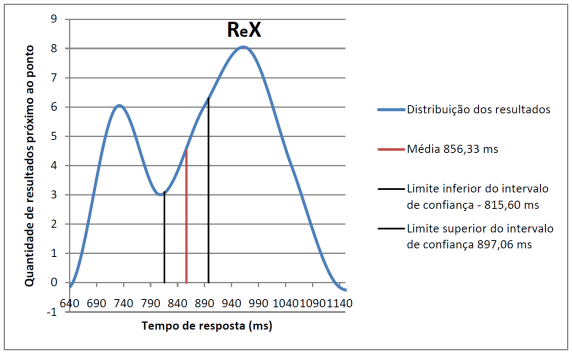


Figura 7. Análise do intervalo de confiança de ReX para o experimento 1

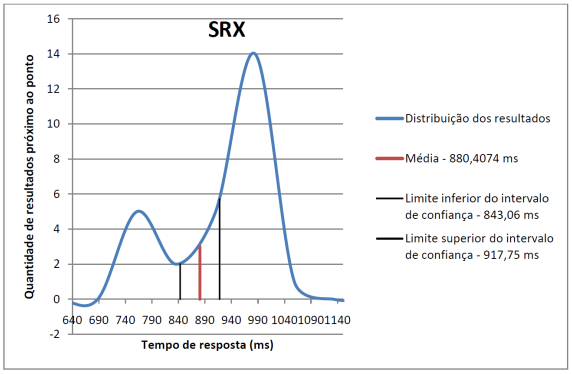


Figura 8. Análise do intervalo de confiança de SRX para o experimento 1

Ao comparar os resultados apresentados nos gráficos, pode-se perceber que o ReX tem um comportamento mais constante do que o SRX, pois conforme a carga do sistema

aumenta, os tempos de resposta do SRX sobem, isso ocorre pelo fato de ele demorar mais para iniciar a realocação, pois ele necessita criar o processo XM e esta operação precisa disputar o sistema juntamente com os demais processos do SO. Assim, quanto mais carga no sistema, mais tempo leva para a criação do processo do XM e consequentemente a realocação de recursos demora mais. Isto faz com que a VM trabalhe sobrecarregada durante mais tempo, a consequência é o aumento da média do tempo de resposta das requisições. Já no caso do ReX o processo XM já está criado e apenas necessita ganhar os recursos do sistema para efetuar a realocação de recursos, e assim a VM passa menos tempo trabalhando no limite e responde ao aumento de requisições mais rapidamente.

4.5. Resultados dos testes para o experimento 2

Os resultados obtidos com o experimento 2 confirmaram os resultados do experimento 1. Onde o ReX obteve maior vantagem quando a carga do sistema aumentava, evidenciando os motivos analisados no experimento 1.

A Tabela 2 a seguir apresenta os dados estatísticos coletados neste teste. Com estes dados, a exemplo do experimento 1, também foram construídos gráficos de distribuição de *Gauss* para análise dos resultados, juntamente com a definição dos intervalos de confiança para os testes de cada sistema.

Tabela 2. Dados Estatísticos para o experimento 2

Dados	ReX	SRX
Média (ms)	172,69	180,18
Mediana (ms)	177,63	172,80
Desvio Padrão (ms)	19,40	19,65
Número de amostras	28	28

A Tabela 2 apresenta o ganho médio de tempo, obtido na execução da rotina de realocação do ReX em relação ao SRX. A análise destes resultados é apresentada nas Figuras 9 (ReX) e 10 (SRX). As duas Figuras foram construídas com intervalo de confiança de 95%. O gráfico referente ao ReX possui um intervalo de confiança variando entre 164,21 ms e 178,34 ms, para o SRX o intervalo varia entre 171,75 ms e 186,78 ms.

