PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 38/08

Estudo de Mecanismos para a Reserva de Processamento em Sistemas Computacionais

Valéria Quadros dos Reis
Renato Fontoura de Gusmão Cerqueira

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 38/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena September, 2008

Estudo de Mecanismos para a Reserva de Processamento em Sistemas Computacionais[1]

Valéria Quadros dos Reis e Renato Fontoura de Gusmão Cerqueira

{vreis, rcerq}@inf.puc-rio.br

Abstract. This paper presents the description of four projects which propose processing reservation in computational systems. The document exposes different ways of implementation which have their architectures analyzed under several aspects such as modularity, portability, scheduling flexibility and adaptation. Finally, in the conclusion section, a brief comparative study is presented, focusing on the main characteristics that must be found in computational systems which provide processing reservation.

Keywords: Resource reservation, middleware, quality-of-service, scheduling

Resumo. Este trabalho apresenta a descrição de quatro projetos com propostas de reserva de processamento em sistemas computacionais. Diferentes formas de implementação são expostas e têm suas arquiteturas analisadas sob diversos aspectos tais como modularidade, portabilidade, flexibilidade de escalonamento e adaptação. Para finalizar, no seção de conclusão, é realizado um breve estudo comparativo dos projetos descritos focando nas principais características que devem estar presentes na implementação de sistemas com reserva de processamento.

Palavras-chave: Reserva de processamento, middleware, qualidade de serviço, escalonamento

[1]Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil. Processo CNPq número 142035/2006-8.

Responsável por publicações:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introdução

Mídias de áudio e vídeo necessitam de processamento periódico constante para que cada quadro de imagem, ou pacote de voz, chegue em um intervalo de tempo determinado de forma que não ocorram retardos ou rajadas de dados que prejudiquem a reprodução das mesmas. Por esse motivo, aplicações multimídia são ditas de tempo real não-críticas. A crescente exibição de mídias em computadores fez com que surgisse uma intensa área de pesquisa em Qualidade de Serviço (*Quality of Service* - QoS) para aplicações multimídia. Nessa área, busca-se garantir que a aplicação do usuário seja executada com uma qualidade mínima esperada sendo, para isso, necessárias a alocação de diversos recursos computacionais tais como rede, CPU e memória. O processamento, em especial, consiste em um item central de estudo de QoS em aplicações multimídia, pois ele é responsável pela codificação e decodificação dos objetos de mídia, atividades fundamentais para a reprodução de vídeos e sons.

Paralelamente à popularização da exibição de mídias em computadores, Grades Computacionais têm surgido como uma forma de compartilhar recursos geograficamente distribuídos. O compartilhamento, porém, se aplicado de maneira não controlada, propicia um comportamento indesejado em sistemas cooperativos: o mau uso dos recursos computacionais. Nessa situação, assim como a descrita anteriormente para a execução de mídias, é desejável a implantação de um mecanismo de reserva de processamento às aplicações. Tal procedimento em uma grade computacional impediria a sobrecarga dos nós da grade, garantindo aos doadores de recursos o direito de estabelecer o limite de capacidade de processamento que desejam disponibilizar ao sistema, assim como garantiria ao cliente da infra-estrutura de grade uma qualidade mínima de processamento.

Este trabalho, tem por objetivo o estudo da reserva de processamento em sistemas computacionais através de diferentes abordagens e implementações. Sistemas Operacionais de Propósito Geral (*General Purpose Operating System - GPOS*) não tratam classes de aplicações de maneira específica e, dessa maneira, gerenciam de forma ineficiente aplicações de tempo real. Casos em que, por exemplo, o escalonamento é feito por um algoritmo de compartilhamento de tempo ou espaço não distinguem quais aplicações são mais importantes que as demais além de serem bastante conservativos, já que fatias de tempo ou espaço alocadas, mesmo em situações em que não se encontram sendo utilizadas, não podem ser cedidas a outros processos. Por outro lado, em sistemas com escalonamento de processos baseados em prioridade, pode haver uma espera demasiada por parte de aplicações classificadas com baixa prioridade.

Algumas iniciativas, tal como os ditos *Resource Kernels* ou *Resource Containers*, visam o desenvolvimento de mecanismos para a reserva de processamento através do desenvolvimento de extensões de *kernel* com o objetivo de prover acesso garantido e no tempo esperado aos recursos de um Sistema Operacional[1, 2, 3, 4]. Outras realizam a reserva de recursos através do desenvolvimento de aplicações que, apesar de executarem no nível de usuário, invocam chamadas ao sistema capazes de alterar a forma com que os processos são escalonados [5, 6].

Resource Kernels e aplicações que alteram o escalonamento dos processos são medidas possíveis de serem tomadas na implantação de um mecanismo de reserva de processamento em Grades Computacionais. Observa-se, porém, o surgimento de uma tendência ao uso de máquinas virtuais para essa finalidade. Máquinas virtuais disponibilizam ao usuário um ambiente personalizado, onde os recursos computacionais são dedicados somente aos

processos do usuário corrente. O isolamento de ambiente resulta em maior controle do uso dos recursos além de aumentar o nível de segurança, pois um processo de uma determinada máquina virtual é incapaz de acessar aos dados de outra máquina virtual[7, 8, 9, 10].

Este documento está dividido em quatro seções. A Seção 2 apresenta a definição de *Resource Kernels*, como os mesmos são implementados e exemplos desses sistemas. A Seção 3 apresenta um sistema executado em nível de usuário que altera a forma como o Sistema Operacional escalona os processos. Na Seção 4, o conceito de máquina virtual é definido e tem o seu funcionamento apresentado. Ainda nessa seção, são descritas as iniciativas de reserva de recursos no projeto Xen. Por fim, a Seção 5 apresenta uma breve comparação entre os diferentes mecanismos de reserva de processamento apresentados assim como as conclusões extraídas deste estudo.

2 Resource Kernels

Um *resource kernel* é um *kernel* modificado para gerenciar recursos através de modelos de reserva. Com ele é possível garantir o acesso de uma aplicação a uma determinada parte dos recursos em qualquer momento de sua execução através de funcionalidades providas pelo núcleo do SO [1]. RT-Mach e Linux/RK, juntamente com o *Portable RK*, consistem nos dois principais *Resource kernels* encontrados na literatura. O RED-Linux, por sua vez, apresenta relevante importância ao convergir em um único *framework* três paradigmas de escalonamento.

2.1 RT-Mach

Real-Time Mach (RT-Mach) é um Sistema Operacional de tempo real desenvolvido na Universidade de Carnegie Mellon e criado com base no *microkernel* Mach[2]. Nele existem mecanismos de reserva de processamento que permitem que aplicações especifiquem suas necessidades de CPU. O *kernel* RT-Mach provê muitas primitivas para a construção de aplicações de tempo real, incluindo primitivas para a escolha de diferentes políticas de escalonamento e para a definição de periodicidade das aplicações [3].

Para garantir uma reserva, uma aplicação deve definir o seu uso de CPU a cada intervalo de tempo como, por exemplo, 10 *ms* a cada 50 *ms*. Aplicações de tempo real com reservas e aplicações sem a necessidade de reserva podem coexistir no RT-Mach. No caso das aplicações sem reserva, as mesmas são atribuídas a uma reserva padrão que só é escalonada quando não existem outras aplicações com reserva garantida na fila de processos prontos.

2.1.1 Arquitetura

O RT-Mach provê um mecanismo intermediário de 2 vias entre a aplicação e a camada de sistema onde a aplicação pode especificar os seus requisitos à camada mais inferior. A camada de sistema é capaz de informar os estados dos recursos alocados a cada aplicação de maneira individual ou considerando outras aplicações co-residentes.

A figura 1 apresenta a arquitetura do servidor do RT-Mach a qual é composta de três *threads*: a de controle de admissão, que determina se novas requisições de QoS podem ser atendidas; a de controle dinâmico de qualidade, que, periodicamente, verifica junto

[2]http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

ao *kernel* o uso dos recursos reservados pelas aplicações; e a de apresentação que ilustra graficamente informações das reservas feitas a cada cliente.

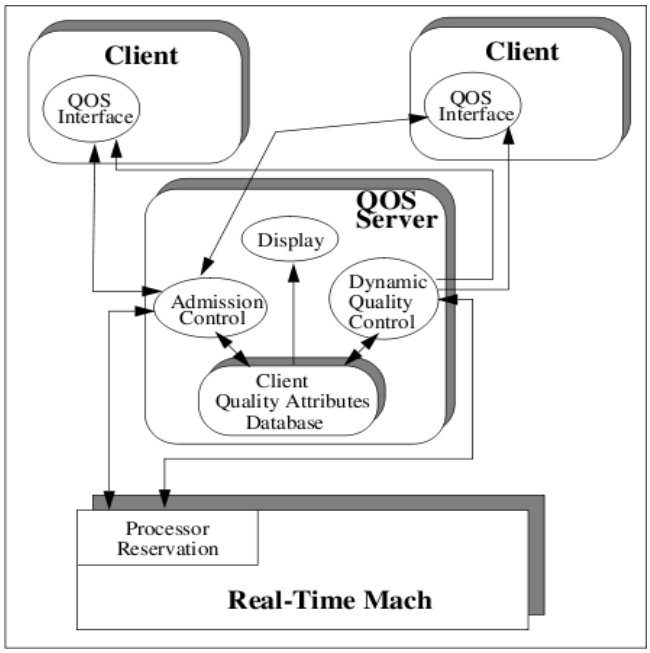

Figura 1: Arquitetura do servidor do RT-Mach [3].

