# Engenharia de Tráfego Multi-Camada para Grades

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Daniel Macêdo Batista e aprovada pela Banca Examinadora.

Esto exemplar corresponde à redação final da Tese/Dissertação devidamente corrigida e defendida por: Daniel Macêdo Batista
e aprovada pela Banca Examinadora.
Campinas, 15 de Junho de 2006
COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Campinas, 23 de junho de 2006.

Prof. Dr. Nelson Luis Saldanha da Fonseca
Instituto de Computação, Unicamp
(Orientador)

Prof. Dr. Fabrizio Granelli
Department of Information and
Communication Technology, Università degli
Studi di Trento (Co-orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP**
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Batista, Daniel Macêdo
B319e Engenharia de tráfego multi-camada para grades / Daniel Macêdo
B32e Batista -- Campinas, [S.P. :s.n.], 2006.

Orientador : Nelson Luis Saldanha da Fonseca; Fabrizio Granelli
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Redes de computadores. 2. Computação em grade (Sistemas de computador). 3. Engenharia de tráfego. I. Fonseca, Nelson Luis Saldanha da. II. Granelli, Fabrizio. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Título em inglês: Multi-layer traffic engineering for grid networks

Palavras-chave em inglês (Keywords): 1. Computer networking. 2. Computational grids (Computer systems). 3. Traffic engineering.

Área de concentração: Sistemas de computação, Redes multimídia

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Nelson Luis Saldanha da Fonseca (IC-UNICAMP)
Prof. Dr. Edmundo Roberto Mauro Madeira (IC-UNICAMP)
Profa. Dra. Jussara Marques de Almeida (DCC-UFMG)

Data da defesa: 23/06/2006

Programa de Pós-Graduação: Mestrado em Ciência da Computação

## TERMO DE APROVAÇÃO

Tese defendida e aprovada em 23 de junho de 2006, pela Banca examinadora composta pelos Professores Doutores:

Jussara Marques de Almeida

**Profa. Dra. Jussara Marques de Almeida**
**DCC / UFMG.**

Edmundo R Mauro Madeira

**Prof. Dr. Edmundo Roberto Mauro Madeira**
**IC / UNICAMP**

**Prof. Dr. Nelson Luis Saldanha da Fonseca**
**IC / UNICAMP.**

Instituto de Computação
Universidade Estadual de Campinas

# Engenharia de Tráfego Multi-Camada para Grades

**Daniel Macêdo Batista**[1]

22 de maio de 2006

**Banca Examinadora:**

- Prof. Dr. Nelson Luis Saldanha da Fonseca
  Instituto de Computação, Unicamp (Orientador)
- Prof. Dr. Edmundo Roberto Mauro Madeira
  Instituto de Computação, Unicamp
- Prof. Dra. Jussara Marques de Almeida
  Departamento de Ciência da Computação, UFMG
- Prof. Dr. Flávio Keidi Miyazawa
  Instituto de Computação, Unicamp (Suplente)

[1]Auxílio Financeiro do CNPq processo 131590/2004-9

# Resumo

Grades são ambientes computacionais caracterizados pela heterogeneidade de recursos e dinamismo. Por serem ambientes dinâmicos, as grades precisam de processos que otimizem a execução das aplicações de forma também dinâmica. Tais processos devem detectar mudanças no estado da grade e tomar medidas para manter o tempo de execução das aplicações o menor possível. Existem diversas propostas de otimização dinâmica de aplicações em grades que visam atender essa necessidade através da migração de tarefas. Esta dissertação propõe uma metodologia que considera variações na disponibilidade dos hosts bem como no estado da rede. A metodologia proposta é baseada nos princípios gerais da engenharia de tráfego e atua em várias camadas da arquitetura Internet. Ela tem como objetivo minimizar o tempo de execução das aplicações e visa ser simples e independente, tanto da aplicação, quanto da grade. Os ganhos obtidos na execução de aplicações em grades com a utilização da proposta, *versus* a execução sem a mesma, são avaliados através de simulação com exemplos implementados usando o simulador de redes `NS-2`. Esta dissertação propõe também uma família de escalonadores baseados em programação inteira e em programação mista para o escalonamento de tarefas em grades que modelam o estado dos hosts bem como o da rede, sendo este o diferencial em relação às demais propostas na literatura.

# Abstract

Grids are dynamic and heterogeneous computing environments which require systematic methods for minimizing the execution time of applications. Such methods needs to detect changes on resource availability so that the execution time of applications can be kept low. The method introduced in this dissertation considers changes on the availability of hosts as well as on the availability of network resources. This method ressembles the Traffic Engineering for the Internet. It was validated via simulation using the `NS-2` simulator. This dissertation also introduces a set of schedulers based on integer and mix programming which considers both host availability as well as network resources availability, differing from other proposals in the literature.

*Ao meu pai José, à minha mãe Bernadete e ao meu irmão Joberson.*

# Agradecimentos

Todos aqueles que já passaram por uma pós-graduação sabem o quanto é difícil chegar ao final de uma dissertação, principalmente quando o curso não é realizado na cidade natal. Aprender a gerenciar o estudo das disciplinas, os prazos para submissão de artigos, as saudades de casa, entre tantas outras coisas do dia-a-dia com as quais não se está acostumado, não é nada trivial e só quem já viveu isso na prática sabe como é.

Assim que comecei o mestrado, eu imaginava que dois anos demorariam muito de passar, o que, no final das contas não ocorreu. Parece que ainda ontem eu estava deixando Salvador para enfrentar um novo desafio na minha vida, que foi morar longe da família em uma cidade onde eu nunca havia pisado. Eu acho que essa impressão de que o tempo passa mais rápido do que deveria ocorre sempre que estamos fazendo o que gostamos e com o apoio dos amigos, independente da distância em que eles se encontram. Desde que decidi que faria o mestrado no IC eu tive esse apoio e gostaria de agradecer a todos os que me ajudaram. Já peço desculpas antecipadas caso esqueça de alguém nas linhas que se seguem.

Em primeiro lugar, agradeço ao meu orientador, o prof. Nelson Fonseca, por me aceitar como orientando, por revisar todos os meus textos durante os últimos dois anos, dois meses e alguns dias, pela busca freqüente por apoio financeiro, não só para mim como para todos os seus outros orientandos, e claro, pela ótima orientação.

Agradeço ao meu co-orientador, o prof. Fabrizio Granelli, pelo apoio (mesmo que via e-mail) nas questões relacionadas com o simulador `GridSim-NS`.

Outros dois professores que merecem meus agradecimentos são o prof. Flávio Miyazawa, pela cooperação nos assuntos relacionados à programação linear, e o prof. Julio Lopez, pela confiança e pelas várias dicas passadas quando fui monitor da disciplina MC-102.

Agradeço ao povo da república, Cléo, Guido, Gustavo, Triste, e Bartho (mesmo o Bartho não sendo um pistolinha oficial) por aguentar a minha chatice ao dizer 'não' para praticamente todos os convites de ir em um barzinho ou no cinema, pelas várias sessões de divx e de Lost e pelas conversas sobre os mais diversos assuntos.

Agradeço aos amigos do IC que foram meus colegas em disciplinas, Bruno, Luiz,

Neumar, Thiago Coelho e Tiago Moronte, pelas várias discussões sobre as disciplinas ou não. Em especial, gostaria de agradecer àqueles mais próximos e com os quais estudei para as provas, fiz trabalhos de disciplinas e que até hoje estão disponíveis para conversar sobre os mais diversos assuntos, André Atanásio, Augusto e Leonardo.

Agradeço à Dani, que também foi monitora de MC-102, pelas várias ajudas com o `SuSy`, pela amizade que nasceu a partir daí, pelas diversas conversas filosóficas, pelas sessões de divx e por ter me ajudado a enxergar a ilusão deste mundo.

Não podia deixar de agradecer aos amigos de Salvador pela disponibilidade de irem tomar um sorvete na Ribeira sempre que eu estive lá, Joselino, Guedes e Saback. Em especial, agradeço àquela que não podia deixar de ser citada, Carlinha (fofinha!!!!) pelos momentos inesquecíveis durante a graduação, pela manutenção da amizade, pela confiança, pelos telefonemas, enfim, agradeço por ela existir e ter marcado tanto a minha vida.

Gostaria de agradecer a todo o pessoal da UFBA, em especial Claudete, Frieda e Gorgonio, pelo apoio dado em todos os projetos dos quais fiz parte durante a minha graduação e pelo incentivo para que eu fizesse o mestrado.

Agradeço aos meus pais, José e Bernadete, por todos os ensinamentos e pelo apoio dado a todas as minhas escolhas. Sem dúvida isso foi essencial para que eu chegasse aqui. Por mais repetitivo que isso seja, eles são os melhores pais do mundo. Agradeço ao meu irmão, Joberson, pela amizade entre nós, pelas várias conversas sobre filmes, jogos e seriados e pelo seu dom musical, que preenche de forma sublime o ambiente da nossa casa em Salvador. Gostaria de agradecer também à minha prima Ciene pelas visitas freqüentes lá em Salvador e pela sua belíssima voz.

Por último, mas não menos importante, agradeço ao povo brasileiro, pelo auxílio financeiro através da bolsa do CNPq, que permitiu que eu me mantivesse focado na pesquisa durante todo o mestrado.

*Together we stand, divided we fall.*

(Roger Waters)

# Sumário

# Lista de Tabelas

# Lista de Figuras

# Lista de Algoritmos

# Capítulo 1

# Introdução

Grades podem ser definidas como sistemas que coordenam recursos, não submetidos a um controle central, com o objetivo de prover qualidades de serviço às aplicações [18]. De um ponto de vista mais prático, grades são sistemas formados por vários recursos, não necessariamente dedicados, e que podem estar distribuídos em qualquer escala geográfica e em diferentes organizações, a fim de solucionar problemas que levariam um tempo muito grande para serem resolvidos pelos modelos de computação convencionais, ou que exigiriam uma configuração de hardware muito custosa de ser adquirida por uma única organização.

Os constantes aumentos na capacidade de transmissão das redes de longa distância ampliaram a largura de banda entre diferentes organizações e viabilizaram a implementação de grades. Isso fez com que as pesquisas sobre computação em grade intensificassem nos últimos anos, principalmente em centros de pesquisa relacionados com física de altas energias [30, 24, 48] e em centros de pesquisa aeroespacial [23, 15, 35], dada as diversas vantagens encontradas na utilização de grades para resolver problemas específicos desses ambientes. Dentre essas vantagens, destacam-se:

- Redução do custo, pois não há a necessidade da aquisição de hardware exclusivo para processamento paralelo;
- Grande chance de que uma aplicação leve menos tempo para concluir sua execução, já que há aproveitamento do tempo de ociosidade de recursos das organizações que fazem parte da grade. Esse aproveitamento amplia as opções de máquinas onde uma aplicação pode ser executada;
- Aumento, teoricamente ilimitado, do poder computacional da grade caso seja necessário, já que a mesma é construída em cima de protocolos padronizados e abertos, o que facilita a entrada de novos recursos.

Apesar de todas estas vantagens, a complexidade na implementação e na gerência de grades ainda impedem que as mesmas estejam disponíveis para que qualquer usuário, via Internet, execute todo tipo de aplicação. A grande maioria das grades concentra-se em grandes centros de pesquisa, ou grandes organizações, e exige uma equipe técnica especializada para serem gerenciadas, bem como usuários com conhecimento acima da média sobre o ambiente computacional utilizado.

Para que o uso das grades torne-se mais difundido, saindo dos centros de pesquisa e tornando-se tão acessível quanto a eletricidade, como previsto por alguns autores [21], há a necessidade de automatizar o funcionamento das mesmas. Com a automatização do funcionamento e gerência das grades, será possível a execução de uma grande variedade de aplicações, sem que os usuários precisem ter conhecimento da complexidade necessária para isso.

O ponto-chave para que as grades sejam automatizadas diz respeito à alocação eficiente dos recursos para as aplicações, já que assim como em outros ambientes baseados em processamento paralelo, as aplicações para grades são definidas como um conjunto de tarefas. Essa alocação é a responsável pela diminuição do tempo de execução das aplicações, pois passa a existir a possibilidade de várias tarefas serem executadas ao mesmo tempo em recursos distintos. A alocação dos recursos para as tarefas de uma aplicação é realizada através de um mapeamento, que associa cada tarefa a um recurso específico, e de um escalonamento, que define o instante de tempo em que cada tarefa deve iniciar sua execução. Pelo fato das grades serem compostas por recursos heterogêneos, a busca pelo melhor escalonamento torna-se mais complexa do que a busca em ambientes paralelos de menor escala, onde todos os recursos costumam ser dedicados, ter capacidade de processamento semelhante e serem interligados por enlaces exclusivos. Algoritmos existentes na literatura com o objetivo de escalonar tarefas em ambientes heterogêneos não são adequados às grades por considerarem que a heterogeneidade existe somente na capacidade de hosts compartilhados, deixando os enlaces de rede de lado. Apesar dessas considerações tornarem o ambiente longe do real, elas facilitam a busca pelo melhor escalonamento.

Outro problema enfrentado por técnicas de escalonamento de tarefas para grades é a mudança no estado dos hosts e dos enlaces de rede que os interligam. Mudanças no estado são causadas pelo fato das grades serem formadas por recursos não dedicados a uma única aplicação. Isso faz com que um escalonamento encontrado no instante atual possa não ser o melhor escalonamento em um instante futuro. Essa ineficiência deve ser detectada através de medições e justifica a migração de tarefas com o objetivo de minimizar o tempo de execução das aplicações. Os processos de medição e migração realizados a fim de otimizar a execução das aplicações em grades remetem ao objetivo da engenharia de tráfego para Internet, que propõe a utilização de técnicas que possibilitem a otimização do uso de recursos e suporte a QoS em uma rede IP. Pode-se dizer que

sem a utilização de técnicas de otimização dinâmica em grades, as mesmas constituem ambientes que executam aplicações baseados em um modelo de melhor esforço.

Há, na literatura, propostas que otimizam as execuções em grades de forma dinâmica através de migrações de tarefas quando são detectadas mudanças no estado dos recursos compartilhados. Apesar da rede ter um importante papel nas grades, ela é negligenciada por essas propostas e mudanças nos atributos dos enlaces não são consideradas como motivadores para a migração de tarefas ou, quando são, não possuem informações detalhadas sobre a vantagem alcançada com a migração.

À primeira vista, pode-se argumentar que com o aumento na capacidade dos enlaces de rede, o estado dos mesmos é um item que não precisa ser verificado por uma proposta para otimizar a execução de aplicações em grades, entretanto, mesmo enlaces com alta capacidade de transmissão podem ser sobrecarregados com as transferências de dados de certas aplicações. Algumas dessas aplicações foram apresentadas no evento iGRID2005, e foram responsáveis por aumentar de cinco a oito vezes a utilização dos enlaces de uma rede de pesquisa nos Estados Unidos [43].

A importância de considerar a rede no projeto da infra-estrutura das grades, e de automatizar o funcionamento das mesmas, também são preocupações no projeto UltraLight [51, 36], que tem como um dos seus objetivos a inclusão da rede como um componente integrante nas grades, bem como na organização européia para pesquisa nuclear (*european organization for nuclear research* – CERN), como pode ser notado nas conclusões da apresentação disponível em [34]. Nesta referência, há a afirmação de que a rede é um componente central e crítico para as futuras implementações de grades que gerarão uma quantidade significante de dados.

Com o reconhecimento da importância de considerar a rede para o correto funcionamento das grades, o objetivo desta dissertação é apresentar uma proposta de otimização dinâmica de aplicações em grades que, além de considerar mudanças no estado dos hosts e a entrada e saída dos mesmos, considera as mudanças no estado da rede como motivador para migração de tarefas. A proposta, denominada engenharia de tráfego para grades, utiliza técnicas semelhantes às utilizadas na engenharia de tráfego para Internet. Tenta-se garantir a execução de tarefas no menor tempo possível, sem intervenção humana e sob constantes mudanças nos estados dos hosts compartilhados e dos enlaces que os interligam.

Outra contribuição desta dissertação é a formulação de diferentes escalonadores de tarefas baseados em programação linear e em seqüências de sorteios. Esses escalonadores podem ser utilizados no processo de engenharia de tráfego. Os escalonadores diferenciam-se em termos da qualidade do escalonamento encontrado bem como do tempo necessário para derivar tais escalonamentos, o que os torna úteis para grades que executem aplicações com limites temporais.

Em resumo, as contribuições desta dissertação são:

- Uma metodologia para escalonar e migrar tarefas em grades, sem intervenção do usuário e independente da aplicação;
- Um conjunto de escalonadores de tarefas que atende a diferentes requisitos temporais das aplicações;
- Discussão de como a diminuição no tempo de execução afeta a qualidade do escalonamento, realizada através da análise do *trade-off* entre qualidade e tempo de execução do conjunto de escalonadores;
- Comprovação da importância de se considerar os recursos da rede nas decisões de alocação de recursos para execução.

Enfatiza-se que as contribuições da dissertação podem levar as grades a alcançarem a ubiqüidade prevista pelos cientistas da Computação e esperada pelos setores da sociedade que demandam computação confiável de alto desempenho.

Os próximos capítulos estão organizados da seguinte forma: no Capítulo 2 são apresentados os conceitos básicos de grades e informações sobre algumas grades existentes ao redor do mundo. Informações gerais sobre escalonamento de tarefas, bem como uma revisão bibliográfica de escalonadores voltados para ambientes heterogêneos, são apresentadas no Capítulo 3. O Capítulo 4 apresenta detalhes sobre os principais passos realizados na otimização dinâmica de aplicações em grades e descreve técnicas disponíveis na literatura. No Capítulo 5, são propostos oito escalonadores de tarefas em grades. Cada um deles é voltado para diferentes requisitos de tempo de execução e qualidade do escalonamento encontrado. Os experimentos realizados e os resultados obtidos com a utilização dos escalonadores propostos também são apresentados nesse capítulo. No Capítulo 6, a proposta de otimização dinâmica é apresentada e os seus passos principais são detalhados. Ainda nesse capítulo, é detalhado o funcionamento do `GridSim-NS`, um simulador de grades desenvolvido sobre o `NS-2` [16], e são apresentados os experimentos realizados e os resultados obtidos com a utilização da engenharia de tráfego para grades. O Capítulo 7 finaliza a dissertação com as conclusões sobre os resultados dos experimentos realizados e os trabalhos futuros.

## 1.1 Publicações realizadas

A partir dos resultados alcançados com experimentos do conjunto de escalonadores, publicou-se os artigos [5] e [6].

Já com relação aos experimentos da metodologia de Engenharia de Tráfego por completa, a partir dos resultados alcançados publicou-se o artigo [4].

# Capítulo 2

# Computação em grade

O desejo de tornar a computação tão acessível quanto a energia elétrica é um grande desafio para a Ciência da Computação. Esse desejo começou a se tornar realidade com o advento da computação em grade, que surgiu como uma solução para resolver problemas computacionais complexos em um intervalo de tempo não alcançado em outros ambientes de alto desempenho.

O termo "grade" surgiu no final dos anos 90 para definir os sistemas distribuídos formados por organizações interligadas através de redes de longa distância [20]. O objetivo desses sistemas era permitir a execução de aplicações distribuídas em larga escala. Apesar do objetivo inicial ter sido a execução de aplicações em larga escala, a utilização do termo "grade" demonstrou o desejo de que o acesso a esse ambiente fosse análogo ao acesso à grade do sistema elétrico: fácil, largamente disponível e padronizado como plugar um eletrodoméstico na tomada [10].

A cooperação das várias organizações em uma grade se dá através do compartilhamento de recursos particulares de cada uma delas. Esses recursos podem ser compartilhados para fornecer os mais diversos serviços para as aplicações: processamento, armazenamento, rede, acesso a hardwares raros, bancos de dados, entre outros.

Apesar da clara vantagem de ter um ambiente de alto desempenho através da interligação de recursos já existentes, as grades não trazem vantagens para todos os tipos de aplicação e organizações, como explicado em [46]. As grades são realmente eficientes para organizações com aplicações paralelas que não possuem uma demanda previamente conhecida. A eficiência vem do fato de que, à medida que for necessário, os recursos da grade podem ser alocados a depender de cada uma das aplicações a fim de minimizar o tempo de execução. Se a demanda é previamente conhecida para as aplicações de uma organização, o ideal para ela é a aquisição de um ambiente paralelo dedicado para as computações paralelas.

Este capítulo fornece explicações sobre a composição e o funcionamento das grades a

fim de contextualizar o restante da dissertação. Ele está organizado da seguinte forma: a Seção 2.1 explica o que são organizações virtuais e as utiliza para exemplificar o funcionamento das grades na prática. Na Seção 2.2, é apresentada uma arquitetura em camadas descrevendo as grades e uma classificação para os vários tipos de grades existentes. A Seção 2.3 explica como as grades são gerenciadas e utilizadas na prática através de *middlewares*. A Seção 2.4 foca na dependência que as grades têm da rede para alcançar os seus objetivos e, finalizando o capítulo, a Seção 2.5 apresenta alguns projetos de grades existentes atualmente.

## 2.1 Organizações virtuais

Uma organização virtual (*Virtual Organization* – VO) é definida, de forma simplificada, como um conjunto de participantes com vários relacionamentos entre si e que desejam compartilhar recursos para realizar alguma tarefa [10]. Os participantes de uma VO são, na maioria das vezes, organizações reais que desejam, através da união com outras, minimizar o tempo de execução de alguma aplicação graças ao compartilhamento de recursos. A definição de VO remete à definição de grade. Enquanto as VOs desejam compartilhar seus recursos, as grades possibilitam os compartilhamentos através da criação e manutenção de VOs sob demanda e da gerência da infra-estrutura necessária.

O compartilhamento entre os participantes das VOs está relacionado com os serviços disponibilizados pelos recursos das grades citados anteriormente. Esse compartilhamento deve ser altamente controlado, com os fornecedores e os consumidores de recursos definindo claramente o que está compartilhado, quem está autorizado a acessar os recursos e as condições sob as quais os compartilhamentos podem ocorrer [19]. O controle é necessário para evitar que as políticas específicas de cada organização sejam desrespeitadas pelas VOs.

Uma característica importante dos recursos compartilhados é que eles não são necessariamente exclusivos da VO. Assim como um recurso pode ser utilizado por mais de uma VO ao mesmo tempo, ele também pode ser utilizado para a execução de uma aplicação interna da organização real. Por essa característica, as VOs são caracterizadas como ambientes dinâmicos, já que não há garantia de que um dado recurso permanecerá na VO durante todo o tempo necessário para executar uma aplicação, e nem de que ele apresentará uma taxa constante para o processamento de aplicações executadas na VO.

Um exemplo que resume a relação entre organizações reais, VOs e as grades pode ser visto na Figura 2.1. Nessa figura é apresentado um diagrama de três organizações reais que compartilham seus recursos (acesso direto a computadores e dados) e compõem duas VOs. Cada VO é criada para resolver um problema específico e é independente da outra, o que é percebido pelos limites envolvendo os recursos de cada uma. Uma grade pode

conter várias VOs, como pode ser visto pelo quadro mais externo que representa a grade formada pelas três organizações reais. Cada elipse na figura limita as políticas internas da organização real que ela envolve. Políticas essas que devem ser respeitadas pelas VOs.

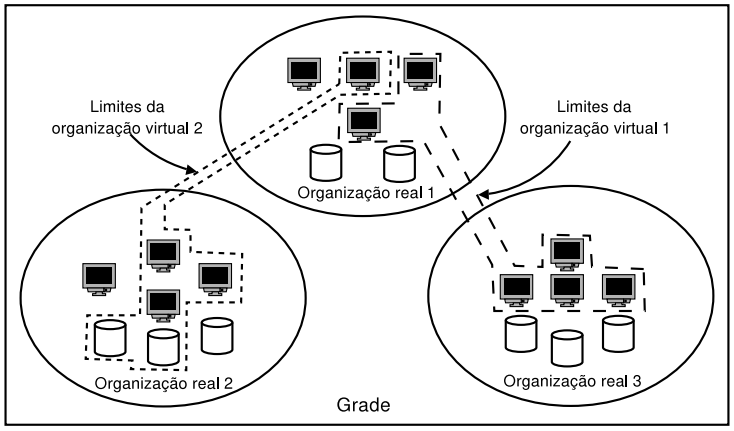



Figura 2.1: Organizações reais, organizações virtuais e grades

Ainda na Figura 2.1, não há a representação das ligações físicas via rede entre os vários recursos. Essa ausência deve-se ao fato da grade ocultar os detalhes de infra-estrutura para as VOs (Nesse exemplo, a rede não constitui um recurso da grade, mas sim um detalhe de infra-estrutura que não precisa ser conhecido pelos participantes).

Como as VOs podem compartilhar os mais diversos recursos, as mesmas são caracterizadas como ambientes heterogêneos. Essa heterogeneidade, e o dinamismo citado anteriormente, são herdados pelas grades e são responsáveis pela complexidade da gerência das grades.

De forma resumida, o papel das grades no controle das VOs, além de garantir o respeito às políticas internas de cada organização real, é fornecer protocolos, serviços e ferramentas que garantam a escalabilidade das VOs. Essas garantia e fornecimentos fazem com que a complexidade necessária para as ligações virtuais em uma VO seja ocultada para as aplicações e para os usuários, que podem acessar os recursos independente da localidade dos mesmos.

## 2.2 Arquitetura de redes em grades

Antes de classificar as grades, é importante analisar uma arquitetura na qual todos os tipos de grades sejam inseridos. Em [19], é apresentada uma arquitetura em camadas para descrever as grades. Essa arquitetura pode ser vista na Figura 2.2, mapeada na arquitetura Internet, já que as duas definem camadas que envolvem desde a rede até a aplicação.

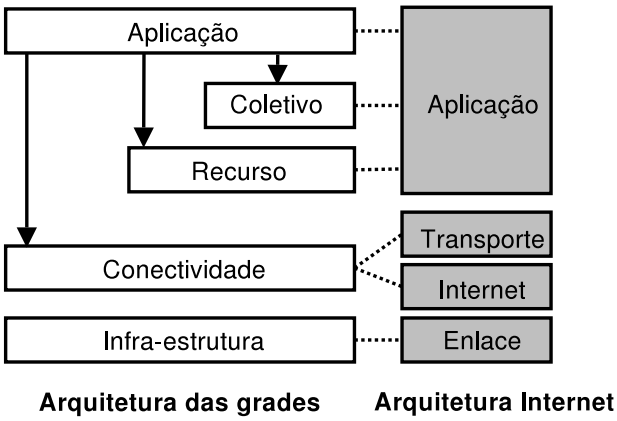



Figura 2.2: Arquitetura das grades e arquitetura Internet (Traduzida de [19])

Assim como na arquitetura Internet, as camadas inferiores na arquitetura das grades fornecem serviços para as camadas superiores. É graças a essa separação de tarefas que os detalhes das camadas inferiores são ocultados e as aplicações que executam nas VOs não precisam se preocupar com os detalhes da infra-estrutura.

A seguir, é apresentado um breve resumo da função de cada uma das camadas da arquitetura das grades.

- **Infra-estrutura:** Responsável por interligar os recursos compartilhados entre si no nível mais baixo. Nesta camada devem ser implementados os protocolos e as operações locais necessários para permitir o acesso e o monitoramento de cada recurso particular. Por exemplo, para o acesso a recursos de processamento, devem ser implementados mecanismos que garantam o uso justo da capacidade ociosa dos processadores e que possibilitem a consulta referente à taxa de processamento disponível;

- **Conectividade:** Nesta camada são definidos os protocolos para comunicação e autenticação requeridos para que dados sejam transmitidos com segurança entre os recursos de uma VO. Os protocolos já existentes e utilizados pela arquitetura Internet como ICMP, IP, UDP e TCP vêm sendo utilizados para as comunicações em grades. Isso não significa que eles sejam as melhores opções, como pode ser visto em [17], onde é demonstrada ineficiência do protocolo TCP para certos tipos de grades;

- **Recurso:** Esta camada é responsável por servir de intermediário entre a aplicação e os recursos individuais, com a utilização dos serviços fornecidos pela camada conectividade. Os protocolos implementados nesta camada estão relacionados com

os objetivos da camada de infra-estrutura e dividem-se em duas categorias: informação, que obtêm informações a respeito do estado de cada recurso, e gerência, que negociam o acesso ao recurso através da análise dos requisitos das aplicações, dos serviços e do estado de cada recurso compartilhado;

- **Coletivo:** Camada que possui protocolos e serviços destinados a realizar consultas e alocações de um conjunto de recursos, ao invés de recursos individuais como é feito na camada anterior. Esta camada implementa protocolos que permitem a seleção de um conjunto de recursos a partir dos requisitos de uma aplicação. Esta camada permite, por exemplo, que o usuário em uma VO solicite a alocação de um conjunto de recursos e o escalonamento da aplicação neles a fim de executar uma aplicação no menor tempo possível;

- **Aplicação:** É a camada onde são implementadas as aplicações finais a serem executadas em uma VO controlada pela grade. Apesar dela estar posicionada imediatamente acima da camada coletivo, ela pode ter acesso direto aos serviços oferecidos pelas camadas conectividade e recurso. A utilização desse acesso depende das necessidades particulares de cada aplicação.

Essa arquitetura resume as funções genéricas que devem ser implementadas por uma grade. No entanto, cada grade pode apresentar soluções particulares para as VOs que ela gerencia. Essas particularidades estão relacionadas principalmente com o objetivo da grade implementada, que pode ser usado para classificar as grades em um número limitado de tipos.

