UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Pós-Graduação em Ciência da Computação

Leonardo José de Lima

Proposta para aumento da escalabilidade do sistema WSE-OS por meio do escalonamento de conexões e gerenciamento da replicação de dados dos servidores

UNESP
2013

Lima, Leonardo José de.
Proposta para aumento da escalabilidade do sistema WSE-OS por meio do escalonamento de conexões e gerenciamento da replicação de dados dos servidores / Leonardo José de Lima. -- São José do Rio Preto, 2013
82 f. : il., gráfs., tabs.

Orientador: Roberta Spolon
Dissertação (mestrado) ï Universidade Estadual Paulista ñJúlio de Mesquita Filhoò, Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Sistemas operacionais distribuídos (Computadores) 3. Sistema de computação virtual. 4. Sistemas de computação sem fio. 5. Redes de computadores ï Escalabilidade. I. Spolon, Roberta. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU ï 681.3

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Campus de São José do Rio Preto

Leonardo José de Lima

Proposta para aumento da escalabilidade do sistema WSE-OS por meio do escalonamento de conexões e gerenciamento da replicação de dados dos servidores

Orientadora: Profª Drª Roberta Spolon

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação - Área de Concentração em Sistemas de Computação, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

UNESP
2013

Leonardo José de Lima

Proposta para aumento da escalabilidade do sistema WSE-OS por meio do escalonamento de conexões e gerenciamento da replicação de dados dos servidores

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação - Área de Concentração em Sistemas de Computação, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Profa. Dra. Roberta Spolon
Professora Adjunta
DCo/FC UNESP - Bauru
Orientadora

Prof. Dr. José Remo Ferreira Brega
Professor Adjunto
DCo/FC UNESP - Bauru

Profa. Dra. Sarita Mazzini Bruschi
Professora Doutora
SSC/ICMC USP - São Carlos

UNESP
2013

AGRADECIMENTOS

A Deus, por me dar forças e determinação para concluir este trabalho, mesmo quando cheguei a acreditar que não era mais possível.

A professora Roberta Spolon, minha orientadora, pela enorme paciência tida comigo enquanto eu desenvolvia este trabalho.

Aos professores Renata Spolon Lobato e Marcos A. Cavenaghi, que ajudaram muito durante o desenvolvimento deste trabalho.

A professora Tânia Nadaleto, pelo grande auxílio para o primeiro passo rumo ao Mestrado.

Aos meus familiares que sempre acreditaram, se orgulharam e incentivaram na conclusão deste.

Ao amigo Takao Matsumura, pela grande ajuda técnica dada durante o curso.

Aos meus amigos Dirceu Marcos Dolce Furlanetto e Patrícia Rodrigues Urbano que me deram forças, puxões de orelha e acreditaram na conclusão deste.

A CAPES pelo apoio financeiro.

A UNESP e ao Departamento de Computação da Faculdade de Ciências de Bauru, que proporcionaram recursos para realização de experimentos deste projeto.

"Where the willingness is great,
the difficulties cannot be great."
Niccolò Machiavelli

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ACL: *Access Control List*
Bash: *Bourne-Again SHell*
CD: *Compact Disc*
Conit: *Consistency Unit*
DDR2: *Double Data Rating 2*
DDR3: *Double Data Rating 3*
DHCP: *Dynamic Host Configuration Protocol*
DVD: *Digital Versatile Disc*
FC13: *Fedora Core 13*
Gb: *Gigabit*
GB: *Gigabyte*
GNOME: *GNU Network Object Model Enviroment*
HD: *Hard Drive*
HTML: *Hyper Text Markup Language*
IEEE: *Institute of Electrical and Electronics Engineers*
IP: *Internet Protocol*
ISU: *Imagem de Sistema Única*
Kb: *Kilobit*
KB*: Kilobyte*
LAN: *Local Area Network*
Mb: *Megabit*
MB: *Megabyte*
ModEsc: Módulo de Escalonamento de Conexões
ModRep: Módulo de Replicação de Dados
MV: Máquina Virtual
PC: *Personal Computer*
PHP: *Hypertext Preprocessor*
RAM: *Random-Access Memory*
RPC: *Remote Procedure Call*
SD: *Secure Digital*
SO: *Sistema Operacional*

SSD: *Solid-State Drive*

SSH: *Secure Shell*

TB: *Terabyte*

TCP: *Transmission Control Protocol*

TCSC: *Thin Client Server Computing*

TMR: *Triple Modular Redundancy*

USB: *Universal Serial Bus*

UTP: *Unshielded Twisted Pair*

VDI: *Virtual Desktop Infrastructure*

VM: *Virtual Machine*

WiMAX: *Worldwide Interoperability for Microwave Access*

WSE-OS: *Wireless Sharing Enviroment – Operating Systems*

RESUMO

Devido a queda gradual no custo de aquisição de novos computadores, há cada vez mais dispositivos computacionais adentrando o mercado. A grande quantidade de novos dispositivos gera heterogeneidade entre eles e esta dificulta a administração de ambientes computacionais, pois é necessário manter os sistemas funcionando em compatibilidade com dispositivos bastante distintos simultaneamente. O sistema WSE-OS propõe uma solução de centralização de dados e recursos que aborda o problema da heterogeneidade de maneira eficaz. Fazendo uso da tecnologia *wireless* a ferramenta WSE-OS utiliza uma estrutura *Thin Client* que permite aos seus clientes executarem instanciações de sistemas operacionais virtualizados armazenados no servidor. Este trabalho apresenta uma proposta que altera a estrutura do WSE-OS incluindo a capacidade de operar com múltiplos servidores, tendo como objetivo aumentar a escalabilidade, disponibilidade e confiabilidade da ferramenta por meio de técnicas de replicação do servidor e escalonamento das conexões. A replicação de dados consiste em detectar as alterações sofridas nos dados contidos em um determinado servidor e transmiti-las aos demais priorizando a consistência entre as réplicas. O escalonamento de conexões funciona ativamente distribuindo os clientes dentre os servidores para melhorar o desempenho da ferramenta.

Palavras-Chaves: Virtualização, Replicação de dados, Sistemas Distribuídos, WSE-OS, Wireless , Escalonamento de Conexões

ABSTRACT

Due to a gradual decrease in the cost of purchasing new computers, there is more and more computing devices entering the market. The large quantity of new devices creates heterogeneity among them and this complicates the administration of computing environments, because is necessary to keep the systems running in compatibility with quite different devices simultaneously. The WSE-OS system proposes a solution for centralizing data and resources that addresses this problem effectively. Using wireless networking technology, the WSE-OS tool uses a Thin Client structure that allows its clients to execute instantiations of virtualized operating systems stored on the server. This paper presents a proposal that changes WSE-OS's structure including the ability to run with multiple servers, having as its goal increase scalability, availability and reliability through server's data replication and staggering of connections. Data replication consists in detecting changes on data from a given server and transmit it to the others prioritizing the consistency among replicas. The staggering of connections works on actively distributing the clients among servers to improve system's performance.

Keywords: Virtualization, Data Replication, Distributed Systems, WSE-OS, Wireless, Connection Staggering

1 INTRODUÇÃO

Esta seção apresenta o escopo e a motivação para o trabalho proposto nessa dissertação. A seção 1.1 mostra os contextos nos quais o trabalho está inserido. Na seção 1.2 são apresentados os motivos para o desenvolvimento deste trabalho. Na seção 1.3 são apresentados os objetivos almejados por este trabalho e a seção 1.4 apresenta a estrutura deste trabalho.

1.1 ESCOPO

Com o constante desenvolvimento da Computação, novos dispositivos adentram e deixam o mercado diariamente, tornando assim complexo manter uma homogeneidade entre eles. A heterogeneidade dos dispositivos computacionais torna mais difícil a administração destes em um determinado ambiente computacional.

Formas mais eficientes para aproveitar todos os recursos disponíveis ou ampliar as capacidades dos dispositivos existentes são necessárias. A virtualização aparece com a finalidade de obter o máximo proveito do *hardware* já existente sem a necessidade de grandes investimentos.

A virtualização é uma técnica computacional em evidência no mercado atual, tanto para empresas quanto para usuários comuns. Sua utilização é recomendada em diversos casos, e tende a ser cada vez mais, já que há constante pesquisa e evolução dessa tecnologia, tanto no *software* quanto no *hardware*.

Diante da grande oferta de dispositivos heterogêneos e redes sem-fio, é possível aliar esses recursos à virtualização e outras tecnologias, permitindo que novos conceitos sejam criados.

O sistema WSE-OS (*Wireless Sharing Environment - Operating Systems*) (DIGIERE, 2011)(CREPALDI, 2011)(CREPALDI et al., 2011)

combina recursos de virtualização, redes sem-fio e de acesso remoto para proporcionar uma experiência inovadora e eficiente aos seus usuários.

O WSE-OS utiliza uma arquitetura cliente-servidor otimizada para redes sem-fio com o intuito de centralizar dados e recursos num único servidor oferecendo novas funcionalidades aos dispositivos conectados.

A centralização proposta possui pontos fortes e fracos que podem ser explorados e otimizados. A centralização facilita a administração do ambiente, mas pode sobrecarregar o ponto centralizador, além de oferecer riscos à integridade dos dados que estão sujeitos a falhas físicas.

Como solução aos pontos fracos, pode-se fazer a distribuição do processamento, mantendo a centralização de recursos, alcançando tolerância a falhas através da replicação de dados.

1.2 MOTIVAÇÃO

O número de dispositivos capazes de operar em redes sem-fio aumenta a cada dia. Cada um possui suas peculiaridades e características, mas todos têm o objetivo de se conectarem a alguma rede.

A grande velocidade em que novos dispositivos são lançados causa heterogeneidade e dificuldade na administração desses dispositivos. Há uma necessidade de interconexão dos dispositivos em uma rede. Essa conexão deve ser de maneira rápida e fácil, respeitando as características de cada um.

A implementação de uma ferramenta na forma de um *middleware* inicializado a partir de mídias secundárias (USB e SD) aumenta consideravelmente a quantidade de computadores aptos a utilizar tal solução proposta pelo WSE-OS.

Com a facilidade de utilização por maior parte dos dispositivos computacionais existentes surge uma preocupação quanto a escalabilidade e confiabilidade dessa ferramenta.

Por meio da ampliação da capacidade de operação da ferramenta, maior disponibilidade, escalabilidade e confiabilidade podem ser agregadas com o uso de replicação de dados e escalonamento das conexões.

1.3 OBJETIVOS

O objetivo deste trabalho é ampliar e melhorar a ferramenta WSE-OS de Crepaldi et al. (2011) por meio de alterações na estrutura, implementação de novas funcionalidades e aprimoramento das funcionalidades existentes.

A atual estrutura do WSE-OS não suporta o funcionamento com mais de um servidor, ficando limitada à capacidade do único servidor existente. A centralização dos recursos em um único dispositivo físico coloca em risco a integridade dos dados armazenados, tornando necessária a criação de algum mecanismo que aumente segurança e permita que a ferramenta se expanda.

Esta dissertação tem por objetivos específicos para o WSE-OS:

- Desenvolvimento de uma estrutura capaz de suportar *n* servidores simultâneos;
- Desenvolvimento uma nova interface para o cliente que seja mais dinâmica e fácil de usar;
- Desenvolvimento do Módulo de Escalonamento de Conexões que permita ao usuário se conectar ao servidor com melhor desempenho no instante da conexão;
- Desenvolvimento do Módulo de Replicação que gerencie as réplicas entre os servidores mantendo a consistência entre elas;
- Realização de experimentos que avaliem o desempenho da ferramenta desenvolvida.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em sete capítulos e os assuntos abordados nesta monografia estão divididos da seguinte forma:

Capítulo 1: Contem a contextualização do escopo de pesquisa, ilustrando as principais motivações para o desenvolvimento do trabalho proposto e os objetivos pretendidos.

Capítulo 2: Descreve o conceito de Replicação de dados, explicando a importância de sua utilização em sistemas distribuídos. Além disto, descreve e compara os principais modelos de consistência existentes para replicação.

Capítulo 3: Descreve o conceito de Tolerância a Falhas, explicando os conceitos básicos, os tipos de falhas e os modelos de falhas. Além disso, mostra as propriedades de um sistema tolerante a falhas, as técnicas de redundância, as falhas na comunicação e a recuperação de falhas.

Capítulo 4: Apresenta a ferramenta WSE-OS descrevendo sua estrutura, o funcionamento dos seus principais elementos e a utilização da ferramenta.

Capítulo 5: Descreve o funcionamento da ferramenta desenvolvida, denominada WSE-OS Escalável, detalhando o funcionamento de seus módulos novos e as técnicas empregadas nos seus funcionamentos.

Capítulo 6: Apresenta a análise dos resultados obtidos dos testes realizados no WSE-OS Escalável.

Capítulo 7: Conclui a dissertação apresentando as considerações do trabalho e indicando futuras contribuições e continuidade do projeto com base nos estudos desenvolvidos a partir do WSE-OS Escalável.

2 REPLICAÇÃO DE DADOS

Segundo Lamport (1987), um sistema distribuído é aquele no qual a falha de um computador que o usuário nem sabia da existência é capaz de tornar seu próprio computador inoperante.

Coulouris et al. (2011) definem que um sistema distribuído é aquele no qual os componentes de *hardware* e *software* localizados em uma rede de computadores comunicam e coordenam suas ações pela troca de mensagens.

Dada a sua natureza, sistemas distribuídos falham, assim como qualquer outro sistema. Contudo, cabe ao desenvolvedor utilizar técnicas que diminuam ou erradiquem os prejuízos causados por essas falhas. Sendo assim, o uso de replicação de dados surge como importante técnica para solucionar tal problema.

2.1 CONSIDERAÇÕES INICIAIS

Segundo Colouris et al. (2011), todos os sistemas computacionais podem falhar.

As falhas ocorrem em diversos elementos que compõem esse sistema: erro humano, falhas na programação de determinado *software*, sobrecarga dos meios de comunicação, ataques por códigos maliciosos, falha no *hardware* ou quedas de energia (COULOURIS, 2011).

Todos esses elementos podem desativar partes do sistema e devido ao alto risco de falhas, tem-se a necessidade da replicação de dados. Dados são comumente replicados com o objetivo de aumentar a confiabilidade e/ou melhorar o desempenho do sistema.

Com a replicação dos dados surge a necessidade de manter a consistência das réplicas. Réplicas consistentes são aquelas que quando uma cópia é atualizada há a garantia que as demais também sejam atualizadas.

2.2 A IMPORTÂNCIA DA REPLICAÇÃO DE DADOS

A replicação de dados é utilizada por dois motivos principais. Ambos são de extrema importância para o funcionamento ideal de um sistema distribuído. A primeira razão é a confiabilidade e a segunda razão é o desempenho.

2.2.1 Confiabilidade

É de extrema importância que os dados contidos num sistema distribuído sejam preservados da maneira correta assim como em qualquer outro tipo de sistema. Cada dia mais, as pessoas tornam-se dependentes de sistemas computadorizados e a perda de dados em um desses sistemas pode causar grandes transtornos.

Um problema em pequena escala seria perder as imagens contidas em um álbum de fotos virtual, simplesmente porque o servidor responsável por armazenar as imagens teve um de seus discos rígidos, que continha seus dados, inutilizado por uma descarga elétrica.

Um problema em grande escala seria um banco perder todas as informações de seus clientes por uma falha humana de um dos seus operadores. Tal problema causaria transtorno a milhões de usuários e à empresa.

Ken Arnold afirma em Venners (2002) que ao desenvolver um sistema distribuído, há de se pensar que "falhas ocorrem o tempo todo".

A replicação de dados é empregada para a obtenção de maior confiabilidade em sistemas distribuídos. A replicação por confiabilidade consiste em criar e gerenciar réplicas dos dados em outros computadores dessa rede. Cada tipo de sistema deve conter algoritmos apropriados para o monitoramento em busca de falhas. Quando ocorre uma falha o sistema deve

ser capaz de restaurar a réplica criada no instante mais próximo e anterior à falha.

O gerenciamento, criação e atualização das réplicas são de extrema importância para se manter um sistema distribuído eficiente e confiável. A freqüência com que as réplicas são atualizadas é primordial para que a menor quantidade possível de dados seja perdida. Em contrapartida, quanto maior for o número de atualizações maior será a sobrecarga na rede devido ao fluxo desses dados (TANENBAUM & STEEN, 2006).

O dilema para se manter uma replicação confiável consiste em balancear o intervalo de atualização das réplicas com a capacidade de transmissão da rede.

2.2.2 Disponibilidade

Partindo do princípio de que um sistema distribuído não tem limitações geográficas e que seus elementos podem estar na mesma sala ou em diferentes continentes e que a latência de acesso a determinado recurso deva ser a menor possível, a replicação de dados é empregada para melhorar a disponibilidade e o desempenho desse sistema.

Por exemplo, um sistema de leilão online contendo um contador de tempo regressivo no qual pessoas do mundo todo tentam ao mesmo tempo arrematar um determinado produto deve ser o mais imparcial possível para que o leilão seja justo com todos os seus usuários. Por razões físicas, um usuário situado no mesmo bairro que o servidor contendo esses dados teria maiores chances de arrematar o produto do que outro situado no ponto mais distante do planeta em relação a esse servidor. Mesmo que ambos dessem o comando para arrematar o produto no mesmo exato instante, considerando que ambos possuem a mesma velocidade de acesso à Internet e o mesmo desempenho em seus computadores. O mais próximo fisicamente leva vantagem. No exemplo, a importância na replicação dos dados para aumentar a

disponibilidade e o desempenho de um sistema é uma excelente solução a fim de otimizar as chances de cada usuário (TANENBAUM & STEEN, 2006).

A replicação por disponibilidade consiste em criar réplicas e disponibilizar essas réplicas em diferentes localizações geográficas. Quanto melhor for a distribuição das réplicas, melhor será o desempenho para os usuários. Por outro lado, com o aumento do número de réplicas há também o aumento no trabalho a ser feito pelo gerenciador de réplicas para atualizá-las. Esse aumento de trabalho acarreta em maior fluxo de informações na rede, podendo sobrecarregá-la e diminuir a eficiência do sistema.

A distribuição de réplicas normalmente ocorre por dois principais motivos, a distância física entre o usuário e a réplica e quando um servidor está sobrecarregado de trabalho. O sistema pode contar com um algoritmo de balanceamento de cargas que será o responsável pela detecção de sobrecargas e distribuição das réplicas de acordo com a necessidade do sistema. O objetivo é sempre manter o melhor desempenho possível.

Para haver o melhor ponto de equilíbrio na distribuição das réplicas devem ser levados em consideração dois itens: a necessidade do sistema e de seus usuários e a capacidade de transmissão de dados da rede.

2.3 CONSISTÊNCIA

Consistência, na replicação de dados, é o termo usado para definir que as réplicas de um sistema distribuído são idênticas.

Segundo Tanenbaum & Steen (2006), uma coleção de cópias é consistente quando as cópias são sempre idênticas. Isto significa que uma operação de leitura realizada numa determinada cópia irá sempre retornar o mesmo resultado. Subsequentemente, quando uma cópia for atualizada, a atualização deve se propagar a todas as demais cópias antes que qualquer outra operação seja realizada.