Clientes acessam o servidor RT-Mach através de chamada a bibliotecas. As chamadas ocorrem, principalmente, para o registro da presença de novos clientes e para requisições e alterações dos parâmetros de QoS das aplicações. O servidor, por sua vez, notifica os clientes sobre mudanças na alocação de recursos através de um canal definido pela aplicação no momento de sua criação.

Atributos de QoS

As aplicações podem submeter requisições de qualidade de serviço para alocação de recursos ao servidor RT-Mach. Na definição dos atributos de QoS é preciso especificar o nível desejado de tempo de processamento e o período em que esse tempo ocorre. O servidor QoS então relaxa esse nível e período, gerando taxas mínimas e máximas de tempo de CPU e período permitidos.

Políticas de Ajuste de Qualidade

As aplicações podem decidir qual política de ajuste de qualidade desejam que seja empregada caso os recursos alocados a elas sofram depreciação ou melhoria de desempenho. Situações como essas ocorrem após a admissão de novos processos e durante a política de controle dinâmico de qualidade.

As opções de política de ajuste consistem nas combinações do par (*tempo_computação, período*). Dessa maneira, frente à mudanças na alocação dos recursos, existem escolhas para: ajustar o tempo de processamento e conservar o período, ajustar o período e con-

servar o tempo de processamento, manter ambos os valores constantes, ajustar cada valor em passos, e negociar ajustes com o cliente.

Controle de Admissão

Uma política de controle de admissão permite que o servidor de QoS decida se aceita uma nova requisição de execução e a que nível de qualidade. No RT-Mach, os parâmetros de reserva de um cliente em execução (processo antigo) podem ser alterados para que novos clientes (processos novos) sejam admitidos. Existem seis diferentes políticas de controle de admissão assim como ilustra a tabela 2, onde *mínimo*, *desejado* e *atual* correspondem ao nível de QoS mínimo, ideal e atual dos clientes.

Política	**Processo novo**	**Processos antigos**
Mínimo-mínimo	mínimo	mínimo
Mínimo-desejado	mínimo	desejado
Mínimo-atual	mínimo	atual
Desejado-mínimo	desejado	mínimo
Desejado-desejado	desejado	desejado
Desejado-atual	desejado	atual

Tabela 1: Políticas de controle de admissão do servidor de QoS.

No caso da política *Mínimo-desejado*, por exemplo, a aplicação admitida pelo controle, se necessário, terá o seu nível de qualidade reduzido, diferentemente das aplicações antigas que, não importa o que ocorra no sistema, devem permanecer com os níveis de alocação desejados.

Políticas de Controle de QoS

O servidor de QoS consulta a cada 5 segundos, o *kernel* para descobrir possíveis clientes que estejam subutilizando ou sobrecarregando os recursos reservados a ele de acordo com os seus parâmetros de QoS. Se detectado algum desvio de comportamento, o servidor é capaz de alterar a alocação de recursos aquela aplicação de acordo com diferentes políticas de controle de QoS com natureza muito próxima às das políticas de controle de admissão.

2.1.2 Casos de Uso

O RT-Mach é um projeto com diversos relatos de uso. Em [3], os autores descrevem uma aplicação de teleconferência distribuída, chamada RT-Phone, implementada com o objetivo de analisar o desempenho do mecanismo de reserva de processamento presente no RT-Mach. Os resultados dos experimentos indicaram que a reserva é capaz de atender aplicações de tempo real dinâmicas e aplicações multimídias de maneira satisfatória. Em [11], Mercer e Rajkumar descrevem ferramentas para a monitoração e controle de recursos. O trabalho [12] de Molano, Juvva e Rajkumar investiga a criação de um sistema de arquivos de tempo real o qual garante o tempo de acesso aceitável a múltiplas aplicações concorrentes que requeiram acesso ao disco em tempos e volumes distintos. Por fim, em [13], Lee *et al* propõe uma arquitetura para o escalonamento de processamento de protocolos de comunicação em sistemas de tempo real que utilizam o RT-Mach. Essa arquitetura deixaria

o controle do tempo para o processamento de protocolos a cargo das aplicações e evitaria que inversões de prioridade dos processos participantes da comunicação ocorressem.

2.2 Linux/RK

Linux/RK é uma implementação de um *Resource Kernel* baseada no Linux e desenvolvida no Real-time and Multimedia Systems Laboratory na Carnegie Mellon University. Além da reserva de recursos, o Linux/RK provê portabilidade através do uso de um subsistema chamado *Portable Resource Kernel*. Esse subsistema consiste em um módulo independente do *kernel* acoplado ao núcleo do SO através de pequenas modificações no código fonte do Sistema Operacional [14, 2].

2.2.1 Mecanismo de Reserva

No Linux/RK, CPU, memória, disco e rede são recursos que podem ser reservados. As reservas são garantidas pelo *kernel* e para fazê-las é preciso especificar os parâmetros de período de recorrência, de tempo de processamento a cada recorrência e de tempo máximo no qual o processamento deve estar disponível em cada período.

Quando uma aplicação não utiliza de forma adequada os recursos reservados a ela, o gerenciador de recursos do *Portable RK*, ou o *kernel* do SO, a notifica e então ela está apta a alterar seus parâmetros de QoS e assim dar início a um novo balanceamento de utilização no sistema.

O Linux/RK apresenta diferentes formas de tratar os diversos recursos gerenciados por ele. Porém, a interface de reserva desses recursos é única para que modificações no *kernel* não sejam necessárias a cada novo tipo de recurso adicionado ao sistema de reservas. Dessa maneira, para cada tipo de recurso há uma interface real, para o tratamento das particularidades de cada classe, e uma interface abstrata, para o tratamento das funções básicas de reserva comuns a todos os recursos.

Para garantir a reserva de recursos, é preciso haver mecanismos para:

- **Controle de Admissão** - A cada pedido de execução, o controle de admissão testa se o mesmo pode ou não ser atendido. No caso de confirmação, uma reserva baseada nos parâmetros de requisição é feita;
- **Política de Escalonamento** - Controla a alocação dinâmica dos recursos para que as tarefas recebam o seu percentual previamente reservado;
- **Observação (enforcement)** - Monitora a atual utilização dos recursos de acordo com os parâmetros de reserva das tarefas evitando o uso indevido dos mesmos;
- **Contabilidade** - Monitora a utilização acumulada de recursos por cada aplicação. Essa informação alimenta as decisões da política de escalonamento e do mecanismo de observação e pode ser consultada também diretamente por outros processos.

2.2.2 Arquitetura

O Linux/RK é formado por dois componentes: o *kernel* do Linux e o *Portable Resource Kernel*. O *Portable RK* é executado como um módulo de um Sistema Operacional Linux.

Enquanto o sistema hospedeiro fornece funções básicas de um SO (gerenciamento de processos, sistema de arquivo e rede), o *Portable RK* as reimplementa adicionando, por exemplo, mecanismos de reserva de recursos, controle de admissão e escalonamento de processos. Como ilustrado na figura 2, esses mecanismos são ativados através de uma interface contendo objetos de *callback*. Os objetos capturam a ocorrência de ações de escalonamento no *kernel* do Linux e as direciona para tratamento no *Portable RK*.

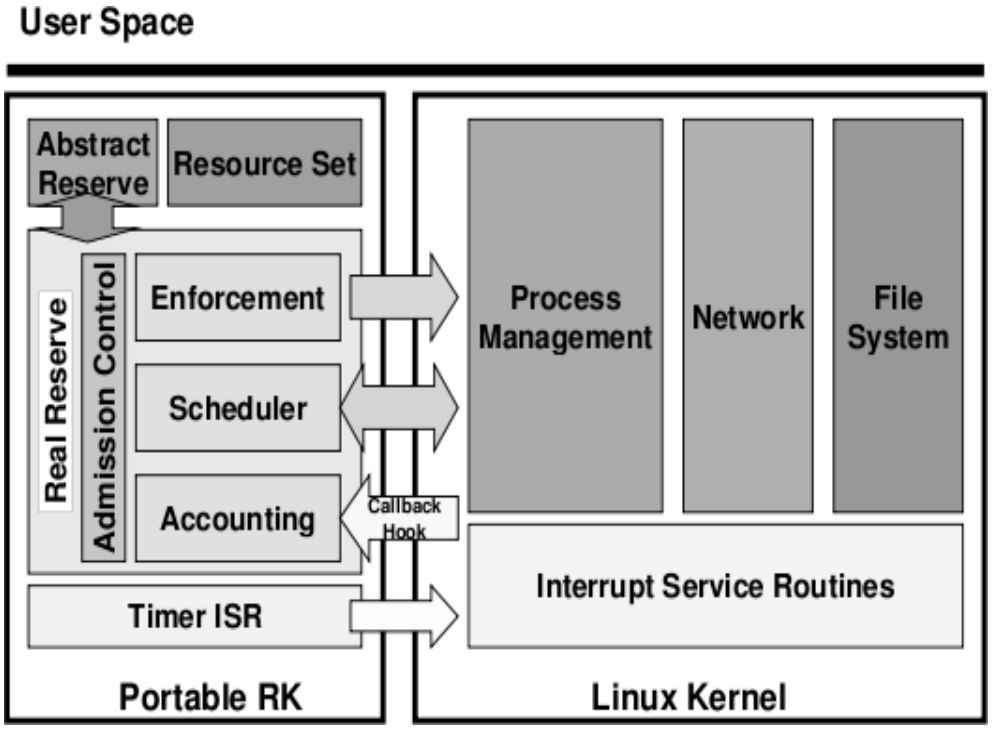

Figura 2: Arquitetura do Linux/RK [2].