Uma sugestão para classificar as grades de acordo com os seus objetivos é apresentada em [46]. Essa classificação define os quatro tipos de grades listados a seguir:

- **Grade computacional:** É a grade que representa a evolução dos sistemas paralelos e distribuídos. É formada por vários recursos que fornecem poder de processamento. As aplicações dos usuários são divididas em tarefas e executadas nos vários recursos disponíveis com o objetivo de minimizar o tempo de execução;

- **Grade de acesso:** É uma grade construída com o objetivo de interligar vários recursos de forma virtual, como se eles estivessem todos em um único computador. Um recurso pode fornecer armazenamento, outro pode fornecer acesso a um equipamento científico, enquanto um terceiro pode fornecer poder de processamento. Um usuário desta grade acessaria esses três serviços como se todos estivessem disponíveis no seu computador local;

- **Grade de dados:** O objetivo destas grades é permitir que grandes conjuntos de dados sejam manipulados entre repositórios com a mesma facilidade que pequenos

arquivos são transferidos. É uma grade híbrida dos tipos computacional e de acesso, onde o serviço a ser acessado é a manipulação dos dados;

- **Grade *Datacentric*:** Tais grades têm por objetivo permitir que sejam realizadas computações sobre grandes repositórios de dados distribuídos. Estas grades são implementadas de forma que as tarefas das aplicações sejam executados nos locais onde os dados estão armazenados e não o contrário.

As propostas para escalonar e otimizar as aplicações em grades, apresentadas nesta dissertação, levam em consideração as grades computacionais. Os serviços acessados pelas aplicações consideradas são o uso da taxa de processamento disponível nos hosts e o uso da banda disponível nos enlaces que interligam esses hosts.

## 2.3 Middlewares e escalonamento de tarefas

Para facilitar a implementação de aplicações para grades, assim como é feito em sistemas distribuídos, é necessário que haja um conjunto de serviços que cuide das necessidades comuns das várias aplicações. Esses serviços recebem o nome de serviços de *middleware* [7], ou simplesmente *middlewares*, e são localizados entre a rede e as aplicações (daí o termo *middle*, que significa "meio" em português).

Para as grades computacionais, uma função importante a ser realizada pelos *middlewares* é a alocação dos recursos e o escalonamento das tarefas de uma aplicação. As ações relacionadas com o escalonamento das tarefas podem ser resumidas nas três fases da Figura 2.3 [42].

A Fase 1 determina quais recursos da grade estão disponíveis a partir das permissões do usuário e dos requisitos mínimos da aplicação. Há uma filtragem de todos os recursos disponíveis e uma lista menor de recursos é gerada como saída nessa fase. A Fase 2 analisa os recursos fornecidos pela Fase 1 e define quais recursos executarão cada uma das tarefas. O ideal é que a escolha dos recursos seja feita objetivando-se minimizar o tempo de execução da aplicação. Finalmente, na Fase 3, as tarefas são executadas nos recursos que foram definidos na fase anterior e o resultado da aplicação é devolvido para o usuário.

Para as grades, os *middlewares* devem assumir as funções das camadas coletivo, recurso e conectividade, interligando dessa forma a aplicação à infra-estrutura.

## 2.4 Dependência da rede

A dependência que as grades em geral têm da rede fica evidente pela arquitetura em camadas. A rede deve fornecer os serviços de comunicação para as camadas superiores.

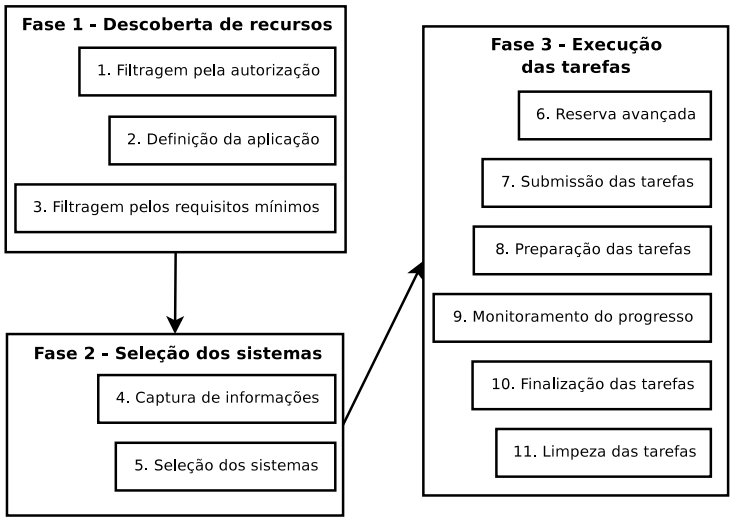



Figura 2.3: Fases do escalonamento (Traduzida de [42])

Um mau fornecimento desses serviços pode acarretar em um mau funcionamento de toda a grade. Em [31], a preocupação com a influência que o desempenho da rede exerce sobre as aplicações fica clara pela afirmação: "Pelo fato da rede fornecer os fios que conectam uma grade, entender o desempenho fornecido pela rede é crucial para alcançar desempenho satisfatório de muitas aplicações para grades" ("*Because the network provides the wires that connect a grid, understanding the performance provided by a network is crucial to achieving satisfactory performance from many grid applications*").

Nas Fases 2 e 3 da Figura 2.3 também fica evidente a dependência que o escalonamento das tarefas em uma grade tem da rede. Recursos cujos enlaces apresentem mais banda disponível são as melhores opções para tarefas que precisam transferir muitos dados. Além disso, o monitoramento do progresso da aplicação deve incluir o monitoramento dos enlaces que interligam os recursos. Enlaces que não são dedicados exclusivamente para a grade têm a banda disponível dependente de outras aplicações que utilizem a rede. Variações na banda desses enlaces podem justificar a migração de uma tarefa para outro recurso, a fim de minimizar o tempo total de execução.

Não é qualquer variação na banda disponível que justifica a migração de uma tarefa. Essa variação depende dos requisitos da aplicação sendo escalonada. Como apresentado em [31], aplicações voltadas para física de altas energias têm uma dependência da banda disponível bem maior do que as chamadas aplicações de computação otimista, que toleram maiores variações na capacidade de transmissão disponível dos enlaces.

A banda disponível nos enlaces não é a única característica da rede que deve ser considerada na gerência de uma grade. Métricas como atraso e *jitter* são exemplos de métricas que podem influenciar aplicações em grades. Grades de acesso, por exemplo, que

fornecem um "computador virtual" para os usuários, são exemplos de ambientes sensíveis a métricas como atraso e *jitter*, já que há uma maior interação por parte do usuário quando comparada com outros tipos de grades. A topologia de rede da grade também é uma característica importante que deve ser considerada, já que ela influencia diretamente a distribuição das aplicações pelos diversos recursos da grade.

A importância da rede na garantia da qualidade prometida pelas grades é considerada em [55] através da Figura 2.4, onde é possível notar que a gerência dos recursos disponibilizados por uma grade não consegue agir de forma eficiente sem o auxílio das tecnologias de QoS fornecidas pelas redes.

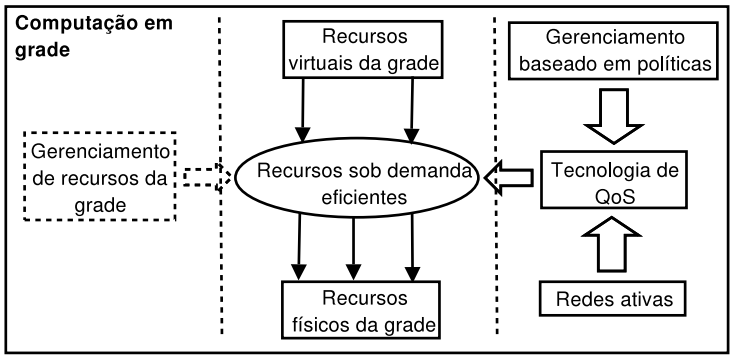



Figura 2.4: Grades e redes (Traduzida de [55])

Como as grades, por definição, são formadas por vários recursos interligados via enlaces não dedicados, o impacto das características da rede não pode ser deixado de lado como nas plataformas de computação paralela tradicionais, confinadas em redes locais com hosts e enlaces dedicados. Modelos e topologias de rede mais próximas do real devem ser usados na análise das soluções voltadas para grades. Experimentos de soluções para grades realizados com modelos e topologias de rede simples não refletem as grades no mundo real e devem ser evitadas na validação de propostas para gerenciamento de grades [11].

## 2.5 Projetos de grades existentes

Várias grades com o propósito de permitir avanços científicos nas mais diversas áreas do conhecimento têm sido projetadas e implementadas nos últimos anos. Muitas delas são formadas por hosts e equipamentos científicos distribuídos em diversos países e interligados por enlaces com capacidade de transmitir dados a uma taxa de até 10Gbps. Também são encontrados projetos de grades menores, implementadas dentro de uma organização específica. Os resultados que vêm sendo alcançados com as grades na prática comprovam

as vantagens em se utilizar tal ambiente computacional para processar e armazenar dados em larga escala.

A *National Aeronautics and Space Administration* (NASA) foi uma das primeiras organizações a propor tecnologias que permitissem a utilização de grades. A sua divisão de supercomputação desenvolveu uma infra-estrutura de computação heterogênea distribuída denominada *Information Power Grid* (IPG) [23]. O projeto da IPG utiliza *clusters* dedicados ao processamento de aplicações paralelas, bem como aproveita ciclos ociosos de estações de trabalho para acelerar a execução de aplicações das mais diversas áreas na NASA como dinâmica de fluídos, análise de dados capturados por telescópios e criação de modelos climáticos. Além de recursos computacionais convencionais, a IPG também interliga instrumentos e bases de dados distribuídos. A IPG é uma grade interna voltada para a solução de problemas computacionais da NASA, mas há a pretensão de uní-la a uma infra-estrutura de grades formada por outras instituições dos Estados Unidos.

O projetos InteGrade [27] e OurGrid [37] são dois projetos de pesquisa em grades desenvolvidos no Brasil que, assim como a IPG, visam aproveitar ciclos ociosos de diversas máquinas. Uma diferença entre a IPG e os dois projetos brasileiros é o fato destes já terem sido iniciados com o objetivo de interligar computadores de diversas organizações. Tanto o InteGrade quanto o OurGrid são projetos baseados na implementação de *middlewares* de código aberto e que podem ser copiados livremente dos sites [27] e [37]. Algumas características do *middleware* implementado no projeto InteGrade são o fato dele ser orientado a objetos e considerar os requisitos de qualidade de serviço, dos usuários locais, nos recursos compartilhados. Já o *middleware* implementado no projeto OurGrid caracteriza-se por utilizar a justiça no acesso aos recursos (participantes que fornecem mais poder de processamento, receberão mais quando necessário) e por ser voltado para aplicações formadas por tarefas sem dependências entre si.

Uma das áreas que mais demanda soluções para processamento de alto desempenho é a física de altas energias (*High Energy Physics* – HEP). A quantidade de informações geradas pelas colisões realizadas em aceleradores de partículas exige uma infra-estrutura com alta capacidade para armazenar e processar dados que podem chegar à ordem de PetaBytes. Dessa forma, as grades são ambientes ideais para atender os requisitos da comunidade de HEP. A implementação de uma grade foi a solução adotada pelo CERN para possibilitar o armazenamento e a análise dos dados que serão gerados pelo *Large Hadron Collider* (LHC), um acelerador de partículas que entra em funcionamento em 2007 e deve produzir 15PB de dados por ano. A grade implementada pelo CERN, denominada *LHC Computing Grid* (LCG) [30], é formada por 13 centros de pesquisa principais na Europa, Ásia e América do Norte, e por diversas organizações denominadas secundárias. Uma infra-estrutura de rede dedicada à grade interliga os 13 participantes principais e forma uma topologia estrela com enlaces de 10Gbps e com o CERN no centro. Apesar

do projeto inicial não incluir enlaces exclusivos para interligar 12 dos 13 participantes principais entre si (com exceção do CERN), a implantação dos mesmos é incentivada pelos responsáveis pela infra-estrutura da grade. O projeto da LCG também define que os participantes secundários devem ser interligados a alguns dos principais. Não há detalhes a respeito das ligações a estes participantes, o que significa que elas podem ser implementadas sobre redes de propósito geral, como a Internet. A grande quantidade de dados a serem acessados torna a rede um componente crítico da LCG, motivo pelo qual há um grupo específico para cuidar dos assuntos relacionados aos enlaces dedicados que interligam o CERN aos demais centros de pesquisa.

Dentre os vários experimentos realizados no LHC que utilizarão a LCG, há um experimento para detectar partículas chamado *Compact Muon Solenoid* (CMS). Alguns recursos computacionais para armazenar e analisar os dados gerados por esse experimento são disponibilizados pelo projeto *São Paulo Regional Analysis Center* (SPRACE) [48]. Além de fornecer recursos para o CMS, o SPRACE também fornece recursos computacionais para o experimento DØ, realizado no Fermilab, laboratório de pesquisas localizado nos Estados Unidos. Os recursos fornecidos pelo projeto são localizados em diversos centros de pesquisa de São Paulo e Rio de Janeiro, conectados entre si através de enlaces da rede GIGA com capacidade de 1Gbps. A interligação aos laboratórios no exterior é realizada através de um enlace de 622Mbps, compartilhado com outras instituições.

Recentemente, entrou em operação no Brasil a grade do Sistema Nacional de Processamento de Alto Desempenho (SINAPAD) [44]. Essa grade é formada pela interligação dos 7 Centros Nacionais de Processamento de Alto Desempenho (CENAPAD) espalhados pelo Brasil. Na primeira fase do projeto, que teve início em 15 de março de 2006, 4 desses 7 centros já estão interligados e compartilhando recursos para a grade em caráter experimental. A interligação dos centros para a grade utiliza enlaces da Rede Nacional de Ensino e Pesquisa (RNP), compartilhando a largura de banda disponível com as demais instituições do Brasil que são interligadas através da RNP. Diferente das grades citadas anteriormente, aplicações das mais diversas áreas podem utilizar a infra-estrutura fornecida pela grade do SINAPAD. De todas as áreas, aquelas que mais tem utilizado o poder computacional da grade até os dias atuais são as aplicações de física e de química.

Como pode ser notado com esses exemplos de grades reais, com exceção de grades construídas dentro de organizações para aplicações internas, a rede que interliga boa parte dos recursos não possui uma topologia conhecida a priori e não é dedicada somente à grade, o que torna necessário um sistema que monitore a rede e proponha escalonamentos de tarefas que levem em consideração o estado atual dos enlaces e que também levem em consideração a taxa de processamento disponível nos hosts compartilhados.

# Capítulo 3

# Escalonamento de tarefas em grades

Para as aplicações tirarem proveito de ambientes distribuídos como as grades, elas precisam ser decompostas em programas menores chamados de tarefas. Dessa forma, torna-se possível a diminuição do tempo de execução da aplicação graças à execução em paralelo de várias tarefas. As tarefas que compõem uma aplicação podem ou não ter dependências entre si. No primeiro caso, as tarefas formam grafos acíclicos direcionados (*Directed Acyclic Graphs* – DAG) para representar as dependências, enquanto que, no segundo caso, as tarefas formam o que é conhecido como *Bag-of-Tasks* (BoT).

Uma vez com a aplicação descrita, o usuário deve submetê-la à grade, que tem a responsabilidade de escalonar as tarefas. Além da descrição da aplicação a ser escalonada, o escalonador precisa de mais duas informações como entrada para devolver o escalonamento das tarefas: a descrição dos recursos compartilhados e o objetivo a ser alcançado com o escalonamento. A lista completa das informações, segundo [11], é a seguinte:

1. Modelo da aplicação a ser escalonada. O modelo deve especificar a estrutura da aplicação e os seus requisitos;

2. Modelo da plataforma. Este modelo deve especificar o estado dos hosts disponíveis e a rede que os interliga;

3. Objetivo que deve ser alcançado com o escalonamento: minimizar o tempo de execução, minimizar o custo, minimizar a probabilidade de falhas, etc.

Com as informações necessárias passadas como entrada, o escalonador deve devolver como saída os identificadores dos hosts nos quais cada tarefa será executada, e os instantes de tempo em que cada execução será iniciada. Porém, o escalonamento não é uma ação trivial, já que escalonar tarefas em sistemas formados por múltiplos recursos distribuídos, da melhor forma possível, é comprovadamente um problema NP difícil [38]. Devido a essa

complexidade, diversos escalonadores tiram proveito de particularidades da rede que interliga os hosts, a fim de diminuir a complexidade do problema e devolver escalonamentos próximos do ótimo em um intervalo de tempo curto. Algumas das particularidades consideradas são: custo de comunicação independente do enlace de rede, custo de comunicação desprezível e topologia de rede fixa [29].

Ambientes paralelos que não possuem particularidades como as mencionadas são classificados como ambientes com processadores arbitrariamente conectados, ou ambientes computacionais heterogêneos. Esses ambientes exigem escalonadores de tarefas mais complexos pela quantidade de variáveis que precisam ser manipuladas. Devido a essa exigência, nem sempre consegue-se encontrar o melhor escalonamento para uma dada aplicação em um curto espaço de tempo. Por esse motivo, escalonadores para ambientes heterogêneos são, normalmente, baseados em buscas aleatórias guiadas por alguma técnica de otimização ou em heurísticas [50]. Tais buscas avaliam várias opções de hosts onde cada tarefa pode ser executada e seleciona uma delas, sem deixar de considerar as restrições da aplicação e as restrições dos recursos da grade. Os escalonadores propostos nesta dissertação, e apresentados no Capítulo 5, são baseados em buscas aleatórias guiadas.

Algoritmos de escalonamento para ambientes heterogêneos são de grande importância para o bom funcionamento das grades, já que as mesmas enquadram-se como ambientes heterogêneos. Além desse fato, os recursos das grades não são dedicados a uma única aplicação, o que gera variação na capacidade disponível dos hosts distribuídos e da rede, exigindo assim escalonadores que encontrem o escalonamento dentro de um intervalo de tempo máximo a fim de que, no momento de distribuição da aplicação para execução, as variações nos hosts e na rede não invalidem o escalonamento sugerido. Tal intervalo de tempo é particular para cada ambiente. Na realidade, até em um mesmo ambiente ele pode sofrer variações. O desconhecimento do intervalo de tempo máximo faz com que escalonadores para grades representem um grande desafio já que precisam encontrar um resultado próximo do ótimo, para um problema NP difícil e no menor tempo possível.

A qualidade do escalonamento devolvido para uma aplicação depende do objetivo levado em consideração. Na dissertação, é considerado que o objetivo do escalonamento é minimizar o tempo de execução da aplicação. Dessa forma, a qualidade do escalonador será medida pela diferença entre o instante de tempo em que a última tarefa termina sua execução e o instante de tempo em que a primeira tarefa inicia sua execução. Essa diferença é conhecida na literatura como "comprimento do escalonamento", ou *makespan*. Quanto menor o *makespan* devolvido por um escalonador, melhor é a sua qualidade.

As seções seguintes deste capítulo fornecem informações sobre os itens de entrada para os escalonadores e sobre escalonadores encontrados na literatura. A Seção 3.1 apresenta particularidades sobre o modelo usado para tarefas independentes. Detalhes sobre o

modelo usado para tarefas dependentes são apresentados na Seção 3.2. Nesta seção, também são apresentadas informações sobre como mapear modelos de aplicações formadas por tarefas independentes em modelos de aplicações formadas por tarefas dependentes. Na Seção 3.3, são apresentados detalhes sobre as descrições dos hosts da grade e da rede que os interliga. Na Seção 3.4, alguns escalonadores para grades, ou para ambientes com características semelhantes às das grades, encontrados na literatura são descritos e têm seus pontos fortes e fracos ressaltados.

## 3.1 Tarefas independentes

Aplicações para grades compostas por tarefas independentes são conhecidas na literatura como aplicações *Bag-of-Tasks* [12]. As tarefas dessas aplicações podem ser executadas em qualquer ordem sem que o resultado final da aplicação seja comprometido. O modelo de uma aplicação BoT pode ser passado para o escalonador como um grafo sem arestas. Cada vértice do grafo representa uma tarefa e tem peso referente ao peso computacional da tarefa representada. Ao contrário de modelos que representam aplicações para serem executadas em ambientes paralelos homogêneos, o peso computacional das tarefas não pode ser representado em unidades de tempo, pois uma tarefa pode levar um tempo diferente a depender do host ao qual ela seja escalonada. Uma opção de representação para os pesos das tarefas é a quantidade de instruções que precisam ser executadas por ela.

Na Figura 3.1, é apresentado o grafo de uma aplicação BoT com 10 tarefas. Cada vértice possui o identificador da tarefa representada. Os valores entre colchetes ao lado de cada identificador representam o peso da tarefa em quantidade de instruções (essa representação será utilizada no restante da dissertação para outros grafos). Como pode ser notado na figura, a independência entre as tarefas faz com que o grafo resultante possua 10 componentes conexas, cada uma contendo um único vértice.

Alguns exemplos de aplicações BoTs são aplicações para manipulação de imagens, mineração de dados e algoritmos para quebra de chaves criptográficas.

É importante observar que, nesta dissertação, não é levada em consideração a incerteza das informações a respeito da quantidade de instruções de cada tarefa. Assume-se que a quantidade de instruções de cada tarefa, na prática, corresponde exatamente ao valor representado pelo peso do vértice relacionado.

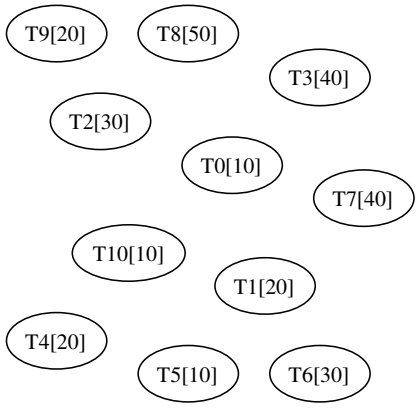



Figura 3.1: Grafo de uma aplicação com tarefas independentes

## 3.2 Tarefas dependentes

Aplicações para grades compostas por tarefas com dependências entre si costumam ser modeladas como grafos acíclicos direcionados (DAGs) [32]. Nesses DAGs, os vértices fornecem as mesmas informações dos vértices de aplicações BoTs. Os arcos representam as dependências de dados entre as duas tarefas e têm pesos referentes à quantidade de bytes que devem ser transmitidos. Um arco $ij$ no DAG define que a tarefa $j$ só pode iniciar a sua execução após a tarefa $i$ ter terminado de executar e após todos os dados de dependência, representados pelo peso do arco $ij$, terem sido transferidos do host onde $i$ executa para o host onde $j$ executa. Da mesma forma que os pesos dos vértices, os pesos dos arcos devem representar a quantidade de bytes, e não o tempo necessário para transmití-los, já que o tempo depende dos enlaces envolvidos na transmissão e que só são conhecidos após o escalonamento das tarefas ter sido realizado.

A Figura 3.2 ilustra o DAG de uma aplicação de astronomia [8] com 30 tarefas. Os pesos dos arcos são representados pelos valores entre colchetes (essa representação será usada no restante da dissertação para outros grafos).

Em um DAG, as tarefas sem arcos de entrada são chamadas de tarefas de entrada, enquanto que as tarefas que não possuem arcos de saída são chamadas de tarefas de saída. Na dissertação, é adotada a convenção de que todos os DAGs possuem uma única tarefa de entrada e uma única tarefa de saída. DAGs que possuam mais de uma tarefa de saída podem ser modificados de acordo com o procedimento descrito em [50] para atender esse requisito. O procedimento consiste na criação de uma tarefa artificial com peso zero e que dependa de todas as tarefas de saída originais. O peso dos arcos que ligam as tarefas de saída originais a essa nova tarefa artificial também deve ser zero. Processo semelhante pode ser utilizado para o caso em que há mais de uma tarefa de entrada.

Algoritmos voltados para o escalonamento de tarefas dependentes pode ser utilizado

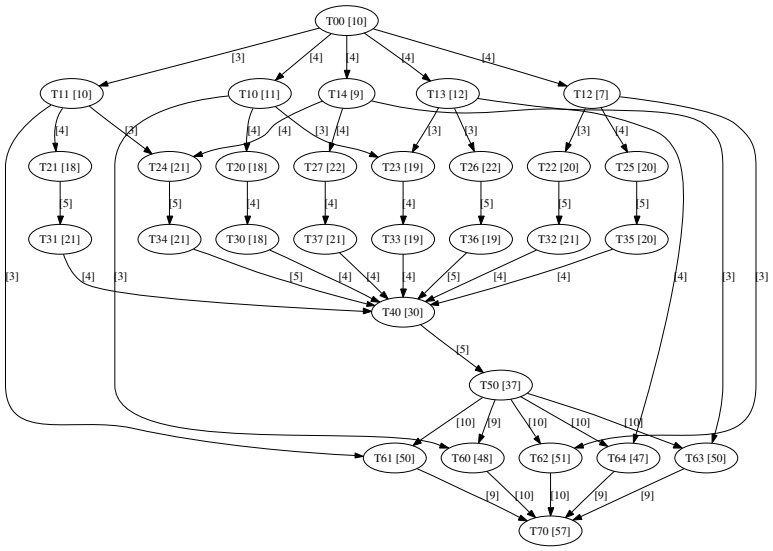



Figura 3.2: Grafo de uma aplicação com tarefas dependentes

para escalonar tarefas independentes através da transformação das tarefas independentes em um DAG. Essa transformação pode ser realizada seguindo o mesmo procedimento descrito para DAGs com mais de uma tarefa de entrada ou saída. Na Figura 3.3 é exibido o DAG da aplicação BoT da Figura 3.1. Nota-se que no DAG criado, todas as tarefas independentes dependem da tarefa artificial de entrada *T0x* e que a tarefa artificial de saída *T10x* depende de todas as tarefas independentes.

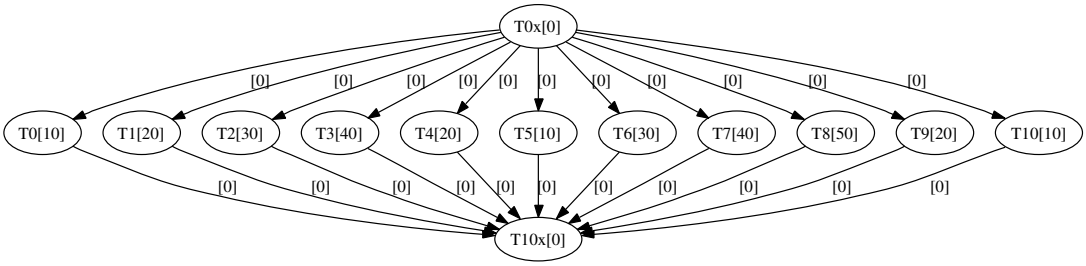



Figura 3.3: DAG criado para a aplicação da Figura 3.1

Também existem aplicações formadas por tarefas dependentes que são descritas por grafos direcionados que podem conter ciclos. As aplicações para grades levadas em consideração pelos escalonadores apresentados nesta dissertação são todas descritas por DAGs, ou seja, são formadas por tarefas com dependências entre si e cujo grafo não contém ciclos.

Alguns exemplos de aplicações descritas por DAGs são aplicações de astronomia, visualização de imagens de alta resolução que requerem muito processamento antes de serem exibidas e simulações dinâmicas de moléculas para pesquisas em química.

Assim como nos pesos dos vértices, nesta dissertação, não é levada em consideração a incerteza das informações a respeito da quantidade de bytes de dependência de cada arco.

## 3.3 Descrição dos recursos compartilhados

A descrição dos recursos compartilhados em uma grade deve fornecer informações atualizadas sobre o estado dos hosts bem como sobre o estado dos enlaces de rede que os interligam. Assim como as aplicações, os recursos da grade podem ser representados por um grafo. Nesse grafo, cada vértice representa um host com peso referente ao tempo gasto para executar uma certa quantidade de instruções. Cada aresta representa a existência de comunicação entre dois hosts, com um peso referente ao tempo necessário para transferir uma certa quantidade de bits. A Figura 3.4 representa o grafo com a topologia formada pelos recursos de uma grade com 13 hosts. As capacidades de transmissão e de processamento disponível estão representadas pelos valores entre colchetes na figura (essa representação será utilizada em todos os grafos de topologia apresentados nesta dissertação).

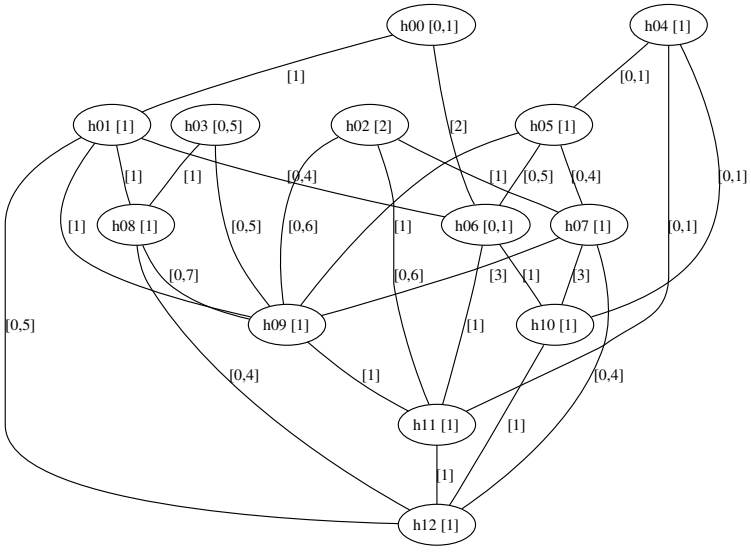



Figura 3.4: Grafo da topologia formada pelos recursos de uma grade

Considera-se que cada aresta do grafo da topologia representa a existência de enlaces que permitem transmissão *full-duplex* entre dois hosts, e que o peso de cada host corresponde exclusivamente à taxa de processamento disponível nele bem como que todas as tarefas são *CPU-intensive*, ou seja, todo o tempo gasto nas suas execuções é gasto com processamento. Outra consideração é a utilização do termo "recurso" para denotar tanto os enlaces de rede quanto os hosts da grade.

