JOSINEY DE SOUZA

AVALIAÇÃO DE ALGORITMOS DE ESCALONAMENTO DE DISCO COM QUALIDADE DE SERVIÇO EM AMBIENTES VIRTUALIZADOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luis Carlos Erpen de Bona

CURITIBA

2014

JOSINEY DE SOUZA

AVALIAÇÃO DE ALGORITMOS DE ESCALONAMENTO DE DISCO COM QUALIDADE DE SERVIÇO EM AMBIENTES VIRTUALIZADOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luis Carlos Erpen de Bona

CURITIBA

2014

S729a Souza, Josiney de
Avaliação de algoritmos de escalonamento de disco com qualidade de serviço em ambientes virtualizados / Josiney de Souza. – Curitiba, 2014.
75f. : il. color.

Dissertação (mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2014.

Orientador: Luis Carlos Erpen de Bona
Bibliografia: p. 70-75.

1. Sistemas de computação virtual. 2. Memoria virtual (Computação). 3. Algoritmos de computador. I. Universidade Federal do Paraná. II. Bona, Luis Carlos Erpen de. III. Título.

CDD: 005.43

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Josiney de Souza, avaliamos o trabalho intitulado, *"Avaliação de Algoritmos de Escalonamento de Disco com Qualidade de Serviço em Ambientes Virtualizados"*, cuja defesa foi realizada no dia 28 de março de 2014, às 10:35 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 28 de março de 2014.

Prof. Dr. Luis Carlos Erpen de Bona
DINF/UFPR – Orientador

Prof. Dr. Dirk von Suchodoletz
Unifreiburg/Alemanha – Membro Externo

Prof. Dr. Carlos Alberto Mazieiro
UTFPR – Membro Interno

Prof. Dr. Fabiano Silva
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço à minha esposa, Cristiane, por todo apoio e carinho prestado nesses anos de mestrado. Sem sua presença, paciência, sabedoria e compreensão esse trabalho não seria possível.

Obrigado ao meu orientador, Bona, pela preocupação com minha causa. Sempre buscou uma forma de tornar essa dissertação concreta: seja criticando os pontos ruins, elogiando as partes boas, animando e desanimando com as diversas conversas ao longo de todo tempo de trabalho.

Obrigado à banca de avaliação. Dirk, pela disposição, presença e apontamentos importantes; Maziero, assim como na proposta, suas críticas direcionaram para um trabalho mais robusto; Fabiano, pelo esclarecimento de pontos relevantes a melhorar no trabalho.

Meus agradecimentos aos pais e familiares, tanto meus quanto de minha companheira, tanto família de sangue quanto de consideração, por sempre me tratarem tão bem, apoiarem e compreenderem as dificuldades e as faltas nas comemorações em nossa terrinha. Foi por uma boa causa.

Sou grato aos professores Direne e Eduardo Almeida pelas conversas de corredor. Toda a calma e experiência de vocês ajudaram-me diminuir os problemas e a enxergar as soluções. Ao colega Pedro Rocha pelos códigos-fontes e pelos apontamentos do melhor a se fazer.

Agradecimentos aos companheiros de laboratório pela companhia nos almoços e momentos de descontração. Suas brincadeiras, vídeos, piadas e casos sempre foram muito apreciados... aos companheiros e ex-companheiros de C3SL, professores ou alunos. Aprendi muito com vocês tanto a parte técnica de trabalho quanto a convivência em grupo...

A todos que conheci ao longo dessa jornada: no futebol, nos laboratórios próximos, os amigos dos amigos, amigos dos amigos dos amigos, ... também a todos que apoiaram essa dissertação, direta ou indiretamente... a todos, meu muito obrigado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Virtualização é uma técnica utilizada em diversas áreas de conhecimento, cada qual com seu objetivo específico. Porém todas possuem um ponto em comum: definem a virtualização como uma forma de abstrair um ambiente físico em um ambiente lógico. O foco deste trabalho é a virtualização de servidores. No contexto de virtualização de servidores, máquinas virtuais (VMs) possuem estudos bem definidos para os componentes processador, memória e rede. O componente disco ainda é alvo de pesquisas, pois discos virtuais adicionam novas características aos sistemas tradicionais de armazenamento. A virtualização de servidores deve permitir e gerenciar a execução de VMs simultaneamente. Com a execução simultânea, é necessário disponibilizar meios de controlar o uso do sistema, como a Qualidade de Serviço (QoS). A QoS surgiu como forma de priorizar determinados atributos de utilização em redes de computadores, mas pode ser expandida a qualquer canal de comunicação. Esta dissertação faz o levantamento dos requisitos necessários para se considerar processos de máquinas virtuais no escalonamento de requisições de disco. O objetivo é escalonar as requisições dos discos virtuais no sistema base e oferecer atributos de QoS diferentes para máquinas virtuais distintas. Como forma de avaliação, um algoritmo para disco físico (HTBS) é adaptado para considerar VMs. O algoritmo resultante é comparado aos escalonadores de disco padrão do Linux.

ABSTRACT

Virtualization is a technique used in various knowledge areas, each of them with a specific goal. But all the areas have one thing in common: they define virtualization as a way to abstract a physical environment in a logical environment. The focus of this thesis is server virtualization. On server virtualization field, virtual machines (VMs) have well defined works for processor, memory and network components. The disk component yet is target of researches, because virtual disks add new features to the storage systems. The server virtualization must enable and manage the simultaneous VMs execution. With simultaneous execution, is needed a way to control the system use, like Quality of Service (QoS). The QoS arise to prioritize some attributes in networks, but it can be used on any communication channel. This thesis survey the requirements needed to have virtual machines processes in the physical disk scheduling requests. The goal is to schedule virtual disk requests on host system and give different QoS attributes to different VMs. As a way to evaluate this work, a physical disk algorithm (HTBS) is adapted to work with VMs. The new algorithm is compared against the Linux default schedulers.

CAPÍTULO 1

INTRODUÇÃO

A virtualização é utilizada em diversas áreas de conhecimento, como sistemas operacionais, arquitetura de computadores e linguagens de programação [45]. Embora cada área defina um objetivo diferente para a virtualização, todas elas possuem um ponto em comum: definem a virtualização como uma forma de abstrair um ambiente físico em um ambiente lógico [37].

O termo virtualização permite entender algumas abordagens de uso em um sistema, como: virtualização de *hardware*, de interface de sistema, de dispositivos de entrada e saída (E/S), de sistema operacional, de chamadas de sistema e de bibliotecas [30]. Contudo, ao se mencionar virtualização, pretende-se falar da tecnologia de máquinas virtuais, que utiliza uma ou mais formas de virtualização [53].

As máquinas virtuais podem ser classificadas em três categorias [30]: de processo, de sistema operacional e de sistema. As máquinas virtuais de processo são ambientes construídos para a execução de um processo ou aplicação específica. As máquinas virtuais de sistema operacional executam sobre o mesmo sistema operacional e isolam espaços de usuários. As máquinas virtuais de sistema simulam um ambiente de *hardware* completo.

A primeira categoria permite a execução de um processo por vez, enquanto as demais categorias permitem que vários processos executem simultaneamente. A execução simultânea desses processos criam um ambiente virtual que simula diversos computadores, cada qual distinto porém executando em uma mesma base. A criação desses ambientes virtuais é conhecida como virtualização de servidores [10].

Na virtualização de servidores, componentes de *hardware* como processadores, discos, placas de rede e memória primária existem tanto nos ambientes físicos como nos ambientes virtuais [53]. Dentre estes, apenas processadores, placas de rede e memória primária virtuais possuem estudos bem definidos [34, 24, 52, 32]; enquanto que os discos virtuais

ainda são alvo de pesquisas, pois adicionam novas características aos sistemas tradicionais de armazenamento [28], como a interação entre escalonadores de disco para VMs, o isolamento de VMs e arquivos grandes como discos virtuais.

A virtualização de servidores permite o processamento simultâneo de tarefas nos diferentes ambientes virtuais, possibilitando que vários usuários executem suas tarefas concorrentemente. Como forma de controlar o acesso aos recursos virtuais e o uso do sistema, a virtualização de servidores permite a utilização de sistemas de qualidade de serviço (ou QoS, do inglês *quality of service*) [51].

A qualidade de serviço é uma área de estudos que surgiu da necessidade de priorizar determinados atributos de utilização em redes de computadores segundo limites específicos. A QoS atua sobre um canal de comunicação para garantir atributos como vazão, latência e rajadas [35]. Embora a qualidade de serviço seja principalmente estudada em redes de computadores, ela é expansível a qualquer canal de comunicação.

As propostas de [35, 42, 28] utilizam QoS para definir parâmetros de atuação sobre o escalonamento de requisições de E/S sobre discos rígidos. Rocha et al. [35, 42] avaliam QoS para disco físico, garantindo os atributos de vazão, latência e rajadas. Ling et al. [28] avaliam QoS para discos virtuais, considerando os atributos vazão e latência. Porém, Rocha et al. [35, 42] não consideram o escalonamento de requisições virtuais e Ling et al. [28], mesmo considerando requisições de máquinas virtuais, configuram o atributo vazão de forma indireta.

O objetivo desta dissertação é oferecer atributos de qualidade de serviço individuais no escalonador de requisições de E/S de máquinas virtuais. Pretende-se descobrir os mecanismos necessários para considerar processos de máquinas virtuais no escalonamento, escalonar as requisições de E/S virtuais no sistema base e oferecer garantias mínimas de execução das VMs; além de classificar os fatores de influência no escalonamento de requisições virtuais.

A avaliação da dissertação é feita com a adaptação do algoritmo de Rocha et al. para o ambiente virtual e experimentos utilizando a ferramenta de *benchmark fio*. Os testes do *fio* consideram os parâmetros de configuração interna do algoritmo adaptado, a avaliação dos

atributos de QoS individuais e a comparação dos resultados obtidos com os escalonadores padrão do Linux.

O restante do texto está organizado da seguinte forma: o Capítulo 2 mostra como os componentes de *hardware* são virtualizados, quais as técnicas de virtualização de servidores existentes e descreve o ambiente de virtualização utilizado (KVM). O Capítulo 3 apresenta conceitos de qualidade de serviço no sistema Linux e trabalhos relacionados à QoS de máquinas virtuais. O Capítulo 4 contém a proposta de trabalho desenvolvida e o Capítulo 5 possui os resultados experimentais em atribuir QoS para VMs. Por fim, o Capítulo 6 encerra a dissertação.

CAPÍTULO 2

VIRTUALIZAÇÃO DE SERVIDORES

A tecnologia de máquina virtual abstrai um ambiente físico em um ambiente lógico. Também conhecida como virtualização, essa tecnologia permite que um simples computador seja capaz de hospedar múltiplas máquinas virtuais; potencialmente cada VM executando um sistema operacional diferente [47].

Um sistema que suporte virtualização possui como componentes: (i) sistema base, (ii) sistema virtual e (iii) monitor de máquina virtual (VMM, do inglês *Virtual Machine Monitor*). A Figura 2.1 apresenta esses componentes. O sistema base, também chamado de *host* ou hospedeiro, é a máquina física que contém todo o ambiente de virtualização; tanto os sistemas virtuais quanto o VMM.

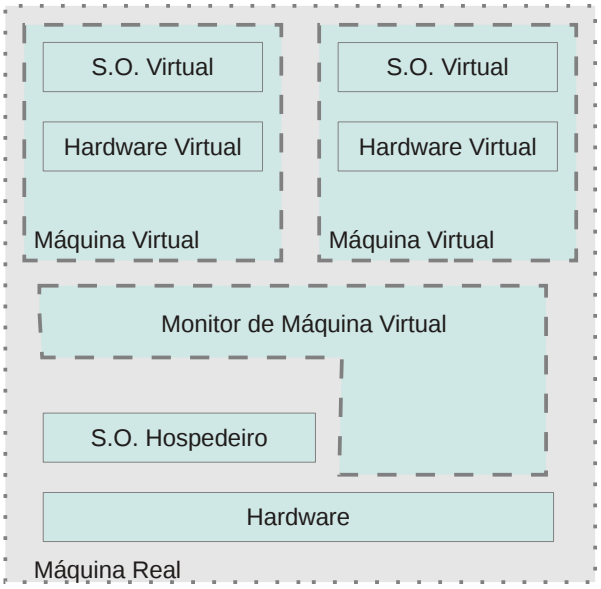


Figura 2.1: Componentes da virtualização

Cada sistema virtual é conhecido por máquina virtual, *guest*, convidado ou hospedado. Assim como o sistema base, máquinas virtuais também possuem *hardware* e sistema ope-

racional. O VMM ou hipervisor é responsável pela intermediação de requisições, decidindo qual máquina virtual deve executar em determinado momento; além de gerenciar os pedidos de acesso a dispositivos da máquina base.

A virtualização de servidores deve prover máquinas virtuais completas para um ou mais sistemas operacionais convidados e suas aplicações, permitindo sua execução isolada e independente [30]. Para cumprir com esse objetivo, a virtualização de servidores pode utilizar um ou mais componentes virtualizados de *hardware*.

As seções seguintes tratam do ambiente da virtualização, suas técnicas e componentes. A Seção 2.1 apresenta as dificuldades em virtualização de servidores. A Seção 2.2 mostra técnicas de virtualização dos recursos físicos dos computadores e de virtualização de servidores. A Seção 2.3 define o ambiente de virtualização utilizado neste trabalho.

2.1 Dificuldades de Virtualização de Servidores

A maioria dos computadores possuem dois modos de operação de *hardware* e de sistema operacional (SO): modo *kernel* e modo usuário [47]. No modo *kernel*, apenas programas confiáveis existem; aos processos desses programas é garantido o acesso a todos os recursos de *hardware*, podendo executar todas as instruções disponíveis no conjunto de instruções que o sistema computacional disponibiliza.

As instruções executadas em modo *kernel* são conhecidas como instruções privilegiadas. Essas instruções alteram o estado do sistema computacional manipulando a memória, trocando estruturas de controle do processador ou realizando operações de E/S. Em contrapartida, instruções não-privilegiadas não mudam o estado do sistema computacional.

No modo usuário estão todos os demais programas que não são considerados confiáveis. Os processos desses programas podem executar instruções não-privilegiadas diretamente. Por outro lado, caso um processo em modo usuário necessite alterar o estado do sistema, ele deve fazer uma chamada de sistema (syscall, do inglês *system call*).

Chamadas de sistema permitem que processos em modo usuário requisitem a execução de instruções privilegiadas ao sistema operacional. A Figura 2.2 ilustra a utilização de syscalls. Através da interface de syscalls um processo em modo usuário solicita uma ação

ao SO; o SO, de maneira controlada e em modo *kernel*, executa a ação caso o processo tenha as permissões necessárias; o SO retorna o controle de execução ao processo que o chamou.

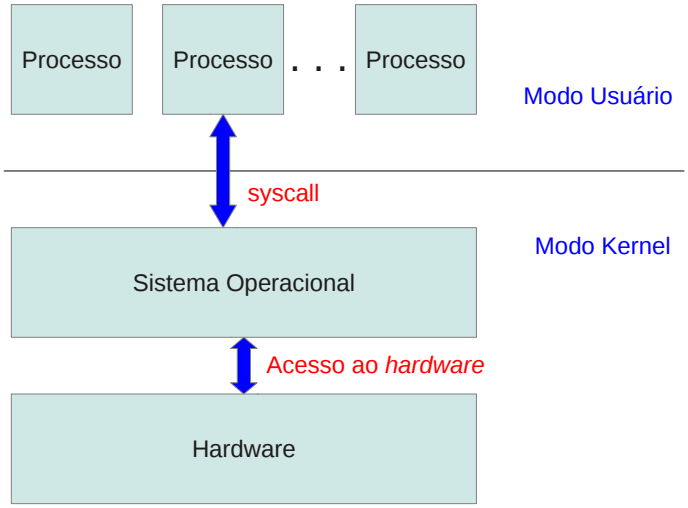


Figura 2.2: Processo acessando *hardware* indiretamente através de syscall

Para que um sistema seja virtualizável, é necessário que o monitor de máquina virtual reconheça processos em modo usuário e possa descobrir quando estes processos necessitam de tratamento diferenciado. Com essa habilidade, o VMM pode entregar uma abstração do *hardware* físico aos sistemas virtuais, gerenciando e modificando as regiões entregues através de chamadas de sistema.

Historicamente, processadores da família PC Intel (Pentium IV e anteriores) possuem instruções não-privilegiadas que consultam ou alteram o estado de execução da máquina real [30]. Essas instruções devem ser executadas em modo *kernel*, sendo conhecidas por instruções sensíveis. Popek e Goldberg [36] definem que um sistema é virtualizável se as instruções sensíveis forem um subconjunto das instruções privilegiadas. Por esse motivo, sistemas com processadores da família PC Intel não podem ser virtualizados de modo clássico.

Recentemente, as CPUs dos computadores vêm incorporando suporte para facilitar a virtualização de componentes [50]; permitindo que a técnica de virtualização clássica seja

utilizada também nos computadores com novos processadores da família PC Intel. Porém, dada a antiga limitação, outras técnicas de virtualização foram desenvolvidas, além da virtualização clássica. A seção seguinte sumariza essas técnicas.

2.2 Técnicas de Virtualização

A virtualização de servidores compreende três abordagens de VMM: virtualização completa, paravirtualização e virtualização de contêiner [12, 29]. A virtualização completa e a paravirtualização compõem a virtualização de servidores que introduz uma camada de virtualização entre o sistema base e os sistemas virtuais, enquanto a virtualização de contêiner compreende a virtualização de nível de SO [16].

A virtualização completa proporciona uma cópia virtual do *hardware* de um sistema, possivelmente diferente do sistema base. Neste tipo de virtualização, as aplicações e SOs virtuais executam sem modificação, devendo cada instrução ser analisada e traduzida para uma ou mais instruções equivalentes do sistema base.

A paravirtualização modifica os sistemas operacionais convidados para remover as chamadas às suas instruções sensíveis [47]. Enquanto a virtualização completa executa máquinas virtuais com SO sem modificação, a paravirtualização modifica o código dos sistemas operacionais virtuais para retirar suas chamadas sensíveis e substituí-las por chamadas ao hipervisor (hypercalls, do inglês *hypervisor calls*). Ao realizar chamadas ao hipervisor, o SO hospedado age de maneira semelhante a um programa em modo usuário que faz syscalls para modificar o estado do sistema.

A virtualização de contêiner, também chamada de virtualização de isolamento, cria ambientes virtuais sobre o mesmo núcleo de sistema. Esse tipo de virtualização não realiza simulações ou virtualização verdadeira; em vez disso, é realizada a divisão do sistema operacional base em espaços de usuários distintos, com cada ambiente virtual recebendo seus próprios recursos lógicos [30]. Apesar de não realizar simulações e virtualização efetiva, esta técnica permite que processos sejam separados em espaços de usuário diferentes semelhantes à organização lógica das máquinas virtuais.

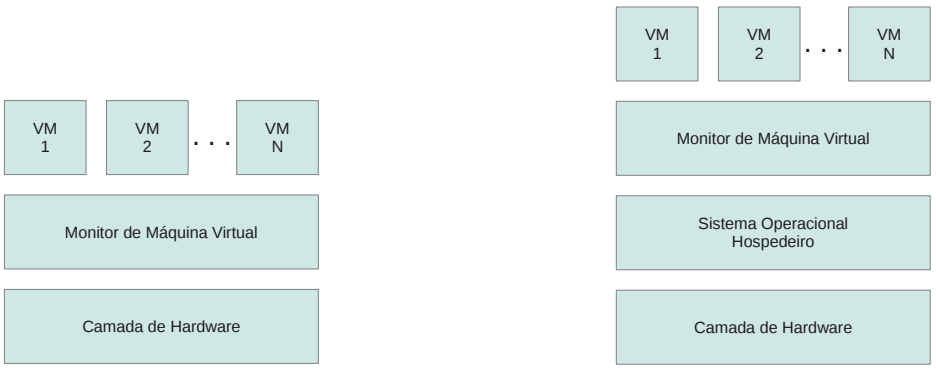
Independentemente do tipo de virtualização, máquinas físicas e máquinas virtuais

possuem componentes físicos em comum: processadores, discos e memória primária [53]. Cada um desses componentes, para existirem no ambiente virtual, devem possuir suporte no ambiente base. As subseções abaixo apresentam técnicas para virtualização desses componentes físicos.

2.2.1 Virtualização de CPU

As CPUs das máquinas virtuais são idênticas às dos sistemas que as contêm e efetuam virtualização, entregando instruções não-privilegiadas diretamente à CPU física; já as instruções privilegiadas são processadas pelo VMM de acordo com as técnicas sumarizadas adiante. As CPUs virtuais possuem seu próprio contexto, possuindo registradores, *buffers* e estruturas de controle.

A Figura 2.3 apresenta técnicas de virtualização de CPU com base no tipo de VMM. Há duas abordagens diferentes para o VMM: (i) hipervisor nativo (hipervisor tipo I) e (ii) hipervisor convidado (hipervisor tipo II). O hipervisor nativo é o tipo de VMM que atende às definições de Popek e Goldberg, monitorando as instruções sensíveis; o hipervisor convidado é o tipo de VMM desenvolvido para contornar as antigas restrições de implementação dos processadores da arquitetura PC Intel. Em ambos os VMMs, o SO virtual não requer modificações para executar.