O *Portable RK* disponibiliza uma API para uso dos programas em nível do usuário. Através dessa API é possível agrupar recursos em conjuntos (*resource sets*) os quais podem ser, também através dessa API, utilizados na criação de reservas para a execução de processos.

2.2.3 Implementação do Escalonamento

A implementação do *Portable RK* depende da existência de um Sistema Operacional cujo escalonador apresente um paradigma de escalonamento baseado em prioridade. Isso porque o *Portable RK* utiliza esse paradigma para emular diferentes políticas de escalonamento.

Uma *thread* do *Portable RK* consiste em um temporizador ISR(*Interrupt Service Routine*). Ela trata da observação, substituição de reservas e ajuste de prioridade dos objetos em execução. Já a segunda *thread* é executada com uma baixa prioridade e trata da execução de objetos que não se encontram priorizados. Quando não há nenhum processo pronto para execução no nível de prioridade mais alto, essa *thread* é invocada e escolhe a próxima tarefa a ser executada e a atribui ao nível de prioridade mais alto reservado a esse propósito.

2.2.4 Casos de Uso

Em circuitos CMOS, reduções na freqüência do processador e na voltagem de alimentação podem reduzir gastos de energia. As técnicas de voltagem escalar dinâmicas visam atingir essas reduções sem prejudicar o tempo de execução de aplicações. Em [15], Saewong e Rajkumar apresentam o uso do Linux/RK como meio para incorporar algoritmos de voltagem escalar dinâmica na CPU visando diminuir o consumo de energia durante a execução de aplicações. Essa pesquisa fez surgir então o *kernel Energy-Aware Linux/RK* [16].

2.3 RED-Linux

O RED-Linux (*Real-time and Embedded Linux*) consiste em um *kernel*, de tempo real desenvolvido por pesquisadores da Universidade da Califórnia em Irvine, cujo objetivo é expandir o domínio Linux de aplicações de propósito geral para aplicações de tempo real e embutidas de forma eficiente e flexível [17, 18].

Para atingir eficiência, um *patch* para *kernel*, com uma temporização de alta resolução, foi implementado. O *kernel* provê baixos atrasos de "preempção"e tratamento de interrupção, sendo possível despachar tarefas para execução em uma média de 50 microssegundos . A flexibilidade, por sua vez, foi provida por meio da construção de um *framework* que suporta os três paradigmas de escalonamento mais utilizados - os dirigidos por prioridade, compartilhamento de tempo e compartilhamento de espaço. Algoritmos de escalonamento dependentes de aplicação podem ser utilizados. Os mesmos algoritmos podem ser implementados em bibliotecas e, posteriormente, reutilizados em outras aplicações.

2.3.1 Paradigmas de Escalonamento

Existem, basicamente, três tipos de paradigmas de escalonamento: o orientado a prioridade, o orientado a tempo e o orientado a espaço. No paradigma orientado a prioridade, a cada decisão de escalonamento do sistema, a tarefa com maior prioridade entre os processos prontos é escolhida para execução. Essa prática, porém, pode resultar em situações de *starvation* nos processos de baixa prioridade.

No segundo paradigma, o compartilhamento de tempo evita que situações de *starvation* ocorram, pois todas as tarefas têm igual fatia de tempo para acessar o processador. Por outro lado, pode haver um desperdício de recursos visto que não é possível usar o tempo ocioso não utilizado por outras tarefas.

O paradigma orientado a espaço não diferencia processos, como no paradigma orientado a prioridade, e não garante um tempo mínimo de processamento às tarefas como no paradigma orientado a tempo. Sua característica é dividir a CPU entre as tarefas de forma que cada uma fique com uma porcentagem dedicada a ela. Esse paradigma não garante uma execução adequada de aplicações de tempo real em situações de alta carga.

Tipicamente, Sistemas Operacionais implementam um único paradigma de escalonamento, deixando a cargo do desenvolvedor escolher qual SO utilizar de acordo com as características de sua aplicação. Essa escolha, pouco flexível, motivou o desenvolvimento do *Framework* Geral de Escalonamento que uniu as características dos três paradigmas descritos em um só. Nesse *framework* é possível implementar diversos algoritmos de escalonamento assim como os descritos em [17].

2.3.2 Atributos de Escalonamento

Para a execução de uma tarefa é preciso especificar o valor de quatro parâmetros: a prioridade, o tempo de início, o tempo permitido de execução e o prazo limite de finalização do processo. A prioridade apresenta a noção de importância entre os processos. Nela, o processo com maior prioridade, tem maior preferência de execução. O tempo de início e término de execução indicam os tempos absolutos em que os processos podem começar a sua execução e devem terminá-la, respectivamente. Por fim, o tempo permitido apresenta o tempo máximo de execução destinado a uma tarefa. Esses quatro parâmetros influenciam a seleção de tarefas para execução assim como será descrito na Seção 2.3.3.

2.3.3 Arquitetura

A arquitetura do RED-Linux é composta por dois componentes independentes, o despachante e o alocador, responsáveis pelo escalonamento de baixo nível e pela definição dos parâmetros de qualidade de serviço, respectivamente. Essa estrutura modularizada do RED-Linux permite que ela seja facilmente reconfigurada para diferentes aplicações através da redefinição da política de escalonamento implementada pelo alocador. Nessa reconfiguração, não há necessidade do módulo do despachante ser alterado. A figura 3 ilustra o relacionamento entre o despachante e o alocador os quais serão descritos em detalhes nas Seções 2.3.3 e 2.3.3.

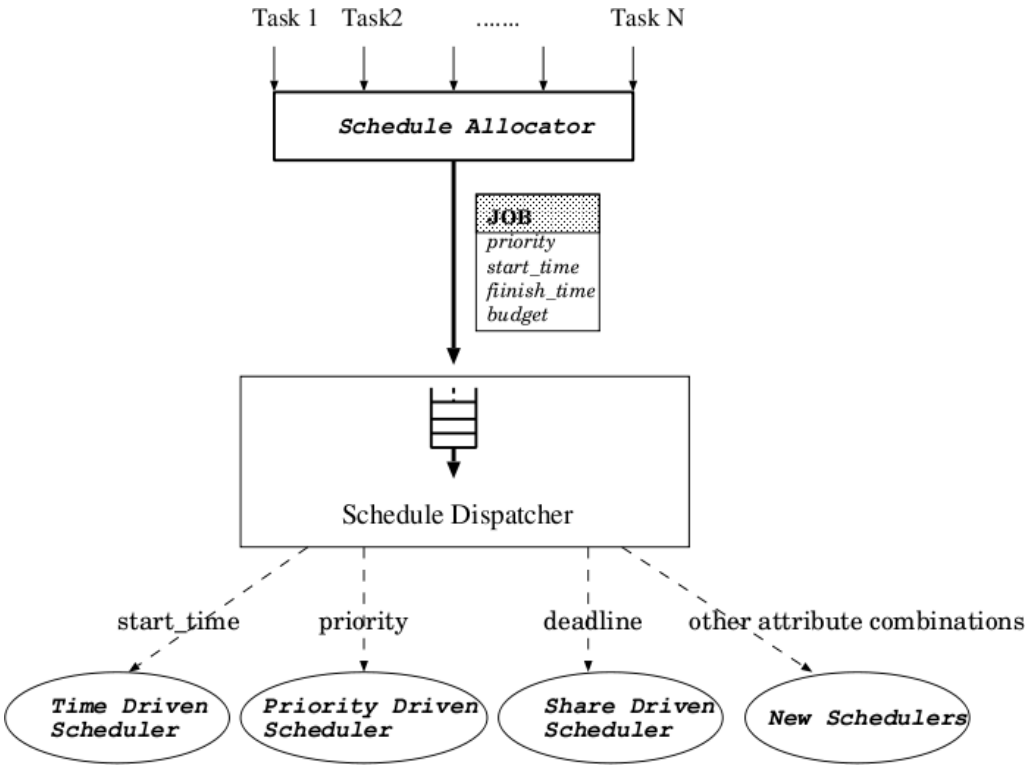

Figura 3: Arquitetura do escalonamento do RED-Linux [17].

Alocador

Em sistemas computacionais, é dever do Sistema Operacional prover um mecanismo flexível de escalonamento, mas não determinar a política a qual deve ser executada por esse mecanismo. No RED-Linux, essa limitação do SO é realizada pelo alocador o qual é capaz de implementar diferentes políticas de escalonamento de acordo com as necessidades de cada aplicação, garantindo a elas a reserva de CPU necessária para que o prazo limite de espera por finalização não seja ultrapassado. As políticas de escalonamento devem ser baseadas nos quatro atributos de escalonamento das tarefas os quais são configurados pelo alocador.