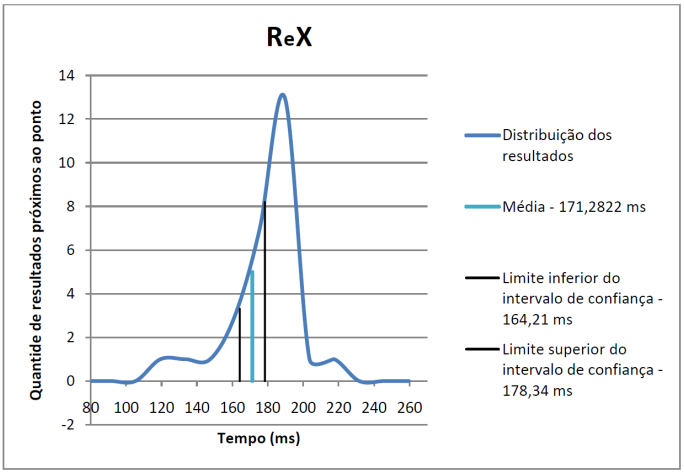


Figura 9. Análise do intervalo de confiança de ReX para o experimento 2

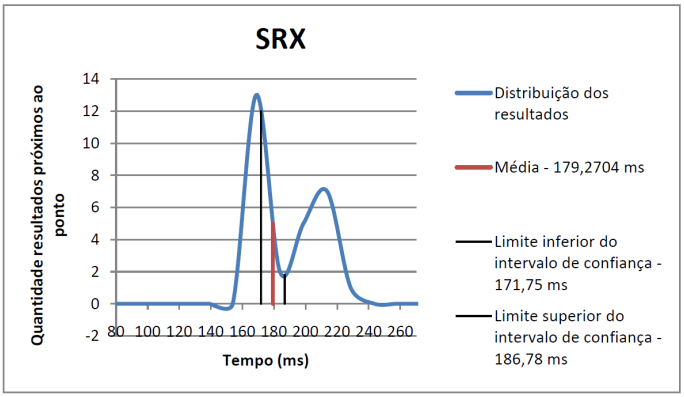
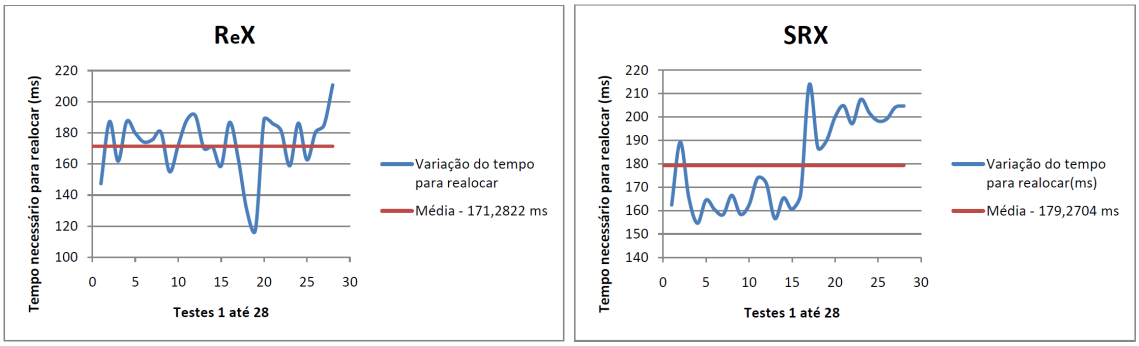


Figura 10. Análise do intervalo de confiança de SRX para o experimento 2

A última análise realizada para o experimento 2 foi para verificar a variação do tempo de execução das rotinas de realocação. As Figuras 11a (para o ReX) e 11b (para o SRX) apresentam os dados referentes a esta análise. Estes gráficos apresentam o tempo de resposta relativo a cada um dos 28 testes. Eles demonstraram que o ReX tem comportamento mais constante, ou seja, analisando as curvas de variação do tempo para realocar, pode-se perceber que no caso do ReX a variação fica mais próxima da reta da média, enquanto o SRX mantém-se mais distante da reta da média.



(a) **(b)**

Figura 11. Análise de comportamento por teste executado

5. Conclusão

Este trabalho apresentou uma melhoria efetuada sobre uma ferramenta capaz de realizar realocação de recursos em ambientes virtualizados, de modo a respeitar acordos de níveis de serviço. Para efetuar a realização dessa melhoria destacam-se quatro etapas como sendo as principais para a efetivação do trabalho. A primeira consistiu na análise do comportamento da ferramenta SRX e como consequência a descoberta das criações e destruições desnecessárias dos processos responsáveis pela realocação de recursos às VMs que necessitam, em determinado momento, de maior quantidade desses recursos. Na segunda etapa foi realizado um estudo sobre o XM, este que é o gerenciador padrão de recursos do virtualizador Xen e, tem acesso direto as rotinas responsáveis pelo processo de realocação de recursos. A terceira foi a etapa de análise do código da ferramenta SRX e a implementação de um módulo (ReX) para ser integrado ao XM, módulo este que contém

as mesmas funcionalidades do SRX, porém tem acesso direto as rotinas de realocação de recursos do ambiente e como consequência evita as criações desnecessárias dos processos de realocação. Na quarta etapa foi realizado o processo de testes sobre o SRX e ReX e em seguida uma comparação dos resultados obtidos para as duas implementações.

