Leonardo Kunrath

Gerência de Reservas de Recursos e Capacidades para a Grade

Florianópolis, Fevereiro de 2008

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Leonardo Kunrath

Gerência de Reservas de Recursos e Capacidades para a Grade

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Carlos Becker Westphall

Florianópolis, Fevereiro de 2008

Gerência de Reservas de Recursos e Capacidades para a Grade

Leonardo Kunrath

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração de Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Dr. Mário Antônio Ribeiro Dantas
Coordenador do Programa de Pós-Graduação em Ciência da Computação

Banca Examinadora

_____

Prof. Dr. Carlos Becker Westphall (Orientador)

_____

Prof. Dr. Mário Antônio Ribeiro Dantas, UFSC

_____

Prof. Dr. João Bosco Mangueira Sobral, UFSC

_____

Prof. Dr. Bruno Richard Schulze, LNCC

Agradecimentos

Agradeço primeiramente a Deus, por ser meu guia, minha força e meu maior incentivo. Agradeço também à minha família, que esteve presente em todos os momentos, tanto os de ânimo quanto os de desânimo, me incentivando, apoiando e acreditando em mim.

Agradeço ao meu orientador, Prof. Dr. Carlos Becker Westphall, por todo apoio e ajuda, e por acreditar no meu trabalho. Agradeço também, e de forma especial, ao Fernando Luiz Koch, amigo que foi fundamental para este trabalho, principalmente pelas idéias, por tirar muitas dúvidas, pela ajuda prestada e pelas palavras de incentivo.

Resumo

Este trabalho apresenta um estudo da gerência de reservas de recursos de Grade. Reservas são fundamentais para QoS em Grades. São propostos uma arquitetura onde usuários, gerenciador de reservas e gerenciadores de recursos são entidades separadas e que se relacionam, e um modelo de representação de recursos e reservas para escalonamento de reservas, baseado em empacotamento em fita. São trabalhados quatro requisitos necessários para uma gerência adequada de reservas de recursos: abrangência, flexibilidade, eficiência e utilização dos recursos. A solução proposta é flexível o bastante para suportar reservas imediatas, antecipadas e sob-demanda, e relaxamento. É também abrangente o bastante para suportar diversos recursos, por meio de reservas de um recurso, de múltiplos recursos simultaneamente, e de parte da capacidade de um recurso. São também propostos três algoritmos de escalonamento de reservas.

Abstract

This work presents a research about Grid resource reservation management. Reservations are very important in Grids. In this work, an architecture where Grid users, a reservation manager and resource managers are distinct entities that interact with each other is proposed; and also a model of reservation scheduling algorithms based on strip-packing. This work tries to support four requirements for reservation management: flexibility, efficiency, resource utilization and support of reservations of many kinds of resources. The solution supports reservations of simple and multiple resources, and of part of a resource capability. It is also flexible enough to support immediate, advance and on-demmand reservations, and also laxity. This work also presents three algorithms for reservation scheduling.

Sumário

1 Introdução

Este capítulo apresenta o tema no qual o trabalho está inserido, seus objetivos, escopo, contribuições e organização.

1.1 Apresentação do Tema

Entende-se por "*computação em Grade*" como uma forma de computação distribuída onde computadores, supercomputadores, *clusters*, dispositivos de geração e armazenamento de dados e outros dispositivos computacionais interagem entre si de forma a compartilhar seus *recursos computacionais* e suas capacidades. Isto gera um sistema com muitos recursos compartilhados, possuindo, assim, uma grande capacidade de processamento e armazenamento e muitas funcionalidades.

Uma Grade possui duas funções principais: interligar dispositivos e juntar uma grande capacidade de processamento e/ou armazenamento. Quanto a interligar dispositivos, usa-se estruturas de Grade para que dispositivos compartilhem suas funcionalidades, tal como, por exemplo, determinados bancos de dados, funções de transformação de dados, dispositivos que geram novos dados, e, assim, ofereçam funções complexas envolvendo tais dados. Esse é o caso de Grades que utilizam-se de sensores, que são dispositivos que captam dados do ambiente, tal como umidade, pressão, movimento, imagens digitais, entre outros. É comum que hajam outros dispositivos que oferecem funções de transformação destes dados, de forma que se possa determinar, por exemplo, as condições atmosféricas de um local, ou se alguma pessoa está presente em um recinto.

Quanto a juntar uma grande capacidade de processamento e/ou armazenamento, é possível que se use uma Grade de forma a permitir a execução aplicações de grande porte. Isto é possível de duas formas. A primeira é através do uso de supercomputadores e/ou *clusters*, que são considerados nós da Grade. Desta forma, pode-se utilizar a Grade para acesso a esses dispositivos e envio de aplicações para que executem neles. Uma outra

maneira é através de *Grades de larga escala*. São assim chamadas porque são compostas por uma grande quantidade de dispositivos (dezenas, centenas ou até mesmo milhares), de forma que a capacidade de processamento e/ou armazenamento disponível seja muito grande. Chama-se isso de "*supercomputação distribuída*".

Uma Grade possui uma plataforma de *software*, que costuma ser chamada de "*middleware*". Esta plataforma provê as funcionalidades básicas de interligação entre os computadores, identificação dos recursos disponibilizados, das aplicações e dos usuários da Grade; funções para localizar recursos desejados, e funções de segurança, tal como criptografia, entre outras. "*Aplicação de Grade*" é o nome que se dá a uma aplicação construída sobre um *middleware* de Grade, de forma que pode utilizar os diversos recursos e funcionalidades da mesma.

As estruturas e arquiteturas de Grade são tecnologias ainda em desenvolvimento. Muitas propostas para melhorar diversas características das Grades, sendo elas a inclusão de novas funcionalidades ou então a melhoria das já existentes, estão surgindo à medida que a popularidade das Grades aumenta. Espera-se, com isso, obter Grades que venham a ser tão simples de usar e tão abrangentes e úteis que seja possível "*a Grade*". Entende-se por "*a Grade*" como uma Grade global, disponível em qualquer lugar do mundo e para qualquer indivíduo. *A Grade* permitiria o compartilhamento de quaisquer recursos ou funcionalidades computacionais, disponibilizando os mesmos de forma global.

Um dos desafios a ser alcançados para que se possa ter *a Grade* é a obtenção de um sistema para prover e gerenciar os recursos dos mais diversos tipos de dispositivos presentes na mesma de forma que possam ser utilizados com *qualidade de serviço*. Chama-se de "*qualidade de serviço*" (*Quality of Service* ou *QoS*) a capacidade de garantir características como desempenho, confiabilidade, disponibilidade, segurança, etc. Assim, um sistema de gerência de recursos capaz de disponibilizar um recurso para um determinado usuário da Grade, ou aplicação do mesmo, de forma ininterrupta, por exemplo, pode ser desejável a outro que não é capaz, uma vez que esta característica, em alguns casos, pode ser muito importante.

Quem determina quais são as características desejadas no uso dos recursos (ou "parâmetros de QoS") é o usuário da Grade que deseja utilizar os mesmos. Caso deseje, ele deve ser capaz de escolher quais os nós que possuem recursos com os parâmetros de QoS desejados e utilizá-los. Um problema quanto a isso é o fato de que alguns recursos da Grade podem ser desejados por mais de um usuário ou aplicação. Isso faz com que seja necessário controlar o acesso aos recursos para que sejam bem utilizados e para resolver

conflitos.

O nó que possui um recurso, ou "*provedor do recurso*" é quem gerencia o acesso ao recurso. Assim, parâmetros de QoS são negociados entre o usuário que deseja usar o recurso e o provedor do mesmo, de forma que se estabeleça um acordo (normalmente sob a forma de um "*SLA*"). Acordos são obtidos no final de um processo de negociação entre o usuário e o provedor do recurso. O usuário busca, nesta negociação, obter o(s) recurso(s) com seus parâmetros de QoS desejados; o provedor busca fazer com que os recursos sejam bem utilizados pelos diversos usuários que os requisitam.

Uma das soluções para que os provedores possam disponibilizar recursos de forma a atender diversos parâmetros de qualidade é através de *reservas de recursos*. Reservar um recurso significa torná-lo disponível para o usuário da Grade que o reservou, ou aplicação do mesmo, de forma ininterrupta e livre de competição com outros usuários e aplicações.

É possível a reserva de recursos como um todo (tal como processadores de um *cluster*) ou de capacidades de um recurso, como é o caso de uma quantidade de banda de rede, por exemplo, onde não é a rede toda (o recurso) que é reservada, mas sim uma certa taxa de transferência de dados (parte da capacidade do recurso). Em outras palavras, há *reservas de recursos* e *reservas de capacidades de um recurso*.

Reservas são uma solução viável e importante para a gerência de recursos para *a Grade*. As formas mais rudimentares de uso de reservas na gerência de recursos se dá com reservas fortemente acopladas aos recursos. Um caso, por exemplo, é o de um recurso que é gerenciado por uma fila de utilizadores e uma reserva se dá quando a fila fica parada para que apenas um usuário ou tarefa o use. Por outro lado, um suporte melhor é dado quando as reservas são fracamente acopladas, que é o caso do uso de um *gerenciador de reservas*.

Um gerenciador de reservas tem como função apenas o agendamento das reservas e escalonamento de novas requisições de reserva nas agendas de reservas. Ele pode gerenciar as reservas de um ou mais recursos, controlados por um ou mais gerenciadores de recursos. Não cabe ao gerenciador de reservas o controle de uso dos recursos. Na verdade, um gerenciador de reservas não precisa estar localizado internamente ao gerenciador de recursos; pode estar localizado inclusive em outro nó da Grade.

Um exemplo de gerenciador de reservas presente em um nó da Grade diferente do nó onde estão os recursos é o seguinte: um usuário $\boldsymbol{A}$ quer reservar um dos recursos do usuário $\boldsymbol{B}$, cujo gerenciador de reservas está presente no usuário $\boldsymbol{C}$. Primeiramente $\boldsymbol{A}$

requisita uma reserva do gerenciador em ***C***. De posse da reserva, ***A*** requisita o uso do recurso em ***B***. Antes de permitir o uso, ***B*** consulta ***C*** para confirmar que a reserva existe.

Gerenciamento de reservas é uma das necessidades das Grades para prover uso dos recursos com qualidade de serviço. Porém, como será apresentado no próximo capítulo, as soluções atuais para o gerenciamento de reservas possuem graves restrições, sendo aplicáveis em apenas algumas situações e apenas para alguns tipos de dispositivos ligados à Grade. Isto traz a motivação para a proposta deste trabalho.

1.2 Objetivos e Contribuições

O **objetivo geral** desta dissertação é apresentar um esquema de reservas de recursos e de capacidades de recursos abrangente, que seja aplicável aos mais diversos tipos de dispositivos presentes em Grades. A idéia é contribuir com idéias, abstrações e algoritmos para o desenvolvimento de gerenciadores de reservas para *a Grade*.

Como **objetivos específicos**, visa-se fazer com que os gerenciadores de reservas:

- ofereçam suporte a **reservas flexíveis**, ou seja, sejam capazes de reservar recursos com *reservas imediatas*, *antecipadas* ou *sob-demanda*, e com limites de tempo flexíveis.
- sejam capazes de fazer uma **boa utilização** dos recursos e capacidades, ou seja, utilizá-los da melhor forma possível. Uma boa utilização significa que mais usuários e aplicações conseguem utilizar os recursos, e que eles ficam menos tempo subutilizados.
- possuam algoritmos de gerência com **baixa complexidade** algorítmica. Isso faz com que o tempo de resposta de um gerenciador de reservas para agendá-las seja baixo, e que o mesmo suporte uma grande quantidade de reservas (seja escalável).

As principais **contribuições** deste trabalho são as propostas de modelagem de reservas e de algoritmos de escalonamento de reservas para suprir os requisitos de utilização, eficiência, flexibilidade e abrangência; e a implementação de algoritmos para escalonamento de diferentes tipos de reserva. As reservas são modeladas de forma homogênea, dando suporte a diversos tipos de recursos.

1.3 Escopo e Limitações do Trabalho

A dissertação aborda, de forma mais detalhada, os seguintes temas:

- **recursos de Grade**: principais tipos e particularidades, e quais tipos de reservas suportam.
- **reservas**: tipos e atributos, e onde se aplicam;
- **gerenciamento de reservas**: abstrações para representar recursos, capacidades e reservas, e algoritmos para agendar e escalonar reservas.

Por questão de **limitação de escopo**, alguns assuntos menos relevantes aos objetivos do trabalho, porém também importantes ao gerenciamento de recursos de Grade, **não** serão abordados neste trabalho. Entre eles, podemos citar:

- **descoberta de recursos**: como os recursos são disponibilizados/anunciados na Grade e como são encontrados por usuários e aplicações;
- **políticas de uso de recursos**: formas de representar particularidades/restrições do uso de certos recursos, e quais ações podem ser tomadas por quem usa e quem disponibiliza tais recursos.
- **acordos/SLAs**: não são abordadas formas de representá-los, nem formas de garantir que sejam cumpridos conforme estabelecidos, e nem as ações que podem ou devem ser tomadas em casos de violação dos acordos.
- **co-alocação de recursos**: arquiteturas, algoritmos e entidades para obter reservas simultâneas de diferentes recursos pertencentes a diferentes domínios administrativos. Co-alocação costuma ser elaborada como uma aplicação/serviço cuja finalidade é utilizar reservas de recursos individuais, de forma sincronizada. Para isso, são necessárias *co-reservas*, ou seja, reservas coordenadas de recursos heterogêneos de diferentes domínios. Não será trabalhada neste trabalho por ser utilizada em apenas algumas situações e por poder ser implementada como um serviço de alto nível que utilizará um conjunto de reservas simples.
- **escalonamento dinâmico**: este trabalho apresenta uma proposta onde o gerenciamento de tarefas e o gerenciamento de reservas de recursos é feito em módulos ou entidades separadas. Desta forma, recursos são reservados para usuários, e o escalonador de tarefas irá escalonar as mesmas para utilização dos recursos reservados.

Em outras palavras, utiliza-se uma abordagem estática, onde se reserva recursos para usuários, e não para aplicações.

- **escalonamento de tarefas**: após a reserva dos recursos necessários, um gerenciador de tarefas toma decisões sobre quais tarefas utilizarão cada recurso e de que forma farão isto. Estas decisões estão fora do escopo do trabalho, que tem como preocupação a reserva adequada dos recursos, independentemente das tarefas que os utilizarão.

1.4 Organização do Trabalho

Este trabalho está organizado de forma:

- Capítulo 2: **Requisitos e Trabalhos Relacionados**. Apresenta os principais requisitos que devem ser cumpridos pelos gerenciadores de reservas e os principais trabalhos relacionados à gerência de reservas de recursos de Grade.
- Capítulo 3: **Gerência de Reservas**. Neste capítulo se dão as principais contribuições teóricas. São apresentadas formas de modelar e reservas e algoritmos de escalonamento de reservas para atender aos requisitos.
- Capítulo 4: **Escalonamento de Reservas**. Apresenta os algoritmos de escalonamento de reservas implementados.
- Capítulo 5: **Resultados Experimentais**. Apresenta estudos de casos e análise do uso dos algoritmos desenvolvidos.
- **Conclusão**. Revisa os principais progressos atingidos ao longo do trabalho e destaca as contribuições.

2 *Análise Teórica e Trabalhos Relacionados*

Este capítulo apresenta o problema trabalhado e analisa a importância da gerência de reservas e quais são dos requisitos necessários para bem gerenciá-las, e também discute sobre os trabalhos relacionados à gerência de reservas.

2.1 Análise do Problema

A pergunta que este trabalho tenta responder é a seguinte: "*Quais são os requisitos e como deve ser modelado um gerenciador de reservas de recursos de Grade?*". O primeiro passo para responder esta pergunta é verificar como se dá o uso de reservas de recursos. Para isso, é apresentado um cenário de uso de recursos de Grade.

2.1.1 Cenário

Na figura 2.1, é apresentado um cenário que exemplifica a questão de gerência de recursos. Um determinado nó da Grade, no caso um *cluster*, compartilha os seguintes recursos na Grade: 16 processadores idênticos, 4 discos rígidos e banda de rede capaz de transferir 800 Kilobytes por segundo de dados. Ao mesmo tempo, em outros nós da Grade, diferentes usuários desejam enviar aplicações diversas para utilizar os recursos disponíveis de acordo com suas necessidades.

Este cenário exemplifica alguns dos problemas com que um sistema de gerenciamento de recursos para Grade deve ser capaz de lidar. Entre eles pode-se citar:

- **Diferentes usuários simultâneos** - é o caso de quando diferentes usuários da Grade desejam utilizar os recursos de um determinado nó ao mesmo tempo. O sistema de gerenciamento de recursos deve saber ponderar isso, de forma a que os diferentes usuários possam ter a chance de utilizar os recursos.

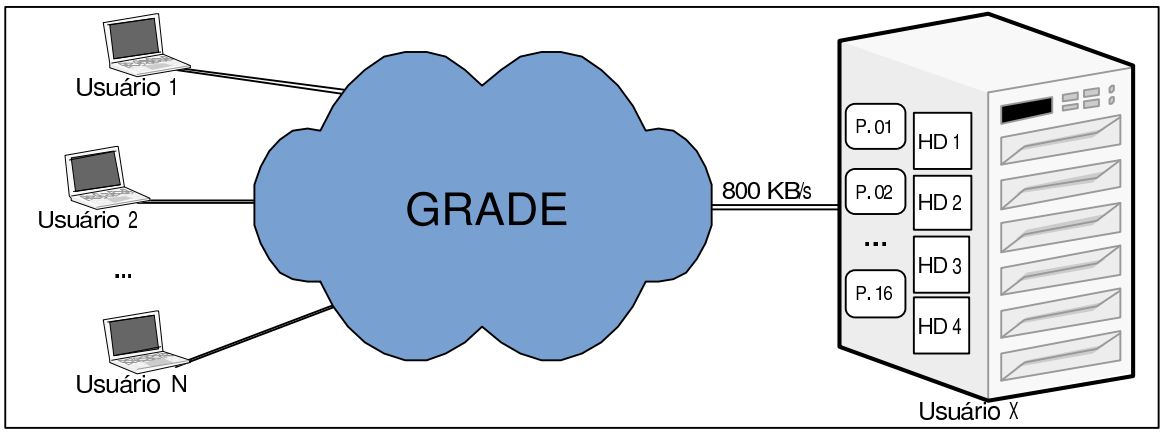

Figura 2.1: Cenário: compartilhamento de recursos em uma Grade.

- **Muitos recursos** - o sistema de gerenciamento de recursos deve ser capaz de gerenciar os diferentes recursos que o nó oferece, de forma a lidar com as diferenças dos mesmos.

- **Muitas tarefas** - uma vez que os usuários vão enviando tarefas para utilização dos recursos, o gerenciador de recursos deve ser capaz de lidar com as mesmas de forma organizada, alternando a posse dos recursos entre elas de forma a estabelecer um compartilhamento mais justo dos mesmos.

Existem duas formas principais de gerenciar recursos de Grade: utilizando reservas ou não utilizando.

Gerência de Recursos sem Reservas

Neste caso, o gerenciador de recursos se baseia na idéia de tarefas ou serviços utilizando os recursos. Pode-se, assim, dividir os recursos em dois grupos: processadores e demais recursos.