O grafo com a topologia formada pelos recursos de uma grade fornece informações momentâneas sobre o estado da grade. A fim de realizar escalonamentos o mais precisos possíveis, esse grafo precisa ser mantido atualizado constantemente, seja através de medições ou de previsões do estado dos hosts e da rede.

## 3.4 Escalonadores existentes

Dentre as várias propostas para escalonar tarefas em sistemas heterogêneos, algumas diretamente relacionadas com as características das grades são encontradas em [32, 45, 8, 25, 22, 39].

A proposta apresentada em [32] é de um algoritmo chamado *Level-Branch Priority* (LBP). O algoritmo escalona as tarefas de um DAG seguindo uma lista de prioridade. Essa lista é construída levando-se em consideração o nível das tarefas no DAG (o maior caminho da tarefa de entrada até a tarefa em questão) e a quantidade de ramificações (arcos de saída) das mesmas. Tarefas nos menores níveis e com mais ramificações são colocadas no início da lista. Um ponto negativo do algoritmo LBP é o fato dele, apesar de ser definido como voltado para grades, não considerar a heterogeneidade nos enlaces da rede, já que na sua formulação, o tempo necessário para transferir os dados de dependência entre duas tarefas é o mesmo independente do enlace utilizado. Os resultados apresentados em [32] não fornecem informações sobre o desempenho do algoritmo com diferentes quantidades de hosts e topologias da rede, diferenças essas que são naturais de ocorrerem em ambientes heterogêneos como as grades.

Em [45], o escalonamento de tarefas não é diretamente relacionado com grades, mas sim com ambientes que possuem enlaces de rede não-dedicados e heterogêneos. O método utilizado para escalonar as tarefas é realizado do ponto de vista das dependências entre as tarefas. Ao invés de escalonar as tarefas nos hosts, os arcos são escalonados nos enlaces. São apresentadas propostas interessantes para diferenciar as características dos enlaces nos grafos que representam a topologia da rede. Resultados que mostram as vantagens de utilizar o escalonamento de arcos são apresentados para uma variedade de topologias de rede. O desempenho do algoritmo em ambientes com recursos heterogêneos como as grades não é analisado, já que nos cenários utilizados, os hosts possuem a mesma taxa de processamento disponível.

Em [8], é apresentada uma heurística para o escalonamento de tarefas em grades. Essa heurística é baseada em sorteios aleatórios de vários escalonamentos possíveis em busca do melhor. A busca pelo melhor escalonamento termina quando um tempo-limite é alcançado. O desempenho da heurística é analisado através da comparação da qualidade dos escalonamentos encontrados por ela com a qualidade dos escalonamentos encontrados por um algoritmo guloso, que escalona uma tarefa por vez no melhor host possível. Com

essa comparação, conclui-se que há perdas consideráveis quando se utiliza escalonadores que tentam encontrar o melhor escalonamento para uma tarefa sem ter noção do DAG (ou *workflow* como é chamado no texto) como um todo. Apesar de reconhecer a heterogeneidade proveniente da rede, os resultados são apresentados para uma única topologia de rede.

Um algoritmo para escalonamento de tarefas independentes que considera requisitos de QoS de aplicações em grades é apresentado em [25]. O algoritmo considera que as aplicações possuem requisitos de QoS que restringem as opções de recursos nos quais as tarefas são escalonadas. A heterogeneidade dos enlaces de rede é levada em consideração pelo algoritmo e, inclusive, nos experimentos realizados, o único requisito de QoS considerado é a largura de banda mínima exigida pelas tarefas.

Em [22], é apresentado um algoritmo para escalonar tarefas de DAGs cujas quantidades de bits a serem transmitidos é desprezível com relação à quantidade de instruções a serem executadas. As tarefas passam muito mais tempo sendo processadas nos hosts do que aguardando a transferência de dados de dependência via rede. O nível das tarefas no DAG afeta a probabilidade das mesmas serem mapeadas para os hosts com maior taxa de processamento disponível. Uma discussão interessante é relacionada com a métrica utilizada para medir a qualidade dos escalonamentos em grades. Segundo [22], a duração do escalonamento em unidades de tempo não seria o ideal já que, como os hosts não são dedicados, as tarefas não passam 100% do tempo sendo executadas. Dessa forma, a métrica ideal seria computar o tempo real de computação que cada tarefa utilizou nos hosts para os quais elas foram escalonadas.

Outra proposta para escalonar tarefas em grades é apresentada em [39]. O escalonamento para as tarefas é encontrado através da implementação de um algoritmo genético. Uma diferença dessa proposta para as considerações feitas nesta dissertação é o fato do grafo das tarefas apresentar ciclos.

# Capítulo 4

# Otimização dinâmica de aplicações em grades

O dinamismo das grades é responsável por exigir técnicas que funcionem em conjunto com o escalonamento das tarefas e que evitem diminuições de desempenho na execução das aplicações. Essas diminuições surgem quando, com o passar do tempo, o escalonamento inicial das tarefas de uma aplicação na grade deixa de ser o melhor devido às mudanças ocorridas no estado dos hosts e da rede que os interliga.

O desempenho das aplicações nas grades pode sofrer diminuição caso ocorra pelo menos uma das seguintes mudanças:

- Degradação no desempenho dos hosts inicialmente alocados: um host selecionado para executar uma dada tarefa pode apresentar diminuição na taxa de processamento disponível caso um usuário local venha a iniciar uma aplicação ou caso outra aplicação na grade utilize esse mesmo host;

- Saída ou falha dos hosts inicialmente alocados: como os hosts da grade não são dedicados à mesma, eles podem sair da grade a qualquer momento, seja por decisão do usuário local ou por apresentar falha;

- Mudança de requisitos da aplicação: em um dado momento da execução, uma aplicação pode apresentar novos requisitos que seriam melhor atendidos caso as tarefas fossem mapeadas em hosts diferentes dos que elas estejam atualmente mapeadas;

- Degradação no desempenho dos enlaces dos hosts inicialmente alocados: um certo caminho na rede, que seria utilizado para transmitir dados de dependência entre duas tarefas da grade, pode apresentar diminuição de desempenho devido à transferências de dados de outras aplicações.

Além de evitar quedas no desempenho, as técnicas devem considerar melhorias no desempenho da aplicação caso um host, ou um caminho na rede, melhor que os atuais torne-se disponível. Os casos em que isso pode acontecer são:

- Entrada de um novo host na grade: um dado host pode ser adicionado na grade por decisão do seu usuário local ou após ter voltado de uma falha;
- Melhora no desempenho de um host não alocado anteriormente: um host que antes estava ocupado com a execução de outra aplicação na grade, ou com a aplicação de um usuário local, pode aumentar a sua taxa de processamento disponível ao finalizar essa execução;
- Melhora no desempenho da rede: a qualidade de um caminho na rede pode melhorar devido à finalização de alguma transferência de dados anterior ou devido à ação de técnicas de engenharia de tráfego para redes.

Dentre as técnicas que podem ser utilizadas em conjunto com o escalonamento de tarefas, destacam-se a replicação das tarefas e a otimização dinâmica das aplicações.

A replicação das tarefas consiste em executar cada uma das tarefas de uma aplicação em mais de um host. Assim que uma das réplicas finaliza sua execução, as demais são interrompidas [12]. Com essa técnica, a probabilidade de uma aplicação apresentar perdas no seu desempenho diminui, já que seria necessário haver quedas no desempenho de vários hosts e vários enlaces de rede ao mesmo tempo para que todas as execuções de uma dada tarefa levassem mais tempo para terminar sua execução, ou transferir seus dados. Um problema dessa técnica é o fato dela ocupar uma quantidade extra de recursos mesmo quando não é necessário. Recursos que poderiam ser alocados para mais de uma aplicação na grade ficam alocados para uma única aplicação. A vantagem da técnica é o fato dela não exigir monitoramento, ou previsão, constante dos hosts e da rede, o que a torna mais fácil de ser implementada.

A otimização dinâmica consiste em utilizar informações atualizadas sobre o estado dos hosts e da rede para propor migrações de tarefas. As informações são conseguidas através de medições e/ou previsões do estado dos hosts e da rede, a fim de detectar mudanças no estado da grade que venham a afetar o desempenho das aplicações em execução. Uma vez detectadas as mudanças, deve ser realizado um reescalonamento das tarefas que ainda não iniciaram sua execução, e deve ser avaliado o ganho alcançado com migrações das tarefas que estejam em execução. A avaliação do ganho alcançado com migrações deve levar em consideração a transferência dos dados que permitam às tarefas migradas continuar as execuções, do ponto onde elas foram interrompidas, nos novos hosts. Ao contrário da replicação, as técnicas de otimização não ocupam recursos de forma desnecessária, porém exigem informações o mais precisas possíveis a respeito do estado da grade, além de

realizarem, constantemente, avaliações sobre possíveis ganhos, o que torna estas técnicas mais complexas de serem implementadas.

A sequência de etapas das técnicas de otimização: monitoramento, reescalonamento e migração, coincide, em termos gerais, com os objetivos da engenharia de tráfego para a Internet [2], que utiliza técnicas para projetar e otimizar uma rede a fim de que ela atenda os requisitos de Qualidade de Serviço das aplicações que utilizam a Internet. De fato, a sequência de etapas para a otimização de aplicações em grades caracteriza-se como uma metodologia de engenharia de tráfego multi-camada, como pode ser notado na Tabela 4.1, onde são listadas as camadas da arquitetura para grades que atendem cada uma das etapas, e as camadas equivalentes na arquitetura Internet.

| Etapa | Camada na arquitetura das grades | Camada na arquitetura Internet |
|---|---|---|
| Monitoramento | Coletivo<br>Recurso | Aplicação |
| Reescalonamento | Coletivo<br>Recurso | Aplicação |
| Migração | Conectividade | Transporte<br>Internet |

Tabela 4.1: Mapeamento das etapas da otimização de aplicações em grades

As seções seguintes deste capítulo apresentam detalhes sobre as etapas da otimização dinâmica e fornecem informações sobre algumas propostas encontradas na literatura. A Seção 4.1 apresenta detalhes sobre o monitoramento e a previsão do estado das grades. O processo utilizado pelo *Network Weather Service* (NWS) é tomado como referência, por ser bastante utilizado na prática. Na Seção 4.2, são apresentadas informações sobre migração de tarefas e detalhes de quando a mesma deve ser realizada para diminuir o tempo de execução das aplicações. Finalizando o capítulo, a Seção 4.3 lista algumas propostas de otimização dinâmica encontradas na literatura, bem como referências relacionadas com etapas específicas.

## 4.1 Monitoramento e previsão do estado

O monitoramento do estado da grade deve utilizar técnicas que meçam a disponibilidade dos hosts bem como técnicas que meçam a disponibilidade da rede. Em se tratando de grades computacionais, a disponibilidade dos hosts corresponde à taxa de processamento disponível, enquanto que a disponibilidade da rede corresponde à banda passante disponível. As medições devem ser feitas em intervalos de tempo regulares, mas de modo

a evitar um alto consumo de processamento dos hosts e um alto consumo da banda disponível nos enlaces. O intervalo de tempo ideal é difícil de ser determinado já que ele depende de características específicas da topologia da grade, das aplicações locais executadas em cada host e das aplicações executadas na grade. Realizar medições separadas pelo menor intervalo de tempo possível seria uma solução para conseguir informações mais precisas, porém, a intrusão gerada com tal solução inviabiliza a sua implementação.

Uma forma de diminuir o impacto das medições é utilizar técnicas que façam a previsão do estado dos hosts e dos enlaces. Com uma quantidade mínima de medições é possível prever a disponibilidade da grade dentro de um certo intervalo de tempo futuro. Com a utilização da previsão, as medições podem ser feitas em intervalos de tempo maiores e, dessa forma, afetar o mínimo possível o estado da grade.

Além de fornecer informações precisas e com o mínimo de intrusão necessária, sistemas de monitoramento e previsão para grades devem permanecer disponíveis durante todo o tempo em que a grade estiver ativa e fornecer dados sobre o estado de todos os recursos.

Todos esses requisitos: não-intrusão, precisão nas previsões, longevidade na execução e ubiquidade, são considerados nos objetivos do NWS, um sistema que produz previsões de desempenho baseadas no histórico de medições realizadas [54] e que é utilizado, ou considerado como referência, por vários sistemas de otimização dinâmica de aplicações em grades. Para os hosts, o NWS mede e prevê a taxa de processamento disponível. Para a rede, o NWS mede e prevê o tempo necessário para estabelecer uma conexão TCP, a latência de rede TCP fim-a-fim e a largura de banda TCP disponível fim-a-fim.

No NWS, a previsão do estado de todas as características de desempenho é realizada por séries temporais. Os dados medidos são agrupados em pares ("instante de tempo","valor medido"). Vários modelos de previsão são aplicados sobre os pares existentes e os valores previstos são comparados com os valores medidos. Aquele modelo que fornecer o menor erro é utilizado para fazer a previsão futura.

O monitoramento da taxa de processamento disponível no NWS é realizada com a correlação de medições feitas por consultas passivas e por consultas ativas. As consultas passivas são realizadas por ferramentas específicas do sistema operacional. Consultas ativas são realizadas através da execução de programas artificiais que são *CPU-intensive*. O tempo de CPU dedicado para esses programas e o tempo real gasto na execução são armazenados, e a razão entre os dois fornece uma informação mais precisa sobre a taxa de processamento disponível no host. O intervalo de tempo em que os programas são executados é ajustado de forma adaptativa: se os últimos valores de taxa de processamento disponível permanecerem relativamente estáveis, o intervalo de tempo é aumentado e, em caso contrário, é diminuído.

O monitoramento das características da rede fim-a-fim é realizado somente por consultas ativas. Para medir o atraso e a banda disponível, uma quantidade fixa de dados

é transferida entre dois hosts da rede. Para medir o tempo necessário para estabelecer uma conexão TCP, uma conexão artificial é estabelecida e finalizada. Uma solução trivial para obter as características de rede fim-a-fim entre todos os hosts, consiste em realizar consultas entre todos os pares de hosts disponíveis na grade. Entretanto, essa solução não é eficiente, uma vez que ela exige $N^2 - N$ consultas, além de incrementar a probabilidade de colisões entre as consultas dos sensores, o que tornaria as medições não-confiáveis. Para evitar a sobrecarga na rede e medições não-confiáveis, sensores configurados nos hosts são organizados de forma hierárquica de modo que as consultas ativas só sejam realizadas em um subconjunto representativo de todos os sensores disponíveis. Essa organização hierárquica consiste em agrupar os sensores dentro de conjuntos chamados cliques. Cada sensor de um dado clique realiza consultas somente com os sensores que estejam no mesmo clique. A hierarquia é organizada através da agregação de diferentes cliques em cada nível e pela promoção de um sensor de cada clique de um nível para participar de um clique do nível seguinte. Uma estratégia para a definição dessa organização hierárquica é criar um clique para cada organização real da grade, e um clique para cada conexão direta entre duas organizações.

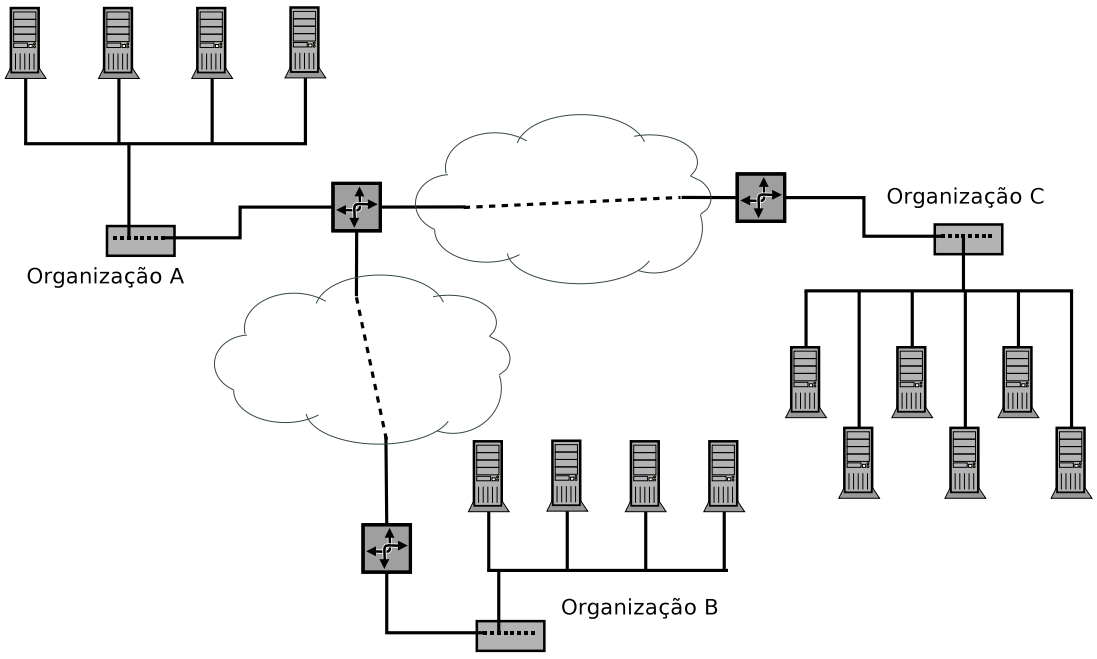



Figura 4.1: Topologia de uma grade formada por três organizações reais

A Figura 4.1 apresenta a topologia de rede de três organizações que fazem parte de uma grade. Na Figura 4.2, é apresentada a organização hierárquica dos sensores para essa grade através de uma representação por grafos. Cada vértice representa um sensor em

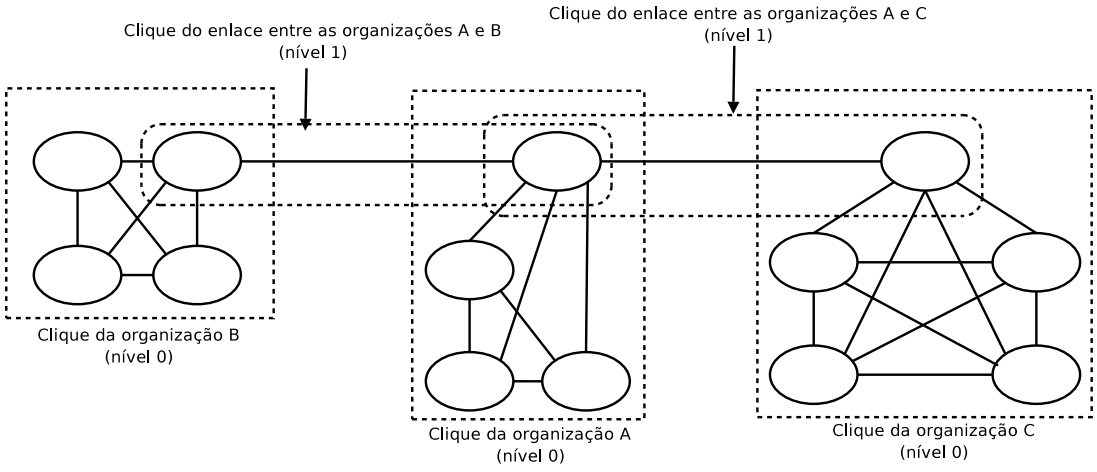



Figura 4.2: Hierarquia de sensores para a grade da Figura 4.1

operação em um host da grade. A presença de uma aresta nessa figura significa que entre os dois sensores são realizadas consultas ativas para medir as características de rede fim-a-fim. Como pode ser notado pela Figura 4.2, em cada organização, todos os hosts realizam consultas entre si e, entre organizações conectadas, (no caso, entre as organizações A e B e entre as organizações A e C) somente dois sensores realizam consultas entre si. A idéia por trás dessa estratégia é o fato de que, dentro de uma organização real, as transferências de dados entre os hosts costumam ser realizadas via uma rede local que não apresenta gargalo para as comunicações. Dessa forma, consultas realizadas entre somente dois hosts de duas organizações reais são suficientes para se ter uma idéia das características de rede fim-a-fim entre todos os hosts das duas organizações.

É importante observar que a utilização de técnicas para prever o estado de hosts em grades não torna desnecessária a utilização das técnicas que verificam a necessidade de migração das tarefas. A primeira justificativa para essa afirmação é o fato de que as previsões podem falhar e, a segunda justificativa, é o fato da previsão não tratar a saída brusca de hosts da grade e nem a entrada de novos hosts, fatores que podem motivar a migração de tarefas.

## 4.2 Reescalonamento e migração

A detecção de modificações no estado da grade, pelos sistemas de monitoramento e previsão, deve ser seguida por um processo que verifique se o escalonamento atual continua sendo o melhor. Uma forma de realizar essa verificação é através de um reescalonamento da aplicação. Todas as tarefas que ainda não foram executadas, ou que estejam em execução, são reescalonadas e o resultado é comparado com o escalonamento anterior.

Caso haja diferenças entre os escalonamentos encontrados, migrações de tarefas devem ser realizadas quando necessário. É importante observar que o reescalonamento deve ser realizado levando em consideração o estado atual dos hosts e da rede, ou seja, os pesos dos vértices e das arestas do grafo que representa a grade devem ser reajustados.

Técnicas para migração de tarefas são propostas desde os primeiros sistemas distribuídos implementados na prática [47]. A migração é realizada para atender diversos objetivos, dentre os quais destacam-se a diminuição no tempo de execução das aplicações distribuídas e o balanceamento de carga nos recursos compartilhados. Em grades, o processo de migração costuma ser chamado de preempção e ganhos alcançados com sua utilização são comprovados com resultados na prática, como pode ser visto na implementação do *middleware* Condor [41].

Para os sistemas de otimização dinâmica, o objetivo a ser alcançado com a migração é a diminuição do tempo de execução das aplicações. Para conseguir esse objetivo, as migrações só devem ser realizadas caso elas diminuam o tempo de execução previsto para a aplicação. O processo de migração de tarefas nem sempre é compensador devido ao tempo gasto com a transferência de dados necessários para que a tarefa continue a sua execução, do ponto onde foi interrompida, no novo host. A Figura 4.3 [47] ilustra o processo de migração com relação à linha do tempo. A seta sob "Fonte" ilustra a execução da tarefa no host inicialmente alocado e a seta sob "Destino" ilustra a continuação da execução da tarefa no novo host alocado. A seta tracejada entre as duas execuções representa a migração dos dados necessários para que a execução da tarefa continue do ponto onde foi interrompida. Dentre os dados necessários, estão o código da tarefa, o estado dos registradores e da memória utilizados e os dados de entrada. Quanto maior o tempo gasto para a transferência dos dados entre dois hosts, menos interessante é a migração.

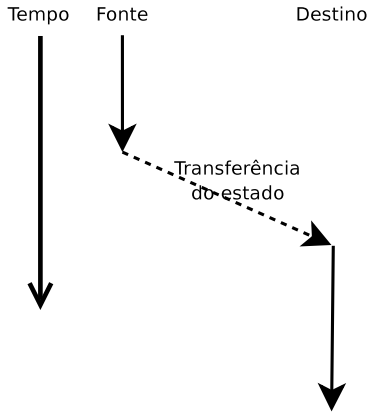



Figura 4.3: Fluxo da execução do processo de migração (Traduzida de [47])

Como explicado em [41], algumas tarefas não podem, na prática, ter a sua execução

interrompida em um determinado host e continuada, do ponto onde houve a interrupção, em outro. Para essas tarefas, a migração deve ser realizada e a execução no host "Destino" deve ser reiniciada sem aproveitar o tempo de execução que foi utilizado no host "Fonte". Há ainda tarefas que conseguem salvar o seu estado somente em determinados pontos de sua execução, denominados *checkpoints*. Caso uma destas tarefas migre em um instante de tempo localizado entre dois *checkpoints*, é possível reiniciar a sua execução no host "Destino" a partir do último *checkpoint* que foi realizado no host "Fonte". Também é importante observar que a migração de uma tarefa só pode ser realizada para hosts que atendam os requisitos da mesma. Tarefas com restrições de sistema operacional, hardware e arquivos de entrada, por exemplo, só podem ser migradas para hosts com características que atendam a essas restrições.

Para tarefas que podem aproveitar o tempo de execução no host "Fonte", a decisão de realizar migração depende da quantidade de dados que devem ser transferidos e da taxa de banda disponível fim-a-fim entre os dois hosts envolvidos. A quantidade de dados a serem transferidos é dependente tanto da tarefa, quanto da porcentagem de execução já concluída. Quanto mais perto a tarefa estiver de finalizar a sua execução, menos compensador torna-se a migração, já que o tempo necessário para transferir os dados pode fazer com que a finalização da tarefa ocorra em um instante de tempo maior do que se a mesma continuasse a sua execução no host anteriormente alocado.

A migração de tarefas em grades também deve levar em consideração a saída brusca de hosts que estejam executando tarefas de alguma aplicação. A saída pode ocorrer por falha do host, por decisão do usuário local, ou por falha em todos os enlaces do host. Nesses casos, pode-se utilizar um servidor que armazene o estado das tarefas em intervalos regulares de tempo e, dessa forma, a tarefa pode reiniciar a sua execução a partir do último estado salvo. Sem a utilização de técnicas semelhantes à implementação deste servidor, não há possibilidade da tarefa reiniciar a sua execução do ponto onde foi interrompida e, portanto, precisa voltar a sua execução do início em um novo host.

## 4.3 Técnicas de otimização existentes

Alguns sistemas para monitoramento e previsão do estado de recursos em grades são apresentados em [49] e [31].

Um sistema de monitoramento e previsão que pretende ser mais escalável e preciso que o NWS é apresentado em [49]. O sistema, nomeado como GHS – *Grid Harvest Service*, tem por objetivo fornecer medições e previsões para grades que executem aplicações que levem horas para finalizar suas execuções. Assim como o NWS, o GHS fornece medição e previsão da taxa de processamento disponível dos hosts e da capacidade de transmissão fim-a-fim entre pares de hosts da grade. A taxa de processamento dos hosts é prevista

a partir da utilização de um modelo analítico que descreve cada host. A latência e a largura de banda disponível da rede são previstas a partir da utilização de redes neurais artificiais. Apesar de focar nas etapas de monitoramento dos hosts, o GHS pode muito bem ser classificado como um sistema para otimização dinâmica de aplicações em grades, já que seu projeto é composto de módulos responsáveis por reescalonar e migrar as tarefas caso hajam variações grandes no estado da grade. Experimentos preliminares comprovam que há ganho em utilizar o GHS na previsão do estado de grades para escalas de tempos maiores que os suportadas pelo NWS.

Uma proposta para monitorar o estado da rede em grades é apresentada em [31]. A proposta sugere a utilização de técnicas passivas de monitoramento quando há tráfego na rede gerado por aplicações da grade, e a utilização de técnicas ativas quando não há aplicação em execução na grade, ou quando o tráfego da rede não consome muita largura de banda. Essa proposta é a base para um sistema de monitoramento chamado Wren, que tem por objetivo fornecer informações de rede que sejam escaláveis de *clusters* até recursos interligados por Wans. Uma contribuição do texto é a apresentação de uma análise dos requisitos de rede para várias classes de aplicações que executam em grades.

Exemplos de sistemas que realizam otimização dinâmica de aplicações em grades são descritos em [1, 52, 26]. Todos eles baseiam-se na execução cíclica das etapas de monitoramento, reescalonamento e migração, com o objetivo de contornar quedas no desempenho da execução de aplicações em grades. Nos três exemplos, mecanismos para as etapas citadas são implementados e incluídos em *middlewares* e *frameworks* que gerenciam a execução de aplicações em grades. Dos três sistemas, somente o descrito em [1] não cita o NWS para a etapa de monitoramento. Entretanto, a proposta apresentada por ele segue os mesmos princípios do NWS.

Em [1], o monitoramento e a decisão para a migração de tarefas são implementados levando em consideração a violação de contratos e a inclusão de novos recursos que melhorem o desempenho da execução das aplicações. Os contratos são firmados entre o usuário e os provedores dos recursos da grade. Eles definem que a aplicação tem que rodar dentro de certos parâmetros e que o provedor deve fornecer serviços que mantenham a aplicação com um desempenho aceitável. Os mecanismos implementados consideram a queda no desempenho da rede como motivo para migrar tarefas, apesar dos resultados apresentados não exibirem ganhos alcançados com a migração para um caso desse. A técnica de escalonamento implementada é baseada no casamento de requisitos em alto nível.

Para a realização da migração, há a possibilidade de que os dados para a tarefa continuar sua execução sejam enviados para um local de armazenamento intermediário, antes de seguirem para o novo host selecionado. Problemas nessa abordagem são o fato de que o local de armazenamento pode vir a tornar-se um gargalo no sistema, caso muitas migrações sejam realizadas ao mesmo tempo, além de gerar uma possível lentidão na mi-

gração, já que a tarefa poderia reiniciar sua execução mais cedo, caso os dados fossem transferidos diretamente para o novo host. A vantagem dessa abordagem é a possibilidade de permitir a migração de tarefas entre hosts que não sejam diretamente ligados entre si.