O alocador interage com o processo despachante através de uma API implementada por duas bibliotecas, uma executada no nível do usuário e outra no nível do *kernel*. A primeira das APIs deve ser utilizada caso o alocador necessite ter acesso a informações da aplicação. Caso contrário, ambas as interfaces podem ser usadas.

O processo alocador é executado em tempo real e deve alocar uma quantidade suficiente de processamento para ser capaz de realizar decisões em tempo de execução. Além disso, tipicamente, ele é atribuído à maior prioridade do sistema para outros processos não sejam capazes de bloqueá-lo.

Despachante

O processo despachante realiza o escalonamento de aplicações de tempo real que tenham sido registradas no alocador. Ele é implementado como um módulo do RED-Linux e se comunica com o *kernel* através de funções definidas na Interface de Escalonamento Virtual (VSI) assim como ilustrado na figura 4 onde GSI representa a Interface de Escalonamento Geral.

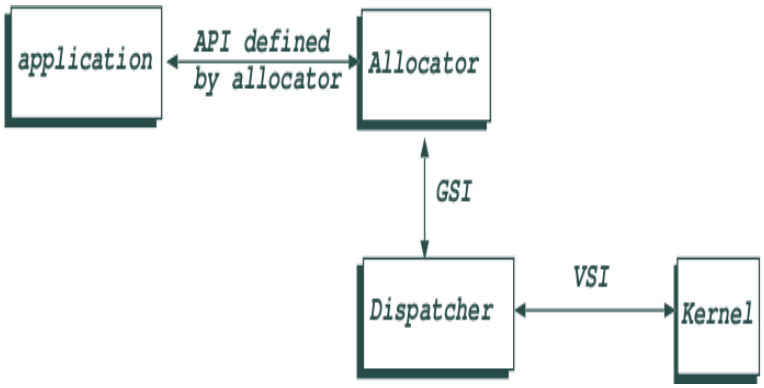

Figura 4: Interação entre componentes RED-Linux através de APIs [17].

A fila de processos gerenciada pelo despachante (representada na figura 3) é dividida em quatro novas filas, cada uma representada por um dos possíveis quatro estados das tarefas: pronto, ativo, bloqueado (*sleep*) e terminado. O estado pronto é o primeiro pela qual uma tarefa passa, só saindo dele quando o tempo de início do processo é alcançado. Nesse momento, essa tarefa entra na fila de processos ativos, de onde já está apta a entrar em execução caso seja selecionada pelo despachante (maior prioridade do grupo de tarefas ativas). Caso uma tarefa ativa entre em espera por algum recurso, ela é movida para a fila de processos bloqueados, saindo desse estado somente quando for sinalizada pelo recurso

em espera. Por fim, as tarefas entram no estado terminado quando os seus tempos de permissão de execução se expiram. A figura 5 ilustra o diagrama de estados e as formas de transição entre eles.

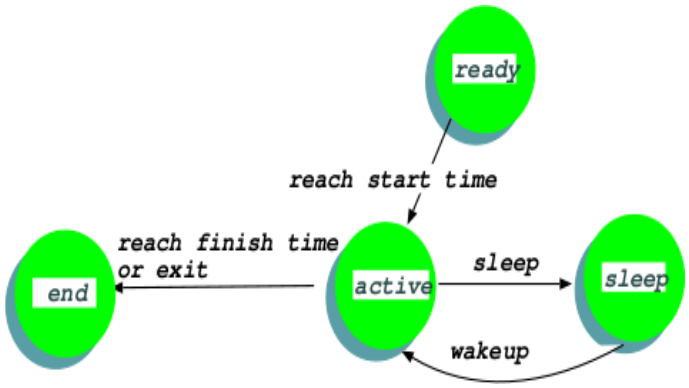

Figura 5: Diagrama de estados das tarefas no RED-Linux [17].

2.3.4 Monitoramento e Adaptação

Muitas políticas de escalonamento utilizam informações dinâmicas para atribuir processos à CPU. Algumas, por exemplo, podem alterar a porcentagem de CPU atribuída a um processo, doar uma fatia de tempo inutilizado ou ainda alterar a prioridade de uma tarefa.

No RED-Linux, o despachante é o encarregado de realizar o monitoramento de execução dos processos e disponibilizar a informação colhida ao alocador. Esse último pode então, adaptar a política de escalonamento ativa baseada nos valores lidos do despachante. Antes de trocar mensagens, esses componentes agrupam informações para que não ocorra sobrecarga na comunicação entre eles.

2.3.5 Casos de Uso

Os casos de uso envolvendo o RED-Linux se baseiam, principalmente, na proposta de políticas de escalonamento para aplicações de tempo real. Em [19], Wang e Lin adicionaram um parâmetro *group number* ao *framework RED-Linux* para a identificação de tarefas aperiódicas e de tempo real. Essa adição possibilitou a integração eficiente de escalonadores hierárquicos no *framework*. Já em [20], Song Wang, Lin e YuChung Wang descrevem a extensão do conceito de grupo de forma que os recursos alocados também possam ser gerenciados de forma hierárquica.

3 Aplicações no Nível do Usuário

A reserva de processamento através de *resource kernels*, apesar de consistir em uma boa prática de desempenho, esbarra na dificuldade de se alterar Sistemas Operacionais em ambientes computacionais já configurados e em uso. Ambientes com essas características podem ser beneficiados com aplicações que gerenciem o processamento no nível do usuário, isto é, sem que alterações no núcleo do sistema sejam necessárias. Esse é o caso do DSRT, um sistema de gerenciamento de CPU executado no nível do usuário, que oferece uma API

que pode ser utilizada por qualquer tipo de aplicação (periódicas ou não) para requisitar reservas de processamento durante a sua execução.

3.1 DSRT

O DSRT (*Dynamic Soft Real-Time*) é um sistema para o gerenciamento de processamento de aplicações de tempo real não-críticas desenvolvido no Departamento de Ciência da Computação da Universidade de Illinois em Urbana Champaign. Com versões para os ambientes Unix e Windows, o DSRT é executado em nível de usuário, não requerendo nenhuma modificação no *kernel* do Sistema Operacional. Essa característica atribui a ele facilidades de implantação em sistemas já configurados [5, 6].

O DSRT adota o paradigma de escalonamento baseado em prioridades, onde processos com maiores prioridades são executados anteriormente aos processos de menor prioridade. No DSRT, processos de tempo real apresentam as mais altas prioridades atribuídas de forma estática, enquanto os demais são atribuídos a prioridades mais baixas de forma dinâmica.

A implementação do gerenciamento de processos é feita por um processo servidor executado como um *daemon* na máquina hospedeira do sistema. Como o servidor deve alterar as prioridades dos processos submetidos à execução, ele deve ser executado com privilégios administrativos. Além disso, por consistir no elemento fundamental do DSRT, ele deve apresentar a maior prioridade entre os processos no nível de usuário do Sistema Operacional.

3.1.1 Estrutura de Prioridades

A estrutura de prioridades implementada pelo DSRT é apresentada na tabela 2. Existem duas classes de processos: os de tempo real e os de tempo compartilhado. A classe de tempo real apresenta prioridades fixas e se divide em dois grupos: processos com alta prioridade e processos em espera com baixa prioridade. Já a classe de compartilhamento de tempo apresenta prioridade dinâmica e agrupa todos os processos que não apresentam a necessidade de serem executados em tempo real, tal como o *broker* que será introduzido na Seção 3.1.2.

Classe	**Prioridade**	**Processo**
Tempo Real	maior possível	despachante
	segunda maior	processos de tempo real em execução
	...	Não utilizadas
Compart. de tempo	qualquer	processos não-tempo-real
Tempo Real	mais baixa	processos de tempo real em espera

Tabela 2: Estrutura de prioridades do DSRT.

O processo despachante, responsável por colocar em execução os processos escalonados pelo *broker*, é executado com a maior prioridade do sistema. Os processos escolhidos por ele para execução recebem a segunda maior prioridade do sistema. Processos escalonados pelo *broker*, mas fora da sua fatia de tempo para processamento, têm a sua prioridade reduzida ao grupo de processos em espera.

Periodicamente, o processo despachante seleciona processos do *pool* de espera e os despacha para execução. Durante o resto do tempo, esse processo dorme e os processos selecionados são executados. Se não houver processos de tempo real prontos para execução, os processos de tempo compartilhado com prioridades dinâmicas entram em execução.

3.1.2 Arquitetura do Servidor

O processo servidor é constituído por três componentes - o *Broker*, a Tabela de Despacho e o Despachante assim como ilustra a figura 6. Nas subseções seguintes, esses componentes e suas interações serão descritos.