Os resultados demonstraram que para ambos experimentos os resultados foram satisfatórios. Pois nos dois métodos utilizados para testar o sistema obtivemos o mesmo comportamento e assim um teste serviu de prova para o outro. A diferença de tempo entre os dois sistemas foi baixa pelo fato de ocorrerem poucas realocações durante os testes, porém mesmo com poucas realocações o ReX obteve vantagem. É possível então concluir que em um ambiente real, onde diversas realocações são realizadas, as diferenças entre os dois sistemas tendem a aumentar, pois nos testes realizados ocorreram poucas realocações e mesmo assim evidenciaram diferença nos tempos.

O ReX também leva vantagem na questão da usabilidade, pois ele não necessita importar bibliotecas e nem ser compilado. Também foi possível comprovar que a criação e destruição frequente do processo XM acarretava em uma carga bastante elevada, pois apesar de o ReX ser escrito em uma linguagem interpretada, teve um ganho sobre o SRX que é compilado em C++.

6. Agradecimentos

Elder M. Rodrigues e Avelino F. Zorzo por disponibilizarem acesso a ferramenta SRX, indispensável para a realização do trabalho. Avelino F. Zorzo possui bolsa de produtividade CNPQ/Brasil e CAPES/PROCAD. Elder M. Rodrigues possui bolsa CAPES e é pesquisador integrante do INCT-SEC.

Referências

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the Art of Virtualization. In *SOSP*, pages 164–177.

Bulpin, J. R. and Pratt, I. A. (2005). Multiprogramming Performance of the Pentium 4 with Hyper-Threading. *USENIX Annual Tech. Conf.*

Chen, Y., Iyer, S., Liu, X., Milojicic, D., and Sahai, A. (2007). SLA Decomposition: Translating Service Level Objectives to System Level Thresholds. *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*.

Chen, Y., Iyer, S., Liu, X., Milojicic, D., and Sahai, A. (2008). Translating Service Level Objectives to lower level policies for multi-tier services. *Cluster Computing*, 11(3):299–311.

Flanagan, D. (2006). *The Definitive Guide, 5th edition*. O'Reilly.

Harold, R. (2000). *Java Network Programming, 2nd edition*. O'Reilly & Associates.

JMeter, A. (2010). Apache JMeter. *http://jakarta.apache.org/jmeter/*. Acesso em 26 de fev. 2010.

Kim, J.-H., Park, J.-H., and Kang, D.-J. (2008). Method to improve the performance of the AdaBoost algorithm using Gaussian probability distribution. *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 1749–1752.

Lutz, M. (2006). *Programming Python*. O'Reilly Media.

Matthews, J. N., Dow, E. M., Deshane, T., Hu, W., Bongio, J., Wilbur, P. F., and Johnson, B. (2008). *Running Xen: A Hands-On Guide to the Art of Virtualization, 1st edition*. Prentice Hall PTR.

Pimentel, A., Hertzberger, L., Struik, P., and Van Der Wolf, P. (2000). Hardware versus Hybrid Data Prefetching in Multimedia Processors: A Case Study. *Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International*, pages 525–531.

Rodrigues, E. M. (2009). Realocação de Recursos em Ambientes Virtualizados. Master's thesis, Dissertação (Mestrado em Ciência da Computação) - Faculdade de Informática - PUCRS, Porto Alegre.

Rodrigues, E. M., Zorzo, A. F., Oliveira, F. M. d., and Costa, L. T. (2006). Reconfiguração de ambientes virtualizados através do uso de Teste Baseado em Modelos e SLAs. *WSO*.

Saavedra, R. H. and Smith, A. J. (1996). Analysis of benchmark characteristics and benchmark performance prediction. 14(4):344–384.

Stroustrup, B. and Hill, M. (1996). A history of C++: 1979–1991. pages 699–769.

Sugerman, J., Venkitachalam, G., and Lim, B. (2001). Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. *USENIX Annual Tech. Conf.*, pages 2–15.

Tomcat, A. (2010). Apache Tomcat. *http://tomcat.apache.org*. Acesso em 15 de jan. 2010.

Willard, W. (2009). *HTML A Beginner's Guide, 4th edition*. McGraw-Hill Osborne Media.