O papel dos processadores é a execução das tarefas. O gerenciador de recursos lida com isso desempenhando o papel de escalonador dinâmico de tarefas, de forma a organizar as mesmas para execução simultânea, por meio de filas.

Outros recursos além de processadores são utilizados pelas tarefas em execução. Por exemplo, uma tarefa pode usar acesso a disco, conexões de rede ou ainda outros dispositivos. O gerenciador de recursos deve oferecer ferramentas para permitir acesso simultâneo, mas também acesso isolado a certos recursos, tal como por meio de travas, para melhor utilização dos mesmos.

Existem alguns dos problemas neste tipo de gerenciamento de recursos. Um deles é a subutilização dos recursos. Isso pode ocorrer caso uma tarefa utilize uma trava em um recurso sem necessidade. Outro problema é a questão de competição: diferentes tarefas podem disputar por recursos como rede, acesso a disco ou ainda outros; o que pode atrasar tarefas, na medida em que elas precisam esperar que os recursos desejados sejam liberados. Outro problema ainda é o caso de aplicações paralelas, que geram várias *threads* ou processos, e assim atrasam as demais aplicações.

Gerência de Recursos com Reservas

Neste caso, os recursos não são utilizados diretamente por tarefas. Um usuário que deseja usar um recurso deve passar por uma etapa de negociação com o usuário servidor do recurso, de forma a obter uma reserva do recurso.

Obter uma reserva significa obter uma garantia de uso exclusivo, em um intervalo finito e determinado de tempo, de um recurso, de certa proporção da capacidade de um recurso, ou de um conjunto de recursos.

Um exemplo que permite uma melhor visualização disso, utilizando o cenário apresentado, pode ser: um usuário deseja processadores para uma aplicação paralela que utiliza quatro processos. Ele pode reservar quatro processadores por um determinado período de tempo, tal como das 10 horas às 15 horas. Neste período ele poderia enviar os processos para os processadores e executá-los de forma exclusiva.

Outro exemplo: outro usuário, interessado em executar uma aplicação que necessita de diferentes recursos, poderia requisitar uma reserva de um processador das 14 horas às 20 horas, outra de um disco rígido no mesmo período, outras duas reservas de 200 Kilobytes por segundo de banda de rede, uma das 14 horas às 16 horas e outra das 18 às 20 horas.

Como se pode perceber, com reservas obtém-se maior controle no uso dos recursos. Diferentemente da estratégia anterior, onde um usuário que quisesse executar uma tarefa a enviava e ela competia com as demais, com reservas um usuário que queira utilizar um recurso deve reservar o mesmo e esperar pelo momento de uso, porém não haverá competição na hora de utilizá-lo. Além disso há garantia da disponibilidade dos recursos, o que evita atrasos. Outro fator ainda que torna fundamental o uso de reservas em Grades é a questão de reservas simultâneas.

A utilização de reservas para controlar o uso dos recursos gera a necessidade de se ter

um gerenciador de reservas. Este gerenciador pode ser interno ao gerenciador de recursos ou externo. Se for externo, ele pode até estar situado em outro nó da Grade, e servirá de entedidade intermediária entre os usuários interessados e o gerenciador de recursos. O gerenciador de reservas possui duas funções principais: *negociação* e *escalonamento*.

A negociação se dá através de troca de mensagens entre os usuários interessados e gerenciador de reservas. Uma negociação começa quando um usuário interessado envia ao gerenciador um pedido de reserva de determinados recursos, por determinado período de tempo. Normalmente, seguem-se outras trocas de mensagens a fim de resolver questões de políticas de acesso, procedimentos tomados em caso de faltas ou em casos de violações, questões de segurança, entre outros. Por fim, o gerenciador verifica a possibilidade de agendar a reserva dos recursos no tempo requerido, e o faz caso seja possível, retornando para o usuário interessado a resposta. A negociação então termina com as decisões tomadas registradas em um documento de acordo.

O escalonamento é o núcleo da gerência de reservas. Ele acontece internamente à negociação, escolhendo os recursos e o período mais adequado dentre os possíveis para que a reserva aconteça. Dele depende o sucesso da gerência de reservas, pois um escalonador que não utilize bem os recursos, não suporte muitas reservas ou que seja pouco eficiente pode tornar o uso dos recursos ainda pior do que o caso sem reservas.

2.1.2 Requisitos Fundamentais

As quatro principais vantagens para gerência de recursos que podem ser obtidas por meio de reservas, e que são requisitos fundamentais de um gerenciador de reservas, são:

- **Uso exclusivo** - quando um usuário reserva recursos, aqueles recursos podem ser utilizados apenas por ele durante o período da reserva, independentemente de haver ou não outros usuários interessados nos recursos. Isso permite o uso ao máximo dos recursos agendados.

- **Uso ininterrupto** - em uma reserva, os recursos são utilizados sem que haja nenhum tipo de interrupção ou preempção. Em outras palavras, cada reserva se dá em um certo período contínuo de tempo. Isso traz a garantia de que as aplicações não serão interrompidas e de que a capacidade computacional reservada não se altera do início ao fim da reserva.

- **Previsibilidade** - como a capacidade computacional reservada é constante, é possível prever o tempo que levará para executar determinada tarefa, tal como executar

uma aplicação, ou transferir uma certa quantidade de dados, ou o número de tarefas que podem ser cumpridas por unidade de tempo.

- **Coordenação** - o usuário pode reservar diferentes recursos de forma simultânea. Assim, é possível executar aplicações que usem vários recursos, garantindo que eles estarão disponíveis.

2.1.3 Requisitos Desejados

Além dos requisitos citados anteriormente, existem alguns outros que são opcionais, mas cuja presença é necessária para que o gerenciador de reservas possa desempenhar bem o seu papel. São eles:

- **Eficiência** - é preciso que a complexidade do algoritmo de escalonamento seja baixa, ou seja, o tempo de resposta do algoritmo não deve ser muito grande, mesmo com um grande número de recursos gerenciados e de reservas já agendadas. Isso significa que, quando um usuário interessado for negociar para reservar um recurso, ele deve receber uma resposta dentro de um tempo curto. Em outras palavras, o escalonamento não deve ser uma operação que gere grandes atrasos.

- **Utilização** - é preciso que o gerenciador de reservas preze pela máxima utilização dos recursos. A utilização em um determinado momento pode ser medida como a capacidade utilizada em relação à capacidade total. Para obter boa utilização, deve-se ser capaz de tomar decisões adequadas quanto aos melhores recursos e ao melhor período no qual realizar cada reserva.

- **Flexibilidade** - o gerenciador deve ser capaz de fazer o agendamento de reservas para uso imediato, antecipado ou na medida que os recursos estiverem disponíveis. Além disso, deve permitir que sejam estabelecidos limites de tempo de início e término das reservas rígidos ou flexíveis, de acordo com a preferência do usuário requisitante.

- **Abrangência** - o escalonador deve ser capaz de prover suporte a reservas de diferentes tipos recursos, utilizando tipos de reservas adequados para cada tipo de recurso.

O sucesso de um gerenciador de reservas irá depender de cumprir estes requisitos. Caso ele os deixe de lado eficiência, não conseguirá atender casos onde há grande competição pelos recursos ou então gerará atrasos que prejudicarão a execução das aplicações

da Grade. Deixar de lado a questão de utilização, através do uso de algoritmos de escalonamento que não procurem maximizar o tempo em que os recursos estarão sendo utilizados, acabará também prejudicando a Grade como um todo, devido ao disperdício da capacidade dos recursos.

Gerenciadores inflexíveis e não abrangentes não apenas prejudicam a Grade, mas tornam inviável o uso de reservas de recursos. A inflexibilidade reduz a chance de se conseguir um recurso. Por exemplo, caso um usuário tenha uma preferência de horário para uso de um recurso, ao requisitar uma reserva do mesmo é possível que este recurso já tenha sido reservado por outro usuário. Quanto à abrangência, o suporte a reservas de apenas um único tipo de recurso, tal como processador, ou banda de rede, acaba tornando o gerenciador útil em apenas alguns casos isolados dentro da Grade. Soluções mais abrangentes, com reservas de vários tipos de recursos, permitem uso coordenado dos mesmos, trazendo maior qualidade de serviço.

A seguir, serão apresentados alguns tipos de reservas que devem ser suportados para que se tenha maior flexibilidade e abrangência.

2.1.4 Tipos de Reservas para maior Flexibilidade

Para maior flexibilidade, podem ser distingüidos três tipos de reservas. O que as diferencia é a questão temporal. São elas *reservas imediatas*, *reservas antecipadas* e *reservas sob-demanda*.

Reservas Imediatas

Reservas imediatas são aquelas onde se deseja utilizar os recursos imediatamente, no momento em que se são requisitadas. Caso o recurso requisitado estiver ocupado, a reserva não se concretiza.

Um gerenciamento de recursos que suporte apenas reservas imediatas se comporta como um sistema sem reservas, com apenas algumas pequenas modificações. Isso quer dizer que é um sistema baseado em tarefas.

Uma reserva imediata de um processador, em um sistema assim, aconteceria da seguinte forma: há várias tarefas em execução no mesmo processador, e uma fila é utilizada para organizá-las. As regras da fila são simples: uma tarefa está em funcionamento por vez e, depois de um certo tempo, ela é interrompida e recolocada no fim da fila, dando lugar para a próxima. Uma reserva imediata pressupõe uma tarefa que não pode ser interrom-

pida. Quando uma destas reservas é colocada na fila, as outras aplicações devem esperar ela ser concluída para que a fila siga adiante. O escalonador de tarefas pode estabelecer um limite máximo para a duração da reserva. Isso impede que as demais aplicações de Grade fiquem paradas por muito tempo.

Na figura 2.2, é apresentado um exemplo de gerência de tarefas de Grade por meio de fila. O exemplo utiliza uma fila que segue a ordem de chegada e um esquema onde cada tarefa que estiver executando (a mais à direita, na figura), após ser interrompida, vai para o fim da fila. Em (a), há apenas tarefas de Grade comuns (Ax). Em (b), há tarefas de Grade comuns (Ax) e tarefas sob forma de reserva imediata de processador (Rx). Como as reservas imediatas normalmente possuem maior prioridade que as demais aplicações de Grade, elas são colocadas à frente das outras aplicações, no começo da fila, porém atrás de outras reservas já feitas. Como o mecanismo de revezamento para uso do recurso não é utilizado em reservas, por elas executarem do início ao fim sem serem interrompidas, a fila ficará parada até que a reserva em posse do recurso termine sua execução.

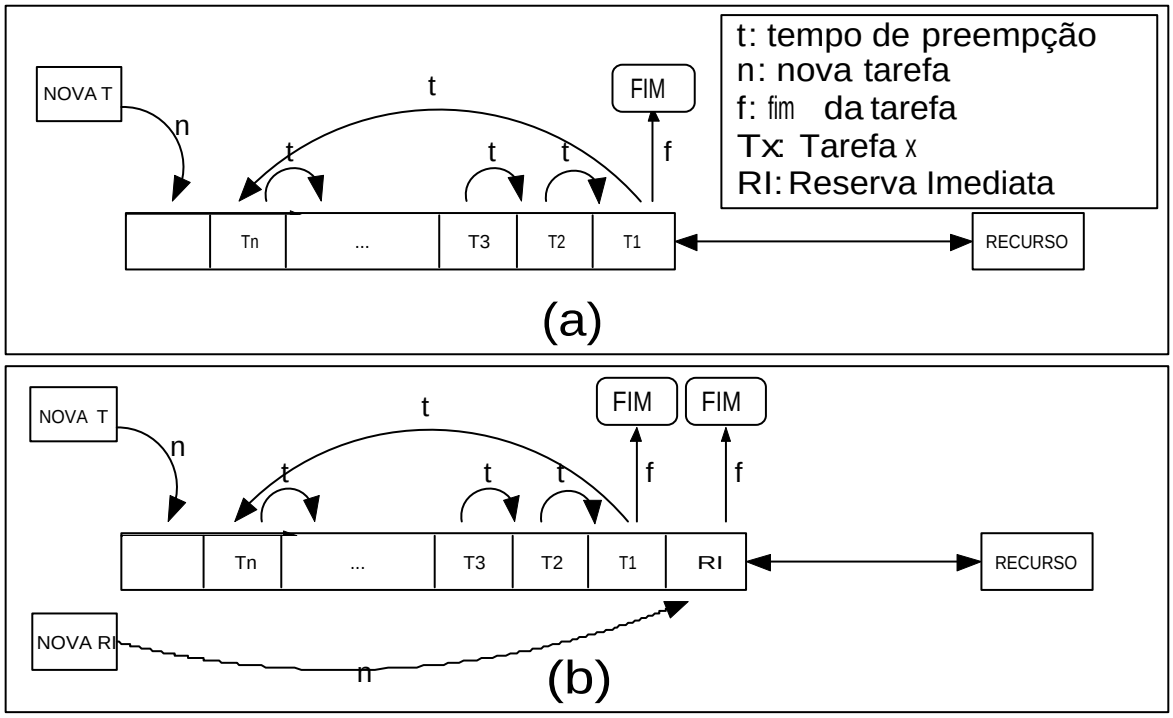

Figura 2.2: Escalonamento de Tarefas e Reserva Imediata.

Reservas Antecipadas

É possível que se negocie a reserva de um recurso muito antes de seu uso. Assim, negocia-se para que uma tarefa de um usuário da Grade inicie e execute de forma exclusiva

a partir de um instante de tempo que é definido na negociação. Por exemplo, é possível reservar um recurso para ser utilizado "a partir das oito horas da manhã".

Uma reserva antecipada possui, assim, dois campos importantes, definidos na negociação entre usuário e provedor do recurso: a duração da reserva e o tempo inicial da mesma. Com algoritmos adequados, é possível usar estes campos de forma a permitir diversas reservas antecipadas em um mesmo nó da Grade. O fator primordial é que elas sejam temporalmente exclusivas, ou seja, duas ou mais reservas não devem estar agendadas para o mesmo período de tempo.

Uma característica importante das reservas antecipadas é o fato de que o usuário que reserva o recurso não o usa imediatamente. Isso gera duas implicações: a primeira, é o fato de que o usuário deve receber uma confirmação de que a reserva está agendada, e uma senha ou outro tipo de identificação para que possa acessar mais tarde o recurso e utilizá-lo. A segunda implicação é que há um grande período de tempo entre a negociação e o uso do recurso. Isso faz com que seja possível, para o usuário, renegociar os termos da reserva ou até mesmo cancelá-la; e para o provedor do recurso, tratar problemas com os recursos, tal como ativar réplicas do recurso caso ele venha a falhar, ou avisar o usuário interessado de eventuais problemas, renegociando termos da reserva com ele.

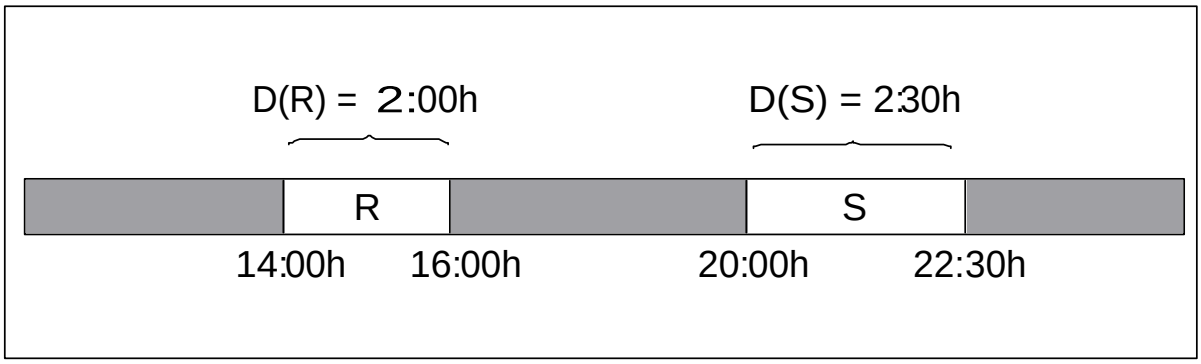

Figura 2.3: Duas reservas antecipadas R e S em uma agenda temporal.

As reservas antecipadas não são colocadas em uma fila normal, mas em uma agenda temporal. A fila com aplicações que não são reservas continua a existir. A figura 2.3 apresenta uma agenda temporal com duas reservas antecipadas, R e S. Estas reservas podem ou não pertencer ao mesmo usuário.

Um sistema de gerenciamento de reservas sem reservas antecipadas é praticamente inviável. Além das características de aumento de QoS pela previsão de início da reserva e melhor utilização dos recursos, já que o provedor pode agendar as reservas de forma organizada, reservas antecipadas também são fundamentais para o uso coordenado de

recursos.

Reservas Sob-Demanda

Este tipo de reserva é utilizado quando um usuário da Grade necessita reservar um recurso para uso imediato. Porém, caso o recurso esteja ocupado, ele busca uma reserva antecipada para o primeiro período possível de disponibilidade do recurso. É possível perceber que, por natureza, este tipo de reserva é flexível, porque permite que o intervalo de reserva do recurso possa ser determinado não na requisição, mas de acordo com as outras reservas já agendadas.

Ao suportar reservas sob-demanda, o gerenciador de reservas garante que será capaz de verificar as reservas agendadas e encontrar o primeiro período de disponibilidade. Além disso, é preciso que, no momento em que a reserva seja concluída, o gerenciador envie uma resposta ao usuário interessado de qual será o intervalo de tempo da reserva, para ele saber quando será a hora certa de utilizar o recurso.

2.1.5 Tipos de Reservas para Maior Abrangência

Enquanto, para prover reservas flexíveis, foram destacados diferentes tipos de reservas quanto ao fator temporal, para a questão de abrangência é necessário suportar diferentes tipos de reserva quanto à capacidade reservada. Por este critério, uma reserva pode ser de um dos três seguintes tipos: *reserva simples*, *reserva múltipla* ou *reserva parcial*.

Reserva Simples

Neste tipo de reserva, apenas um recurso é considerado, mesmo que o nó da Grade possua vários. Uma característica importante deste tipo de reserva é que o recurso só possui dois estados possíveis em um determinado instante de tempo: reservado ou disponível. Não se considera a possibilidade de haver mais do que uma reserva ao mesmo tempo, nem de se reservar parte do recurso. Se um processador for reservado, por exemplo, apenas o usuário que o reservou poderá utilizá-lo no intervalo referente à reserva.

Este tipo de reserva é utilizado quando se deseja reservar um único recurso por completo, independentemente de ele poder ser utilizado por mais do que um usuário ao mesmo tempo. Por exemplo, caso se deseje reservar toda a capacidade de rede para um usuário, pode-se utilizar este tipo de reserva. Porém, como a rede pode ser utilizada por mais do

que um usuário ao mesmo tempo, seria mais adequado o uso de outro tipo de reserva para a rede.

As figuras 2.2 e 2.3 exemplificam reservas de um único recurso. É possível visualizar, pelas figuras, que o recurso pode estar, em um determinado momento, totalmente reservado ou então totalmente disponível para novas reservas.