(a) Hipervisor tipo I

(b) Hipervisor tipo II

Figura 2.3: *Técnicas de virtualização baseadas nos tipos de VMM*

Um hipervisor nativo executa diretamente sobre o *hardware* da máquina base. Neste tipo de sistema de virtualização, não há um sistema operacional intermediando as operações gerenciadas pelo VMM; o próprio VMM age como um sistema operacional e intercepta as chamadas e comandos internos dos sistemas virtuais. O VMM desta técnica de virtualização é o único programa a executar em modo *kernel*, podendo acessar diretamente o *hardware*. Os sistemas virtuais não podem executar instruções privilegiadas diretamente, pois executam em modo usuário.

A Figura 2.4 mostra as visões dos níveis de abstração para um hipervisor tipo I. No interior das VMs, o SO convidado executa com a ilusão de estar no modo *kernel* (modo *kernel* virtual), enquanto os demais programas das VMs executam com a visão de estar no modo usuário (modo usuário virtual).

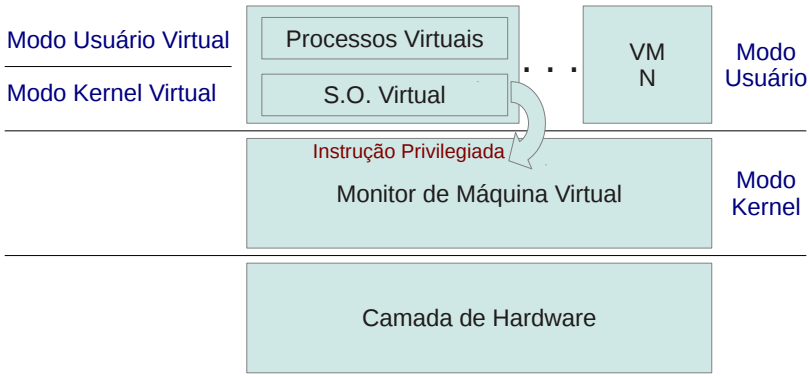


Figura 2.4: Níveis de abstração para hipervisor tipo I

Ao considerar um sistema hospedeiro com tecnologia de virtualização, quando uma máquina virtual necessita executar uma instrução sensível, o VMM intercepta a chamada e verifica qual o modo de execução virtual originou a instrução. Se a instrução vier do SO virtual, o VMM realiza os passos necessários para completar a instrução; se a instrução vier do modo usuário virtual, o VMM simula a ação do *hardware* virtual diante da instrução sensível executada em modo usuário.

Ao considerar um sistema hospedeiro sem tecnologia de virtualização, a VM que originou a instrução sensível pode travar; pois a VM é um processo em modo usuário e instruções sensíveis nesse modo são ignoradas. Em máquinas virtuais com tecnologia de

virtualização, a instrução sensível é interceptada pelo sistema operacional virtual; em VMs sem tecnologia de virtualização, a instrução sensível é ignorada.

Um hipervisor convidado executa sobre o sistema operacional da máquina base. Neste tipo de sistema de virtualização, o VMM executa em modo usuário, competindo com os demais processos em execução do sistema base pela posse de recursos. O VMM desta técnica de virtualização realiza a tradução binária de programas virtuais: com os programas carregados em memória, antes de executá-los, o VMM procura por blocos básicos e os substitui por chamadas ao VMM.

Blocos básicos são blocos de códigos cujas execuções terminam com instruções *jump*, *call*, *trap* ou qualquer outra instrução que altere o fluxo de controle [47]. Na tradução binária, cada bloco básico é examinado em busca de instruções sensíveis; se encontradas, essas instruções sensíveis são substituídas por chamadas a rotinas de tratamento do VMM. A última instrução do código, que altera o contador de programa (*program counter*) também é substituída por uma chamada de rotina do VMM. Após esses passos, o bloco básico é mantido na memória do VMM e então executado.

A Figura 2.5 ilustra a tradução binária no hipervisor tipo II. Após executar um bloco básico, o controle é retornado ao VMM, que deve decidir por qual bloco executar na sequência. Se o próximo bloco a ser executado já tiver sido traduzido, ele pode ser executado imediatamente; caso contrário, o bloco deve ser traduzido e armazenado na memória do VMM antes de executar.

2.2.2 Virtualização de E/S

Os sistemas computacionais atuais possuem soluções bem definidas para tratar da utilização de CPU e de memória nos sistemas virtuais; contudo, a virtualização de E/S ainda é um desafio e não possui uma solução amplamente aceita [54]. Algumas abordagens desenvolvidas para virtualização de E/S são sumarizadas na Figura 2.6: virtualização completa, paravirtualização, emulação por *software* e acesso direto ao dispositivo.

A virtualização completa ocorre em VMMs do tipo autônomo (que não requerem auxílio de outras aplicações para executar - hipervisor tipo I) e acontece em etapas: o SO

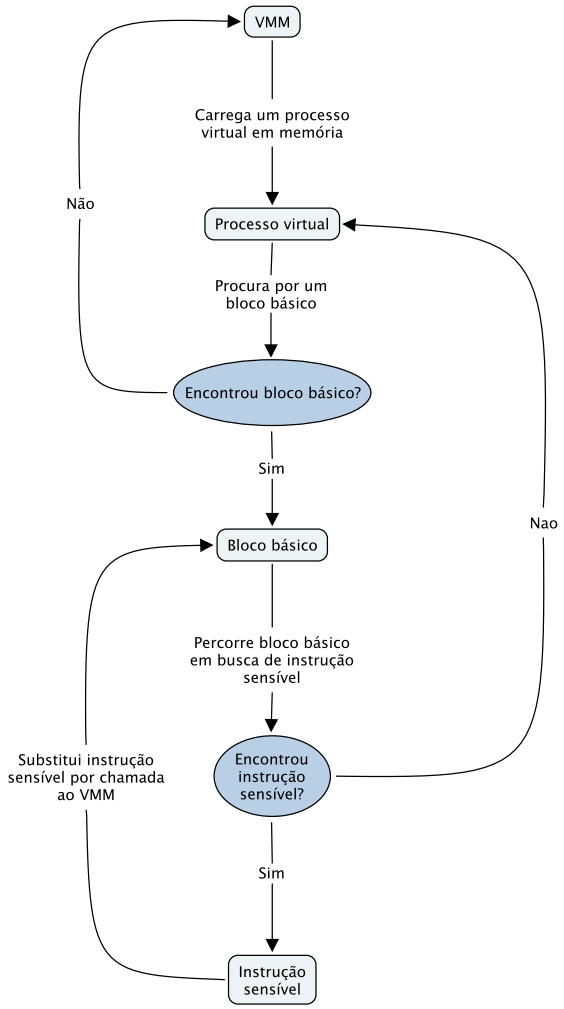


Figura 2.5: Tradução binária em hipervisor tipo II

convidado gera uma instrução de acesso ao dispositivo de E/S virtual, o VMM captura a requisição ainda em formato de *hardware* virtual, decodifica a instrução virtual em uma ou mais instruções equivalentes para o *hardware* físico e guia o dispositivo físico para o atendimento da requisição.

A paravirtualização de E/S acontece por meio de dois *drivers* de comunicação entre os sistemas virtuais e o sistema físico: *frontend* e *backend*. O *driver* de *frontend* é instalado no sistema virtual; sua função é coletar a instrução de E/S virtual e encaminhá-la ao *driver* de *backend*. O *driver* de *backend* é instalado no VMM; ele recebe a requisição virtual, interpreta-a e encaminha-a para o dispositivo.

A emulação por *software* ocorre em VMMs do tipo convidado (VMMs que executam

Passo	Virtualização Completa	Paravirtualização
1	(Guest) S.O. gera instrução virtual	(Guest) S.O. gera instrução virtual
2	(Guest) Driver S.O. pede E/S	(Guest) Frontend pede E/S
3	(VMM) Captura instrução virtual	(VMM) Backend captura instrução virtual
4	(VMM) Decodifica instrução virtual	(VMM) Decodifica instrução virtual
5	(VMM) Encaminha instrução real a dispositivo	(VMM) Encaminha instrução real a dispositivo
6	(Host) Dispositivo realiza E/S	(Host) Dispositivo realiza E/S
7		
8		

Passo	Emulação por Software	Acesso Direto
1	(Guest) S.O. gera instrução virtual	(Guest) S.O. gera instrução virtual
2	(Guest) Driver S.O. pede E/S	(Guest) Driver S.O. pede E/S
3	(VMM) Captura instrução virtual	(Host) Dispositivo realiza E/S
4	(VMM) Decodifica instrução virtual	
5	(VMM) Pede E/S para S.O. host	
6	(Host) Atende syscall	
7	(Host) Encaminha E/S	
8	(Host) Dispositivo realiza E/S	

Figura 2.6: Quadro comparativo de passos necessários para realizar a virtualização de E/S

sobre o SO do sistema base). O VMM convidado não tem acesso direto ao *hardware* do sistema base pois é uma aplicação em modo usuário. As instruções de E/S geradas no sistema hospedado devem ser interceptadas pelo VMM, o VMM deve decodificar a instrução, encaminhar o pedido de acesso ao *hardware* para o SO (instrução privilegiada) e o núcleo do sistema base deve tratar a requisição de E/S através de syscalls.

O acesso direto ao dispositivo permite que o *hardware* físico seja acessado diretamente pelos sistemas virtuais, sem interferência do VMM. Esta abordagem deve tratar as características de isolamento e de compartilhamento, características geralmente tratadas pelo SO hospedeiro ou pelo VMM. Isolamento refere-se a garantir que uma VM seja independente de qualquer outra e que não interfiram ou acessem informações de outros contextos virtuais. O compartilhamento refere-se a garantir o múltiplo acesso de VMs ao dispositivo ao mesmo tempo.

2.2.3 Virtualização de Memória

Atualmente, os sistemas operacionais dispõem do artifício de memória virtual (ou paginação): a cada aplicação, o SO entrega a abstração de existir um espaço de endereçamento

possivelmente maior que a quantidade de memória RAM disponível no sistema computacional. Esse espaço de endereçamento é dividido em partes menores chamadas de páginas. O SO elege que páginas serão carregadas na memória do computador em um dado instante de tempo e pode mudar a disposição de páginas à medida que as aplicações necessitem utilizar páginas ainda não mapeadas.

Na paginação, existem dois níveis de endereços: os endereços virtuais, que são as porções de memória com as quais as aplicações trabalham; e os endereços físicos (ou endereços de máquina), que são as porções de memória existentes efetivamente em um sistema computacional. Quando uma aplicação requer que certa página seja utilizada, esta informa o endereço virtual desejado e o SO traduz o endereço virtual em endereço físico através da Unidade de Gerenciamento de Memória (MMU, do inglês *Memory Management Unit*), utilizando internamente uma tabela de páginas (*page table*) para organizar as páginas.

Ao se utilizar virtualização, deve-se permitir que os sistemas virtuais utilizem memória virtual. Nesse sentido, um novo nível de endereço deve ser adicionado. A Figura 2.7 mostra os três níveis necessários. Na visão das máquinas virtuais, existem os endereços virtuais (do espaço de endereçamento das aplicações virtuais) e os endereços físicos (endereçamento físico em relação ao *hardware* virtual), pseudo-físicos de fato. Na visão do hipervisor, existem os endereços físicos (os endereços pseudo-físicos provenientes das VMs) e os endereços de máquina (endereços físicos efetivos).

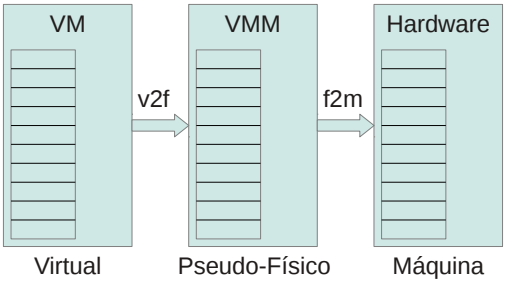
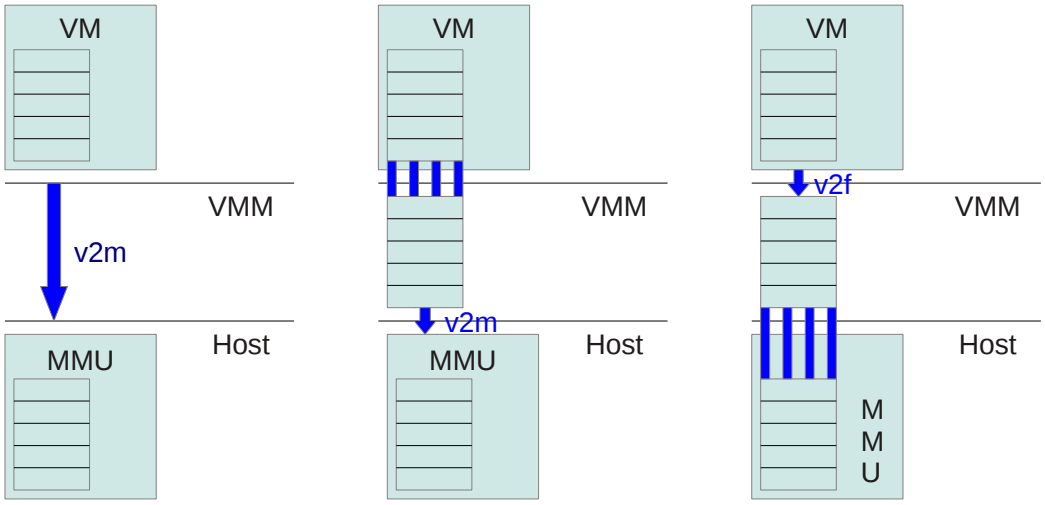


Figura 2.7: Níveis de endereçamento em um sistema com tecnologia de virtualização

Em um sistema virtual, a aplicação virtual possui seu espaço de endereçamento vir-

tual e se comunica com a pseudo-memória física (memória mantida pelo VMM) através da tabela de páginas da relação virtual-VMM (ou relação v2f). O VMM controla a relação f2m, que traduz as páginas pseudo-físicas em endereços de máquina, e pode reaver páginas da relação v2f através de consulta à tabela de páginas do sistema convidado. Para controlar a relação f2m, o VMM pode utilizar uma dentre três técnicas de virtualização de memória: paravirtualização da MMU, tabela de páginas *shadow* e virtualização assistida por *hardware* [52].

A Figura 2.8 apresenta as técnicas de virtualização de memória. A técnica de paravirtualização de MMU consiste em substituir o mapeamento feito na tabela de páginas do ambiente virtual (mapeamento v2f) por uma composição que traduz diretamente endereços virtuais em endereços de máquina (mapeamento v2m). O VMM não mantém tabelas de páginas, apenas realiza a composição de endereços e valida qualquer atualização que o ambiente virtual faça em sua tabela de páginas, protegendo a escrita na tabela de páginas virtual e exigindo que o SO convidado faça chamadas ao hipervisor para atualizações de tabela.



(a) Paravirtualização de MMU

(b) Tabela de páginas *shadow*

(c) Virtualização assistida por *hardware*

Figura 2.8: *Técnicas de virtualização de memória*

A técnica de tabela de páginas *shadow* consiste em manter uma tabela de páginas que

mapeia endereços virtuais diretamente em endereços de máquina, enquanto o ambiente virtual pode manter sua própria tabela de páginas que traduz endereços virtuais para pseudo-físicos. O VMM liga sua tabela de páginas à MMU do ambiente base e deve manter o conteúdo de sua tabela consistente com o conteúdo da tabela de páginas do ambiente virtual.

A técnica de virtualização assistida por *hardware* consiste em manter uma tabela de páginas estendida no VMM. A MMU do ambiente base está ligada tanto à tabela de páginas do ambiente virtual quanto à tabela estendida. A tabela do ambiente virtual traduz endereços virtuais em endereços pseudo-físicos, enquanto a tabela estendida mapeia os endereços pseudo-físicos em endereços de máquina.

2.3 *Kernel-Based Virtual Machine* - KVM

O *Kernel-Based Virtual Machine* ou KVM é uma solução de virtualização de servidores para processadores da arquitetura PC Intel [25, 20]. Esta solução é do tipo virtualização completa e utiliza o *kernel* Linux como VMM para escalonar processos, prover suporte e controle de acesso ao *hardware* e gerenciar a memória dos ambientes virtuais. O VMM está incorporado ao Linux através de módulo de *kernel* e age como uma interface para as máquinas virtuais [44].

Cada VM executa dentro de uma versão modificada do emulador Qemu [39], enquanto o emulador em si executa no sistema base como um processo comum. A memória de cada sistema virtual é mapeada para o espaço virtual de memória da tarefa do emulador Qemu e os processadores virtuais das VMs constituem-se como *threads* do processo existente na máquina base.

A virtualização de CPU no KVM é feita com hipervisor tipo I, requerendo suporte de virtualização nos processadores do ambiente base [25]. Este suporte adiciona um novo modo de operação além do modo *kernel* e do modo usuário: modo *guest*. O modo *guest* permite que as instruções sensíveis utilizadas pelas máquinas virtuais sejam capturadas pelo hipervisor. Uma VM nesse modo executa nativamente até que um evento ou requisição de E/S aconteça; ao sair do modo *guest*, a execução continua no modo *kernel*.

A virtualização de memória no KVM acontece tanto com a tabela de páginas *shadow* quanto com a virtualização assistida por *hardware*, caso esta esteja disponível. Adicionalmente, o KVM suporta uma característica do *kernel* chamada Kernel Same-Page Merging (KSM) [20]. A KSM analisa a memória de cada VM e condensa as páginas de memórias iguais em uma única página. Todas as VMs que teriam a mesma página apontam para esta única cópia comum; apenas quando uma VM tenta mudar essa página comum, a essa VM é dada uma cópia privativa usando a estratégia Copy-on-Write (COW).

A virtualização de E/S acontece através de virtualização completa [25]. Uma requisição de E/S deve surgir durante a execução da VM (em modo *guest*), trocar a execução para o modo *kernel* e na sequência para o modo usuário. Quando a execução estiver em modo usuário, a requisição de E/S deve ser tratada; neste caso o modo usuário deve solicitar ao modo *kernel* a realização da requisição de E/S através de syscall, da mesma forma que aplicações em espaço de usuário. Adicionalmente, KVM pode usar paravirtualização de E/S através de VirtIO [20, 18].

O KVM é a solução de virtualização escolhida para este trabalho pois permite que suas máquinas virtuais sejam manipuladas tais quais qualquer outro processo comum do Linux; dado que elas executam encapsuladas em um emulador Qemu, este um processo comum. Essa característica permite que métodos de diferenciação de processos sejam utilizados (Subseção 3.2.2) e os controles e prioridades necessários sejam aplicados para cumprimento do objetivo do trabalho (Capítulo 4).

CAPÍTULO 3

QUALIDADE DE SERVIÇO NO SISTEMA LINUX

A qualidade de serviço (ou QoS, do inglês *Quality of Service*) é um método utilizado para controlar fluxos de dados sobre um canal de comunicação. Ela é responsável por dar medidas de desempenho previsíveis aos serviços oferecidos sobre esse canal; medidas que não devem se alterar com condições externas como quantidade de fluxos concorrentes ou quantidade de informações geradas [35].

Embora a QoS seja estudada principalmente na área de redes de computadores, ela pode ser aplicada também sobre outros canais de comunicação. No âmbito deste trabalho, a QoS é estudada com o objetivo de permitir a medição de desempenho de discos virtuais; sendo cada disco virtual um arquivo sobre o sistema de arquivos do disco físico.

As seções a seguir mostram os conceitos de qualidade de serviço no sistema operacional Linux. A Seção 3.1 apresenta o histórico da qualidade de serviços, os conceitos básicos e as medidas de QoS abordadas neste trabalho. A Seção 3.2 mostra o suporte de QoS disponível para o Linux. A Seção 3.3 contém uma seleção de trabalhos publicados que tratam de QoS para discos físicos e virtuais.

3.1 Qualidade de Serviço

Historicamente, as redes de computadores disponibilizam apenas o suporte do tipo melhor-esforço (*best-effort*) [51]. Neste tipo de suporte, nenhuma garantia, previsão ou diferenciação sobre o canal de comunicação pode ser assumida. Isso impede que determinados fluxos de dados funcionem corretamente sobre as redes, pois necessitam de medidas mínimas de desempenho.

Como alternativa ao suporte de melhor-esforço, grupos de trabalho da IETF (do inglês *Internet Engineering Task Force*) propõem modelos de serviços. Dentre os modelos propostos, os modelos de serviços integrados (IntServ, do inglês *Integrated Services*) [8] e de

serviços diferenciados (DiffServ, do inglês *Differentiated Services*) [6] se destacam.