Dada a natureza de um sistema distribuído, seus usuários estão em localidades diferentes e justamente para garantir maior “disponibilidade” desse

sistema, os dados também devem estar. Quando uma atualização ocorre no sistema, é de extrema importância que todas as cópias sejam atualizadas também.

Réplicas sofrem atualizações com maior ou menor intensidade, baseadas nas políticas do sistema distribuído. A atualização ocorre em uma determinada réplica e deve se propagar a todas as cópias no menor tempo possível, em um sistema ideal.

Diferentes políticas (ou protocolos) de replicação fornecem diferentes níveis de consistência a um sistema. Durante o desenvolvimento de um sistema distribuído é necessário que se use diferentes níveis de consistência para os diferentes tipos de processo do sistema ou que se use um mecanismo de replicação com consistência forte o bastante para todas as aplicações (AL-EKRAM & HOLT, 2010).

A fim de se obter um sistema com melhor desempenho, a consistência pode ser aplicada de maneiras diferentes. Essas maneiras consistem em permitir uma especificada diferença entre as cópias por um determinado período de tempo.

É importante notar que não há uma solução perfeita para a replicação de dados. Soluções eficientes só podem ser atingidas com o relaxamento na consistência. Não existe também uma regra geral para o quanto essa consistência pode ser relaxada, essa decisão é extremamente dependente do tipo de aplicação usada (AL-EKRAM & HOLT, 2010).

Diferentes modelos de consistência existem para solucionar diferentes necessidades. São apresentados os mais importantes nas subseções a seguir.

2.3.1 Modelos centrados nos dados

Um depósito de dados (*data store*) pode estar fisicamente distribuído por diversos computadores, mas assume-se que cada processo que pode ter acesso aos dados de um depósito possui uma cópia local ou próxima disponível do depósito completo. Uma operação é classificada como escrita

quando ela altera os dados, caso contrário, é considerada leitura (TANENBAUM & STEEN, 2006).

Um modelo de consistência consiste num conjunto de regras definido entre o depósito de dados e os processos. Um processo necessita ter acesso à versão mais recente desse dado (TANENBAUM & STEEN, 2006).

Na ausência de um relógio global, torna-se difícil precisar qual foi a última operação de escrita realizada nos dados. Como alternativa é necessário fornecer outras definições que levem a uma série de modelos de consistência. Cada modelo consiste em limitar os tipos de acessos realizados pelo processo. Modelos com mais restrições são mais fáceis de usar, mas têm baixo desempenho, enquanto modelos com menos restrições têm desempenho muito melhor, mas são bem mais complicados de utilizar (TANENBAUM & STEEN, 2006).

- **Consistência contínua**

Há diferentes maneiras para os aplicativos definirem quais inconsistências serão toleradas por eles. Yu & Vahdat (2002) fazem uma abordagem geral distinguindo três eixos diferentes para definir inconsistência. Os eixos são (TANENBAUM & STEEN, 2006):

  - Desvio em valores numéricos entre as réplicas - a inconsistência ocorre quando o valor dos dados ultrapassa o limite imposto pelo sistema. Variações entre os valores são permitidas, mas há limites que se ultrapassados acusam inconsistência. Essa inconsistência também pode ser acusada com um limite ao número de atualizações que uma determinada réplica sofreu sem que as demais fossem atualizadas;
  - Desvio por diferença de idade entre as réplicas - a inconsistência se dá pela última vez que uma réplica foi atualizada. O sistema controla e estipula tempo máximo para que uma cópia fique desatualizada;

- Desvio em relação à ordem das operações de atualização - permite-se que atualizações sejam aplicadas a cópias locais até que exista um entendimento global entre todas as réplicas. Consequentemente essas atualizações poderão ser revertidas e reaplicadas na ordem correta antes que se tornem permanentes.

Yu & Vahdat (2002) introduziram um método para calcular a inconsistência das réplicas no qual a unidade foi batizada de Conit (*Consistency Unit*).

- **Ordenação consistente de operações**

Tanto na computação paralela quanto na computação distribuída, múltiplos processos terão necessidade de compartilhar recursos e acessá-los simultaneamente. Tal necessidade fez com que pesquisadores desenvolvessem um modelo de consistência importante que é largamente utilizado. São apresentados os modelos conhecidos como consistência sequencial e sua variante mais frágil conhecida como consistência causal.

Consistência sequencial

Consistência sequencial é melhor definida por Lamport (1979) num contexto de memória compartilhada em ambiente multiprocessado. Um depósito de dados é tido como consistente quando satisfaz a seguinte condição:

> O resultado de qualquer execução é o mesmo como se as operações (leitura e escrita) de todos os processos no depósito de dados fossem executadas em alguma ordem sequencial e as operações de cada processo aparecessem nesta sequência na ordem especificada por seu programa (LAMPORT, 1979).

Segundo Tanenbaum & Steen (2006), a definição de Lamport (1979) significa que quando processos executam concorrentemente em máquinas

diferentes, qualquer intercalação válida de operações de leitura e escrita é um comportamento aceitável, mas todos os processos vêem a mesma intercalação de operações. O processo vê a escrita de todos, mas vê apenas as suas próprias leituras.

Existe uma notação especial na qual são desenhadas operações de um processo ao longo de um eixo de tempo. Este eixo de tempo é sempre desenhado horizontalmente (linha horizontal da Figura 1) com o tempo aumentando da esquerda para a direita.

A Figura 1 representa a operação de um processo *P1* realizando a escrita a um item de dado *x* modificando seu valor para *a* em sua cópia local. Em seguida é propagado para as demais cópias. No exemplo, *P2* leva certo tempo até enxergar a atualização do item *x*, primeiro enxerga *NIL*, depois enxerga *a*. Esse comportamento é aceitável para um sistema distribuído (TANENBAUM & STEEN, 2006).

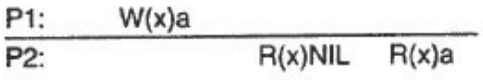

Figura 1 - Notação de consistência sequencial (TANENBAUM & STEEN, 2006).

Consistência causal

Consistência causal define um critério mais fraco de consistência do que a sequencial. Consistência causal faz uma distinção entre eventos que são potencialmente relacionados e os que não são. Ou seja, se um evento *b* é causado ou influenciado por um evento anterior *a*, a causalidade requer que todos vejam primeiro *a* e depois vejam o *b* (TANENBAUM & STEEN, 2006).

Por outro lado, se dois processos espontânea e simultaneamente realizam escrita em dois itens de dados diferentes, estes não são causalmente relacionados. Recebem a denominação de processos concorrentes. Para que um depósito de dados seja considerado causalmente consistente é necessário que sigam a seguinte condição de Tanenbaum & Steen (2006):

> Escritas que são potencialmente causalmente relacionadas devem ser vistas por todos os processos na mesma ordem. Escritas concorrentes podem ser vistas em diferente ordem em diferentes máquinas (TANENBAUM & STEEN, 2006).

A implementação de consistência causal requer que sejam monitorados quais processos viram quais escritas. Faz-se necessário a construção e manutenção de um grafo para o monitoramento (AHAMAD et al., 1993).

Consistência x Coerência

Os processos discutidos até então executam operações de leitura e escrita num conjunto de dados.

Em um modelo de consistência é descrito o que pode ser esperado em relação àquele conjunto quando múltiplos processos operam aquele dado concorrentemente. O conjunto é considerado consistente se ele atender as regras descritas pelo modelo.

Por outro lado, um modelo de coerência descreve o que pode ser esperado de um e apenas um item de dado. Um item de dados replicado em vários lugares é tido como coerente quando suas cópias seguem as regras definidas pelo modelo de coerência ao qual estão ligados. Funciona como um modelo de consistência sequencial, mas para um único item de dados (TANENBAUM & STEEN, 2006).

Na prática, ao decorrer de escritas concorrentes, todos os processos verão as atualizações ocorrerem na mesma ordem.

2.3.2 Modelos centrado no cliente

Em modelos centrados nos dados, o foco era manter uma visualização consistente de um depósito de dados, entretanto algo importante que se deve considerar, é que o depósito de dados pode estar sendo atualizado por

processos concorrentes e que é necessário prover consistência para tal situação de concorrência.

É fundamental para sistemas distribuídos ser capaz de lidar com operações concorrentes em dados compartilhados e manter ao mesmo tempo uma consistência sequencial. Apenas com o uso de transações e travas, que são mecanismos de sincronização, se pode garantir uma consistência sequencial, sem sacrificar o desempenho.

- **Consistência eventual**

Consistência eventual é o nome dado para o modelo no qual é comum que operações de atualização não ocorram por grande período de tempo e garante que ao longo do tempo todas as réplicas se tornarão consistentes. Armazéns de dados que são consistentes possuem a propriedade de que na ausência de atualizações, todas as réplicas tendem a convergir a cópias exatas umas das outras.

Esse modelo apenas requer a garantia que todas as atualizações serão propagadas para todas as cópias. O custo de implementação do modelo de consistência eventual costuma ser pequeno e conflitos de escrita simultânea são normalmente fáceis de solucionar, já que um grupo reduzido de processos pode fazer atualizações (TANENBAUM & STEEN, 2006)(TERRY et al., 1998).

- **Consistência leituras monotônicas**

Para que um depósito seja considerado provedor do modelo de consistência leitura monotônicas é necessário que siga a seguinte condição: “se um processo lê o valor de um item de dado *x*, qualquer operação de leitura sucessiva em *x* por aquele processo sempre irá retornar o mesmo valor ou outro mais recente”. O modelo garante que se um processo leu o valor *x* num determinado instante *t*, ele jamais encontrará uma versão mais antiga de *x*

posteriormente (TANENBAUM & STEEN, 2006). Leituras monotônicas utilizam notação similar àquela representada na Figura 1.

- **Consistência escritas monotônicas**

Um depósito de dados é considerado com consistência de escrita monotônica quando segue a seguinte condição: “uma operação de escrita por um processo em um item de dado *x* é completado antes de qualquer operação de escrita sucessiva em *x* pelo mesmo processo”.

Segundo Tanenbaum & Steen (2006), tal condição significa que uma operação de escrita num item *x* é realizada apenas se a cópia já foi atualizada por alguma operação de escrita anterior, que pode ter ocorrida em outras cópias de *x*. Se necessário, novas escritas deverão esperar até que as anteriores tenham terminado.

- **Consistência leia-suas-escritas**

O modelo leia-suas-escritas é tido como consistente se a seguinte condição for cumprida: “o efeito de uma operação de escrita por um processo em um item de dados *x* sempre será visto por uma subsequente operação de leitura em *x* pelo mesmo processo”.

Tanenbaum & Steen (2006) define que a consistência leia-suas-escritas ocorre quando uma operação de escrita é sempre completada antes que uma sucessiva operação de leitura pelo mesmo processo, sem importar onde a operação de leitura foi feita.

- **Consistência escritas-segue-leituras**

Este modelo de consistência é aquele em que atualizações são propagadas como resultado de operações de leituras anteriores. A consistência é mantida quando a seguinte condição é seguida: “uma operação de leitura por um processo em um item de dado *x* seguindo uma operação de leitura anterior em *x* pelo mesmo processo é garantida ocorrer no mesmo ou em um mais recente valor de *x* que foi lido”.

Qualquer operação sucessiva de escrita por um processo em um item de dados *x* será realizada em uma cópia de x que esteja atualizada com o valor mais recente lido por aquele processo, define Tanenbaum & Steen (2006).

2.4 CONSIDERAÇÕES FINAIS

O uso de replicação de dados ocorre por dois motivos principais: pelo aumento do desempenho de um sistema distribuído ou para aumentar a confiabilidade desse sistema.

Entretanto, devido às limitações de um sistema distribuído, principalmente na transferência de dados por rede de grande escala, surge o problema da consistência. Num sistema real a consistência é justamente o fator que tende a atrapalhar o desempenho de um sistema em prol de sua confiabilidade e disponibilidade. Cabe ao desenvolvedor medir as necessidades do sistema e definir qual o melhor modelo de consistência deve ser aplicado.

Replicação e tolerância a falhas são técnicas complementares e interdependentes. Para que a replicação de dados ocorra de maneira eficiente, é necessário que o sistema conte com uma política de tolerância a falhas que supervisione essa replicação.

3 TOLERÂNCIA A FALHAS

A natureza de sistemas distribuídos prevê a ocorrência de falhas parciais, diferentemente de sistemas não distribuídos. A principal causa desse comportamento é que muitas vezes o sistema não é capaz de identificar se um processo falhou ou está respondendo demasiadamente devagar.

Falhas ocorrerão em sistemas distribuídos. Conforme Cristian (1991), um servidor pode ser considerado correto quando ele provê correta e satisfatoriamente os serviços que se propõe a fazer. Logo, uma falha de servidor ocorre quando o mesmo não funciona da maneira esperada.

Saha (2005) define que falha significa que um sistema não está atendendo os requerimentos do usuário de alguma forma durante seu funcionamento. Um erro é parte do estado de um sistema que pode levá-lo à uma falha. A causa de um erro é chamada de falta.

Cabe ao desenvolvedor do sistema, analisar as possíveis maneiras que esse sistema pode falhar e definir como serão tratadas essas falhas a fim de manter um sistema confiável, disponível, seguro e capaz de se recuperar.

Este capítulo tem o intuito de identificar os principais tipos de falhas e as principais técnicas de recuperação dessas falhas.

3.1 TIPOS DE FALHAS

Os tipos de falhas são definidos em transientes, intermitentes ou permanentes (SAHA, 2005). Falhas transientes são aquelas que tendem a ocorrer uma única vez, e normalmente estão associadas a uma interferência externa, como uma queda de energia.

Falhas intermitentes são aquelas que surgem de tempos em tempos e costumam ser bastante complicadas de identificar, já que ela tende a ocorrer sem um padrão definido.

Falhas permanentes são aquelas que ocorrem o tempo todo e normalmente são mais fáceis de detectar. Esse tipo de falha usualmente requer a substituição ou reparo do dispositivo faltoso ou correção no código-fonte.

3.2 MODELOS DE FALHAS

Existem diversos modelos de falhas que foram classificadas de acordo com suas características para facilitar a identificação, pesquisa e desenvolvimento de soluções.

As principais falhas são por omissão, por queda, de temporização, de resposta e arbitrária (bizantina).

- **Falha por omissão**

Falhas por omissão podem ser do tipo de recebimento ou de envio. Elas ocorrem quando um servidor não responde a uma requisição ou nunca recebeu tal requisição. Normalmente essas falhas são causadas por problema na entrega e recebimentos de mensagens, apontando problemas na forma de comunicação.

- **Falha por queda**

Falhas por queda ocorrem quando um servidor que funcionava normalmente, deixa, em definitivo, de responder a tudo que lhe for solicitado ou produzir qualquer resposta. Tais falhas só podem ser recuperadas através da reinicialização do processo e/ou servidor.

- **Falha de temporização**

Falhas de temporização ocorrem quando um determinado servidor leva um tempo demasiado longo para responder a uma solicitação, extrapolando o

tempo limite especificado. Pode também ocorrer, de uma resposta ser rápida demais e o receptor ainda não estar preparado para a mesma, entretanto esse tipo de ocorrência é menos comum.

- **Falha de resposta**

Falhas de resposta são caracterizadas pelo erro de um servidor ao responder uma determinada requisição. Podem ocorrer falhas de valor ou de transição de estado. Na falha de valor, o servidor dará uma resposta contendo valor errado ao requisitante. Na falha de transição de estado, o servidor reage de forma inesperada a uma solicitação, podendo desencadear comportamentos não esperados àquela situação.

- **Falha arbitrária ou bizantina**

Falhas bizantinas são aquelas nas quais um servidor pode produzir uma resposta arbitrária, ou seja, de sua própria vontade, desobedecendo as regras e restrições do sistema. Este tipo de falha é o mais grave, pois pode induzir os demais servidores a tomarem uma resposta arbitrária como a correta.

3.3 PROPRIEDADES DE UM SISTEMA TOLERANTE A FALHAS

Segundo Tanenbaum & Steen (2006) tolerância a falha significa que um sistema distribuído pode funcionar normalmente, provendo seus recursos e serviços, tolerando a presença de falhas.

- **Confiabilidade**

Confiabilidade é a capacidade que um sistema possui em se manter continuamente funcionando sem interrupção por longos períodos de tempo. Um

sistema distribuído pode ter seu nível de confiabilidade associado a sua capacidade de tolerar falhas.

- **Disponibilidade**

Disponibilidade é a propriedade que um sistema distribuído tem de estar disponível ao seus usuários toda vez que lhe for solicitado, sem importar se houveram falhas anteriores de pequena duração. A ocorrência de uma solicitação realizada por um cliente não ser respondida pelo servidor significa que houve uma falha de disponibilidade.

- **Segurança**

Segurança é a propriedade que mostra a capacidade de um sistema manter a integridade e confidencialidade de seus dados e funções mesmo perante a falhas por um determinado tempo. O termo segurança é complexo e abrange as outras propriedades, tornando complicado a medição do nível de segurança por se tratar de uma propriedade qualitativa.

- **Capacidade de recuperação**

Essa propriedade representa a capacidade de um sistema distribuído em detectar que falhou e automaticamente disparar mecanismos que o auxiliem a retomar seu funcionamento normal. A capacidade de recuperação automática é de difícil obtenção devido a sua complexidade.

3.4 REDUNDÂNCIA

Redundância é a principal técnica usada para o tratamento de falhas. A redundância é empregada de forma a garantir que um processo faltoso seja substituído por outro correto, mascarando as falhas.

São usados três tipos de redundância com o intuito de mascarar falhas em sistemas distribuídos. Redundância de informação, de tempo e física.

Segundo Curiac et al. (2009), redundância de informação é o uso de informações adicionais com a finalidade de detectar erros e se recuperar de falhas. Por exemplo, a inserção de bits de paridades nas mensagens de rede para checar a integridade dos dados e recriar a mensagem sem a necessidade de retransmissão.

Redundância de tempo é a técnica que usa de esperas e reexecuções para realizar uma atividade que falhou. Uma determinada atividade que falha ao ser executada no instante t, é agendada pelo sistema para que seja executada novamente após x segundos.

Redundância física é também conhecida com redundância de hardware. Ela consiste em duplicar (normalmente) os recursos de hardware permitindo que a falha de um item faltoso seja mascarada pelo funcionamento do item remanescente. Como exemplo, dois discos rígidos podem ser configurados de modo que ambos contenham os mesmos dados, onde a falha de um deles não acarretará na perda dos dados.

Em circuitos eletrônicos a redundância física é também bastante empregada e a principal técnica utilizada para tolerar falhas é conhecida como Redundância Modular Tripla (*Triple Modular Redandancy* ou TMR). Segundo Johnson (1996), o conceito consiste em triplicar os recursos de hardware e fazer que ocorra uma votação (onde o mais votado vence) para determinar qual a saída correta do sistema. Caso um dos módulos falhe, os dois restantes irão mascarar os resultados do módulo faltoso após a votação ser realizada.

A Figura 2 ilustra um exemplo de TMR simples, com apenas um votante. É importante ressaltar que o votante também é um componente e também está sujeito a falhas. A utilização de recursividade e de mais votantes aumenta a redundância e a confiabilidade do sistema.