Reserva Múltipla

Consideremos o exemplo do cenário apresentado neste capítulo. Há dezesseis processadores idênticos localizados no mesmo nó e sendo compartilhados na Grade. Suas reservas são controladas pelo mesmo gerenciador de reservas. Suponha-se agora que um usuário da Grade deseje reservar oito processadores para uma aplicação paralela. Caso apenas reservas simples fossem suportadas, ele teria que fazer oito reservas, tentando coordenar para que acontecessem ao mesmo tempo. Pode-se perceber que seria um desperdício de tempo e um esforço adicional muito grande.

Entende-se por *reserva múltipla* como uma única reserva de vários recursos no mesmo intervalo de tempo. Na suposição feita, ao invés de "oito reservas de um processador", tentando coordená-las para um mesmo intervalo, seria feita "uma única reserva de oito processadores". Isso não apenas facilita o uso de aplicações paralelas, como também aumenta consideravelmente a eficiência do escalonamento de reservas, pois uma quantidade menor de reservas é necessária.

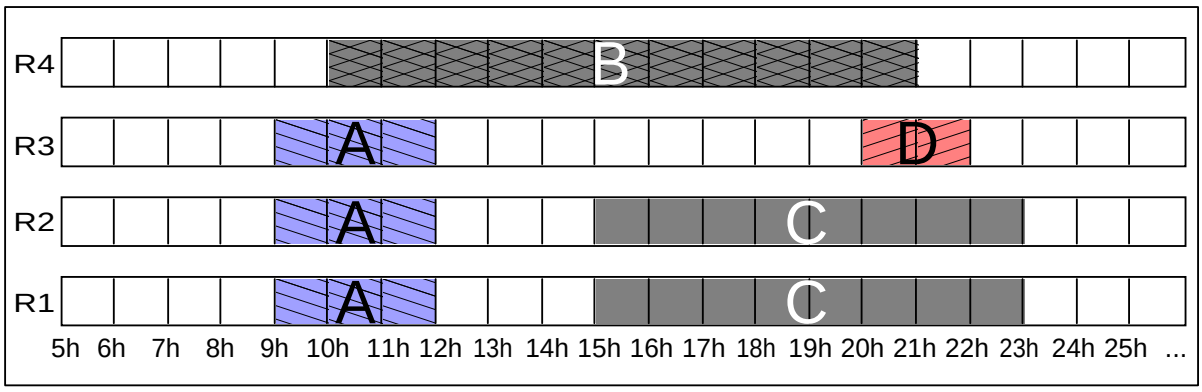

Figura 2.4: Exemplo de reservas múltiplas em quatro recursos (A e C).

A figura 2.4 apresenta um exemplo com duas reservas múltiplas (A e C) e duas reservas simples (B e D). A é uma reserva de três recursos por três horas, e C é uma reserva de dois recursos por oito horas. É possível perceber pelo exemplo que reservas simples e reservas múltiplas podem co-existir para o mesmo conjunto de recursos. Na verdade, reservas

simples podem ser utilizadas como "reservas múltiplas de um único recurso", de forma que um sistema de gerenciamento de reservas que suporte reservas múltiplas suportará também reservas simples.

Reserva Parcial

Este tipo de reserva difere-se das reservas simples e múltiplas por uma característica importante: não é o recurso que será reservado, mas sim parte de sua capacidade.

Um dos exemplos deste tipo de reserva é dado no cenário apresentado. Ele apresenta um nó que possui banda de rede capaz de transferir 800 KB/s de dados. O recurso é a rede. Sua capacidade é a taxa de transferência. Caso fosse reservada uma taxa de transferência de 200 KB/s, não seria toda a rede reservada, mas sim apenas uma parte de sua capacidade. Em outras palavras, uma *reserva parcial* é uma reserva de parte da capacidade de um recurso.

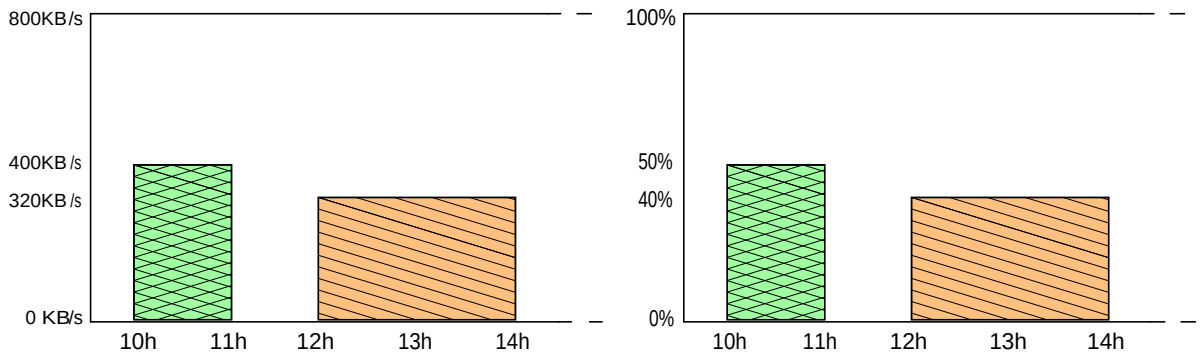

Figura 2.5: Representações de reservas parciais.

A figura 2.5 apresenta um exemplo com duas reservas: uma de 400 KB/s por uma hora, e outra de 320 KB/s por duas horas, sendo que a capacidade total de transferência de dados é de 800 KB/s. Há duas representações para as mesmas reservas: na esquerda, as reservas estão representadas pela unidade de capacidade (no caso, KB/s); na direita, estão representadas por porcentagem da capacidade total, uma vez que 800 KB/s é 100%, 400 KB/s é 50% e 320 KB/s é 40% da capacidade total. Esta representação da direita é mais genérica pois pode ser usada para outros tipos de recursos. Assim, ao invés de ter que lidar com diversas unidades de capacidade (para disco rígido, por exemplo, seria em *bytes*), basta que o gerenciador de reservas saiba lidar com porcentagem da capacidade total, e haja um simples conversor (de unidade para porcentagem) para cada tipo de recurso.

2.2 Trabalhos Relacionados

2.2.1 GARA

Este é o primeiro trabalho que aborda reservas antecipadas de recursos para Grades. Ele apresenta o *GARA* (*Globus Architecture for Reservation and Allocation*, ou seja, arquitetura para reserva e alocação de recursos para o *Globus*). O Globus é um *middleware* de Grade muito utilizado (provavelmente o mais utilizado), e que adota diversos padrões e é genérico o suficiente para que se possa construir, através dele, Grades e aplicações de Grade.

GARA foi construído para a segunda versão do Globus (BRUNETT et al., 1998), como uma extensão do *GRAM* (*Grid Resources Allocation Manager* (CZAJKOWSKI et al., 1998), o serviço de gerência de recursos do Globus. A partir da terceira versão do Globus, que evoluiu para atender à nova arquitetura de Grades *OGSA* (*Open Grid Services Architecture*) (FOSTER et al., 2002), baseados em *Web Services*, o GARA passou a ser usado através de interfaces para acesso a componentes do antigo Globus 2.

No GARA, os recursos são apresentados como *objetos* genéricos. Assim, banda de rede, processadores, arquivos, acesso a bancos de dados, e todos os demais tipos de recursos seriam representados pelo mesmo tipo de objeto. Esta visão dos recursos permitia que se desenvolvesse interfaces homogêneas para publicação, escolha e acesso dos mesmos, porém tornava complicado o próximo passo, na hora do uso, onde as diferenças entre os recursos vinham à tona.

Para reservas, este modelo homogêneo simplificou muito a implementação de reservas. Porém, todas as reservas eram feitas através de *timeslot tables* ou simplesmente *slot tables*, que são trabalhados através de gerenciadores chamados de *timeslot managers* (FERRARI; GUPTA; VENTRE, 1995), (FERRARI; GUPTA; VENTRE, 1997) e (HOO; JOHNSTON, 1999). *Slot tables* são estruturas bidimensionais que levam em conta a período de tempo da reserva e a porcentagem do recurso que será utilizada. Em outras palavras, o GARA suporta reserva parcial de recursos. Porém, infelizmente, reservas simples e reservas múltiplas não são suportadas, o que reduz muito a eficiência e abrangência do gerenciador, pois, para recursos como processadores, onde seria o ideal reservas múltiplas, são realizadas diversas reservas parciais, onde a capacidade é o poder de processamento (ciclos) dos processadores.

2.2.2 Reservas em Transferências de Dados

Diversos trabalhos (BURCHARD; HEISS; ROSE, 2003), (BURCHARD, 2003), (BURCHARD; DROSTE-FRANKE, 2003), (BURCHARD et al., 2004), (BURCHARD; LINNERT, 2004), (BURCHARD, 2004), (BURCHARD, 2005a), (BURCHARD et al., 2005), (BURCHARD; LINNERT; SCHNEIDER, 2005), (BURCHARD; SCHNEIDER; LINNERT, 2005), (BURCHARD et al., 2005) e (BURCHARD, 2005b) abordam dois problemas principais. O primeiro é a reserva de taxa de transmissão de dados em redes locais de alto desempenho. O segundo é a questão de resolver faltas ocorridas em recursos de forma a poder cumprir as reservas já agendadas.

Para identificar cada reserva, estabeleceu-se um conjunto de atributos que definem a capacidade de transferência de dados, o período de tempo da reserva e quais os dois nós da rede que trocarão os dados. Um gerenciador de reservas de uma certa rede local poderia, assim, estabelecer diversas reservas simultâneas para transferir dados passando os mesmos por uma rota que inclui diversos nós da rede.

O gerenciador utiliza apenas reservas parciais, através de um algoritmo bidimensional, como em (FOSTER et al., 1999). Foi também apresentado uma nova forma de reserva parcial, chamada de reserva maleável (*malleable reservation*). Esta reserva permite que a capacidade e a duração da reserva possam ser definidas livremente pelo gerenciador de reservas. Assim, ao invés de requisitar uma certa capacidade de transferência por uma certa duração de tempo, requisita-se a transferência de uma certa quantidade de dados. O gerenciador então escolherá os valores de duração e capacidade; menor duração implica em maior capacidade usada e vice-versa.

Para resolver problemas ocorridos por causa de faltas, é apresentada da seguinte solução: por suportar reservas antecipadas, há um tempo entre a negociação pela reserva e o momento de utilização do recurso. Neste intervalo, caso ocorra alguma falta, busca-se uma nova rota, que é informada ao usuário que requisitou a reserva.

Um dos principais problemas destes trabalhos é que são suportadas apenas reservas parciais de capacidade de transferência de dados em redes. Outros tipos de reservas e outros recursos não são considerados.

2.2.3 Gerenciadores Mediadores entre Grades e Clusters

Diversos trabalhos abordaram a questão de criar interfaces e gerenciadores capazes de interagir com *clusters* e *supercomputadores*, fazendo com que seus processadores possam ser reservados por usuários de Grade. A idéia é servir de ponte entre a Grade e estes dispositivos. Estes gerenciadores de reservas têm como características: são intermediários entre a Grade e os *clusters*; e são responsáveis pelo escalonamento das reservas, construída como uma camada de alto nível sobre os gerenciadores de recursos locais dos *clusters*.

(SMITH; FOSTER; TAYLOR, 2000) aborda a questão de reservas múltiplas de processadores utilizando-se de gerenciadores de recursos projetados para supercomputadores. A idéia é reservar processadores de supercomputadores através de uma integração do *middleware* de Grade com o gerenciador de recursos local (*Local Resource Manager*). Assim, um usuário pode requisitar uma reserva de um certo número de processadores para serem usados simultaneamente em um período de tempo pré-definido; então a requisição é repassada ao gerenciador de recursos local, que cuidará dos detalhes. Além disso, ele apresenta um estudo de algoritmos para aplicações que devem ser executadas completamente de uma vez e para aplicações que podem ser paradas e reiniciadas posteriormente. Foram utilizadas filas, uma para cada processador, e através delas diferentes técnicas foram combinadas para as requisições. Foi assumido, também, que não se sabe ao certo quando uma aplicação irá terminar, e então é preciso que se use algoritmos para prever isso. Foi também utilizado *backfilling*, ou seja, aplicações podem ser reservadas fora da ordem de chegada, desde que não sejam atrasadas as aplicações que já estão na fila. Assumiu-se que as aplicações reservadas podem ter diferentes prioridades. Os resultados mostraram que aplicações que podem ser reiniciadas permitem melhor utilização dos recursos. Os principais problemas deste trabalho são a falta de suporte de outros recursos além de processadores e a falta de suporte de reservas parciais.

(ELMROTH; TORDSSON, 2004) expõe a idéia de um esquema onde os usuários indicam quais são os seus *benchmarks* desejados, ou seja, quais são os parâmetros de qualidade de serviço exigidos. Com isso, o gerenciador irá escolher os recursos necessários e agendar as reservas. O gerenciador de recursos possui duas funções principais: escolher os melhores recursos para as aplicações de Grade e agendar reservas. Elas são agendadas através de um mecanismo que interage com o gerenciador de recursos local e possui duas operações: requisitar reservas e liberar reservas. Cada reserva é definida com os valores de número de processadores e duração da reserva, ou seja, são suportadas reservas múltiplas. Um terceiro atributo, que é uma conta à qual será atribuída a reserva, também pode ser

enviada na requisição de reserva. O gerenciador, então, cria a reserva, acopla a ela o identificador do usuário beneficiado, e a envia para gerenciador local. Quando termina o tempo da requisição, o gerenciador envia um comando ao gerenciador local para liberar os recursos. Reserva de capacidades e outros recursos além de processadores não são suportados.

(ROEBLITZ; SCHINTKE; WENDLER, 2004) também apresenta o gerenciamento de reservas de processadores por meio de reservas antecipadas. É proposto um tipo de reserva chamada de *reserva elástica*. Ela possui como atributos os limites de tempo para começar e para terminar, e número máximo e mínimo de processadores. Além dos limites, são definidos: duração referencial, poder de processamento referencial e número de processadores referencial. A função do gerenciador de reservas é agendar cada reserva em um período adequado e com um número adequado de processadores. É possível que haja processadores com diferentes desempenhos, então o algoritmo não busca a solução ideal, por ser um problema com grande complexidade algorítmica e inviável. É utilizado um algoritmo que gera algumas possíveis soluções e, entre elas, a mais viável, e que se adequa melhor aos valores referenciais, é escolhida. Embora reservas múltiplas sejam suportadas, o mesmo não acontece com outros tipos de reservas e outros recursos além de processadores.

Três trabalhos, (SIDDIQUI et al., 2005), (SIDDIQUI; VILLAZÓN; FAHRINGER, 2006) e (WIECZOREK et al., 2006), tratam sobre o gerenciamento de recursos do *Askalon* (FAHRINGER et al., 2005), um ambiente para desenvolver e executar aplicações de Grade. Infelizmente, os únicos recursos que estes trabalhos gerenciam são processadores, com reservas simples e múltiplas.

Em (SIDDIQUI et al., 2005), o Askalon é modificado para funcionar com o Globus Toolkit versão 4 (GT4) (FOSTER, 2005) e com *WS-GRAM*, que é uma implementação do *GRAM* (CZAJKOWSKI et al., 1998) através de *WSRF* (*Web Services Resource Framework*). O gerenciamento de recursos é feito de forma que as reservas sejam modeladas como *WS-resources*. A arquitetura apresenta um modelo onde o usuário que deseja reservar processadores se comunica com um gerenciador de reservas. Este se comunica com o gerenciador de recursos local. Na hora de utilizar os processadores, o usuário deve se comunicar com o WS-GRAM, que verificará se o usuário tem uma reserva feita e se é para aquele momento e, se for, ele dá acesso aos mesmos.

Em (SIDDIQUI; VILLAZÓN; FAHRINGER, 2006), um gerenciador de recursos modelado em três camadas é apresentado. A primeira camada é a de alocação. Ela é im-

plementada através de um algoritmo de empacotamento em fita adaptado para reservas, que foi chamado de *VSHSH*. A segunda camada é a de co-alocação, construída sobre a camada anterior, e gerencia uma ou mais alocações. A terceira camada é a de coordenação, que busca resolver problemas gerados pela competição de diversos usuários pelos mesmos recursos, gerando soluções não conflitantes, através de uma re-organização das reservas na agenda ou através da redução do número de recursos utilizados por cada usuário.

Em (WIECZOREK et al., 2006), a idéia principal é que um usuário envie um *workflow*, ou seja, uma aplicação que contém um conjunto de sub-processos definidos. Ao recebê-lo, o gerenciador reserva os processadores necessários para que o *workflow* seja executado adequadamente. São propostos dois algoritmos para as reservas. O primeiro, chamado de *attentive*, recebe uma requisição de reserva para um período determinado. Se houver processadores disponíveis, este será o período da reserva. Caso não haja, são geradas alternativas e o usuário requisitante pode escolher alguma, desistir ou re-negociar. O segundo algoritmo chama-se *progressive*. Este algoritmo é uma extensão do primeiro que faz com que, ao se ter mais de um usuário e, portanto, mais de um *workflow*, o algoritmo não permita que um dos usuários tome todos os recursos. Isso é feito através do estabelecimento de limites para que cada usuário só possa fazer um certo número de reservas por vez.

2.2.4 Outros Casos de Reservas em Grades

(KEAHEY; MOTAWI, 2004) propõe uma arquitetura chamada *VAS* (*Virtual Application Service*). Nesta arquitetura, usuários da Grade utilizam um serviço persistente chamado *VAS factory*, pelo qual são instanciados serviços que utilizam diversos recursos. O objetivo é facilitar a execução de serviços em tempo real. O gerenciamento de recursos é feito por meio de reservas parciais, utilizando um gerenciador chamado DSRT (NAHRSTEDT; CHU; NARAYAN, 1999). Pode-se utilizar reservas imediatas e antecipadas, porém só é considerado o recurso processador. Reservas múltiplas e outros recursos não são suportados.

(FAROOQ; MAJUMDAR; PARSONS, 2005) traz um estudo sobre o uso de *relaxamento* em reservas simples e antecipadas. Relaxamento permite que o gerenciador de reservas tenha a flexibilidade para escolher o intervalo de tempo mais adequado para a reserva, ao invés de simplesmente tentar reservar a mesma em um instante pré-definido. É apresentado um estudo do desempenho de um algoritmo de gerenciamento de reservas ao utilizar-se de reservas com diferentes valores de relaxamento. A conclusão que se chega

é que este fator é muito importante para a boa utilização dos recursos. O ponto fraco do trabalho é que o gerenciador não suporta nem reservas parciais nem reservas múltiplas.