O modelo de serviços integrados requer recursos de rede reservados à priori sobre o caminho de conexão. O protocolo RSVP (do inglês *Resource ReSerVation Protocol*) [40] especifica a troca de mensagens necessária para realizar a reserva de recursos do modelo IntServ. O RSVP atua na camada de rede do modelo OSI e permite que diversos transmissores enviem dados para vários grupos de receptores, auxiliando na configuração de multidifusão (*multicast*). Também pode ser utilizado para reservar capacidade de comunicação de tráfego de unidifusão (*unicast*), como *Network File System* (NFS) e gerenciamento de rede privada virtual (ou VPN, do inglês *Virtual Private Network*).

O modelo de serviços diferenciados caracteriza-se por definir classes de serviços (ou CoS, do inglês *Class of Service*). Cada classe apenas reune e diferencia o tráfego de rede, mas não possui garantias sobre os serviços. O protocolo MPLS (do inglês *Multiprotocol Label Switching*) [43] especifica a diferenciação do tráfego de rede através de CoS. O MPLS atua entre as camadas de enlace e de rede do modelo OSI e atribui um rótulo (*label*) para cada mensagem. O roteamento é baseado nesses rótulos, independentemente do fluxo ou do protocolo encapsulado pelo pacote MPLS. Uma alternativa ao protocolo MPLS é o campo TOS (*Type of Service*) do cabeçalho IPv4 ou o campo *Traffic Class* do cabeçalho IPv6.

Independentemente do suporte de qualidade de serviço disponibilizado, a QoS define critérios para a avaliação de sistemas [11]. Esses critérios são especificados através de atributos de QoS, parâmetros que indicam o desempenho de sistemas. Existem dois grupos de atributos de QoS: baseados em tecnologias (ou QoS propriamente dita) e baseados em usuários (ou qualidade de experiência - QoE).

O grupo de atributos baseados em tecnologias define características que são medidas objetivamente através de modelos matemáticos. Os principais atributos de QoS propriamente dita são: vazão (*throughput*), latência (*delay*) e flutuação (*jitter*) [11]. O grupo de atributos baseados em usuários define características que são medidas subjetivamente [26]. A QoE (do inglês *Quality of Experience*) mede a percepção dos usuários sobre um serviço. Neste trabalho, considera-se a QoS propriamente dita e os atributos vazão e

latência.

A vazão é uma medida de largura de banda (*bandwidth*) usada para descrever a quantidade de informação que pode ser transmitida em determinado intervalo de tempo. Trata-se da capacidade de comunicação de um meio. No caso do acesso a discos, tanto físicos quanto virtuais, quanto maior a largura de banda reservada, mais requisições serão atendidas por unidade de tempo. O padrão de acesso das aplicações interfere na largura de banda: aplicações de acesso aleatório a disco devem reservar o disco por um intervalo de tempo maior que aplicações de acesso sequencial para imprimirem a mesma vazão.

Acessar posições do disco aleatoriamente pode ocasionar a movimentação do braço de leitura do disco e aumentar o tempo de busca (*seek time*) para uma operação de leitura ou escrita. Caso a posição de disco desejada não seja a mesma posição apontada pelo braço de leitura no momento da operação, o braço deve se mover até o ponto pretendido. Da mesma forma, a posição de disco desejada pode estar em outra porção do disco. Neste caso, é necessário rotacionar os *platters* e movimentar o braço de leitura. Esta rotação configura a latência de rotação (*rotational delay*); ambos tempos podem aumentar o atraso do tempo de resposta de uma operação.

A latência é uma medida usada para descrever o intervalo de tempo decorrido desde o início do envio de uma informação até o início do recebimento dessa informação pelo destinatário. Não há uma relação entre vazão e latência em um canal de comunicação. Assim como em [35], assume-se que latência é o intervalo de tempo decorrido desde a criação de uma requisição até a entrega dos dados à aplicação.

Apesar de não ser tratado neste trabalho, existem outros atributos de QoS, como rajada e flutuação. A rajada é uma medida que descreve uma utilização adicional do canal de comunicação. Este atributo pode ser dividido em duração máxima da rajada e tamanho máximo da rajada. O primeiro, é o intervalo máximo de tempo que um fluxo pode utilizar o canal de comunicação acima da vazão especificada (mas não acima do tamanho máximo da rajada); retornando à vazão máxima estabelecida após esse período. O segundo, é o tamanho máximo que a largura de banda pode assumir temporariamente.

Para que o atributo rajada possa ser utilizado, é necessário que o fluxo acumule ca-

pacidade de transmissão de dados. Um fluxo que permaneça um intervalo de tempo sem utilizar o canal de comunicação, quando voltar a utilizá-lo, poderá ter uma vazão temporariamente superior à especificada, limitada à duração máxima da rajada.

A flutuação é um atributo estudado principalmente na área de redes de computadores [51] e mede a variação na recepção de dados no receptor. Fatores como alta concorrência podem causar congestionamento que, por sua vez, podem proporcionar uma variação no tempo de atendimento de requisições; podendo aumentar o atraso do serviço.

No acesso a discos, pode haver situação semelhante quanto aplicações multimídia utilizam constantemente o disco. Para atenuar o efeito da flutuação em discos, normalmente utiliza-se *buffers*. Porém *buffers* são implementações em nível de aplicação [35], enquanto o escalonamento de requisições está na camada de blocos do sistema. As seções seguintes apresentam formas de atribuir qualidade de serviço no acesso a disco no sistema operacional Linux.

3.2 Suporte de QoS em Linux

No sistema operacional Linux, existem duas formas de permitir o uso de qualidade de serviço no acesso a disco: escalonadores de disco e módulos de *kernel* auxiliares. Escalonadores de disco gerenciam as filas de requisições das aplicações do sistema; módulos de *kernel* auxiliares estendem as capacidades dos escalonadores de disco. As subseções abaixo detalham cada um desses suportes.

3.2.1 Escalonadores de Disco

Em sistemas multi-tarefas, as aplicações podem emitir requisições concorrentemente; porém os discos atendem apenas uma requisição por vez [30]. As demais requisições ainda não atendidas são mantidas em filas de requisições. O escalonador de disco é a parte do sistema responsável por gerenciar a ordem de atendimento das requisições pendentes. Os escalonadores de disco padrão do sistema Linux são: NOOP, Deadline, Anticipatory e CFQ.

O escalonador NOOP [38] realiza as operações mínimas necessárias para manipular as filas de requisições de E/S. Todas as requisições desse escalonador são armazenadas em uma mesma fila de requisições e atendidas pela ordem de chegada. O NOOP é usado principalmente em dispositivos que não são baseados em discos e ambientes de *software* e *hardware* especializados que possuem escalonamento de E/S próprio, como discos físicos com suporte TCQ (do inglês *Tagged Command Queuing*) [14].

O escalonador Deadline [38] adiciona a noção de prazo para uma requisição de E/S realizar sua operação. Cada requisição é mantida em uma dentre cinco listas diferentes: duas listas organizadas por endereço lógico do bloco de requisições, uma para leitura e outra para escrita (lista ordenada); duas listas organizadas por ordem de enfileiramento, uma para leitura e outra para escrita (lista enfileirada); uma lista de requisições a serem enviadas para o dispositivo de armazenamento (lista de envio).

Inicialmente, deve-se verificar se há requisições pendentes na lista de envio. Se sim, essas requisições são atendidas. Caso contrário, move-se um conjunto de requisições de uma das outras quatro listas para a lista de envio, o que ocorrer primeiro: (i) se há requisições de escrita pendentes e nenhuma requisição de escrita foi selecionada por um período de tempo; (ii) se há requisições de leitura com prazo ultrapassado na lista enfileirada; (iii) se há requisições de leitura pendentes na lista ordenada; (iv) se há requisições de escrita pendentes em qualquer lista.

O escalonador Anticipatory (AS, do inglês *Anticipatory Scheduler*) [22, 38] implementa uma política não conservativa de escalonamento. Quando aplicações emitem requisições, estas requisições são mantidas em um conjunto de requisições a serem atendidas. Escalonadores conservativos realizam o escalonamento e o envio de requisições sempre que houver alguma pendente. O escalonador AS espera uma certa quantidade de tempo antes de enviar uma nova requisição ao dispositivo de armazenamento.

Essa técnica de espera trata a ociosidade enganosa (*deceptive idleness* [21]) no escalonamento de requisições e aumenta a localidade de dados. Escalonadores não conservativos aguardam unidades de tempo sem realizar trabalhos à espera de novas requisições da mesma tarefa com endereços lógicos próximos aos já atendidos. Escalonadores conser-

vativos podem trocar a aplicação atual por inferir enganosamente a ociosidade da tarefa atendida.

O escalonador AS é composto por três partes: (i) escalonador de disco; (ii) núcleo de antecipação de requisições; (iii) heurística de redução de busca (*seek*). O componente (i) possui uma política de escalonamento que não possui ciência de técnicas de antecipação. O componente (ii) possui mecanismos genéricos de lógica e de temporização, dependendo do componente (iii) para inferir quando aguardar por novas requisições e por quanto tempo aguardar. Nas versões mais recentes do *kernel* Linux, o suporte ao escalonador AS foi retirado [27, 4].

O objetivo do escalonador CFQ (do inglês *Completely Fair Queuing*) [3, 38] é alocar de forma justa a largura de banda de requisições de E/S entre todas as requisições existentes. Possui um número N de filas de requisições de E/S internas e uma fila simples de envio de requisições. Cada uma das N filas recebe uma quantidade de tempo para acesso ao disco de acordo com a prioridade dos processos cujas requisições de E/S pertencem.

Durante a operação de enfileiramento, o escalonador CFQ utiliza o descritor de processos do processo atual para descobrir qual fila deve ser selecionada e enfileira a nova requisição sob a disciplina de "Primeiro a chegar é o primeiro a sair"(ou FIFO, do inglês *First In, First Out*). Durante a operação de desenfileirar, usa a disciplina *Round Robin* e percorre a primeira posição das diversas filas de processos.

3.2.2 Módulos de *Kernel* Auxiliares

Módulos de *kernel* auxiliares são módulos carregados no núcleo do sistema Linux para estender as características de QoS dos escalonadores de disco. Esses módulos auxiliares permitem configurar atributos de QoS para diferentes canais de comunicação através da diferenciação e do controle do fluxo de informações. Este trabalho faz uso do módulo auxiliar Control Groups.

O Control Groups [31] (ou cgroups) é um módulo auxiliar incorporado ao *kernel* Linux para realizar o controle de recursos e isolamento de escopo de nomes. O cgroups divide as tarefas principais e derivadas (tarefas-filhas) do sistema em subsistemas: módulos que

agrupam tarefas e parâmetros específicos para controlar as tarefas de acordo com os recursos existentes. Os subsistemas disponíveis no cgroups são: conjunto de CPUs, de memória, de tráfego de rede e de requisições de E/S para dispositivos de bloco.

Dentre os subsistemas disponíveis, o subsistema de controle para dispositivos de bloco (cgroups-blkio [41]) permite configurar garantias de execução de requisições de E/S. O cgroups-blkio possui duas abordagens de QoS: divisão proporcional de peso e largura máxima de banda do disco. A primeira, permite uma divisão justa de tempo de disco através de pesos; a segunda, controla o limite superior de banda de disco.

Mesmo garantindo tempo de CPU justo, a primeira abordagem do cgroups-blkio não permite fazer asserções diretas sobre a latência ou a vazão de tarefas; apenas permite uma divisão proporcional do tempo de disco entre tarefas. Esse controle de tempo de disco é dependente de escalonador, disponível apenas para o escalonador CFQ.

A segunda abordagem limita a largura máxima da banda de disco. Apesar de rajadas não serem consideradas neste trabalho, a limitação de banda pode comprometer o atributo. Se limitado, não é possível configurar rajadas; de outro modo, se as rajadas forem incorporadas no limite máximo de vazão, é possível que o limite maior de banda seja assumido como vazão e usado a todo instante. Nessa visão, o conceito de rajadas deixa de existir e apenas o conceito de vazão permanece.

O cgroups-blkio é utilizado no trabalho desenvolvido apenas como forma de agrupar e identificar processos ou grupo de processos. O escalonamento de requisições de discos virtuais e a qualidade de serviço disponível às VMs é baseada no conjunto de trabalhos sumarizados na próxima seção.

3.3 Qualidade de Serviço para Discos Virtuais

Esta seção possui um conjunto de trabalhos publicados pela comunidade científica que analisam a relação entre discos virtuais e os sistemas hospedeiros que os abrigam. Trabalhos que considerem alocação de E/S de disco em VMM foram desenvolvidos e publicados apenas nos últimos anos, ao contrário de trabalhos sobre alocação de CPU, memória e rede para VMM [28].

Os trabalhos [34] e [24] estudam o impacto que aplicações de computação intensiva têm sobre aplicações de E/S intensiva, com foco sobre a utilização de processador. Ongaro et al. [34] combinam computação intensiva com diferentes cargas de trabalho de aplicações de E/S intensivas, reordenando a fila de tarefas a executar tendo em consideração a quantidade de créditos remanescentes. Concluem que aplicações sensíveis à latência têm melhor desempenho quando não estão no mesmo domínio virtual que aplicações de computação intensiva.

Kim et al. [24] propõem um mecanismo de inferência para aumentar a capacidade de resposta de tarefas orientadas a E/S quando presentes em cargas de trabalho orientadas a computação intensiva com equidade completa de CPU. Desenvolvem um mecanismo de inferência com impulsionamento parcial de prioridades e com mecanismos de correlação. O impulsionamento parcial de prioridades é realizado quando o VMM infere que uma tarefa orientada a requisição de E/S está presente em uma CPU virtual e um evento de E/S acontece, aumentando a prioridade da CPU virtual pelo tempo que a tarefa trata o evento. O mecanismo de correlação auxilia o VMM a priorizar uma CPU virtual apenas na presença do evento e quando a CPU virtual tiver uma requisição de E/S para executar.

Em ambos os casos, concluiu-se que os escalonadores padrões das soluções de virtualização avaliadas priorizam a equidade para aplicações de computação intensiva e que as técnicas empregadas podem aumentar o desempenho de requisições de E/S. Porém os métodos propostos não consideram formas de estabelecer QoS sobre o canal de comunicação. Kim et al. [24] atribuem um mecanismo de inferência do tipo melhor-esforço, sem um comportamento previsível para as requisições de E/S do disco virtual.

O trabalho [7] examina se o escalonamento tradicional feito para discos magnéticos beneficia um sistema em camadas como o da virtualização. Esse trabalho realiza um estudo experimental dos escalonadores de E/S presentes no *kernel* Linux; avaliando todas as combinações de interação da dupla (escalonador-da-máquina-base; escalonador-da-máquina-virtual). Os experimentos consistem em executar *benchmarks* para avaliar o desempenho de cada dupla de escalonadores sob critérios de vazão e de equidade.

A conclusão obtida por [7] é: (i) o melhor escalonador para as VMs depende do

workload que executam; (ii) o escalonador NOOP leva a uma melhor vazão no VMM; (iii) o escalonador CFQ causa uma melhor equidade na consolidação de VMs, mas a um custo de vazão e de latência e (iv) a dupla de escalonadores padrão do Linux não é ótima.

O trabalho [23] corrobora com o descrito anteriormente. Neste, é estudado como se comporta a equidade e a latência dos discos virtuais, tendo em consideração a camada mais próxima da VM. Os escalonadores testados são modificados na máquina virtual; na máquina base varia-se dentre todos os escalonadores do Linux. São feitos três experimentos para verificar o comportamento dos discos virtuais: (i) características de latência dos discos virtuais; (ii) escalonadores AS e CFQ e (iii) escalonadores AS e CFQ modificados.

Para o primeiro experimento, a execução dos *benchmarks* mostra que a característica de latência do disco virtual depende das VMs consolidadas na plataforma compartilhada e não da característica de latência do disco físico; o segundo, revela que a escolha do escalonador na máquina base desempenha um importante papel para a equidade, tendo CFQ uma menor degradação na equidade; o terceiro, ao mudar dinamicamente os parâmetros de configuração dos escalonadores originais, consegue-se uma melhora na latência do disco virtual.

Apesar das conclusões de ambos os trabalhos, eles não definem formas de estabelecer QoS sobre a execução de máquinas virtuais. Em nenhum dos casos, é possível configurar limites ou quantidades mínimas de vazão ou latência. Ainda, o primeiro trabalho não considera *background jobs*, apenas máquinas virtuais; o segundo, usa um dos *benchmark* para simular a carga de trabalho que os *background jobs* gerariam.

O estudo em [48] regulamenta uma política de escalonamento na máquina base para aumentar a performance das máquinas virtuais. Para evitar a interferência de *background jobs*, propõe-se a criação de uma fila prioritária para processos de VMs. Cada processo da fila prioritária pode assumir um dentre três estados: URGENT (quando, pelo histórico de comportamento, faz requisições de E/S intensivas), HAVE (quando ainda tem tempo de CPU disponível - *time slices*), OVER (quando não há mais tempo de CPU disponível).

Ainda que considere a interferência de *background jobs* nas máquinas virtuais, esse trabalho trata apenas do desempenho de CPU. Mesmo que as VMs se beneficiem da dife-

renciação dos *background jobs*, não é possível prever o comportamento do disco. Kesavan et al. [23] e Boutcher et al. [7] mostram que, para um disco virtual melhorar seus atributos de performance, depende da relação de escalonadores base-virtual e do tipo de carga de trabalho gerada.

Rocha et al. [35, 42] tratam do escalonamento de disco físico considerando os atributos de QoS vazão, latência e rajadas sobre um núcleo não conservativo. Apesar de não considerar processos de máquinas virtuais, o escalonamento detalhado nesses trabalhos serve como base para a avaliação do levantamento desenvolvido nessa dissertação. O núcleo desses trabalhos é adaptado para receber processos de VMs e os detalhes do escalonador original são descritos na subseção 3.3.1.

Ling et al. [28] regulamentam QoS para o disco virtual dos sistemas computacionais e aceleram as leituras sequenciais e aleatórias em 17% e 25% respectivamente comparado ao CFQ, diminuindo o tempo de resposta das requisições. Por ser um trabalho com uma estrutura próxima à estrutura presente em [35, 42], os detalhes do trabalho [28] são descritos na subseção 3.3.2.

3.3.1 High-Throughput Token Bucket Scheduler - HTBS

O HTBS [35, 42] é um escalonador de requisições de E/S para disco físico baseado nos algoritmos BFQ [49] e pClock [19]. Do algoritmo BFQ, o HTBS utiliza o conceito de prevenção de *deceptive idleness*; do algoritmo pClock, o HTBS utiliza o conceito de configuração individual dos parâmetros de QoS.

A prevenção de *deceptive idleness* é feita com a técnica de escalonamento não conservativo. Assim como o BFQ, o HTBS aguarda determinada quantidade de tempo pelo surgimento de uma nova requisição da mesma aplicação. Diante de requisições síncronas, essa estratégia evita a ociosidade enganosa. O algoritmo pClock é conservativo: atende requisições caso haja alguma pendente.

Por ser não conservativo, o HTBS considera o padrão de acesso das filas de requisições de E/S. Requisições sequenciais possuem endereço lógico adjacentes entre duas requisições consecutivas. O mecanismo de inferência do padrão de acesso das filas no HTBS considera

apenas as primeiras requisições emitidas.

Caso a requisição atual possua endereço próximo ao da requisição anterior, a fila é marcada temporariamente como sequencial. Caso a requisição atual não possua endereço próximo ao da requisição anterior, a fila é marcada como aleatória até que o escalonador seja reconfigurado. Esse método não permite que alguma requisição aleatória seja intercalada durante a execução da carga de trabalho.

No ambiente virtual, as cargas de trabalho sequenciais podem emitir requisições aleatórias entre as requisições sequenciais. A proposta de avaliação desse trabalho sugere que, em vez de considerar apenas as primeiras requisições emitidas, considere-se um breve histórico antes de decidir qual o padrão de acesso da fila de requisições.

A configuração individual de parâmetros de QoS é feita com as técnicas de etiquetas (*tags*) e com a contagem de símbolos (*tokens*). As etiquetas permitem controlar a latência de requisições. A cada requisição é atribuída uma etiqueta de início (tempo que a requisição chegou no escalonador) e uma etiqueta de término (tempo limite para a requisição executar). A etiqueta de término é construída a partir da etiqueta de início adicionada à latência esperada para as requisições. Antes de atender uma requisição, deve-se encontrar a menor etiqueta de término dentre as requisições passíveis de envio.

A contagem de símbolos permite controlar a vazão e as rajadas. Uma fila de requisições possui determinada quantidade de símbolos disponíveis para uso. A cada envio de requisição, uma quantidade de símbolos é consumida. Novos símbolos são disponibilizados às filas quando novas requisições surgem, sendo a atribuição de símbolos proporcional à vazão configurada para a fila.

Se uma fila consome todos seus símbolos, ela registra uma quantidade negativa de símbolos proporcional à quantidade de símbolos consumida. Ter símbolos negativos faz com que as etiquetas de início recebam um acréscimo de valor e fiquem no futuro, aumentando também o valor da etiqueta de término e atrasando a escolha da fila e das requisições para envio.