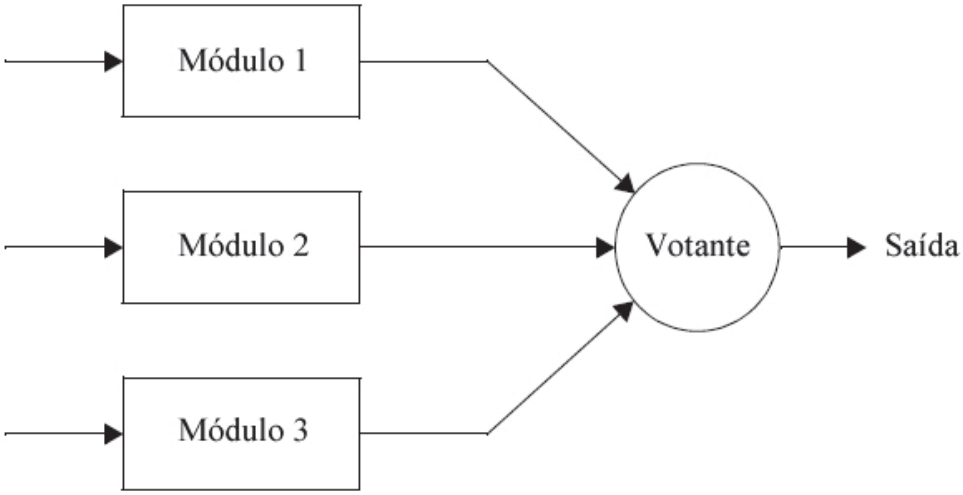

Figura 2 - Exemplo de Redundância Modular Tripla (TMR).

3.5 FALHAS NA COMUNICAÇÃO CLIENTE/SERVIDOR

Sistemas distribuídos são formados por um número indeterminado de computadores se comunicando entre si por uma mesma rede. Considerando os modelos de falhas abordados anteriormente, é importante ressaltar que a maioria desses modelos podem ser afetados por falha de comunicação.

Uma comunicação confiável pode ser obtida com a escolha de um protocolo confiável de transporte como o TCP. O TCP é capaz de mascarar falhas por omissão usando reconhecimentos e retransmissões, onde um cliente TCP não saberá da existência dessas falhas. Por outro lado, as falhas por queda de conexão são visíveis ao cliente TCP, embora seja possível que o sistema distribuído mascare a falha realizando uma reconexão automática, caso ainda seja possível (TANENBAUM & STEEN, 2006).

3.5.1 Falhas em Chamada de Procedimento Remoto (RPC)

Segundo Peterson e Davie (2011), RPC é mais que um simples protocolo, é um mecanismo popular de estruturação de sistemas distribuídos.

Afirmam que sua popularidade ocorre por ele ter sua semântica baseada em chamadas de procedimentos locais, já a aplicação realiza uma chamada a um procedimento sem se importar se é local ou remoto e bloqueia até a chama retornar.

Para Birrell e Nelson (1984), um RPC ocorre quando um procedimento remoto é invocado, o ambiente que realizou a chamada é suspenso, os parâmetros são transferidos através da rede até o ambiente em que devem ser executados e o procedimento desejado é realizado lá. Ao final do procedimento os seus resultados são transferidos de volta ao ambiente que realizou a chamada, onde a execução retoma seu funcionamento.

A principais razões pelo uso de RPC são sua simplicidade (semântica simples), eficiência (comunicação rápida) e genericidade (funciona local ou remoto).

Tanenbaum e Steen (2006) ressaltam que o RPC faz bem seu trabalho até que um problema ocorra, e é perante falhas que as diferenças entre chamadas remotas e locais se destacam. Essas falhas não são fáceis de mascarar e eles as caracterizam em cinco classes de falhas em RPC: cliente não localiza servidor, pedido do cliente é perdido, servidor cai após receber o pedido, resposta do servidor é perdida e cliente cai após enviar pedido.

- **Cliente não localiza o servidor**

O cliente é incapaz de localizar qualquer servidor e tal fato será detectado como uma falha. Essa falha pode ser tratada pelo desenvolvedor por meio da implementação de exceções. O sistema ao detectar a ocorrência de uma exceção NOSERVER, pode iniciar uma rotina que irá aguardar pela disponibilidade de servidores (Redundância de Tempo). Entretanto, o mascaramento de tal falha é comprometido em tal cenário.

- **A mensagem de pedido do cliente ao servidor é perdida**

A mensagem de pedido enviada é perdida por algum motivo e uma falha é detectada. O tratamento é relativamente simples e consiste em iniciar um temporizador ao enviar a mensagem. Caso o tempo estipulado exceda o limite antes de receber uma reposta do servidor ou reconhecimento, o cliente envia uma nova mensagem. Contudo, a quantidade de tentativas de tentativas de envio também devem ser tratadas, assim como a capacidade do servidor em detectar uma retransmissão (se a mensagem não tiver sido perdida). Em casos de constantes falhas pode-se chegar a uma conclusão errônea de que o servidor não pode ser localizado (NOSERVER).

- **O servidor cai após receber o pedido**

O servidor cai após receber o pedido é de difícil tratamento. A queda pode ocorrer dois estágios após o recebimento do pedido: logo após o recebimento do pedido ou pode ocorrer logo após a execução, mas antes de enviar a resposta. Em nenhum dos dois casos uma resposta é enviada. O tratamento em cada caso é diferente.

O tratamento pode ser realizado genericamente usando a "semântica ao menos uma vez", na qual o cliente fica tentado até receber uma resposta que garante que a execução ocorrerá ao menos uma vez, mas podendo executar várias. Essa abordagem pode ocasionar custos computacionais ou materiais excessivos, já que um pedido de impressão ao servidor seria executado inúmeras vezes ocupando a impressora e gastando tinta e papel desnecessariamente.

Outro tratamento genérico utilizado (ou filosofia) é a "semântica no máximo uma vez", onde o cliente tanta executar apenas uma vez, desiste e informa a falha. Nessa proposta o cliente garante que não haverá requisições, nem execuções duplas, mas pode possivelmente ficar sem nenhuma execução. Como exemplo, uma requisição para aumento no valor de um

determinado produto não seria executada, causando conflitos e discrepâncias com o valor esperado na venda.

Outro tratamento genérico é o "nada garantir", onde o pedido é enviado, mas ao cair do servidor nenhuma informação é dada ao cliente. A execução pode ocorrer zero ou muitas vezes sem que o cliente tome conhecimento. É uma abordagem de fácil implementação, mas insuficiente na maioria dos casos.

A tratamento de falhas quando ocorre a queda do servidor após o recebimento é complexo e requer uma combinação de técnicas e abordagens tanto no cliente quanto no servidor de acordo com as necessidades do sistema.

- **Mensagem de resposta do servidor ao cliente é perdida**

A mensagem de resposta enviada é perdida por algum motivo e uma falha é detectada. A utilização de um temporizador no cliente é uma maneira de tratar superficialmente tal problema. O maior problema nesse caso é identificar em qual estágio ocorreu realmente a falha. A mensagem de resposta pode nunca ter sido emitida ou se perdido ou nem ter sido formulada ainda devido a lentidão do servidor ou não ter chegado por uma lentidão na rede.

Algumas requisições são simples e podem ser repetidas, pois não gerarão nenhum problema sério, mas em outros casos é extremamente importante que isso não ocorra. O uso de identificadores sequenciais nas requisições pode ser implementado como forma de evitar que retransmissões sejam executadas, mas de qualquer forma requerem uma mensagem de resposta. Outra técnica é implementar algum bit identificador no cabeçalho de toda mensagem original para diferenciá-las das retransmissões e tratar esse tipo de mensagem com mais cautela no servidor.

- **Cliente cai após enviar um pedido**

O cliente envia um pedido ao servidor e cai antes de receber sua resposta. Um problema é gerado, pois é gerado um órfão. Órfão é o nome

dado aos pedidos em que o solicitante (pai) não é mais encontrado. Esse órfão gera custos computacionais e precisam ser tratados de forma satisfatória.

Nelson (1981) propõe quatro soluções para o tratamento de órfãos e as compara. Na solução "exterminação", apenas nós com órfãos são conferidos e apenas o processo em que está órfão é abortado. Na solução "expiração", todos os nós com órfãos periodicamente conferem seus próprios processos em busca de órfãos, apenas os processos com órfãos são abortados. Na solução "reencarnação", todos os nós são conferidos e todo trabalho remoto é abortado. Na solução "reencarnação gentil", todos os nós são conferidos, mas apenas os processos com órfãos são abortados.

Tanenbaum e Steen (2006), ressaltam que as quatro soluções apresentadas por Nelson (1981) são grosseiras e podem causar problemas imprevisíveis e cita como exemplo o extermínio de um órfão que tenha colocados travas em arquivos ou realizado modificações que serão utilizadas no futuro por outros processos e não poderão ser revertidas.

3.6 RECUPERAÇÃO DE FALHAS

Uma importante parte de sistemas tolerantes a falhas é sua habilidade em se recuperar delas. Segundo Baldoni et al. (1985), recuperação de um erro é a tarefa que faz com que o sistema saia de um estado errôneo e retorne a um estado livre de erro. As técnicas de recuperação de erros são normalmente classificados em recuperação adiante e recuperação retroativa.

Randell et al. (1978) afirmam que as técnicas de recuperação visam colocar o sistema em um estado no qual o processamento pode prosseguir e as falhas podem ser evitadas.

3.6.1 Recuperação retroativa

A recuperação retroativa consiste em restaurar o sistema a um estado sabidamente livre de erro, antes de prosseguir com o funcionamento do sistema. Tal restauração do sistema torna-se possível por meio do salvamento do estado do sistema em determinada frequência de tempo.

A eficiência desta técnica depende da criação de pontos de recuperação em dispositivos de armazenamento não volátil, como discos rígidos, CDs, DVDs, SSDs, etc. Esses pontos são criados em disco e em caso de falhas, o sistema retoma seu funcionamento através da leitura e restauração dessas informações de acordo com seus critérios.

O uso de memória volátil (RAM e *cache*) para armazenamento de pontos de recuperação são inapropriados, pois dispositivos de armazenamento voláteis perdem seus dados quando há falta de energia ou são reiniciados. Entretanto o uso desses dispositivos geram um aumento de desempenho devido ao fato de serem mais rápidos que os não voláteis.

Existem diversas abordagens e técnicas para a realização de recuperação retroativa, mas a implementação de cada uma delas depende dos requisitos e características do sistema. A criação de pontos de recuperação e restauração retroativa é cara em termos de desempenho e as propriedades do sistema definirão a viabilidade do uso de tal recurso.

Segundo Randell et al. (1978), a recuperação retroativa é simples devido a dois fatores, a avaliação dos danos é tratada separadamente da avaliação de como continuar fornecendo um serviço especifico; e porque a origem da falha não importa em nada para a recuperação do sistema.

3.6.2 Recuperação adiante

Baldoni et al. (1985) definem que a recuperação adiante é baseada na tentativa de continuar fazendo uso do estado que foi encontrado com erro.

Afirmam que tal feito só é possível quando a natureza do erro e suas consequencias podem ser completamente avaliadas, para então remover os erros e permitir que o sistema siga adiante.

Para Tanembaum e Steen (2006), o problema principal para a recuperação adiante é que é necessário saber previamente quais problemas podem ocorrer. Apenas nesse caso é possível corrigir os erros e mover o sistema para um estado adiante.

Ao contrário da recuperação retroativa que pode ser desenvolvida de maneira mais generalizada, a recuperação adiante deve ser desenvolvida como parte integrante do sistema em que atua. Apesar dessa técnica requerer um envolvimento maior com vários aspectos do sistema, ela ainda pode ser bastante simples e efetiva. A simplicidade das falhas esperadas é proporcional a simplicidade do esquema de recuperação adiante, podendo muitas vezes ser mais simples e eficiente do que o uso de recuperação retroativa, já que repara apenas o que foi feito errado (Randell et al., 1978).

Tanto a recuperação retroativa como a adiante devem e podem ser usadas em combinação para obter um sistema mais tolerante a falhas.

3.6.3 Pontos de recuperação (*checkpoints*)

A criação de pontos de recuperação consiste em gravar o estado do sistema em armazenamento estável, com o intuito de ser possível restaurar o sistema em caso de necessidade.

Segundo Kshemkalyani e Singhal (2008), há um estado local para cada elemento de um sistema distribuído. O estado local (*snapshot*) é composto pelo estado de sua memória local e o histórico de suas atividades. O estado de um canal é caracterizado pelo conjunto de mensagens enviadas através dele, excluindo-se as mensagens recebidas. A coleção de estados locais dos elementos formam o estado global. Entretanto, as ausências de um relógio

global e de memória compartilhada tornam complexa a criação de um estado global eficiente, dada a natureza distribuída do sistema.

Chandy e Lamport (1985) afirmam que para determinar um estado global de sistema, um processo *p* deve contar com a colaboração dos outros processos que devem gravar seus próprios estados locais e enviá-los a *p*.

Um conjunto arbitrário de checkpoints locais não podem formar um *checkpoint* global consistente, porque este depende muito do fluxo correto das mensagens trocadas pelos processos.

Considerando que um *checkpoint* foi criado com sucesso através de um algoritmo, é importante ressaltar que a restauração com sucesso desse *checkpoint* é tarefa tão ou mais complicada de se realizar.

De acordo com Ng (1988), há dois problemas existentes na utilização de um sistema de *checkpoints*. O primeiro é a perda de trabalho causada por falhas e de como lidar para essa perda poder ser reduzida. O segundo é a restauração ao estado livre de erro e de como encontrar o mais recente checkpoint consistente dentro de um conjunto de processos se comunicando e fazê-los voltarem ao estado desejado.

Há na literatura diversas propostas de algoritmos que visam aprimorar o gerenciamento de *checkpoints*, cada uma delas abordando de maneiras diferentes como lidar com os desafios inerentes a natureza de um sistema distribuído. Em Kshemkalyani e Singhal (2008) são analisados diversos algoritmos que foram desenvolvidos para o gerenciamento de checkpoints.

O conceito de *checkpoint* ou *snapshots* é um recurso bastante utilizado em Virtualização por dar a habilidade de salvar e restaurar o estado de uma Máquina Virtual (MV). Um *snapshot* de Máquina Virtual consiste em armazenar o estado do processador, dos dispositivos emulados e de sua memória volátil em armazenamento não volátil (Park et al., 2011). Entretanto, *snapshots* de Máquinas Virtuais podem se tornar um problema quanto ao tamanho do arquivo gerado que pode atingir vários *gigabytes*. A seção 7.2 sugere um estudo da viabilidade do uso de *checkpoints* de Máquinas Virtuais em trabalhos futuros baseados na ferramenta WSE-OS Escalável.

3.7 CONSIDERAÇÕES FINAIS

Um sistema distribuído está sujeito a falhas constantemente. A habilidade de um sistema em mascarar falhas e se recuperar delas é definida como tolerância a falhas. Um sistema tolerante a falhas deve sempre que possível obter maneiras de mascarar e se recuperar de falhas de maneira transparente ao usuário.

As falhas podem e irão ocorrer independentemente de como o sistema foi construído. Podem ocorrer falhas no *hardware*, na rede, no *software* ou na combinação desses elementos.

O uso de redundância é a principal técnica para a obtenção de tolerância a falhas. As redundâncias de tempo, de informação e física devem ser usadas em conjunto para assegurar que quando um elemento do sistema falhe, haja um outro que o substitua e mantenha o funcionamento estável.

As técnicas de replicação de dados e tolerância a falhas estão muito ligadas uma a outra, pois replicação é também uma forma de redundância e ao mesmo tempo a consistência dessas replicações depende muito da capacidade tolerar falhas do sistema. Afinal, uma replicação mal sucedida pode propagar um estado errôneo e prejudicar o sistema todo se não houver uma política de recuperação eficiente.

4 A FERRAMENTA WSE-OS

WSE-OS é a sigla para *Wireless Sharing Environment - Operating Systems*. Esta ferramenta computacional, desenvolvida por Crepaldi (2011) e Digiere (2011), cria um ambiente de compartilhamento de Sistemas Operacionais (SO) focado ao uso em redes sem fio. A ferramenta associa técnicas de *Thin Client* (TCSC), *Desktop* Virtual (VDI), Virtualização e acesso remoto seguro em sua composição (CREPALDI et al., 2011).

4.1 CONSIDERAÇÕES INICIAIS

Devido ao contínuo avanço na computação móvel e na heterogeneidade dos dispositivos disponíveis no mercado, surge aos administradores o problema de como gerenciar esses diversos dispositivos e adequá-los ao ambiente em que serão usados. Uma solução para este problema pode ser criada a partir da centralização dos dados e recursos de um sistema. Tal centralização num ambiente de rede sem-fio foi proposta pela ferramenta WSE-OS (CREPALDI et al., 2011).

O WSE-OS é uma ferramenta que tem por finalidade facilitar a administração de um ambiente computacional. Ela foi desenvolvida para operar em um ambiente de rede sem-fio (*wireless*), sendo composta por dois módulos principais: cliente e servidor.

A ferramenta WSE-OS permite que um usuário, usando qualquer PC capaz de executar um dispositivo de armazenamento secundário (USB *flash drive,* cartão de memória SD, etc.) contendo o cliente WSE-OS corretamente instalado, conecte-se ao servidor WSE-OS em um ambiente de *desktop* remoto e execute suas máquinas virtuais a partir desse servidor (CREPALDI, 2011).

A Figura 3 ilustra a arquitetura do sistema WSE-OS e como interagem clientes e servidor.