(DECKER; SCHNEIDER, 2007) apresenta um estudo sobre co-alocação de recursos para *workflows*. A intenção é gerenciar eficientemente as atividades de cada workflow, mapeando as mesmas para os diferentes recursos que elas exigem. Este mapeamento é possível com o uso de reservas antecipadas. Foram apresentados algoritmos baseados em heurísticas, de forma a reduzir a complexidade algorítmica. No trabalho, um *framework* foi utilizado para modelar e executar algoritmos. Tomou-se por base o algoritmo *HEFT* (*Heterogeneous Earliest-Finish-Time*) (TOPCUOUGLU; HARIRI; WU, 2002). A partir deste, foi desenvolvido o HEFT-Sync, que traz como vantagem o tratamento de dependências, o suporte a reservas antecipadas e suporte à co-alocação de recursos. Porém, foi constatado que o HEFT-Sync teve a maioria de seus *workflows* rejeitados devido a não conseguir encontrar recursos disponíveis em muitas requisições com dependências, dentro do tempo da co-alocação. Para isso foi desenvolvido o *HEFT-SyncBT*. Este algoritmo, diferente do anterior, ao invés de agendar as reservas para o primeiro intervalo livre disponível, registra os possíveis períodos de agendamento e os guarda em uma árvore. Assim, caso ocorra problemas de falta de recursos, algumas reservas podem ter seus intervalos modificados de acordo com as opções registradas na árvore e, assim, problemas locais podem ser resolvidos. O algoritmo aborda reservas de processadores e de banda de rede. Um ponto forte do gerenciador é apresentar reservas flexíveis para cada reserva, pois os limites de início e fim de cada reserva não estão definidos, podendo ser escolhidos de acordo com as dependências do workflow. O ponto fraco do algoritmo é não considerar reservas múltiplas.

(CASTILLO; ROUSKAS; HARFOUSH, 2007), por fim, aborda algoritmos para escalonamento de reservas, com a intenção de fazer uma boa utilização dos recursos. A forma de deixar os algoritmos mais eficientes proposta foi através do uso de árvores binárias balanceadas para armazenar os intervalos de tempo em que os recursos estariam disponíveis. Foram propostos três algoritmos: *first-fit*, *min-LIP* e *min-TIP*. O *first-fit* busca na árvore o primeiro intervalo onde cada nova reserva caberia. O *min-LIP* e o *min-TIP* buscam o período com menor tamanho possível onde caberia cada nova reserva; com a diferença que o *min-LIP* ordena os intervalos livres por ordem de início dos mesmos, enquanto *min-TIP* ordena eles por ordem de término. Os algoritmos propostos, porém, são restritos a processadores e não suportam reservas múltiplas e nem parciais.

2.2.5 Considerações

Foram considerados como trabalhos relacionados apenas aqueles que apresentaram alguma proposta de reservas de recursos para Grades com características fundamentais para um gerenciador de reservas, tal como implementação de reservas antecipadas. Alguns trabalhos, tal como (LIVNY; RAMAN, 1998), que discute sobre o gerenciamento de recursos do *Condor*, que por sinal não suporta reservas antecipadas e baseia-se em escalonamento de tarefas dinâmico, não foram considerados importantes para este trabalho, e por isso não foram citados.

	Reservas abrangentes			Reservas flexíveis		Recursos Suportados		
	RS	RM	RP	RA	Flexibilidade	processador	banda	outros
GARA	sim	não	sim	sim	sim	sim	sim	sim
Smith et al 2000	sim	sim	não	sim	indireta*	sim	não	não
Burchard et al	sim	não	sim	sim	sim (R. maleável)	não	sim	não
Elmroth et al 2004	sim	sim	não	sim	não**	sim	não	não
Roeblitz et al 2004	sim	sim	não	sim	sim (R. elástica)	sim	não	não
Askalon	sim	sim	não	sim	indireta*	sim	não	não
Keahey et al 2004	sim	não	sim	não	não	sim	não	não
Farooq et al 2005	sim	não	não	sim	sim (relaxamento)	sim	não	não
Decker et al 2007	sim	sim	não	sim	sim	sim	sim	não
Castillo et al 2007	sim	não	não	sim	sim	sim	não	não

RS = reserva simples; RM = reserva múltipla; RP = reserva parcial; RA = reserva antecipada;

* Tempo de início e término fixos. Caso o intervalo não esteja disponível, o escalonador indica outros períodos possíveis e o usuário pode escolher um.

** tempo de início fixo. Porém, o término não é definido, e sim previsto por algoritmos.

Figura 2.6: Trabalhos relacionados à gerência de reservas.

Como é possível observar pela figura 2.6, os diferentes trabalhos citados não conseguem cumprir todos os requisitos desejados para o gerenciamento eficiente de reservas. O trabalho que mais chega perto de cumpri-los é o GARA, porém ele não suporta reservas múltiplas e é um projeto que não evoluiu para acompanhar as novas tendências de computação em Grade baseadas em OGSA.

É possível perceber também que nenhum trabalho suporta reservas múltiplas e reservas parciais ao mesmo tempo. O ideal seria que, de acordo com os tipos dos recursos, o usuário provedor dos mesmos pudesse escolher o tipo de reserva que seria utilizada, simples, parcial ou múltipla, e o escalonador deveria ter suporte para as mesmas.

Outra observação importante é o fato de que apenas o GARA pôde reservar outros tipos de recurso além de processador e banda de rede. Este fator se deu porque ele

oferece abstrações homogêneas para representar os recursos, de forma que são reservados *objetos do tipo recurso*. Em outras palavras, é preciso que o gerenciamento de reservas seja genérico e com baixo acoplamento em relação aos recursos, possibilitando assim a reserva de diversos tipos de recursos.

3 Gerência de Reservas

Este capítulo apresenta algumas propostas para um gerenciamento de reservas adequado. Primeiramente, é apresentada uma arquitetura onde é definida a forma com que os usuários, o gerenciador de reservas e os gerenciadores de recursos interagem. Depois, é apresentada a forma com que a reserva é modelada. A seguir, são apresentadas abstrações para modelar os recursos e reservas em um escalonador, de forma a permitir algoritmos eficientes e com boa utilização dos recursos. Por fim, são feitas algumas considerações a respeito das propostas.

3.1 Gerenciador de Reservas

Um gerenciador de reservas pode ou não ser acoplado ao gerenciador de recursos. Esta proposta apresenta uma arquitetura de gerência de recursos onde o gerenciador de reservas está situado em uma camada superior ao gerenciador de recursos local. Na verdade, é possível que o gerenciador de reservas esteja localizado em um computador diferente do que possui os recursos, tendo como únicas funções negociação e agendamento de reservas.

Por ser uma camada superior, um gerenciador de reservas pode ser responsável pelas reservas de recursos de diferentes computadores, ou até mesmo ser responsável pelas reservas de todos os recursos de uma determinada rede local ou *organização virtual* (FOSTER; KESSELMAN; TUECKE, 2001). Isto pode ser melhor visualizado na figura 3.1, que mostra um exemplo onde um gerenciador de reservas é responsável pelas reservas dos recursos de todos os computadores de uma organização virtual. Nesta figura, há oito agendas temporais para o agendamento de reservas: as três da esquerda organizam reservas múltiplas (uma de discos rígidos e as outras duas de processadores) de recursos dos usuários dez e onze; as três centrais organizam reservas simples ou parciais de recursos do usuário 12; as duas da direita representam reservas parciais das capacidades de transferência de dados dos usuários onze e doze.

Uma característica importante que um gerenciador de reservas deve possuir é que ele

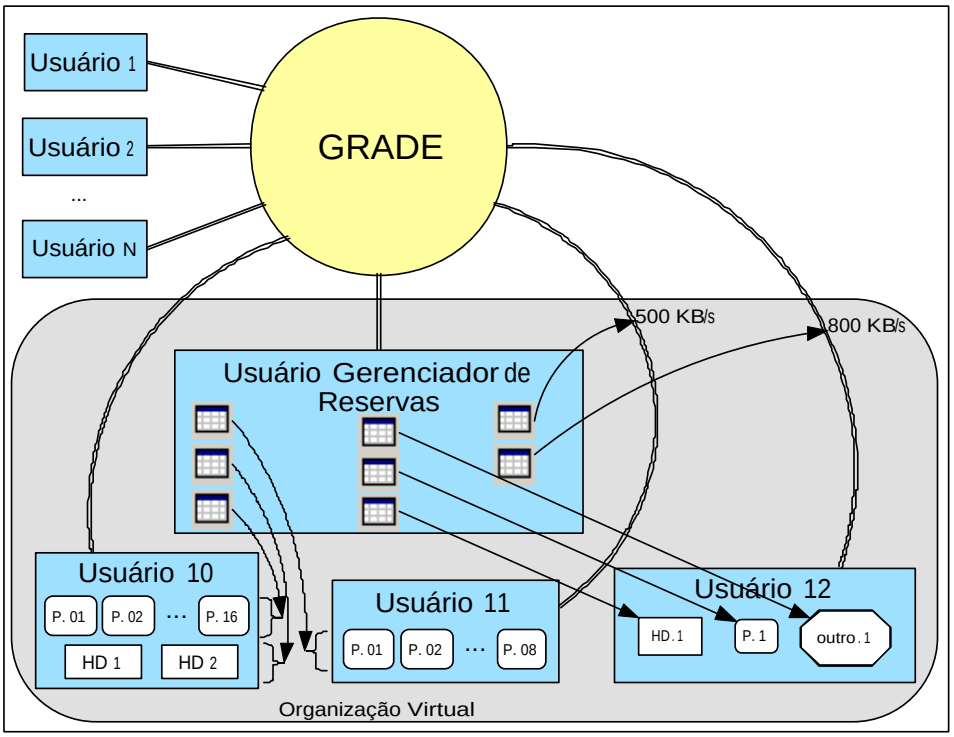

Figura 3.1: Gerenciador de reservas em uma organização virtual.

deve ser genérico o bastante para ser capaz de trabalhar com reservas de diferentes tipos, com recursos heterogêneos e com gerenciadores de recursos locais diversos. Por isso, o ideal é que os algoritmos de escalonamento de reservas trabalhem de forma homogênea e genérica, e o gerenciador tenha interfaces padronizadas e protocolos para comunicação com usuários e gerenciadores locais de recursos.

Outro ponto importante é o fato de que o gerenciador de reservas trabalha apenas com as reservas; o escalonamento de tarefas e controle de uso dos recursos não é seu papel. Ele é utilizado pelos usuários para obter reservas, e pelos gerenciadores de recursos para verificar se os usuários que estão requisitando os recursos possuem realmente reservas dos mesmos.

3.2 Reservas e Atributos

No momento da requisição da reserva, o usuário deve fornecer alguns parâmetros que indicam os recursos/capacidades a serem agendados, suas preferências e os limites tolerados. Este trabalho apresenta a seguinte proposta de parâmetros passados em uma requisição de reserva:

- *Tipo do Recurso*: indica o tipo do recurso desejado.

- *Capacidade desejada*: indica o número de recursos reservados ou a capacidade a ser reservada. O formato deste parâmetro depende do tipo do recurso.
- *Duração*: é o intervalo de tempo que durará a reserva.
- *Limite para iniciar*: é o instante de tempo estabelecido como o mais cedo possível para a reserva iniciar.
- *Limite para terminar*: é o instante de tempo estabelecido como máximo para que a reserva termine. A reserva não pode ser agendada para terminar após este parâmetro.

Além dos atributos passados como parâmetro em uma requisição, uma reserva ainda possui mais três atributos: *tempo de início* e *tempo de término*, que define o intervalo exato no qual a reserva foi agendada; e um atributo identificador, representado por um número inteiro.

3.2.1 Relaxamento

Existe um fator primordial para estabelecer reservas com flexibilidade: o uso de *relaxamento* (ou *laxity*) (FAROOQ; MAJUMDAR; PARSONS, 2005). Este fator é calculado pela relação entre a diferença entre os limites para iniciar e terminar, e a duração da reserva. Se esta diferença for igual à duração, não há relaxamento. Porém, se a diferença entre os limites for maior que a duração, há relaxamento, ou seja, há flexibilidade para que o gerenciador possa escolher o melhor intervalo para agendar a reserva.

Com relaxamento, um usuário não estabelece um tempo único para a reserva do recurso, mas sim uma faixa de tempo no qual o gerenciador de reservas terá a liberdade de escolher qual o melhor intervalo para a mesma. Isto permite melhor utilização dos recursos, uma vez que o gerenciador terá condições de identificar possíveis soluções e escolher a que considerar mais adequada, dentro das especificações do usuário.

Por outro lado, caso o usuário faça questão de estabelecer o período exato onde a reserva deve ser efetuada, ou seja, sem relaxamento, basta que ele envie os parâmetros de limite para iniciar e para terminar com uma diferença exatamente igual à duração. Através disso, haverá um único intervalo possível para a reserva, bastando ao gerenciador a verificação da disponibilidade de recursos neste intervalo.

O relaxamento é um valor implícito obtido pelos limites e pela duração de uma reserva, ou seja, ele é definido para cada reserva pelo usuário requisitante. A fórmula para se

determinar o relaxamento é: $\textit{Relaxamento} = \frac{(\textit{limite para terminar} - \textit{limite para iniciar})}{\textit{duração}} - 1$. Através desta fórmula é possível perceber que, quando a diferença entre os limites for igual à duração, o relaxamento será nulo. Se for o dobro da duração, o relaxamento terá valor de 100%. Se for o triplo será de 200%; e assim por diante. O relaxamento é expresso em porcentagem.

Relaxamento é muito importante, principalmente para reservas antecipadas. Uma tentativa de agendamento de uma requisição de reserva sem relaxamento corre grave risco de ser mal sucedida, pois é possível que já haja alguma reserva para o intervalo rígido requisitado. Com relaxamento, há mais possibilidades de se obter um intervalo onde a reserva pode ser agendada.

3.3 Modelagem de Recursos e Reservas para Escalonamento de Reservas

Para que não haja conflitos entre as reservas agendadas, é necessário organizá-las. O escalonamento é a etapa do gerenciamento em que um algoritmo recebe uma requisição de reserva e verifica a possibilidade de agendamento da mesma dentro dos parâmetros passados, de acordo com os recursos disponíveis. Para saber quais recursos estarão disponíveis, é necessária a análise das reservas já agendadas. A complexidade desta análise depende da forma com que elas são modeladas e organizadas.

Este trabalho propõe o uso de empacotamento em fita, que é um problema da matemática onde se busca encaixar (ou "empacotar") retângulos em um espaço geométrico no formato de uma fita; com algumas adaptações para que seja utilizado na modelagem das reservas e dos outros elementos no escalonador de reservas.

3.3.1 Empacotamento em Fita

É um problema da matemática que possui dois elementos geométricos principais: uma *fita* e diversos *retângulos*. A quantidade de retângulos pode variar.

A fita é especificada como um objeto bidimensional, com as dimensões comprimento e largura. A largura é dada por um valor finito e conhecido, constante em toda a fita. O comprimento é um valor grande o suficiente, com a seguinte característica: o início é definido, porém o fim não é definido. Em outras palavras, a fita possui um início conhecido mas um fim não conhecido.

Os retângulos também são objetos bidimensionais. Eles possuem tanto largura quanto comprimento finitos e conhecidos. As dimensões de diferentes retângulos podem variar entre si. É possível também que se tenha retângulos mais largos que compridos, quadrados, e outros mais compridos que largos ao mesmo tempo.

O problema está definido da seguinte forma: deve-se encaixar ou "empacotar" os retângulos na fita. Por empacotar, entende-se mover o retângulo sobre a fita, de forma que toda a área do retângulo encontre-se dentro da área da fita. Isso faz com que seja impossível encaixar na fita retângulos que possuam uma largura maior do que a da fita. Outro fator fundamental é que nenhum retângulo pode ocupar a mesma área pertencente a outro retângulo, ou seja, a intersecção de área dos retângulos é sempre vazia.

Na figura 3.2, é apresentado um exemplo de empacotamento em fita. Na fita, cinco retângulos estão empacotados. Caso um sexto retângulo chegasse para ser empacotado, ele só poderia ser empacotado caso sua largura não fosse maior que a da fita.

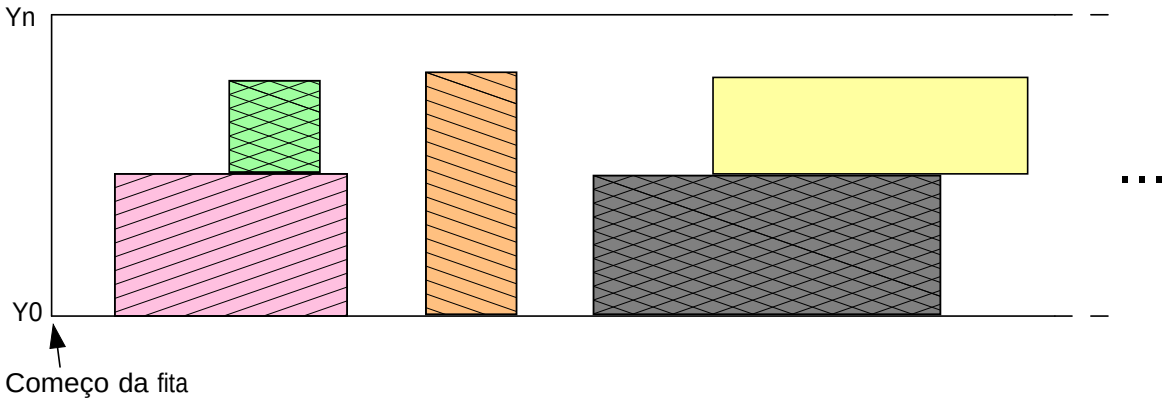

Figura 3.2: Algoritmo de empacotamento em fita.

O empacotamento pode-se dar com diferentes objetivos. O objetivo que mais se leva em conta, normalmente, é a utilização da fita. Por utilização entende-se a área da fita que está sendo ocupada por retângulos em relação à área que vai do início da fita até o último retângulo. Uma melhor utilização se dá quando se consegue encaixar os diferentes retângulos de forma a que preencham melhor os espaços livres, e que os retângulos fiquem o máximo possível no início da fita. Além deste objetivo pode haver outros, tal como ter os retângulos empacotados por ordem de tamanho, ou então ter o máximo de retângulos ocupando espaços próximos, ou outro objetivo qualquer. Porém, a melhor utilização acaba normalmente sendo o objetivo mais buscado.

Existem diversas formas de se tentar alcançar a solução, ou seja, diversos algoritmos para isso. Aí surge uma característica interessante do problema: a busca de uma solu-

ção ideal para melhor utilização dos retângulos envolve tentar todas as combinações de organização dos retângulos na fita. Esse problema é, computacionalmente falando, um problema NP-completo.

Por ser de tal complexidade, diversos algoritmos seguindo diferentes heurísticas foram criados para tentar gerar boas soluções de utilização da área da fita, porém com baixa complexidade algorítmica. Algumas das heurísticas organizam os retângulos por ordem de largura, outras por ordem de comprimento, outras ainda colocando próximos uns dos outros os retângulos com mesmo comprimento ou mesma largura. Outro tipo de heurística interessante é dado pelo estabelecimento de cortes virtuais na fita, separando assim a fita em partes (também chamadas de *gavetas*) e tentando organizar os retângulos de diferentes maneiras em cada gaveta. A forma como os cortes são estabelecidos, bem como a forma posterior de organização podem ser bem variadas. Além disso é possível estabelecer cortes verticais e também cortes horizontais, ou até mesmo ambos na mesma heurística. Com essa variedade de técnicas, é possível estabelecer um grande número de possíveis soluções para o problema. Uma "melhor" solução, em geral, não é estabelecida, porque diferentes soluções podem ser consideradas como melhores dependendo de onde o empacotamento em fita é usado.