No escalonador HTBS, cada fila de requisições é mantida em um grupo interno de controle. Cada grupo possui uma lista que armazena as requisições antes do envio e valores

comuns a toda fila, como vazão, rajada, latência e *tokens*. Os grupos são diferenciados de acordo com a divisão de cgroups feita na máquina base, sendo cada grupo associado a um cgroup existente.

A configuração de vazão e de símbolos do grupo está relacionada ao peso que o cgroups da requisição possui na máquina base. Conjuntos de cgroups diferentes possuem valores de vazão e símbolos distintos. Porém, originalmente, o HTBS não implementa diferenciação para os parâmetros de vazão, latência e rajadas. Todos os grupos devem ter seus valores definidos estaticamente. A proposta de avaliação desse trabalho sugere o uso de classes de valores para cada grupo interno como forma de diferenciar os parâmetros de QoS.

3.3.2 Flubber

O estudo em [28] propõe a alocação de recursos de disco para garantir latência e vazão nas VMs, usando um *framework* (Flubber) em dois níveis no VMM. O primeiro nível, mais próximo das VMs, é responsável por controlar a banda de disco das máquinas virtuais. O controle é feito por mecanismos de contagem de símbolos (nomeados créditos), regulando a taxa de vazão das requisições. Os créditos são reabastecidos em três ocasiões: (i) na troca da rodada quando créditos de todas VMs são usados; (ii) quando todas VMs mandaram requisições e as filas de pendências estão vazias; (iii) uma VM é criada ou destruída. Em todas as ocasiões, as VMs voltam ao seu crédito máximo.

No segundo nível, mais próximo do disco físico, é feito o controle de latência das requisições. O Flubber dá prioridade de execução para as máquinas virtuais mesmo em face de *background jobs*. Três filas de controle garantem a prioridade das VMs: (i) fila ordenada; (ii) fila VM-EDF; (iii) fila comuns. A primeira fila, ordenada pelo endereço lógico das requisições, possui todas as requisições do sistema, tanto requisições de VMs quanto requisições de *background jobs*. A segunda fila, organizada pelo prazo máximo de atendimento (EDF/*deadline*), possui requisições de todas as VMs. A terceira fila, organizada por ordem de chegada (*FIFO*), possui requisições dos *background jobs*.

As três filas são geridas por dois algoritmos do segundo nível: (i) algoritmo de classificação e (ii) algoritmo de lote e atraso. O primeiro algoritmo faz a classificação do tipo de

requisição criada e ainda não atendida. Se ela for originada de uma máquina virtual, um prazo é atribuído para que essa requisição se complete e a requisição é ordenada na fila VM-EDF. Caso contrário, a requisição é enfileirada na fila comuns por ordem de chegada.

O segundo algoritmo controla o atendimento de requisições. Se a fila VM-EDF possui requisições pendentes, a cabeça dessa fila é escolhida para atendimento se seu prazo expirou ou se a VM que originou a requisição é a mesma VM que originou a última requisição atendida. Senão, verifica-se na fila ordenada a possibilidade de haver uma requisição sucessiva da última atendida com endereço lógico contínuo do endereço lógico da última requisição. Se nenhum desses cenários ocorrer, um intervalo é introduzido.

Durante o intervalo, se surgir uma requisição com endereço lógico contínuo da última requisição atendida, então essa nova requisição é atendida. Se durante esse intervalo nenhuma nova requisição se encaixar nos critérios, então a cabeça da fila VM-EDF é atendida. Somente quando a fila VM-EDF estiver vazia é que a fila comuns é atendida; assim que a primeira volta a ter requisições pendentes, a fila de comuns perde a prioridade de atendimento.

O Flubber possui semelhanças com o HTBS. Primeiro, é não conservativo; o *framework* espera durante um intervalo de tempo pelo surgimento de novas requisições da mesma VM. Segundo, é possível configurar parâmetros de QoS individualmente; no caso de ambos, os parâmetros latência e vazão. Terceiro, há uma contagem de símbolos ou créditos utilizados pelas filas de requisições.

As diferenças do Flubber com relação ao HTBS estão na renovação de créditos ou símbolos e na atribuição de largura de banda para as máquinas virtuais. A renovação de créditos do Flubber ocorre sempre ao máximo estabelecido; para o HTBS, a renovação é parcial e proporcional tanto à quantidade de tempo decorrida desde o último abastecimento de símbolos quanto à largura de banda configurada para a aplicação.

Diferentemente do Flubber, as mudanças propostas no HTBS consideram uma relação direta de valores de símbolos para configurar a largura de banda, em vez de uma relação indireta. Nos experimentos do Flubber, a quantidade de créditos a ser consumida é calculada com base em um fator de intervalo (β) e a proporção de créditos entre as VMs

(c_i). Na modificação do HTBS configura-se a quantidade exata de *bytes* desejada para a largura de banda.

Apesar de semelhante às mudanças propostas no HTBS, o Flubber não é comparado diretamente aos resultados de execução do escalonador resultante das mudanças no HTBS. Flubber foi implementado em um sistema de paravirtualização (especificamente Xen [5]), enquanto a mudança sugerida no HTBS considera sistema de virtualização completa (KVM). Do Flubber, apenas é utilizado as mesmas ferramentas de testes e mesmos cenários de avaliação.

CAPÍTULO 4

VIRTUAL-HIGH-THROUGHPUT TOKEN BUCKET SCHEDULER

O problema dissertado neste trabalho é oferecer atributos de qualidade de serviço no acesso a discos virtuais baseando-se no escalonamento de requisições de entrada e saída do ambiente de virtualização no sistema base. Busca-se identificar quais mecanismos de controle são necessários para considerar processos de máquinas virtuais no escalonamento de tarefas e oferecer requisitos de QoS distintos para máquinas virtuais diferentes.

Como forma de avaliar a proposta de trabalho, um algoritmo de escalonamento de requisições de E/S é adaptado. O algoritmo escolhido para essa adaptação é o High-Troughput Token Bucket Scheduler (HTBS [35, 42]), pois trata-se de um algoritmo de escalonamento que permite ajustar requisitos de QoS de forma independente. No âmbito deste trabalho, o algoritmo resultante da adaptação recebe o nome de virtual-High-Throughput Token Bucket Scheduler (vHTBS).

As seções a seguir tratam da proposta de trabalho. A Seção 4.1 apresenta uma visão geral do problema dissertado e dos mecanismos de controle necessários para considerar processos de máquinas virtuais no escalonamento de discos. A Seção 4.2 enumera as mudanças necessárias no algoritmo base. A Seção 4.3 contém o método de predição desenvolvido para descobrir o padrão de acesso das filas de requisições de E/S.

4.1 Visão Geral

Os sistemas de armazenamento compartilhados possuem três componentes [35]: (a) um disco, capaz de atender requisições de leitura e escrita de dados a blocos de armazenamento chamados setores; (b) um escalonador de requisições de disco; (c) filas de requisições. Cada fila de requisição emite requisições de padrão aleatório ou sequencial. No padrão aleatório, não é possível descobrir com antecedência qual o próximo setor do disco a ser consultado

pela requisição seguinte. No padrão sequencial, o próximo setor acessado depende do setor atual e do tamanho de E/S realizado pela requisição atual.

A Tabela 4.1 apresenta o comportamento de filas de E/S quando submetidas a cargas de trabalho sequencial e aleatória em um ambiente virtual. As duas primeiras linhas da tabela são execuções de carga de trabalho aleatória com uma e oito *threads* respectivamente. As duas últimas linhas da tabela são execuções de carga de trabalho sequencial com uma e 16 *threads* respectivamente. O campo "Nº Requisições" representa o total de requisições emitidas pela carga de trabalho; o campo "Sequencias" é a quantidade de requisições sequenciais dentre todas as requisições emitidas; o campo "Relação" mostra, para cada padrão de acesso, a relação entre o número de requisições sequenciais e o total de requisições emitidas.

Padrão	**Número de *threads***	**Nº Requisições**	**Sequenciais**	**Relação**
Aleatório	1	38615	9223	0,2388
Aleatório	8	117921	28819	0,2443
Sequencial	1	16312	16063	0,9847
Sequencial	16	49609	48792	0,9835

Tabela 4.1: Padrão aleatório

As execuções para a carga de trabalho aleatória mostram que a relação entre a quantidade de requisições sequenciais e a quantidade de requisições totais é de aproximadamente 25%; para a carga de trabalho sequencial, a relação entre a quantidade de requisições sequenciais e a quantidade de requisições totais é de aproximadamente 98%. Portanto, uma fila de E/S é dita sequencial se a maioria de suas requisições são sequenciais; assim como uma fila de E/S é dita aleatória se a maioria de suas requisições são aleatórias.

Para permitir a configuração de atributos de QoS no escalonamento de requisições de E/S para discos virtuais no ambiente físico, primeiramente é necessário identificar quais os mecanismos de controle necessários para considerar processos de máquinas virtuais no escalonamento de discos. As características relacionadas a este problema são [28]: (i) interação entre escalonadores de disco para VMs ou níveis de abstração dos sistemas de armazenamento; (ii) isolamento de VMs e (iii) arquivos grandes como discos virtuais.

A primeira característica refere-se aos níveis de abstração dos sistemas de armazena-

mento. Em um sistema sem virtualização, a relação entre os três componentes citados é direta: uma aplicação emite requisições de E/S para o escalonador do dispositivo e o escalonador se comunica com o dispositivo solicitando a escrita ou a leitura de dados.

Em um sistema com virtualização existe ao menos um nível a mais de abstração para a relação citada. A Figura 4.1 ilustra os níveis de abstração dos sistemas de armazenamento. Do ponto de vista das VMs, os sistemas de armazenamento são dedicados e cada aplicação emite requisições a esse sistema (nível 1); do ponto de vista do sistema base, cada sistema de armazenamento de máquina virtual é um subconjunto do seu próprio sistema e as aplicações virtuais emitem requisições a seus respectivos espaços de armazenamento (nível 0). Caso o sistema virtual possua tecnologia de virtualização, os níveis de abstração aumentam recursivamente.

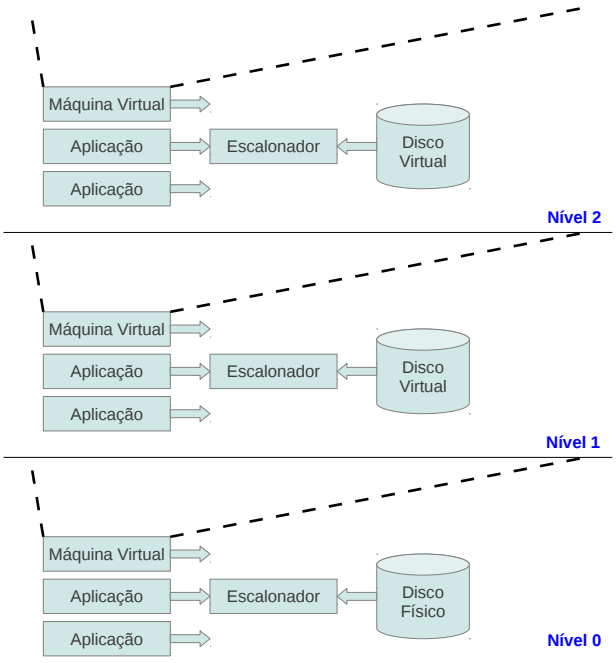


Figura 4.1: Níveis de abstração dos sistemas de armazenamento

A primeira característica é tratada pelo próprio sistema operacional e depende do tipo de virtualização utilizado. Se for um sistema de virtualização completa, o VMM intercepta a syscall que acessa o disco virtual ou realiza a tradução binária para transformar a requisição virtual em uma requisição de ambiente físico. Se for um sistema de

paravirtualização, o VMM gerencia a hypercall até que a requisição seja completada.

Mesmo com a intervenção do VMM para o tratamento da requisição virtual, é importante que o escalonador da máquina base trate as requisições de VMs de modo diferenciado. Um escalonamento virtual (realizado dentro da VM - nível 1) pode ser prejudicado ou desfeito quando chegar ao escalonamento real (realizado na máquina base - nível 0). O escalonador da máquina base deve controlar a ordem e a quantidade de requisições por unidade de tempo para permitir uma otimização do fluxo de requisições virtuais.

A segunda característica refere-se ao isolamento de VMs. Quando máquinas virtuais executam suas tarefas, é necessário que um ambiente virtual não tenha conhecimento do ambiente virtual de outra máquina. Cada VM deve ser autônoma tal qual um computador real que, a princípio, só se comunica com outro computador através de sua placa de rede.

O isolamento de VMs pode não atender a ordem de prioridades dos processos. Um processo mais prioritário dentro de uma VM pode ter que esperar por um processo menos prioritário executar primeiro na máquina física [48]. Isso ocorre quando o processo da máquina física tem uma prioridade menor que o processo virtual, mas uma prioridade maior que a VM com quem compete.

Esse caso pode ser tratado com a priorização das requisições ou das filas de requisições. Mesmo que um processo no sistema base seja eleito para executar antes de um processo de uma VM com prioridade global maior, se o escalonador priorizar processos de máquinas virtuais, a requisição originada pela VM será atendida antes da requisição do sistema base.

A terceira característica refere-se a como os discos são mapeados para o sistema base. Na maioria dos casos, as VMs possuem seu disco mapeado para arquivos grandes no sistema de arquivos da máquina base. Estes arquivos de disco estão sujeitos a fatores como *seek time*, latência de rotação, posição de dados na superfície do disco físico e caches [1, 9].

Assim como as VMs estão para as aplicações do sistema base, os discos virtuais estão para os arquivos que as aplicações do sistema base manipulam. É importante considerar padrões de acessos que as VMs terão em seus discos virtuais. Para cargas de trabalho

que predominem padrão de acesso sequencial, é importante considerar a localidade dos dados. Nesses cenários faz-se importante manter um escalonador não conservativo em vez de conservativo.

A próxima seção mostra a síntese dessas características com suas respectivas formas de tratamento mencionadas nesta seção. O escalonador vHTBS é definido e as adaptações necessárias no algoritmo base são discriminadas.

4.2 Adaptações no Algoritmo HTBS

Conforme a seção anterior, para se considerar processos de máquinas virtuais no escalonamento de tarefas de E/S de disco, é necessário que: (i) as requisições de E/S originadas dos ambientes virtuais sejam reconhecidas; (ii) as requisições de E/S para discos virtuais sejam priorizadas; (iii) o padrão de acessos das aplicações virtuais seja considerado. As considerações são assim tratadas: (i) através do agrupamento de tarefas, (ii) através de mecanismos de QoS e (iii) através de mecanismo de predição de padrão de acesso.

4.2.1 Agrupamento de Tarefas

A divisão lógica de tarefas de VMs para este trabalho é vista em duas partes: controles na máquina base e controles no escalonador de requisições de E/S. A primeira parte estrutura as tarefas no sistema de arquivos do sistema base, a segunda parte estrutura as tarefas no escalonador de requisições de E/S.

Do lado do sistema base, a identificação de processos de máquinas virtuais é feita com rótulos de cgroups. Cada VM possui um rótulo diferente do rótulo raiz; podendo ser um mesmo rótulo para processos de mesma origem ou rótulos distintos para processos de origens diferentes. Do lado do escalonador, cada VM é identificada pelo cgroups da máquina base e suas requisições são armazenadas em um grupo de controle que representa a fila de requisições para esse processo de máquina virtual.

O algoritmo HTBS não permite uma diferenciação completa entre máquinas virtuais. Caso haja cgroups habilitado na máquina base, o algoritmo vai identificar o grupo de

cgroups, mas vai tratar as tarefas de todas as máquinas virtuais como pertencentes a um único grupo. O comportamento padrão do HTBS é atribuir os mesmos atributos de QoS a todas as VMs.

A primeira adaptação no HTBS é permitir mais de um grupo de processos configurado no sistema. Para que as VMs tenham atributos diferentes, é necessário que seus atributos sejam armazenados para consulta e comparação com os mecanismos de tratamento de requisições. Originalmente, no HTBS todos os processos do sistema base têm os mesmos atributos de QoS, inclusive os processos de VMs. Essa diferenciação recebe o nome de *classe de escalonamento* no escopo desta proposta.

A Figura 4.2 exemplifica o conceito de classe de escalonamento. Cada classe de escalonamento tem quatro parâmetros de configuração: (i) nome de cgroups; (ii) vazão; (iii) latência; (iv) rajadas. O formato de dados desses atributos são: (i) refere-se ao nome de cgroups que a classe está associada, em formato de cadeia de caracteres (*string*); (ii) é a quantidade esperada de vazão da classe, em B/s (*bytes* por segundo); (iii) é a latência esperada, em ms (milissegundos); e (iv) é a quantidade de rajadas da classe, em B/s.

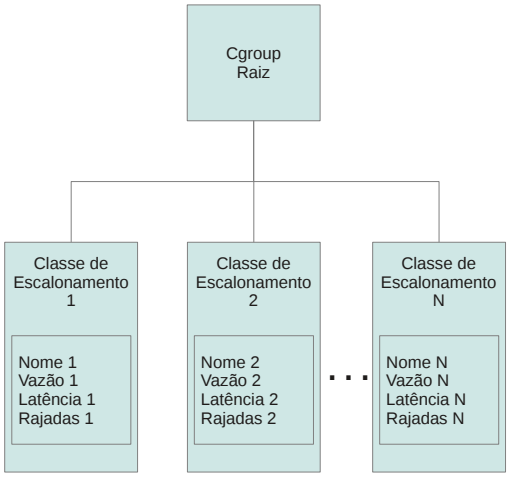


Figura 4.2: Classe de escalonamento

O vHTBS cria um diretório para cada classe de escalonamento no sistema de arquivos da máquina base e mantém um arquivo para cada atributo de uma classe. Esses dados são utilizados na criação de um novo grupo do HTBS, quando uma nova requisição chega

para o escalonador e ela pertence a um grupo que ainda não teve requisições escalonadas. A partir deste ponto, essa e futuras requisições do mesmo grupo são atendidas segundo os parâmetros escolhidos.

A segunda adaptação no HTBS é a contagem de símbolos. A contagem de símbolos original é realizada sobre a unidade de medida IOPS (E/S por segundo, do inglês *Input/Output Per Second*), sendo consumido um símbolo a cada emissão de requisição. Os testes do HTBS consideram que as requisições possuem tamanhos iguais; contudo, o tamanho das requisições podem variar de acordo com a carga de trabalho.

Se os dados da contagem de símbolos forem unidades, eles representam a quantidade de requisições de E/S que uma fila pode emitir por unidade de tempo; se forem unidades de informação digital (por exemplo, *bytes*), representam a quantidade de informação que uma fila pode emitir ou recuperar por unidade de tempo. Com a primeira, é necessário converter a quantidade de informação desejada em quantidade de requisições de E/S equivalente; com a segunda, a contagem é direta.

Optou-se pela segunda forma como método de contagem de símbolos. A cada requisição emitida, o valor do tamanho da requisição é decrescido do acumulado de símbolos disponíveis. Isso permite que as filas considerem requisições de E/S de tamanho variado, consumindo símbolos proporcionalmente ao tamanho de cada E/S.

4.2.2 Mecanismos de QoS

Os atributos considerados neste trabalho são: vazão e latência. Cada atributo possui mecanismos capazes de priorizar uma determinada fila de requisições. Para o atributo vazão, a priorização acontece através da contagem de símbolos; para o atributo latência, a priorização acontece através de etiquetas.

A contagem de símbolos pode controlar a vazão e as rajadas das filas de requisições. Pode-se representar a quantidade máxima de vazão com um valor positivo para a quantidade de símbolos. A medida que as requisições são emitidas, a quantidade de símbolos restantes é decrescida. Quanto maior a quantidade de símbolos, mais emissões uma fila pode realizar; consequentemente, maior pode ser sua vazão. Quanto menor, menos emissões

são realizadas, permitindo a emissão de outras filas.

As etiquetas podem controlar a latência das filas de requisições. Assumindo-se um valor inicial para uma requisição e um valor de latência para todas as requisições (latência da fila), pode-se calcular um valor máximo para término (*deadline*) de cada requisição adicionando-se a latência da fila ao valor inicial da requisição.

Requisições mais próximas ou que ultrapassaram o valor de término devem executar preferencialmente. Requisições mais distantes do valor de término podem ter sua emissão retardada para priorizar as demais filas. Para se priorizar determinada fila, basta reduzir seu valor de latência; cada requisição dessa fila produz um valor de *deadline* menor em relação às demais requisições de filas concorrentes.

O HTBS já possui esses mecanismos de priorização de filas, bastando apenas definir os valores apropriados a cada um dos atributos de QoS. Porém o algoritmo original não possui um mecanismo de predição do padrão de acesso das filas de E/S que considere requisições aleatórias intercaladas às requisições sequenciais. A seguir, descreve-se o mecanismo de predição desenvolvido no vHTBS.

4.3 Mecanismo de Predição de Padrão de Acesso de Filas de E/S

No HTBS, a predição do padrão de acesso das filas considera apenas as primeiras requisições de E/S. O pseudo-código abaixo apresenta o mecanismo de predição de padrão de acesso do HTBS (Algoritmo 1). Se o próximo setor esperado, calculado com base na requisição anterior, estiver calculado e a fila tiver histórico de acessos sequenciais, então é verificado se a predição de próximo setor está correta. Se estiver correta, a fila continua com o padrão sequencial; caso contrário, a fila é marcada como não sequencial.