No esquema apresentado em [26], a migração das tarefas pode ocorrer se houver mudança no estado dos hosts ou mudança nos requisitos das aplicações. A segunda situação ocasiona a migração se o host atual em que uma tarefa estiver mapeada deixar de ser a melhor opção com as mudanças nos requisitos da aplicação. Mudanças no estado da rede são consideradas ao avaliar a necessidade de migração das tarefas, entretanto, a única mudança considerada é a desconexão dos hosts. A queda na banda disponível não é citada como motivo para a migração das tarefas. Para o escalonamento das tarefas é implementado um algoritmo guloso, que mapeia cada tarefa para o host que a faz executar da melhor forma, sem considerar as dependências com outras tarefas. Uma vez tendo realizado esse escalonamento, métodos para detectar perda de desempenho passam a entrar em ação com o objetivo de otimizar o escalonamento inicial. O problema dessa técnica, apesar de fornecer o primeiro escalonamento de forma rápida, é que ela não leva em consideração o custo que isso pode implicar na rede, já que há grandes chances de que muitas migrações sejam realizadas posteriormente.

Em [52], a etapa de migração é a mais focada e ela pode ocorrer nos mesmos casos apresentados em [1]. O monitoramento do progresso e a migração de tarefas são tratados como duas ações a serem realizadas de forma acoplada, a fim de evitar que migrações sejam realizadas desnecessariamente, quando uma tarefa estiver perto de finalizar sua execução. A decisão de migração é realizada comparando o tempo de execução da tarefa no novo host com o tempo que ela levaria para executar no host antigo. Comparando os dois tempos, a migração é realizada caso o ganho seja maior que 30%. Os próprios autores reconhecem que essa não é uma boa abordagem já que casos menores que 30% poderiam ter ganhos reais se a migração fosse realizada. Para o cálculo do tempo de finalização no novo host, além de considerar o tempo que a tarefa levará para executar o restante da computação, é adicionado um *overhead* fixo que foi coletado com experimentos em ambiente reais, o que não significa que sejam válidos para todas as aplicações. A queda no desempenho da rede é considerada como motivo para a migração das tarefas e, inclusive, há resultados que apresentam ganho alcançado com a migração de tarefas por tal motivo. A técnica de escalonamento é baseada na construção de um modelo de execução específico para a aplicação. Com a criação desse modelo é possível ter conhecimento sobre os requisitos em nível de aplicação das tarefas sendo escalonadas, o que torna as decisões de escalonamento mais precisas para uma variedade de aplicações. O problema dessa técnica é a necessidade de recursos dedicados para fazer a geração do modelo e a verificação dos requisitos.

Uma breve discussão sobre migração de tarefas é apresentada em [8]. A mesma heurística utilizada para escalonar tarefas é proposta para a verificação da necessidade de

migração, mas não há resultados que comprovem a funcionalidade dessa técnica.

Uma última referência relacionada com sistemas para otimização dinâmica em grades é apresentada em [39]. É apresentada uma proposta para escalonar tarefas de aplicações modeladas como grafos direcionados que podem conter ciclos. A proposta é baseada na solução de um algoritmo genético. O algoritmo é re-executado durante a execução da aplicação a fim de verificar possíveis ganhos com a migração das tarefas. Um problema desse sistema é o fato dele não considerar as mudanças no estado da rede como motivo para a migração das tarefas.

# Capítulo 5

# Escalonadores de tarefas

Como explicado nos capítulos anteriores, um método de otimização dinâmica para grades, depende de um escalonador de tarefas que encontre o melhor escalonamento possível em um curto intervalo de tempo, e de técnicas que monitorem e proponham migrações para as tarefas, uma vez que a aplicação tenha iniciado a sua execução.

Neste capítulo, são apresentados oito escalonadores de tarefas de DAGs em grades baseados em busca aleatória guiada. Esses escalonadores levam em consideração a heterogeneidade da rede e dos hosts das grades, ao contrário de propostas existentes [32, 8, 45]. Ao contrário da proposta apresentada em [45], os escalonadores não levam em consideração o impacto do roteamento e nem a correlação das transferências de dados das tarefas entre si.

A variedade de escalonadores deve-se ao fato deles diferirem entre si com relação ao tempo de execução para encontrar o escalonamento, e ao *makespan* do escalonamento encontrado. Escalonadores que apresentam os melhores escalonamentos, ou seja, os menores *makespans*, tendem a levar um maior tempo de execução do que os que apresentam os piores escalonamentos. Como os limites temporais para que um escalonamento seja proposto dependem de características específicas de cada aplicação, e de cada conjunto de recursos de uma grade, torna-se interessante ter escalonadores destinados para casos onde a qualidade do escalonamento é o principal objetivo, e para casos onde um menor tempo de execução para devolver o escalonamento é o principal objetivo. Em ambos os casos, o tempo de execução do escalonador deve ser o menor possível com relação ao intervalo de tempo em que ocorram mudanças bruscas no estado da grade.

Este capítulo está organizado da seguinte forma: a Seção 5.1 apresenta considerações gerais sobre todos os escalonadores propostos. Os escalonadores são apresentados da Seção 5.2 até a Seção 5.6. Um resumo sobre as propostas é apresentado na Seção 5.7. Finalizando o capítulo, os experimentos realizados, e os resultados alcançados, são apresentados na Seção 5.8.

## 5.1 Considerações gerais

Os escalonadores apresentados nas próximas seções podem ser classificados de acordo com a abordagem utilizada nas buscas pelo melhor escalonamento. A primeira abordagem é baseada na formulação do problema de escalonamento como problemas de programação linear (PL) inteira e de PL mista. Seis escalonadores dessa abordagem são apresentados. A segunda abordagem é baseada em uma sequência de sorteios dos hosts onde cada tarefa vai executar. Dois escalonadores baseados nessa abordagem são apresentados. Diferentemente da proposta apresentada em [8], não há limite de tempo para que o melhor escalonamento seja encontrado. O limite imposto é na quantidade de sorteios realizados. De um modo geral, os escalonadores da segunda abordagem executam mais rápidos do que os da primeira, embora estes obtenham melhores escalonamentos.

Nas duas abordagens, o objetivo é minimizar o tempo de execução de uma aplicação para grades sob as seguintes restrições:

- Cada tarefa $i$ só pode iniciar a sua execução após todas as tarefas das quais $i$ depende terem terminado as suas execuções e terem transferido seus dados para $i$;
- Cada tarefa só pode estar mapeada para, no máximo, um host em cada instante de tempo;
- Duas tarefas dependentes só podem estar mapeadas em hosts que possuem enlace entre si (cada host tem um enlace virtual com peso zero a ele mesmo);
- Cada host só pode executar no máximo uma tarefa por vez.

Dentre os seis escalonadores baseados em PL, três deles consideram que o escalonamento ocorre em uma linha do tempo contínua ($\in \mathbb{R}_+$). Embora apresentem resultados mais precisos, estes escalonadores podem consumir muito tempo de processamento para obter as soluções, já que o escalonamento de uma tarefa pode ocorrer, teoricamente, em infinitos pontos da linha do tempo. Os outros três escalonadores baseados em PL consideram que o escalonamento ocorre em uma linha do tempo discreta ($\in \mathbb{Z}_+$). Apesar de haver perda de precisão e apresentarem soluções mais distantes da ótima do que as dos três escalonadores que consideram a linha do tempo contínua, eles exigem menos tempo de processamento.

Na abordagem baseada em PL, são utilizadas variáveis binárias como $X_{i,k}$ para definir o mapeamento das tarefas nos hosts. O valor 0 para essas variáveis define que a $i$-ésima tarefa não executa no $k$-ésimo host, enquanto que o valor 1 define que a $i$-ésima tarefa executa no $k$-ésimo host. Apesar das soluções exatas obtidas com as implementações dos PLs serem ótimas, ou próximas da ótima, elas podem exigir um tempo de execução muito

longo (mesmo aquelas que consideram a linha do tempo discreta). Uma estratégia comum para obter resultados em tempos mais curtos é gerar soluções fracionárias através do relaxamento de um PL inteiro, e usar algum método que arredonde as soluções fracionárias para soluções inteiras. No caso específico das formulações propostas, as variáveis $X_{i,k}$ passam a ter valores no intervalo $[0, 1]$. Dois algoritmos baseados em relaxamento são aplicados sobre cada uma das duas formulações originais (a que considera a linha do tempo inteira e a que considera a linha do tempo real), o que totaliza os seis escalonadores baseados em PL. Nesses algoritmos que utilizam o relaxamento, o valor da variável $X_{i,k}$ é tratado como a probabilidade da $i$-ésima tarefa ser executada no $k$-ésimo host.

Para a abordagem baseada em sequência de sorteios, assim como nos algoritmos que utilizam relaxamento, a variável $X_{i,k}$ representa a probabilidade da $i$-ésima tarefa ser mapeada para o $k$-ésimo host. Cada escalonador nesta abordagem é baseado em um algoritmo iterativo que busca $P$ escalonamentos possíveis, um por iteração, e retorna o melhor que tiver sido encontrado. O primeiro passo em cada iteração dos algoritmos consiste em associar uma probabilidade inicial para cada variável $X_{i,k}$. Essa probabilidade é a única diferença entre os algoritmos desta abordagem. No primeiro algoritmo, a probabilidade de mapear cada tarefa é uniformemente distribuída entre os hosts. Assim, caso haja $m$ hosts, a probabilidade de que a $i$-ésima tarefa seja mapeada para cada host é $1/m$. No segundo algoritmo, a probabilidade inicial é mais "consciente", e leva em consideração as características das tarefas e dos hosts de forma similar ao que é realizado em problemas de escalonamento baseados em lista [32, 28]. Para ambos algoritmos, à medida que as tarefas são mapeadas nos hosts, as restrições do DAG e da topologia da rede são respeitadas a fim de obter um escalonamento possível. Como na primeira abordagem, o tempo de execução e a qualidade da solução também tendem a ser diferentes. Em geral, o escalonador baseado no algoritmo que define as probabilidades iniciais "conscientemente" executa mais lento mas produz soluções com melhor qualidade.

A entrada em comum a todos os escalonadores é composta pelo DAG da aplicação a ser escalonada e pelo grafo da rede formada pelos hosts da grade. Considera-se que qualquer DAG de entrada tem apenas uma tarefa de entrada e de saída. No DAG devem estar informadas as seguintes características:

- $n$: quantidade de tarefas ($n \in \mathbb{N}$);
- $I_i$: quantidade de instruções da tarefa $i$ ($I_i \in \mathbb{R}_+$);
- $B_{i,j}$: quantidade de bits transmitidos entre as tarefas $i$ e $j$ ($B_{i,j} \in \mathbb{R}_+$);
- $\mathcal{D}$: conjunto de arcos $\{ij : \ i < j$ e existe um arco de $i$ para $j$ no DAG$\}$;
- $s_0$: instante em que a tarefa de entrada inicia sua execução. Em todos os experimentos realizados, é considerado $s_0 = 0$.

Já no grafo dos recursos, as seguintes características devem ser informadas:

- $m$: quantidade de hosts ($m \in \mathbb{N}$);
- $TI_k$: tempo que o host $k$ leva para executar 1 instrução ($TI_k \in \mathbb{R}_+$);
- $TB_{k,l}$: tempo necessário para enviar 1 bit do host $k$ para o host $l$ ($TB_{k,l} \in \mathbb{R}_+$);
- $\mathcal{N}$: conjunto de enlaces $\{kl$ : host $k$ tem enlace para o host $l\}$. $\forall k, kk \in \mathcal{N}$. Se $kl \in \mathcal{N}$ então $lk \in \mathcal{N}$;
- $\delta(k)$: conjunto de hosts ligados ao host $k$ na rede. $k$ faz parte desse conjunto.

Com relação à nomenclatura utilizada, para todos os escalonadores considera-se que os hosts são identificados por rótulos de 1 a $m$, enquanto que as tarefas são identificadas por rótulos de 1 a $n$. As tarefas são rotuladas seguindo uma ordem topológica do DAG, ou seja, cada arco $(i, j)$ tem $i < j$, onde $i, j \in \{1, \ldots, n\}$. O conjunto de todas as tarefas é denotado por $\mathcal{J} = \{1, \ldots, n\}$ e o conjunto de todos os hosts por $\mathcal{H} = \{1, \ldots, m\}$.

Para uma tarefa $i$ qualquer, o conjunto de tarefas das quais ela depende é chamado de predecessores de $i$, representado por $pred(i)$, e o conjunto de tarefas dependentes dela é chamado de sucessores de $i$, representado por $suc(i)$.

Por último, o termo $T_{\max}$ é utilizado na modelagem de alguns dos escalonadores, bem como na avaliação da qualidade dos escalonamentos encontrados. $T_{\max}$ é o tempo necessário para executar a aplicação caso todas as tarefas sejam mapeadas para o host mais rápido da rede, ou seja, $T_{\max} = \min TI \sum_{i=1}^{n} I_i$, onde $\min TI$ é o menor valor de $TI$.

## 5.2 Programação linear com a linha do tempo contínua

O primeiro escalonador baseado em PL considera a linha do tempo contínua e é um PL misto. Por considerar a linha do tempo contínua, esse é o escalonador com maior probabilidade de encontrar o melhor escalonamento possível, apesar de exigir mais tempo de execução para isso. O escalonamento final é obtido a partir dos valores encontrados para as seguintes variáveis:

- $X_{i,k}$: variável binária que assume o valor 1 se a tarefa $i$ é mapeada para o host $k$ e 0 caso contrário ($X_{i,k} \in \{0, 1\}$);
- $s_i$: variável que define o instante em que a tarefa $i$ inicia sua execução ($s_i \in \mathbb{R}_+$).

A formulação deste escalonador é dada por:

$$\text{Minimize} \quad (I_n \sum_{k=1}^{m} TI_k X_{n,k}) + s_n$$

sujeito a

$$s_i \geq s_0 \qquad \text{para } i \in \mathcal{J}; \tag{C1}$$

$$s_j \geq s_i + \sum_{k \in \mathcal{H}} [(I_i TI_k X_{i,k}) + \sum_{l \in \delta(k)} (B_{i,j} TB_{k,l} VA_{i,k,j,l})] \qquad \text{para } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}; \tag{C2}$$

$$s_j \geq s_i + \sum_{k \in \mathcal{H}} (I_i TI_k VA_{i,k,j,k}) - y(1 - P_{i,j}) \qquad \text{para } i,j \in \mathcal{J}, \quad i \neq j, \quad ij \notin \mathcal{D}, \quad ji \notin \mathcal{D}; \tag{C3}$$

$$s_i \geq s_j + \sum_{k \in \mathcal{H}} (I_j TI_k VA_{j,k,i,k}) - yP_{i,j} \qquad \text{para } i,j \in \mathcal{J}, \quad i \neq j, \quad ij \notin \mathcal{D}, \quad ji \notin \mathcal{D}; \tag{C4}$$

$$\sum_{k \in \mathcal{H}} X_{i,k} = 1 \qquad \text{para } i \in \mathcal{J}; \tag{C5}$$

$$\sum_{k \in \mathcal{H}} \sum_{l \in \delta(k)} VA_{i,k,j,l} = 1 \qquad \text{para } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}; \tag{C6}$$

$$2\, VA_{i,k,j,l} \leq X_{i,k} + X_{j,l} \qquad \text{para } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad k,l \in \mathcal{H}, \quad kl \in \mathcal{N}; \tag{C7}$$

$$VA_{i,k,j,l} - X_{i,k} - X_{j,l} \geq -1 \qquad \text{para } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad k,l \in \mathcal{H}, \quad kl \in \mathcal{N}; \tag{C8}$$

$$2\, VA_{i,k,j,k} \leq X_{i,k} + X_{j,k} \qquad \text{para } i,j \in \mathcal{J}, \quad i \neq j, \quad ij \notin \mathcal{D}, \quad ji \notin \mathcal{D}, \quad k \in \mathcal{H}; \tag{C9}$$

$$VA_{i,k,j,k} - X_{i,k} - X_{j,k} \geq -1 \qquad \text{para } i,j \in \mathcal{J}, \quad i \neq j, \quad ij \notin \mathcal{D}, \quad ji \notin \mathcal{D}, \quad k \in \mathcal{H}; \tag{C10}$$

$$VA_{i,k,j,l}, X_{i,k}, P_{i,j} \in \{0,1\} \qquad \text{para } i,j \in \mathcal{J}, \quad k,l \in \mathcal{H}. \tag{C11}$$

Nesta formulação, a função objetivo consiste em minimizar o tempo de execução da aplicação, que equivale ao instante de tempo em que a tarefa de saída do DAG, a tarefa $n$, finaliza sua execução. Esse tempo de execução é calculado pela soma do instante de tempo em que a tarefa $n$ inicia sua execução ($s_n$) com o tempo necessário para executar as instruções da tarefa $n$ ($I_n \sum_{k=1}^{m} TI_k X_{n,k}$)).

As restrições (C1) determinam que todas as tarefas começam suas execuções após o instante de tempo em que a tarefa de entrada iniciou a sua execução ($s_0$). As restrições (C2) determinam que a tarefa $j$ deve iniciar sua execução após os seus predecessores finalizarem suas execuções e as transferências de dados para $j$. As restrições (C3) e (C4) determinam que se duas tarefas sem dependências entre si são escalonadas para o mesmo host, uma delas será executada primeiro. A variável binária $P_{i,j}$ é 1 se a tarefa $i$ é executada primeiro (nesse caso, a restrição (C4) sempre é satisfeita) e 0 se $j$ é executada primeiro (restrição (C3) é sempre satisfeita). A constante $y$ é um número positivo suficientemente grande (e.g., $T_{max}$). A restrição (C5) determina que a tarefa $i$ deve ser escalonada para algum host $k$ da grade. A restrição (C6) determina que há apenas uma tupla $(i, k, j, l)$ tal que as tarefas $i$ e $j$ são escalonados em hosts $k$ e $l$, respectivamente. As restrições (C7), (C8), (C9) e (C10) determinam que $VA_{i,k,j,l}$ é 1 se e somente se $X_{i,k} + X_{j,l}$ é 2. Estas restrições são usadas para garantir que duas tarefas com dependência entre si sejam mapeadas em hosts com enlace entre si.

Esta formulação usa no máximo $(\frac{m^2}{2} + \frac{3m}{2} + 3)n^2 - (\frac{m^2}{2} + \frac{3m}{2} + 1)n$ restrições (já que há no máximo $(m^2 - m)/2$ enlaces na rede e $(n^2 - n)/2$ arcos no DAG) e o número total de variáveis é $(m^2 + 1)n^2 + (m + 1)n$.

No restante da dissertação, o escalonador baseado na solução exata deste problema é denotado por PLMTC.

## 5.3 Programação linear com a linha do tempo discreta

O segundo escalonador baseado em PL é modelado como um PL inteiro e considera a linha do tempo discreta, o que significa que uma tarefa só pode iniciar e finalizar a sua execução em pontos discretos da linha do tempo. Com pontos discretos para a execução das tarefas, o escalonador tende a devolver o escalonamento em um intervalo de tempo menor do que o escalonador apresentado na seção anterior, que considera a linha do tempo contínua.

A formulação deste escalonador utiliza o valor $T_{max}$ como o maior instante de tempo possível no qual uma tarefa pode finalizar a sua execução. Denota-se $\mathcal{T} = \{1, \ldots, T_{max}\}$.

O escalonamento é obtido a partir dos resultados encontrados para as seguintes variáveis

apresentadas na formulação:

- $x_{i,t,k}$: variável binária que assume valor 1 se a tarefa $i$ termina sua execução no instante $t$ e no host $k$, caso contrário, a variável assume valor 0 ($x_{i,t,k} \in \{0,1\}$);
- $f_i$: variável que define o instante em que a tarefa $i$ finaliza sua execução ($f_i \in \mathbb{N}^*$).

A descrição desta formulação é a seguinte:

Minimize $f_n$

sujeito a

$$\sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{H}} x_{j,t,k} = 1 \qquad \text{para } j \in \mathcal{J}; \qquad \text{(D1)}$$

$$f_j = \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{H}} t\, x_{j,t,k} \qquad \text{para } j \in \mathcal{J}; \qquad \text{(D2)}$$

$$x_{j,t,k} = 0 \qquad \text{para } j \in \mathcal{J}, \quad k \in \mathcal{H}, \quad t \in \{1,\ldots,\lceil I_j\, TI_k\rceil\}; \qquad \text{(D3)}$$

$$\sum_{k\in\delta(l)} \sum_{s=1}^{\lceil t - I_j\, TI_l - B_{i,j}\, TB_{k,l}\rceil} x_{i,s,k} \geq \sum_{s=1}^{t} x_{j,s,l} \qquad \text{para } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad \text{para } l \in \mathcal{H}, \quad t \in \mathcal{T}; \qquad \text{(D4)}$$

$$\sum_{j\in\mathcal{J}} \sum_{s=t}^{\lceil t + I_j\, TI_k - 1\rceil} x_{j,s,k} \leq 1 \qquad \text{para } k \in \mathcal{H}, \quad t \in \mathcal{T}, \quad t \leq \lceil T_{max} - I_j\, TI_k\rceil; \qquad \text{(D5)}$$

$$\sum_{l\in\mathcal{H}} VA_{j,l} = 1 \qquad \text{para } j \in \{2,\ldots,n\}; \qquad \text{(D6)}$$

$$(|\{i : ij \in \mathcal{D}\}| + 1)\, VA_{j,l} \leq \sum_{t\in\mathcal{T}} x_{j,t,l} + \sum_{i:ij\in\mathcal{D}} \sum_{k\in\delta(l)} \sum_{t\in\mathcal{T}} x_{i,t,k} \qquad \text{para } j \in \{2,\ldots,n\}, \quad l \in \mathcal{H}; \qquad \text{(D7)}$$

$$|\{i : ij \in \mathcal{D}\}| + VA_{j,l} \geq \sum_{t\in\mathcal{T}} x_{j,t,l} + \sum_{i:ij\in\mathcal{D}} \sum_{k\in\delta(l)} \sum_{t\in\mathcal{T}} x_{i,t,k} \qquad \text{para } j \in \{2,\ldots,n\}, \quad l \in \mathcal{H}; \qquad \text{(D8)}$$

$$VA_{j,l},\ x_{j,t,l} \in \{0,1\} \quad \text{para } j \in \mathcal{J}, \quad l \in \mathcal{H}, \quad t \in \mathcal{T}. \quad \text{(D9)}$$

A função objetivo desta formulação, da mesma forma que na formulação que considera a linha do tempo contínua, consiste em minimizar o instante de tempo em que tarefa $n$ finaliza sua execução. Para esta formulação, esse instante de tempo equivale ao valor da variável $f_n$.

As restrições (D1) garantem que uma tarefa só pode executar em um único host e finalizar a sua execução em um único instante de tempo. As restrições (D2) relacionam as variáveis $f_j$ e $x_{j,t,k}$ entre si, para garantir que o tempo de finalização definido por $f_j$ seja igual ao instante $t$ em que $x_{j,t,k}$ vale 1, já que as tarefas só finalizam as suas execuções em um único instante de tempo. As restrições (D3) garantem que o tempo de finalização de uma tarefa é maior ou igual ao tempo gasto para a mesma executar todas as suas instruções. As restrições (D4) definem que a $j$-ésima tarefa só pode iniciar a sua execução após todos os seus predecessores terem finalizado as suas execuções e as transferências dos dados para $j$. As restrições (D5) garantem que cada host só pode executar no máximo uma tarefa em qualquer instante de tempo e as restrições (D6) a (D8) garantem que tarefas dependentes são escalonadas para hosts que possuem um enlace entre si.

Para esta formulação, a quantidade de restrições é no máximo $(n^2 + n + 2)\frac{m}{2}T_{max} + (2n-2)m + 3n - 1$ e a quantidade de variáveis é $(mT_{max} + m + 1)n$.

No restante do artigo, o escalonador baseado na solução exata deste problema é denotado por PLITD.

Uma observação importante é o fato dos resultados computacionais obtidos com esta formulação serem dependentes da unidade de tempo considerada na discretização da linha do tempo. Se a unidade de tempo é muito grande, os resultados podem ser obtidos mais rápidos, embora os erros de aproximação tornem-se maiores, o que diminui a qualidade dos escalonamentos encontrados.

## 5.4 Programação linear com o relaxamento das variáveis inteiras

Os próximos quatro escalonadores baseados em PL são aqueles que aplicam o relaxamento das variáveis inteiras que definem o mapeamento das tarefas. Os escalonadores são baseados na execução dos PLs com relaxamento juntamente com um algoritmo que arredonda os valores das variáveis que definem o mapeamento. Dois algoritmos para realizar esse arredondamento são propostos e cada um é aplicado no relaxamento dos dois escalonadores apresentados nas seções anteriores, definindo assim, mais quatro escalonadores baseados

em PL.

O relaxamento do PLMTC consiste em substituir o conjunto $\{0, 1\}$ nas restrições C11 da formulação, pelo intervalo pelo intervalo $[0, 1]$. Para o PLITD, o relaxamento consiste em substituir o conjunto $\{0, 1\}$ nas restrições D9 pelo intervalo $[0, 1]$. Além disso, para o PLITD também é necessário definir as variáveis $X_{i,k}$ como o somatório $\sum_{t=1}^{T_{max}} x_{i,t,k}$.

O primeiro algoritmo, denominado arredondamento aleatório, é apresentado em Algoritmo 1. Ele resolve o PL uma única vez e os valores das variáveis são usadas como probabilidades para $P$ sorteios. Cada sorteio define um escalonamento e aquele que alcança o menor tempo de execução é devolvido como solução.

**Algoritmo 1** Arredondamento aleatório

**Entrada:** *PL*: relaxamento do programa linear que escalona $\mathcal{J}$ em $\mathcal{H}$.
$P$: quantidade de sorteios.
**Saída:** Escalonamento de $\mathcal{J}$ em $\mathcal{H}$.

1: Defina o melhor escalonamento como aquele que executa todas as tarefas de $\mathcal{J}$ seqüencialmente no melhor host de $\mathcal{H}$.
2: Defina $X$ como a solução fracionária ótima de $PL$, sendo que $X = (X_{i,k})$.
3: **para** $P$ vezes **faça**
4:  **para** cada tarefa $i$ de $\mathcal{J}$, seguindo uma ordem topológica, **faça**
5:  Defina a probabilidade de mapear a tarefa $i$ para o host $k$ como sendo $X_{i,k}$.
6:  Selecione o host para executar $i$ a partir das probabilidades de mapeamento anteriores.
7:  **fim para**
8:  Obtenha o tempo de início de execução para cada tarefa $i$ considerando o tempo de finalização de suas dependências.
9:  Salve o escalonamento se ele fornece um *makespan* menor que os anteriores.
10: **fim para**
11: Retorne o escalonamento encontrado.

O escalonador que utiliza o Algoritmo 1 sobre o relaxamento do PLMTC é denotado por TC-R1, enquanto que o escalonador que o utiliza sobre o PLITD é denotado por TD-R1.

O segundo algoritmo, denominado arredondamento aleatório iterativo, é apresentado em Algoritmo 2. Nesse algoritmo, para cada iteração, uma tarefa é mapeada para um host através das probabilidades e o PL é re-executado com a inclusão desse mapeamento. O algoritmo termina quando não há mais tarefas a serem mapeadas.

O escalonador que utiliza o Algoritmo 2 sobre o relaxamento do PLMTC é denotado por TC-R2, enquanto que o escalonador que o utiliza sobre o PLITD é denotado por TD-R2.

**Algoritmo 2** Arredondamento aleatório iterativo

**Entrada:** *PL*: relaxamento do programa linear que escalona $\mathcal{J}$ em $\mathcal{H}$.
**Saída:** Escalonamento de $\mathcal{J}$ em $\mathcal{H}$.

1: Defina o melhor escalonamento como aquele que executa todas as tarefas de $\mathcal{J}$ seqüencialmente no melhor host de $\mathcal{H}$.
2: Defina $X$ como a solução fracionária ótima de *PL*, sendo que $X = (X_{i,k})$.
3: **para** cada tarefa $i$ de $\mathcal{J}$, seguindo uma ordem topológica, **faça**
4: Defina a probabilidade de mapear a tarefa $i$ para o host $k$ como sendo $X_{i,k}$.
5: Selecione o host $l$ para executar $i$ a partir das probabilidades de mapeamento anteriores.
6: Adicione a *PL*, a restrição de que a tarefa $i$ deve ser mapeada no host $l$.
7: Defina $X$ como a solução fracionária ótima do novo *PL*, sendo que $X = (X_{i,k})$
8: **fim para**
9: **se** escalonamento encontrado tem *makespan* menor que o escalonamento inicial **então**
10: Retorne o escalonamento encontrado.
11: **senão**
12: Retorne o escalonamento inicial.
13: **fim se**

## 5.5 Sorteio com probabilidades uniformes

O primeiro escalonador baseado em uma sequência de sorteios é apresentado nesta seção. Ele é baseado em um algoritmo iterativo que sorteia os hosts onde cada tarefa será executada. As probabilidades de uma tarefa ser mapeada em um host são iguais para todos os hosts, exceto para aqueles cujas restrições de existência de enlaces não permitam a execução da tarefa. Em cada iteração (são realizadas $P$ iterações no total), o algoritmo sorteia um host para cada tarefa seguindo a ordem topológica do DAG, as restrições de dependência do DAG e as restrições de conectividade da rede. O algoritmo proposto é descrito pelo Algoritmo 3.