Há algumas especificações que podem ser feitas que geram diferentes formas de empacotamento. As principais são sobre quando que os retângulos são conhecidos e se eles podem ou não ter sua posição na fita modificada. Assim, surgem dois tipos principais de problemas de empacotamento: *on-line* e *off-line*.

- *Empacotamento On-Line* - o que há a mais do que a definição padrão de empacotamento em fita são duas definições adicionais. A primeira é que, uma vez que se determinou em que lugar da fita um certo retângulo irá ficar, essa decisão não pode ser refeita, ou seja, não se pode voltar atrás e alterar o lugar. A segunda é que só se conhece, no máximo, um retângulo por vez. Assim, deve-se empacotar tal retângulo, ou seja, decidir em qual lugar ele vai ficar na fita, e é só depois disso que é possível saber qual é o próximo retângulo. Em outras palavras, os retângulos vão sendo empacotados na ordem em que vão chegando e, uma vez empacotados, não são mais alterados. Este problema traz soluções com menor complexidade, porque se leva em conta apenas o retângulo que está na vez e a necessidade de determinar qual lugar é o melhor para ele. Por outro lado, a utilização da área da fita não é muito boa, porque não se sabe quais retângulos virão posteriormente, e porque a ordem da chegada dos retângulos altera o resultado final do empacotamento, uma

vez que não se pode voltar atrás.

- *Empacotamento Off-Line* - é como se fosse o oposto do empacotamento on-line. Os retângulos podem ser reorganizados na fita o quanto se desejar, e/ou se conhece todos os retângulos antes de começar a empacotar. Esse tipo de problema gera soluções com maior complexidade, porque são levados em conta todos os retângulos simultaneamente. Por outro lado, é possível a obtenção de uma melhor utilização da área da fita, pois há mais possibilidades. Inclusive, a solução de utilização ideal da fita, que é extremamente complexa, é definida apenas para empacotamento off-line.

3.3.2 Empacotamento em Fita para Reservas Múltiplas

Empacotamento em fita é um problema que pode ser adaptado e aplicado na gerência de reservas de recursos através de algumas definições. Essas definições são as seguintes:

- Fita: representa o conjunto de recursos idênticos que podem pertencer à mesma reserva múltipla.
- Retângulo: cada um representa uma reserva.
- Comprimento: o comprimento da fita e o de cada retângulo representam o tempo. Um retângulo é uma reserva com um tempo limitado. A fita tem como começo o instante de tempo "atual" (do momento), e está aberta a reservas futuras.
- Largura: representa a quantidade de recursos. A fita tem como largura o número total de recursos para uma certa reserva múltipla. Cada retângulo tem como largura a quantidade de recursos da reserva.
- Tipo de empacotamento: utiliza-se empacotamento em fita on-line. Isso ocorre porque as requisições de reserva são feitas de forma independente, ou seja, são agendadas as reservas de acordo com a chegada das requisições. Além disso, uma vez que uma reserva é agendada, o usuário recebe como resposta o intervalo da mesma. Isso significa que o retângulo que representa a reserva não pode ser modificado de posição na fita, o que faz com que o algoritmo não possa voltar atrás quanto à escolha de onde posicionar o retângulo.

A figura 3.3 mostra um exemplo de utilização de um algoritmo de empacotamento em fita na gerência de reservas múltiplas. O exemplo é o mesmo da figura 2.4, ou seja, quatro processadores idênticos. O algoritmo modela os diversos recursos como uma fita

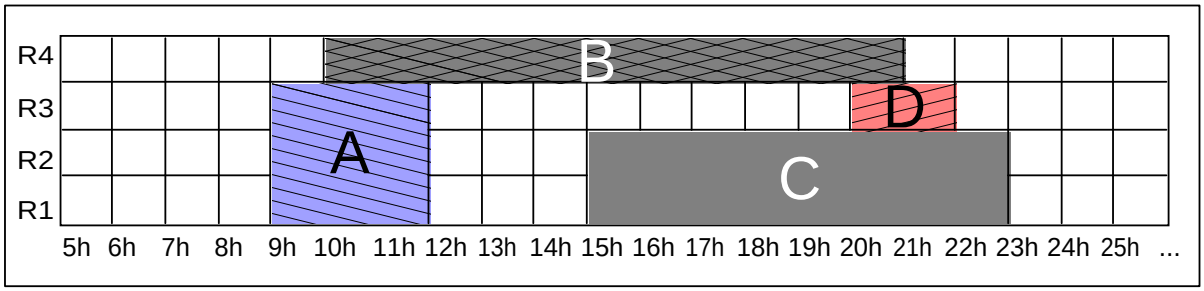

Figura 3.3: Empacotamento em fita para reservas múltiplas.

e as reservas como retângulos, para melhor controle. É possível notar que, neste tipo de utilização de empacotamento em fita, os valores de largura são restritos, porque é apenas possível reservar um, dois, três ou quatro processadores. Assim, estes são os únicos valores de largura aceitos. Uma outra característica importante é o fato de que a largura e comprimento não representam dimensões geométricas, mas sim o número de recursos para a largura, e o tempo para o comprimento.

É possível notar também outra particularidade dos algoritmos de empacotamento em fita: a modelagem de reservas como retângulos. Isso facilita a implementação de algoritmos de verificação da utilização dos recursos, por juntar recursos e tempo em retângulos: basta achar a área da fita ocupada para achar a utilização.

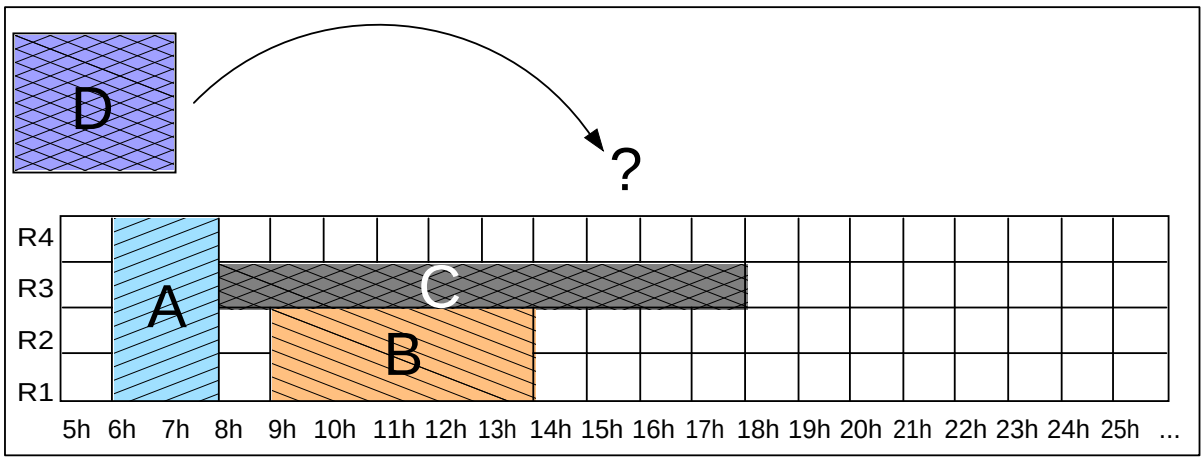

Figura 3.4: Problema de empacotamento para encaixar retângulo.

Há, porém, um ponto fraco na utilização desta modelagem: retângulos não permitem reservas de recursos que não sejam vizinhos. Isso pode ser observado na figura 3.4, onde é possível verificar que não se pode agendar uma reserva **D** de três recursos, para o período entre 14h e 18h, mesmo havendo três recursos disponíveis, porque não é possível formar um retângulo com eles. A reserva teria que entrar a partir de 18h. No próximo capítulo, é apresentado um algoritmo que consegue solucionar este problema.

Outro fator interessante que deve-se levar em conta é a questão de representação das reservas. É impossível estabelecer uma ordem onde uma começa apenas após o término da anterior, porque várias reservas podem estar agendadas para os mesmos intervalos de tempo, utilizando os diversos recursos. Assim, algoritmos para agendamento devem percorrer a fita em busca de um lugar onde se encaixe o novo retângulo/reserva. A forma com que isso é feito determinará a complexidade do algoritmo e a utilização da fita.

O uso de empacotamento em fita *on-line*, para escalonar reservas, possui uma particularidade. As reservas antecipadas, que possuem limites de início e término, não podem ser colocadas em qualquer lugar da fita. Elas precisam ser colocadas em uma região temporal da fita dentro dos limites particulares de cada reserva. Isso pode complicar o algoritmo, já que esta restrição faz com que não se obtenha uma utilização tão boa do recurso, porque poderia haver outros lugares na fita que gerariam uma melhor disposição das reservas. Mesmo assim, há um ponto positivo: o uso de apenas uma região da fita para agendar limita o número de opções, o que faz com que o algoritmo possa escolher mais rápido em qual lugar a reserva será agendada, ou determinar mais rápido se ela pode ou não ser agendada.

3.3.3 Empacotamento em Fita para Reservas Parciais

Empacotamento em fita também pode ser utilizado para escalonamento de reservas parciais, utilizando mecanismos similares aos usados para reservas múltiplas, pois também é um problema com duas dimensões: tempo e capacidade reservada. Assim, a dimensão de tempo continua sendo representada pelo comprimento da fita; porém a largura da fita, ao invés de representar a quantidade de recursos, representará a capacidade do recurso reservada.

Como comentado no capítulo anterior, há diversos tipos de capacidades que podem ser reservadas: banda de rede (em KB/s), poder de processamento (em ciclos), espaço de armazenamento (em KB ou MB), entre outros. Para tornar o algoritmo genérico, utiliza-se a representação da capacidade como porcentagem da capacidade total, como é possível visualizar na figura 2.5, onde as reservas parciais já estão sendo representadas por meio de empacotamento em fita.

3.4 Avaliação da Proposta

No capítulo anterior, foram apresentados os quatro principais requisitos desejados em um gerenciador de reservas: abrangência, flexibilidade, eficiência e boa utilização dos recursos. Neste capítulo são apresentadas: uma arquitetura para o gerencidor de reservas, os atributos das reservas e o uso de empacotamento em fita na modelagem de recursos e reservas para o escalonamento das mesmas.

A escolha dos atributos das reservas é fundamental para suprir o requisito de flexibilidade. Como estão estabelecidos a duração e os limites para início e término, o escalonador tem a flexibilidade para escolher o melhor intervalo e onde a reserva será efetuada. Além disso, é preciso suportar relaxamento, ou seja, limites para as reservas possivelmente maiores que as durações das mesmas, o que torna as reservas mais flexíveis, melhorando a utilização dos recursos e reduzindo a quantidade de requisições de reservas recusadas.

O uso de uma modelagem baseada em empacotamento em fita é proposto para a questão de eficiência e boa utilização dos recursos, pois ele utiliza abstrações para modelar os recursos e as reservas de uma forma a simplificar o escalonamento, o que provê eficiência, e facilitando a organização das reservas, o que contribui para otimizar a utilização dos recursos. As questões de utilização e eficiência irão depender também dos algoritmos que utilizarão empacotamento em fita. Os algoritmos desenvolvidos serão apresentados no próximo capítulo.

A abrangência é trabalhada pela proposta de uso de empacotamento em fita diferenciado para reservas parciais e para reservas múltiplas, o que torna o gerenciador capaz de suportar estes dois tipos de reserva. Isto é algo que nenhum dos trabalhos relacionados, apresentados no capítulo anterior, conseguem fazer simultaneamente.

Pode-se concluir, assim, que a proposta tem dois pontos fundamentais: uma **arquitetura** de gerenciamento de reservas, que ressalta a importância do gerenciador de reservas em relação a usuários e gerenciadores de recursos; e uma **modelagem** homogênea das reservas e recursos, através de empacotamento em fita, para escalonamento de reservas, de forma a cumprir os requisitos de utilização, eficiência, flexibilidade e abrangência. A proposta, por outro lado, não leva em conta particularidades do uso dos recursos, tais como políticas de uso ou forma de acesso.

4 Escalonamento de Reservas

Este capítulo apresenta três algoritmos desenvolvidos para o escalonamento de reservas. Eles são baseados na modelagem de recursos e reservas através de empacotamento em fita, e procuram cumprir os requisitos de boa utilização dos recursos e eficiência.

4.1 Algoritmo para Reservas Múltiplas

Para reservas múltiplas, foi desenvolvido um algoritmo baseado em empacotamento em fita on-line. O importante do algoritmo é determinar, ao chegar uma requisição de reserva, se ela pode ou não ser agendada e, caso possa, em que intervalo de tempo e quais os recursos agendados.

Ao chegar a requisição, a primeira coisa a se fazer é encontrar possíveis intervalos de tempo nos quais a reserva poderia ser agendada. Vamos chamá-los de "intervalos candidatos". São candidatos porque, para que se possa agendar a reserva em um deles, é preciso que haja recursos suficientes no intervalo.

A forma de se encontrar estes candidatos é através das reservas já agendadas (retângulos já empacotados na fita) e dos valores de duração, limite para iniciar e para terminar da requisição feita. Os passos para isto são os seguintes:

1. Encontram-se todos os valores de término de cada um dos retângulos já empacotados. Nestes valores, recursos são liberados, ou seja, é possível que, ao término de alguma reserva, haja recursos disponíveis suficientes para a nova reserva.

2. Dos valores encontrados, são descartados aqueles que não permitem que a reserva seja realizada entre os valores de limite para iniciar e terminar. Assim, sobram apenas valores para início da reserva possíveis, ou candidatos.

3. Além destes, é acrescentado como candidato o próprio valor do limite para iniciar, pois é sempre um valor temporalmente possível de agendar uma reserva.

A figura 4.1 apresenta a representação de uma fita com alguns retângulos empacotados. Uma nova requisição de reserva de três recursos por três horas chega, e então, determinam-se os candidatos pelos passos recém listados. Os primeiros intervalos são determinados pelo primeiro passo, e têm seus inícios representados na figura pelas setas pequenas. Depois, no segundo passo, sobram apenas os intervalos possíveis (candidatos), com inícios representados pelas setas grades. É acrescido como candidato também o valor do limite para iniciar, também com início representado por uma seta grande.

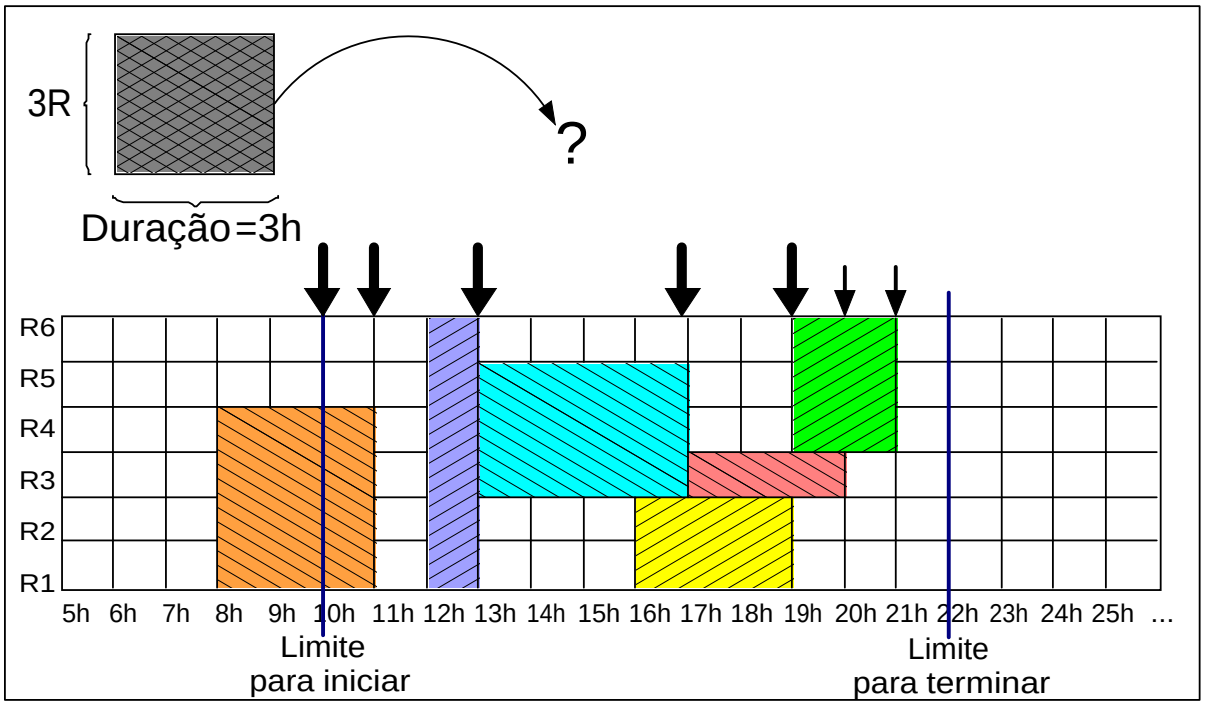

Figura 4.1: Encontrando os intervalos candidatos para agendamento.

Após achar os intervalos candidatos, é preciso identificar se, em pelo menos um deles, há recursos suficientes para efetuar a reserva. Esta verificação é feita para cada um dos candidatos, até que se encontre um em que a reserva possa ser efetuada. Os passos para isso são os seguintes:

1. Acham-se todas as reservas agendadas para o período do intervalo candidato. Elas são as que não cumprem nenhum dos dois requisitos: (i) começar e terminar antes do começo do intervalo; e (ii) começar e terminar depois do fim do intervalo candidato.

2. São analisadas essas reservas e determinados os recursos utilizados e os disponíveis no período.

3. Caso haja recursos suficientes para a reserva, ela é efetuada.

4. Caso não haja, este intervalo candidato é descartado, e o algoritmo repete o procedimento para o próximo candidato.

5. Caso em nenhum dos intervalos candidatos seja possível agendar a reserva, chega-se à conclusão de que é impossível efetuar a reserva.

Este algoritmo citado faz um bom uso dos recursos porque é capaz de determinar todas as possibilidades que se encaixam na descrição da requisição de reserva. Além disso, ele traz uma melhoria quanto ao conceito de empacotamento em fita por permitir que retângulos possam ser separados com cortes horizontais em sub-retângulos. Isso quer dizer que é possível agendar uma reserva para recursos não vizinhos na fita.