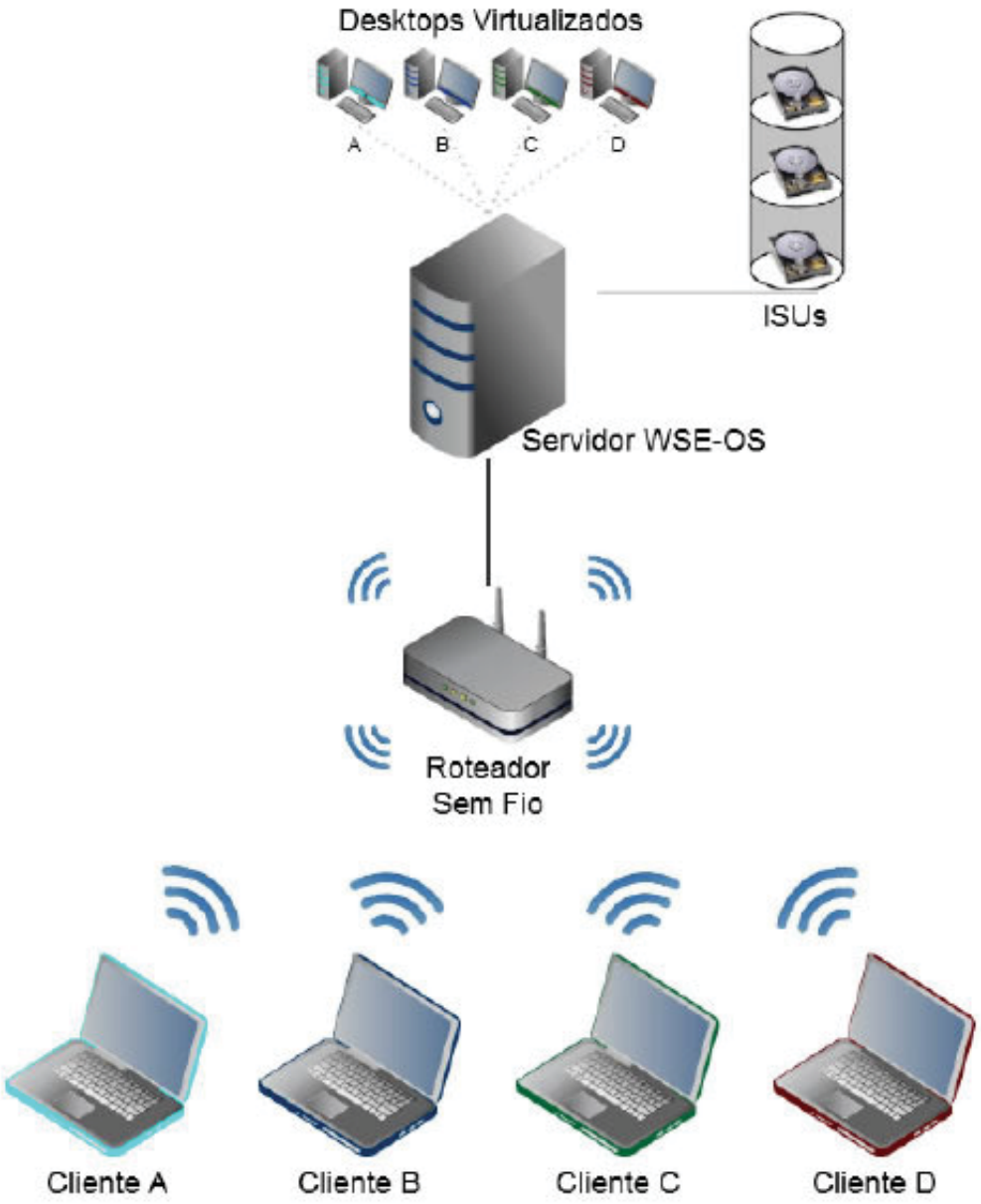

Figura 3 - Arquitetura do sistema WSE-OS (DIGIERE, 2011).

A arquitetura do WSE-OS (Figura 3), composta pelos módulos cliente e servidor, é descrita nas próximas seções.

4.2 CLIENTE WSE-OS

O cliente WSE-OS foi desenvolvido no conceito TCSC (*Thin Client Server Computing*[1]), ele é executado a partir de uma mídia removível secundária, dispensando a necessidade de disco rígido na máquina cliente. Crepaldi (2011) utiliza o nome "*Middleware* de comunicação *wireless* WSE-OS" para o sistema responsável pela comunicação sem-fio que opera no cliente. Ele fica situado entre o *hardware* do cliente e o sistema responsável pelo gerenciamento do servidor. É o responsável pelo gerenciamento da comunicação entre cliente e servidor, e pelos processos internos do cliente. A Figura 4 ilustra a disposição dos módulos do sistema WSE-OS.

O *middleware* possui três módulos principais interdependentes que realizam a transmissão e recebimento dos dados:

- Base de gerenciamento Operacional - foi desenvolvida a partir de uma versão recompilada do *kernel* do Linux (Fedora Core 13 x86 ou FC13) que oferece controle de periféricos, processos e suporte à rede sem-fio;
- Mecanismo de criação de conexões seguras - usa técnicas de criptografia e túnel orientado a fluxo TCP para criar conexões seguras com outros computadores. É baseado no protocolo Secure Shell ou SSH (SSH, 2012) com o intuito de manter a confidencialidade e integridade no tráfego dos dados pela rede sem-fio (CREPALDI, 2011);
- Módulo de comunicação cliente - é responsável pela comunicação entre cliente e servidor realizado por meio de invocação a métodos remotos baseados em eventos. Para a parte gráfica, o cliente utiliza o aplicativo FreeNX Client, que possui mecanismo de empacotamento e desempacotamento para o

[1] *Thin Client* é o nome dado ao computador ou *software* que tem pequena parte do processamento executado nele mesmo, normalmente apenas tarefas básicas de entrada e saída de dados, sendo extremamente dependente do servidor (NIEH et. al., 2003).

suporte de orientação a objetos distribuídos, para ter acesso remoto a um ambiente de infraestrutura de *desktop* virtual[2] (VDI) (CREPALDI, 2011).

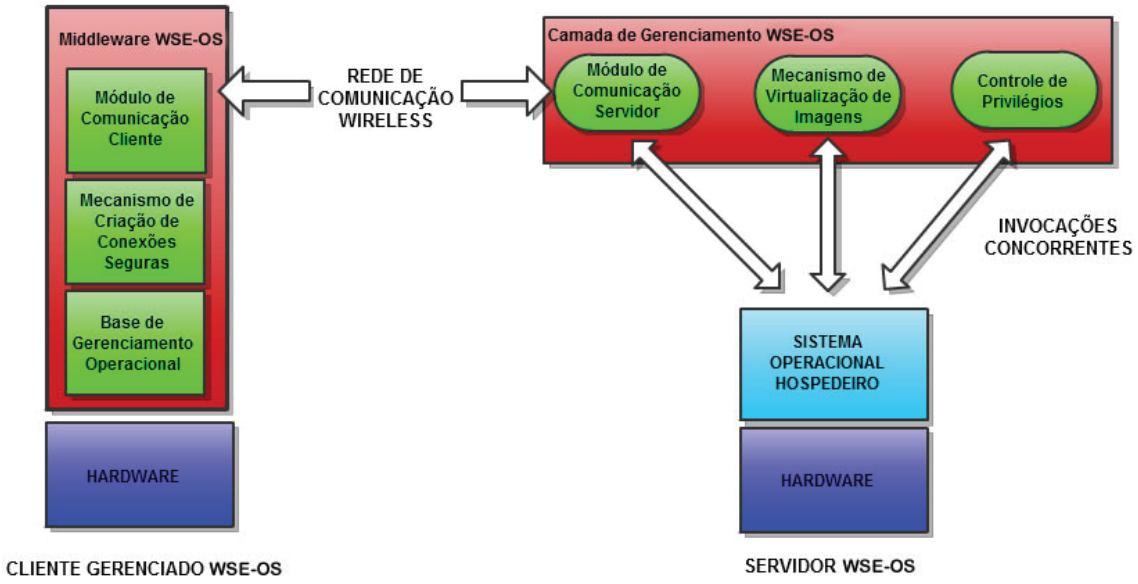

Figura 4 - Arquitetura do WSE-OS. Adaptado de (DIGIERE, 2011).

4.3 SERVIDOR WSE-OS

O servidor WSE-OS é composto por um sistema desenvolvido por Digiere (2011) denominado “Camada de Gerenciamento do WSE-OS”. Esse sistema é composto por ferramentas de acesso remoto, sistema de arquivos e virtualização combinados para oferecer uma solução na qual um cliente remoto possa interagir com uma instanciação de sistema operacional hospedado no servidor (DIGIERE, 2011).

O sistema operacional do servidor é o Linux Fedora Core 13 x86 e sobre ele fica situada a “Camada de gerenciamento do WSE-OS” que é composta por

[2] VDI é o conceito de executar aplicativos e sistemas operacionais dentro de máquinas virtuais alocadas em um servidor. O usuário tem acesso ao desktop e aplicativos de maneira completa e isolada e/ou de acordo com as políticas impostas pelo administrador.

três módulos independentes que se complementam a fim de fornecer acesso a um sistema operacional ao cliente (DIGIERE, 2011):

- Módulo de Comunicação Servidor - composto pelo aplicativo FreeNX Server que é responsável por criar o ambiente de Desktop virtual para o cliente, pelo SSH que é responsável em manter a conexão cliente-servidor segura, pelo SSHFS (Secure Shell File System) que permite que um dispositivo conectado na máquina cliente seja acessado pelo Sistema Operacional virtualizado;
- Mecanismo de Virtualização de Imagens - composto pelo aplicativo VirtuaBox que é o responsável pela execução das Máquinas Virtuais (MV). Ele carrega a ISU[3] selecionada pelo usuário e gerencia a execução das MVs de forma isolada e independentes garantindo as integridades do servidor e clientes (DIGIERE, 2011);
- Controle de Privilégios - composto por um *script* desenvolvido em Python (PYTHON, 2012) que é responsável em gerenciar as requisições de acesso ao servidor e de intermediar a comunicação entre os módulos do sistema. Age de maneira segura, permitindo que o cliente tenha acesso apenas ao *menu* de seleção de sistema operacional.

4.4 UTILIZAÇÃO DO WSE-OS

O WSE foi desenvolvido no modelo cliente/servidor, onde maior parte do processamento é executado no servidor. Sua estrutura prevê a utilização de *n* clientes e apenas um (1) servidor que centraliza os recursos e armazena os dados de seus clientes.

[3] Imagem de Sistema Única ou ISU é o nome dado ao disco rígido virtual, simulando num único arquivo, todo o conteúdo de um disco rígido físico.

O WSE-OS é livre e gratuito. Ele foi desenvolvido utilizando apenas aplicativos gratuitos e sua estrutura é adequada para a utilização em ambiente acadêmico. A centralização de recursos permite que um conteúdo específico para determinada aula seja preparado em uma ISU e disponibilizado no servidor a todos os seus clientes (alunos), independendo das características específicas do computador de cada cliente.

Tal utilização pode ser feita em vários ambientes, como escolas, empresas e eventos. O WSE-OS pode ainda ser utilizado em ambiente privado, para disponibilizar recursos comuns a uma mesma família de maneira mais eficiente e econômica, já que utiliza da tecnologia de virtualização.

Usuários comuns podem se beneficiar da virtualização utilizada no WSE-OS de maneira simples e sem custo. Como exemplo, um usuário migra de seu atual sistema operacional para uma versão mais atual, mas depois de feita a migração nota que sua impressora ainda em bom estado, não é compatível com o novo sistema operacional. A virtualização serve nesse caso como solução sem custo para tal problema. Uma máquina virtual contendo o sistema operacional antigo com a impressora instalada no mesmo permite maior longevidade a impressora.

O WSE-OS é indicado para situações em que vários usuários tenham a necessidade de acessar um mesmo ambiente computacional, não importando a heterogeneidade de seus dispositivos.

4.5 CONSIDERAÇÕES FINAIS

A natureza do sistema WSE-OS permite que exista um gerenciamento centralizado dos recursos de um ambiente computacional por meio de sua arquitetura *Thin Client* que centraliza todos os dados no servidor WSE-OS. A heterogeneidade dos clientes deixa de ser um problema para o administrador, já que qualquer computador do tipo PC munido de uma placa de rede sem-fio pode ter acesso ao mesmo conteúdo que qualquer outro cliente, independentemente da configuração desses clientes.

A facilidade de acesso ao sistema WSE-OS e a praticidade na utilização, podem ocasionar um problema quanto à escalabilidade e disponibilidade desse sistema, já que o projeto inicial opera com apenas um servidor e esse pode ter sua capacidade de processamento superada gerando problemas e interrupções no funcionamento. Outros potenciais problemas não tratados são a ocorrência de falhas no *hardware* e a perda dos dados de todos os clientes.

A monografia tem por finalidade apresentar uma proposta para melhorar o funcionamento da ferramenta do WSE-OS por meio da implementação de novos recursos e aprimoramento dos existentes.

5 A FERRAMENTA WSE-OS ESCALÁVEL

Neste Capítulo é apresentada a ferramenta WSE-OS Escalável que foi desenvolvido visando a melhoria do WSE-OS existente. Foram analisados a proposta e o funcionamento do WSE-OS e foi desenvolvido o WSE-OS Escalável de acordo com os objetivos definidos por este trabalho.

5.1 CONSIDERAÇÕES INICIAIS

A ferramenta WSE-OS descrita no Capítulo 4 oferece diversas vantagens para a administração de um sistema devido a centralização dos seus dados e recursos e sua versatilidade de poder ser útil em diversos cenários diferentes.

Entretanto, o WSE-OS foi desenvolvido com uma estrutura com de *n* clientes para 1 servidor e esse fator unido a sua característica centralizadora acabam limitando a escalabilidade do sistema.

Digiere (2011) concluiu após seus experimentos, entre outras coisas, que a quantidade de memória RAM do servidor, é um dos fatores que determina a quantidade de execuções concorrentes de Máquinas Virtuais (clientes) suportadas pelo sistema antes de ocorrer degradação substancial ao desempenho. Foi observado que a degradação anormal no desempenho do clientes ocorreu quando a quantidade de memória RAM utilizada pelas máquinas virtuais excedeu 50% da memória RAM total do servidor.

A capacidade da ferramenta WSE-OS em ser escalável é limitada por dois problemas principais: capacidade do servidor e capacidade da rede. O WSE-OS Escalável propõe uma estrutura que visa um aumento da escalabilidade, disponibilidade e confiabilidade do WSE-OS original.

5.2 PROPRIEDADES DA FERRAMENTA WSE-OS ESCALÁVEL

Tendo em vista os problemas de escalabilidade do WSE-OS, o WSE-OS Escalável alterou a estrutura de seu antecessor e implementou a capacidade de utilização de n servidores simultaneamente. O Capítulo 6 apresenta os experimentos realizados com a finalidade de encontrar um valor máximo a n.

A alteração na estrutura para o funcionamento com *n* servidores foi implementada com a finalidade de sanar o problema de escalabilidade do sistema gerado pela capacidade do único servidor existente que ocorre no WSE-OS. O WSE-OS Escalável propõe que a quantidade de servidores utilizados pode variar de acordo com a demanda do sistema.

A Figura 5 ilustra a estrutura da ferramenta WSE-OS Escalável utilizando *n* servidores e *m* clientes. A comunicação entre os servidores e o roteador ocorre de forma cabeada, enquanto a comunicação entre clientes e roteador ocorre de forma sem fio (*wireless*).

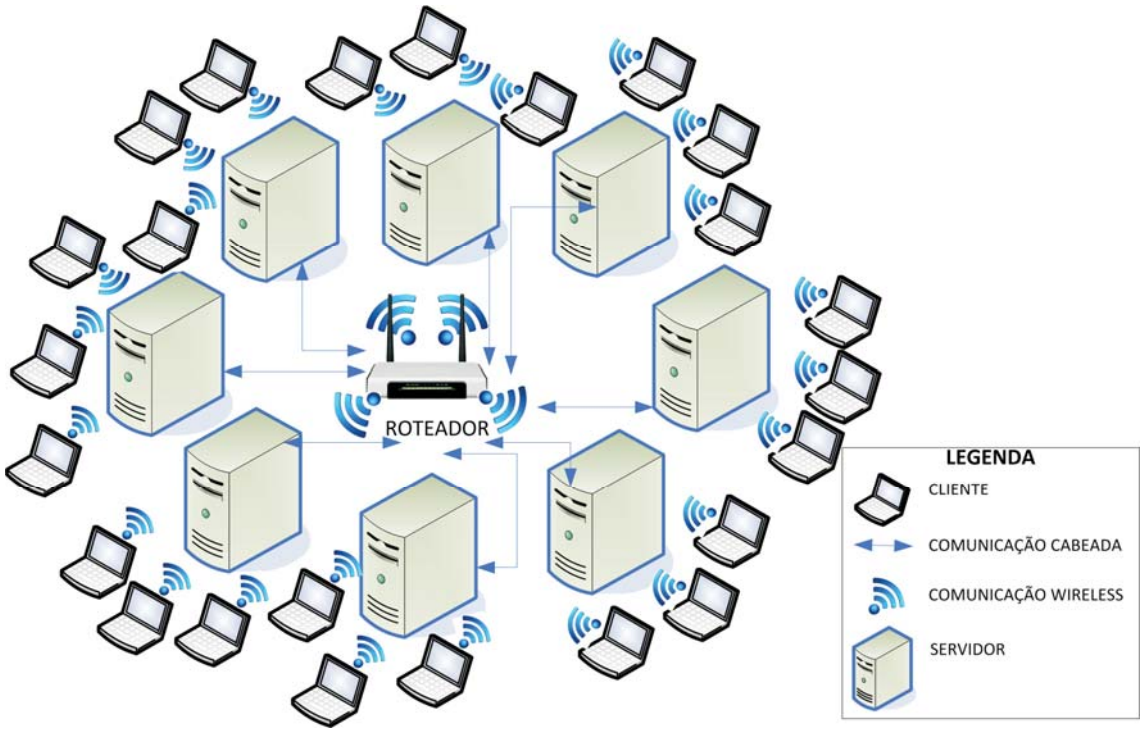

Figura 5 - Estrutura da ferramenta WSE-OS EScalável

Como consequência da alteração para *n* servidores, alterações importantes foram necessárias para que o sistema pudesse se beneficiar e manter o sistema confiável e funcional.

O WSE-OS Escalável implementou dois módulos principais na estrutura existente, o Módulo de Escalonamento de Conexões (ModEsc) e o Módulo de Replicação de Dados dos Servidores (ModRep). A ferramenta foi desenvolvida de maneira modular, visando a facilidade na inserção e remoção de qualquer um de seus elementos sem afetar o sistema num todo.

O WSE-OS Escalável foi implementado de maneira que não houvesse um servidor principal no qual, em funcionamento normal, todos os servidores devem conter o mesmo conteúdo em maior parte do tempo. Para a obtenção de tal requisito de forma satisfatória surge a necessidade da replicação dos dados dos servidores a fim de manter a consistência entre os mesmos. O Módulo de Replicação de Dados dos Servidores foi implementado como o responsável por esta tarefa.

Com a existência de *n* servidores disponibilizando o mesmo conteúdo e *m* clientes utilizando esses recursos, surge a necessidade de distribuir os clientes de forma inteligente dentre os servidores disponíveis. Essa tarefa é responsabilidade do Módulo de Escalonamento de Conexões que foi implementado no cliente do WSE-OS Escalável.

A implementação dos módulos foi desenvolvida utilizando a linguagem de programação Bash (*Bourne-Again SHell*) por essa oferecer os todos os recursos necessários de maneira simples e eficaz. A linguagem Bash permite a criação de scripts (.sh) que serão executados pelo sistema operacional (Linux) sem a necessidade de compilação (Johnson, 2009).

Os módulos desenvolvidos foram compostos por meio da junção de aplicativos já existentes em um script Bash, que coordena e combina as entradas e saídas desses aplicativos oferecendo uma ferramenta com funções específicas.

5.3 CLIENTE WSE-OS ESCALÁVEL

O cliente do WSE-OS Escalável utiliza o mesmo conceito de seu antecessor, realizando apenas os ajustes necessários para o funcionamento apropriado com *n* servidores.

A Figura 6 exibe uma visão simplificada dos elementos principais que compõem o cliente. Ao usuário são visíveis apenas as interfaces do WSE-OS Escalável do NX Client.