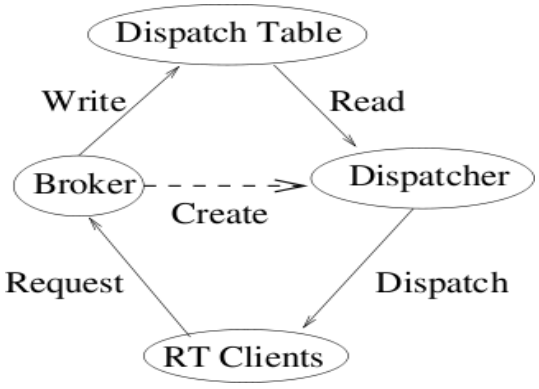

Figura 6: Arquitetura do servidor do DSRT [5].

Broker

Como ilustrado na figura 6, o *broker* é a porta de entrada para requisições de execução dos clientes. É função do *broker* decidir, através de um controle de admissão, se tais requisições podem ser escalonadas. No caso afirmativo, o processo cliente submetido tem a sua prioridade alterada para a classe de espera e, consecutivamente, é inserido no grupo de processos dessa classe. Em seguida, um plano de escalonamento para o processo, baseado em um algoritmo pré-determinado, é escrito na tabela de despacho.

Como a admissão e o escalonamento de processos não necessitam ser feitos em tempo real, o *broker* consiste em um processo *daemon*, com privilégios de administrador para a alteração de prioridades dos processos clientes, executado com prioridade dinâmica. De maneira diferente, o despacho dos processos não pode apresentar um tempo de execução variável e, dessa forma, essa tarefa é realizada por um novo processo despachante de tempo real criado pelo *broker*.

Tabela de Despacho

A tabela de despacho é uma área fixa de memória compartilhada entre o *broker* e o despachante. Nela, o *broker* escreve o escalonamento planejado para um determinado processo. O despachante, por sua vez, lê dessa tabela os escalonamentos já determinados e os executa.

A tabela de despacho é composta por *slots* que correspondem a fatias de tempo da(s) CPU(s). Um *slot* pode ser atribuído a um processo de tempo real, a um grupo de processos

de tempo real ou entregue ao escalonador do Sistema Operacional o qual pode colocar processos de baixa prioridade, isto é, processos sem a característica de tempo real, em execução. Um número mínimo de *slots* livres é mantido pelo *broker* e dedicado aos processos de baixa prioridade a fim de se evitar a ocorrência de *starvation* nos mesmos. Esse número mínimo é flexível e deve ser ajustado para refletir as necessidades de uso de cada máquina.

Despachante

O despachante é um processo criado pelo *broker* para a execução de um escalonamento escrito, pelo último, na tabela de despacho. Ele é finalizado quando não há mais nenhum processo na tabela de despacho, ou seja, quando não houver nenhum processo de tempo real a ser escalonado no sistema.

Como a execução de um escalonamento não pode sofrer atrasos, o despachante consiste no processo com a maior prioridade fixa possível. Ao encaminhar um "*slot*"para execução, o despachante adormece e só é acordado por um temporizador de tempo real que o sinaliza, periodicamente, ao início de cada novo *slot*. Ocorrem então três etapas considerando-se apenas um processo alocado para cada *slot*: i) a interrupção do processo em execução (1 troca de contexto); ii) a alteração das prioridades do processo interrompido para classe de processos em espera e do processo do novo *slot* para a classe de processos em execução (2 chamadas ao sistema); iii) o processo interrompido entra no estado de dormência e o processo da classe em execução é escalonado (1 troca de contexto).

Clientes

O processo cliente é inicializado com um nível de prioridade dinâmico, mas tem esse nível alterado pelo *broker* e pelo despachante quando o mesmo é aceito e despachado para execução, respectivamente. No cliente, a submissão de um pedido de execução ao servidor deve ser feita seguindo a sintaxe utilizada pelo RT Mach [3], ou seja, na forma (*período=p, porcentagem de CPU utilizada=u*). Além disso, uma descrição da forma como o processador deve ser reservado para execução pode ser feita como apresentado na Seção 3.1.3.

3.1.3 Classes de Serviços de CPU

A reserva descreve a forma que o processador será utilizado durante a execução do processo de tempo real. As opções de tempo são constante, variável, periódico e aperiódico, o que resulta em cinco classes de serviços.

CPU_RT_PCPT(Periodic Constant Processing Time)

Classe de processos cuja execução não deve ultrapassar o pico de processamento declarado a cada período. Se o pedido de reserva for aceito, a porcentagem de CPU especificada é garantida.

CPU_RT_PCCPT(Periodic Constrained Constant Processing Time)

O servidor sempre reserva para um processo PCCPT a porcentagem de CPU requisitada a cada período. Esta classe é muito importante para a execução de aplicações que não foram alterados para utilizar a API do DSRT.

CPU_RT_PVPT(Periodic Variable Processing Time)

Além do período e do pico de processamento, esta classe necessita de mais dois parâmetros para a execução de seus processos: o tempo sustentável e a tolerância a disparos. Processos da classe PVPT não devem ultrapassar o pico de processamento a cada período declarado e os seus disparos acumulados - definidos pelo uso cumulativo de processamento além do tempo sustentável - não devem exceder a tolerância de disparos esperada. Caso a reserva requisitada seja aceita, a porcentagem de CPU especificada pelo tempo sustentável é garantida a cada período. Além disso, os disparos de processamento por, garantidamente, serem limitados superiormente, quando ocorrem, são executados com a mais alta prioridade possível.

CPU_RT_ACPU(Aperiodic Constant Processing Utilization)

Processos ACPU devem especificar a quantidade relativa de processamento a cada iteração. O valor informado deve então ser menor que o pico de processamento útil - o único parâmetro de execução dessa classe. Caso a reserva seja aceita, o tempo correspondente ao pico de processamento útil é garantido a cada período;

CPU_RT_EVENT

Nestsa classe, as reservas de processamento são válidas apenas para um único período no qual a execução não ultrapassa, em nenhum momento, o pico de processamento esperado.

3.1.4 Probe

O DSRT apresenta um mecanismo de *probe* que permite, de forma automática e precisa, determinar a que classe de serviço de CPU uma aplicação pertence. Além disso, ele também é capaz de determinar quanto tempo de processamento é necessário em uma reserva de processamento, basta que o processo a ser chamado não apresente nenhuma reserva válida e, no caso de um processo periódico, especifique o seu período de execução.

3.1.5 Monitoramento e Adaptação

Um processo pode ser iniciado para monitorar e informar, local ou remotamente, o estado do servidor e os processos de tempo real em execução. A partir das informações fornecidas pelo monitor, alterações na reserva de CPU podem ser invocadas automaticamente de acordo com estratégias de adaptações previamente definidas.

As estratégias de adaptação podem ser exponenciais, as quais calculam a média de uso de processamento em intervalos constantes de tempo; e estatísticas, as quais computam o número de iterações fora do padrão nesses mesmos intervalos.

3.1.6 Casos de Uso

Desde a sua primeira versão, o DSRT vêm sendo empregado em diversos projetos. Em [21], Foster *et al* descreve a inserção do DSRT em uma arquitetura para reserva antecipada e co-alocação de recursos para realização de QoS em infraestruturas de computadores paralelos e uniprocessados além de redes de serviços integrados. Em [22], Kim e Nahrstedt propõem um modelo de *broker* que permite a reserva antecipada e imediata de recursos.

Esse modelo apresenta um controle de admissão que separa a tarefa de escalonamento da de negociação (*brokerage*). A separação permite a redução do tempo de negociação para uso de recursos.

Dentre os casos de uso do DSRT há a descrição do trabalho [23], no qual Yuan propõe o R-EDF(*Reservation-based Preemptive Earliest Deadline First*) um algoritmo para escalonamento "preemptivo"baseado nos *deadlines* mais próximos das aplicações. Trabalhos mais recentes, porém, concentram-se em propostas para a construção de *frameworks*, esquemas e políticas de contenção de energia em dispositivos móveis através da adaptação da qualidade de serviço oferecida às aplicações[24, 25, 26, 27, 28].

4 Máquinas Virtuais

As vantagens de compatibilidade e portabilidade de *software* tornam-se evidentes quando se analisa a grande diversidade de *hardware* e Sistemas Operacionais existentes no mercado. Máquinas virtuais eliminam a necessidade de recorrentes desenvolvimentos para diferentes sistemas computacionais através da adição de códigos a uma plataforma de execução, criando a aparência de plataformas diferentes. Máquinas virtuais podem ainda apresentar Sistema Operacionais e conjuntos de instruções diferentes daqueles implementados no nível mais baixo do *hardware* [29].

Máquinas virtuais apresentam ambientes isolados de execução garantindo segurança e simplicidade de migração de processos. Essas características são bastante indicadas para a solução de problemas encontrados em ambientes de Grades Computacionais, onde códigos arbitrários podem ser executados em uma ampla variedade de plataformas [30]. Muitas pesquisas têm se voltado ao emprego de Máquinas Virtuais como mecanismos de *sandbox* para grades computacionais [7, 8, 9, 10, 31]. Dentre as iniciativas, destaca-se o Xen, uma implementação de monitor de máquina virtual (VMM) que têm ganho grande espaço nas pesquisas sobre virtualização de ambientes.