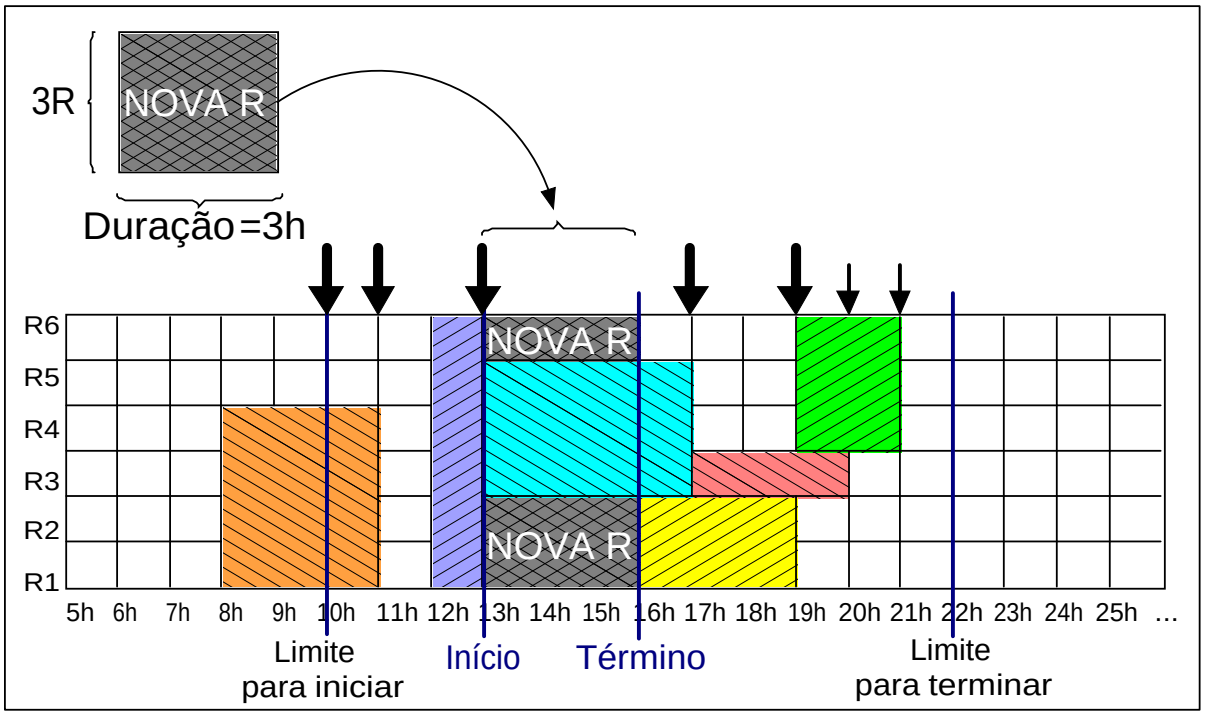

Figura 4.2: Agendando uma reserva de recursos.

A figura 4.2 mostra como o algoritmo procede após encontrar os intervalos candidatos (representados pelas setas). Na figura, o primeiro candidato é o que começa no limite para iniciar da requisição. Neste intervalo não há recursos suficientes, ou seja, não é possível reservar. Então é testado o próximo candidato, mas ocorre o mesmo. No terceiro candidato, há três recursos disponíveis, portanto será possível reservar. É possível perceber que o retângulo da reserva teve que ser dividido em dois, porque estavam disponíveis apenas o primeiro, o segundo e o sexto recursos. Mesmo assim, por estarem disponíveis, foram utilizados, e a reserva pode ser realizada com sucesso.

4.2 Algoritmos para Reservas Parciais

Para agendar reservas parciais, as soluções presentes na literatura ficam muito a desejar. Por isso, foi desenvolvida uma abstração sobre empacotamento em fita capaz de aumentar a eficiência do algoritmo e melhorar a utilização dos recursos: os *Variable Slots* (KUNRATH; WESTPHALL; KOCH, 2008). Foi também implantado uma forma de permitir que o dono do recurso possa disponibilizar apenas uma certa quantidade de capacidade para ser reservada, tal como 70% da capacidade máxima.

4.2.1 Limitando Capacidade Máxima

Um problema que pode ser levantado quanto às necessidades dos usuários e provedores de recursos é a questão de que um provedor pode querer limitar a porcentagem máxima de capacidade permitida para ser disponibilizada para reservas de Grade.

No proposta deste trabalho, é possível limitar a largura máxima utilizada da fita. Assim, a fita seria reduzida a uma "sub-fita", ou seja, uma fita com mesmo comprimento mas largura inferior à capacidade máxima. A largura desta nova sub-fita é a largura estabelecida pelo provedor do recurso. Por exemplo, na figura 4.3, um corte longitudinal é utilizado para produzir uma sub-fita com a quantidade de capacidade estabelecida pelo provedor, que é, neste exemplo, 70% da capacidade máxima. A nova sub-fita tem a mesma função do caso de uma fita sem restrições, ou seja, aceita-se retângulos / reservas com no máximo a largura / capacidade da sub-fita. Retângulos que exigem mais do que a largura da sub-fita são recusadas. No caso da figura, uma reserva que pedisse mais de 70% da capacidade máxima da fita, que seria um valor superior à capacidade da sub-fita, seria recusada.

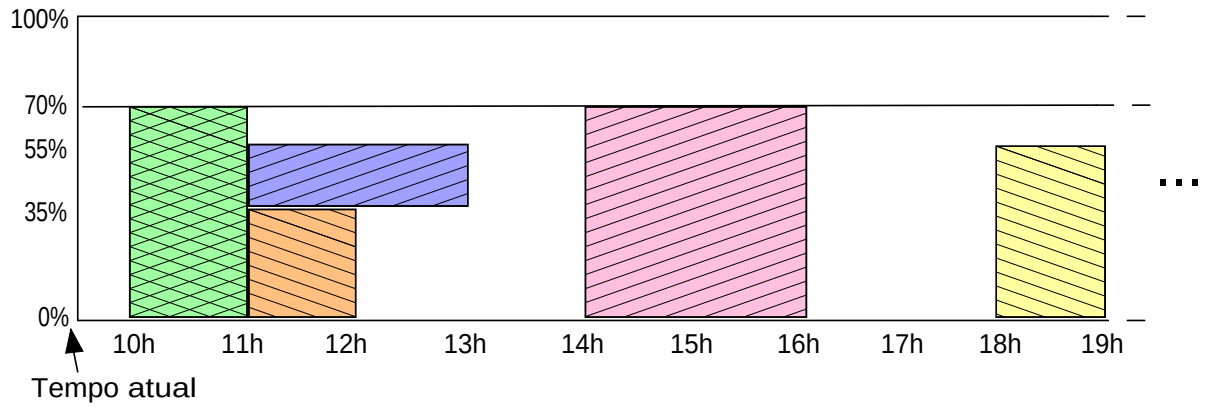

Figura 4.3: Suporte à limitação de capacidade reservada.

4.2.2 Variable Slots

Nas reservas múltiplas, a informação sobre quais dos diferentes recursos são agendados para cada reserva é importante. Por exemplo, se há quatro processadores, reservar os dois primeiros ou então os dois últimos não é a mesma coisa, por mais que ambas sejam reservas de dois processadores. Em reservas parciais isso não acontece: reservar os primeiros 30% da capacidade ou os últimos 30% é a mesma coisa. Este fato abre espaço para algumas estratégias. Uma destas estratégias que foi criada e desenvolvida para este trabalho é o uso de *Variable Slots* (VS). Eles são estruturas utilizadas para organizar reservas agendadas e representá-las, aumentando a utilização dos recursos e a eficiência.

VS são formados por reservas parciais inteiras ou partes de reservas parciais. Estas são juntadas no mesmo VS quando estão reservando capacidades do recurso em um mesmo período de tempo. Em outras palavras, VS são estruturas determinadas por intervalos de tempo e compostas por reservas e/ou pedaços de reservas. Eles servem para que se determine a quantidade total de capacidade utilizada em diferentes períodos de tempo, obtida pelas diversas reservas. Por exemplo, na figura 4.4 (a), está representada uma fita com duas reservas R1 e R2. Em (b), R2 é dividida em duas partes porque isso não altera a reserva, já que continua a mesma quantidade de capacidade no mesmo intervalo de tempo. Em (c), são utilizados VS. No caso da figura, há três VS: o primeiro é determinado pelo intervalo em que só há a reserva R1; o segundo é determinado pelo intervalo em que há as reservas R1 e R2; e o terceiro pelo intervalo em que só há a reserva R2.

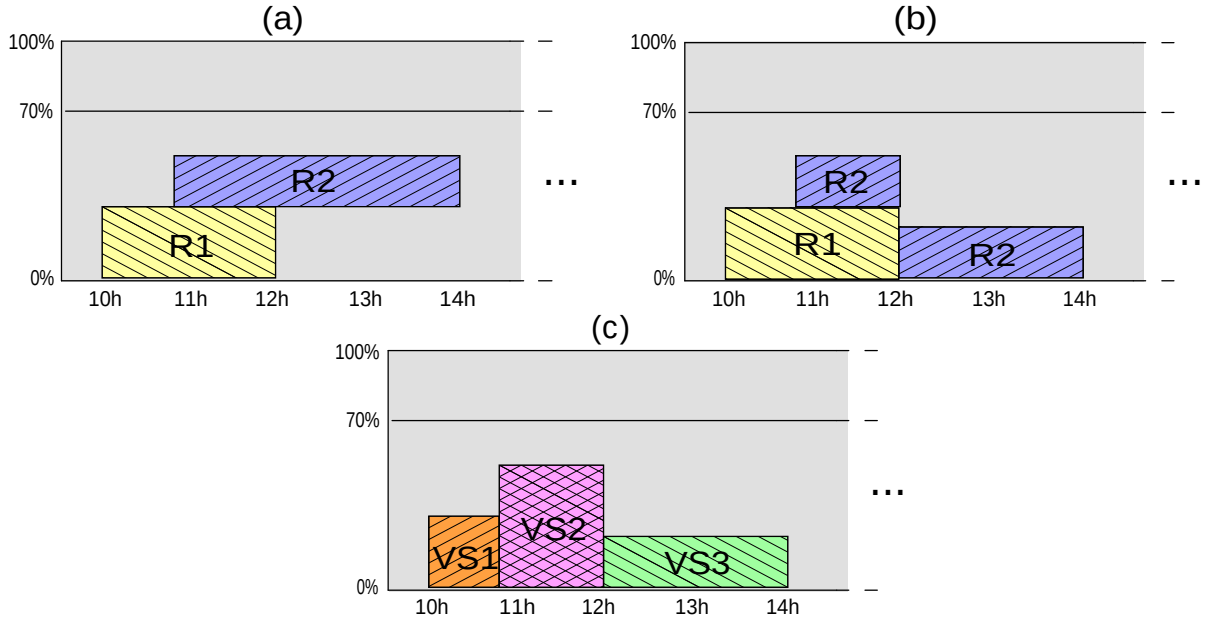

Figura 4.4: Empacotamento em fita com *Variable Slots*.

VS são muito úteis para representar a capacidade utilizada do recurso. Através deles, é possível verificar muito mais rapidamente quais são os períodos que possuem capacidade suficiente para a reserva. Na figura 4.4 (c), por exemplo, basta verificar a capacidade restante de cada um dos três VS, ou seja, a distância entre o limite superior de cada um dos VS e a capacidade máxima (70% no exemplo da figura). A capacidade delimitada por VS1 é a capacidade de R1, a delimitada por VS2 é a de R1 mais a de R2, e a por VS3 é a de R2 somente.

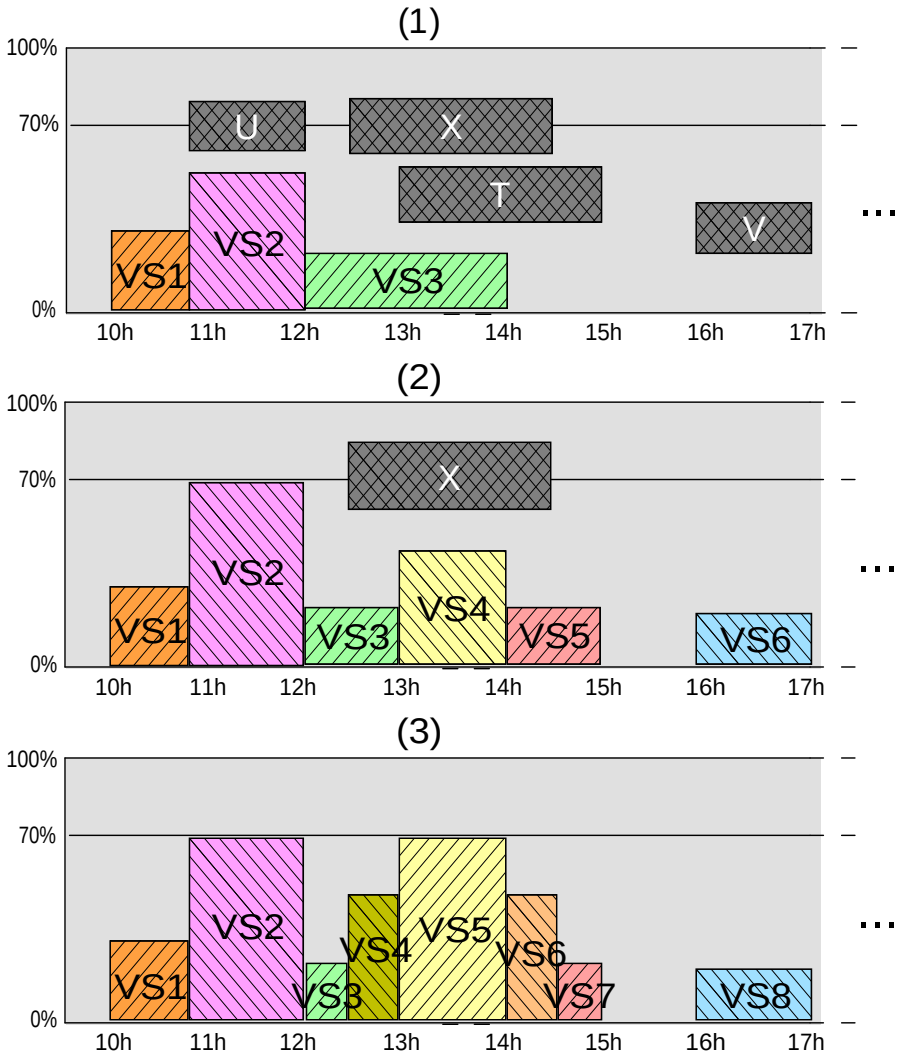

Figura 4.5: Exemplos de empacotamento com *Variable Slots*.

No exemplo da figura 4.4, vamos agora acrescentar algumas reservas. As reservas T, U, V e X serão adicionadas conforme aparece na figura 4.5 (1), ou seja, cada reserva está posicionada na fita no período em que ficará, porém ainda não organizadas. Em (2), aparecem os VS conforme ficam após englobarem as três primeiras reservas. Em (3) aparecem os VS após a última reserva. Na figura, ao acrescentar-se a reserva T, o terceiro

VS é dividido em dois. Isso ocorre porque há diferenças de capacidade e, por isso, um novo VS se forma a partir das reservas. Para englobar U, como ele possui o mesmo período do segundo VS, não foi preciso criar um novo VS, apenas se juntar a ele. Com V, como o período escolhido não possui nenhum VS, um novo VS é criado para conter V. Por fim, a chegada de X acaba gerando mais dois VS, pois é preciso partir dois dos VS existentes ao meio.

Uma particularidade dos VS é o seguinte: um VS pode possuir diversas reservas ou partes de reservas. Uma reserva pode ser representada por diversos VS. Porém, uma reserva não pode possuir apenas um pedaço do VS, ou seja, do começo ao fim do intervalo de tempo de um VS, ele terá as mesmas reservas ou pedaços de reservas. Isso pode ser exemplificado através da figura 4.6, onde (a) representa as reservas, e (b) os VS destas mesmas reservas. As duas primeiras reservas utilizam a mesma quantidade de capacidade do recurso, começam uma logo depois da outra, e são representadas por VS diferentes. O mesmo acontece nas três últimas reservas. Note que a quantidade total de capacidade dos dois VS que representam as três reservas é a mesma, porém eles são dois VS ao invés de um só. Mesmo assim, as duas últimas reservas formam um só VS, pois se encaixam inteiras no VS.

Este fator foi assim determinado pelo fato de que, se fosse permitido que um VS tivesse reservas que acabassem no meio dele, isso traria uma complexidade adicional ao algoritmo, pois as verificações se tornariam muito mais exaustivas, já que o VS teria sub-intervalos com diferentes reservas.

4.2.3 Algoritmo SUVS

foi desenvolvido neste trabalho o algoritmo *SUVS* ("*Scheduling Using Variable Slots*", ou "*escalonando com uso de VS*"). Este algoritmo utiliza a abstração de VS para agendar reservas paciais da capacidade de um recurso de forma eficiente e com alta utilização do recurso. O algoritmo funciona da seguinte forma:

1. Todas as reservas estão sob a forma de uma lista de VS. O algoritmo começa a percorrer a lista, até chegar no primeiro VS onde o período do mesmo tem algo em comum com o período entre os limites da requisição.

2. A partir daí, continua-se percorrendo a lista e marcando quais são os VS localizados entre os limites.

3. Tendo feito isso, tenta-se agendar a reserva para começar no limite para iniciar.

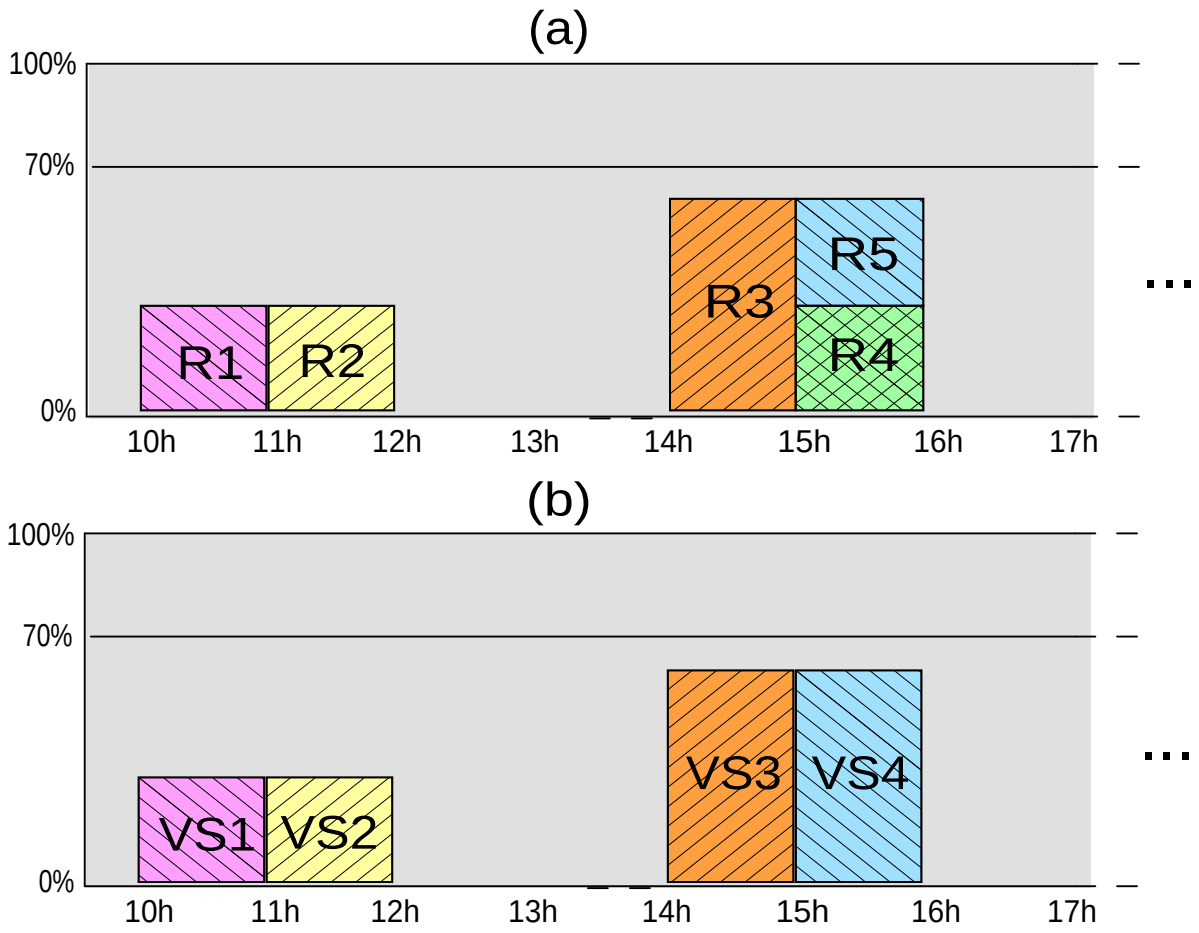

Figura 4.6: Organizando reservas em Variable Slots.