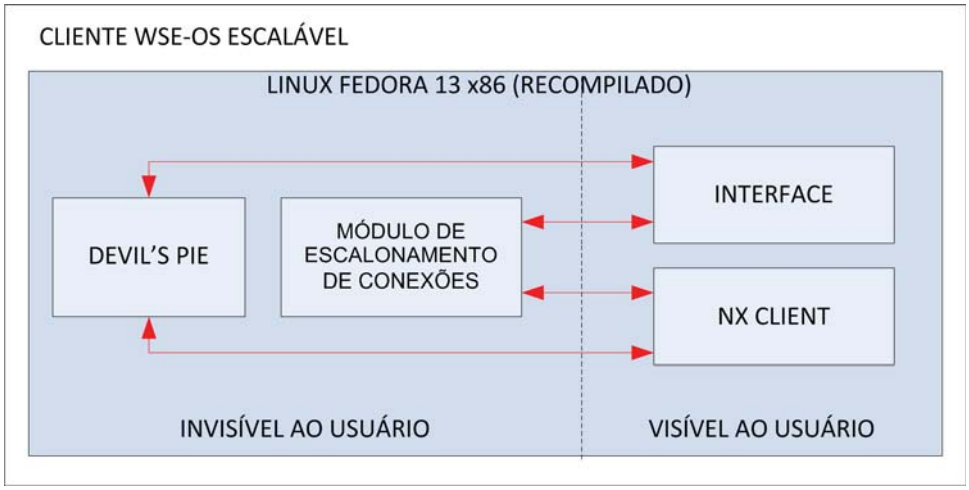

Figura 6 - Estrutura simplificada do cliente WSE-OS Escalável

Como no WSE-OS, o aplicativo Devil's Pie executa em segundo plano controlando as janelas criadas e garantindo que o usuário tenha acesso apenas ao conteúdo pertinente à operação do WSE-OS Escalável. A utilização deste aplicativo garante segurança ao sistema e eficiência ao usuário (CREPALDI, 2011).

O WSE-OS Escalável funciona apresentando uma interface contendo informações sobre os servidores disponíveis e opções de configurações que serão utilizadas para criar o arquivo de conexão ao servidor. Por sua vez, o NX Client utiliza o arquivo de conexão criado para criar uma sessão de conexão segura com o servidor.

A função do NX Client é realizar o acesso ao servidor criando uma Área de Trabalho Remota (VDI) onde o usuário possa operar a máquina virtual selecionada. Crepaldi (2011) descreve o funcionamento do NX Client assim:

> O NXClient está modelado para instanciar uma imagem de um sistema operacional no servidor WSE-OS. Isto faz com que o cliente seja denominado *thin client*, ou seja, a partir da execução deste módulo cria-se a comunicação baseada em TCSC em modo gráfico (CREPALDI, 2011).

5.3.1 Interface do Cliente WSE-OS Escalável

Com a mudança na abordagem do sistema de 1 para *n* servidores, foram necessários ajustes na interface para atender satisfatoriamente as novas características implementadas no O WSE-OS Escalável.

A interface do WSE-OS foi desenvolvida em bash utilizando chamadas XDialog para realizar a construção de janelas baseadas no X Window System 11 e oferecer uma interface amigável ao usuário (CREPALDI, 2011). A interface desenvolvida por Crepaldi (2011) atende ao WSE-OS satisfatoriamente, mas é insuficiente ao WSE-OS Escalável pela falta de dinamismo da linguagem utilizada.

O WSE-OS Escalável teve sua interface desenvolvida na linguagem PHP, por essa apresentar as características necessárias para um funcionamento mais objetivo e agradável ao usuário. O PHP foi escolhido pela facilidade de programação, por permitir uma passagem de comandos direto da interface para o sistema operacional e pala habilidade de criar uma interface dinâmica e atrativa.

Foram necessárias as instalações de alguns aplicativos para o funcionamento adequado da nova interface. Os principais são:

- PHP (php.i686 - versão 5.3.6-1.fc13) - é o núcleo da linguagem PHP e possui a função de permitir que a linguagem seja executada no cliente;
- Apache (httpd.i686 - versão 2.2.17-1.fc13.1) - possui a função de servidor HTTP e é responsável pela execução da interface PHP;

A execução da interface PHP ocorre no navegador Mozilla Firefox (firefox.i686 - versão 10.0.2-1.fc13.remi) que é parte do pacote de aplicativos básicos do sistema operacional do *middleware* (Fedora 13).

O navegador Firefox teve dois complementos (*add-on*) adicionados e habilitados. O primeiro executa o navegador somente no modo tela cheia (*fullscreen*), impedindo que o usuário acesse áreas não pertinentes ao WSE-OS Escalável. O segundo não permite que o navegador tenha acesso a Internet, acessando apenas o própria interface que está localizada em um diretório do servidor local (*localhost*).

A interface está dividida em duas áreas principais (Figura 7). A primeira possui opções de otimização do desempenho da ferramenta e inserção do usuário. A segunda conta com informações sobre o status da conexão e dos servidores.

Figura 7 - Interface do cliente WSE-OS Escalável.

As "Opções Avançadas" da Figura 7 são opções já apresentadas por Crepaldi (2011) no cliente do WSE-OS original e têm por finalidade permitir a otimização do desempenho do cliente. As opções de configuração de memória *cache* e compressão de imagens foram mantidas no WSE-OS Escalável por terem sido comprovadas como eficientes em Crepaldi (2011).

O botão "Conectar" tem a finalidade de submeter os valores do campo "Usuário" e dos campos da seção "Opções Avançadas" para o Módulo de Escalonamento de Conexões a fim de processá-las.

O cadastramento de usuário e senha é realizado pelo Gerente da ferramenta WSE-OS Escalável mediante solicitação presencial do usuário. A ferramenta não conta com um meio de cadastramento eletrônico de novo usuário a partir do cliente para evitar que pessoas não autorizadas utilizem o sistema.

A área "Status" da Figura 7 foi desenvolvida exclusivamente para o WSE-OS Escalável com a finalidade de apresentar as informações obtidas por meio do funcionamento do Módulo de Escalonamento de Conexões (ModEsc). As informações estão contidas em um *iframe* e são atualizadas automaticamente em um determinado intervalo de tempo para refletir as mudanças ocorridas na rede e informar o usuário em caso de falhas. O "Status" apresenta o IP do cliente ou o notifica se o mesmo estiver desconectado da rede. Apresenta a quantidade de servidores disponíveis na rede e indica o servidor recomendado e selecionado pelo ModEsc para a conexão.

A habilidade de atualizar apenas o conteúdo necessário, de maneira simples e eficiente foi um dos principais fatores que levaram a utilização da linguagem PHP combinada ao HTML.

O botão "Desligar" inicia um *script* Bash que realiza o desligamento de forma segura do computador.

5.3.2 Módulo de Escalonamento de Conexões

A escalabilidade proposta requereu uma integração de *hardware* e *software* para atingir seus objetivos. O WSE-OS Escalável foi implementado de maneira que os clientes sejam responsáveis pelo escalonamento das conexões aos servidores.

O escalonamento de conexões consiste em distribuir de maneira apropriada os clientes pelos servidores disponíveis de acordo com os critérios estabelecidos no desenvolvimento. A implementação, no cliente, de um módulo capaz de escalonar as conexões foi a maneira encontrada para obter benefício da estrutura com *n* servidores criada.

O ModEsc consiste numa série de arquivos e aplicativos gratuitos funcionando em conjunto para oferecer um melhor desempenho ao sistema e maior comodidade ao usuário. A sua composição está baseada na interoperabilidade de alguns aplicativos presentes tanto no cliente como no servidor. O módulo está localizado no cliente, mas há a necessidade da utilização de aplicativos situados no servidor para a obtenção de um resultado mais satisfatório.

O núcleo do ModEsc é um *script* Bash ("decideserver.sh") que funciona da seguinte maneira:

- O script é iniciado automaticamente pelo sistema operacional;
- Localiza e armazena seu IP. Caso o cliente não possua um IP, o *script* é reiniciado;
- Utilizando seu IP, executa o aplicativo Nmap para escanear a rede e encontrar os dispositivos conectados a ela e os armazena;
- Gera uma matriz (matriztodos) contendo todos os IPs encontrados, exceto o próprio;
- Percorre os elementos da matriz em busca de servidores via ssh. Quando um servidor é localizado, seu IP, quantidade de sessões ativas do VirtualBox e a porcentagem de memória RAM disponível são armazenados criando uma nova matriz bidimensional

(matrizservers). Caso nenhum servidor for encontrado, a execução do *script* é reiniciada;

- O algoritmo percorre a matriz que contém informações dos servidores. Ordena os servidores de acordo com três critérios respectivamente estabelecidos: menor quantidade de sessões ativas do Virtualbox, maior porcentagem de memória RAM disponível e menor número de IP. Após a ordenação é eleito o servidor indicado e armazenado seu IP;
- O algoritmo compara o servidor recém indicado com o servidor indicado anteriormente e em caso de divergência, altera o arquivo de conexão inserindo o mais recente IP recomendado;
- O script "decideserver.sh" reinicia sua execução.

O aplicativo Nmap ("*Network Mapper*") é uma ferramenta gratuita de código aberto usada para descobrir vulnerabilidades, serviços e computadores em uma rede (NMAP, 2013). No WSE-OS Escalável, o Nmap é utilizado para encontrar os servidores ativos na rede.

Outro dois arquivos interagem ativamente compondo o ModEsc. O arquivo processa.php é executado por meio do comando "Conectar" presente na interface, sendo responsável por processar as alterações realizadas pelo usuário e combiná-las com as alterações realizadas pelo núcleo do ModEsc.

Por sua vez, o *script* abrenx.sh atua como monitor, aguardando por alterações realizadas pelo processa.php ao arquivo de conexão. Quando detectada uma alteração, é criado um arquivo de sessão e o NX Client é executado para utilizá-lo.

Em resumo, o ModEsc modifica o arquivo padrão de conexão (default.nxs) do NX Client de acordo com a entrada de dados do usuários e dos seus *scripts*. Tais modificações geram um arquivo de sessão que visa a conexão ao servidor mais indicado para usufruir do ambiente escalável da ferramenta. A Figura 8 ilustra o fluxo de execução do ModEsc.

Foi desenvolvido um *script* (resolution.sh) que é executado uma única vez ao iniciar o sistema. Ele detecta a resolução nativa do dispositivo do cliente e a armazena para que a seja utilizada pelo arquivo processa.php.

O arquivo default.nxs foi disponibilizado na ferramenta WSE-OS, já propriamente configurado para atender ao seu projeto. Contudo, o WSE-OS Escalável requereu a alteração em algumas opções:

- opção que executa o módulo de Controle de Privilégios do servidor;
- opção que define a porta de conexão;
- opção que define altura da resolução;
- opção que define largura da resolução.

O Anexo B apresenta o código-fonte do arquivo decideserver.sh.

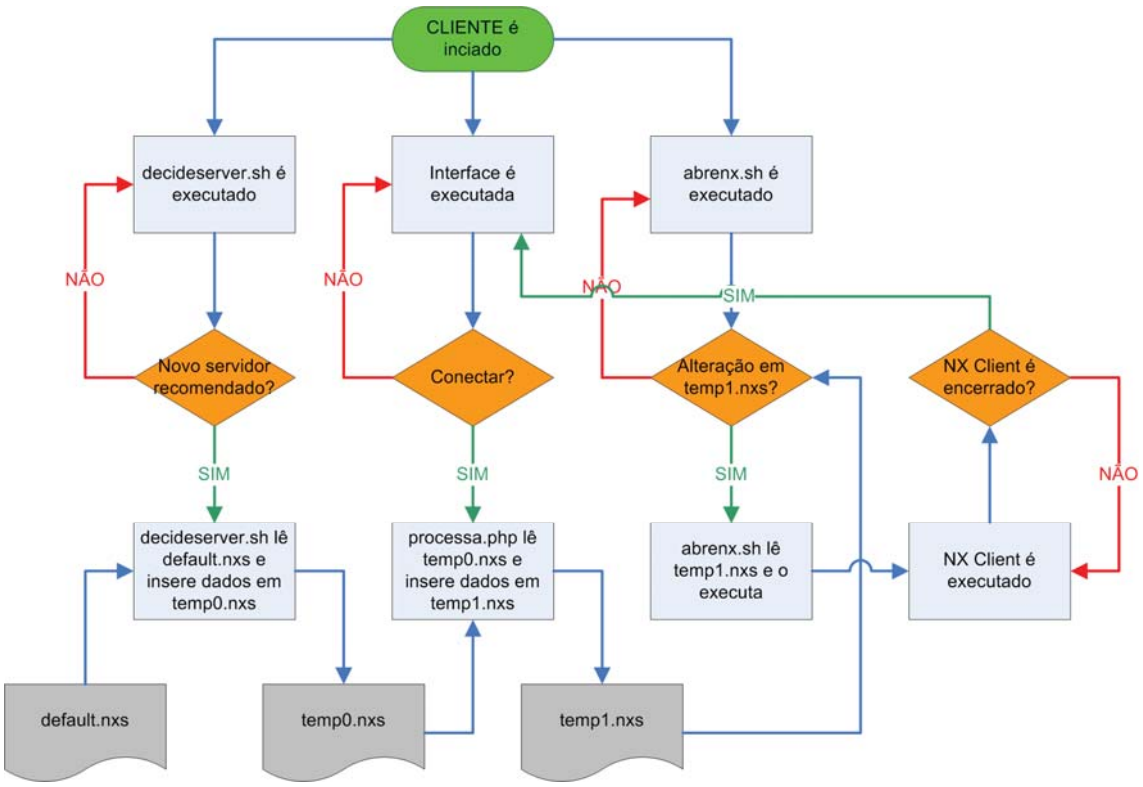

Figura 8 - Fluxograma de execução do ModEsc do cliente WSE-OS Escalável

5.4 SERVIDOR WSE-OS ESCALÁVEL

A alteração na estrutura do WSE-OS para *n* servidores, tornou necessário a inclusão de novos elementos e alteração em alguns dos existentes.

O servidor WSE-OS Escalável manteve a estrutura básica de funcionamento apresentada por Digiere (2011) que é composta pelos módulos de Controle de Privilégios, Mecanismo de Virtualização e Comunicação Servidor. Entretanto, foram necessários a inclusão do Módulo de Replicação de Dados (ModRep) e de outras ferramentas administrativas para facilitar e organizar a administração do sistema.

Apenas pequenas alterações foram realizadas no *script* do módulo de Controle de Privilégios para adequá-lo visualmente ao WSE-OS Escalável. Os arquivos compõem o módulo foram movidos para o diretório padrão (*/bin/wseos/*) da ferramenta a fim de obter uma organização maior.

Para que ocorra o funcionamento adequado, é necessário que todos os servidores do sistema possuam o WSE-OS Escalável instalado, já que os servidores devem ser todos idênticos quanto ao *software*.

A opção por servidores idênticos foi tomada para facilitar o processo de inserção de novos servidores. A opção em não diferenciar nenhum dos servidores, permite que a instalação do WSE-OS Escalável seja a única tarefa anterior a utilização do mesmo que o administrador necessite realizar. A atribuição de IP é feita de forma automática e o nome dos servidores na rede são indiferentes. A única identidade visível ao usuário é o IP da máquina, que é temporário.

Como no WSE-OS, os módulos Mecanismo de Virtualização e Comunicação Servidor não sofreram alterações por atenderem aos requisitos do WSE-OS Escalável.

5.4.1 Administração do Servidor WSE-OS Escalável

O servidor WSE-OS Escalável trata-se de uma instalação comum do Linux Fedora 13 x86 acrescida de atualizações e módulos necessários para o cumprimento de sua função.

Observando a estrutura do WSE-OS foi concluído que havia necessidade da criação de uma estrutura que definisse níveis de usuários. O WSE-OS Escalável definiu uma estrutura com 3 níveis de usuários: *root*, gerente e usuários.

O nível *root* é padrão do sistema operacional Fedora e tem acesso total a todos os arquivos. Este nível de usuário é necessário para a instalação de aplicativos (incluindo a ferramenta WSE-OS Escalável) e realização de reparos e ajustes na ferramenta quando necessário.

O nível Gerente é criado durante a instalação do WSE-OS Escalável. Ele tem a finalidade de gerenciar os usuários da ferramenta e administrar seus arquivos relacionados às Máquinas Virtuais. Foi criado na *desktop* do Gerente um arquivo que dá acesso a interface de gerenciamento de usuários.

O nível usuários é composto pelo usuários que acessam a ferramenta utilizando o cliente do WSE-OS Escalável. A criação desses usuários são feitas pelo Gerente mediante solicitação do usuário.

O aplicativo Eiciel (eiciel.i686 versão 0.9.6.1-3.fc12) foi instalado para servir como ferramenta auxiliar ao *root* e ao Gerente. O Eiciel é um aplicativo que permite de modo visual que se modifique as permissões ACL dos arquivos. Lista de Controle de Acessos (ACL) permitem a extensão nas permissões de acesso aos arquivos, para além das permissões padrões do Linux. O uso de ACL na ferramenta WSE-OS Escalável permite dar acesso total ao Gerente aos arquivos de todos os usuários sem alterar os proprietários originais (SHARMA, 2008).

5.4.2 Módulo de Replicação de Dados do Servidores

Toda alteração nos usuários do sistema, assim como em seus arquivos devem ser replicadas a todos os outros servidores do sistema. Tal exigência define a necessidade da implementação de um *software* capaz de realizar a replicação mantendo a consistência entre as réplicas.

O funcionamento do ModRep consiste no monitoramento ativo na ocorrência de alterações nos arquivos sensíveis ao WSE-OS Escalável. Quando uma alteração é detectada, inicia-se um processo de localização dos demais servidores ativos e faz-se a replicação dos dados divergentes a eles.

O ModRep combina aplicativos gratuitos de terceiros por meio de *scripts* Bash formando uma ferramenta única. O arquivo principal para o funcionamento do ModRep é o *script* "script.sh", responsável pelo monitoramento das alterações e replicação dos dados.

O ModRep conta ainda com o outro *script*, o "daemon.sh", que é iniciado junto ao sistema operacional e funciona por meio de um laço de repetição responsável por executar continuamente o *script* principal ("script.sh") e os *scripts* auxiliares "contasessoes.sh" e "freeram.sh". Os dois últimos são responsáveis em prover dados que serão utilizados principalmente pelo Módulo de Escalonamento de Conexões.

O "script.sh" é o núcleo do ModRep e funciona da seguinte maneira:

- O script é iniciado automaticamente pelo *script* "daemon.sh";
- Verifica se o seu *status* permite replicação. Caso não permita o *script* é reiniciado;
- Carrega a lista contendo os arquivos a serem monitorados;
- Compara os arquivos monitorados procurando por alterações entre o estado atual e sua verificação armazenada. Caso não haja alterações o *script* é reiniciado;
- Localiza e armazena seu IP. Caso o cliente não possua um IP, o *script* é reiniciado;

- Utilizando seu IP, executa o aplicativo Nmap para escanear a rede e encontrar os dispositivos conectados a ela e os armazena;
- Gera uma matriz (matriztodos) contendo todos os IPs encontrados, exceto o próprio; Caso nenhum outro IP seja encontrado, o *script* é reiniciado;
- Percorre os elementos da matriz em busca de servidores via SSH. Quando um servidor é localizado, seu IP é armazenado em um nova matriz (matrizservers). Caso nenhum outro servidor seja encontrado, o *script* é reiniciado;
- Percorre a matriz dos servidores e os trava para impedir que o gerente efetue modificações durante o recebimento de uma réplica;
- Percorre a matriz dos servidores e executa o aplicativo Rsync, para realizar a replicação dos dados sensíveis a todos os servidores;
- Percorre a matriz dos servidores e os destrava para permitir os servidores voltem ao seu estado normal.