4.1 Xen

O Xen consiste em um monitor de máquina virtual, ou *hypervisor*, originalmente desenvolvido por pesquisadores da Universidade de Cambridge. Ele permite aos Sistemas Operacionais compartilhar seus recursos de maneira segura e controlada sem perder em funcionalidade e nem em desempenho [32]. Suas principais características são:

- Migração de máquinas virtuais em execução entre diferentes máquinas físicas;
- Suporte a até 32 CPUs virtuais por máquina virtual instalada;
- Suporte às plataformas x86/32, x86/64, PowerPC e POWER5;
- Tecnologia de Virtualização Intel (VT-x) para a instalação de máquinas virtuais com sistemas operacionais não-modificados;
- Escalabilidade de até 100 máquinas virtuais por *host*;
- Suporte excelente para *hardware*, incluindo quase todos os *drivers* Linux.

4.1.1 Arquitetura

O modelo de proteção de processadores Intel x386 (primeira infra-estrutura de testes do Xen) é composto por quatro anéis, onde o anel rotulado 0, e com maior prioridade, executa o Sistema Operacional; o anel 3, as aplicações dos usuários; enquanto os anéis 1 e 2 são raramente utilizados. No Xen, essa estrutura de anéis é alterada: o monitor executa no anel 0 enquanto os SOs clientes executam no anel 3 [3]. Durante a inicialização, o Xen inicia um novo *kernel* alterado no anel 1, formando juntamente com o anel 0 o domínio 0. Esse domínio apresenta privilégios especiais de gerenciamento sendo capaz de, por exemplo, criar novos domínios, destruí-los ou migrá-los. Dentro do domínio 0 é executado um processo *daemon* denominado *xend* responsável por funções de gerenciamento relacionadas às máquinas virtuais. Dessa maneira, o *xend* deve ser executado com privilégios de administrador do sistema.

No projeto de desenvolvimento do Xen, um dos objetivos básicos é separar políticas dos mecanismos de implementação em si. Assim, apesar do monitor estar envolvido em tarefas como escalonamento de uma CPU entre domínios, não há nem mesmo a necessidade de ele ter conhecimento de questões de mais alto nível como, por exemplo, de que maneira essa CPU será compartilhada. A arquitetura resultante dessa prática modularizada entre políticas e mecanismos faz com que o monitor apresente somente operações de controle básicas exportadas para domínios autorizados. Decisões mais complicadas, tais como o controle de admissão, devem ser tratadas por programas executando sobre os Sistemas Operacionais clientes.

A figura 7 ilustra a estrutura de uma máquina física executando o monitor Xen com diferentes SOs clientes. A estrutura é formada por múltiplas camadas com diferentes privilégios, onde o monitor Xen pertence à camada com maior prioridade.

A comunicação entre o monitor e os domínios ocorre via chamadas síncronas dos domínios para o monitor, e via notificações do monitor para os domínios. As chamadas síncronas podem ser feitas através de *hypercall*[4] enquanto as notificações são entregues através de mecanismos de eventos assíncronos.

4.1.2 Escalonamento

O Xen inclui em sua inicialização opções de escalonamento similares àquelas que dividem o tempo de CPU pelos processos dos usuários. Na versão 2.0, a opção padrão consistia no algoritmo BVT(*Borrowed Virtual Time*) o qual provê uma baixa latência no processo de despacho de eventos aos domínios do Xen. No BVT, a troca de contexto é similar ao conceito de *quantum* nos escalonadores tradicionais, o que provê um compartilhamento justo do tempo de CPU [33]. Outras opções de escalonadores eram:

- **Atropos** - Consiste em um escalonador de tempo real que provê garantias no uso da CPU. Cada domínio tem um período e uma fatia de tempo associado a ele;
- **Round-Robin** - Escalonador incluso apenas como demonstração. Ele compartilha o tempo de CPU entre os diversos domínios de maneira cíclica e equivalente;

[3] Por esse motivo de alteração de prioridades entre os Sistemas Operacionais clientes e o monitor é que o último é chamado de *hypervisor*.

[4] Uma *hypercall* é uma abstração dos domínios para o *hypervisor* assim como uma chamada ao sistema é uma abstração da aplicação para o *kernel*. Domínios usam *hypercalls* para requisitar operações com privilégios ao monitor Xen.

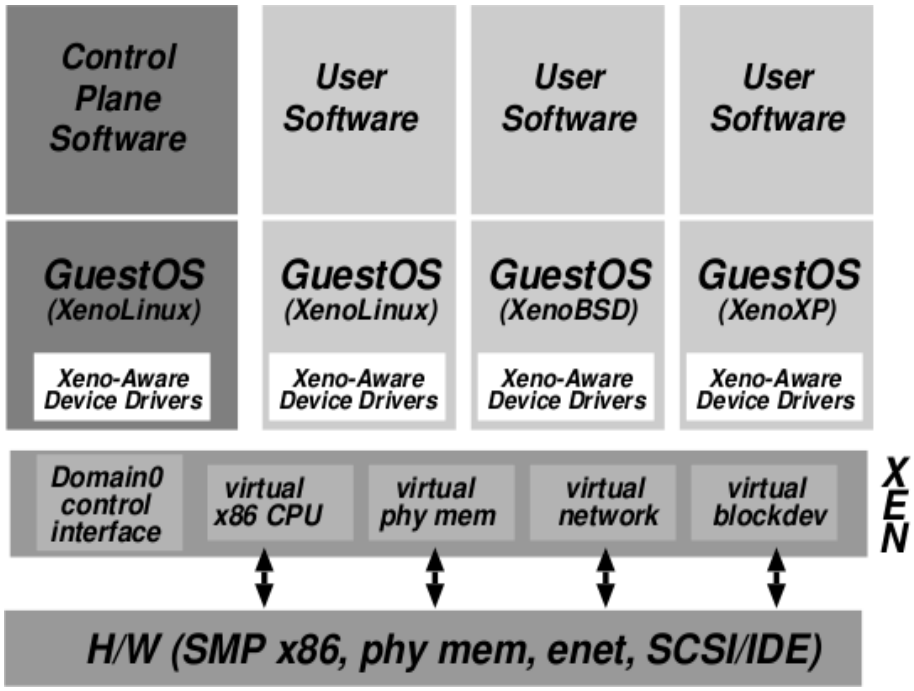

Figura 7: Estrutura de uma máquina executando o monitor Xen e diversos SOs clientes [32].

- **sEDF** - Escalonador que utiliza algoritmos de tempo real tal como o *Earliest Deadline First* para garantir tempos de execução precisos. Ele provê compartilhamento de CPU através da atribuição de pesos aos domínios.

Na versão 3.0, o escalonador padrão passou a ser baseado em créditos. Esse escalonador balanceia, de forma automática, a carga das CPUs virtuais dos sistemas clientes entre todas as CPUs reais de uma máquina SMP. No escalonador de créditos, cada domínio apresenta um peso e um limite. Um domínio com peso 512, por exemplo, terá o dobro de CPU alocada se comparado a um domínio de peso 256. O parâmetro limite fixa o máximo de CPU, em porcentagem, que um domínio pode alocar mesmo em situações que haja ciclos ociosos no sistema.

4.1.3 Migração de Domínios

Máquinas virtuais do Xen podem ser movidas de uma máquina a outra de maneira regular ou direta. Na maneira regular, antes de ser transferida, a máquina virtual é parada e tem seu conteúdo de memória copiado. Na máquina destino, a MV transferida é então reiniciada. Já na migração direta, a MV é transferida sem que nenhuma pausa na sua execução seja realizada.

Na migração, o endereço MAC e IP de um domínio são movidos com ele. Dessa maneira, as máquinas fonte e destino devem estar localizadas na mesma camada e subrede IP. Ambas as máquinas devem estar executando o *daemon xend*. Além disso, a máquina destino deve contar com recursos físicos suficientes para a acomodação do novo domínio.

4.1.4 Casos de Uso

Em [8, 34, 35] utiliza-se o Xen na construção de protótipos de *sandboxes* para ambientes de grades computacionais. Os ambientes construídos provêem facilidades para a execução de códigos arbitrários e para o gerenciamento dos recursos, pois máquinas virtuais atuam como contêineres isolando os ambientes de cada cliente. O Xen também está presente na implementação de ambientes abstratos de execução - os *ambientes de trabalho virtuais*[5] - criados dinamicamente para o uso de clientes autorizados em máquinas gerenciadas pelo Globus Toolkit 4[6] [36].

O projeto XenoServer[7] está construindo uma infra-estrutura distribuída e aberta, onde clientes pagantes podem submeter suas aplicações de forma segura para execução em máquinas virtuais hospedadas em computadores espalhados globalmente. As taxas a serem pagas devem ser baseadas no uso de recursos ou na reserva dos mesmos. Mecanismos de armazenamento, descoberta de serviços, gerenciamento de recursos e segurança estão sendo desenvolvidos para atingir o objetivo proposto pelo projeto.

5 Estudo Comparativo e Conclusão

Esta seção destaca as principais características de cada projeto descrito nesta monografia. Tópicos como maturidade e portabilidade são expostos e comparados com o intuito de destacar as vantagens e desvantagens das abordagens adotadas na implementação dos sistemas.