O escalonador baseado na implementação deste algoritmo é denotado por SU.

## 5.6 Sorteio com probabilidades "conscientes"

O segundo escalonador baseado em uma sequência de sorteios difere do apresentado na seção anterior pela definição das probabilidades iniciais. Ao invés de probabilidades uniformes, as probabilidades iniciais são atribuídas levando em consideração características dos hosts da grade como listado a seguir:

- A probabilidade da tarefa ser mapeada para um host é diretamente proporcional à taxa de processamento disponível no host;

**Algoritmo 3** Sorteio com probabilidades uniformes

**Entrada:** $\mathcal{J}$: DAG com as tarefas.
$\mathcal{H}$: grafo com os hosts.
$P$: quantidade de sorteios.
**Saída:** Escalonamento de $\mathcal{J}$ em $\mathcal{H}$.

1: Defina o melhor escalonamento como aquele que executa todas as tarefas de $\mathcal{J}$ seqüencialmente no melhor host de $\mathcal{H}$.
2: **para** $P$ vezes **faça**
3: Atribua a probabilidade para mapear cada tarefa para cada host como $1/m$.
4: **para** cada tarefa $i$ de $\mathcal{J}$, seguindo uma ordem topológica, **faça**
5: **se** todas as probabilidades da tarefa $i$ forem zero **então**
6: Vá para o passo 12
7: **fim se**
8: Sorteie um host $k$ para executar a tarefa $i$, com as probabilidades atuais.
9: Normalize as probabilidades das tarefas $\in suc(i)$ levando em consideração as restrições de rede (se uma tarefa $j$ é dependente da tarefa $i$, todas as probabilidades para mapear $j$ para hosts que não possuam enlace vindo do host $k$ são definidas como zero).
10: Calcule o tempo de início de execução da tarefa $i$ considerando o tempo de finalização de execução e de transferência de dados das tarefas $\in pred(i)$.
11: **fim para**
12: Salve o escalonamento se ele tem um *makespan* menor que os anteriores.
13: **fim para**
14: Retorne o escalonamento encontrado.

- A probabilidade da tarefa ser mapeada para um host é diretamente proporcional à quantidade de enlaces do host;
- A probabilidade da tarefa ser mapeada para um host é diretamente proporcional à capacidade de transmissão dos enlaces do host.

Esses itens são aplicados de acordo com a expressão 5.1, que calcula as probabilidades iniciais $X_{i,k}$ para cada par $(i, k)$, onde $i$ representa uma tarefa do DAG e $k$ representa um host da grade.

$$X_{i,k} = \left(\frac{\frac{1}{TI_k}}{\sum_{j=0}^{m} \frac{1}{TI_j}} \times \frac{1}{3}\right) + \left(\frac{|\delta(k)| - 1}{\sum_{j=1}^{m} |\delta(j)| - m} \times \frac{1}{3}\right) + \left(\frac{\sum_{l \in \delta(k) - \{k\}} \frac{1}{TB_{k,l}}}{\sum_{j=1}^{m} \sum_{l \in \delta(j) - \{j\}} \frac{1}{TB_{j,l}}} \times \frac{1}{3}\right) \quad (5.1)$$

O primeiro termo da soma descrita na expressão 5.1 realiza o cálculo referente à taxa de processamento disponível em cada host. O segundo termo realiza o cálculo referente

à quantidade de enlaces de cada host e, o terceiro termo, realiza o cálculo referente à capacidade de transmissão dos enlaces de cada host. Cada um dos três itens considerados possui o mesmo peso para o cálculo da probabilidade inicial $X_{i,k}$.

No entanto, essas considerações não são suficientes para encontrar um bom escalonamento pois, somente com elas, os hosts com maior taxa de processamento disponível, mais enlaces e maior capacidade de transmissão tornam-se ocupados com a maioria das tarefas, enquanto os demais hosts tornam-se ociosos, não explorando portanto, os benefícios do paralelismo. Para evitar esse tipo de problema, o algoritmo também usa características das tarefas, como é feito em abordagens de escalonamento baseado em lista [32, 28]:

- Tarefas nos menores níveis do DAG têm probabilidades mais altas de serem mapeadas nos hosts com maior taxa de processamento disponível e com enlaces que possuam as melhores capacidades;
- Tarefas com maior quantidade de arcos têm probabilidades mais altas de serem mapeadas em hosts com maior quantidade de enlaces;
- Tarefas com maior quantidade de dados a serem transferidos (considerando os seus arcos de saída) têm probabilidades mais altas de serem mapeadas em hosts que estão conectados a outros hosts por enlaces com maior capacidade de transmissão;
- Tarefas com maior quantidade de instruções têm probabilidades mais altas de serem mapeadas em hosts com maior taxa de processamento disponível.

O nível considerado pelo primeiro item é o maior caminho da tarefa de entrada do DAG até a tarefa em questão, considerando que cada arco tem peso igual à unidade.

Esses itens são utilizados para modificar os valores das probabilidades $X_{i,k}$ calculados pela expressão 5.1. A modificação das probabilidades é realizada pelo Algoritmo 4 e pelo Algoritmo 5, que devem ser executados em sequência sobre as probabilidades encontradas pela expressão 5.1.

O Algoritmo 4 modifica as probabilidades iniciais levando em consideração o nível e a quantidade de arcos de saída de cada tarefa (a variável $h$, utilizada no algoritmo, representa o maior nível do DAG e a função `abs` devolve o valor absoluto do valor passado como parâmetro). O Algoritmo 5 modifica as probabilidades devolvidas pelo Algoritmo 4 levando em consideração a quantidade de bytes transferidos e a quantidade de instruções de cada tarefa.

O escalonador baseado na utilização das probabilidades conscientes é descrito pelo Algoritmo 6 e é denotado por SC. Nos passos 3, 4 e 5 deste algoritmo, são aplicados, respectivamente, a expressão 5.1, o Algoritmo 4 e o Algoritmo 5.

**Algoritmo 4** Modificação das probabilidades iniciais - Primeira parte

**Entrada:** $\mathcal{J}$: DAG com as tarefas; $\mathcal{H}$: grafo com os hosts; matriz $X$, $n \times m$ com as probabilidades iniciais das $n$ tarefas executarem em cada um dos $m$ hosts.
**Saída:** $X$: matriz $n \times m$ com probabilidades modificadas pelos dois primeiros itens das probabilidades "conscientes".

**para** cada nível $n$ do DAG com exceção do último **faça**
  **para** cada tarefa $i$ do nível $n$ **faça**
    Ordene, de forma crescente, as probabilidades $X$ da tarefa $i$ e armazene a ordem dos hosts $\mathcal{H}$ na lista $L$.
    **para** $k$ de 1 a $\frac{m}{2}$ **faça**
      $fator = X_{i,L[m-k+1]}(\frac{h-n}{h+1})$
      $X_{i,L[m-k+1]} = X_{i,L[m-k+1]} + fator$
      $X_{i,L[k]} = X_{i,L[k]} - fator$
    **fim para**
    Normalize as probabilidades da tarefa $i$.
  **fim para**
**fim para**
**para** cada tarefa $i$ do DAG com exceção da tarefa de saída **faça**
  $qtde = 0$
  **para** cada host $k$ da grade $|X_{i,k} > 0$ **faça**
    $qtde = qtde + 1$
    $dif[k] = |suc(i)| - |\delta(k)| - 1$
  **fim para**
  Ordene, de forma crescente, os valores absolutos do vetor $dif$ e armazene a ordem dos hosts $\mathcal{H}$ na lista $L$.
  **para** cada host $k$ da grade $|X_{i,k} > 0$ **faça**
    **se** $|suc(i)| - abs(dif[k]) \leq 0$ **então**
      $dif[k] = 1$
    **senão**
      $dif[k] = |suc(i)| - abs(dif[k])$
    **fim se**
  **fim para**
  $soma = \sum_{k=1}^{qtde} dif[k]$
  **para** $k$ de 1 a $\frac{qtde}{2}$ **faça**
    $fator = X_{i,L[k]}(\frac{dif[k]}{soma})$
    $X_{i,L[k]} = X_{i,L[k]} + fator$
    $X_{i,L[qtde-k+1]} = X_{i,L[qtde-k+1]} - fator$
  **fim para**
  Normalize as probabilidades da tarefa $i$.
**fim para**
Retorne a matriz $X$.

## 5.7 Resumo dos escalonadores

A Figura 5.1 apresenta os escalonadores organizados pelas suas características em comum: baseados em PL ou baseados em sorteio e, dentre os baseados em PL, sem utilizar relaxamento das variáveis inteiras ou com o relaxamento das variáveis inteiras.

No próximo capítulo são apresentadas as etapas da engenharia de tráfego para grades. Os escalonadores aqui apresentados são utilizados na etapa responsável pelo escalonamento de tarefas, bem como nas etapas responsáveis pelo reescalonamento e pela migração das tarefas. Para estas duas etapas, os escalonadores devem ser re-executados sobre um DAG modificado de modo que seja considerado o impacto devido à migração dos dados que permitam às tarefas continuarem suas execuções em outros hosts.

**Algoritmo 5** Modificação das probabilidades iniciais - Segunda parte

**Entrada:** $\mathcal{J}$: DAG com as tarefas; $\mathcal{H}$: grafo com os hosts; matriz $X$, $n \times m$ com as probabilidades devolvidas pelo Algoritmo 4.
**Saída:** $X$: matriz $n \times m$ com probabilidades "conscientes" das $n$ tarefas executarem em cada um dos $m$ hosts.

**para** cada nível $n$ do DAG com exceção do último **faça**
  $soma = \sum_{i \in \text{nível} n} \sum_{j \in suc(i)} B_{i,j}$
  **para** cada tarefa $i$ no nível $n$ do DAG **faça**
    $somaI = \sum_{j \in suc(i)} B_{i,j}$
    $qtde$ = quantidade de probabilidades $X_{i,k} \neq 0$
    Ordene, de forma crescente, as probabilidades $X$ da tarefa $i$, diferentes de zero, e armazene a ordem dos hosts $\mathcal{H}$ na lista $L$.
    **para** $k$ de 1 a $\frac{qtde}{2}$ **faça**
      $fator = X_{i,L[qtde-k+1]}(\frac{somaI}{soma})$
      $X_{i,L[qtde-k+1]} = X_{i,L[qtde-k+1]} + fator$
      $X_{i,L[k]} = X_{i,L[k]} - fator$
    **fim para**
    Normalize as probabilidades da tarefa $i$.
  **fim para**
**fim para**
**para** cada nível $n$ do DAG **faça**
  $soma = \sum_{i \in \text{nível} n} I_i$
  **para** cada tarefa $i$ no nível $n$ do DAG **faça**
    $qtde$ = quantidade de probabilidades $X_{i,k} \neq 0$
    Ordene, de forma crescente, as probabilidades $X$ da tarefa $i$, diferentes de zero, e armazene a ordem dos hosts $\mathcal{H}$ na lista $L$.
    **para** $k$ de 1 a $\frac{qtde}{2}$ **faça**
      $fator = X_{i,L[qtde-k+1]}(\frac{I_i}{soma})$
      $X_{i,L[qtde-k+1]} = X_{i,L[qtde-k+1]} + fator$
      $X_{i,L[k]} = X_{i,L[k]} - fator$
    **fim para**
    Normalize as probabilidades da tarefa $i$.
  **fim para**
**fim para**
Retorne a matriz $X$.

## 5.8 Experimentos realizados

Nesta seção, são apresentados experimentos realizados para avaliar a eficácia dos vários escalonadores propostos nas seções anteriores. Os experimentos têm o objetivo de avaliar os escalonadores com relação ao tempo de execução e à qualidade do escalonamento encontrado. Dois tipos de DAGs e uma variedade de configurações para a topologia da grade foram utilizados. No restante da dissertação, estes experimentos são referenciados como "primeiro grupo de experimentos".

Todos os algoritmos propostos foram implementados em `C`, sendo que, na implementação dos escalonadores baseados em PL, foi utilizada a biblioteca de otimização `Xpress-MP` [13] versão 2005A. Todos os experimentos foram executados em um computador Pentium 4, com freqüência de 3,2GHz, 2GB de memória RAM e sistema operacional GNU/Linux Gentoo 1.6.14.

Os modelos de aplicações e de topologias de grades simulados são baseados em casos reais, a fim de minimizar a artificialidade dos experimentos. As aplicações são compostas

**Algoritmo 6** Sorteio com probabilidades "conscientes"

**Entrada:** $\mathcal{J}$: DAG com as tarefas.
$\mathcal{H}$: grafo com os hosts.
$P$: quantidade de sorteios.
**Saída:** Escalonamento de $\mathcal{J}$ em $\mathcal{H}$.

1: Defina o melhor escalonamento como aquele que executa todas as tarefas de $\mathcal{J}$ seqüencialmente no melhor host de $\mathcal{H}$.
2: **para** $P$ vezes **faça**
3: Calcule as probabilidades iniciais $X_{i,k}$ pela expressão 5.1.
4: Execute o Algoritmo 4.
5: Execute o Algoritmo 5.
6: **para** cada tarefa $i$ de $\mathcal{J}$, seguindo uma ordem topológica, **faça**
7: **se** todas as probabilidades da tarefa $i$ forem zero **então**
8: Vá para o passo 14
9: **fim se**
10: Sorteie um host $k$ para executar a tarefa $i$, com as probabilidades atuais.
11: Normalize as probabilidades das tarefas $\in suc(i)$ levando em consideração as restrições de rede (se uma tarefa $j$ é dependente da tarefa $i$, todas as probabilidades para mapear $j$ para hosts que não possuam enlace vindo do host $k$ são definidas como zero).
12: Calcule o tempo de início de execução da tarefa $i$ considerando o tempo de finalização de execução e de transferência de dados das tarefas $\in pred(i)$.
13: **fim para**
14: Salve o escalonamento se ele tem um *makespan* menor que os anteriores.
15: **fim para**
16: Retorne o escalonamento encontrado.

por tarefas dependentes, com os pesos das tarefas expressos em $10^6$ milhões de instruções e com os pesos das dependências de dados expressos em gigabytes. Os hosts das grades têm a taxa de processamento disponível expressa em milhões de instruções por segundo (MIPS) e a capacidade de transmissão dos enlaces expressa em gigabits por segundo (Gbps). Na representação da topologia da rede como grafo, os pesos dos hosts são expressos em $\text{MIPS}^{-1}$ e os pesos dos enlaces são expressos em $\text{Gbps}^{-1}$, para torná-los consistentes com a entrada esperada pelos escalonadores.

Dois tipos de aplicações foram simuladas. O primeiro tipo é baseado na aplicação `Griz` [40], uma aplicação de renderização remota apresentada no evento iGrid2002. O DAG desse tipo que foi utilizado na maioria dos experimentos pode ser visto na Figura 5.2. Ele possui 6 tarefas com pesos variando no intervalo $[45, 55]$ e com o peso das dependências variando no intervalo $[0,5 - 0,625]$. O segundo tipo é baseado na aplicação `Montage` [35], uma aplicação de astronomia para processamento de imagens que foi desenvolvida em um

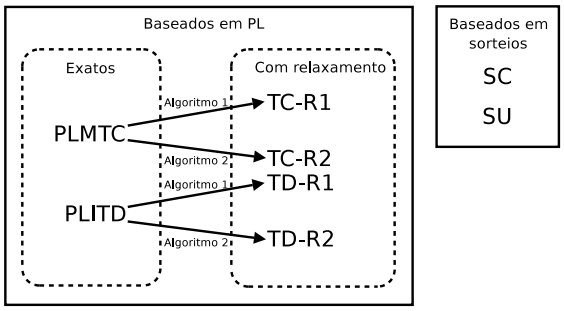



Figura 5.1: Resumo dos escalonadores propostos

projeto de pesquisa executado pela NASA e pelo *California Institute of Technology*. O DAG utilizado nos experimentos pode ser visto na Figura 5.3. Ele possui 26 tarefas com pesos variando nos mesmos intervalos do DAG do primeiro tipo.

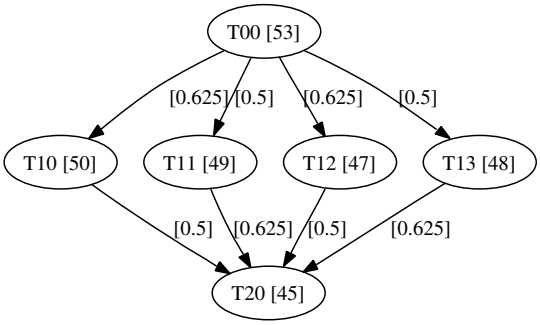



Figura 5.2: DAG do primeiro tipo (renderização remota)

Os algoritmos TC-R1, TD-R1, SU e SC foram executados com $P = 10000$. Para os algoritmos baseados na modelagem que considera a linha do tempo discreta (PLITD, TD-R1 e TD-R2), a unidade de tempo considerada foi 8 minutos para o DAG do primeiro tipo e 16 minutos para o DAG do segundo tipo.

Diferentemente de alguns experimentos realizados na literatura para avaliar escalonadores de tarefas [32, 45, 8], as propostas dos escalonadores aqui apresentados foram avaliados com diversas topologias de redes, formadas por hosts e enlaces heterogêneos. As topologias das grades simuladas foram geradas pelo método Doar-Leslie [14]. O método Doar-Leslie é um método para geração de grafos aleatórios que modelam redes com características próximas de redes reais (vértices geralmente representam roteadores e arestas representam enlaces). Doar e Leslie tomaram como base o método de geração de grafos proposto por Waxman [53] e o modificaram através da inclusão de um fator de adaptação. Esse fator corrige o problema, presente no método original de Waxman, de haver aumento no grau dos vértices com o aumento da quantidade de vértices do grafo gerado. A entrada

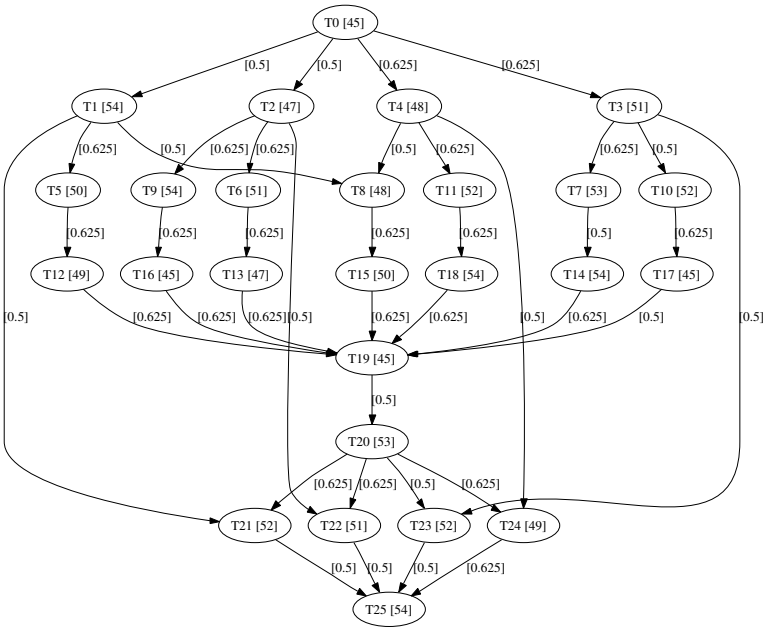



Figura 5.3: DAG do segundo tipo (processamento de imagens)

para o método é composta pelos seguintes valores:

- $N$: quantidade de nós da rede ($N \in \mathbb{N}^*$);
- $\alpha$: relação entre a quantidade de enlaces de longa distância e a quantidade de enlaces de curta distância ($\alpha \in \mathbb{R}_+^*, \alpha \leq 1$);
- $\beta$: fator que define a quantidade de enlaces de cada nó ($\beta \in \mathbb{R}_+^*, \beta \leq 1$);
- $e$: grau médio dos nós da rede ($e \in \mathbb{N}^*, e \leq N$);
- $k$: constante para garantir que o grau do grafo gerado seja próximo do valor de $e$ ($k \in \mathbb{N}^*$).

O funcionamento do método Doar-Leslie é descrito no Algoritmo 7. Nos experimentos realizados, os vértices dos grafos gerados representam hosts que fazem parte de uma grade.

O termo $(ke/N)$ é o fator de adaptação proposto por Doar e Leslie. O valor de $k$, presente no fator, deve ser tal que gere um grafo com grau médio próximo do valor de $e$, o grau médio desejado. Para encontrar o valor de $k$, o método deve ser executado com vários valores e, aquele que gerar o grafo com grau médio mais próximo do valor de $e$, deve ser utilizado. Duas particularidades foram incluídas na implementação do método de Doar-Leslie. A primeira diz respeito ao valor de $e$, que não é passado como entrada. O seu valor é calculado através da equação $e = \beta N$. A segunda diz respeito ao peso

**Algoritmo 7** Método Doar-Leslie

**Entrada:** $N$: quantidade de nós da rede ($N \in \mathbb{N}^*$).
$\alpha$: relação entre a quantidade de enlaces de longa distância e a quantidade de enlaces de curta distância ($\alpha \in \mathbb{R}_+^*, \alpha \leq 1$).
$\beta$: fator que define a quantidade de enlaces de cada nó ($\beta \in \mathbb{R}_+^*, \beta \leq 1$).
$e$: grau médio dos nós da rede ($e \in \mathbb{N}^*, e \leq N$).
$k$: constante para garantir que o grau do grafo gerado seja próximo do valor de $e$ ($k \in \mathbb{N}^*$).
**Saída:** Grafo aleatório com a topologia de uma rede.

1: **repita**
2: Distribua $N$ vértices sobre um plano cartesiano em posições definidas aleatoriamente por uma distribuição uniforme.
3: $L$ = maior distância euclidiana entre dois vértices.
4: **para** cada par de vértices $x, y$ **faça**
5: $d$ = distância euclidiana entre $x$ e $y$.
6: $P = \beta(ke/N)\exp(-d/(L\alpha))$
7: Assuma $P$ como a probabilidade de haver uma aresta entre $x$ e $y$ e sorteie a existência da aresta.
8: **se** pelo sorteio anterior, houver uma aresta **então**
9: Crie uma aresta com peso $d$ entre os vértices $x$ e $y$.
10: **fim se**
11: **fim para**
12: **até** Grafo gerado ser conexo

dos hosts, atributo que não é definido pelo método de Doar-Leslie. Eles são atribuídos seguindo uma distribuição uniforme no intervalo $(0, 4 - 2]$.

Como o peso dos enlaces, para os experimentos realizados, representa o tempo necessário para transferir uma certa quantidade de bits, no restante da dissertação será considerado que $\alpha$ define a relação entre a quantidade de enlaces com menos banda disponível e a quantidade de enlaces com mais banda disponível (a referência para a classificação dos enlaces como "menos" ou "mais" é a média da banda disponível de todos os enlaces). O plano cartesiano utilizado nos experimentos é composto por eixos com pontos pertencentes a $\mathbb{R}$ e tem dimensão 5×5, o que faz os pesos dos enlaces variarem no intervalo $[0, 5\sqrt{2}]$.

Nos experimentos realizados, são comparados os tempos de execução e os comprimentos dos escalonamentos encontrados por cada escalonador. Para cada um dos dois tipos de DAG, são realizadas variações na topologia da grade através dos parâmetros $N$, $\beta$ e $\alpha$ do método Doar-Leslie. Quando não for explicitamente informado, a topologia da grade simulada possui $N = 50$, $\alpha = 0,9$ e $\beta = 0,5$ nos experimentos realizados com o DAG do primeiro tipo, e $N = 25$, $\alpha = 0,9$ e $\beta = 0,5$ nos experimentos realizados com o DAG do

segundo tipo.

As métricas utilizadas para comparar os escalonadores são o tempo de execução do escalonador e o *speedup* do escalonamento encontrado. O valor do *speedup* é calculado por $T_{max}/makespan$, onde $makespan$ representa o comprimento do escalonamento encontrado. Quanto mais distante de 1, melhor a qualidade do escalonamento encontrado. Os resultados dos experimentos são exibidos em tabelas, com o melhor *speedup* conseguido por um escalonador apresentado em negrito. Os *speedups* conseguidos pelos demais escalonadores, diferentes do melhor, são apresentados em termos de porcentagem do melhor, ou seja, $(speedup_{conseguido}/speedup_{melhor}) \times 100\%$. O tempo de execução para o escalonador que conseguiu o melhor *speedup* também é apresentado em negrito a fim de facilitar a comparação do melhor escalonamento encontrado com o tempo de execução que foi necessário. A fim de comparar todos os escalonadores com um mesmo valor, as tabelas que fornecem os *speedups* alcançados também fornecem o maior *speedup* possível, que é calculado por $T_{max}/T_{min}$, onde $T_{min}$ é o limite inferior téorico para o tempo de execução de uma aplicação representada por um DAG. $T_{min}$ é igual à soma dos tempos de execução das tarefas que formam o maior caminho do DAG, considerando que todas elas são executadas seqüencialmente no host mais rápido da grade. O maior caminho é encontrado em termos do peso das tarefas.

É importante observar que o escalonador PLMTC não tem seus resultados apresentados para os experimentos com o DAG do primeiro tipo pois, para o ambiente utilizado, o seu tempo de execução é muito maior do que o exigido pelos outros escalonadores, além de apresentar um crescimento bem mais acentuado do que todos os outros à medida que a topologia da grade torna-se maior. Para um experimento com $N = 10$, o PLMTC consumiu $8,28$ segundos, enquanto que, dos outros escalonadores, o mais lento foi o TD-R1 com $0,52$ segundos. Já no experimento com $N = 40$, o PLMTC teve o tempo de execução maior que 1 hora, enquanto que, dos outros escalonadores, o mais lento foi o PLITD com um tempo de execução igual a $12,3$ segundos. Por motivos semelhantes, os experimentos realizados com o DAG do segundo tipo, além de não terem os resultados alcançados com o PLMTC apresentados, também não têm os resultados alcançados com os escalonadores TC-R1, TC-R2, TD-R2 e PLITD apresentados.

As subseções a seguir apresentam os resultados alcançados para cada uma das modificações na topologia da grade. Da Subseção 5.8.1 à Subseção 5.8.3 são apresentados resultados alcançados nos experimentos com o primeiro tipo de DAG. Da Subseção 5.8.4 à Subseção 5.8.6 são apresentados resultados alcançados nos experimentos com o segundo tipo de DAG. Na Subseção 5.8.7 são apresentadas as conclusões a respeito dos resultados obtidos nos experimentos.

### 5.8.1 DAG do primeiro tipo com variação da quantidade de hosts

As Tabelas 5.1 e 5.2 apresentam os resultados dos experimentos com variação na quantidade de hosts da grade ($N$) para o primeiro tipo de DAG. Os valores sob o título "Máximo" representam o limite superior do *speedup* para cada configuração da grade.