O *script* foi desenvolvido de forma simples e objetiva, com funções bem definidas que são chamadas pelo execução principal do programa. O uso de troca de mensagens entre os servidores e arquivos sinalizadores (*flags*) foram a maneira encontrada para tentar minimizar as falhas que afetam qualquer sistema distribuído. A Figura 9 ilustra o fluxo de execução do ModRep.

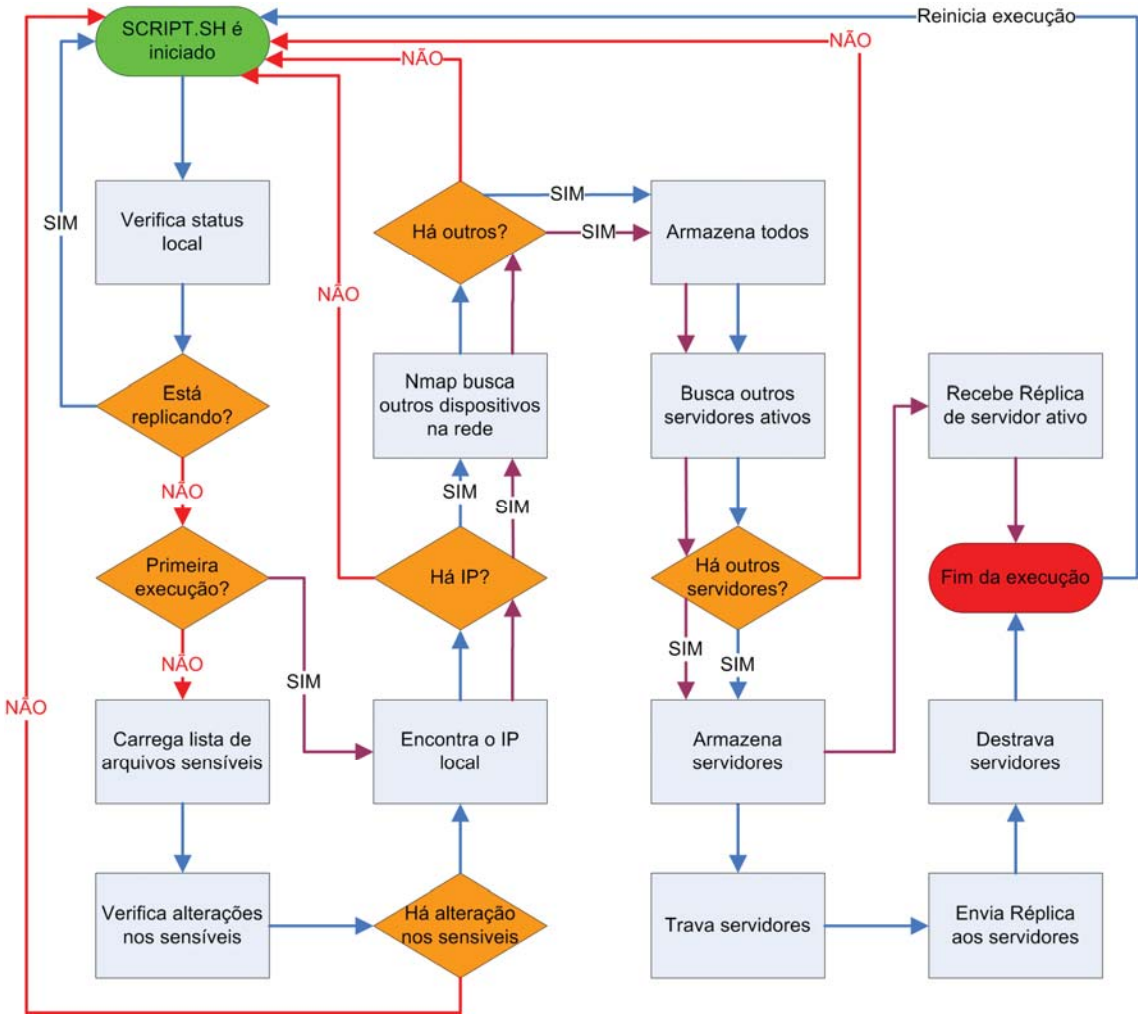

Figura 9 - Fluxograma de execução do ModRep do servidor WSE-OS Escalável

Nesse intuito, a criação de uma função que detecte quando um servidor é iniciado serve como aliada na tolerância a falhas. A função verifica se é a primeira execução desde que o sistema foi reiniciado e caso seja, ele solicita os dados (por replicação) de outro servidor ativo na rede. As setas roxas da Figura 9 indicam o fluxo de execução diferenciado dessa função.

A estrutura desenvolvida não prevê a existência de um servidor mestre ou primário. Cada servidor é responsável apenas por seus próprios dados, até o momento em que uma alteração local é detectada. O servidor que detecta uma alteração, irá travar os demais servidores ativos e propagar sua réplica a todos os outros servidores, um por vez. A destrava de todos só ocorre após o

último servidor ter recebido a réplica para evitar que um outro servidor inicie um processo de replicação antes do processo atual ser finalizado.

Digiere (2011) desenvolveu o servidor WSE-OS para operar com máquinas virtuais imutáveis com a finalidade de permitir que diversas máquinas possam utilizar uma mesma ISU concorrentemente.

Para o WSE-OS Escalável, a utilização de várias ISU em modo imutável favorece a administração do sistema, pois as únicas alterações possíveis de ocorrer, podem apenas serem causadas pelos administradores (*root* e Gerente) da ferramenta. A centralização nas tarefas administrativas criada, auxiliam em manter a consistência das réplicas.

O aplicativo Rsync é uma ferramenta gratuita e de código aberto que possui a função de realizar transferências entre arquivos de uma rede de forma bastante eficaz e segura (RSYNC, 2013).

O Anexo A apresenta o código-fonte do arquivo script.sh.

5.5 CONSIDERAÇÕES FINAIS

A ferramenta WSE-OS Escalável expande as capacidades de seu antecessor mantendo seus benefícios já existentes.

Esta ferramenta busca solução para melhorar a segurança e desempenho do WSE-OS por meio do uso de técnicas de escalonamento de conexões e replicação de dados dos servidores.

A centralização dos dados é mantida, só que de forma mais organizada do que a atualmente existente. A existência de *n* servidores permitem maior disponibilidade e escalabilidade ao sistema e podem proporcionar uma experiência mais agradável aos seus usuários.

6 EXPERIMENTOS

Este capítulo apresenta os resultados dos experimentos realizados sobre o WSE-OS Escalável para criar uma expectativa de desempenho desta ferramenta. Os testes realizados buscaram indicar os limites da escalabilidade proposta por meio do uso da informação já existente em Crepaldi (2011) e Digere (2011) e combiná-las aos testes realizados por este trabalho.

O WSE-OS Escalável propõe aumento da escalabilidade da ferramenta WSE-OS por meio da possibilidade da utilização de *n* servidores. Estes testes foram realizados com o intuito de definir um limite para *n*, se possível.

Largura de banda da rede, memória RAM e processamento do servidor, são os três fatores limitantes ao desempenho do sistema WSE-OS Escalável.

Memória RAM e capacidade de processamento são limites locais, que definem a capacidade de um (1) único servidor, enquanto largura de banda define a capacidade global da ferramenta.

6.1 ANÁLISE DO DESEMPENHO DO WSE-OS

Digiere (2011) identificou problemas de desempenho quando a quantidade de memória RAM utilizada pelas máquinas virtuais ultrapassava 50% da memória RAM disponível ao sistema operacional do hospedeiro (servidor). O aplicativo Virtualbox apresenta alertas em sua interface durante a configuração da memória RAM da Máquina Virtual quanto a capacidade recomendada. Na versão (3.2.14) do Virtualbox que é utilizada pelo WSE-OS e pelo WSE-OS Escalável a recomendação é de 50%.

De posse dos dados de degradação apresentados por Digere (2011) referentes ao uso da memória RAM, define-se que 50% da memória de um servidor seja o limite recomendado para a utilização pelas Máquinas Virtuais do

WSE-OS Escalável. Por exemplo, um servidor com 8GB de memória RAM deve no máximo disponibilizar 4GB para todas as Máquinas Virtuais.

6.2 WSE-OS X WSE-OS ESCALÁVEL: COMPARATIVO DE *BOOT*

Com a finalidade de avaliar o desempenho entre as ferramentas WSE-OS e WSE-OS Escalável foi realizada uma medição comparando (tempo médio e desvio padrão) o *boot* das duas ferramentas em um mesmo dispositivo de *hardware*.

Os experimentos foram realizados pela medição do tempo de *boot* por meio de cronometragem. Foram medidos o tempo decorrido desde a seleção da mídia de *boot* até o carregamento da Interface por completo. Os resultados foram obtidos após 10 medições com cada ferramenta. A Tabela 1 apresenta a configuração do computador utilizado nos testes.

Tabela 1 - Configuração do Ciente 01

Cliente	Processador	Memória RAM	Chipset	Rede	Método de boot	Tipo de dispositivo
Cliente 01	Core 2 Duo T5750	4GB DDR2	Intel Mobile GM965 Express	Intel Wireless WiFi Link 4965AGN	Leitor interno de SD	Notebook

A Tabela 2 apresenta os resultados obtidos na comparação do tempo de *boot* entre as ferramentas WSE-OS e WSE-OS Escalável utilizando o Cliente 01 da Tabela 1.

Tabela 2 - *boot* do WSE-OS x WSE-OS Escalável

Cliente	Ferramenta	Tempo médio de *boot* (s)	Desvio padrão
Cliente 01	WSE-OS	32,32	0,69
Cliente 01	WSE-OS Escalável	42,81	1,38

A ferramenta WSE-OS Escalável é, em média, 10,49 segundos ou 32,4% mais lenta durante o *boot* do que a ferramenta WSE-OS. O WSE-OS Escalável foi desenvolvido com mais funcionalidades que o WSE-OS.

Funcionalidades essas que requerem a inicialização de *scripts* junto ao Sistema Operacional do cliente, afetando o seu desempenho de *boot*.

6.3 ANÁLISE DO *BOOT* DO CLIENTE WSE-OS ESCALÁVEL

O cliente WSE-OS Escalável sofreu várias modificações com a inclusão do ModEsc. Com o intuito de estabelecer uma relação entre desempenho de *boot* do cliente e a configuração de *hardware* utilizada, foram medidos os tempos de *boot* dos clientes apresentados na Tabela 3.

Os experimentos foram realizados pela medição do tempo de *boot* por meio de cronometragem. Foram medidos o tempo decorrido desde a seleção da mídia de *boot* até o carregamento da Interface por completo. Os resultados foram obtidos após 10 medições com cada ferramenta. O cliente utilizado nos testes estava instalado em um cartão SD da Sandisk de 8GB Class 4.

Tabela 3 - Configuração de *hardware* dos clientes utilizados

Cliente	Processador	Memória RAM	Chipset	Rede	Método de *boot*	Tipo *de* PC
Cliente 02	Core 2 Duo T5750	4GB DDR2	Intel Mobile GM695 Express	Intel PRO/Wireless 3945ABG	Leitor interno de cartão SD	Notebook
Cliente 03	Core 2 Duo T5750	4GB DDR2	Intel Mobile GM695 Express	Intel PRO/Wireless 3945ABG	ChipsBnk Multi-Reader USB Device	Notebook
Cliente 04	Core 2 Duo T5750	4GB DDR2	Intel Mobile GM695 Express	SMCWUSB-G	ChipsBnk Multi-Reader USB Device	Notebook
Cliente 05	Core 2 Duo T5750	4GB DDR2	Intel Mobile GM695 Express	SMCWUSB-G	Leitor interno de cartão SD	Notebook
Cliente 06	Core i7 2620M	4GB DDR3	Intel HM65 Express	SMCWUSB-G	ChipsBnk Multi-Reader USB Device	Notebook
Cliente 07	Core 2 Quad Q6600	4GB DDR2	Intel G31Express	SMCWUSB-G	ChipsBnk Multi-Reader USB Device	Desktop

Alguns clientes utilizaram placas de rede sem-fio USB externas, pela razão da placa interna não funcionar automaticamente com o cliente. Por incapacidade de operar ou por inexistir leitor de cartão SD interno, alguns clientes utilizaram um leitor de cartão SD USB.

A Tabela 4 apresenta os resultados obtidos pelos clientes descritos na Tabela 03.

Tabela 4 - Resultados obtidos pelos clientes da Tabela 3

Cliente	Cliente 02	Cliente 03	Cliente 04	Cliente 05	Cliente 06	Cliente 07
Tempo médio de *boot*	39,76	44,56	44,23	41,87	41,94	39,66
Desvio Padrão	0,39	0,91	0,85	0,31	1,26	1,84

Os clientes 02 e 03 da Tabela 3 possuem a mesma configuração de *hardware*, divergindo apenas quanto ao método de *boot* utilizado. O cliente 02 utilizando o Leitor interno de SD realizou o *boot* em tempo médio 39,76 segundos com um desvio padrão de 0,39. O cliente 03 executou a mesma tarefa em 44,56 segundos (desvio padrão de 0,91), sendo 12% mais lento que o cliente 02 utilizando um leitor de cartão SD conectado a sua porta USB (ChipsBnk Multi-Reader USB Device). Essa comparação prova que a escolha no método de *boot* influencia diretamente no desempenho do *boot* do cliente da ferramenta WSE-OS Escalável.

Os clientes 02 e 05 da Tabela 3 possuem a mesma configuração de *hardware*, divergindo apenas quanto ao dispositivo de rede utilizado. O cliente 02 utiliza a placa de rede interna (Intel PRO/Wireless 3945ABG), enquanto o cliente 05 utiliza uma placa de rede conectada a porta USB (SMCWUSB-G). O cliente 05 realizou o *boot* em tempo médio de 41,87 segundos, sendo 5,3% mais lento que o cliente 02. Os resultados sugerem que a utilização do dispositivo de rede USB impactou negativamente no desempenho do Cliente 05.

Os clientes 04, 06 e 07 da Tabela 3 apresentam mesma configuração de dispositivo de rede e o mesmo método de *boot*, mas divergem quanto aos demais itens. Processador, memória RAM, *chipset* e tipo de PC são os elementos principais para determinar o desempenho de um computador. O

cliente 04 obteve o pior desempenho entre os três (44,23 segundos), sendo 5,4% mais lento que o cliente 06 e 11,5% mais lento que o cliente 07. A diferença entre os clientes se dá pelo poder de processamento dos computadores testados como esperado. O cliente 07 possui desempenho melhor que 04 e 06 por se tratar de um *Desktop*. A diferença entre o desempenho dos clientes 04 e 06 é menor por ambos serem *Notebooks*. Entretanto, a vantagem ainda favorece o cliente 06 que dispõe de tecnologia mais nova e poderosa que a do cliente 04.

Observando o impacto negativo dos dispositivos USB e o impacto positivo no uso de melhores processadores, conclui-se que o poder de processamento do cliente é o fator mais relevante ao desempenho do sistema durante o *boot*.

6.4 ANÁLISE DA REPLICAÇÃO DO WSE-OS ESCALÁVEL

A maneira escolhida para desenvolver o ModRep propõe a manutenção da consistência como apertada, mas de modo que o impacto no desempenho dos clientes seja o menor possível. A replicação ocorre sempre de maneira que apenas 1 servidor envia a cópia para apenas 1 outro, onde o servidor enviando os dados, primeiro envia tudo a um servidor, e só depois irá iniciar o envio para um outro servidor. Tal método foi escolhido por considerar a replicação mais segura em caso de falhas.

Para medir a quantidade de tempo decorrido durante a replicação dos dados entre os servidores, foram realizados testes com essa finalidade. Os teste consistiam em iniciar um processo de replicação em um servidor com arquivos de diferentes tamanhos e analisar os registros (log) do ModRep para obter o tempo levado por cada tarefa. Não havia clientes conectados a rede quando os testes foram realizados.

Os testes de replicação foram realizados com 4 (quatro) servidores idênticos em *hardware* com as configurações de acordo com o Servidor 01 da

Tabela 5. O roteador utilizado para os testes foi o TP-Link Gigabit Wireless N TL-WR1043ND. O cabeamento utilizado foi no padrão UTP CAT 5e.

Tabela 5 - Configuração do Servidor 01

Cliente	Servidor 01
Processador	Intel Core i5-760
Chipset	Intel H55
Memória RAM	4GB DDR3
GPU	Nvidia Geforce GT220
Rede	Gigabit Realtek RTL8111E
Disco Rígido (HD)	Westen Digital WD10EARS-00Y SATA2 1TB

Os valores foram obtidos a partir do cálculo da média de 10 execuções de cada Configuração do teste da Tabela 6. As colunas da Tabela 6 representam as diferentes configurações de cenários de teste. Os arquivos utilizados para as transferências eram os mesmos em todos os casos.

Tabela 6 - Tempo de execução de replicação sem clientes

Configuração do teste	A	B	C	D	E	F	G	H	I
Número de servidores recebendo dados	1	1	1	2	2	2	3	3	3
Tamanho do arquivo (MB)	0,05	2600	5200	0,05	2600	5200	0,05	2600	5200
Tempo médio levado (s)	5	67	130	7	131	257	8	191	379
Velocidade média da transferência (MBps)	0,01	38,81	40,00	0,01	39,69	40,47	0,01	40,84	41,16

Analisando os dados obtidos na Tabela 6, observa-se que o tempo necessário para um ciclo de execução/replicação do script.sh, que faz parte do ModRep é de 5 segundos. Este valor corresponde ao tempo médio utilizado pela Configuração de teste A. Os valores de tempo são obtidos a partir do arquivo registro do ModRep e somente os fornece em números inteiros.

Os valores médios de tempo apresentados na Tabela 6 foram arredondados para o número inteiro abaixo. Contanto, pode-se afirmar que o tempo médio extra necessário para cada novo servidor é entre 1 e 2 segundos. Essa conclusão é tomada após analisar os tempos das Configurações D e G.

Há um aumento de desempenho que favorece o uso de transferências maiores para um mesmo servidor. Na Configuração de teste C há um aumento de 3,08% em relação a B que faz uma transferência menor.

Entre as Configurações de teste B, E e H houve aumento nas velocidades médias gradativamente, assim como nas Configurações de teste com transferências de 5200MB (C, F e I). O aumento proporcional das velocidades em Configurações de teste com maior número de servidores, é benéfico a proposta de escalabilidade deste trabalho. Tal aumento provavelmente se deve ao uso de memórias *cache* e de predição (*prefetch*) de caminhos por parte dos processadores.

As vantagens apresentadas nas transferências maiores e para mais servidores, satisfazem muito bem às necessidades do WSE-OS Escalável, principalmente porque as replicações na maioria das vezes irão replicar arquivos .vdi (ISU).

6.5 ANÁLISE DA DEGRADAÇÃO NA REDE DO WSE-OS ESCALÁVEL

O bom funcionamento das ferramentas WSE-OS e WSE-OS Escalável são dependentes do desempenho da rede, portanto faz-se necessário entender o impacto que o aumento no número de clientes causa na mesma.

O WSE-OS Escalável funciona por meio do envio e recebimento de dados entre cliente e servidor. O cliente se conecta ao servidor por meio de uma sessão remota do NX, sendo esta a grande responsável pelo consumo de banda.