5.1 Motivação

Os projetos apresentados neste estudo foram desenvolvidos por diferentes motivações. RT-Mach, Linux/RK e DSRT foram propostos especificamente para o gerenciamento de recursos com reserva de processamento apesar de atingirem esse objetivo de maneiras distintas (no nível de *kernel* e usuário). De forma diferente, o RED-Linux e o Xen formam desenvolvidos as propostas de flexibilizar o escalonamento de aplicações e virtualizar ambientes. Nesses dois projetos, a reserva de recursos veio como uma conseqüência do avanço das funcionalidades de gerenciamento dos recursos.

5.2 Maturidade

Neste documento, entende-se por maturidade a confirmação de uma abordagem como eficiente e precisa na solução de um determinado problema. Conseqüentemente, um projeto que apresente significativa maturidade tende a ser aceito e referenciado por diversos trabalhos relacionados assim como utilizado como parte de projetos mais abrangentes. Nesse sentido, o RT-Mach, o Linux/RK e o DSRT podem ser classificados como maturos, pois são utilizados em muitos trabalhos acadêmicos e citados, na literatura, em diversas propostas de desenvolvimento de mecanismos de reserva de recursos, assim como brevemente descrito nas seções de casos de uso correspondentes.

[5] http://workspace.globus.org/
[6] http://www.globus.org
[7] http://www.xenoservers.net/

O RED-Linux incentivou trabalhos relevantes principalmente na área de políticas de escalonamento. Esse sistema, porém, apesar de apresentar meios para a reserva de recursos - tais como a modularidade entre o alocador e despachante o que facilita a implementação de um controle de admissão - tem essa funcionalidade pouco explorada em seus trabalhos. Por esse motivo, o RED-Linux apresenta uma baixa maturidade em relação à reserva de processamento.

Em situação semelhante ao RED-Linux, encontra-se o Xen. Esse monitor de máquinas virtuais vem ganhando força entre os pesquisadores da área de gerenciamento de recursos, mas é discutível se tal fato é conseqüente do seu bom desempenho como monitor ou vindo do ressurgimento do interesse por técnicas de virtualização. De qualquer maneira, trabalhos que relacionam o Xen à reserva garantida de processamento ainda estão em fase inicial e ainda não geraram resultados significantes para a literatura.

5.3 Portabilidade

Resource kernels têm a dificuldade de apresentar alterações no núcleo do Sistema Operacional e, dessa forma, tendem a não ser portáveis. São os casos do RT-Mach e do RED-Linux. O Linux/RK, por outro lado, atinge certa portabilidade através do acoplamento do módulo *Portable Resource Kernel* ao *kernel* do Sistema Operacional e de uma pequena alteração ao núcleo do SO. O DSRT, por executar em nível de usuário, é facilmente portado para diferentes Sistemas Operacionais do tipo Unix e Windows. O Xen, assim como os *Resource kernels* necessita de uma pequena alteração no Sistema Operacional hospedeiro, mas caso o *hardware* da máquina hospedeira seja equipado com tecnologia de virtualização Intel VT ou AMD Pacifica, essa alteração torna-se desnecessária e a instalação de um ambiente Windows, por exemplo, cujo código é fechado, se torna viável. Todos os projetos apresentam a opção de execução de aplicações legadas em seus sistemas.

5.4 Flexibilidade de Escalonamento

A flexibilidade no escalonamento de aplicações está presente em todos os projetos estudados ainda que de maneira limitada. No RT-Mach existem primitivas que permitem às aplicações escolher diferentes políticas de escalonamento. No Xen, há a possibilidade de escolha de política de escalonamento no momento da inicialização do monitor. No Linux/RK, o *Portable RK* utiliza as prioridades do SO para emular diferentes políticas de escalonamento. Já no DSRT, a tarefa de implementação de regras de escalonamento é executada pelo *broker*. Por fim, no RED-Linux, a motivação da construção do *Framework* Geral de Escalonamento permite a implementação de novas políticas de escalonamento de maneira simplificada e altamente abrangente.

5.5 Suporte a Adaptação

Nos documentos utilizados como referência para este estudo, foram encontradas citações de suporte a adaptação dos parâmetros de escalonamento para todos os projetos. No RT-Mach a adaptação é realizada pelo Servidor de QoS. Já no RED-Linux o despachante é o responsável por detectar a necessidade de alteração de parâmetros e notificar o alocador que, em seguida, executa a mudança. No Linux/RK o *Portable RK*, ou mesmo o *kernel* do SO, são os responsáveis por notificar a aplicação sobre a necessidade de alteração de seus

parâmetros de QoS. No DSRT, a tarefa de monitoramento é realizada por um processo local ou remoto que, quando necessário, invoca mudanças de parâmetros automaticamente de acordo com estratégias previamente definidas.

5.6 Probe e Classes de Aplicação

O DSRT é o único sistema estudado que apresenta classificações para os diferentes tipos de execução de aplicações. Especificar as características de uma aplicação ajuda na obtenção de melhores desempenhos. Por outro lado, saber quais são essas características - periódica ou não, com tempo de processamento variável ou não - pode ser uma tarefa difícil. Dessa maneira, o DSRT dispõe de um mecanismo de *probe* responsável por determinar, por exemplo, a que classe de serviço de CPU uma determinada aplicação pertence e quanto tempo de processamento essa aplicação necessita. A tabela 3 sintetiza as principais características de cada projeto.

Características	**RT-Mach**	**Linux/RK**	**RED-Linux**	**DSRT**	**Xen**
Maturidade	alta	alta	baixa	alta	baixa
Portabilidade	não	sim	não	sim	sim
Flex. de Esc.	sim	sim	sim	sim	sim
Sup. a Adap.	sim	sim	sim	sim	não
Probe	não	não	não	sim	não

Tabela 3: Tabela comparativa dos sistemas de reserva de processamento.

5.7 Conclusão

Este trabalho objetivou o estudo de diferentes sistemas para a reserva de processamento em ambientes computacionais. De maneira geral, conclui-se que as reservas de processamento são realizadas com base no paradigma de escalonamento orientado a prioridades. Para que aplicações possam ser escalonadas, os processos gerenciadores das reservas, tipicamente, devem ser executados com privilégios de administrador e implementados em módulos independentes dos mecanismos executores dos escalonamentos.

Para a garantia da reserva de processamento, é preciso haver mecanismos que monitorem os recursos e controlem a admissão de novas aplicações. Esses mecanismos evitam que aconteçam sobrecargas nos servidores e, quando combinados com políticas dinâmicas de escalonamento, garantem uma maior utilização dos recursos computacionais.

Tradicionalmente, implementações para reserva de processamento são feitas de duas maneira - no nível do *kernel* ou do usuário - onde a escolha é baseada, principalmente, na necessidade de portabilidade do mecanismo de reserva em diferentes ambientes computacionais. Recentemente, uma terceira abordagem de reserva de recursos, através de virtualização, tem ganho destaque. Máquinas Virtuais permitem a criação dinâmica de diversos ambientes personalizados em uma única máquina física. Esses ambientes, por serem executados dentro de contêineres, são altamente seguros e migráveis. Além disso, máquinas virtuais oferecem um forte esquema de garantia de uso de recursos, pois, tipicamente, uma são configuradas com uma quantidade de memória específica e um tamanho de disco pré-determinado. Mecanismos para a reserva de CPU, por sua vez, ainda são bastante limitados.

Muitos trabalhos têm proposto o gerenciamento de recursos em ambientes distribuídos através de ambientes virtuais. As principais iniciativas vêm de pesquisadores ligados ao estudo de serviços Web para Grades Computacionais. Porém, no momento em que a reserva de processamento for consolidada, MVs poderão ser utilizadas para a execução de diversos tipos de aplicações como as multimídia e de tempo real, citadas neste estudo.

Referências

[1] RAJKUMAR, R.; JUVVA, K.; MOLANO, A. ; OIKAWA, S.. **Resource kernels: a resource-centric approach to real-time and multimedia systems**. Readings in multimedia computing and networking, p. 476–490, 2001.

[2] OIKAWA, S.; RAJKUMAR, R.. **Portable RK: A portable resource kernel for guaranteed and enforced timing behavior**. In: IEEE REAL TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM, p. 111–120, 1999.

[3] LEE, C.; RAJKUMAR, R. ; MERCER, C.. **Experience with processor reservation and dynamic qos in real-time mach**. In: IN PROCEEDINGS OF THE MULTIMEDIA JAPAN 96, Japan, March 1996.

[4] BANGA, G.; DRUSCHEL, P. ; MOGUL, J. C.. **Resource containers: A new facility for resource management in server systems**. In: IN PROCEEDINGS OF THE 3RD CONFERENCE ON SYMPOSIUM ON OPERATING SYSTEMS DESIGN & IMPLEMENTATION - OSDI '99, p. 45–58, New Orleans, USA, 1999. USENIX Association.

[5] CHU, H.-H.; NAHRSTEDT, K.. **A soft real time scheduling server in unix operating system**. In: IDMS '97: PROCEEDINGS OF THE 4TH INTERNATIONAL WORKSHOP ON INTERACTIVE DISTRIBUTED MULTIMEDIA SYSTEMS AND TELECOMMUNICATION SERVICES, p. 153–162, London, UK, 1997. Springer-Verlag.