Pelos resultados exibidos, nota-se que os melhores escalonamentos foram conseguidos pelos escalonadores baseados em PL, com uma única exceção em que o SC conseguiu o melhor escalonamento, mas que não ficou muito distante dos escalonamentos encontrados pelos TD-R1 e PLITD. Os escalonadores TD-R2 e TC-R2 de modo geral foram os que apresentaram os piores resultados. Isso se deve ao fato de que o sorteio do mapeamento ocorre uma única vez, o que aumenta a probabilidade de que o escalonamento sugerido não seja o melhor. O escalonador PLITD produziu a maioria dos melhores escalonamentos, seguido pelo TC-R1 e pelo SC. Os escalonamentos encontrados pelos escalonadores baseados em sorteio apresentam uma queda no *speedup* com o aumento de $N$. Isso é justificado pelo fato de que, com mais hosts, a probabilidade de que o melhor escalonamento seja conseguido diminui caso ele seja feito de forma aleatória. Os escalonamentos encontrados pelo SC sempre foram melhores que os encontrados pelo SU, o que mostra que a atribuição "consciente" das probabilidades iniciais faz diferença. Para todos os experimentos, os resultados produzidos pelos TD-R1 e PLITD sempre ficaram muito próximos, sendo que os escalonamentos conseguidos com o segundo só não foram melhores para dois valores de $N$.

| $N$ | *Speedup* | | | | | | | Máximo |
|---|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC | |
| 10 | 77,72% | 77,72% | 99,31% | 77,55% | 98,51% | 99,07% | **1,289432** | 1,986486 |
| 40 | 89,21% | 73,36% | 99,86% | 73,26% | **1,365060** | 81,79% | 85,10% | 1,98437 |
| 70 | **1,556116** | 64,34% | 91,51% | 82,48% | 99,38% | 77,25% | 84,52% | 1,983607 |
| 100 | **1,534463** | 65,55% | 97,73% | 65,17% | 94,18% | 70,73% | 79,10% | 2,016393 |
| 130 | 94,38% | 66,71% | 91,73% | 66,23% | **1,509969** | 69,67% | 75,53% | 2 |
| 160 | 93,43% | 62,23% | 91,33% | 62,11% | **1,610028** | 68,26% | 73,84% | 1,983607 |
| 190 | 62,49% | 62,49% | 81,82% | 62,27% | **1,606021** | 65,29% | 74,71% | 1,983333 |

Tabela 5.1: *Speedup* para o DAG do primeiro tipo com variação de $N$

Nota-se que para a maioria dos escalonadores, o tempo de execução aumenta com o aumento de $N$. Nota-se, também, que o tempo de execução dos escalonadores baseados em PL crescem de forma mais acentuada que os baseados em sorteio. Enquanto os últimos

| $N$ | Tempo de execução (segundos) | | | | | | |
|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC |
| 10 | 0,28 | 0,05 | 0,52 | 0,21 | 0,17 | 0,08 | **0,07** |
| 40 | 3,64 | 0,64 | 1,63 | 2,00 | **12,30** | 0,60 | 0,48 |
| 70 | **16,87** | 2,47 | 5,14 | 5,19 | 4,38 | 1,80 | 1,38 |
| 100 | **78,02** | 7,32 | 10,02 | 9,55 | 17,80 | 3,40 | 2,56 |
| 130 | 71,12 | 19,92 | 17,29 | 23,96 | **81,31** | 5,86 | 4,49 |
| 160 | 119,98 | 55,83 | 26,16 | 33,18 | **24,85** | 8,56 | 6,54 |
| 190 | 345,77 | 82,04 | 25,64 | 32,05 | **134,03** | 11,89 | 8,98 |

Tabela 5.2: Tempo de execução para o DAG do primeiro tipo com variação de $N$

executam na ordem de unidades de segundos, e assim permanecem com o aumento de $N$, os primeiros alcançam rapidamente a ordem de dezenas e centenas de segundos. Nota-se que, na grande maioria dos experimentos, o relaxamento das variáveis diminuiu bastante o tempo de execução dos escalonadores baseados em PL. Para uma grade com 190 hosts, o tempo de execução de cerca de 134,03 segundos para o PLITD cai para 25,64 segundos com o relaxamento do escalonador TD-R1 e para 32,05 segundos com o relaxamento do escalonador TD-R2. O PLITD e o TC-R1 são os escalonadores com crescimento mais acentuado de todos. Entre os escalonadores baseados em sorteio, o SC apresenta menores tempos de execução do que o SU, diferente do previsto. Isso ocorre porque em cada iteração no algoritmo SC, há menos opções de hosts para que as tarefas sejam executadas, já que as probabilidades iniciais descartam os piores hosts. Com uma menor opção de hosts, cada iteração do algoritmo tende a executar mais rápido, diminuindo assim o tempo de execução do escalonador como um todo. O que leva à previsão errada é o fato da definição das probabilidades iniciais deste escalonador ser mais custosa computacionalmente do que no SU. Na prática, entretanto, esse custo é compensado pelo ganho alcançado na execução mais rápida de cada iteração.

Nota-se que para quatro das sete variações de $N$ realizadas, o melhor escalonamento foi encontrado pelo escalonador que teve o maior tempo de execução. Em uma das outras três variações, o melhor escalonamento foi encontrado pelo escalonador que teve o segundo maior tempo de execução.

Comparando-se o melhor escalonamento encontrado para cada variação, com o limite superior, nota-se que, em média, o melhor escalonamento foi 75,13% do máximo.

### 5.8.2 DAG do primeiro tipo com variação do grau médio dos hosts

Os resultados dos experimentos que variam o grau médio dos hosts da grade ($\beta$) para o DAG do primeiro tipo são apresentados nas tabelas 5.3 e 5.4.

| $\beta$ | *Speedup* | | | | | | | Máximo |
|---|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC | |
| 0,1 | 71,06% | 71,06% | 84,72% | 81,08% | 83,81% | 96,38% | **1,408825** | 2 |
| 0,22 | 72,97% | 72,97% | 96,23% | 72,55% | 96,46% | 89,29% | **1,378274** | 2,016667 |
| 0,34 | 70,09% | 69,64% | 91,51% | 69,64% | **1,435858** | 97,30% | 98,64% | 2,015151 |
| 0,46 | 98,17% | 66,10% | **1,517342** | 65,90% | 97,64% | 81,37% | 92,93% | 2 |
| 0,58 | 89,92% | 66,10% | **1,522505** | 69,92% | 99,64% | 89,51% | 92,97% | 2,016667 |
| 0,7 | 98,82% | 65,32% | 98,82% | 65,31% | **1,531184** | 77,78% | 90,87% | 2 |
| 0,82 | 91,71% | 66,10% | **1,524155** | 77,74% | 65,61% | 73,64% | 87,03% | 2,016129 |

Tabela 5.3: *Speedup* para o DAG do primeiro tipo com variação de $\beta$

| $\beta$ | Tempo de execução (segundos) | | | | | | |
|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC |
| 0,1 | 0,82 | 0,56 | 1,69 | 0,86 | 18,84 | 1,38 | **1,10** |
| 0,22 | 5,97 | 0,71 | 1,66 | 1,33 | 12,21 | 1,38 | **1,09** |
| 0,34 | 4,56 | 1,24 | 2,33 | 1,41 | **32,28** | 1,26 | 0,99 |
| 0,46 | 4,01 | 1,24 | **3,53** | 3,08 | 7,49 | 1,08 | 0,81 |
| 0,58 | 4,85 | 0,93 | **2,35** | 1,77 | 2,22 | 0,96 | 0,73 |
| 0,7 | 7,30 | 1,73 | 3,04 | 3,69 | **3,46** | 0,38 | 0,30 |
| 0,82 | 9,98 | 1,24 | **2,60** | 3,05 | 4,29 | 0,10 | 0,09 |

Tabela 5.4: Tempo de execução para o DAG do primeiro tipo com variação de $\beta$

Os escalonadores PLITD e TD-R1 produziram os melhores *speedups* com exceção de dois valores de $\beta$, nos quais o SC saiu-se melhor. Somente para um dos valores de $\beta$, os resultados produzidos pelos escalonadores TD-R1 e PLITD apresentam uma grande diferença. Assim como nos experimentos com variação no valor de $N$, os escalonamentos conseguidos pelos escalonadores TC-R2 e TD-R2 foram os piores. Entre os escalonadores baseados em sorteio, o SC sempre apresentou os melhores resultados.

Pelos dados das tabelas, é possível notar que o tempo de execução do escalonador TC-R1 cresce com o aumento de $\beta$, porém de uma forma bem menos acentuada do que nos

experimentos com variação de $N$. Os demais escalonadores baseados em PL não seguiram um padrão com relação ao tempo de execução. De modo geral, os demais escalonadores que relaxam as variáveis inteiras, aumentaram o tempo de execução também de forma menos acentuada do que nos experimentos com variação de $N$. De um modo geral, o tempo de execução do PLITD para topologias com maior valor de grau médio dos hosts, é menor do que para topologias com menor valor do grau médio. Isso ocorre porque, no primeiro caso, o problema de PL possui menos restrições para serem analisadas e consegue-se encontrar um escalonamento viável em um tempo menor. Já para os escalonadores baseados em sorteio, o tempo de execução diminui com o aumento de $\beta$, sendo que os seus tempos foram sempre menores do que os tempos dos escalonadores baseados em PL. Isso é justificado pelo fato dos algoritmos 3 e 6 executarem o passo da linha 5 mais rápidos, já que a quantidade de probabilidades atribuídas a zero é menor, por haver mais hosts conectados entre si, diminuindo assim a chance de executar a normalização das probabilidades várias vezes. O tempo de execução do SC mantém-se menor do que o do SU, assim como ocorreu nos experimentos com variação de $N$. De modo geral, os tempos de execução são menores do que os dos experimentos anteriores.

A relação entre melhor escalonamento e maior tempo de execução não foi tão uniforme como nos experimentos com variação de $N$. Somente para o valor de $\beta = 0,82$, o melhor escalonamento foi encontrado pelo escalonador que exigiu maior tempo de execução. Para os outros seis valores de $\beta$, em quatro deles (0,22, 0,46, 0,7 e 0,82) o escalonador que levou o maior tempo de execução teve o escalonamento encontrado no máximo 8,29% distante do melhor escalonamento.

Comparando-se o melhor escalonamento encontrado para cada variação, com o limite superior, nota-se que, em média, o melhor escalonamento foi 73,37% do máximo.

### 5.8.3 DAG do primeiro tipo com variação da quantidade de enlaces com menos banda disponível e com mais banda disponível

A última variação na topologia da grade para o DAG do primeiro tipo analisa o desempenho dos escalonadores frente a variações na relação entre a quantidade de enlaces com menos banda disponível e a quantidade de enlaces com mais banda disponível ($\alpha$). Os resultados são apresentados nas Tabelas 5.5 e 5.6

Para seis dos sete valores de $\alpha$, os melhores escalonamentos foram encontrados pelo PLITD, sendo que no único experimento em que o PLITD não foi o melhor, ele produziu um escalonamento 7,73% pior do que o melhor. Entre os escalonadores baseados em sorteio, o SC só foi pior que o SU em um experimento. A diferença entre os *speedups* conseguidos pelo PLITD e pelo TD-R1 é pequena para a maioria dos experimentos (Somente

| $\alpha$ | *Speedup* | | | | | | | Máximo |
|---|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC | |
| 0,2 | 72,81% | 72,81% | 90,99% | 72,42% | **1,380837** | 77,84% | 89,72% | 2,016129 |
| 0,3 | 92,53% | 61,79% | 92,53% | 89,71% | **1,624240** | 78,61% | 84,50% | 2 |
| 0,4 | 100,00% | 68,55% | **1,469889** | 81,06% | 92,27% | 86,55% | 78,44% | 2 |
| 0,5 | 74,89% | 74,89% | 92,55% | 83,85% | **1,341922** | 85,77% | 90,06% | 2,016393 |
| 0,6 | 87,12% | 64,61% | 92,15% | 86,03% | **1,552311** | 75,85% | 86,59% | 1,983333 |
| 0,7 | 68,58% | 68,58% | 82,75% | 71,95% | **1,459533** | 75,38% | 82,35% | 1,983333 |
| 0,8 | 91,03% | 68,17% | 93,71% | 67,72% | **1,476768** | 83,85% | 86,73% | 2 |

Tabela 5.5: *Speedup* para o DAG do primeiro tipo com variação de $\alpha$

| $\alpha$ | Tempo de execução (segundos) | | | | | | |
|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC |
| 0,2 | 7,71 | 1,24 | 2,11 | 2,60 | **13,64** | 1,06 | 0,81 |
| 0,3 | 4,90 | 1,02 | 2,45 | 1,88 | **9,75** | 1,00 | 0,75 |
| 0,4 | 4,29 | 1,22 | **2,26** | 2,79 | 5,22 | 1,08 | 0,84 |
| 0,5 | 3,33 | 2,02 | 2,22 | 2,64 | **3,56** | 0,93 | 0,73 |
| 0,6 | 5,67 | 1,36 | 2,22 | 3,09 | **2,10** | 0,99 | 0,77 |
| 0,7 | 3,67 | 1,08 | 2,22 | 2,51 | **3,28** | 0,98 | 0,76 |
| 0,8 | 4,88 | 0,95 | 2,37 | 1,98 | **29,85** | 0,93 | 0,70 |

Tabela 5.6: Tempo de execução para o DAG do primeiro tipo com variação de $\alpha$

para um valor de $\alpha$ ela é maior que 10%).

Pelos resultados encontrados, nota-se que não há um padrão perceptível no tempo de execução dos escalonadores, com exceção do escalonador TD-R1 que mantém o seu tempo de execução estável. O PLITD foi o escalonador que apresentou o maior tempo de execução para a maioria dos experimentos. De modo geral, os valores dos tempos de execução foram próximos dos alcançados nos experimentos com variação no valor de $\beta$.

Na maioria dos experimentos, os melhores escalonamentos foram encontrados pelos escalonadores que exigiram mais tempo de execução. Na maioria dos experimentos em que isso não ocorreu, a diferença para o melhor escalonamento foi no máximo 7,73%.

Comparando-se o melhor escalonamento encontrado para cada variação, com o limite superior, nota-se que, em média, o melhor escalonamento foi 73,63% do máximo.

### 5.8.4 DAG do segundo tipo com variação da quantidade de hosts

Os primeiros resultados dos experimentos realizados com o DAG do segundo tipo são apresentados nas Tabelas 5.7 e 5.8. Essas tabelas apresentam o desempenho dos escalonadores TD-R1, SU e SC sob variações na quantidade de hosts da grade, quantidade que é representada pelo valor de $N$. Ao contrário dos experimentos realizados com o DAG do primeiro tipo, a variação de $N$ não ocorre no intervalo $[10, 190]$, mas sim no intervalo $[10, 40]$, já que o tempo de execução dos escalonadores para o DAG do segundo tipo sofre um crescimento mais acentuado, com o aumento de $N$, do que para o DAG do primeiro tipo, o que faz com que os experimentos levem mais tempo para serem realizados. Esse crescimento mais acentuado ocorre devido à estrutura mais complexa do DAG, tanto em matéria de quantidade de tarefas, quanto em quantidade de arcos e níveis.

| $N$ | *Speedup* | | | Máximo |
|---|---|---|---|---|
| | TD-R1 | SU | SC | |
| 10 | 98,85% | 89,13% | **1,417471** | 3,215686 |
| 15 | **2,157490** | 90,26% | 95,42% | 3,213166 |
| 20 | **1,612934** | 74,57% | 86,38% | 3,213018 |
| 25 | 95,96% | 83,70% | **1,557311** | 3,225989 |
| 30 | **1,648856** | 76,96% | 88,31% | 3,229885 |
| 35 | **1,268355** | 89,61% | 88,15% | 3,226994 |
| 40 | **1,445182** | 82,24% | 86,77% | 3,221591 |

Tabela 5.7: *Speedup* para o DAG do segundo tipo com variação de $N$

| $N$ | Tempo de execução (segundos) | | |
|---|---|---|---|
| | TD-R1 | SU | SC |
| 10 | 12,18 | 0,24 | **0,30** |
| 15 | **489,09** | 0,51 | 0,54 |
| 20 | **23,27** | 0,66 | 0,74 |
| 25 | 503,10 | 1,02 | **1,19** |
| 30 | **2580,34** | 1,43 | 1,83 |
| 35 | **746,31** | 1,86 | 2,18 |
| 40 | **117,55** | 2,30 | 2,72 |

Tabela 5.8: Tempo de execução para o DAG do segundo tipo com variação de $N$

Pelos resultados exibidos, nota-se que o escalonador TD-R1 gerou os melhores escalonamentos para cinco das sete variações realizadas em $N$. Nas duas variações em que o TD-R1 não gerou os melhores escalonamentos, os mesmos foram gerados pelo escalonador SC. Nesses casos, o escalonamento encontrado pelo TD-R1 foi, no mínimo, 95,96% daquele encontrado pelo SC. O escalonador SU gerou os piores escalonamentos para todas as variações de $N$, o que mostra que, mesmo para um DAG diferente, a atribuição consciente das probabilidades feita pelo SC é eficiente.

Assim como nos experimentos realizados com o DAG do primeiro tipo, o tempo de execução dos escalonadores baseados em sorteio é da ordem de unidades de segundos e aumenta linearmente com o aumento de $N$. O tempo de execução do escalonador TD-R1 não apresenta um comportamento padronizado com o aumento de $N$. Uma característica que pode ser notada é o fato dos tempos de execução serem bem maiores do que aqueles gastos por esse mesmo escalonador com o DAG do primeiro tipo. Para o experimento realizado com $N$ igual a 30, o tempo de execução do TD-R1 foi de aproximadamente 43 minutos. No geral, o tempo de execução do TD-R1 fica na ordem de centenas de segundos, o que o torna ineficiente para ser utilizado em grades que exigam que um escalonamento seja encontrado em um intervalo da ordem de segundos. Ao contrário do verificado nos experimentos com o DAG do primeiro tipo, o tempo de execução do escalonador SC foi um pouco maior que o do escalonador SU. Esse comportamento ocorre devido à estrutura mais complexa do DAG, o que faz com que o SC forneça mais opções de hosts para executar cada tarefa, ao contrário do que ocorreu com o DAG mais simples, do primeiro tipo. Com uma quantidade maior de hosts para executar cada tarefa, as iterações do SC e do SU tendem a executar em um tempo mais próximo, o que faz com que o processo de atribuição das probabilidades iniciais faça diferença ao definir o tempo de execução do escalonador. Como o SC exige mais tempo de processamento para o cálculo das probabilidades iniciais, o seu tempo de execução torna-se maior do que o tempo de execução do SU.

Comparando-se o tempo de execução de cada escalonador com a qualidade do escalonamento encontrado, nota-se que os melhores escalonamentos foram encontrados pelo escalonador que exigiu maior tempo de processamento em cinco das sete variações de $N$. Nas outras duas variações, o escalonador que exigiu maior tempo de processamento encontrou o segundo melhor escalonamento.

Comparando-se o melhor escalonamento encontrado para cada valor de $N$, com o limite superior, nota-se que, em média, o melhor escalonamento foi 49,27% do máximo, um valor menor do que aquele calculado para os experimentos com o DAG do primeiro tipo devido à maior quantidade de arcos e tarefas do DAG, o que diminui a probabilidade de que o limite superior do *speedup* possa ser alcançado na prática.

### 5.8.5 DAG do segundo tipo com variação do grau médio dos hosts

Os resultados dos experimentos que variam o valor de $\beta$ da grade para o DAG do segundo tipo são apresentados nas Tabelas 5.9 e 5.10.

| $\beta$ | *Speedup* | | | Máximo |
|---|---|---|---|---|
| | TD-R1 | SU | SC | |
| 0,10 | **1,312589** | 93,72% | 90,26% | 3,210811 |
| 0,22 | 97,46% | 89,02% | **1,451858** | 3,212291 |
| 0,34 | **1,433219** | 84,35% | 93,55% | 3,22807 |
| 0,46 | **1,407436** | 92,18% | 99,47% | 3,217617 |
| 0,58 | **1,480795** | 85,95% | 99,70% | 3,221622 |
| 0,70 | 92,77% | 71,06% | **1,601030** | 3,222892 |
| 0,82 | **1,681045** | 73,43% | 88,12% | 3,215116 |

Tabela 5.9: *Speedup* para o DAG do segundo tipo com variação de $\beta$

| $\beta$ | Tempo de execução (segundos) | | |
|---|---|---|---|
| | TD-R1 | SU | SC |
| 0,10 | **26,69** | 1,36 | 1,55 |
| 0,22 | 28,58 | 1,18 | **1,35** |
| 0,34 | **48,62** | 1,04 | 1,29 |
| 0,46 | **445,61** | 1,02 | 1,20 |
| 0,58 | **47,52** | 1,01 | 1,22 |
| 0,70 | 54,15 | 0,53 | **0,55** |
| 0,82 | **41,22** | 0,60 | 0,60 |

Tabela 5.10: Tempo de execução para o DAG do segundo tipo com variação de $\beta$

Assim como nos experimentos com variação de $N$, os melhores escalonamentos foram encontrados pelo escalonador TD-R1 para cinco dos sete valores de $\beta$, e pelo escalonador SC para os outros dois. O escalonador SU encontrou os piores escalonamentos para seis dos sete valores de $\beta$. A única exceção foi para $\beta$ igual a 0,10, em que o pior escalonamento foi encontrado pelo SC.

Assim como nos experimentos com variação de $\beta$ para o DAG do primeiro tipo, o tempo de execução dos escalonadores baseados em sorteio são os menores e decrescem

com o crescimento de $\beta$. Os tempos de execução do escalonador TD-R1 não segue um padrão de crescimento ou decrescimento. De modo geral, os tempos de execução são menores do que aqueles gastos para os experimentos com variação de $N$, mas são muito maiores do que os gastos pelos escalonadores nos experimentos realizados com o DAG do primeiro tipo.

Comparando-se o tempo de execução de cada escalonador com a qualidade do escalonamento encontrado, nota-se que, assim como nos experimentos com variação de $N$, os melhores escalonamentos foram encontrados pelo escalonador que exigiu maior tempo de processamento em cinco das sete variações de $\beta$. Nas outras duas variações, o escalonador que exigiu maior tempo de processamento encontrou o segundo melhor escalonamento.

Comparando-se o melhor escalonamento encontrado para cada valor de $\beta$, com o limite superior, nota-se que, em média, o melhor escalonamento foi 46,02% do máximo.

### 5.8.6 DAG do segundo tipo com variação da quantidade de enlaces com menos banda disponível e com mais banda disponível

A última variação na topologia da grade para o DAG do segundo tipo analisa o desempenho dos escalonadores sob variações no valor de $\alpha$. As Tabelas 5.11 e 5.12 apresentam os resultados dos experimentos realizados com a variação de $\alpha$.

| $\alpha$ | *Speedup* | | | Máximo |
|---|---|---|---|---|
| | TD-R1 | SU | SC | |
| 0,2 | 98,17% | 99,88% | **1,919990** | 3,209559 |
| 0,3 | **1,590171** | 77,69% | 86,87% | 3,213873 |
| 0,4 | **1,569223** | 82,84% | 95,81% | 3,224852 |
| 0,5 | **1,655768** | 81,74% | 89,97% | 3,214612 |
| 0,6 | 83,56% | 83,60% | **1,592988** | 3,212644 |
| 0,7 | **1,438691** | 81,28% | 84,83% | 3,222892 |
| 0,8 | **1,701353** | 73,00% | 84,82% | 3,227273 |

Tabela 5.11: *Speedup* para o DAG do segundo tipo com variação de $\alpha$

Assim como nos outros experimentos para o DAG do segundo tipo, apresentados anteriormente, com a variação de $\alpha$ o melhor escalonamento foi encontrado pelo TD-R1 para cinco dos sete valores. Nos outros dois valores, o melhor escalonamento foi encontrado pelo SC.

| $\alpha$ | Tempo de execução (segundos) | | |
|---|---|---|---|
| | TD-R1 | SU | SC |
| 0,2 | 67,07 | 1,07 | **1,19** |
| 0,3 | **146,37** | 1,06 | 1,21 |
| 0,4 | **40,22** | 0,99 | 1,14 |
| 0,5 | **65,52** | 1,07 | 1,25 |
| 0,6 | 107,37 | 1,08 | **1,36** |
| 0,7 | **33,26** | 1,08 | 1,22 |
| 0,8 | **40,95** | 1,05 | 1,27 |

Tabela 5.12: Tempo de execução para o DAG do segundo tipo com variação de $\alpha$

Com relação ao tempo de execução, não há um padrão perceptível para nenhum dos escalonadores. De modo geral, o escalonador TD-R1 executa na ordem de dezenas de segundos, enquanto os escalonadores baseados em sorteio têm tempo de execução menor que 2 segundos. Para todos os valores de $\alpha$, o SU executou mais rápido que o SC.

Relacionando-se a qualidade do escalonamento encontrado com o tempo de execução do escalonador, nota-se que os melhores escalonamentos foram encontrados pelo escalonador mais lento para cinco dos sete valores de $\alpha$. Para os dois valores de $\alpha$ em que isso não ocorreu, o escalonador mais lento encontrou o segundo melhor escalonamento.

Comparando-se o melhor escalonamento encontrado para cada valor de $\alpha$ com o limite superior, nota-se que, em média, o melhor escalonamento foi 50,92% do máximo.

### 5.8.7 Discussão dos resultados encontrados

Pelos resultados alcançados com os experimentos que avaliaram os escalonadores com o primeiro tipo de DAG, pode-se concluir que os escalonadores baseados no Algoritmo 2 não apresentaram bons resultados. Os demais escalonadores baseados em PL tiveram resultados próximos entre si, sendo que aqueles baseados no Algoritmo 1 ficaram próximos do melhor escalonamento na grande maioria dos experimentos e com tempos de execução entre os menores. O melhor escalonador, com relação ao *speedup* do escalonamento encontrado, foi o PLITD. Como ele também é o que leva, na maioria das vezes, o maior tempo para executar e não apresenta resultados muito distantes do TD-R1, que executa bem mais rápido, é razoável utilizar o TD-R1 em situações nas quais uma pequena diferença para o melhor escalonamento seja tolerada. Como era de se esperar, os escalonadores baseados em sorteios executam mais rápidos mas não apresentam resultados tão bons quanto os encontrados pelos TC-R1, TD-R1 e PLITD. O escalonador SC apresentou me-

lhores resultados que o SU, exceto em um experimento, o que mostra que a atribuição "consciente" das probabilidades faz diferença no escalonamento encontrado. O SC é uma opção de escalonador a ser usado caso o tempo de execução devolvido pelo TD-R1 seja considerado grande para que um escalonamento viável seja devolvido para a aplicação.

Os resultados alcançados com os experimentos que avaliaram os escalonadores com o segundo tipo do DAG apresentam situações em que o SC é uma boa opção a ser utilizada. Apesar do escalonador TD-R1 ter encontrado o melhor escalonamento em 15 das 21 topologias de rede utilizadas, o seu tempo de execução foi muito elevado, chegando quase a 45 minutos de execução. Os escalonamentos encontrados pelo SC foram os melhores em 6 topologias e foram os segundos melhores nas outras 15. Assim como nos experimentos realizados com o DAG do primeiro tipo, nos experimentos com o DAG do segundo tipo o SC só não encontrou um escalonamento melhor que o SU para uma única topologia de grade.

# Capítulo 6

# Engenharia de tráfego para grades

A topologia formada pelos recursos compartilhados de uma grade pode ser considerada como um subconjunto da topologia formada pelos vários nós da Internet. É necessário que haja uma análise cuidadosa ao se definir onde cada tarefa será executada na grade, assim como é necessário uma análise a respeito da melhor rota que um pacote seguirá na Internet. Como visto nos capítulos anteriores, a definição do host onde cada tarefa executará deve ser realizada de modo a aproveitar, da melhor forma possível, a taxa de processamento disponível em cada um dos hosts compartilhados e a banda disponível em cada um dos enlaces. Tal definição deve levar em consideração as transferências de dados entre tarefas dependentes.

Uma vez as tarefas alocadas, é natural que ocorram mudanças no estado dos hosts da grade e dos enlaces que os interligam. Dessa forma, é necessário observar e medir o estado dos recursos a fim de avaliar o ganho que pode ser alcançado com a migração de tarefas, do mesmo modo que ganhos são alcançados com rotas alternativas para os pacotes na Internet. Isso leva à conclusão de que, assim como a Internet, grades precisam de técnicas que analisem os requisitos das aplicações, monitorem os hosts e a rede e otimizem o desempenho.

Na Internet, as técnicas são coordenadas pela engenharia de tráfego. Por definição, a engenharia de tráfego para Internet concentra-se na otimização do desempenho de redes IP operacionais [2]. Ela engloba a aplicação de tecnologias e de princípios metodológicos para medir, modelar, caracterizar e controlar o tráfego da Internet [3]. Existem as mais diversas técnicas que podem ser aplicadas a fim de alcançar os objetivos da engenharia de tráfego. Essas técnicas buscam projetar e otimizar uma rede a fim de que ela atenda os requisitos de qualidade de serviço das aplicações que utilizam a Internet.

Para as grades, é apresentada neste capítulo, a metodologia "engenharia de tráfego para grades", que possui, em essência, o mesmo objetivo da engenharia de tráfego para Internet, que é otimizar a utilização da rede e dos hosts compartilhados a fim de atender

os requisitos de qualidade de serviço das aplicações, nesse caso, específicas para grades. A engenharia de tráfego para grades segue a mesma idéia geral das propostas de otimização dinâmica de aplicações para grades, ou seja, há o monitoramento do estado dos recursos e a verificação de ganhos com migrações de tarefas.

A diferença da engenharia de tráfego para outras propostas semelhantes [1, 52, 26] está no fato dela considerar o custo causado pelas transferências de dados tanto no escalonamento quanto na migração das tarefas, sendo que o custo para este último caso não é fixo e depende do progresso de execução das tarefas. Os dados necessários para que uma tarefa continue a sua execução, caso haja migração, são transferidos diretamente entre os dois hosts envolvidos: o que foi alocado inicialmente para executar a tarefa e aquele para onde a tarefa migrará. Diferentemente da proposta apresentada em [1], não é considerado que há um repositório provisório para os dados da migração. Diferentemente da proposta apresentada em [52], não há um limite mínimo de ganho para que as migrações ocorram.

Para ilustrar a necessidade da engenharia de tráfego para grades, a Figura 6.1 e a Figura 6.2 apresentam um cenário de exemplo no qual ela é utilizada para otimizar a execução de uma aplicação.