O próximo setor esperado a ser descoberto, calculado com base na requisição atual, só é obtido se a fila mantém um padrão de acesso sequencial. Uma vez marcada como não sequencial, a predição de próximo setor não é mais calculada e a fila se mantém como não sequencial até que o escalonador seja reconfigurado.

```
1 if (Existe proximo_setor_esperado && fila_eh_sequencial) then
2 |  if proximo_setor_esperado != setor_atual then
3 |  |  Marca fila como não sequencial;
4 |  end
5 end
6 if (fila_eh_sequencial) then
7 |  proximo_setor_esperado = setor_atual + qtde_setores_atual;
8 end
```

Algoritmo 1: Mecanismo de predição de padrão de acesso do HTBS

Este método de predição de padrão de acesso não permite que as filas emitam requisições não sequenciais. Nessa configuração, as filas devem emitir requisições sequenciais desde o início até o fim da carga de trabalho. Contudo, uma carga de trabalho para ser considerada sequencial, deve emitir a maioria de requisições sequenciais e não necessariamente a totalidade. Por isso, a terceira adaptação no HTBS é a criação de um mecanismo de predição de padrão de acesso das filas de E/S.

4.3.1 Algoritmo de Predição

O algoritmo de predição proposto considera a contagem de requisições emitidas desde o início da carga de trabalho até um valor de limite menor que o fim da execução da carga de trabalho. Nesse algoritmo, uma carga de trabalho sequencial pode emitir requisições aleatórias e ser considerada sequencial, desde que a maioria de suas requisições sejam sequenciais.

O Algoritmo 2 contém o pseudo-código do mecanismo de predição de padrão de acesso desenvolvido para o vHTBS. Independentemente do padrão de acesso da fila estar definido, três ações acontecem pelo tempo que a carga de trabalho durar: contagem do número de requisições emitidas, previsão do próximo setor esperado e contagem de acertos da predição.

A contagem do número de requisições emitidas acontece até o padrão da fila ser considerado definido (linha 4). A previsão do próximo setor esperado (linha 22) é feita com a soma do setor base da requisição atual e da quantidades de setores utilizados pela requisição atual. A contagem de acertos da predição (linha 2) guarda a quantidade de

```
1  if (setor_atual == proximo_setor_esperado) then
2  |  Incrementa quantidade_de_acertos;
3  end
4  if (num_req_emitidas <= LIMITE) then
5  |  A cada PORCAO porções de requisições begin
6  |  |  fator = num_req_emitidas / PORCAO;
7  |  |  fronteira_minima = num_req_emitidas * peso[fator];
8  |  |  if (quantidade_de_acertos >= fronteira_minima) then
9  |  |  |  acumulador_fase_de_acertos = acumulador_fase_de_acertos + fator;
10 |  |  |  Marca fila temporariamente como sequencial
11 |  |  else
12 |  |  |  Marca fila temporariamente como aleatória
13 |  |  end
14 |  |  if (fator == MAX_RODADAS) then
15 |  |  |  Marca padrão como definido;
16 |  |  |  if (acumulador_fase_de_acertos > LIMIAR) then
17 |  |  |  |  Marca fila como sequencial;
18 |  |  |  end
19 |  |  end
20 |  end
21 end
22 proximo_setor_esperado = setor_atual + qtde_setores_atual;
```

Algoritmo 2: Mecanismo de predição de padrão de acesso proposto no vHTBS

acertos obtida pelo algoritmo de predição. Um acerto é contado quando o setor da requisição atual é idêntico ao próximo setor esperado calculado da requisição anterior.

O mecanismo de predição faz asserções sobre o padrão de acesso a cada porção de requisições (linha 5). Em cada ponto de asserção, o mecanismo verifica uma fronteira mínima aceitável para o número de requisições até o momento e acumula um peso se a quantidade de acertos atingiu essa fronteira mínima (linha 9).

A fronteira mínima é calculada com base no fator da porção de requisições. O fator é a n-ésima porção verificada e indexa um vetor de pesos para cada porção (linha 7). A fronteira mínima constitui-se como o M-percentil da quantidade de requisições emitidas. Caso a quantidade de acertos tenha atingido o patamar de M acumulador de acertos.

O algoritmo infere o padrão de acesso após MAX_RODADAS rodadas de predição (linha 14). O padrão de acessos é marcado como definido e somente se o valor acumulado na fase de acertos for menor que um determinado limiar (linha 16), a fila é marcada como não sequencial.

Com a mudança no algoritmo de predição de padrão de acesso das filas de E/S e com o agrupamento de tarefas, o vHTBS aproveita os mecanismos de QoS do HTBS e atende máquinas virtuais, respectivamente. Isso permite configurar medidas de QoS diferentes para máquinas virtuais diferentes, desde que estejam em classes de escalonamento distintas. O próximo capítulo demonstra experimentalmente a combinação desses fatores no escalonamento de requisições de E/S para discos virtuais.

CAPÍTULO 5

RESULTADOS EXPERIMENTAIS

O capítulo anterior enumerou as características necessárias para escalonar requisições de discos virtuais e dar atributos de qualidade de serviço às máquinas virtuais. As características identificadas para atender a esses requisitos são o agrupamento de tarefas, mecanismos de QoS (por exemplo, etiquetas e contagem de símbolos) e a predição do padrão de acesso das filas de E/S.

O capítulo atual apresenta os resultados experimentais da execução de cargas de trabalho aleatórias e sequenciais para diferentes escalonadores de discos. Com os testes, pretende-se mostrar a possibilidade de configurar atributos de QoS individuais para máquinas virtuais diferentes. Pretende-se também classificar os fatores de influência no atendimento de requisições virtuais no escalonamento do sistema base.

Os escalonadores de disco avaliados nos testes são: (i) CFQ, (ii) Deadline, (iii) HTBS e (iv) vHTBS. O escalonador (i) é o padrão do sistema Linux com a capacidade de dividir proporcionalmente o tempo de acesso ao disco através de pesos (cgroups-blkio); (ii) é o escalonador presente no *kernel* Linux, sem modificações; (iii) é o algoritmo base original deste trabalho e (iv) é o algoritmo base deste trabalho com as extensões descritas no capítulo anterior.

As seções a seguir detalham os cenários de testes. A Seção 5.1 detalha o ambiente de testes e as configurações das cargas de trabalho para os testes realizados. As Seções 5.2 e 5.3 mostram os testes realizados para comprovar a influência dos parâmetros de configuração T_{wait} e B_{max} no escalonamento de requisições virtuais. A Seção 5.4 avalia a eficiência de atributos de QoS individuais para máquinas virtuais. Por fim, a Seção 5.5 compara os resultados dos diversos escalonadores avaliados com o vHTBS.

5.1 Ambiente de Testes

A máquina física utilizada para os testes possui as seguintes configurações de *hardware*: Dell OptiPlex 990, 64 bits, Intel Core i5 3.10GHz, suporte a virtualização (*flag* vmx), 4 GB de memória RAM, disco rígido SATA Seagate Barracuda 7200 RPM de 500 GB com suporte a NCQ (reordenamento interno ao dispositivo de armazenamento objetivando a redução de latências decorrente de movimentos mecânicos [13]). Ter o suporte a NCQ habilitado não apresentou diferenças significativas na vazão das VMs comparado ao suporte a NCQ desabilitado. As configurações de *software* são: Linux Kubuntu 11.10, *kernel* 3.3.7 com cgroups-blkio, KVM versão 0.14.1 .

As configurações de máquinas virtuais são: 256 MB de memória RAM e 100 GB de disco virtual com formato RAW. Todos os arquivos de imagem dos discos virtuais são mantidos em um disco rígido SATA Seagate Barracuda ES de 750 GB dividido em quatro partições. Cada imagem de disco virtual está armazenada em uma partição diferente para minimizar os efeitos de *merge* durante o escalonamento de requisições.

O formato de disco virtual RAW foi escolhido pois, dentre os tipos suportados pelo KVM, é o único formato que permite a montagem do SO virtual em um ponto de montagem no sistema base. Essa característica permite realizar testes com os *benchmarks* nas mesmas regiões de disco que o sistema virtual ocupa sem possuir a sobrecarga das camadas de virtualização.

O sistema operacional das VMs é Debian 7.2.0 com *kernel* padrão da distribuição. Evita-se a *page cache* do sistema base (opção *cache=none* do KVM) e usa-se a biblioteca nativa do Linux para E/S assíncrono (opção *aio=native* do KVM). Os discos virtuais utilizam a interface VirtIO. As demais configurações das máquinas virtuais são as padrões.

O objetivo inicial dos testes era executar cenários semelhantes aos do *framework* Flubber. Os testes iniciais foram feitos com as ferramentas de *benchmark sysbench* [46] e *filebench* [17]. O *sysbench* foi utilizado para gerar a carga de trabalho de cada VM consolidada; o *filebench* foi utilizado para gerar a carga de trabalho dos processos concorrentes na máquina base (*background processes*).

Para os testes iniciais, foi considerado ambientes *mono-thread* e *multi-threads*, com e

sem *background jobs*. Porém o *benchmark* utilizado não permite a configuração do tempo de execução dos testes acima do tamanho dos arquivos lidos, nem de forma cíclica. As máquinas virtuais iniciavam ao mesmo tempo, porém terminavam em instantes diferentes.

Nos testes iniciais, o *cache* do sistema base não foi evitado e a biblioteca de E/S assíncrona utilizada foi a POSIX AIO em vez da Linux AIO. A POSIX AIO é uma biblioteca de E/S assíncrona que utiliza um *pool* de *threads* para simular o comportamento do dispositivo [15]. Essa escolha tornou os testes imprecisos e de difícil compreensão do comportamento do escalonador vHTBS. Os testes realizados consideram outra ferramenta de *benchmark*. Os testes iniciais completos estão disponíveis no Anexo A.

Os testes realizados utilizam o *fio* [2], uma ferramenta para *benchmark* ou estresse de *hardware* com requisições de E/S. O *fio* permite configurar o tipo de requisição de E/S (síncrono ou assíncrono), o padrão de acesso das requisições (aleatório ou sequencial), o modo de acesso (leituras ou escritas), tamanho dos arquivos, tamanho dos blocos de acesso, quantidades de arquivos ou trabalhos concorrentes e duração dos testes.

Nos testes, o *fio* é configurado para gerar cargas de trabalho aleatória e sequencial; ambas sintéticas. A execução da ferramenta de *benchmark* é realizada em um subconjunto das seguintes máquinas virtuais: kvm1 e kvm2, VMs com padrão de acesso aleatório; kvm3 e kvm4, VMs com padrão de acesso sequencial.

De modo geral, as máquinas sequenciais possuem configuração de 30 MB/s para largura de banda e 1 milissegundo para latência; as máquinas aleatórias possuem configuração de 0,5 MB/s de largura de banda e 25 ms de latência. Internamente ao escalonador vHTBS, a configuração do padrão de acesso da fila de requisições da máquina virtual kvm3 é marcado como sequencial; o padrão de acesso da fila de requisições da máquina virtual kvm4 é marcado tanto sequencial como aleatório em execuções diferentes.

Por padrão, cada VM faz a leitura completa de um arquivo do seu sistema de arquivos virtual com tamanho total de 2 GB. O tamanho de blocos das requisições virtuais é de 16 KB. Todos os testes consideram apenas leituras e as leituras dos arquivos pelas VMs são baseadas em tempo de execução: a carga de trabalho é configurada para executar por um minuto; se dentro de um minuto todo o arquivo for lido, a carga de trabalho reinicia

a leitura dos blocos do arquivo.

Ambas máquinas virtuais iniciam a execução no mesmo instante. Antes da execução, as VMs e a máquina base são sincronizadas com NTP [33] e as informações de *cache* dos sistemas virtuais são apagadas. Os resultados mostrados são a média de três execuções de cada carga de trabalho.

Os testes realizados com a ferramenta *fio* estão agrupados em três categorias: influência do parâmetro T_{wait}, influência do parâmetro B_{max} e avaliação de atributos individuais de QoS. As duas primeiras categorias possuem testes para classificar os fatores de influência no atendimento de requisições virtuais, nomeadamente os atributos T_{wait} e B_{max}. A terceira categoria apresenta testes que comprovam a possibilidade de configurar e atender requisitos de QoS individuais a VMs diferentes.

5.2 Influência do Parâmetro T_{wait}

O parâmetro de configuração T_{wait} dos algoritmos HTBS e vHTBS define o tempo máximo que uma aplicação ativa, sequencial e com símbolos disponíveis pode aguardar pelo surgimento de novas requisições síncronas e sequenciais. Se uma requisição da aplicação ativa surgir durante esse tempo de espera, o escalonador deve atender a nova requisição; caso contrário, o escalonador muda a aplicação ativa.

A Figura 5.1 apresenta a vazão das máquinas virtuais sequenciais diante de diferentes valores para T_{wait}. O tempo configurado para que uma aplicação espere pela chegada de novas requisições síncronas e sequenciais varia de 10 milissegundos até 1 segundo. Ambas máquinas virtuais são marcadas como sequenciais no escalonador e são configuradas com 20 requisições sequenciais consecutivas (B_{max}).

Um tempo de ociosidade baixo faz com que o escalonador troque a aplicação ativa por outra, perdendo a localidade dos dados. Se o tempo que a aplicação for mantida ociosa à espera de novas requisições sequenciais e síncronas for menor que o tempo de criação de novas requisições, o escalonador troca a aplicação ativa. Esse comportamento é observado com T_{wait} igual a 10 ms e 20 ms.

Com o tempo mínimo de 30 ms de espera por novas requisições, a carga de trabalho

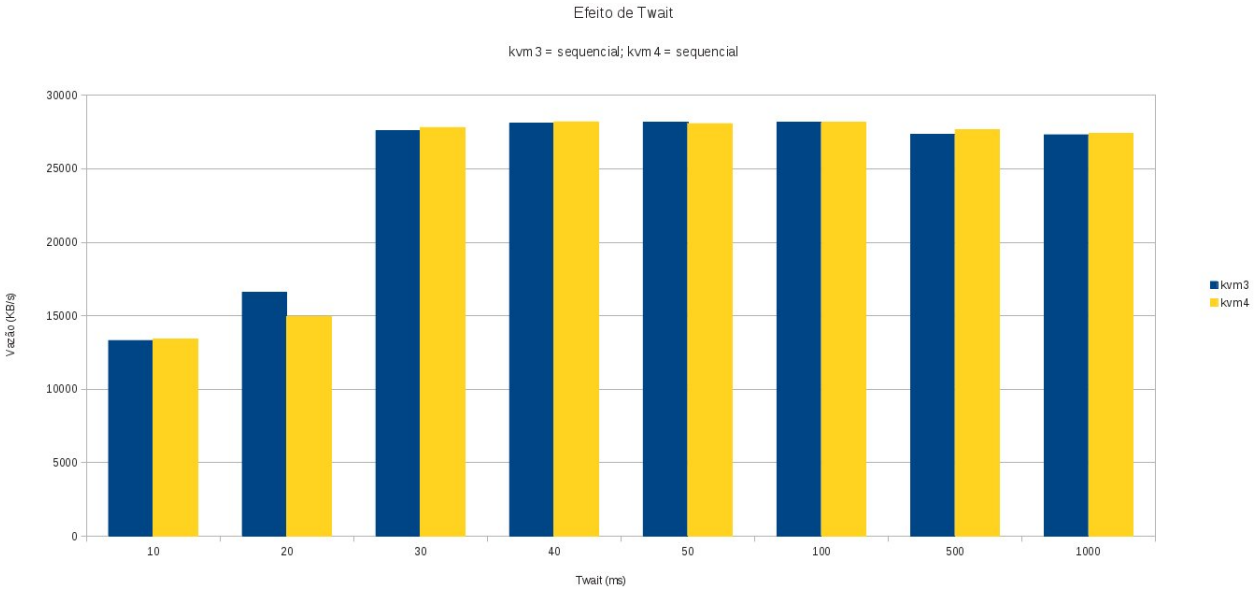


Figura 5.1: Efeito de T_{wait} na vazão máquinas virtuais sequenciais

das máquinas virtuais aproveita a localidade de dados. Nessas condições, a vazão das VMs está limitada apenas à quantidade máxima de requisições sequenciais consecutivas que podem ser enviadas antes de trocar a aplicação ativa.

A Figura 5.2 apresenta a vazão das mesmas máquinas virtuais sequenciais; porém, internamente ao escalonador vHTBS, kvm3 é mantida sequencial e kvm4 é marcada como de padrão de acesso aleatório. O tempo configurado para que a aplicação sequencial espere pela chegada de novas requisições varia de 10 milissegundos até 1 segundo. Ambas VMs são configuradas com 30 MB/s de largura de banda e B_{max} igual a 20 requisições.

O comportamento padrão do escalonador é priorizar a vazão das cargas de trabalho sequenciais. Ao aguardar T_{wait} milissegundos pelo surgimento de novas requisições e enviar até B_{max} requisições consecutivas antes de trocar a aplicação ativa, kvm3 mantém a vazão estabelecida desde que T_{wait} seja compatível com o tempo de surgimento de novas requisições sequenciais e síncronas.

Da mesma forma que com ambas máquinas virtuais marcadas como sequenciais, é necessário no mínimo 30 ms de espera da VM marcada como sequencial para que o escalonador evite a ociosidade enganosa e troque a aplicação ativa. A espera em ociosidade permite que novas requisições surjam e conserve a localidade de dados no atendimento de

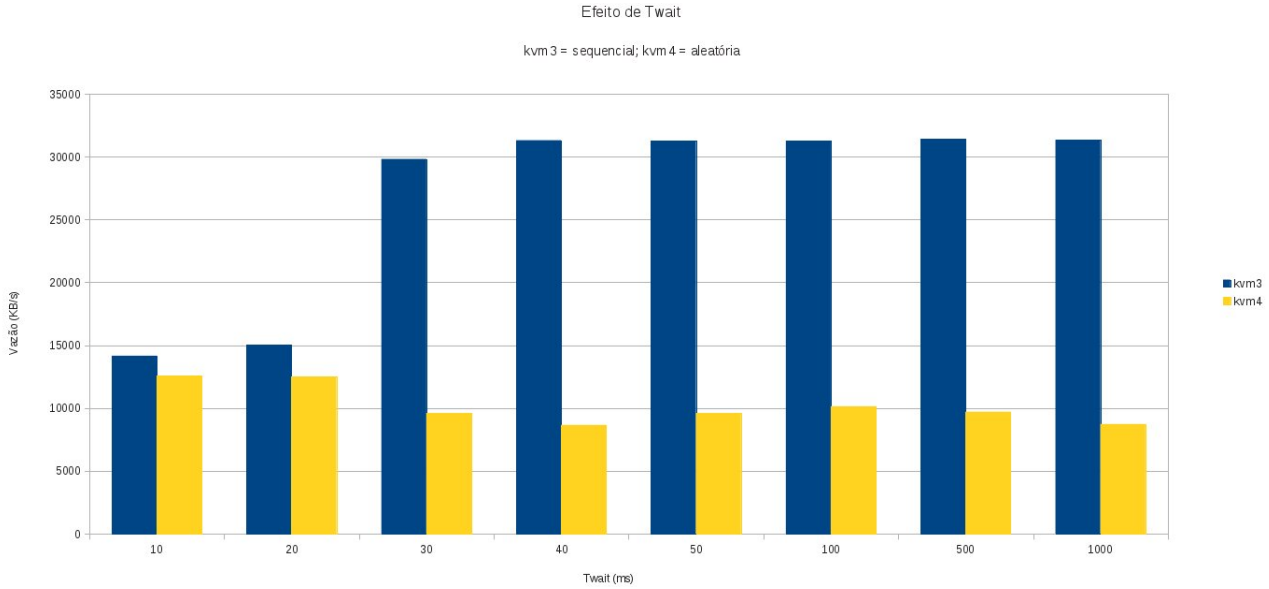


Figura 5.2: Efeito de T_{wait} na vazão máquinas virtuais mistas

requisições.

5.3 Influência do Parâmetro B_{max}

O parâmetro de configuração B_{max} dos algoritmos HTBS e vHTBS define a quantidade máxima de requisições sequenciais consecutivas que uma aplicação ativa, sequencial e com símbolos disponíveis pode emitir. Após atingir esse limite, o escalonador deve trocar a aplicação ativa e atender no mínimo uma requisição da nova aplicação ativa, caso haja requisições pendentes.

A Figura 5.3 mostra a vazão das máquinas virtuais sequenciais diante de diferentes valores para B_{max}. A quantidade máxima de requisições sequenciais consecutivas que uma aplicação ativa pode emitir varia de 10 a 1000 requisições. Ambas máquinas virtuais são marcadas como sequenciais no escalonador e são configuradas com 50 milissegundos de tempo de espera (T_{wait}).