Segundo NOMACHINE (2013), os usuários do NX Server necessitam de 20kbps de largura de banda para executar confortavelmente uma sessão pelo Linux. O valor recomendado pelo desenvolvedor para o uso de sessões do NX Server é bastante baixo. Equivale a 2,5 Kilobytes por sergundo (KBps) ou 0,002441MBps. Entretanto, é importante esclarecer que há uma distinção entre o NX Server da NOMACHINE (2013) e a versão livre do FREENX Server utilizada por esta ferramenta.

Utilizando os recursos disponíveis foi criado um cenário que busca simular a utilização real da ferramenta WSE-OS Escalável. Os testes foram realizados utilizando 2 computadores com a configuração do Servidor 01 da

Tabela 05 atuando como servidores e mais 25 computadores com configurações heterogêneas atuando como clientes. Os 27 computadores foram interligados por 2 *swtiches Gigabit* profissionais da D-Link e 3COM.

Após 10 execuções de cada cenário foram calculados o tempo médio e o desvio padrão dos valores obtidos. O teste consistiu em medir o tempo decorrido durante a replicação de um arquivo padrão de 2600MB do servidor A para o servidor B. O arquivo utilizado foi o mesmo utilizado pela Configuração do teste B da Tabela 6.

Para avaliar o impacto do número de clientes e da replicação de dados dos servidores sessões NX foram criadas progressivamente para que fosse possível observar o consumo de banda feito pelos clientes pela velocidade média da replicação. Foram testados com 0, 5, 10, 15, 20 e 25 sessões NX, onde 0 indica uma replicação sem clientes e é usado como base para a avaliação do consumo de banda.

Toda sessão NX criada executava o aplicativo gnome-system-monitor, aplicativo que indica graficamente a atividade do sistema operacional, para garantir uma atividade constante de troca de dados entre cliente e servidor com o intuito de representar o uso esperado do WSE-OS Escalável.

A Tabela 7 exibe os dados obtidos durante os testes. A velocidade média da replicação é obtida em Megabytes por segundo por meio da divisão do tamanho do arquivo replicado pelo tempo médio da transferência.

Tabela 7 - Impacto das sessões NX na rede

Replicação de 2600MB do servidor A para o B						
Quantidade de sessões NX	0	5	10	15	20	25
Tempo médio (s)	61,5	70,8	87,4	159,4	257,3	393,6
Desvio padrão	0,85	1,23	1,17	6,47	20,96	20,72
Velocidade média (MBps)	42,28	36,72	29,75	16,1	10,10	6,61
Consumo por sessão NX (MBps)	0	1,11	1,25	1,73	1,61	1,43

O consumo por sessão NX foi calculado usando a velocidade média com 0 sessões NX (42,28MBps) e subtraindo deste valor a velocidade média obtida para cada quantidade de sessões. Por exemplo, com 10 sessões, a velocidade média da replicação foi reduzida para 29,75MBps, criando uma redução de

12,53MBps. Esse valor foi dividido pelas 10 sessões NX, totalizando um consumo de 1,25MBps por sessão.

A Figura 10 exibe o gráfico que representa a variação da velocidade média em função da quantidade sessões NX ativas.

O gráfico indica uma variação anormal no acréscimo de tempo médio entre 10 e 15 sessões NX. Entre 5 e 10 sessões há aumento de 100% com apenas 23,4% de aumento no tempo médio de replicação. Entre 10 e 15 sessões há aumento de 50% na sessões NX seguido de aumento de 82,4% no tempo médio de replicação. Há um ponto crítico no cenário testado que aponta variação anormal entre 10 e 15 sessões NX, mas que se normaliza nos demais cenários analisados (com 20 e 25 sessões).

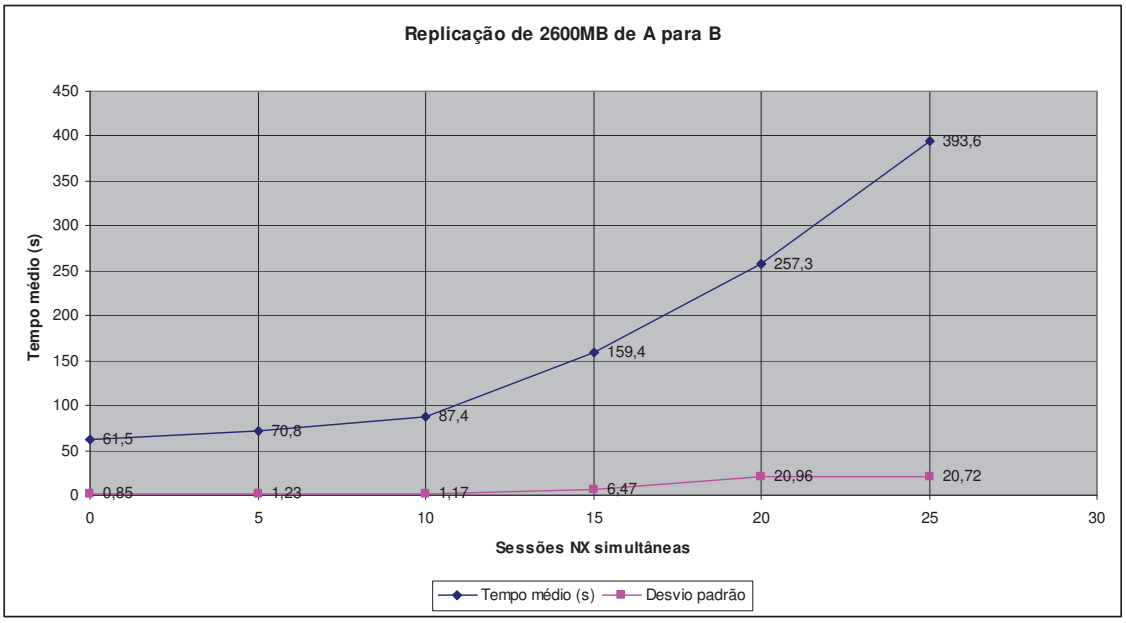

Figura 10 - Gráfico de consumo de banda das sessões NX

Com 15 sessões há um aumento do tempo médio de 82,4% em relação ao anterior, com 20 sessões o aumento é de 61,4% e com 25 sessões é de 53,0%. A Figura 11 ilustra com clareza a anomalia ocorrida entre 10 e 15 sessões, mas aponta uma estabilização no aumento para os demais cenários. É importante ressaltar que não há dados suficientes para afirmar qual o limite de sessões concorrentes. Embora a linha que representa o aumento no tempo

médio tenda a diminuir, a linha representando a quantidade de sessões NX também diminui, deixando de ser interessante por degradar demasiadamente a rede.

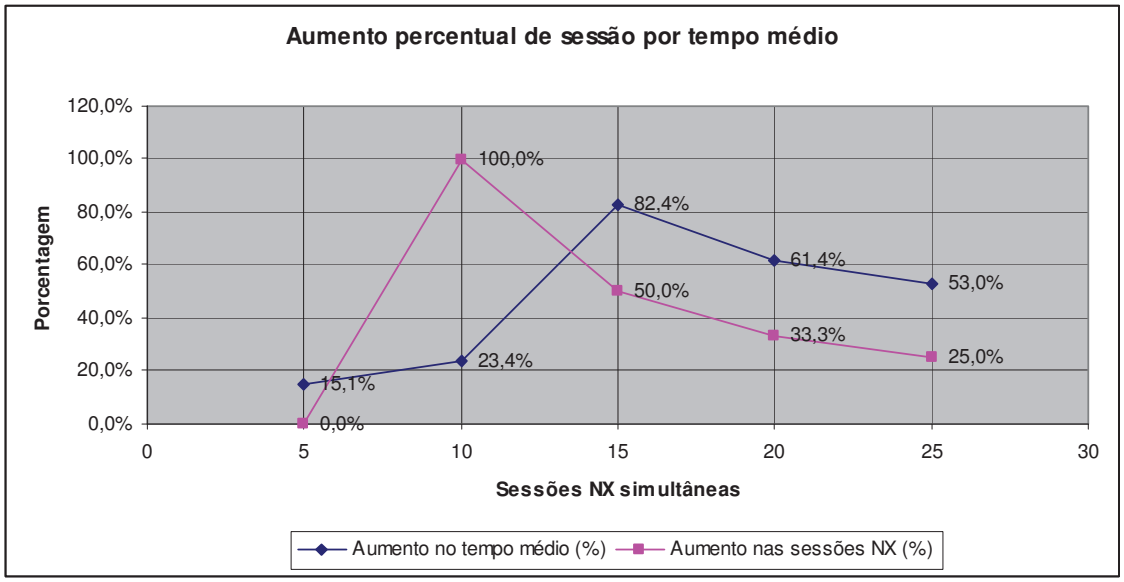

Figura 11 - Aumento de sessão NX por tempo médio em %

Observando a Figura 11, pode-se sugerir que a anomalia na replicação possa ter sido influenciada pelo desempenho ruim do servidor recebendo as sessões NX. A sobrecarga de tarefas no servidor podem ter causado gargalos e impedido um funcionamento mais eficaz da rede.

Diversas outras variáveis devem ser consideradas para a definição de um limites para as quantidades máximas de servidores (*n*) e clientes (*m*) suportados pela ferramenta WSE-OS Escalável. As principais variáveis são:

- Equipamento de rede utilizado (*switch*, *hub* e roteador);
- Desempenho dos servidores;
- Escalonamento das sessões NX.

Trabalhos futuros que disponham de mais recursos de *hardware* podem ser importantes na definição de limites mais concretos sobre a quantidade de sessões NX utilizadas.

6.6 ANÁLISE DO CONSUMO DE MEMÓRIA RAM PELAS SESSÕES NX

Foram medidos também o consumo de memória RAM por parte de cada sessão NX no servidor. A quantidade de RAM consumida por cada sessão não ou pouco aumentou a medida que mais sessões foram incluídas. O consumo medido está na Tabela 8. Foi utilizado o Cliente 02 da Tabela 3 para a realização dos testes.

Tabela 8 - Consumo de RAM por sessão NX

Nº sessões	RAM total consumida no dispositivo (MB)	Consumo de RAM por sessão (MB)
0	180	0
1	285	105
2	391	106
3	497	106
4	606	109
5	715	109
6	821	106
7	928	107
8	1038	110
9	1148	110
10	1257	109

A dados da Tabela 8 mostram que diferentemente das Máquinas Virtuais, as sessões NX têm um impacto menor sobre a utilização da memória RAM dos servidores, porém no WSE-OS Escalável uma sessão NX sempre estará obrigatoriamente ocorrendo em conjunto de uma Máquina Virtual.

6.7 CONSIDERAÇÕES FINAIS

Os experimentos realizados não são capazes de definir um limite para o número de servidores e clientes simultâneos utilizando a ferramenta WSE-OS Escalável.

Por outro lado, é possível definir que o limite recomendado para a execução satisfatória em um servidor é que o somatório das memórias RAM de cada máquina Virtual deve ser menor ou igual ao da memória do servidor. Lembrando que o poder de processamento desse servidor dever ser

compatível ao requerido por essas Máquinas Virtuais independente da quantidade de memória RAM para evitar degradação de desempenho.

Utilizando os valores informados pelos desenvolvedores, pode-se fazer uma estimativa da capacidade da ferramenta WSE-OS Escalável, são eles:

- A velocidade máxima teórica do padrão IEEE 802.11N é de 300mbps, que equivale a 37,5MBps;
- A velocidade para execução confortável de uma (1) sessão NX, segundo a NOMACHINE (2013), é de 0,002441MBps;

De acordo com os valores supracitados, seriam possíveis o valor máximo de 15 mil sessões NX simultâneas. Obviamente esse valor não pode ser atingido em um cenário real.

Os valores fornecidos pelos desenvolvedores das tecnologias são teóricos e diferem da realidade. A utilização da ferramenta WSE-OS Escalável requer utilização da rede pelos Sistemas Operacionais dos clientes e dos servidores e pelos módulos da ferramenta. Muitos outros fatores como latência, colisões, interferências e gargalos contribuem para a degradação da rede.

7 CONCLUSÕES

Este trabalho apresentou uma solução combinando técnicas de replicação de dados, tolerância a falhas e escalonamento de conexões para agregar funcionalidades à ferramenta WSE-OS existente que combina as tecnologias de VDI à virtualização para oferecer uma ferramenta otimizada para operar em redes sem-fio.

Apesar da mobilidade oferecida pelo WSE-OS, melhorias são necessárias. No sistema WSE-OS, os dados de todos os clientes são armazenados no único servidor existente e esse fator gera a necessidade de uma solução que garanta a integridades desses dados.

Este trabalho propôs melhorias no atual WSE-OS através da implementação de dois novos módulos: Módulo de Replicação de Dados dos Servidores e o Módulo de Balanceamento de Carga. Tais módulos permitem a inserção de novos servidores e da replicação de seus dados além de garantir que o cliente se conecte ao servidor com maior disponibilidade. A Replicação de Dados dos servidores no WSE-OS é primordial para assegurar a integridade dos dados armazenados e o modelo de consistência apertada escolhido é determinante para o bom funcionamento da ferramenta.

O WSE-OS Escalável é caracterizado pelo uso de ferramentas gratuitas e código-fonte aberto. A estrutura inicial para a implantação do WSE-OS Escalável é de baixo custo, ainda menor que o necessário pela WSE-OS. Tal fato se deve pelo WSE-OS Escalável poder contar com vários servidores, podendo assim iniciar o projeto com um de menor potência e em qualquer momento adicionar outros novos.

Os usuários contam com uma interface remodelada e de mais fácil acesso, proporcionando uma experiência mais agradável e eficiente. Os administradores contam com uma estrutura melhor definida e mais segura por meio de ferramentas auxiliares e melhores políticas de acesso.

Este modelo contribui significantemente no ambiente acadêmico, devido a sua simplicidade em distribuir um conteúdo para dispositivos heterogêneos.

7.1 CONTRIBUIÇÕES DESTE TRABALHO

A principal contribuição deste trabalho é a proposta de ampliação nas capacidades de escalabilidade do sistema WSE-OS, permitindo que mais clientes sejam conectados simultaneamente com a possibilidade da utilização de múltiplos servidores consistentes.

A alteração na estrutura, além da disponibilidade aumenta a confiabilidade do sistema, por evitar que falhas físicas em um servidor ocasionem na perda de todos os dados.

Este trabalho contribui na área acadêmica com um estudo bibliográfico sobre a técnicas de replicação de dados e de tolerância a falhas.

7.2 TRABALHOS FUTUROS

- Análise da adaptação do WSE-OS Escalável para execução em redes de maior alcance como, por exemplo, padrão IEEE 802.16 WiMAX;
- Adaptação do WSE-OS Escalável para uso através da Internet, permitindo aos usuários usufruir da execução de aplicativos que demandem grande poder de processamento como jogos e editores de vídeo;
- Análise da expansão dos critérios para seleção de servidores e aprimoramento do ModEsc;
- Extensão e adaptação do cliente para execução em dispositivos móveis com os sistemas operacionais Android e iOS, permitindo que usuários tenham acesso a um sistema operacional remotamente virtualizado;

- Aprimoramento na interface do servidor através da integração de novas ferramentas parar auxiliar nas criações de máquinas Virtuais e gerenciamentos das ISUs;
- Implementação de um módulo de Balanceamento de Cargas capaz de migrar a conexão do cliente e o estado de uma Máquina Virtual em execução para outro servidor mais disponível;
- Análise da ampliação da capacidade da rede por meio da inclusão de novos dispositivos e ampliação na lista de placas de redes suportadas pelo cliente; e
- Análise dos pontos fracos a fim de obter um aprimoramento nas políticas de tolerância a falhas da ferramenta.

8 REFERÊNCIAS BIBLIOGRÁFICAS

AHAMAD, M.; NEIGER, G.; BURNS, J.; KOHLI P.; HUTTO, P. **Causal Memory: Definitions, Implementation and Programming**. Georgia Tech Technical Report 93/54. 1993. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.3356 > Acesso em: 14 mar 2013.

ALVISI, L.; MARZULLO, K. 1998. **Message Logging: Pessimistic, Optimistic, Causal, and Optimal**. IEEE Trans. Softw. Eng. 24, 2 (February 1998), 149-159. DOI=10.1109/32.666828 http://dx.doi.org/10.1109/32.666828

AL-EKRAN, R.; HOLT, R. **Multi-consistency Data Replication**. In Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS '10). IEEE Computer Society, Washington, DC, USA, 568-577. DOI=10.1109/ICPADS.2010.67. 2010. Disponível em: < http://dx.doi.org/10.1109/ICPADS.2010.67 >. Acesso em: 16 nov 2012.

BALDONI, R.; BRZEZINSKI, J.; HELARY, J. M.; MOSTEFAOUI, A.; RAYNAL, M. **Characterization of consistent global checkpoints in large-scale distributed systems**. Distributed Computing Systems, 1995., Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of , vol., no., pp.314,323, 28-30 Aug 1995 doi: 10.1109/FTDCS.1995.525000

BIRREL, A.; NELSON, B. J. 1984**. Implementing remote procedure calls**. ACM Trans. Comput. Syst. 2, 1 (February 1984), 39-59. DOI=10.1145/2080.357392 http://doi.acm.org/10.1145/2080.357392

CHANDI, K. M.; LAMPORT, L. 1985. **Distributed snapshots: determining global states of distributed systems.** ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75. DOI=10.1145/214451.214456 http://doi.acm.org/10.1145/214451.214456

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. 2011. **Distributed Systems: Concepts and Design (5th ed.)**. Addison-Wesley Publishing Company, , USA.

CREPALDI, L. G. **Middleware de Comunicação entre Objetos Distribuídos para**
Gerenciamento de Computadores baseado em Redes Sem Fio (WSE-OS). 2011. Dissertação (Mestrado) - Universidade Estadual Paulista “Julio de Mesquita Filho”, Bauru.

CREPALDI, L. G.; DIGIERE, A.R.; SPOLON, R.; CAVENAGHI, M.A.; LOBATO, R. S. **A Management Tool for the Replication of Operating Systems in Wireless Communication Networks.** Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual , vol., no., pp.86-92, 18-22 July 2011
doi: 10.1109/COMPSAC.2011.19. Disponível em: < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6032328&isnumber=6032314 >. Acesso em: 12 fev 2013.

CRISTEA, V.; LAVINIA, A.; DOBRE C.; POP, F. A **Failure Detection System for Large Scale Distributed Systems**. Int. J. Distrib. Syst. Technol. 2, 3 (July 2011), 64-87. 2011. DOI=10.4018/jdst.2011070105 http://dx.doi.org/10.4018/jdst.2011070105

CRISTIAN, F. **Understanding fault-tolerant distributed systems**. Commun. ACM 34, 2 (February 1991), 56-78. 1991. DOI=10.1145/102792.102801 http://doi.acm.org/10.1145/102792.102801

CURIAC, D.; VOSENCU, C.; PESCARU, D.; JURCA, L.; DOBOLI, A. **Redundancy and its applications in wireless sensor networks**: a survey. W. Trans. on Comp. 8, 4 (April 2009), 705-714. 2009.

DIGIERE, A. R. **Camada de Gerenciamento para Comunicação entre Computadores Baseada em Redes Sem Fio (WSE-OS)**, 2011. Dissertação (Mestrado) - Universidade Estadual Paulista “Julio de Mesquita Filho”, Bauru.