4. Para ver se é possível agendar neste período, verifica-se todos os VS que ficariam no período da reserva e, caso a soma da capacidade de cada um com a capacidade requisitada seja inferior ou igual a capacidade máxima, a reserva pode ser agendada.

5. Se a reserva puder ser agendada, ela é anexada aos VS. Isso é feito através do método de partir a reserva para cair em diferentes VS. Este método pode também gerar novos VS, quando parte da reserva (ou ela toda) cai em um intervalo sem reservas, ou então se houver a necessidade de se partir um VS em dois. Assim que a reserva é efetuada e armazenada como VS, o algoritmo termina, retornando a mensagem de que foi possível agendar a reserva, e informando os valores de início e término da reserva.

6. Se não puder ser agendada, o início da reserva é ajustado temporariamente para ser igual ao instante em que termina o VS testado em que a capacidade do mesmo mais a capacidade requisitada foi maior do que a capacidade permitida.

7. Se o instante de término da reserva for maior que do que o limite para terminar, a reserva não pode ser agendada, e o algoritmo termina.

8. Caso contrário, repete-se os passos de quatro em diante.

Este algoritmo foi desenvolvido de forma a suportar reservas antecipadas com as seguintes particularidades, referentes as limites:

- **limites definidos**. É o caso mais complexo, onde tanto o limite para iniciar quanto o para terminar estão definidos.
- **Limite para iniciar não definido e limite para terminar definido**: neste caso, a reserva poderia começar a qualquer momento, mas há um limite máximo de tempo onde ela deve terminar. Assim, o algoritmo coloca o limite para iniciar como sendo o "tempo atual" (do momento), ou seja, a reserva poderia ser agendada para começar imediatamente ou então para começar posteriormente.
- **Limites não definidos**: é um caso onde a reserva poderia começar a qualquer momento e não há um limite para quando ela irá terminar. O algoritmo coloca o limite para iniciar como sendo o "tempo atual" e o limite para terminar como sendo "infinito". Isso faz com que o término da reserva nunca seja maior que o limite, ou seja, a reserva sempre poderá ser agendada. No pior caso, ela pode ser agendada para acontecer depois da última reserva agendada.
- **Limite para iniciar definido e limite para terminar não definido**: a reserva deve começar a partir de um instante definido, mas não há limites para quando deve terminar. Como a anterior, é sempre possível efetuar a reserva. Por "sempre", entende-se que os casos onde é sempre impossível agendar a reserva, que são quando a capacidade requisitada é maior que a capacidade total, ou então quando a requisição contém erros, já tenham sido eliminados desde o começo.

A figura 4.7 apresenta um exemplo de utilização do algoritmo SUVS. No exemplo, tenta-se agendar uma nova reserva (com tracejado duplo). O algoritmo primeiramente identifica os VS envolvidos (do terceiro ao penúltimo). Ao tentar agendar R em no limite para iniciar (passo 3), não é possível (verificação do passo 4), porque há um VS em que a capacidade do VS mais a requisitada de R ultrapassa a capacidade máxima estabelecida pelo provedor do recurso (80% da capacidade total, no exemplo dado). Então tenta-se reservar logo depois deste VS (passo 6). Ao verificar que agora é possível (passo 4), a reserva é anexada aos VS, como é mostrado na figura 4.7 na parte de baixo.

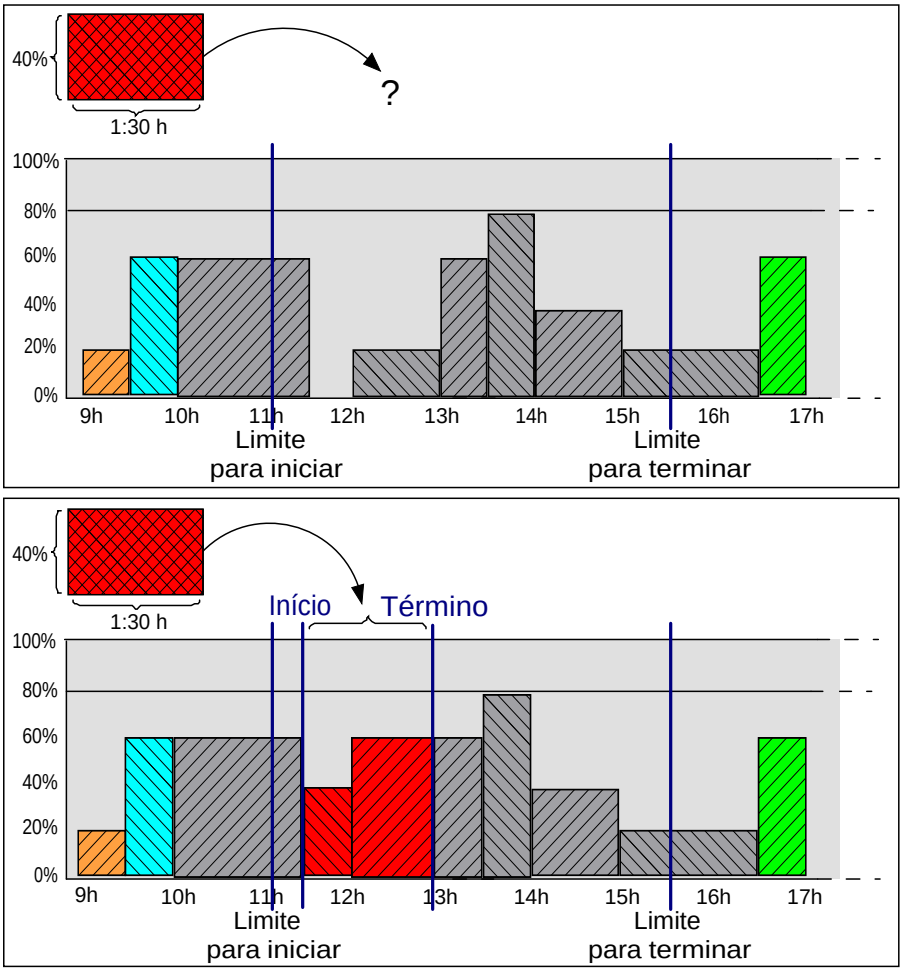

Figura 4.7: Exemplo de uso do algoritmo SUVS.

4.2.4 Algoritmo SUVS-P

O algoritmo SUVS-P é uma variação do algoritmo SUVS, que leva em conta a questão de *prioridade* de alguns tipos de reserva.

No algoritmo, são consideradas as reservas antecipadas sem limite de término definido como reservas com prioridade mais baixa que as demais. Assim, caso chegue uma reserva com limite para terminar finito e definido, ou seja, restrições de tempo para término da reserva, as reservas que não têm limite de término definido podem ser reajustadas para darem lugar a esta. Isto ocorre apenas no caso de, sem reajuste, ser impossível agendar a reserva de maior prioridade, e com o reajuste, ser possível.

Uma implicação deste algoritmo é que as reservas de baixa prioridade podem ser modificadas. Isso faz com que seja necessário que o provedor do reservas informe ao usuário dono desta reserva que ela teve seus valores de início e término modificados.

Os procedimentos do algoritmo SUVS-P, para uma reserva de baixa prioridade, são os mesmos de SUVS. Porém, para uma reserva de alta prioridade, são os seguintes:

1. Tenta-se agendar a reserva com os passos de SUVS. Se for possível agendar a reserva, o algoritmo termina.

2. Caso não seja possível, coloca-se o tempo de início da reserva como o limite para iniciar novamente.

3. Então, percorre-se cada um dos VS presentes no intervalo da reserva, identificando se a capacidade da nova reserva, mais a capacidade de VS excluindo-se a capacidade das reservas de baixa prioridade, não é maior que a capacidade máxima permitida.

4. Se pelo menos para um VS for maior, muda-se o início da reserva para o fim daquele VS, e verifica-se se ela não terminaria após o limite para terminar. Se terminaria, o algoritmo acaba com a impossibilidade de a reserva ser agendada. Se não, repete-se os passos 3 em diante.

5. Se não for maior para todos os VS, é porque é possível agendar a reserva ali, caso se elimine as reservas de baixa prioridade.

6. Remove-se as reservas de baixa prioridade dos VS, colocando-as em uma lista.

7. Anexa-se a nova reserva aos VS.

8. Após isso, as reservas que foram removidas são agendadas novamente pelo algoritmo SUVS.

9. Então, o usuário que requisitou recebe a confirmação, com os valores de início e término definidos. Os usuários das reservas modificadas (as que foram removidas) recebem também os novos valores de início e término das mesmas.

A figura 4.8 apresenta um exemplo de utilização do algoritmo SUVS-P. Os VS que possuem apenas reservas (e pedaços de reservas) com prioridade alta estão representadas com retângulos tracejados. Os VS que têm também reservas ou pedaços com prioridade baixa estão representados sem tracejado na parte equivalente a estas. Uma nova reserva de prioridade alta chega. Tenta-se agendar ela, porém não é possível (passo 1). Depois, verifica-se se é possível agendar caso se remova reservas de baixa prioridade (passos 2, 3, 4 e 5). Por fim, verifica-se que isto é possível; no caso, para intervalo de 13:00h a 14:30h. Na parte de baixo da figura 4.8, é possível visualizar que uma reserva de baixa prioridade

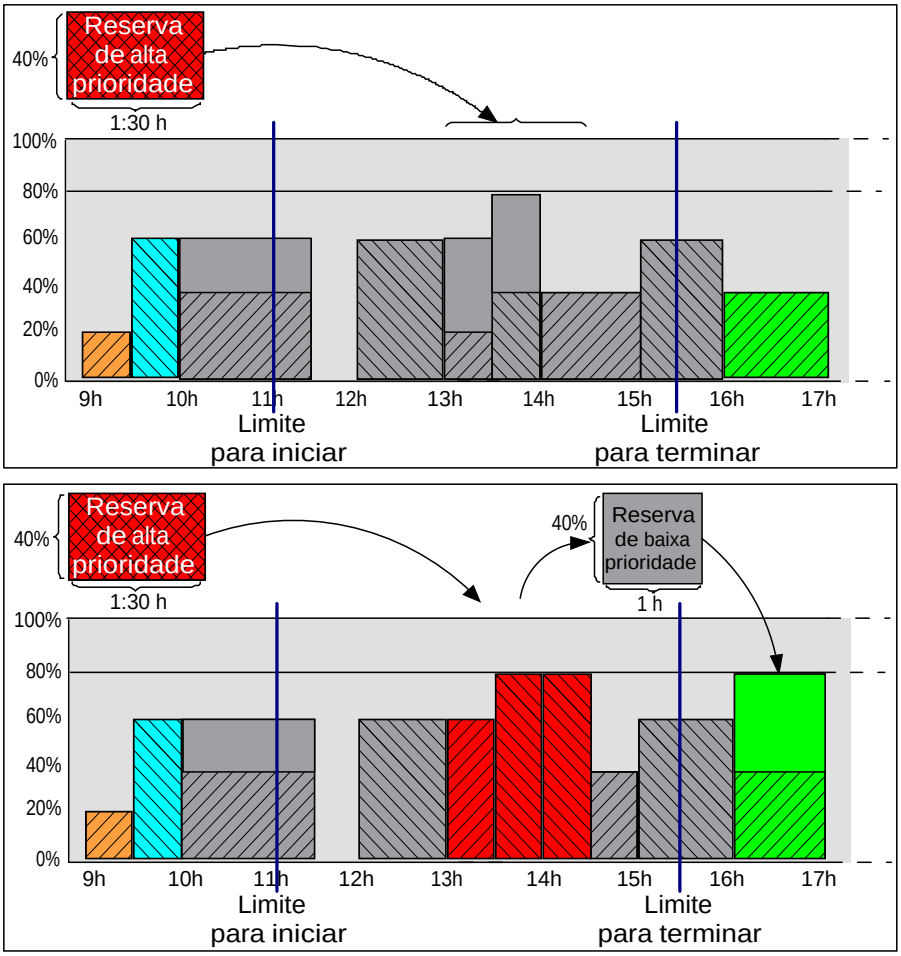

Figura 4.8: Exemplo de uso do algoritmo SUVS-P.

teve que sair para dar lugar à nova reserva. A nova reserva é agendada e, então, a reserva removida também é agendada.

5 Resultados Experimentais

Este capítulo tem por finalidade apresentar um estudo sobre o uso dos algoritmos de escalonamento propostos SUVS e SUVS-P, sob diferentes circunstâncias. O objetivo não é testar extensivamente os algoritmos de forma a registrar o desempenho dos mesmos, mas apresentar alguns testes por questão de validação e estudo de diferentes resultados obtidos pela variação de alguns fatores. Os fatores estudados são o número de reservas antecipadas, a proporção de reservas com limite para terminar definido e a taxa de relaxamento. Os resultados observados são a média de utilização da capacidade do recurso e a proporção de reservas recusadas.

5.1 Tecnologias Utilizadas

A idéia inicial para os testes foi o uso de simulador. Neste intuito, estudou-se a possibilidade de uso do *GridSim* (BUYYA; MURSHED, 2002), por ser um dos mais completos e por apresentar uma proposta de reservas antecipadas. Porém, foi possível observar que o GridSim não é capaz de suportar as diferentes definições propostas. Ele considera uma reserva como um conjunto de *Gridlets*, sendo cada um destes um conjunto de informações sobre uma tarefa e seu ambiente de processamento. Em outras palavras, não há distinção entre gerência de reservas e gerência de tarefas, e cada reserva é definida como um conjunto de processadores que será utilizado por um determinado conjunto de tarefas em um certo intervalo de tempo.

Neste sentido, há uma grande diferença entre as definições do GridSim e as apresentadas neste trabalho. A proposta deste trabalho é de um gerenciador de reservas que interage com gerenciadores de recursos para proporcionar reservas de diversos tipos de recursos, não apenas processadores, para usuários da Grade. Isto se difere da proposta do GridSim, que reserva processadores para tarefas, ao invés de recursos para usuários. Além disto, as reservas propostas pelo mesmo não apresentam os atributos necessários para torná-las flexíveis, nem suporte para reservas paciais, o que o torna inadequado para

os testes.

Tendo visto que o GridSim e os demais simuladores estavam muito longe da proposta deste trabalho, foi implementada uma aplicação para os testes. Esta utilizou os algoritmos SUVS e SUVS-P para escalonamento, verificando como se comporta o escalonamento com variações do número de reservas agendadas e dos limites das mesmas. Os testes não consideraram a interação do usuário com o gerenciador de reservas, mas sim a questão de escalonamento de reservas, e por isso os testes foram realizados em um único computador, sem considerar-se latência de rede.

Na aplicação utilizada para os testes, foram utilizadas duas entidades que interagem entre si. A primeira entidade é a que gera requisições de reserva e as envia. A segunda entidade representa o gerenciador de reservas, que tem como funções receber as requisições de reservas, escaloná-las e devolver uma resposta. As entidades foram implementadas como objetos. As seguintes classes de objetos compuseram a aplicação:

- **Requisitor**: é a classe que representa a entidade que requisita as reservas. Seus atributos são o número de requisições e a taxa de envio das mesmas, e os valores de duração, limite para iniciar e para terminar e capacidade de cada requisição de reserva.
- **GerenciadorDeReservas**: é a classe que implementa a entidade que gerencia as reservas. Nela são estabelecidos os atributos de capacidade máxima que pode ser reservada e duração máxima das reserva.
- **SUVS**: é a classe que implementa o algoritmo SUVS.
- **SUVSP**: é a classe que implementa o algoritmo SUVS-P.
- **Requisicao**: é a classe que tem como atributos os parâmetros usados pelo objeto Requisitor ao fazer uma requisição.
- **Reserva**: é a classe que representa uma reserva e seus atributos. Cada objeto desta classe é construído a partir de um objeto Requisicao, possuindo assim seus atributos. Ainda há dois atributos que são obtidos no momento do escalonamento: o efetivo início e término da reserva.
- **Recurso**: classe onde as informações sobre o recurso são armazenadas.
- **VS**: classe que implementa os Variable Slots.

- **RespostaDaRequisicao**: classe que guarda as informações dos testes relativas ao escalonamento das requisições.

A aplicação foi desenvolvida uma aplicação em Java 6 (também conhecido como Java 1.6). Além disso, os testes foram realizados em um computador com processador Atlon-XP de 1,5 Ghz, 1 GB de memória RAM e sistema operacional Linux. A distribuição de Linux utilizada foi o Kubuntu, versão Dapper (6.06), e o kernel (núcleo) utilizado foi o 2.6. O ambiente de programação utilizado foi o Eclipse.

5.2 Resultados

Através desta aplicação, sendo executada com diversas variações de atributos, foi possível coletar alguns dados estatísticos referentes ao funcionamento dos algoritmos SUVS e SUVS-P. Os gráficos das figuras 5.1 e 5.2 apresentam resultados obtidos.

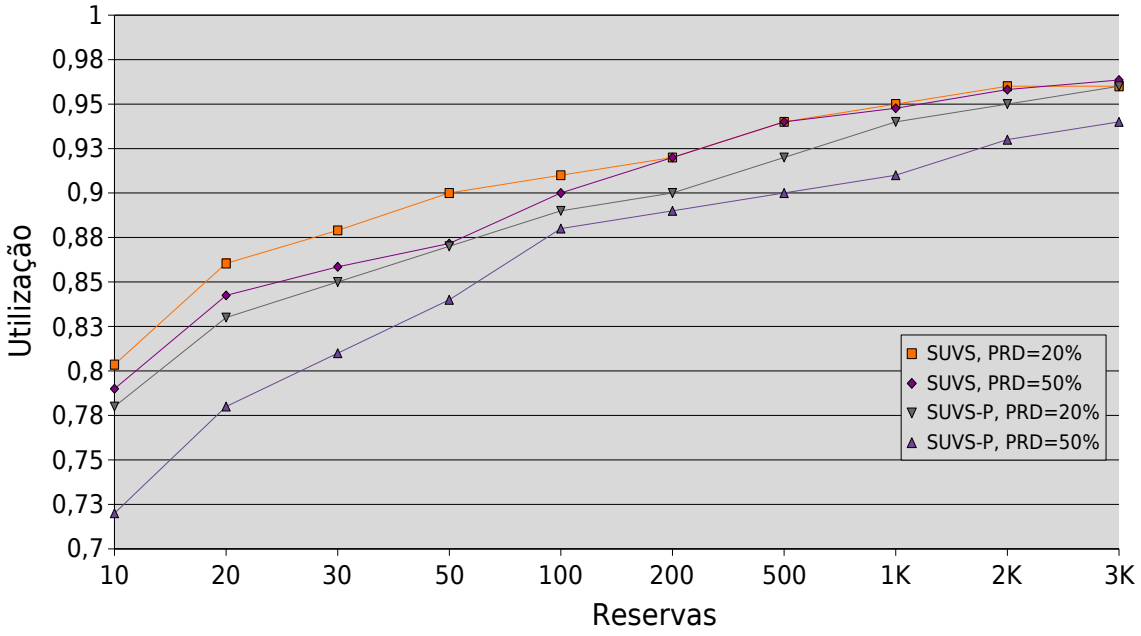

Figura 5.1: Impacto do número de reservas e PRD na utilização do recurso.