Quanto menor o valor de B_{max}, menor é o valor de vazão agregado das máquinas virtuais. Com B_{max} igual a 10 requisições, o valor de vazão agregado é de aproximadamente 50 MB/s para ambas VMs; com B_{max} igual a 40 requisições, o valor de vazão agregado é de aproximadamente 60 MB/s. O maior valor de vazão agregada obtido é de

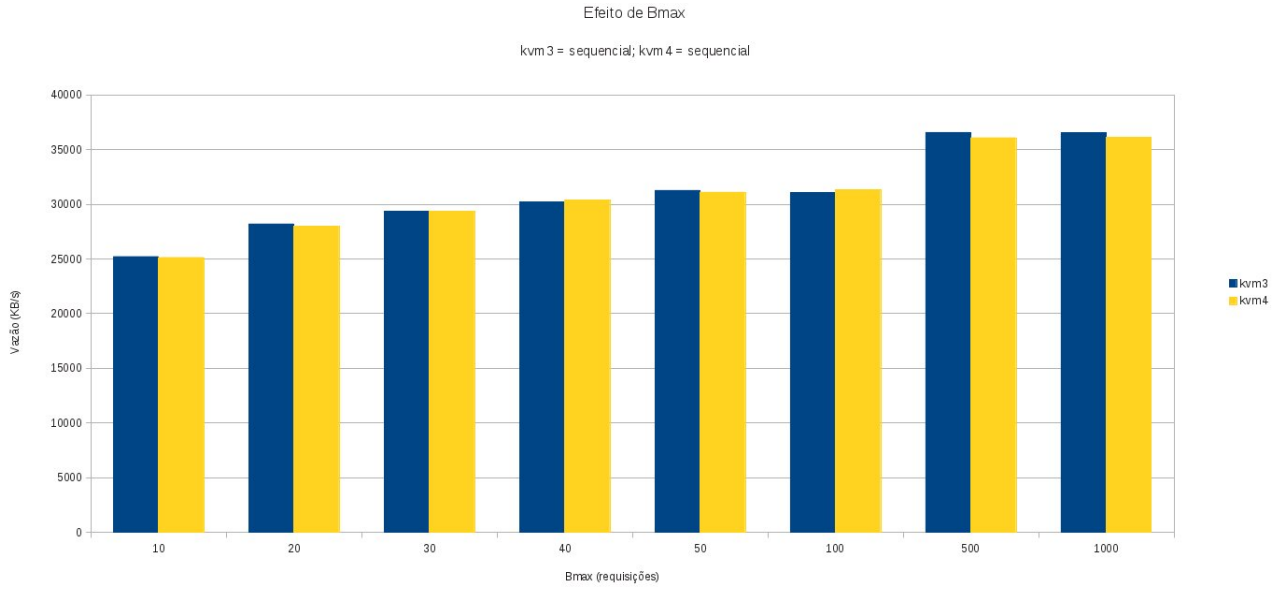


Figura 5.3: Efeito de B_{max} na vazão máquinas virtuais sequenciais

aproximadamente 70 MB/s, valor próximo do limite do disco.

O menor valor de vazão agregado, com B_{max} igual a 10 requisições, justifica-se pelo baixo acúmulo de requisições por uma aplicação ativa. Nos testes, supondo que sempre haja demanda por requisições das máquinas virtuais, se kvm3 executar primeiro, 10 requisições de kvm3 serão atendidas antes do escalonador trocar a aplicação ativa para kvm4. A VM kvm4, por sua vez, enviará 10 requisições antes de kvm3 voltar a enviar requisições.

À medida que o valor de B_{max} aumenta, maior é o acúmulo de requisições pela aplicação ativa. Porém, a possibilidade de acumular muitas requisições faz com que as demais aplicações fiquem à espera de utilização do meio. Esperar que várias requisições sequenciais consecutivas sejam atendidas pode causar inanição das demais requisições de outras filas [35, 42].

A Figura 5.4 apresenta a vazão das mesmas máquinas virtuais sequenciais; porém, internamente ao escalonador vHTBS, kvm3 é mantida sequencial e kvm4 é marcada como de padrão de acesso aleatório. A quantidade máxima de requisições sequenciais consecutivas varia de 10 a 1000 requisições. Ambas VMs são configuradas com 30 MB/s de largura de banda e T_{wait} igual a 50 milissegundos.

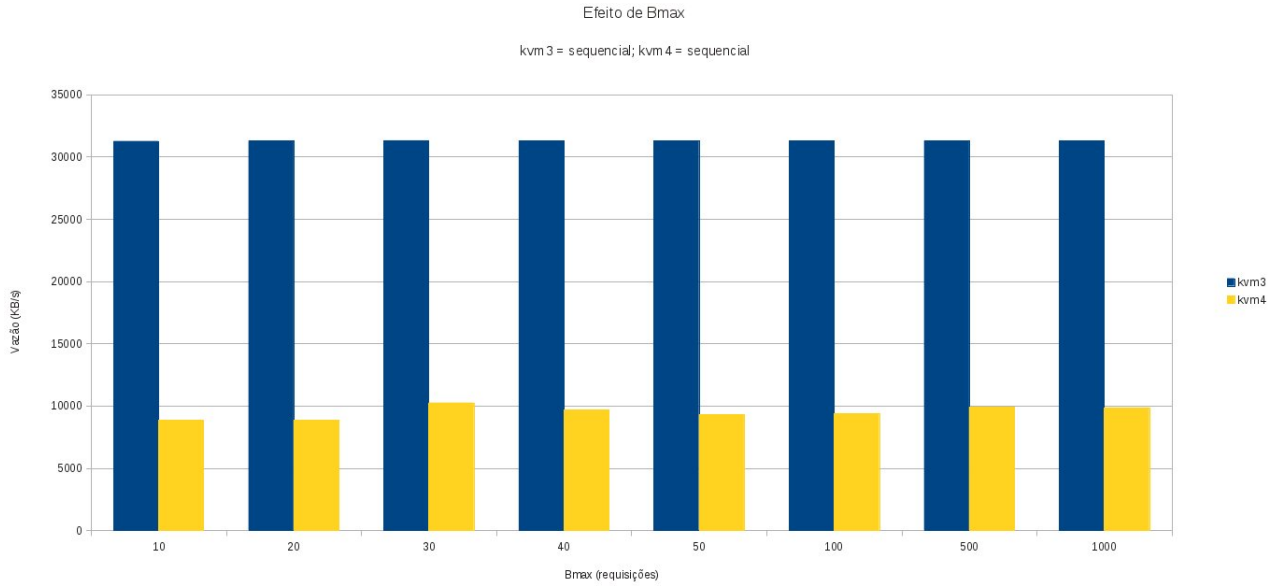


Figura 5.4: Efeito de B_{max} na vazão máquinas virtuais mistas

Apesar de variar o valor de B_{max}, a vazão de kvm3 não sofre interferência. O escalonador prioriza requisições sequenciais em face de requisições aleatórias: as requisições de kvm3, limitadas à quantidade máxima de B_{max} são atendidas antes das requisições de kvm4. Quando o controle do dispositivo está com kvm4, esta envia tantas quantas requisições existirem antes que kvm3 solicite serviço novamente. Quando houver demanda de kvm3, o escalonador permitirá que até o limite de requisições seja emitido antes de trocar a aplicação ativa.

5.4 Avaliação de Parâmetros Individuais de QoS

O escalonador vHTBS permite a configuração dos atributos de QoS largura de banda e latência. A largura de banda é a quantidade de vazão mínima estipulada para uma aplicação ou VM. A latência é a quantidade máxima de tempo entre o surgimento de uma requisição até o término da requisição na fila do dispositivo. O vHTBS permitem garantir valores para esses atributos de QoS.

5.4.1 Largura de Banda do vHTBS

Para a largura de banda, as máquinas virtuais foram configuradas com valores variáveis de largura de banda. A máquina virtual kvm3 varia entre 10 MB/s, 15 MB/s, 30 MB/s, 45 MB/s e 60 MB/s, enquanto a máquina virtual kvm4 mantém 15 MB/s. Adicionalmente, ambas VMs variam entre 1 MB/s, 30 MB/s e 60 MB/s de largura de banda simultaneamente. Ambas máquinas virtuais são consideradas sequenciais pelo escalonador. O valor de T_{wait} e B_{max} configurados são 30 ms e 30 requisições respectivamente.

A Figura 5.5 apresenta a vazão das máquinas virtuais sequenciais em face da largura de banda estipulada. Existe um valor mínimo e um valor máximo de largura de banda configurável para o escalonador HTBS. Dentro desse intervalo, é possível dizer exatamente qual a vazão mínima que uma VM terá. Nos testes realizados, o valor mínimo garantido é de 15 MB/s; o valor máximo, é de aproximadamente 30 MB/s para larguras de banda iguais e aproximadamente 35 MB/s para larguras de banda diferentes.

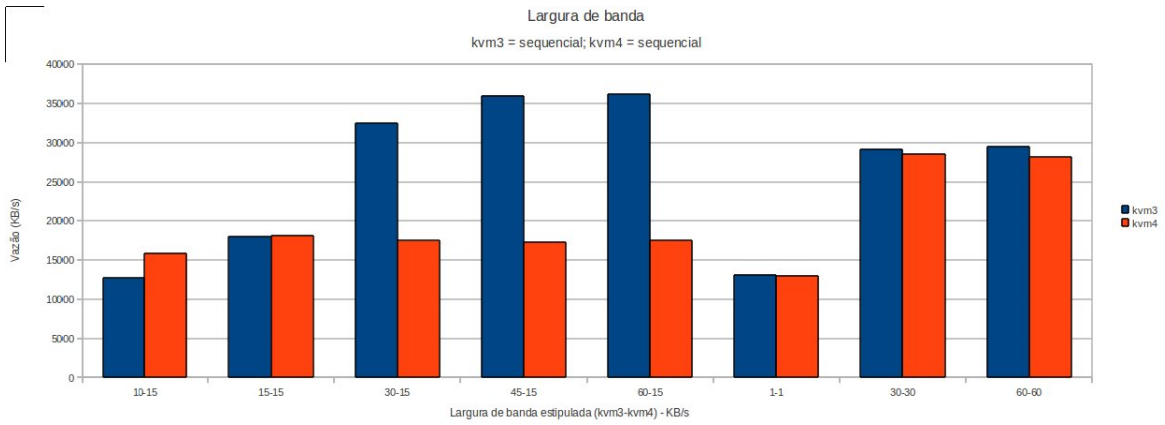
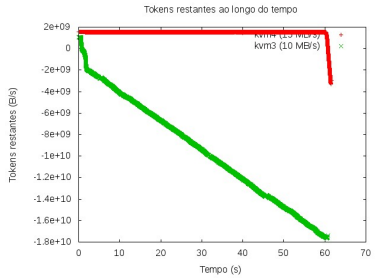


Figura 5.5: Largura de banda configurada para VMs sequenciais

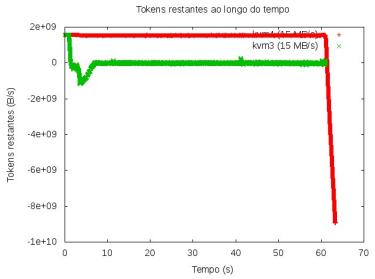
Se a largura de banda for configurada abaixo do limite mínimo, o escalonador vHTBS se comporta como o escalonador Deadline. Nessas condições, a carga de trabalho consome mais símbolos que a quantidade disponível, possuindo uma quantidade negativa de símbolos. Uma vez negativa a quantidade de símbolos, à VM não é permitido aguardar T_{wait} milissegundos por novas requisições da mesma fila nem enviar B_{max} requisições antes de trocar a aplicação ativa. Todo o escalonamento é baseado nas etiquetas de início e término das requisições, apresentando-se como prazos para as requisições terminarem.

Se a largura de banda for configurada acima do limite máximo, o escalonador vHTBS garante a vazão máxima disponível de acordo com a vazão das demais VMs concorrentes. Para kvm3 configurado com 45 MB/s e 60 MB/s, o vHTBS garante o valor mínimo de kvm4 e entrega o máximo de largura de banda disponível até o limite do disco e do valor estipulado para kvm3.

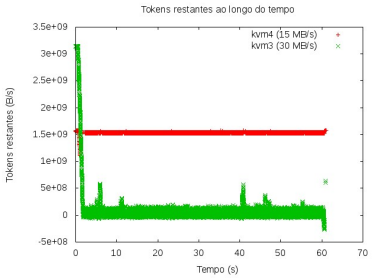
A Figura 5.6 apresenta a quantidade de símbolos restantes ao longo do tempo de execução para VMs mistas. A máquina virtual kvm3, marcada como sequencial internamente ao escalonador vHTBS, varia a largura de banda como no experimento anterior. A VM kvm4, marcada como aleatória internamente ao vHTBS, possui largura de banda fixa de 15 MB/s. O valor de T_{wait} e B_{max} é respectivamente 50 ms e 50 requisições.



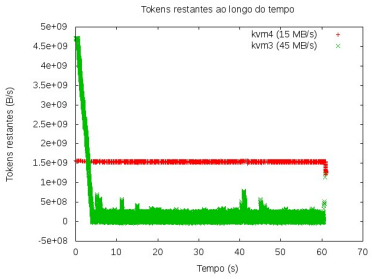
(a) kvm3 = 10 MB/s; kvm4 = 15 MB/s



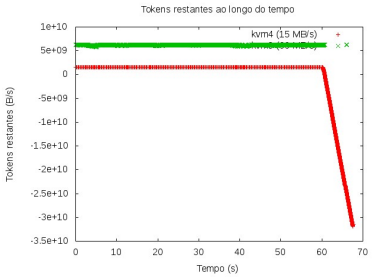
(b) kvm3 = 15 MB/s; kvm4 = 15 MB/s



(c) kvm3 = 30 MB/s; kvm4 = 15 MB/s



(d) kvm3 = 45 MB/s; kvm4 = 15 MB/s



(e) kvm3 = 60 MB/s; kvm4 = 15 MB/s

Figura 5.6: *Símbolos restantes ao longo do tempo para VMs mistas*

Para as larguras de banda abaixo (Figura 5.6.a) e acima (Figura 5.6.e) do intervalo configurável, a quantidade de símbolos restantes se assemelha ao cenário de VMs sequenciais. Abaixo do valor mínimo, a máquina virtual sequencial consome mais símbolos que a demanda; impedindo que a VM aguarde ociosamente por novas requisições e não en-

vie requisições consecutivamente. Acima do valor máximo, existem mais símbolos que a demanda; o excesso de símbolos garante a vazão de kvm3 e entrega a largura de banda restante a kvm4.

Dentro do intervalo de largura de banda configurável (Figuras 5.6.b, 5.6.c e 5.6.d), a VM sequencial apresenta quantidade de símbolos próxima de zero a maior parte do tempo de execução do *benchmark*. Próximo de zero, a VM não consome símbolos em excesso nem em falta; garantindo a vazão estipulada sempre que houver demanda de trabalho. Novamente para kvm4, a vazão observada depende da largura de banda restante após o uso pelas máquinas sequenciais.

As caudas presentes nos gráficos das Figuras 5.6.b e 5.6.e representam a execução isolada da VM kvm4. Desde o início do decréscimo da quantidade de símbolos até o fim da execução da carga de trabalho, kvm4 executa sozinha. Nesse intervalo de tempo, kvm3 já terminou a execução da carga de trabalho e 60 segundos de execução se passaram; kvm4 ainda executa pois a quantidade de 50 requisições para B_{max} causa inanição, atrasando a execução da carga de trabalho marcada como aleatória.

5.4.2 Latência do vHTBS

A Figura 5.7 apresenta a latência das máquinas virtuais em relação à latência configurada. As máquinas virtuais utilizadas possuem padrão de acesso aleatório. A máquina virtual kvm1 é configurada com 25 ms de latência e a VM kvm2 varia entre 10 ms, 25 ms e 100 ms de latência. Ambas máquinas virtuais são configuradas com largura de banda igual a 0,5 MB/s. Os parâmetros de configuração T_{wait} e B_{max} são irrelevantes, pois no acesso aleatório não há tempo de espera por novas requisições da mesma fila nem há o envio consecutivo de requisições sequenciais.

Independentemente do escalonador considerar o padrão de acesso das máquinas virtuais como aleatório ou sequencial, a carga de trabalho de acesso aleatório possui, com HTBS e vHTBS, o mesmo desempenho que com o escalonador Deadline. A falta de aproveitamento de requisições consecutivas faz com que o escalonamento seja realizado apenas com base nas etiquetas de início e fim das requisições.

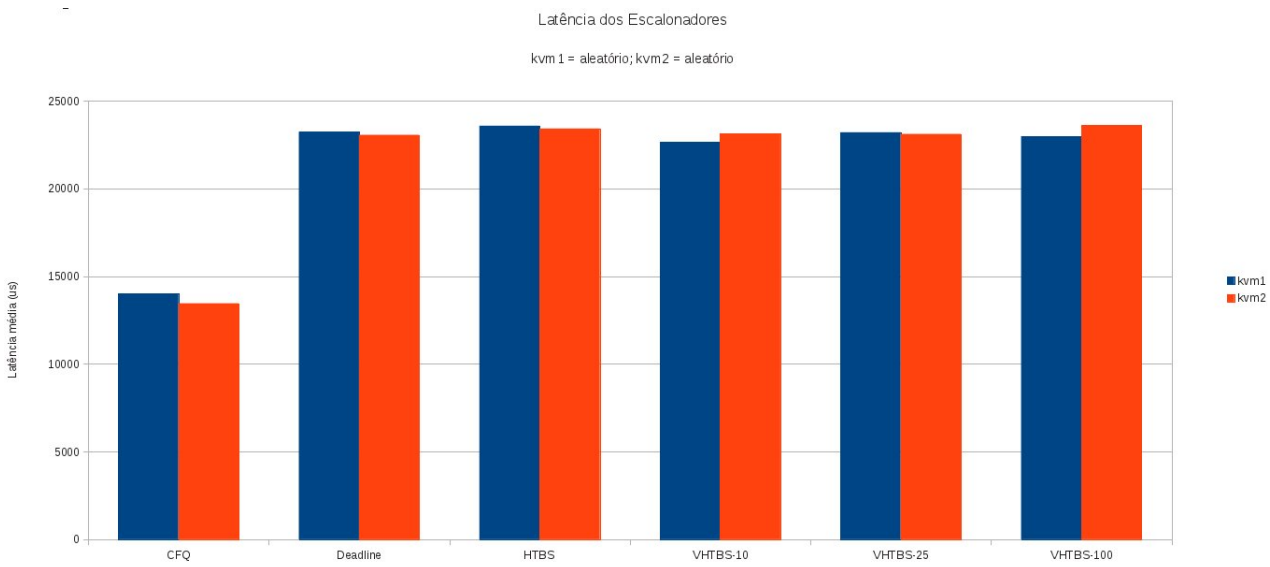


Figura 5.7: Latência para VMs aleatórias

A latência média das máquinas virtuais pode variar se houver: (i) uma máquina virtual marcada como sequencial e (ii) com valores de T_{wait} e B_{max} suficientemente grandes para esperar o mínimo de tempo até novas requisições da mesma fila surgirem e enviar até B_{max} requisições consecutivas. Até o momento, considerar as VMs como aleatórias, não garante valores de vazão nem de latência média.

Como trabalho futuro, está o estudo da influência do atributo de QoS latência dos escalonadores HTBS e vHTBS. Pelo observado, quando nenhuma máquina virtual é marcada como sequencial no escalonador, o vHTBS se comporta do mesmo modo que o escalonador Deadline. Não há a espera por novas requisições da mesma fila nem o envio consecutivo de requisições. Esse também é o comportamento quando as filas sequenciais consomem mais símbolos que os disponíveis.

5.5 Comparação entre Escalonadores de Disco

A Figura 5.8 apresenta a vazão das máquinas virtuais sequenciais com os escalonadores avaliados. As configurações dos escalonadores CFQ e Deadline são a padrão do sistema; o escalonador HTBS e vHTBS são configurados com 1920 IOPS e 30 MB/s de largura de banda para cada VM, respectivamente, T_{wait} igual a 30 ms e B_{max} igual a 30 requisições.

Nos escalonadores HTBS e vHTBS, ambas máquinas virtuais são consideradas sequenciais.