ELNOZAHY, E. N.; ALVISI, L.; WANG, Y.; JOHNSON, D. B. 2002. **A survey of rollback-recovery protocols in message-passing systems**. ACM Comput. Surv. 34, 3 (September 2002), 375-408. DOI=10.1145/568522.568525 http://doi.acm.org/10.1145/568522.568525

JOHNSON, B. An introduction to the design and analysis of fault-tolerant systems. **InFault-tolerant computer system design.** 1996. Dhiraj K. Pradhan (Ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA 1-87.

JOHNSON, C. F. A. 2009. **Pro Bash Programming: Scripting the Gnu/Linux Shell (1st ed.)**. Apress, Berkely, CA, USA.

KOREN, I.; KRISHNA, C. M. 2007. **Fault-Tolerant Systems**. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

KSHEMKALYANI, A. D.; SINGHAL, M. 2008. **Distributed Computing: Principles, Algorithms, and Systems (1 ed.)**. Cambridge University Press, New York, NY, USA.

LAMPORT, L. "**How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs,**" Computers, IEEE Transactions on , vol.C-

28, no.9, pp.690-691, Sept. 1979 doi: 10.1109/TC.1979.1675439. 1979. Disponível em: < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1675439&isnumber=35185 >. Acesso em: 28 out 2012.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. **The Byzantine Generals Problem**.ACM Trans. Program. Lang. Syst. 4, 3 (July 1982), 382-401. 1982. DOI=10.1145/357172.357176 http://doi.acm.org/10.1145/357172.357176

LAMPORT, L. "**Email was sent by Leslie Lamport at 12:23:29 PDT on May 28, 1987, message ID 8705281923.AA09105@jumbo.dec.com, Subject: Distribution**." 1987. Disponível em: < http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt >. Acesso em: 28 mar 2013.

NELSON, B. J. 1981. **Remote Procedure Call**. Ph.D. Dissertation. Carnegie Mellon Univ., Pittsburgh, PA, USA. AAI8204168.

NG, P. **A commit protocol for checkpointing transactions**. Reliable Distributed Systems, 1988. Proceedings., Seventh Symposium on , vol., no., pp.22,31, 10-12 Oct 1988 doi: 10.1109/RELDIS.1988.25777

NIEH, J.; YANG, S. J.; NOVIK, N. **Measuring thin-client performance using slow-motion benchmarking**. ACM Trans. Comput. Syst. 21, 1 (February 2003), 87-115. DOI=10.1145/592637.592640. 2003. Disponível em: < http://doi.acm.org/10.1145/592637.592640 >. Acesso em: 24 dez 2012.

NMAP. **Nmap - Free Security Scanner For Network Exploration & Security Audits.** 2013. Disponível em: < http://nmap.org/ >. Acesso em 13 de julho de 2013.

NOMACHINE. 2013. **Article: #AR12B00119. How much bandwidth is required for each NX 3.5.0 user running a remote Linux session?.** Disponível em: < http://www.nomachine.com/ar/view.php?ar_id=AR12B00119 >. Acesso em 13 de julho de 2013.

OXFORD. **Oxford Online Dictionaries**, 2011. Disponível em: <http://oxforddictionaries.com/definition/mainframe?region=us >. Acesso em: 10 nov. 2012.

PARK, E.; EGGER, B.; LEE, J. 2011. **Fast and space-efficient virtual machine checkpointing**. SIGPLAN Not. 46, 7 (March 2011), 75-86. DOI=10.1145/2007477.1952694 http://dl.acm.org/citation.cfm?doid=2007477.1952694

PETERSON, L. L.; DAVIE, B. S. 2011. **Computer Networks, Fifth Edition: A Systems Approach (5th ed.)**. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

PRIBERAM. **Dicionário Priberam da Língua Portuguesa - Significado de replicar** 2011. Disponível em: < http://www.priberam.pt/dlpo/default.aspx?pal=replicar >. Acesso em: 10 nov 2012.

PYTHON. **Python Programming Language – Official Website**. 2012. Disponível em: < http://www.python.org/ >. Acesso em: 27 mar 2012.

RANDEL, B.; LEE, P.; TRELEAVEN, P. C. 1978. **Reliability Issues in Computing System Design**.ACM Comput. Surv. 10, 2 (June 1978), 123-165. DOI=10.1145/356725.356729 http://doi.acm.org/10.1145/356725.356729

RSYNC. 2013. **rsync**, Disponível em: < http://rsync.samba.org/ > Acesso em 13 de julho de 2013.

SAHA, G. K. 2005. **Software fault tolerance through run-time fault detection**. Ubiquity2005, December (December 2005), 2-2. DOI=10.1145/1115544.1115546 http://doi.acm.org/10.1145/1115544.1115546
SHARMA, S. 2008. **Smart ACL management with Eiciel**. June 18, 2008 (4:00:00 PM). DisponÍvel em: < http://archive09.linux.com/feature/138169 > Acesso em: 10 jun 2013.

SSH, "**Open SSH**". 2012. Disponível em: < http://www.openssh.com/ >. Acesso em: 24 mar 2013.

TANENBAUM, A. S.; VAN STEEN, M. **Distributed Systems: Principles and Paradigms (2nd Edition)**. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 2006.

TANENBAUM, A. S.; WETHERALL, D. J. 2010. **Computer Networks (5th ed.)**. Prentice Hall Press, Upper Saddle River, NJ, USA.

TERRY, D. B.; PETERSEN, K.; SPREITZER, M. J.; THEIMER, M. M. "**The Case for Non-transparent Replication: Examples from Bayou**". IEEE Data Engineering, December 1998, pages 12-20. 1998. Disponivel em: < http://www2.parc.com/csl/projects/bayou/pubs/dataeng-98/DataEngineeringDec98.frame.pdf > Acesso em:16 jul 2013.

VENNERS, B. 2002. **Designing Distributed Systems: A Conversation with Ken Arnold, Part III**. October 23, 2002. Disponivel em: < http://www.artima.com/intv/distrib.html > Acesso em: 10 jun 2013.

YU, H.; VAHDAT, A. **Design and evaluation of a conit-based continuous consistency model for replicated services**. ACM Trans. Comput. Syst. 20, 3

(August 2002), 239-282. DOI=10.1145/566340.566342. 2002. Disponível em: < http://doi.acm.org/10.1145/566340.566342 >. Acesso em: 16 nov 2012.

YU, H.; VAHDAT, A. **Consistent and automatic replica regeneration**. Trans. Storage 1, 1 (February 2005), 3-37. 2005. DOI=10.1145/1044956.1044958 http://doi.acm.org/10.1145/1044956.1044958

ANEXO A

CÓDIGO-FONTE EM BASH DO ARQUIVO PRINCIPAL DO MODREP.

Arquivo "script.sh".

```
#!/bin/bash
function achasensiveis
{
	IFS=$';'
	sensiveis=`cat /bin/wseos/sensiveis.wseos`
}
function comparaarquivos
{
	echo `ls -R $sensiveis -o -l --full-time | awk '/-/{print $6 " "
$8}'` > '/bin/wseos/monitfilesnow.wseos'
	dtaltnow=`cat '/bin/wseos/monitfilesnow.wseos'`
#	echo "now"
#	echo $dtaltnow
#	echo "---------------------"
	dtaltold=`cat '/bin/wseos/monitfilesold.wseos'`
#	echo "old"
#	echo $dtaltold
#	echo "---------------------"
	if [ "$dtaltold" = "$dtaltnow" ]
	then
			echo "-------------"
			echo "nada alterado"
			echo "-------------"
			precisareplic=0
	else
			if [ "`cat '/bin/wseos/breca.wseos'`" = "1" ]
			then
					echo "-------"
					echo "brecado"
					echo "-------"
					cp     '/bin/wseos/monitfilesnow.wseos'
'/bin/wseos/monitfilesold.wseos'
					precisareplic=0
					echo "0" > "/bin/wseos/breca.wseos"
			else
					echo "----------------"
					echo "precisa replicar"
					echo "----------------"
					precisareplic=1
			fi
	fi
}
function achameuip
{
```

```
	#encontra meus ips na faixa valida-------
	meuip=`/sbin/ifconfig | awk '/Bcast/{print $2}'`
	meuip=${meuip:5}
	echo "meu ip eh:" $meuip
}
function achatodos
{
	#encontra as maquinas ativas---
	a=$meuip
	b=${a##*.}
	c=${a//$b/*}
	result=`nmap -sP $c | awk '/for /{print $5}'`
	echo "${result// /\n}" > '/bin/wseos/output.wseos'
}
function guardatodos
{
	#guarda todos ips dos outros numa matriz---
	ips='/bin/wseos/output.wseos'
	i=0
	while read line
	do
		if [ "$line" != "$meuip" ]
		then
			matriztodos[$i]=$line
			echo $line
		fi
		i=$(($i+1))
	done < $ips
	echo "-----------------"
	echo "encontrados:" $(($i-1))
	echo "-----------------"
}
function achaservers
{
	#guarda servers ativos numa matriz----
	k=0
	for (( j=0 ; j < $i ; j++))
	do
		if	[	"`ssh	-o	ConnectTimeout=2
root@${matriztodos[j]} 'cat /bin/wseos/server.wseos'`" = "server" ]
		then
			matrizservers[$k]=${matriztodos[$j]}
			k=$(($k+1))
			echo ${matriztodos[j]} "eh server"
		fi
	done
	if (( "$k" <= 0 ))
	then
		echo "-------------------------"
		echo "nenhum servidor a replicar"
		echo "-------------------------"
		nenhumoutro=1
		echo "1" > "/bin/wseos/breca.wseos"
	fi
	echo "------------------------------"
```

```
        echo "total de servers ativos:" $k
        echo "------------------------------"
    }
    function trava
    {
        for (( m=0 ; m < $k ; m++))
        do
            `ssh  root@${matrizservers[m]}  'setfacl  -R  -m
u:gerente:r-- /home/'`
            `ssh  root@${matrizservers[m]}  'echo  "travado"  >
"/bin/wseos/status.wseos"'`
            echo ${matrizservers[m]} "travado"
        done
    }
    function replica
    {
        echo "replicando" > '/bin/wseos/status.wseos'
        for (( m=0 ; m < $k ; m++))
        do
            if   [   "`ssh   root@${matrizservers[m]}   'cat
/bin/wseos/status.wseos'`" = "travado" ]
            then
                `rsync  -av  --detect-renamed  --delete-
delay    --exclude    '*.gvfs'    --exclude    'gvfs-metadata*'    /home/
root@${matrizservers[m]}:/home/`
                `rsync -av --delete --exclude '*.gvfs'
--exclude 'gvfs-metadata*' /home/ root@${matrizservers[m]}:/home/`

                echo ${matrizservers[m]} "replicado"
            fi
        done
        echo "ok" > '/bin/wseos/status.wseos'
    }
    function destrava
    {
        for (( m=0 ; m < $k ; m++))
        do
            `ssh  root@${matrizservers[m]}  'setfacl  -R  -m
u:gerente:rwx /home/'`
            echo ${matrizservers[m]} "destravado"
            cp                '/bin/wseos/monitfilesnow.wseos'
'/bin/wseos/monitfilesold.wseos'
            `ssh   root@${matrizservers[m]}   'echo   "1"   >
"/bin/wseos/breca.wseos"'`
            `ssh   root@${matrizservers[m]}   'echo   "ok"   >
"/bin/wseos/status.wseos"'`
            echo "1" > '/bin/wseos/breca.wseos'
        done
    }
    function checaprimeira
    {
        echo "entrou em checaprimeira"
        achameuip
        achatodos
        guardatodos
```

```
            achaservers
            if [ "$nenhumoutro" = "1" ]
            then
                        echo "nenhum outro server"
                        echo "ok" > '/bin/wseos/status.wseos'
            else
                        if    [    "`ssh    root@${matrizservers[0]}    'cat
/bin/wseos/status.wseos'`" = "ok" ]
                        then
                                    echo  ">>  Inicio  do  recebimento  da
réplica de" ${matrizservers[0]} " (1a vez)<<"
                                    `setfacl -R -m u:gerente:r-- /home/`
                                    `ssh root@${matrizservers[0]} 'setfacl
-R -m u:gerente:r-- /home/'`
                                    `ssh   root@${matrizservers[0]}   'echo
"replicando" > "/bin/wseos/status.wseos"'`
                                    `rsync                                -av
root@${matrizservers[0]}:/etc/{passwd,group,shadow,gshadow} /etc/`
                                    `rsync  -av  --detect-renamed  --delete-
delay       --exclude       '*.gvfs'       --exclude       'gvfs-metadata*'
root@${matrizservers[0]}:/home/ /home/`
                                    `ssh root@${matrizservers[0]} 'setfacl
-R -m u:gerente:rwx /home/'`
                                    `ssh   root@${matrizservers[0]}   'echo
"ok" > "/bin/wseos/status.wseos"'`
                                    `setfacl -R -m u:gerente:rwx /home/`
                                    echo "1" > '/bin/wseos/breca.wseos'
                                    echo "ok" > '/bin/wseos/status.wseos'
                        fi
            fi
            echo
            echo  ">>  Recebida  a  réplica  de"  ${matrizservers[0]}  "  (1a
vez)<<"
            echo
            echo
      }
      #--fim das funcoes-----
      #--programa principal-------
      echo
      echo $(date)
      if [ "`cat '/bin/wseos/status.wseos'`" = "replicando" ]
      then
            echo "*** em replicacao (enviando) - tente mais tarde ***"
      fi
      if [ "`cat '/bin/wseos/status.wseos'`" = "travado" ]
      then
            echo "*** em replicacao (recebendo) - tente mais tarde ***"
      fi
      if [ "`cat '/bin/wseos/status.wseos'`" = "primeiravez" ]
      then
            checaprimeira
      fi
      if [ "`cat '/bin/wseos/status.wseos'`" = "ok" ]
      then
            achasensiveis
```

```
	comparaarquivos
fi
if [ "$precisareplic" = "0" ]
then
	echo "nada a replicar"
fi
if [ "$precisareplic" = "1" ]
then
	achameuip
	if [ "$meuip" = "" ]
	then
		echo "erro! server fora da rede"
	else
		achatodos
		guardatodos
		achaservers
		trava
		replica
		destrava
	fi
fi
echo "---------------"
echo "fim da checagem"
echo "---------------"
echo $(date)
echo
#----------------------------
```

ANEXO B

CÓDIGO-FONTE EM BASH DO ARQUIVO PRINCIPAL DO MODESC.

Arquivo "decideserver.sh".

```
#!/bin/bash
#inicio funcoes------------
function achameuip
{
	#encontra meus ips---
	meuip=`/sbin/ifconfig | awk '/Bcast/{print $3}'`
	echo $meuip > '/var/www/html/mensagem1.wseos'
}
function achatodos
{
	#encontra as maquinas ativas---
	a=$meuip
	b=${a##*.}
	c=${a//$b/*}
	result=`nmap -sP $c | awk '/for /{print $5}'`
	echo "${result// /\n}" > '/var/www/html/todosips.wseos'
}
function guardatodos
{
	#guarda todos ips dos outros numa matriz---
	ips='/var/www/html/todosips.wseos'
	i=0
	echo
	echo "Outros dispositivos na rede WSEOS:"
	while read line
	do
			if [ "$line" != "$meuip" ]
			then
				matriztodos[$i]=$line
				echo $line
			fi
			i=$(($i+1))
	done < $ips
	echo "-----------------"
	echo "Total encontrado:" $(($i-1))
	echo "-----------------"
	echo
}
function achaservers
{
	#guarda servers ativos numa matriz----
	echo "Buscando servidores WSEOS"
	k=0
	for (( j=0 ; j < $i ; j++))
```

```
            do
                        ocupacao="0"
                        if      [       "`ssh       -o       ConnectTimeout=2
root@${matriztodos[j]} 'cat /bin/wseos/server.wseos'`" = "server" ]
                        then
                                    ocupacao=`ssh    root@${matriztodos[j]}
'cat /bin/wseos/sessoesativas.wseos'`
                                    freeram=`ssh     root@${matriztodos[j]}
'cat /bin/wseos/freeram.wseos'` #teste
                                    matrizservers[$k]=${matriztodos[$j]}
                                    matrizocupacao[$k]=$ocupacao
                                    matrizfreeram[$k]=$freeram
                                    echo   ">   "   ${matriztodos[j]}   "eh
server, tem" $ocupacao "cliente(s) conectado(s) e" $freeram "% de RAM livre."
                                    echo        ${matriztodos[j]}        >>
'/var/www/html/ativos.wseos'
                                    echo               ${matrizocupacao[k]}
${matriztodos[j]} ${matrizfreeram[k]} >> '/var/www/html/ativocup.wseos'
                                    k=$(($k+1))
                        fi
            done
            echo "-------------------------------"
            echo "total de servers ativos:" $k
            echo "-------------------------------"
            echo $k > '/var/www/html/qtdservers.wseos'
      }
      function achamenorcarga
      {
            echo
            echo "Escolhendo servidor com menor carga"
            listaocup='/var/www/html/ativocup.wseos'
            m="1"
            minip="x"
            minocup="70000"
      #     maxfreeram="101"
            while read linha
            do
                        if  [  "$linha"  =  ""  ]  #corrige  falha  de  1a  linha
vazia
                        then
                                    m=$(($m+1))
                        else
                                    linhaocup=`echo   $linha   |   awk   '/
/{print $1}'`
                                    linhaip=`echo $linha | awk '/ /{print
$2}'`
                                    if [[ "$linhaocup" -lt "$minocup" ]]
                                    then
                                                minocup=$linhaocup
                                                minip=$linhaip
                                                pos=$m

                                    fi
                                    m=$(($m+1))
                        fi
```

```
            done < $listaocup
            echo   "Servidor   recomendado:"   $minip".   Possui"   $minocup
"cliente(s)."
            echo   "Servidor   recomendado:"   $minip".   Possui"   $minocup
"cliente(s)conectados." > '/var/www/html/mensagem2.wseos'
      }
      function montanxs

      {
            cat                                '/var/www/html/default.nxs'|sed
"s/SEW_OS_IP/$minip/g">'/var/www/html/temp0.nxs'
            echo "----------------------------"
            echo "Ultimo servidor recomendado:" $oldrecomend
            oldrecomend=$minip
      }
      function naopara
      {
            continua=1
            while continua="1"
            do
            echo "---------------INICIO-----------------"
            achameuip
            if [ "$meuip" = "" ]
            then
                        echo "fora da rede"
                        echo "0" > '/var/www/html/mensagem1.wseos'
                        rm '/var/www/html/ativos.wseos'
                        rm '/var/www/html/ativocup.wseos'
                        sleep 2
            else
                        echo "Meu IP é:" $meuip
                        achatodos
                        guardatodos
                        echo    ""    >    '/var/www/html/ativos.wseos'    >
'/var/www/html/ativocup.wseos'
                        achaservers
                        if [ "$k" != "0" ]
                        then
                                    achamenorcarga

                        fi
            fi
            if [ "$oldrecomend" != "$minip" ]
            then
                        montanxs
            fi
            echo "---------------FIM-----------------"
            echo
            echo
            done
      }
      #fim da funcoes------
      #programa principal
      echo       "0"       >       '/var/www/html/qtdservers.wseos'       >
'/var/www/html/mensagem1.wseos'
```

```
echo "" > '/var/www/html/mensagem2.wseos'
oldrecomend="0"
naopara
#fim programa
```