[6] KAMADA, J.; YUHARA, M. ; ONO, E.. **User-level realtime scheduler exploiting kernel-level fixed priority scheduler**. In: IN PROCEEDINGS OF THE MULTIMEDIA JAPAN 96, Japan, March 1996.

[7] KEAHEY, K.; DOERING, K. ; FOSTER, I.. **From sandbox to playground: Dynamic virtual environments in the grid**. In: GRID '04: PROCEEDINGS OF THE FIFTH IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING (GRID'04), p. 34–42, Washington, DC, USA, 2004. IEEE Computer Society.

[8] SANTHANAM, S.; ELANGO, P.; ARPACI-DUSSEAU, A. ; LIVNY, M.. **Deploying virtual machines as sandboxes for the grid**. In: II WORKSHOP ON REAL, LARGE DISTRIBUTED SYSTEMS (WORLDS 2005), San Francisco, CA, December 2005.

[9] KRSUL, I.; GANGULY, A.; ZHANG, J.; FORTES, J. A. B. ; FIGUEIREDO, R. J.. **Vmplants: Providing and managing virtual machine execution environments for grid computing**. In: SC '04: PROCEEDINGS OF THE 2004

ACM/IEEE CONFERENCE ON SUPERCOMPUTING, p. 7, Washington, DC, USA, 2004. IEEE Computer Society.

[10] FIGUEIREDO, R. J.; DINDA, P. A. ; FORTES, F.. **A case for grid computing on virtual machines**. icdcs, 00:550, 2003.

[11] MERCER, C. W.; RAJKUMAR, R.. **An interactive interface and rt-mach support for monitoring and controlling resource management**. In: RTAS '95: PROCEEDINGS OF THE REAL-TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM, p. 134, Washington, DC, USA, 1995. IEEE Computer Society.

[12] MOLANO, A.; JUVVA, K. ; RAJKUMAR, R.. **Real-time filesystems. guaranteeing timing constraints for diskaccesses in rt-mach**. In: PROCEEDINGS OF THE 18TH IEEE REAL-TIME SYSTEMS SYMPOSIUM (RTSS'97), p. 155–165, San Francisco, CA, USA, December 1997.

[13] LEE, C.; YOSHIDA, K.; MERCER, C. ; RAJKUMAR, R.. **Predictable communication protocol processing in real-time mach**. rtas, 00:220, 1996.

[14] OIKAWA, S.; RAJKUMAR, R.. **Linux/rk: A portable resource kernel in linux**, 1998.

[15] SAEWONG, S.; RAJKUMAR, R. R.. **Practical voltage-scaling for fixed-priority rt-systems**. In: RTAS '03: PROCEEDINGS OF THE THE 9TH IEEE REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM, p. 106, Washington, DC, USA, 2003. IEEE Computer Society.

[16] SAEWONG, S.; RAJKUMAR, R.. **Optimal static voltage-scaling for real-time systems**. Submitted to the 23rd IEEE International Real-Time Systems Symposium (RTSS-2002), 2002. Disponível em http://www.cs.cmu.edu/ rtml/papers.html.

[17] WANG, Y.-C.; LIN, K.-J.. **Implementing a general real-time scheduling framework in the red-linux real-time kernel**. In: RTSS '99: PROCEEDINGS OF THE 20TH IEEE REAL-TIME SYSTEMS SYMPOSIUM, p. 246, Washington, DC, USA, 1999. IEEE Computer Society.

[18] LIN, K.-J.; WANG, Y.-C.. **The design and implementation of real-time schedulers in red-linux.** Proceedings of the IEEE, 91(7):1114–1130, 2003.

[19] WANG, Y.-C.; LIN, K.-J.. **The implementation of hierarchical schedulers in the red-linuxscheduling framework**. In: 12TH EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, 2000. EUROMICRO RTS 2000., p. 231–238, Stockholm, Sweden, 2000.

[20] WANG, S.; LIN, K.-J. ; WANG, Y.. **Hierarchical budget management in the red-linux scheduling framework**. In: PROCEEDINGS OF THE 14TH EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, p. 76–83, 2002.

[21] FOSTER, I.; KESSELMAN, C.; LEE, C.; LINDELL, R.; NAHRSTEDT, K. ; ROY, A.. **A distributed resource management architecture that supports advance reservations and co-allocation**. In: PROCEEDINGS OF THE 7TH INTERNATIONAL WORKSHOP ON QUALITY OF SERVICE, p. 27–36, London, UK, 1999.

[22] KIM, K.; NAHRSTEDT, K.. **A resource broker model with integrated reservation scheme**. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO (ICME2000), volumen 2, p. 859–862, New York, NY, USA, 2000.

[23] YUAN, W.; NAHRSTEDT, K. ; KIM, K.. **R-edf: A reservation-based edf scheduling algorithm for multiple multimedia task classes**. In: RTAS '01: PROCEEDINGS OF THE SEVENTH REAL-TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM (RTAS '01), p. 149, Washington, DC, USA, 2001. IEEE Computer Society.

[24] YUAN, W.; NAHRSTEDT, K. ; GU, X.. **Coordinating energy-aware adaptation of multimedia applications and hardware resource**. In: M3W: PROCEEDINGS OF THE 2001 INTERNATIONAL WORKSHOP ON MULTIMEDIA MIDDLEWARE, p. 60–63, New York, NY, USA, 2001. ACM Press.

[25] YUAN, W.; NAHRSTEDT, K.. **A middleware framework coordinating processor/power resourcemanagement for multimedia applications**. In: IEEE GLOBAL TELECOMMUNICATIONS CONFERENCE (GLOBECOM '01), volumen 3, p. 1984–1988, San Antonio, Texas, USA, 2001.

[26] YUAN, W.; NAHRSTEDT, K.. **Integration of dynamic voltage scaling and soft real-time scheduling for open mobile systems**. In: NOSSDAV '02: PROCEEDINGS OF THE 12TH INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, p. 105–114, New York, NY, USA, 2002. ACM Press.

[27] YUAN, W.; NAHRSTEDT, K.. **Energy-efficient soft real-time cpu scheduling for mobile multimedia systems**. In: SOSP '03: PROCEEDINGS OF THE NINETEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, p. 149–163, New York, NY, USA, 2003. ACM Press.

[28] YUAN, W.; NAHRSTEDT, K.. **Practical voltage scaling for mobile multimedia devices**. In: MULTIMEDIA '04: PROCEEDINGS OF THE 12TH ANNUAL ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA, p. 924–931, New York, NY, USA, 2004. ACM Press.

[29] SMITH, J.; NAIR, R.. **Virtual machines: architecture, implementations and Applications**, chapter An Overview of Virtual Machine Architecture. Morgan Kaufmann Publishers, 2004. Capítulo Disponível em http://www.cis.upenn.edu/ cis700-6/04f/papers/smith-vm-overview.pdf.

[30] FOSTER, I. T.. **The anatomy of the grid: Enabling scalable virtual organizations**. In: EURO-PAR '01: PROCEEDINGS OF THE 7TH INTERNATIONAL EURO-PAR CONFERENCE MANCHESTER ON PARALLEL PROCESSING, p. 1–4, London, UK, 2001. Springer-Verlag.

[31] ADABALA, S.; CHADHA, V.; CHAWLA, P.; FIGUEIREDO, R.; FORTES, J.; KRSUL, I.; MATSUNAGA, A.; TSUGAWA, M.; ZHANG, J.; ZHAO, M.; ZHU, L. ; ZHU, X.. **From virtualized resources to virtual computing grids: the in-vigo system**. Future Gener. Comput. Syst., 21(6):896–909, 2005.

[32] BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I. ; WARFIELD, A.. **Xen and the art of virtualization**. In: SOSP '03: PROCEEDINGS OF THE NINETEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, p. 164–177, New York, NY, USA, 2003. ACM Press.

[33] DUDA, K. J.; CHERITON, D. R.. **Borrowed-virtual-time (bvt) scheduling: supporting latency-sensitive threads in a general-purpose scheduler**. In: SOSP '99: PROCEEDINGS OF THE SEVENTEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, p. 261–276, New York, NY, USA, 1999. ACM Press.

[34] RAMAKRISHNAN, L.; IRWIN, D.; GRIT, L.; YUMEREFENDI, A.; IAMNITCHI, A. ; CHASE, J.. **Grid allocation and reservation—toward a doctrine of containment: grid hosting with adaptive resource control**. In: SC '06: PROCEEDINGS OF THE 2006 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, p. 101, New York, NY, USA, 2006. ACM Press.

[35] WOLINSKY, D.; AGRAWAL, A.; BOYKIN, P. O.; DAVIS, J.; GANGULY, A.; PARAMYGIN, V.; SHENG, P. ; FIGUEIREDO, R.. **On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations**. In: IN PROCEEDINGS OF THE FIRST WORKSHOP ON VIRTUALIZATION TECHNOLOGIES IN DISTRIBUTED COMPUTING (VTDC), Tampa, Florida, USA, November 2006.

[36] KEAHEY, K.; FOSTER, I.; FREEMAN, T.; ZHANG, X. ; GALRON, D.. **Virtual workspaces in the grid**. In: EUROPAR 2005, volumen 3648, p. 421–431, Lisbon, Portugal, Lisbon, Portugal 2005. Springer.