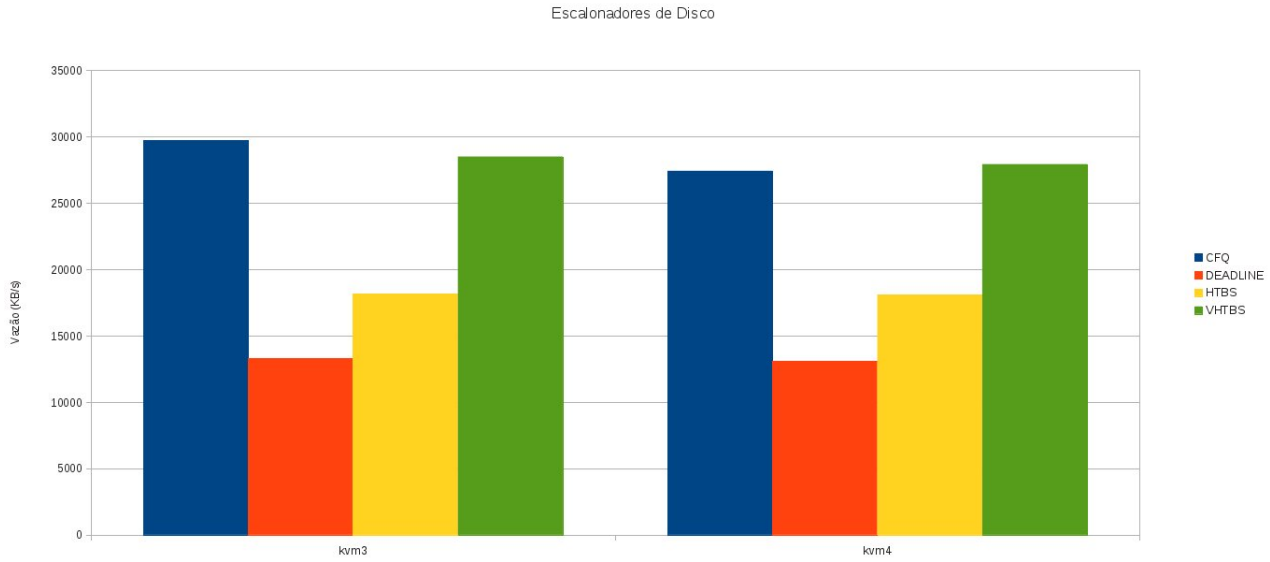


Figura 5.8: Vazão dos escalonadores para VMs sequenciais

O escalonador CFQ apresenta a maior vazão para ambas as máquinas virtuais e a maior vazão agregada. A característica do escalonador CFQ é dividir de forma justa o tempo de acesso aos dispositivos entre todas as aplicações que fazem uso do dispositivo. Como kvm3 e kvm4 possuem a mesma carga de trabalho sobre o mesmo dispositivo, a vazão apresentada por essas VMs é semelhante.

O escalonador Deadline possui a menor vazão para ambas as máquinas virtuais e menor vazão agregada. O Deadline não possui mecanismo de prevenção da ociosidade enganosa habilitado por padrão. Dependendo da configuração, o Deadline pode agir como o escalonador Anticipatory, porém a configuração original está limitada apenas ao prazo de criação e de término das requisições.

O escalonador HTBS apresenta uma vazão intermediária. Apesar de considerar mecanismos de QoS e permitir sua configuração através de parâmetros, o algoritmo original não considera que o padrão de acesso das VMs, mesmo sendo sequencial, possa incluir alguma requisição aleatória. Ainda que as VMs tenham seu padrão de acesso configurado no código-fonte (*hard-coded*), o algoritmo original não dimensiona adequadamente o tempo de espera por novas requisições (T_{wait}).

O escalonador vHTBS possui vazão próxima da vazão obtida com o escalonador CFQ.

Ao considerar o padrão de acesso das requisições sequenciais, o vHTBS permite que essas se acumulem e sejam enviadas consecutivamente antes de ceder a vez de execução a outra VM. Adicionalmente, o vHTBS considera o tempo de espera das requisições de forma adequada, ativando o *timer* de *kernel* após o envio de uma requisição sequencial e desativando o *timer* quando uma nova requisição surge.

Com um limite de requisições sequenciais consecutivas (B_{max}) maior, é possível obter vazão ligeiramente maior que o CFQ. Com B_{max} igual a 50 requisições, a vazão de kvm3 chega a aproximadamente 28,5 MB/s e de kvm4 a aproximadamente 30,5 MB/s. É necessário ponderar entre alta vazão e justiça no acesso das requisições pois, à medida que o valor de B_{max} aumenta, alguma fila de requisições pode sofrer inanição.

Extendendo os trabalhos futuros, está o desenvolvimento de mais testes de consolidação. Pretende-se considerar mais casos de testes com mais VMs em um cenário misto. Ademais, pretende-se considerar também a influência que processos comuns no sistema base (*background jobs*) possam ter sobre os atributos de QoS das máquinas virtuais.

CAPÍTULO 6

CONCLUSÃO

A virtualização de servidores permite que os sistemas computacionais utilizem diversas máquinas virtuais executando simultaneamente. Para gerenciar a execução simultânea, é necessário controlar os recursos e dividi-los entre os processos. A qualidade de serviço proporciona uma divisão justa dos recursos da máquina base entre as VMs existentes e os demais processos.

Esta dissertação apresenta um levantamento da área de QoS para máquinas virtuais, mais especificamente na QoS de discos virtuais. Primeiramente, revisa-se os trabalhos já desenvolvidos nessa área de atuação; posteriormente, descreve-se um método para garantir requisitos de QoS diferentes a máquinas virtuais diferentes.

Os trabalhos revisados mostram que a escolha de escalonadores para a máquina base e para as máquinas virtuais impactam no desempenho das VMs. Dentre esses, o *framework* Flubber reordena as requisições de máquinas virtuais em dois níveis: o primeiro, próximo às VMs, controla a vazão das máquinas virtuais; o segundo, mais próximo do disco físico, controla a latência das requisições das VMs.

Enumera-se como requisitos para considerar processos de máquinas virtuais no escalonamento de requisições de disco: agrupamento de tarefas, mecanismos de QoS e predição do padrão de acesso. O agrupamento de tarefas permite a configuração individual de parâmetros de QoS; os mecanismos de QoS, garantias de execução; a predição do padrão de acesso, informa como as requisições se comportam.

O método descrito nesta dissertação para garantir requisitos diferentes de QoS a VMs considera como base o escalonador HTBS; algoritmo escrito para processos de discos físicos que permite a configuração individual de parâmetros de QoS: vazão, latência e rajadas. O HTBS é adaptado para o ambiente virtual utilizando classes de escalonamento. Cada classe de escalonamento separa logicamente os processos de máquina virtual através de

cgroups.

Como forma de avaliação, testes com os *benchmarks sysbench*, *filebench* e *fio* são feitos. O *sysbench* e o *filebench* são utilizados nos testes iniciais. O *sysbench* é escolhido para gerar testes semelhantes aos testes do *framework* Flubber. O *filebench* é escolhido para simular cargas de trabalho em *background*. Dada a complexidade dos testes realizados com os *benchmarks*, opta-se pela utilização do *fio*.

O *fio* é utilizado para gerar cargas de trabalho sequenciais e aleatórias pela quantidade de tempo estipulada. Através do *fio* descobre-se a influência dos parâmetros de configuração T_{wait} e B_{max} do HTBS, mostra-se como o vHTBS pode garantir requisitos de QoS e compara-se o desempenho do vHTBS com os escalonadores padrão do Linux.

A influência do parâmetro de configuração T_{wait} está no tempo das requisições sequenciais surgirem. Se o valor de T_{wait} for pequeno, o escalonador pode trocar a fila ativa por outra, sem evitar a ociosidade enganosa. Se o tempo de espera for alto, as filas não ativas podem esperar desnecessariamente a fila ativa, como por exemplo quando a fila ativa termina suas requisições.

A influência do parâmetro de configuração B_{max} está na quantidade de requisições sequencias consecutivas enviada pela fila ativa. Quanto menor o valor de B_{max}, menor é a latência média entre as VMs e mais justo é o escalonamento. Quanto maior o número de requisições sequenciais consecutivas, maior é a vazão da VM, porém as demais filas podem sofrer inanição.

Para os casos de testes, o escalonador CFQ apresenta a maior vazão; o Deadline, a menor vazão e o HTBS uma vazão intermediária. O vHTBS apresenta vazão próxima do melhor caso, alcançado pelo CFQ. Dependendo dos parâmetros de configuração escolhidos, o vHTBS pode ligeiramente ultrapassar o desempenho do CFQ.

Existe um intervalo de valores para os quais o vHTBS pode garantir vazão. Nesse intervalo, a vazão de uma VM depende também do tipo de acesso da fila de requisições. Se ambas VMs forem sequenciais, o escalonador distribui de forma justa a largura de banda existente. Se alguma VM for aleatória, o escalonador prioriza o acesso sequencial, se houver, e divide a largura de banda restante entre as VMs aleatórias através das

etiquetas de prazos.

Para a latência, o escalonador vHTBS não cumpriu os requisitos de QoS estipulados. Como trabalho futuro, está a compreensão da influência da latência nas requisições dos escalonadores, a execução de cargas de trabalho mistas em cenários com mais VMs consolidadas e a influência de trabalhos concorrentes (*background jobs*) quando executados junto com as VMs.

APÊNDICE A

TESTES INICIAIS

Os testes iniciais consideram dois *benchmarks* diferentes: *sysbench* [46] e *filebench* [17]. O *sysbench* é utilizado para gerar a carga de trabalho de cada VM; o *filebench* é utilizado para gerar carga de trabalho simulando *background jobs* pois permite que a carga de trabalho continue a ser gerada pela quantidade de tempo especificada como parâmetro da aplicação. Ambos geram carga de trabalho sintéticas.

O *sysbench* é uma ferramenta de *benchmark multi-thread* para avaliação de parâmetros do sistema operacional, como vazão e latência. As *threads* das cargas de trabalho executam em paralelo e pode-se limitar o número total de requisições, o tempo total para o *benchmark* ou ambos.

Os modos de testes possíveis para o *sysbench* são: *cpu, threads, mutex, memory, fileio e oltp*. O modo *cpu* calcula números primos usando números inteiros de 64 bits; o modo *threads* mede a performance de escalonadores diante de *threads* competindo por uma certa quantidade de *mutexes*; o modo *mutex* examina a performance da implementação de *mutex*; o modo *memory* realiza leituras ou escritas sequenciais em memória; o modo *fileio* pode gerar diversas cargas de trabalho do tipo E/S; o modo *oltp* mede a performance de uma base de dados real. Para o escopo deste trabalho, o modo escolhido foi o *fileio*.

As cargas de trabalho para o modo *fileio* podem ser leituras, releituras, escritas ou re-escritas. Cada teste pode ter padrão de acesso síncrono, assíncrono ou mapeado em memória. Os valores padrão do número de arquivos, do tamanho de bloco, do tamanho total dos arquivos e o método de acesso são respectivamente 128, 16 KB, 2 GB e síncrono.

A configuração das cargas de trabalho dos testes realizados é baseada na configuração dos testes de estudo experimental de atribuição de créditos do Flubber: kvm1, VM de carga de trabalho com padrão de acesso aleatório, considera 8 *threads*, 128 arquivos e tamanho total de 1 GB; kvm3, VM de carga de trabalho com padrão de acesso sequencial,

considera 16 *threads*, 128 arquivos e tamanho total de 2 GB.

Adicionalmente, a carga de trabalho em kvm1 é configurada para executar mais operações que a configuração padrão do *sysbench*. A princípio, cargas de trabalho com padrão de acesso aleatório possuem um limite de 10.000 operações. Para que a carga de trabalho execute sobre o tamanho total do arquivo, esse limite é alterado. No caso de kvm1, o novo limite de operação é marcado como 65.536 operações.

Todos os testes consideram apenas leituras para as VMs. Ambas máquinas virtuais iniciam suas execuções no mesmo instante. Antes de cada execução, as VMs e a máquina base são sincronizadas por NTP [33] e as informações de *cache* dos sistemas são apagadas. Os resultados mostrados são a média de cinco execuções de cada carga de trabalho.

A.1 Padrão de Acesso das Requisições Virtuais

Com os testes iniciais utilizando o *benchmark sysbench*, é possível perceber a importância do padrão de acesso das requisições para cada carga de trabalho. O escalonador utilizado é o HTBS e as métricas apresentadas são a vazão da carga de trabalho, a latência média das requisições e a quantidade de requisições emitidas pelas cargas avaliadas.

As Tabelas A.1 e A.2 sintetizam todas as combinações de cargas de trabalho aleatória e sequencial consolidadas em ambientes virtuais distintos. Os cenários avaliados são: (i) ambas cargas consideradas aleatórias pelo escalonador (cenário *Rnd*); (ii) ambas cargas consideradas sequenciais pelo escalonador (cenário *Seq*); (iii) apenas kvm3 considerada sequencial (cenário *kvm3-seq*); (iv) apenas kvm1 considerada sequencial (cenário *kvm1-seq*). Para os cenários (iii) e (iv), o HTBS infere o padrão de acesso de kvm1 e kvm3, respectivamente.

A Tabela A.1 mostra o comportamento observado da VM kvm1. Os cenários que produzem as maiores vazão e menor latência são os cenários *Rnd* e *kvm3-seq*, que consideram corretamente o padrão de acesso da carga de trabalho aleatória. Quando considerada sequencial pelo HTBS, a vazão de kvm1 diminui e a latência média aumenta.

A Tabela A.2 mostra o comportamento observado da VM kvm3. Quanto mais requisições kvm3 emite, maior é vazão e menor é a latência média das requisições; igual-

Métrica	Rnd	Seq	kvm3-seq	kvm1-seq
Vazão (MB/s)	5,7643	5,5562	5,8623	5,5364
Latência (ms)	21,71	22,49	21,35	22,57
Requisições	34.537	34.499	34.424	34.533

Tabela A.1: Atributos de QoS para carga de trabalho aleatória

mente, quando o padrão de acesso de kvm1 é reconhecido corretamente pelo escalonador HTBS, kvm3 apresenta as maiores vazões e menores latências.

Métrica	Rnd	Seq	kvm3-seq	kvm1-seq
Vazão (MB/s)	42,716	15,034	45,608	15,094
Latência (ms)	6,28	16,63	5,91	16,57
Requisições	17.031	16.595	16.975	16.571

Tabela A.2: Atributos de QoS para carga de trabalho sequencial

A interferência de kvm1 nas métricas de kvm3 depende do padrão de acesso que o escalonador considere. O cenário que considera o padrão de acesso correto de cada carga de trabalho (cenário *kvm3-seq*) produz a combinação com vazão mais elevada e menor latência média para ambas cargas simultaneamente.

Considerar erroneamente um padrão de acesso aleatório como sequencial (cenário *Seq*) faz a carga de kvm1 interferir nas métricas de kvm3. Da mesma forma, se erroneamente considerar um padrão de acesso sequencial como aleatório e um padrão de acesso aleatório como sequencial (cenário *kvm1-seq*), compromete-se as métricas avaliadas.

A.1.1 Percentagem de Requisições Sequenciais

Por padrão, o escalonador HTBS considera todas as cargas de trabalho como sequenciais antes das cargas emitirem as primeiras requisições. Após a emissão de requisições, o HTBS infere o padrão de acesso de cada carga ou fila de requisições através do mecanismo de predição do padrão de acesso.

Sem modificações, o HTBS verifica se a requisição atual possui endereço lógico próximo da requisição anterior. Em caso afirmativo, a fila de requisições é mantida como sequencial até a próxima requisição surgir, quando novamente o padrão de acesso é avaliado.

Caso contrário, o padrão é marcado como aleatório e permanece nesse estado até que o escalonador seja reconfigurado.

As Tabelas A.3 e A.4 apresentam informações sobre a percentagem de requisições sequenciais para as cargas de trabalho aleatória e sequencial respectivamente sobre os cenários descritos anteriormente. As quatro primeiras linhas das tabelas correspondem a percentagem de requisições sequenciais após 500, 1.000, 1.500 e 2.000 requisições emitidas pelas filas de E/S. As duas últimas linhas das tabelas contêm a percentagem de requisições sequenciais sobre o total de requisições emitidas pelas cargas de trabalho e o desvio padrão sobre o total de requisições sequenciais.

A Tabela A.3 considera o padrão de acesso aleatório. Os cenários *Rnd* e *kvm3-seq* emitem menos requisições sequenciais, cenários responsáveis pelas maiores vazões e menores latência média. Os cenários *Seq* e *kvm1-seq* emitem mais requisições sequenciais, responsáveis pelas menores vazões e maiores latência média. Independentemente do cenário observado, as requisições sequenciais no padrão de acesso aleatório não ultrapassam 11% das requisições emitidas.

Métrica	**Rnd**	**Seq**	**kvm3-seq**	**kvm1-seq**
500	10,80%	04,73%	05,85%	05,80%
1000	05,40%	02,37%	02,93%	02,90%
1500	03,60%	01,58%	01,95%	01,93%
2000	02,70%	01,18%	01,46%	01,45%
Total	00,25%	00,07%	00,09%	00,12%
Desvio Padrão (total)	58,06	5,15	24,09	32,62

Tabela A.3: Percentagem de requisições sequenciais para carga de trabalho aleatória

A Tabela A.4 considera o padrão de acesso sequencial. Observa-se que os cenários *Rnd* e *kvm3-seq* emitem sempre acima de 85% de requisições sequenciais; os cenários *Seq* e *kvm1-seq* emitem sempre acima de 60% de requisições sequenciais. Ao final da execução da carga de trabalho, nenhum cenário emitiu menos que 95% de requisições sequenciais.

Os valores obtidos permitem a escolha de valores base para o algoritmo de predição do padrão de acesso do vHTBS ou para ilustrar a importância que o padrão de acesso das filas de E/S possuem no escalonamento de requisições. De forma geral, escolher

Métrica	Rnd	Seq	kvm3-seq	kvm1-seq
500	87,36%	60,80%	85,05%	62,00%
1.000	92,42%	78,60%	91,90%	79,23%
1.500	94,55%	84,29%	94,27%	84,22%
2.000	95,67%	87,70%	95,45%	87,68%
Total	97,86%	96,41%	97,96%	96,36%
Desvio Padrão (total)	75,11	106,02	30,42	71,15

Tabela A.4: Percentagem de requisições sequenciais para carga de trabalho sequencial

erroneamente o padrão de acesso de alguma fila pode interferir nas métricas de alguma carga de trabalho.

Para o mecanismo de predição, inicialmente todas as cargas de trabalho do vHTBS são marcadas como sequenciais. A cada 500 requisições, o padrão de acesso da fila de E/S é verificado e marcado temporariamente como sequencial ou aleatório de acordo com um limiar escolhido. Opta-se por no mínimo 50% de requisições sequenciais dentre as requisições emitidas a cada 500 porções de requisições emitidas para inferir uma carga de trabalho como sequencial. Define-se como limite para escolha definitiva do padrão de acesso considerado pelo escalonador o valor de 2.000 requisições.

A.2 Testes Iniciais Realizados com *Sysbench*

Os demais testes realizados com a ferramenta *sysbench* são classificados em quatro classes diferentes: testes iniciais, testes *mono-thread*, testes *multi-thread* e testes com *background jobs*. Os testes iniciais são feitos para inferir valores base de vazão para as máquinas virtuais e suas cargas de trabalho. Os demais testes são realizados para demonstrar o comportamento do agrupamento de tarefas e da predição de padrão de acesso das filas de E/S para disponibilizar atributos de QoS às VMs.

A.2.1 Testes Iniciais no Sistema Base

Para obter valores base de vazão para as VMs, inicialmente executou-se no ambiente base as cargas de trabalho que são executadas dentro das VMs. Alocou-se os arquivos de testes do *benchmark* na partição que contém as imagens de discos virtuais KVM. Essa decisão faz

com que as leituras dos arquivos desse teste estejam sob os mesmos fatores que os discos virtuais (por exemplo, *seek time* e latência de rotação), pois estão alocados no mesmo disco físico e mesma partição dos discos virtuais. Todos os testes iniciais consideram o escalonador padrão do Linux.

As Tabelas A.5 e A.6 apresentam os valores da consolidação na máquina base para cada carga de trabalho. A métrica "Tempo" mede por quanto tempo em segundos a carga de trabalho executou. As métricas "Min", "Med", "Max" e "Perc.95" medem respectivamente em milissegundos o menor tempo de resposta, o tempo de resposta médio, o maior tempo de resposta e o maior tempo de resposta após descartar 5% das requisições de E/S mais longas emitidas pela carga de trabalho. A métrica "Vazão" mede a vazão obtida pela carga de trabalho.

A Tabela A.5 mostra os valores da carga de trabalho aleatória na consolidação. A vazão da carga de trabalho aleatória na presença de *background jobs* é 46,51% menor em comparação à vazão dos testes de somente consolidação para o ambiente *mono-thread*. Para o ambiente *multi-thread*, a vazão da carga de trabalho aleatória na presença de *background jobs* é 5,77% maior que a vazão dos testes de somente consolidação.

Métrica	***monothread***	***multi-thread***	***bg jobs - mono***	***bg jobs - multi***
Tempo (s)	253,3399	199,0363	473,6283	188,1287
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	3,86	24,30	7,23	22,96
Max (ms)	227,12	2.652,24	4.980,97	2.470,03
Perc.95 (ms)	12,68	109,96	19,60	107,52
Vazão (MB/s)	4,0413	5,1455	2,1620	5,4425

Tabela A.5: Carga de trabalho aleatória consolidada na maquina base

A Tabela A.6 mostra os valores da carga de trabalho sequencial na consolidação. A vazão da carga de trabalho sequencial na presença de *background jobs* é 15,35% menor que a vazão dos testes de somente consolidação para o ambiente *mono-thread*. Para o ambiente *multi-thread*, a vazão da carga de trabalho sequencial na presença de *background jobs* é 3,49% menor que a vazão dos testes de somente consolidação.