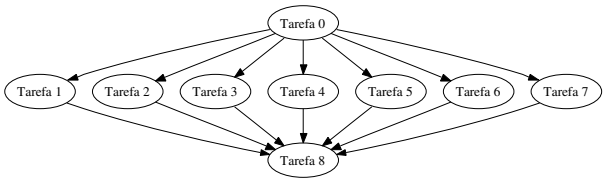
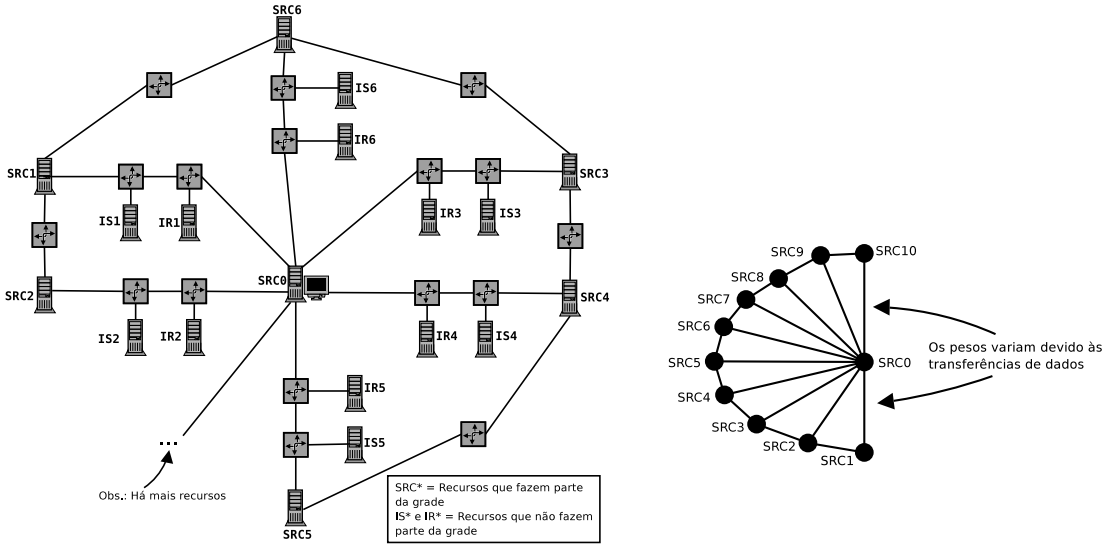



Figura 6.1: Exemplo de DAG para ilustrar a aplicação da engenharia de tráfego

Pode-ser ver na Figura 6.1, o DAG de uma aplicação que será executada em uma organização virtual formada pelos hosts *SRCi* ($i \in \{0, 1, \ldots, 10\}$), que são interligados de acordo com a topologia detalhada na Figura 6.2(a). A Figura 6.2(b) ilustra o grafo da rede contendo somente os hosts que fazem parte da organização virtual.

Se todos os hosts compartilhados fossem interligados por enlaces dedicados e tivessem suas taxas de processamento exclusivas para a grade, o mapeamento das tarefas seria bem simples e só precisaria ser feito uma única vez; a *Tarefa 1* seria a primeira a executar no host *SRC0*. Uma das tarefas intermediárias (*Tarefa 2* a *Tarefa 7*) seria executada no mesmo host *SRC0* e cada uma das demais executariam em um dos vários hosts *SRCi* ($i \neq 0$), após os dados serem transferidos da *Tarefa 1* para cada um deles. Os resultados das execuções das tarefas intermediárias seriam então enviados para o host *SRC0*, no qual a *Tarefa 8* seria executada.

Entretanto, os tráfegos concorrentes entre os hosts *IRi* e *ISi* (Figura 6.2(a)), contribuem com a carga dos enlaces que interligam os vários hosts da organização virtual. Além desses tráfegos concorrentes, também há aplicações que modificam a taxa de proces-




(a) Topologia da rede (b) Grafo da organização virtual

Figura 6.2: Exemplo de rede para ilustrar a aplicação da engenharia de tráfego

samento disponível nos hosts. Assim, algumas questões relacionadas com o desempenho da aplicação podem ser formuladas e, dentre elas, destacam-se estas: vale a pena manter as tarefas nos hosts inicialmente mapeados? Haverá algum ganho se a execução de uma tarefa for interrompida e transferida para um host com enlaces menos congestionados?

A proposta de engenharia de tráfego para grades visa responder essas perguntas. Ela consiste em um conjunto de passos que monitoram o estado dos hosts da grade, e dos enlaces que os interligam, e avaliam se aplicações em execução podem ter seu tempo de execução minimizado se houver migração de tarefas. O impacto que a migração causa na rede é levado em consideração, o que exige um monitoramento do progresso das tarefas a fim de avaliar se as mesmas estão próximas de finalizar a sua execução. Os passos devem ser executados periodicamente a fim de capturar variações no ambiente que possam causar aumento no tempo de execução das aplicações. A cada execução dos passos, o escalonamento proposto pode diferir do anterior e, caso isso aconteça, as tarefas que forem escalonadas para outros hosts devem ter o impacto da sua migração avaliado. Se a migração diminuir o tempo de execução da aplicação, a mesma deve ser realizada. A proposta resume-se nos passos apresentados no fluxograma da Figura 6.3, que são detalhados logo a seguir:

1. A partir de um DAG e um grafo representando, respectivamente, as dependências entre as tarefas da aplicação e a topologia da rede que interliga os vários hosts da grade, são definidos mapeamento e escalonamento que visam responder as seguintes perguntas: i) Em qual host uma tarefa será executada? ii) Em qual instante de

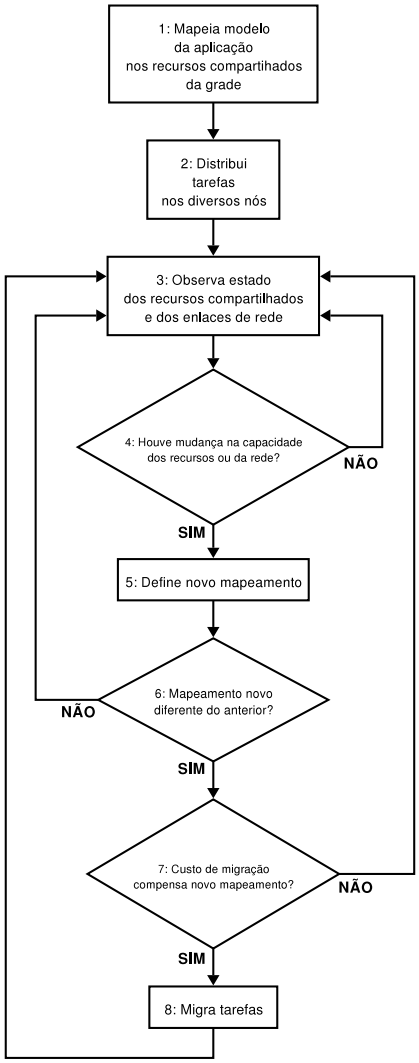



Figura 6.3: Proposta de engenharia de tráfego para grades

tempo as execuções e as transferências de dados devem ocorrer?

2. O código e os dados necessários para cada uma das tarefas são transferidos para os locais onde as mesmas serão executadas. Após as transferências, as execuções são iniciadas;

3. Durante a execução das tarefas, monitora-se os enlaces e os hosts a fim de se detectar mudanças que possam ocasionar aumento no tempo de execução da aplicação;

4. Após coletar o estado dos enlaces e dos hosts, compara-se o mesmo com o estado anterior. Caso haja alguma mudança, define-se um novo mapeamento para as tarefas

(Passo 5). Se o estado continuar o mesmo, volta-se ao processo de observação (Passo 3);

5. A partir do novo estado da grade, são definidos novos mapeamento e escalonamento para as tarefas. Desta vez, somente as tarefas que ainda não terminaram sua execução são analisadas;

6. O novo mapeamento encontrado é comparado com o mapeamento anterior. Se eles forem iguais, volta-se ao processo de observação (Passo 3). Caso contrário, segue-se para a verificação do ganho alcançado com a migração (Passo 7);

7. Com o novo mapeamento, analisa-se o custo para migrar as tarefas do local atual para o novo local. Se o custo compensar realiza-se a migração das tarefas (Passo 8). Caso contrário, volta-se ao processo de observação (Passo 3);

8. Dada a decisão de se migrar algumas tarefas, efetua-se a migração e retorna-se ao processo de observação (Passo 3).

O Passo 1 depende de um algoritmo que mapeie e escalone as tarefas nos hosts de uma grade. Esse algoritmo deve considerar a heterogeneidade da rede e dos hosts em uma grade, ao mesmo tempo em que encontra um escalonamento o mais rápido possível. Os algoritmos apresentados no Capítulo 5 são utilizados neste passo.

Nos Passos 2 e 8, não há a especificação de protocolos para a transferência dos dados. As características da grade podem impor restrições para os protocolos utilizados. Exemplos dessas características são requisitos de segurança, que exigirão protocolos que realizem autenticação ou que utilizem criptografia, e a interligação de hosts por enlaces de longa distância, que costumam representar problemas para a eficiência de protocolos padrões para transferência de dados. Exemplos de protocolos para transferência eficiente de grande quantidade de dados entre hosts interligados por enlaces de longa distância, e que podem ser utilizados, são o GridFTP [9] e o bbftp [33]. No Passo 8, considera-se que, após interromper a execução de uma tarefa em um dado ponto, é possível recomeçar a sua execução do ponto onde ela foi interrompida, e que é conhecida a quantidade de dados que devem ser transferidos para o novo local a fim de que a execução possa continuar. Os detalhes de como isso deve ser feito na prática não são apresentados nesta dissertação, mas são tratados em [41].

No Passo 3, devem ser utilizadas técnicas para monitorar e/ou prever o estado dos hosts e dos enlaces nas grades. Na prática, o NWS ou o GHS podem ser utilizados neste passo.

Os Passos 5, 6 e 7 podem ser realizados com os mesmos escalonadores do Passo 1. O objetivo a ser alcançado com o reescalonamento continua sendo o de minimizar o tempo

de execução da aplicação. Detalhes sobre a utilização dos escalonadores para verificar os ganhos com a migração das tarefas são apresentados na seção 6.1.

A metodologia de engenharia de tráfego para grades pode ser incluída na arquitetura mostrada na Figura 2.3, da forma como está detalhada na Figura 6.4. Os blocos em pontilhado são substituídos pelos passos da engenharia de tráfego: substitui-se blocos da Fase 2 e da Fase 3 por uma nova fase, que é responsável pela engenharia de tráfego nas grades. Os blocos não modificados da Fase 3 passam a fazer parte da Fase 4, responsável pelas ações de finalização das aplicações.

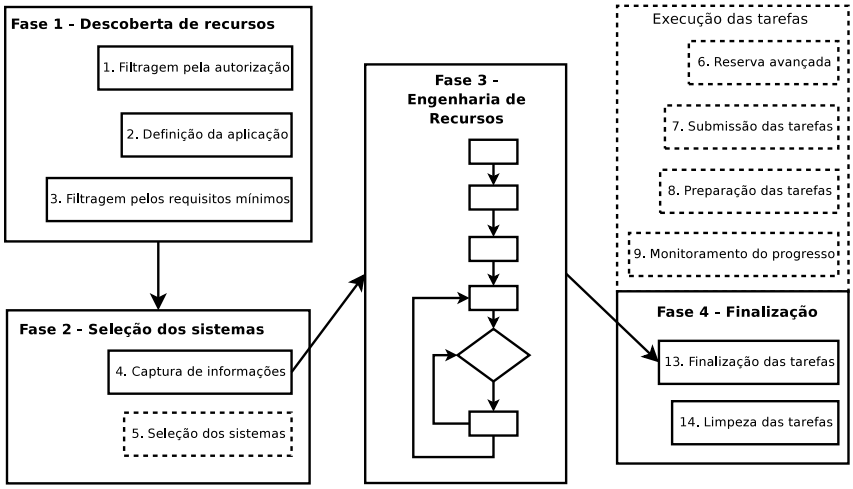



Figura 6.4: Inclusão da engenharia de tráfego no esquema da Figura 2.3

As seções deste capítulo apresentam mais informações relacionadas com a engenharia de tráfego para grades de acordo com a seguinte organização: a Seção 6.1 apresenta detalhes sobre como utilizar os escalonadores apresentados no capítulo anterior para verificar a necessidade de migração de tarefas. A Seção 6.2 apresenta, em linhas gerais, o `GridSim-NS`, um simulador de grades desenvolvido sobre o simulador de redes `NS-2` [16] que é utilizado para comprovar a eficácia da proposta de engenharia de tráfego. Finalizando o capítulo, os experimentos realizados e os resultados obtidos são apresentados na Seção 6.3.

## 6.1 Reescalonamento e migração de tarefas

As únicas diferenças entre as ações de realizar o primeiro escalonamento das tarefas e os reescalonamentos são a mudança do estado de alguns hosts e/ou enlaces e o fato de algumas tarefas já terem finalizado suas execuções, o que faz com que as mesmas não precisem ter um mapeamento encontrado. Dessa forma, é possível reescalonar as tarefas

com o mesmo escalonador utilizado no Passo 1 da engenharia de tráfego para grades, bastando que o escalonador receba o escalonamento anterior, o estado atual dos hosts e o instante de tempo atual, a fim de saber quais tarefas já terminaram sua execução e quais são passíveis de migração.

A utilização de um algoritmo de escalonamento como base para um algoritmo de reescalonamento e migração de tarefas também é proposta em [39]. A diferença para a proposta apresentada nesta dissertação, é o fato do escalonador em [39] ser voltado para aplicações descritas por grafos direcionados que podem conter ciclos.

O Algoritmo 8 apresenta a proposta de realizar os Passos 5, 6 e 7 da engenharia de tráfego para grades com o mesmo escalonador do Passo 1.

**Algoritmo 8** Reescalonamento e migração de tarefas

**Entrada:** Escalonamento anterior; $\mathcal{J}$: DAG atual com as tarefas; $\mathcal{H}$: grafo com o estado atual dos hosts e dos enlaces; Instante de tempo; $E$: Escalonador.

1: **para** cada tarefa $i$ de $\mathcal{J}$ em execução **faça**
2: Mude o peso de $i$ para a quantidade de instruções já executadas.
3: Crie uma nova tarefa $i'$ com peso igual à quantidade de instruções de $i$ que faltam ser executadas.
4: Mova todos os arcos de saída de $i$ para $i'$.
5: Crie um novo arco $ii'$ com peso igual à quantidade de bytes que precisam ser transferidos para que a tarefa $i$ continue sua execução caso seja migrada.
6: Atribua à variável *host* o identificador do host onde $i$ foi escalonada no escalonamento anterior.
7: Crie uma nova restrição para $E$ que obrigue a tarefa $i$ a ser escalonada no host *host*.
8: **fim para**
9: **para** cada tarefa $k$ finalizada ou que tenha começado a receber dados de dependência **faça**
10: Atribua à variável *host* o identificador do host onde $k$ foi escalonada no escalonamento anterior.
11: Crie uma nova restrição para $E$ que obrigue a tarefa $k$ a ser escalonada no host *host*.
12: **fim para**
13: Execute $E$ com as novas restrições e com o novo DAG.
14: **para** cada tarefa $i$ de $\mathcal{J}$ em execução **faça**
15: **se** host onde $i'$ for escalonada $\neq$ host onde $i$ foi escalonada no escalonamento anterior **então**
16: Migre a tarefa $i$ para o novo host.
17: **fim se**
18: **fim para**

O funcionamento do algoritmo baseia-se na modificação do DAG original da aplicação. Para cada tarefa do DAG original que esteja em execução, é criada uma nova tarefa que herda os arcos de saída da tarefa original. Essa nova tarefa possui um único arco de entrada que a interliga à tarefa original. O peso desse arco deve representar a quantidade de dados necessários para que a tarefa original possa continuar sua execução, caso venha a ser migrada. O peso da nova tarefa deve representar a quantidade de instruções que a tarefa original precisará executar para concluir a sua execução, caso venha a ser migrada. Uma vez realizadas as mudanças no DAG original, executa-se o escalonador. Nessa execução, as tarefas que já finalizaram suas execuções, ou que já começaram a receber dados de suas dependências, devem ser mantidas fixas nos hosts anteriormente alocados. No escalonamento encontrado, a migração é realizada caso a nova tarefa seja mapeada para um host diferente do host onde a tarefa original havia sido mapeada.

É importante observar que o Algoritmo 8 considera as seguintes premissas para a migração das tarefas:

- Todas as tarefas podem interromper a sua execução e recomeçá-la do ponto onde foi interrompida a qualquer instante;
- Não há a utilização de servidores para manter o estado de execução das tarefas, ou seja, todos os dados que permitam a continuação da execução da tarefa são transferidos diretamente do host onde a tarefa estava mapeada para o host onde a tarefa estiver sendo enviada;
- As aplicações são compostas por tarefas que tenham o tamanho do código irrelevante frente aos demais dados de dependência e, portanto, a quantidade de bits referente ao código, não é considerada;
- Os requisitos de portabilidade, de hardware e de software das tarefas são atendidos por qualquer host da grade e, portanto, qualquer host é considerado um potencial destino para as tarefas migradas.

A aplicação do algoritmo fica mais fácil de ser entendida através da sua utilização em um simples exemplo. Na Figura 6.5, está representado o DAG de uma aplicação que é executada em uma grade. Considere que, no primeiro escalonamento, as tarefas são mapeadas para os hosts cujos identificadores estão declarados entre parênteses.

Com as constantes medições, nota-se uma mudança significativa no estado do host *h1* em um instante de tempo quando somente a tarefa *T5* está sendo executada (Supondo que todas as tarefas *T0* a *T4*, *T6* e *T7* já tenham terminado sua execução e que a tarefa *T8* esteja aguardando a chegada dos dados da tarefa *T5*). Com a aplicação do algoritmo nesse instante, um novo DAG é construído com a tarefa *T5* dividida em duas: uma que

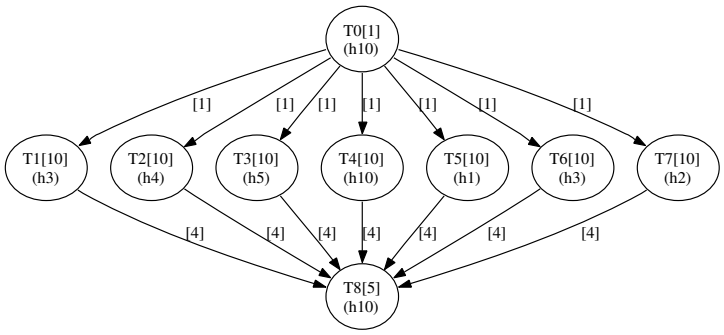



Figura 6.5: Exemplo de DAG para ilustrar a aplicação do Algoritmo 8

representa as instruções que já foram executadas e outra que representa as instruções que ainda precisam ser executadas, como pode ser visto na Figura 6.6. Como pode-se notar, uma nova tarefa (*T5'*) é criada a fim de representar a possível migração da tarefa *T5* que executava no host *h1*. A tarefa *T5'* tem um único arco de entrada ligando-a à tarefa original (*T5*) e possui todos os arcos de saída da tarefa original. A quantidade de dados do arco de entrada e a quantidade de instruções da nova tarefa devem ser conhecidas previamente. Vamos considerar que, nesse caso, no instante em que o reescalonamento é realizado, a tarefa *T5* finalizou 4 instruções, que para a migração ser realizada serão precisos transferir dados equivalentes a um peso de 1 e que a tarefa *T5'* executará 6 instruções para finalizar a execução da tarefa *T5* que iniciou sua execução no host *h1*.

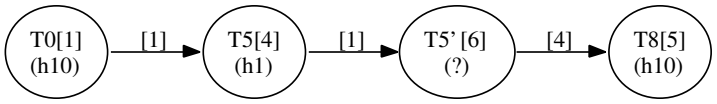



Figura 6.6: Modificação na tarefa *T5* do DAG da Figura 6.5

Com o novo DAG construído, é executado o escalonador, que receberá como entrada, além do novo DAG, o mapeamento que deve ser mantido fixo para as tarefas que já finalizaram sua execução. É importante notar que o fato de manter algumas tarefas fixas faz com que o escalonador execute mais rápido do que na busca pelo primeiro escalonamento, quando o mapeamento para todas as tarefas tem que ser encontrado. Uma vez tendo concluída a execução do algoritmo, deve-se comparar se o host encontrado para a tarefa *T5'* é diferente do host onde a tarefa *T5* estava sendo executada, *h1*. Se não for diferente, a tarefa continua sua execução sem ser migrada. Caso contrário, a migração deve ocorrer.

## 6.2 GridSim-NS

O simulador `GridSim-NS`, desenvolvido na Universidade de Trento, é uma ferramenta para simulação de grades que mapeia as aplicações nos nós de uma rede de comunicação. Ele foi desenvolvido com o objetivo de analisar o impacto causado pelas transferências de dados das grades nos recursos de comunicação disponíveis.

A fim de minimizar o trabalho de desenvolvimento, optou-se por implementar o simulador sobre algum simulador de redes já existente. Decidiu-se desenvolver o `GridSim-NS` como uma extensão para o `NS-2`, um simulador de redes moderno, largamente utilizado e de código aberto. A extensão consiste em um conjunto de objetos que é adicionado a um nó simples do `NS-2`, a fim de permitir que o mesmo atue como membro de uma grade. O maior esforço de desenvolvimento no projeto do `GridSim-NS` diz respeito à modificação da seção referente à geração do tráfego no `NS-2`, já que essa modificação deve refletir as características principais dos nós de uma grade.

A Figura 6.7 apresenta a estrutura geral do processo de simulação realizado pelo `GridSim-NS`. Os blocos em cinza representam os blocos que precisaram ser criados ou modificados no `NS-2`. Os blocos delimitados pela linha tracejada representam o núcleo do `GridSim-NS`. Eles são responsáveis por mapear a aplicação na topologia de rede, e por gerar o arquivo do cenário que será efetivamente simulado. Detalhes sobre cada um dos blocos do processo de simulação são apresentados a seguir.

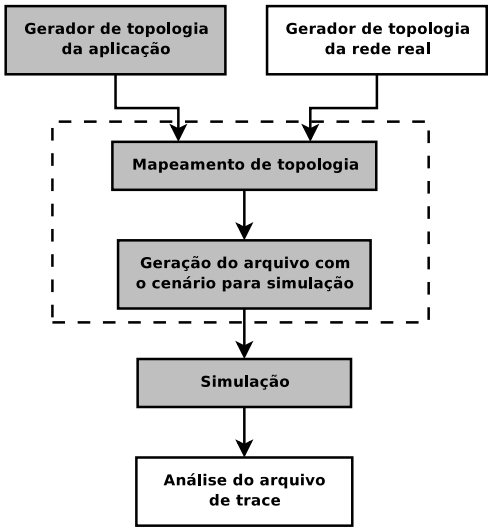



Figura 6.7: Processo de simulação de grades no `GridSim-NS`

A entrada para o simulador é composta de duas topologias: a topologia lógica, construída a partir do DAG da aplicação a ser simulada, e a topologia da rede de comunicação, sobre a qual a grade está implementada. O `GridSim-NS` mapeia o DAG na topologia da

rede, ou seja, cada tarefa da aplicação é atribuída para um dos vários hosts da rede. Nesse passo, pode ser utilizado algum método definido pelo usuário para mapear as tarefas ou alguns dos algoritmos de escalonamentos já existentes no simulador. O `GridSim-NS` permite, por padrão, a utilização de três algoritmos básicos para fazer o mapeamento entre as duas topologias: mapeamento aleatório, no qual cada tarefa é mapeada em um host da topologia de forma aleatória, mapeamento ordenado, no qual as tarefas são mapeadas nos hosts levando em consideração o grau de conectividade do host e a distância para os seus vizinhos, e mapeamento pela distância mínima, que funciona da mesma forma que o mapeamento ordenado com a diferença de que ele não permite que múltiplas tarefas sejam mapeadas para um mesmo host da rede.

Uma vez realizado o mapeamento por algum dos métodos padrões do `GridSim-NS`, ou por algum método implementado pelo usuário, é gerado o cenário para a simulação. No penúltimo passo, a simulação propriamente dita, o `GridSim-NS` não influencia no funcionamento do `NS-2`. Nenhuma mudança no formato de saída do `NS-2` é realizada, o que permite que o último passo, a análise do arquivo de *trace*, seja realizado de forma manual ou por alguma das diversas ferramentas para análise estatística existentes.

Na próxima seção são apresentados os resultados de experimentos realizados para comprovar a eficácia de engenharia de tráfego para grades através do simulador `GridSim-NS`. Os prós e contras dos escalonadores propostos no capítulo anterior, que representam um papel importante na engenharia de tráfego, também são detalhados através dos resultados dos experimentos.

## 6.3 Experimentos realizados

Nesta seção, são apresentados dois grupos de experimentos realizados para avaliar a eficácia da utilização dos escalonadores propostos para migrar tarefas, e a eficácia da engenharia de tráfego para grades. Um grupo de experimentos (referenciado no restante da dissertação como "segundo grupo de experimentos") analisa o uso dos escalonadores para reescalonar e migrar as tarefas. A métrica utilizada para comparar os escalonadores neste grupo é o ganho alcançado no tempo de execução da aplicação escalonada com a realização de migrações. Da mesma forma que nos experimentos realizados para avaliar a eficácia dos escalonadores, no capítulo anterior, dois tipos de DAGs e várias topologias de grade foram utilizados. O outro grupo de experimentos (referenciado no restante da dissertação como "terceiro grupo de experimentos") avalia todo o procedimento de engenharia de tráfego para grades. São avaliados os ganhos alcançados sob modificações no estado de uma grade simulada no `NS-2` durante a execução de um DAG específico. O impacto do intervalo de monitoramento no funcionamento da engenharia de tráfego também é analisado.

Em todos os experimentos, o escalonamento e a verificação de necessidade de migração foram realizados pelos mesmos escalonadores apresentados no Capítulo 5, seguindo todas as particularidades de implementação citadas naquele capítulo. O Algoritmo 8, voltado para a verificação de migração foi implementado em `C`. Os experimentos que avaliam a engenharia de tráfego para grades foram realizados no `NS-2` versão 2.26 compilado com a inclusão do `GridSim-NS` versão 1.1. Todos os experimentos detalhados nesta seção foram executados no mesmo ambiente dos experimentos do primeiro grupo.

Assim como nos experimentos realizados com os escalonadores de tarefas, os modelos de aplicações e de topologias de grades simulados são baseados em casos reais, a fim de minimizar a artificialidade dos experimentos. Os mesmos detalhes a respeito dos pesos das tarefas, hosts, enlaces e arcos daqueles experimentos continuam válidos.

O segundo grupo de experimentos utilizou topologias de grades que foram utilizadas nos experimentos realizados no capítulo anterior, ou seja, geradas pelo método Doar-Leslie. No terceiro grupo de experimentos, as topologias simuladas para as grades são baseadas na topologia que ilustra a aplicação da engenharia de tráfego (Figura 6.2). Esta topologia, contendo um host no centro interligado a hosts periféricos que possuem enlaces entre si, remete à topologia formada pelos centros de pesquisa principais que compõem o LCG.

O segundo grupo de experimentos realizados analisa o desempenho dos escalonadores apresentados na função de reescalonar e migrar as tarefas sob mudanças no estado da grade. Os escalonadores são comparados entre si pelo ganho alcançado no comprimento do escalonamento quando migrações de tarefas são efetuadas. São realizadas quatro mudanças no estado da grade: i) melhora na taxa de processamento dos hosts ociosos; ii) melhora na capacidade de transmissão dos enlaces dos hosts ociosos; iii) piora na taxa de processamento dos hosts inicialmente alocados; iv) piora na capacidade de transmissão dos enlaces entre os hosts inicialmente alocados. Por hosts ociosos, classifica-se os hosts que não executam tarefas dos DAGs, e por hosts alocados, classifica-se os hosts que executam tarefas do DAG. A melhora no estado da grade é realizada através da divisão dos pesos no grafo da rede por 4. Já a piora no estado da grade é realizada através da multiplicação dos pesos no grafo da rede por 4. Essas modificações no estado da grade são realizadas no instante de tempo igual à metade do melhor comprimento do escalonamento encontrado sem a realização de migrações. O Algoritmo 8 é aplicado sobre o DAG original nesse instante de tempo e o DAG gerado é passado como entrada para os escalonadores. Para cada tipo de DAG, os mesmos escalonadores dos experimentos realizados no capítulo anterior foram utilizados: TC-R1, TC-R2, TD-R1, TD-R2, PLITD, SU e SC para o DAG do primeiro tipo; e TD-R1, SU e SC para o DAG do segundo tipo.

O terceiro grupo de experimentos analisa todo o processo de engenharia de tráfego para grades através da simulação da execução de uma aplicação no `NS-2`, compilado com

a inclusão do `GridSim-NS`. Os experimentos avaliam o ganho alcançado com a utilização da engenharia de tráfego sob diversas mudanças no estado da grade. A influência do intervalo de monitoramento no ganho alcançado com a engenharia de tráfego também é avaliada. Em todos os experimentos, a única métrica utilizada na análise de desempenho é o tempo de execução da aplicação simulada.

A Figura 6.8 ilustra o DAG da aplicação utilizada no terceiro grupo de experimentos. Ela é semelhante à aplicação do primeiro tipo que foi utilizada nos experimentos anteriores. As diferenças entre as duas são os pesos das tarefas, e das dependências, e a quantidade de tarefas no nível 1 do DAG. A topologia formada pelos recursos da grade é a mesma que foi utilizada para ilustrar a aplicação da engenharia de tráfego (Figura 6.2). As bandas disponíveis entre o host *SRC0* e os hosts *SRC1* a *SRC10* inicialmente valem 100Mbps para cada par de hosts. A banda disponível nos enlaces que interligam os hosts *SRC1* a *SRC10* entre si, inicialmente vale 33,33Mbps para cada par de hosts. O atraso considerado para todos os enlaces é de 0,2ms em um sentido. Com relação à taxa de processamento disponível, o host *SRC0* possui 1600MIPS e os hosts *SRC1* a *SRC10* possuem 8000MIPS cada.