A figura 5.1 apresenta um gráfico que representa os valores obtidos nos testes dos algoritmos SUVS e SUVS-P, onde se considerou a possibilidade de variar a porcentagem de reservas que possuem o limite pala terminar finito e definido (Porcentagem de Reservas com Deadline ou simplesmente PRD) iguais a 20% e 50%. Além disso, para diferentes números de reservas agendadas, a utilização do recurso era diferente. Pelo gráfico, é

possível perceber que, na medida que o número de reservas agendadas aumenta, aumenta a utilização do recurso. Este comportamento já era esperado, uma vez que, quanto mais se requisita um recurso, mais tempo ele passará ocupado e, portanto, será mais utilizado. Além disso, quanto maior a chegada de requisições de reservas com diferentes durações e capacidades, maior a chance de que os "buracos" na agenda de reservas sejam preenchidos.

Quanto aos valores de PRD, é possível perceber que, em um ambiente com mais reservas com limite para terminar definido, há menos opções de lugares na agenda para encaixá-la. Assim, estas reservas não fazem tão boa utilização dos recursos.

Outro fator importante é a questão do algoritmo utilizado. SUVS-P teve uma taxa menor (portanto pior) de utilização que SUVS. Isso ocorreu porque, ao ajustar as reservas com limite para terminar como prioritárias, elas deslocam outras reservas, tomando seu lugar. Como estas reservas não se encaixam tão bem quanto as sem limite para terminar finito, este mecanismo de prioridades acaba reduzindo a utilização média do recurso.

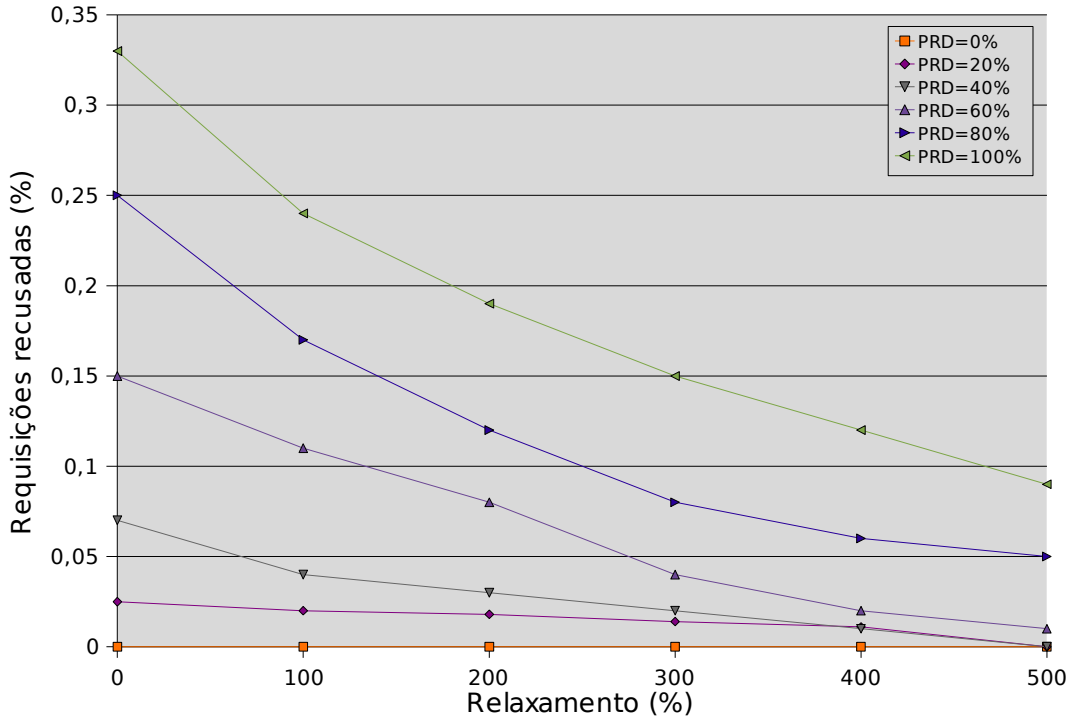

Figura 5.2: Impacto de relaxamento e PRD na taxa de requisições recusadas.

A figura 5.2 apresenta, por outro lado, o estudo do uso de diferentes valores de relaxamento. Neste exemplo, focou-se no algoritmo SUVS. Foram utilizados os valores de 0% a 500% (a diferença entre os limites foi de zero a seis vezes maior do que a duração) e, quanto maior o valor de relaxamento, menor a quantidade de reservas recusadas (e

conseqüentemente maior utilização do recurso). É importante ressaltar que os valores de relaxamento apresentados foram utilizados apenas em reservas com limite para terminar definido, cuja proporção é representada pelo valor PRD. Por isso, é natural que a variação do relaxamento faça menos diferença nos casos onde PRD é baixo, e não faça diferença onde PRD é zero.

Outro fator importante é a questão de diferentes valores de PRD. Cada valor, variando de 0% a 100%, é representado por uma curva. Desta forma, é possível perceber que, quanto maior a quantidade de reservas com limite para terminar definido, maior a quantidade de reservas recusadas. O pior caso é com todas as reservas com limite para terminar e sem relaxamento, pois neste caso, um terço das reservas foram recusadas. O melhor caso é com PRD=0%: já que não há limite máximo de tempo para a reserva acabar, é sempre possível agendar este tipo de reserva pois, no pior caso, será agendada para acontecer após todas as outras já agendadas.

6 Conclusões e Trabalhos Futuros

Grades permitem o compartilhamento de recursos distribuídos e heterogêneos. Para que isto seja possível, é necessário que se tenha uma estrutura de software capaz de gerenciar recursos. A questão se torna ainda mais importante quando se trata *da Grade*, ou seja, o que se espera das tecnologias de Grade para o futuro, onde haverá uma Grade global capaz de permitir que os mais diversos recursos e serviços sejam compartilhados.

Para *a Grade*, a gerência de recursos deve levar em conta QoS. Muitas aplicações de Grade só podem ser realizadas caso sejam utilizados recursos com algumas características, ou parâmetros de qualidade, como disponibilidade, confiabilidade, acesso ininterrupto, entre outros. Muitos trabalhos têm proposto prover isso através de reservas de recursos, onde certos recursos ou certa porcentagem da capacidade de um recurso são dedicados exclusivamente para um usuário requisitante, para uso ininterrupto por um intervalo de tempo. O uso de reservas traz a necessidade de gerenciamento das mesmas.

Este trabalho aborda a questão de gerenciamento de reservas de recursos de Grade. São trabalhados diferentes tipos de reservas: reservas imediatas, antecipadas e sob-demanda, assim classificadas pelo critério de tempo de utilização em relação ao tempo de requisição da reserva; e reservas simples, múltiplas e parciais, pelo critério de capacidade reservada. Além disso, são discutidas formas de se gerenciar estes tipos de reservas.

Este trabalho propõe, no capítulo 3, uma *arquitetura* que enfatiza a importância de uma entidade chamada de "gerenciador de reservas", e uma *modelagem* que utiliza empacotamento em fita para representar recursos e reservas, para que se tenha um escalonamento eficiente, abrangente, flexível e com melhor utilização dos recursos. A flexibilidade é suportada pela definição dos atributos das reservas e suporte a relaxamento. A abrangência é trabalhada pelo suporte a reservas simples, parciais e múltiplas. A eficiência e utilização se dão pelo escalonamento.

No capítulo 4, são apresentados três algoritmos de escalonamento de reservas. Eles fazem uso da modelagem apresentada no capítulo 3. O primeiro deles escalona reservas

simples e múltiplas. Os outros dois, chamados de SUVS e SUVS-P, escalonam reservas parciais da capacidade de um recurso. Para os dois últimos foram criados os Variable Slots, que são estruturas usadas para organizar as reservas e melhorar o escalonamento. Foi dado também suporte à limitação de capacidade máxima agendada, por meio de um corte longitudinal na fita que representa o recurso a ser reservado. A diferença entre SUVS e SUVS-P é que o segundo utiliza um mecanismo de prioridade em que determinadas reservas, as que possuem restrições de tempo mais flexíveis, podem ser reagendadas para dar lugar a reservas de maior prioridade.

Nos testes realizados, porém, chegou-se à conclusão de que com SUVS o recurso é melhor utilizado que com SUVS-P. Os testes também mostraram que a utilização dos recursos aumenta com o aumento do número de reservas antecipadas, e que reservas com o atributo limite para terminar definido como infinito, bem como o uso de relaxamento, melhoram a utilização dos recursos e reduzem a quantidade de requisições de reservas recusadas. Embora os testes tenham sido úteis para coletar estas informações importantes a respeito do escalonamento de reservas, eles levaram em conta apenas a questão de escalonamento, deixando de lado os fatores de negociação e latência de rede.

Para trabalhos futuros, é possível citar algumas carências da gerência de reservas que ainda precisam ser supridas. Uma delas é o suporte a co-reservas, ou seja, reservas múltiplas de recursos heterogêneos de diferentes domínios ou organizações virtuais, que podem ser necessárias em casos de aplicações que utilizem uma quantidade muito grande de recursos. Outra é a questão de aplicações que geram grandes fluxos de tarefas, tal como *workflows*, que poderiam ser melhor suportados pelo uso de algoritmos off-line, pois se conheceria antecipadamente os diversos recursos necessários por cada tarefa. Outra sugestão de trabalho futuro é a elaboração de uma interface para facilitar a requisição de reservas, inclusive que permita a visualização das reservas já agendadas e da utilização dos recursos. Outra carência a ser suprida é a inexistência de simuladores de Grade capazes de realizar um gerenciamento de reservas mais abrangente, como este trabalho propõe. Por fim, segue a sugestão de um estudo de testes exaustivos que visem dar uma visão mais quantitativa do gerenciamento de reservas, inclusive utilizando os algoritmos propostos e tipos de reservas apresentados neste trabalho.

Referências Bibliográficas

BRUNETT, S. et al. Application experiences with the globus toolkit. In: *HPDC '98: Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 1998. p. 81. ISBN 0-8186-8579-4.

BURCHARD, L.-O. On the performance of computer networks with advance reservation mechanisms. In: *ICON '03: Proceedings of the 11th IEEE International Conference on Networks*. Sydney, Australia: IEEE, 2003. ISBN 0-7803-7788-5. ISSN 1531-2216.

BURCHARD, L.-O. *Advance Reservations of Bandwidth in Computer Networks*. Tese (Doutorado) — Berlin University of Technology, aug 2004.

BURCHARD, L.-O. Analysis of data structures for admission control of advance reservation requests. *IEEE Transactions on Knowledge and Data Engineering*, v. 17, n. 3, 2005.

BURCHARD, L.-O. Networks with advance reservations: Applications, architecture, and performance. *Journal of Network and Systems Management*, Kluwer Academic Publishers, v. 13, n. 4, p. 429–449, dec 2005.

BURCHARD, L.-O.; DROSTE-FRANKE, M. Fault tolerance in networks with an advance reservation service. In: JEFFAY, K.; STOICA, I.; WEHRLE, K. (Ed.). *11th International Workshop on Quality-of-Service (IWQoS)*. Berkeley, CA, USA: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v. 2707), p. 215–230. ISBN 3-540-40281-0.

BURCHARD, L.-O.; HEISS, H.-U.; ROSE, C. A. F. D. Performance issues of bandwidth reservations for Grid computing. In: *SBAC-PAD '03: Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*. Washington, DC, USA: IEEE Computer Society, 2003. p. 82–90. ISBN 0-7695-2046-4.

BURCHARD, L.-O. et al. The Virtual Resource Manager: An architecture for SLA-aware resource management. In: *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2004. p. 126–133. ISBN 0-7803-8430-X.

BURCHARD, L.-O.; LINNERT, B. Failure recovery in distributed environments with advance reservation management systems. Springer-Verlag, Germany, v. 3278, p. 112–123, 2004.

BURCHARD, L.-O. et al. A quality-of-service architecture for future Grid computing applications. In: *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 2*. Washington, DC, USA: IEEE Computer Society, 2005. p. 132. ISBN 0-7695-2312-9.

BURCHARD, L.-O.; LINNERT, B.; SCHNEIDER, J. A distributed load-based failure recovery mechanism for advance reservation environments. In: *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2005. v. 2, p. 1071–1078. ISBN 0-7803-9074-1.

BURCHARD, L.-O. et al. VRM: A failure-aware Grid resource management system. In: *SBAC-PAD '05: Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*. Washington, DC, USA: IEEE press, 2005. p. 218–225. ISBN 0-7695-2446-X.

BURCHARD, L.-O.; SCHNEIDER, J.; LINNERT, B. Rerouting strategies for networks with advance reservations. In: *First IEEE International Conference on e-Science and Grid Computing (e-Science 2005)*. Melbourne, Australia: IEEE CS Press, 2005. p. 446–453. ISBN 0-7695-2448-6.

BUYYA, R.; MURSHED, M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. Wiley Press, v. 14, p. 1175–1220, 2002. ISSN 1532-0626.

CASTILLO, C.; ROUSKAS, G. N.; HARFOUSH, K. On the design of online scheduling algorithms for advance reservations and QoS in Grids. In: *21th International Parallel and Distributed Processing Symposium (IPDPS 2007)*. Long Beach, California, USA: IEEE, 2007. p. 1–10.

CZAJKOWSKI, K. et al. A resource management architecture for metacomputing systems. In: *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1998. v. 1459, p. 62–82. ISBN 3-540-64825-9.

DECKER, J.; SCHNEIDER, J. Heuristic scheduling of Grid workflows supporting co-allocation and advance reservation. In: SCHULZ, B. et al. (Ed.). *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid07)*. Washington, DC, USA: IEEE CS Press, 2007. p. 335–342. ISBN 0-7695-2833-3.

ELMROTH, E.; TORDSSON, J. A Grid resource broker supporting advance reservations and benchmark-based resource selection. In: DONGARRA, J.; MADSEN, K.; WASNIEWSKI, J. (Ed.). *PARA*. Germany: Springer-Verlag, 2004. (Lecture Notes in Computer Science, v. 3732), p. 1061–1070. ISBN 3-540-29067-2.

FAHRINGER, T. et al. ASKALON: A Grid application development and computing environment. In: *GRID'05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. Seattle, USA: IEEE Computer Society Press, 2005. p. 122–131. ISBN 0-7803-9492-5.

FAROOQ, U.; MAJUMDAR, S.; PARSONS, E. W. Impact of laxity on scheduling with advance reservations in Grids. In: *MASCOTS '05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. Washington, DC, USA: IEEE Computer Society, 2005. p. 319–324. ISBN 0-7695-2458-3.

FERRARI, D.; GUPTA, A.; VENTRE, G. Distributed advance reservation of real-time connections. In: *NOSSDAV '95: Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*. London, UK: Springer-Verlag, 1995. p. 16–27. ISBN 3-540-60647-5.

FERRARI, D.; GUPTA, A.; VENTRE, G. Distributed advance reservation of real-time connections. *Multimedia Systems*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 5, n. 3, p. 187–198, 1997. ISSN 0942-4962.

FOSTER, I. et al. A distributed resource management architecture that supports advance reservations and co-allocation. In: *Proceedings of the 7th IEEE International Workshop on Quality of Service (IWQoS'99)*. London, England: IEEE, 1999. ISBN 9780780356719.

FOSTER, I. et al. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. The Globus Alliance, feb 2002. Disponível em: <http://www.globus.org/research/papers/ogsa.pdf>. Acesso em: 30 mar 2007.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, Sage Publications, Inc., v. 15, n. 3, p. 200–222, 2001.

FOSTER, I. T. Globus toolkit version 4: Software for service-oriented systems. In: JIN, H.; REED, D. A.; JIANG, W. (Ed.). *Network and Parallel Computing: IFIP International Conference, NPC 2007, Dalian, China, September 18-21, 2007, Proceedings (Lecture Notes in Computer Science)*. Germany: Springer-Verlag, 2005. (Lecture Notes in Computer Science, v. 3779), p. 2–13. ISBN 3-540-29810-X.

HOO, G.; JOHNSTON, W. QoS as middleware: Bandwidth reservation system design. In: *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 1999. p. 23. ISBN 0-7695-0287-3.

KEAHEY, K.; MOTAWI, K. *The taming of the Grid: virtual application services*. Argonne National Lab., IL USA, 2004. 20 p.

KUNRATH, L.; WESTPHALL, C. B.; KOCH, F. L. *Towards Advance Reservation in Large-Scale Grids*. [S.l.]: IEEE Computer Society Press, 2008. To appear in *The Proceedings of The Third International Conference on Systems (ICONS 2008)*. IEEE Computer Society Press.

LIVNY, M.; RAMAN, R. High-throughput resource management. In: FOSTER, I.; KESSELMAN, C. (Ed.). *The Grid: Blueprint for a New Computing Infrastructure*. [S.l.]: Morgan Kaufmann, 1998. cap. 13.

NAHRSTEDT, K.; CHU, H.-H.; NARAYAN, S. QoS-aware resource management for distributed multimedia applications. *Journal on High-Speed Networking*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 7, n. 3-4, p. 229–257, 1999. ISSN 0926-6801.

ROEBLITZ, T.; SCHINTKE, F.; WENDLER, J. Elastic Grid Reservations with User-Defined Optimization Policies. In: *Proceedings of the Workshop on Adaptive Grid Middleware (AGridM'04)*. Washington, DC, USA: IEEE, 2004. v. 1.

SIDDIQUI, M.; VILLAZÓN, A.; FAHRINGER, T. Grid allocation and reservation - Grid capacity planning with negotiation-based advance reservation for optimized QoS. In: *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida*. New York, NY, USA: ACM Press, 2006. p. 103. ISBN 0-7695-2700-0.

SIDDIQUI, M. et al. Advanced reservation and co-allocation of Grid resources: A step towards an invisible Grid. In: *Proceedings of the INMIC 2005*. Karachi, Pakistan: IEEE, 2005. p. 1–6. ISBN 0-7803-9430-5.

SMITH, W.; FOSTER, I.; TAYLOR, V. Scheduling with advanced reservations. In: *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2000. p. 127–132. ISBN 0-7695-0574-0.

TOPCUOUGLU, H.; HARIRI, S.; WU, M. you. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, Piscataway, NJ, USA, v. 13, n. 3, p. 260–274, 2002. ISSN 1045-9219.

WIECZOREK, M. et al. Applying advance reservation to increase predictability of workflow execution on the Grid. In: *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2006. p. 82. ISBN 0-7695-2734-5.