Os valores base escolhidos de vazão para os testes *mono-thread*, *multi-thread* e testes

Métrica	*monothread*	*multi-thread*	*bg jobs - mono*	*bg jobs - multi*
Tempo (s)	57,1027	84,7442	67,9387	88,8101
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	0,44	10,35	0,52	10,84
Max (ms)	192,21	1.463,61	934,94	1.915,76
Perc.95 (ms)	1,38	4,40	1,38	4,40
Vazão (MB/s)	35,847	24,280	30,347	23,435

Tabela A.6: Carga de trabalho sequencial consolidada na maquina base

com *background jobs* são de 4 MB/s para a carga de trabalho aleatória e de 20 MB/s para a carga de trabalho sequencial. Esses valores representam a vazão média aproximada das cargas de trabalho quando executam em ambiente virtual *mono-thread* com HTBS. Os valores base escolhidos de latência para os testes são de 20 ms e de 10 ms para as cargas aleatória e sequencial, respectivamente. Esse valores são baseados nas latências apresentadas nos testes iniciais.

A.2.2 Testes *Mono-thread*

A Tabela A.7 mostra os valores obtidos para kvm1 segundo os três escalonadores de disco avaliados. Comparado aos valores de execução do *benchmark* na máquina base, a vazão dos escalonadores CFQ e HTBS é, respectivamente, 7,13% e 1,69% menor que a vazão base; a vazão obtida com o vHTBS é 9,77% maior que a vazão base. Os valores de latência média (tempo de resposta médio) são 8,03% e 1,81% maiores que o valor base para os escalonadores CFQ e HTBS respectivamente; o vHTBS possui um valor 8,81% menor que o valor base.

Métrica	Base (CFQ)	CFQ	HTBS	vHTBS
Tempo (s)	253,3399	273,4670	257,9111	230,7829
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	3,86	4,17	3,93	3,52
Max (ms)	227,12	148,57	674,01	680,76
Perc.95 (ms)	12,68	21,37	19,12	17,96
Vazão (MB/s)	4,0413	3,7534	3,9734	4,4362

Tabela A.7: Resultados de kvm1 no ambiente *mono-thread*

A Tabela A.8 mostra os resultados obtidos para kvm3 segundo os três escalonadores

avaliados. A vazão dos escalonadores CFQ, HTBS e vHTBS é, respectivamente, 62,38%, 40,33% e 34,02% menor que o valor base de vazão. A latência média dos escalonadores CFQ, HTBS e vHTBS é 163,63%, 65,90% e 50,00% maior que a latência média base.

Métrica	**Base (CFQ)**	**CFQ**	**HTBS**	**vHTBS**
Tempo (s)	57,1027	151,7791	95,6518	86,6180
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	0,44	1,16	0,73	0,66
Max (ms)	192,21	175,19	148,25	141,25
Perc.95 (ms)	1,38	2,04	1,94	1,40
Vazão (MB/s)	35,847	13,489	21,393	23,652

Tabela A.8: Resultados de kvm3 no ambiente *mono-thread*

No ambiente *mono-thread*, os escalonadores avaliados não atendem completamente os requisitos de QoS estipulados. Todos os escalonadores atendem o requisito latência, mas nem todos atendem o requisito vazão. Para kvm1, os escalonadores CFQ e HTBS se aproximam do valor estipulado, enquanto o vHTBS supera o valor base em 10%. Para kvm3, os escalonadores avaliados se aproximam abaixo de 70% do valor estipulado.

A.2.3 Testes *Multi-thread*

A Tabela A.9 mostra os valores obtidos para kvm1 segundo os três escalonadores de disco avaliados. A vazão dos escalonadores CFQ, HTBS e vHTBS é, respectivamente, 1,86%, 26,75% e 27,73% maior que a vazão base. A latência média de CFQ, HTBS e vHTBS é 1,94%, 21,16% e 21,77latência média base, respectivamente.

Métrica	**Base (CFQ)**	**CFQ**	**HTBS**	**vHTBS**
Tempo (s)	199,0363	195,2190	156,9368	155,7081
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	24,30	23,83	19,16	19,01
Max (ms)	2.652,24	2.210,54	6.694,55	5903,91
Perc.95 (ms)	109,96	120,52	92,40	91,74
Vazão (MB/s)	5,1455	5,2414	6,5221	6,5727

Tabela A.9: Resultados de kvm1 no ambiente *multi-thread*

A Tabela A.10 mostra os resultados obtidos para kvm3 segundo os três escalonadores avaliados. O escalonador CFQ apresenta vazão 59,04% menor que a vazão base; os esca-

lonadores HTBS e vHTBS apresentam vazão 98,76% e 112,95% maior que a vazão base, respectivamente. A latência média do CFQ é 137% maior que a latência média base; os escalonadores HTBS e vHTBS possuem latência média 49,18% e 52,95% menor que a latência média base, respectivamente.

Métrica	**Base (CFQ)**	**CFQ**	**HTBS**	**vHTBS**
Tempo (s)	84,7442	201,0017	43,1035	39,8992
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	10,35	24,53	5,26	4,87
Max (ms)	1.463,61	548,48	934,25	1.822,03
Perc.95 (ms)	4,40	90,46	4,69	4,41
Vazão (MB/s)	24,280	10,190	48,261	51,705

Tabela A.10: Resultados de kvm3 no ambiente *multi-thread*

No ambiente *multi-thread*, o escalonador CFQ não atende os requisitos de QoS. Para kvm1, a latência é maior que o valor estabelecido, apesar de ter maior vazão e menor latência que os valores base. Para kvm3, tanto a latência quanto a vazão estão fora dos valores estabelecidos. Os escalonadores HTBS e vHTBS apresentam valores no mínimo 25% maior que os valores base no requisito vazão e no mínimo 20% menos latência média que os valores base.

A.2.4 Testes com *Background Jobs* em Ambiente *Mono-thread*

A Tabela A.11 mostra os valores obtidos para kvm1 segundo os três escalonadores de disco avaliados. A vazão dos escalonadores CFQ, HTBS e vHTBS é, respectivamente, 23,53%, 23,20% e 20,85% menor que a vazão base. A latência média de CFQ, HTBS e vHTBS é 30,70%, 30,15% e 26,27% maior que a latência média base, respectivamente.

A Tabela A.12 mostra os resultados obtidos para kvm3 segundo os três escalonadores avaliados. Os escalonadores CFQ, HTBS e vHTBS apresentam, respectivamente, 85,73%, 84,44% e 84,36% menos vazão que o valor base. A latência média dos escalonadores CFQ, HTBS e vHTBS é 594,23%, 536,53% e 532,69% maior que a latência média base, respectivamente.

No ambiente *mono-thread* com *background jobs*, os escalonadores CFQ, HTBS e vHTBS

Métrica	Base (CFQ)	CFQ	HTBS	vHTBS
Tempo (s)	473,6283	619,2864	616,6421	598,4714
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	7,23	9,45	9,41	9,13
Max (ms)	4.980,97	10.329,71	10.168,64	78.480,84
Perc.95 (ms)	19,60	21,59	19,17	19,01
Vazão (MB/s)	2,1620	1,6534	1,6606	1,7114

Tabela A.11: Resultados de kvm1 no ambiente *mono-thread* com *background jobs*

Métrica	Base (CFQ)	CFQ	HTBS	vHTBS
Tempo (s)	67,9387	472,9544	433,7474	431,6507
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	0,52	3,61	3,31	3,29
Max (ms)	934,94	4.309,76	6.943,07	66.119,58
Perc.95 (ms)	1,38	1,37	1,43	1,41
Vazão (MB/s)	30,347	4,331	4,725	4,747

Tabela A.12: Resultados de kvm3 no ambiente *mono-thread* com *background jobs*

não atende os requisitos de QoS para kvm1. A latência em kvm1 está dentro dos valores estabelecidos, porém a vazão está abaixo tanto do valor base quanto do valor estabelecido. Para kvm3, os valores obtidos estão dentro dos valores estabelecidos, porém a vazão está abaixo do valor base e a latência média está acima do valor base.

A.2.5 Testes com *Background Jobs* em Ambiente *Multi-thread*

A Tabela A.13 mostra os valores obtidos para kvm1 segundo os três escalonadores de disco avaliados. A vazão do escalonador CFQ é 64,38% menor que o valor base; HTBS e vHTBS apresentam, respectivamente, 14,22% e 13,38% menos vazão que os valores base. A latência média de CFQ é 180,70% maior que a latência média base; HTBS e vHTBS possuem latência média 12,50% e 11,91% menor que a latência média base, respectivamente.

A Tabela A.14 mostra os resultados obtidos para kvm3 segundo os três escalonadores avaliados. O escalonador CFQ apresenta vazão 83,62% menor que a vazão base; HTBS e vHTBS apresentam, respectivamente, 83,74% e 79,22% mais vazão que o valor base. A latência média do escalonador CFQ é 500,64% maior que a latência média base; HTBS e

Métrica	Base (CFQ)	CFQ	HTBS	vHTBS
Tempo (s)	188,1287	527,9503	164,5876	165,8938
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	22,96	64,45	20,09	20,25
Max (ms)	2.470,03	7.197,72	4.287,28	4.573,82
Perc.95 (ms)	107,52	254,84	98,09	98,95
Vazão (MB/s)	5,4425	1,9389	6,2165	6,1708

Tabela A.13: Resultados de kvm1 no ambiente *multi-thread* com *background jobs*

vHTBS têm 45,95% e 44,84% mais latência que o valor base, respectivamente.

Métrica	Base (CFQ)	CFQ	HTBS	vHTBS
Tempo (s)	88,8101	533,3551	47,9563	48,9657
Min (ms)	0,00	0,00	0,00	0,00
Med (ms)	10,84	65,11	5,86	5,98
Max (ms)	1.915,76	3.487,15	1.259,66	901,29
Perc.95 (ms)	4,40	236,83	4,45	5,22
Vazão (MB/s)	23,435	3,840	43,061	42,002

Tabela A.14: Resultados de kvm3 no ambiente *multi-thread* com *background jobs*

No ambiente *multi-thread* com *background jobs*, o escalonador CFQ não atende os requisitos de QoS. Tanto para kvm1 quanto para kvm3, o CFQ atinge vazão no mínimo 65% menor e latência média no mínimo 180% maior que os valores base, aproximadamente. Os escalonadores HTBS e vHTBS não atendem apenas a latência média estipulada para kvm1, porém se aproxima dos valores estipulados de QoS.

Os resultados apresentados pelos testes iniciais dificultam a compreensão dos efeitos de se variar a largura de banda e a latência atribuída às máquinas virtuais. Essas combinações e medições não permitem chegar a uma conclusão segura dos fatos acerca dos escalonadores padrão do Linux, do HTBS e das extensões sugeridas ao HTBS.

Todos os testes foram feitos com a emulação Posix de E/S assíncrono (POSIX AIO). Com a POSIX AIO, um *pool* de *threads* realiza o trabalho de requisições de E/S, podendo emular as requisições. Realizar a emulação pode adicionar atrasos no atendimento das requisições, comprometendo as medições [15]. Assim, para simplificar o ambiente de testes, optou-se pela troca da ferramenta de *benchmark* e do mecanismo de E/S assíncrono para Linux AIO.

BIBLIOGRAFIA

[1] M. Andrews, M.A. Bender, e L. Zhang. New algorithms for the disk scheduling problem. *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, páginas 550 –559, outubro de 1996.

[2] J. Axboe. Fio - flexible i/o tester. `http://freecode.com/projects/fio`, Acesso em 22 de julho de 2013.

[3] Jens Axboe. Linux block io - present and future. *Proceeding of the Ottawa Linux Symposium*, páginas 51–61, 2004.

[4] Jens Axboe. block: remove the anticipatory io scheduler. *Árvore git do kernel Linux*, julho de `http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=492af6350a5ccf087e4964104a276ed358811458` . Acesso em 17 de dezembro de 2013.

[5] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, e Andrew Warfield. Xen and the art of virtualization. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, páginas 164–177, New York, NY, USA, 2003. ACM.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, e W. Weiss. An architecture for differentiated services. Relatório técnico, IETF, Network Working Group, RFC 2475, 1998.

[7] David Boutcher e Abhishek Chandra. Does virtualization make disk scheduling passé? *SIGOPS Oper. Syst. Rev.*, 44(1):20–24, março de 2010.

[8] R. Braden, D. Clark, e S. Shenker. Integrated services in the internet architecture: an overview. Relatório técnico, IETF, Network Working Group, RFC 1633, 1994.

[9] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, e A. Silberschatz. Disk scheduling with quality of service guarantees. *Multimedia Computing and Systems, 1999. IEEE International Conference on*, volume 2, páginas 400 –405 vol.2, julho de 1999.

[10] Lawrence C. Miller, CISSP. *Server Virtualization for Dummies, Oracle Special Edition.* John Wiley & Sons, Inc., 1 edition, 2012.

[11] D. Chalmers e M. Sloman. A survey of quality of service in mobile computing environments. *Communications Surveys Tutorials, IEEE*, 2(2):2–10, 1999.

[12] Jianhua Che, Yong Yu, Congcong Shi, e Weimin Lin. A synthetical performance evaluation of openvz, xen and kvm. *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, páginas 587–594, 2010.

[13] Intel Corporation e Seagate Technology. Native command queuing, an exciting new performance feature for serial ata. *Online: http://www.seagate.com/docs/pdf/whitepaper/D2c_tech_paper_intc-stx_sata_ncq.pdf*, Acesso em 16 de outubro de 2013.

[14] Western Digital. Tagged command queuing (tcq). *Online: http://www.wdc.com/en/library/sata/2579-001076.pdf*, Acesso em 17 de dezembro de 2013.

[15] Página do Linux KVM. Virtio/block/latency. `http://www.linux-kvm.org/page/Virtio/Block/Latency#posix-aio-compat_versus_linux-aio`, Acesso em 10 de janeiro de 2014.

[16] João Carlos Carvalho dos Santos Ramos. Security challenges with virtualization. Dissertação de mestrado, Universidade de Lisboa, dezembro de 2009.

[17] Filebench. `http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Main_Page`, Acesso em 18 de fevereiro de 2013.

[18] Charles David Graziano. A performance analysis of xen and kvm hypervisors for hosting the xen worlds project. Dissertação de mestrado, Iowa State University, 2011.

[19] Ajay Gulati, Arif Merchant, e Peter J. Varman. pclock: an arrival curve based approach for qos guarantees in shared storage systems. *SIGMETRICS Perform. Eval. Rev.*, 35(1):13–24, junho de 2007.

[20] Red Hat Inc. Kvm - kernel based virtual machine. Relatório técnico, Red Hat Inc., 2009.

[21] Sitaram Iyer. The effect of deceptive idleness on disk schedulers. Dissertação de mestrado, Rice University, 2003.

[22] Sitaram Iyer e Peter Druschel. Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous i/o. *SIGOPS Oper. Syst. Rev.*, 35(5):117–130, outubro de 2001.

[23] Mukil Kesavan, Ada Gavrilovska, e Karsten Schwan. On disk i/o scheduling in virtual machines. *Proceedings of the 2nd conference on I/O virtualization*, WIOV'10, páginas 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[24] Hwanju Kim, Hyeontaek Lim, Jinkyu Jeong, Heeseung Jo, e Joonwon Lee. Task-aware virtual machine scheduling for i/o performance. *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, páginas 101–110, New York, NY, USA, 2009. ACM.

[25] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, e Anthony Liguori. kvm: the linux virtual machine monitor. *Proceedings of the Linux Symposium*, páginas 223–230, junho de 2007.

[26] Kyriakos Kritikos, Barbara Pernici, Pierluigi Plebani, Cinzia Cappiello, Marco Comuzzi, Salima Benrernou, Ivona Brandic, Attila Kertész, Michael Parkin, e Manuel Carro. A survey on service quality description. *ACM Comput. Surv.*, 46(1):1:1–1:58, julho de 2013.

[27] Jeffrey B. Layton. 2.6.33 is out! say good bye to the anticipatory scheduler. *Linux Magazine*, `http://www.linux-mag.com/id/7724`. Acesso em 17 de dezembro de 2013.

[28] Xiao Ling, Hai Jin, S. Ibrahim, Wenzhi Cao, e Song Wu. Efficient disk i/o scheduling with qos guarantee for xen-based hosting platforms. *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, páginas 81 –89, maio de 2012.

[29] M. Mahjoub, A. Mdhaffar, R.B. Halima, e M. Jmaiel. A comparative study of the current cloud computing technologies and offers. *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, páginas 131–134, 2011.

[30] Carlos Alberto Maziero. *Sistemas Operacionais: Conceitos e Macanismos*. Online: `http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-cap09`, 2013. Acesso em 4 de junho de 2013.

[31] Paul B. Menage. Adding generic process containers to the linux kernel. *Proceedings of the Linux Symposium*, 2:45–57, 2007.

[32] Aravind Menon, Alan L. Cox, e Willy Zwaenepoel. Optimizing network virtualization in xen. *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, páginas 2–2, Berkeley, CA, USA, 2006. USENIX Association.

[33] D.L. Mills. Network time protocol (ntp). Relatório técnico, IETF, Network Working Group, RFC 958, 1985.

[34] Diego Ongaro, Alan L. Cox, e Scott Rixner. Scheduling i/o in virtual machine monitors. *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, páginas 1–10, New York, NY, USA, 2008. ACM.

[35] Pedro Eugênio Rocha Pedreira. Sistema de armazenamento compartilhado com qualidade de serviços e alto-desempenho. Dissertação de mestrado, Universidade Federal do Paraná (UFPR), 2011.

[36] Gerald J. Popek e Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, julho de 1974.

[37] Matthew Portnoy. *Virtualization Essentials*. Wiley, 2012.

[38] Steven L. Pratt e Dominique A. Heger. Workload dependent performance evaluation of the linux 2.6 i/o schedulers. *Proceeding of the Ottawa Linux Symposium*, páginas 425–448, 2004.

[39] QEMU. `http://wiki.qemu.org/Main\_Page`, Acesso em 15 de maio de 2013.

[40] Ed. R. Braden, L. Zhang, S. Berson, S. Herzog, e S. Jamin. Resource reservation protocol (rsvp). Relatório técnico, IETF, Network Working Group, RFC 2205, 1997.

[41] Andrea Righi. cgroup: block device i/o controller(v8). *http://thread.gmane.org/gmane.linux.kernel.containers/5975*, Acesso em 5 de fevereiro de 2013.

[42] P.E. Rocha e L.C.E. Bona. A qos aware non-work-conserving disk scheduler. *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, páginas 1 –5, abril de 2012.

[43] E. Rosen, A. Viswanathan, e R. Callon. Multiprotocol label switching architecture. Relatório técnico, IETF, Network Working Group, RFC 3031, 2001.

[44] Amit Shah. Deep virtue: Kernel-based virtualization with kvm. *Linux Magazine, http://www.linux-magazine.com/Issues/2008/86/KVM*, páginas 37–39, janeiro de 2008. Acesso em 15 de maio de 2013.

[45] J.E. Smith e R. Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[46] Sysbench. `http://sysbench.sourceforge.net/docs`, Acesso em 21 de fevereiro de 2013.

[47] Andrew S. Tanenbaum. *Modern Operating Systems*. Pearson Prentice Hall, 3 edition, 2008.

[48] Ding Tao, Hao Qin-fen, Zhang Bing, Zhang Tie-gang, e Huai Li-ting. Scheduling policy optimization in kernel-based virtual machine. *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, páginas 1 –4, dezembro de 2010.

[49] P. Valente e F. Checconi. High throughput disk scheduling with fair bandwidth distribution. *Computers, IEEE Transactions on*, 59(9):1172 –1186, setembro de 2010.

[50] Leendert van Doorn. Hardware virtualization trends. *Proceedings of the 2nd international conference on Virtual execution environments*, VEE '06, páginas 45–45, New York, NY, USA, 2006. ACM.

[51] Srinivas Raju Vegesna. *IP Quality of Service*. Cisco Press, 2001.

[52] Xiaolin Wang, Jiarui Zang, Zhenlin Wang, Yingwei Luo, e Xiaoming Li. Selective hardware/software memory virtualization. *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, páginas 217–226, New York, NY, USA, 2011. ACM.

[53] Chris Wolf e Erick M. Halter. *Virtualization: From the Desktop to the Enterprise*. Apress, 2005.

[54] Binbin Zhang, Xiaolin Wang, Rongfeng Lai, Liang Yang, Yingwei Luo, Xiaoming Li, e Zhenlin Wang. A survey on i/o virtualization and optimization. *Proceedings of the The Fifth Annual ChinaGrid Conference*, CHINAGRID '10, páginas 117–123, Washington, DC, USA, 2010. IEEE Computer Society.

JOSINEY DE SOUZA

AVALIAÇÃO DE ALGORITMOS DE ESCALONAMENTO DE DISCO COM QUALIDADE DE SERVIÇO EM AMBIENTES VIRTUALIZADOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luis Carlos Erpen de Bona

CURITIBA

2014