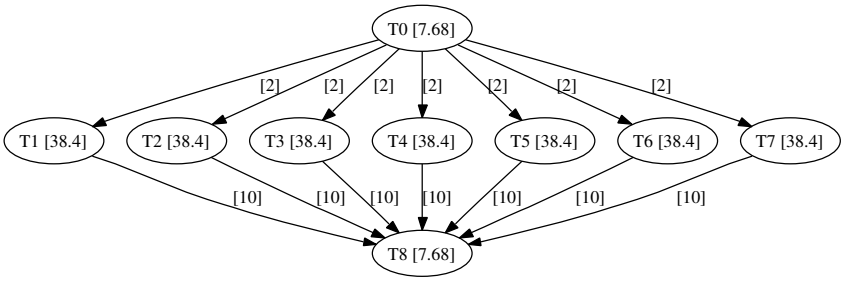



Figura 6.8: DAG da aplicação simulada no terceiro grupo de experimentos

A partir dos resultados do primeiro e do segundo grupo de experimentos, na maioria dos experimentos do terceiro grupo, é utilizado o TD-R1 para escalonar as tarefas e o PLITD para reescalonar e verificar a necessidade de migração. As exceções ocorrem nos experimentos que analisam o impacto do intervalo de monitoramento. Nesses experimentos, também é avaliado o impacto da qualidade dos reescalonamentos e das propostas de migração encontrados pelos escalonadores PLMTC e PLITD.

As subseções a seguir apresentam os resultados alcançados para cada um dos experimentos realizados. Primeiro, são relatados os resultados alcançados com os experimentos do segundo grupo. Da Subseção 6.3.1 à Subseção 6.3.4 são apresentados os resultados alcançados nos experimentos com o primeiro tipo de DAG. Da Subseção 6.3.5 à Subseção 6.3.8 são apresentados os resultados alcançados nos experimentos com o segundo tipo de DAG.

Os experimentos do terceiro grupo são relatados da Subseção 6.3.9 à Subseção 6.3.12: a Subseção 6.3.9 apresenta a utilização da engenharia de tráfego na situação ideal, quando não houve mudanças no estado da grade. O tempo de execução da aplicação nessa situação é útil para comparações nos outros experimentos. A Subseção 6.3.10 apresenta experimentos que avaliam o ganho ao se utilizar a engenharia de tráfego na situação em que houve diminuição da banda disponível de enlaces responsáveis pela transferência de dados entre tarefas da aplicação. A Subseção 6.3.11 apresenta experimentos que avaliam o ganho ao se utilizar a engenharia de tráfego na situação em que novos hosts são incluídos na grade durante a execução da aplicação. Os novos hosts possuem enlaces com mais banda disponível do que os enlaces dos hosts que já faziam parte da grade. A Subseção 6.3.12 apresenta experimentos que avaliam a importância do intervalo de monitoramento para o bom funcionamento da engenharia de tráfego.

As conclusões a respeito dos resultados dos experimentos são apresentadas na Subseção 6.3.13.

### 6.3.1 DAG do primeiro tipo com melhoria no estado dos hosts ociosos

A Tabela 6.1 exibe os resultados alcançados no reescalonamento e migração das tarefas do DAG do primeiro tipo sob melhoria no estado dos hosts ociosos. Os resultados apresentados mostram o ganho do *makespan* alcançado por cada escalonador com relação ao *makespan* sem a realização da migração. O ganho é calculado por $(1 - \frac{m_a}{m_{sm}}) \times 100\%$, onde $m_a$ é o *makespan* alcançado e $m_{sm}$ é o *makespan* sem migração. Os campos da tabela com o conteúdo "–" representam casos em que os escalonadores não sugeriram migrações (Para todas as outras tabelas com resultados neste grupo de experimentos, o ganho também será expresso dessa forma).

Pelos resultados, é possível notar que para cada topologia da grade, pelo menos três escalonadores sugeriram migrações que melhoraram o comprimento do escalonamento da aplicação. Os casos em que não houve sugestão de migração, de modo geral, diminuem com o crescimento na quantidade de hosts da grade, o que era esperado já que, com mais hosts na grade, há o aumento na opção de hosts para os quais as tarefas em execução podem migrar. Dessa forma, pode-se concluir que há ganhos alcançados com o monitoramento do estado dos hosts não alocados a fim de detectar melhorias e realizar migrações das tarefas em execução.

Os escalonadores baseados em sorteio realizaram migrações para tarefas em todas as topologias da grade. Em cinco das sete topologias geradas, as migrações realizadas pelo SC fizeram a aplicação executar mais rápido do que as migrações realizadas pelo SU, comportamento semelhante ao que foi encontrado nos experimentos do primeiro grupo,

| $N$ | Ganho alcançado com a migração | | | | | | |
|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC |
| 10 | – | – | 11,99% | – | 8,21% | 12,28% | 12,28% |
| 40 | 11,93% | – | – | – | – | 18,30% | 17,79% |
| 70 | – | – | 18,70% | 7,73% | 16,41% | 15,72% | 19,69% |
| 100 | – | 17,52% | 28,25% | 11,74% | 10,97% | 21,94% | 22,31% |
| 130 | 19,95% | 25,14% | 25,98% | 15,65% | 19,72% | 14,67% | 21,80% |
| 160 | 15,93% | 17,40% | 19,13% | 10,07% | 20,93% | 16,65% | 17,55% |
| 190 | 16,13% | 23,86% | 25,04% | 10,58% | 23,44% | 15,46% | 20,04% |

Tabela 6.1: Ganho alcançado com a migração para o DAG do primeiro tipo sob melhoria no estado dos hosts ociosos

em que a qualidade do SC foi, em sua maioria, melhor que a qualidade do SU.

De modo geral, a qualidade do TC-R1 e do TD-R2 não foram boas enquanto que o TC-R2 e o TD-R1 não apresentaram um padrão perceptível nos ganhos alcançados com a migração. Os ganhos apresentados pelo PLITD cresceram com o crescimento da quantidade de hosts da grade mas só foram os melhores para um único valor de $N$. O escalonador que forneceu as melhores sugestões de migração foi o TD-R1, em três das sete topologias da grade. Os escalonadores SC e SU propuseram as melhores migrações em duas topologias enquanto que o PLITD propôs em uma topologia.

### 6.3.2 DAG do primeiro tipo com melhoria no estado dos enlaces livres

Os resultados dos experimentos com melhoria no estado dos enlaces livres da grade durante a execução do DAG do primeiro tipo são apresentados na Tabela 6.2.

Ao contrário dos experimentos exibidos na subseção anterior, foram poucos os casos em que os escalonadores sugeriram migrações. Nos casos em que migrações foram sugeridas, o ganho médio ficou bem abaixo do ganho médio alcançado com a migrações para hosts ociosos que tiveram sua taxa de processamento melhorada (3,59% contra 17,41%). Isso ocorre porque a probabilidade de haver ganho com a migração devido à mudanças no estado dos enlaces é menor do que quando ocorrem mudanças no estado dos hosts, já que o estado da rede tem que ser considerado tanto para as futuras transferências de dados de dependência entre as tarefas quanto para as transferências de dados das tarefas passíveis de migração.

Com relação à qualidade das migrações sugeridas pelos escalonadores, não há um padrão perceptível pelos resultados alcançados. De um modo geral, os escalonadores

| $N$ | Ganho alcançado com a migração | | | | | | |
|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC |
| 10 | – | – | – | – | – | – | – |
| 40 | 5,78% | – | – | – | – | – | – |
| 70 | 3,31% | – | 2,59% | – | – | – | 1,55% |
| 100 | – | – | 3,55% | – | 4,49% | – | 5,00% |
| 130 | – | – | 6,39% | – | 2,19% | 3,10% | 5,29% |
| 160 | 3,58% | 1,44% | 1,98% | – | – | – | – |
| 190 | – | – | – | – | – | – | – |

Tabela 6.2: Ganho alcançado com a migração para o DAG do primeiro tipo sob melhoria no estado dos enlaces livres

TC-R1, TD-R1 e SC, nessa ordem, foram os que conseguiram os maiores ganhos com a migração.

Os resultados destes experimentos são importantes pois mostram que a melhora na capacidade dos enlaces nem sempre constitui um bom motivo para realizar as migrações das tarefas. Mesmo melhoras da ordem de 4 vezes na banda passante podem trazer perdas caso as migrações sejam realizadas sem uma análise completa de todo impacto causado na grade.

### 6.3.3 DAG do primeiro tipo com piora no estado dos hosts alocados

Os experimentos realizados para reescalonar e migrar tarefas do DAG do primeiro tipo sob piora no estado dos hosts alocados têm seus resultados apresentados na Tabela 6.3.

Assim como nos experimentos com melhora no estado dos hosts ociosos, os resultados destes experimentos mostram que boa parte dos escalonadores apresentou sugestões de migração que trouxeram ganhos na execução da aplicação. Para todas as topologias da grade, pelo menos dois escalonadores sugeriram migrações que melhoraram o comprimento do escalonamento da aplicação.

Em média, os ganhos alcançados com a migração foram melhores do que os alcançados quando houve melhora no estado dos hosts ociosos (41,46% contra 17,41%). Essa diferença é justificada pelo fato de que o comprimento do escalonamento sem a realização de migração ($m_{sm}$) e com a piora no estado de hosts já alocados é maior do que o comprimento do escalonamento sem a realização de migração e com a melhora de hosts não alocados (uma aplicação não piora o comprimento do seu escalonamento quando ocorre melhora no estado dos hosts ociosos, o que não é verdade quando ocorre piora no estado dos hosts

| $N$ | Ganho alcançado com a migração | | | | | | |
|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC |
| 10 | – | – | 42,46% | 42,46% | 42,46% | 42,46% | 42,46% |
| 40 | – | – | – | – | – | 47,76% | 46,29% |
| 70 | – | – | 47,54% | 38,21% | 41,94% | 43,36% | 43,84% |
| 100 | 49,06% | 47,78% | 50,16% | – | 48,11% | 48,11% | 49,68% |
| 130 | – | – | 25,64% | 24,63% | 22,81% | 20,60% | 19,83% |
| 160 | 37,13% | 43,24% | 49,22% | 36,14% | 48,79% | 43,09% | 48,13% |
| 190 | – | 46,01% | – | – | – | 43,07% | 45,72% |

Tabela 6.3: Ganho alcançado com a migração para o DAG do primeiro tipo sob piora no estado dos hosts alocados

alocados). Essa mudança no valor de $m_{sm}$ faz com que os ganhos sejam maiores nos casos de migração com piora no estado dos hosts alocados do que com melhora no estado dos hosts ociosos, já que o aumento de $m_{sm}$ faz o cálculo do ganho ($(1-\frac{m_a}{m_{sm}})\times 100\%$) fornecer um valor maior.

Os ganhos dos escalonadores mantiveram-se próximos do valor médio com o crescimento de $N$. Os escalonadores baseados em sorteio forneceram sugestões de migração e conseguiram ganhos para todos os valores de $N$. Os escalonadores PLITD e TD-R1 sugeriram migrações para cinco das sete topologias. Os demais escalonadores sugeriram migrações para, no mínimo duas, e no máximo quatro, topologias da grade.

Com relação ao ganho alcançado com as migrações, os melhores escalonadores foram TD-R1, SC e SU, nessa ordem.

### 6.3.4 DAG do primeiro tipo com piora no estado dos enlaces entre hosts alocados

Os últimos experimentos realizados para reescalonar e migrar tarefas do DAG do primeiro tipo têm seus resultados exibidos na Tabela 6.4. Os valores na tabela apresentam os ganhos alcançados com a realização da migração de tarefas sob piora no estado dos enlaces entre hosts alocados.

Pelos resultados, nota-se que foram sugeridas migrações em mais casos do que nos experimentos em que foi realizada melhora no estado dos enlaces entre hosts ociosos. Além disso, o ganho médio com as migrações também foi maior do que naqueles experimentos (11,33% contra 3,59%). Isso é justificado pelo fato do comprimento do escalonamento sem migração ser maior, como foi explicado na subseção anterior.

Para cada valor de $N$, pelo menos dois escalonadores sugeriram migrações que trouxe-

| $N$ | Ganho alcançado com a migração | | | | | | |
|---|---|---|---|---|---|---|---|
| | TC-R1 | TC-R2 | TD-R1 | TD-R2 | PLITD | SU | SC |
| 10 | – | 9,10% | 11,18% | – | 7,47% | 11,18% | 11,18% |
| 40 | 7,89% | – | 12,20% | – | 11,73% | 11,50% | 7,89% |
| 70 | 15,31% | 12,99% | 18,04% | 10,05% | 10,83% | 9,82% | 15,09% |
| 100 | – | 18,43% | 22,85% | – | 21,62% | 11,46% | 15,23% |
| 130 | – | – | 2,23% | – | – | – | – |
| 160 | 7,30% | – | 8,65% | – | 7,30% | – | – |
| 190 | 1,92% | 6,82% | – | – | – | – | – |

Tabela 6.4: Ganho alcançado com a migração para o DAG do primeiro tipo sob piora no estado dos enlaces entre hosts alocados

ram ganhos no comprimento do escalonamento. As sugestões de migração do escalonador TD-R1 trouxeram ganho para seis das sete topologias de grade utilizadas nos experimentos. O escalonador PLITD conseguiu ganhos para cinco e os escalonadores SU, SC, TC-R1 e TC-R2 para quatro. O escalonador TD-R2 sugeriu migrações para apenas uma topologia dos experimentos.

Com relação à qualidade das migrações sugeridas pelos escalonadores, não há um padrão perceptível pelos resultados alcançados. De um modo geral, os escalonadores TD-R1, PLITD e SC, nessa ordem, foram os que conseguiram os maiores ganhos com a migração.

### 6.3.5 DAG do segundo tipo com melhoria no estado dos hosts ociosos

A Tabela 6.5 exibe os resultados alcançados para os primeiros experimentos de reescalonamento e migração com o DAG do segundo tipo. Os resultados apresentados mostram o ganho alcançado com a migração das tarefas frente à melhoria no estado dos hosts ociosos.

Nota-se que para todas as topologias da grade, os três escalonadores utilizados sugeriram propostas de migração para as tarefas em execução. Entre os escalonadores baseados em sorteio, o SC conseguiu ganhos maiores do que os alcançados pelo SU para a maioria das topologias da grade. Assim como nos experimentos com melhoria no estado dos hosts ociosos para o DAG do primeiro tipo, o escalonador TD-R1 fez as melhores sugestões de migração para três das sete topologias, entretanto, de um modo geral, os melhores ganhos foram alcançados pelo escalonador SC, seguido pelo TD-R1.

O ganho médio alcançado com as migrações nestes experimentos foi de 21,08%, um pouco maior que o ganho médio alcançado nos experimentos realizados com o DAG do

| $N$ | Ganho alcançado com a migração | | |
|---|---|---|---|
| | TD-R1 | SU | SC |
| 10 | 22,16% | 10,60% | 25,30% |
| 15 | 20,48% | 19,95% | 18,71% |
| 20 | 12,41% | 10,22% | 11,23% |
| 25 | 27,98% | 26,36% | 26,05% |
| 30 | 17,25% | 17,81% | 19,63% |
| 35 | 19,26% | 22,02% | 20,57% |
| 40 | 30,68% | 31,60% | 32,48% |

Tabela 6.5: Ganho alcançado com a migração para o DAG do segundo tipo sob melhoria no estado dos hosts ociosos

primeiro tipo submetido a mudanças semelhantes na grade, que foi de 17,41%.

### 6.3.6 DAG do segundo tipo com melhoria no estado dos enlaces livres

Os resultados dos experimentos com melhoria no estado dos enlaces dos hosts ociosos da grade durante a execução do DAG do segundo tipo são apresentados na Tabela 6.6.

| $N$ | Ganho alcançado com a migração | | |
|---|---|---|---|
| | TD-R1 | SU | SC |
| 10 | 5,90% | 2,66% | – |
| 15 | 6,08% | 5,15% | 4,47% |
| 20 | 4,69% | – | – |
| 25 | 9,91% | 8,81% | 15,68% |
| 30 | 4,57% | – | 3,46% |
| 35 | 5,87% | – | 1,00% |
| 40 | – | – | 9,62% |

Tabela 6.6: Ganho alcançado com a migração para o DAG do segundo tipo sob melhoria no estado dos enlaces livres

Assim como nos resultados alcançados com o DAG do primeiro tipo sob mudanças semelhantes na grade, nem todos os escalonadores utilizados neste experimento sugeriram migrações, o que é justificado pelo fato dos enlaces afetarem tanto as transferências futuras de dados quanto as transferências necessárias para as migrações, como explicado

na Subseção 6.3.2. Da mesma forma que avaliado nos experimentos realizados com o DAG do primeiro tipo, o ganho médio alcançado com a migração nestes experimentos foram menores que nos experimentos em que ocorreram melhorias no estado dos hosts ociosos (6,27% contra 21,08%).

Os melhores ganhos foram alcançados pelos escalonadores TD-R1, SC e SU, nessa ordem. A relação entre os ganhos do TD-R1 e do SC é semelhante àquela nos experimentos realizados com o DAG do primeiro tipo sob melhoria no estado dos enlaces livres.

### 6.3.7 DAG do segundo tipo com piora no estado dos hosts alocados

Esta seção apresenta os resultados alcançados com a utilização dos escalonadores para reescalonar e migrar tarefas do DAG do segundo tipo sob piora no estado dos hosts alocados. A Tabela 6.7 exibe os ganhos alcançados com a migração para cada um dos escalonadores em cada uma das sete topologias de grade utilizadas.

| $N$ | Ganho alcançado com a migração | | |
|---|---|---|---|
| | TD-R1 | SU | SC |
| 10 | 36,43% | 28,17% | 38,61% |
| 15 | 41,42% | 40,37% | 41,07% |
| 20 | 32,83% | 27,15% | 32,22% |
| 25 | 53,61% | 46,49% | 49,11% |
| 30 | 46,66% | 39,38% | 43,00% |
| 35 | 40,97% | 35,93% | 39,84% |
| 40 | 54,81% | 52,03% | 52,55% |

Tabela 6.7: Ganho alcançado com a migração para o DAG do segundo tipo sob piora no estado dos hosts alocados

De forma semelhante ao analisado nos resultados para os experimentos com o DAG do primeiro tipo, a média dos ganhos alcançados com a migração foram maiores do que os alcançados quando houve melhora no estado dos hosts ociosos (41,55% contra 21,08%) pela diferença no comprimento do escalonamento sem migração, conforme explicado na Subseção 6.3.3. O ganho médio alcançado nestes experimentos ficaram bem próximos dos ganhos alcançados com modificações semelhantes na grade para o DAG do primeiro tipo (41,55% contra 41,46%).

Assim como nos experimentos que avaliaram os escalonadores sob melhora no estado dos hosts ociosos, todos os escalonadores sugeriram migrações que trouxeram ganhos para a execução da aplicação e os melhores escalonadores foram TD-R1, SC e SU, nessa ordem.

### 6.3.8 DAG do segundo tipo com piora no estado dos enlaces entre hosts alocados

Os resultados alcançados nos últimos experimentos realizados para reescalonar e migrar tarefas do DAG do segundo tipo são apresentados na Tabela 6.8.

| $N$ | Ganho alcançado com a migração | | |
|---|---|---|---|
| | TD-R1 | SU | SC |
| 10 | 12,98% | 4,53% | 12,42% |
| 15 | 1,30% | 2,79% | 2,53% |
| 20 | – | – | 4,35% |
| 25 | 24,09% | 22,43% | 24,33% |
| 30 | 12,55% | 10,45% | 7,64% |
| 35 | – | – | – |
| 40 | – | 5,19% | 4,44% |

Tabela 6.8: Ganho alcançado com a migração para o DAG do segundo tipo sob piora no estado dos enlaces entre hosts alocados

Assim como nos experimentos semelhantes que foram realizados para o DAG do primeiro tipo, o ganho médio alcançado nestes experimentos foi melhor do que o alcançado nos experimentos em que houve a melhora no estado dos enlaces ociosos (10,13% contra 6,27%), o que é justificado pelo fato do comprimento do escalonamento sem migração ser maior.

Com relação aos ganhos alcançados para diferentes topologias da grade, o SC sugeriu migrações que levaram a ganhos em seis das sete topologias de grade. Os ganhos com o SU foram alcançados em cinco das sete topologias e, os ganhos com o TD-R1 foram alcançados em quatro das sete topologias.

De modo geral, os maiores ganhos com as migrações foram alcançados pelos escalonadores TD-R1, SC e SU, nessa ordem.

### 6.3.9 Ambiente ideal

O primeiro experimento do terceiro grupo tem o objetivo de verificar o tempo de execução da aplicação na situação em que todos os hosts e enlaces são dedicados à grade. Os passos da engenharia de tráfego foram realizadas e os resultados para cada um são detalhados a seguir.

**Passo 1**: O escalonamento encontrado para a aplicação, pelo escalonador TD-R1, mapeia as tarefas da seguinte forma: $0 \rightarrow$ *SRC0*, $1 \rightarrow$ *SRC2*, $2 \rightarrow$ *SRC5*, $3 \rightarrow$ *SRC8*,

4 → *SRC4*, 5 → *SRC1*, 6 → *SRC9*, 7 → *SRC10*, 8 → *SRC0*. O início de execução da tarefa 0 se dá no instante 0min. As Tarefas de 1 a 7 iniciam suas execuções no instante 82,66min. A tarefa 8 inicia sua execução no instante 175,96min. O instante de finalização da execução devolvido pelo escalonador é 255,96min.

**Passo 2**: As tarefas são atribuídas aos diversos hosts e a simulação é iniciada no `NS-2`. As transferências das dependências entre as tarefas são realizadas no `NS-2` via FTP.

**Passos 3 e 4**: Um procedimento no `NS-2` verifica o estado da grade de 40 em 40 minutos. Como não há nenhuma variação no estado dos mesmos, já que eles são dedicados à grade, o escalonamento inicial mantém-se fixo durante toda a simulação.

Nesse experimento, o tempo de finalização da aplicação no `NS-2` foi de 257min, bem próximo do valor encontrado pelo escalonador no Passo 1.

### 6.3.10 Diminuição na banda disponível

Os próximos experimentos realizados têm, por objetivo, verificar o ganho obtido com a engenharia de tráfego na situação em que os enlaces não são dedicados exclusivamente à grade. Desse modo, os enlaces podem sofrer mudanças nas suas bandas disponíveis. As mudanças simuladas nos experimentos foram causadas devido à criação de dois tráfegos de interferência, um entre os hosts *IR2* e *IS2*, e outro entre os hosts *IR5* e *IS5*. Esses tráfegos afetam a capacidade de transmissão dos hosts *SRC2* e *SRC5* ao host *SRC0*. Ambos os tráfegos surgem no instante de tempo 90min, são UDP e transferem dados a uma taxa constante de 90Mbps. Eles prolongam-se enquanto a aplicação estiver em execução. Como o escalonamento inicial das tarefas foi o mesmo encontrado no experimento descrito na subseção anterior (0 → *SRC0*, 1 → *SRC2*, 2 → *SRC5*, 3 → *SRC8*, 4 → *SRC4*, 5 → *SRC1*, 6 → *SRC9*, 7 → *SRC10*, 8 → *SRC0*), a variação da capacidade de transmissão dos hosts *SRC2* e *SRC5* ao host *SRC0*, afeta as transferências dos dados de dependência das Tarefas 1 e 2 para a Tarefa 8.

No primeiro experimento descrito nesta subseção, a aplicação foi executada e a engenharia de tráfego não foi utilizada. Nesse experimento, as mudanças devido aos tráfegos de interferência foram ignoradas e as tarefas 1 e 2 permaneceram, respectivamente, nos hosts *SRC2* e *SRC5* durante toda a execução da aplicação. Com esse mapeamento fixo, as transferências de dados das tarefas 1 e 2 para a tarefa 8, competiram com os tráfegos de interferência. A Figura 6.9 exibe o gráfico com a vazão da transferência dos dados das tarefas 1 e 2 para a tarefa 8. Por causa dos tráfegos de interferência (UDP, CBR, a 90Mbps), as vazões das tarefas 1 e 2 para a tarefa 8, sofreram variações e mantiveram-se próximas à taxa de 10Mbps, apesar dos enlaces terem capacidade de transmitir dados a uma taxa de 100Mbps.

Com a variação nas capacidades de transmissão, e sem a utilização da engenharia de

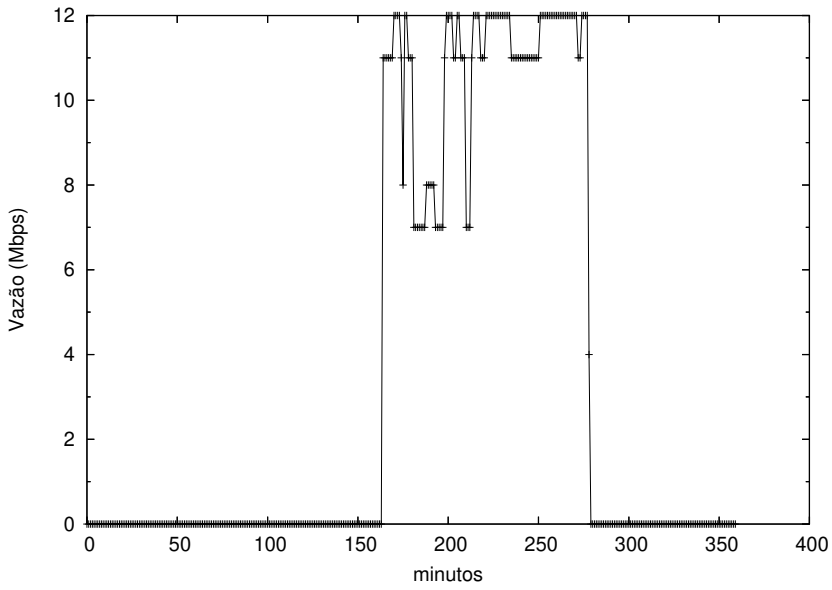



Figura 6.9: Vazão das tarefas 1 e 2 para a tarefa 8 com tráfego de interferência

tráfego, o tempo de execução da aplicação na grade foi de 358min. Comparando esse valor com o alcançado no experimento que considerou o ambiente ideal (257min), nota-se que as mudanças no estado dos enlaces levaram a um tempo de execução $\cong 39,3\%$ maior.

No experimento seguinte, foram gerados os mesmos tráfegos de interferência do experimento anterior, com a diferença de que a engenharia de tráfego foi utilizada. Os resultados da execução de cada passo da engenharia de tráfego para esse experimento são detalhados a seguir.

**Passo 1**: O escalonamento encontrado para a aplicação, pelo escalonador TD-R1, é o mesmo do experimento anterior, já que os hosts e os enlaces inicialmente encontram-se no mesmo estado daquele experimento.

**Passo 2**: As tarefas são atribuídas aos diversos hosts e a simulação é iniciada no `NS-2`.

**Passos 3 e 4**: Um procedimento no `NS-2` verifica o estado dos enlaces e dos hosts. Esse procedimento é configurado para ser realizado de 40 em 40 minutos. Na terceira vez que ele é realizado (no instante igual a 120min) há a detecção de mudança na banda disponível dos hosts *SRC2* e *SRC5* ao host *SRC0*: ela cai de 100Mbps para 10,62Mbps.

**Passo 5**: No instante 120min, um novo DAG é construído para a aplicação. Considera-se que a quantidade de bytes necessários para a migração das tarefas em execução é proporcional ao progresso da execução das tarefas. Quanto mais perto de finalizar a execução, uma maior quantidade de bytes precisarão ser enviados na migração. A quantidade de bytes para migração da tarefa $i$ é calculada por $p_i \times (\sum_{j \in suc(i)} peso(ij) + \sum_{j \in pred(i)} peso(ji))$, onde $p_i$ é a porcentagem de execução da tarefa $i$, $pred(i)$ é o conjunto com os predecessores

da tarefas $i$ e $suc(i)$ é o conjunto com os sucessores da tarefa $i$. As novas tarefas divididas (que representam as tarefas migradas) têm quantidade de instruções igual à quantidade de instruções da tarefa original sendo migrada subtraída da quantidade de instruções que já foram executadas até o instante da migração. Não é considerado *overhead* na quantidade de instruções para as tarefas divididas pois, caso isso fosse realizado, uma tarefa que se mantivesse fixa no seu host seria penalizada desnecessariamente. Dessa forma, o DAG modificado é aquele apresentado na Figura 6.10. Como pode ser notado, as Tarefas 1 a 7 do DAG anterior (Figura 6.8) são divididas, já que no instante 120min elas ainda estão em execução e são passíveis de migração.

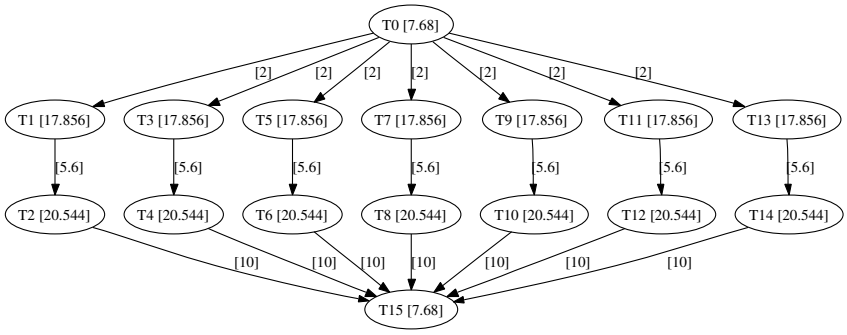



Figura 6.10: DAG para migração no instante 120min

Esse novo DAG e o mapeamento anterior são passados como entrada para o escalonador PLITD. Pelo novo escalonamento, as Tarefas 1 e 2 devem migrar, respectivamente, para os hosts *SRC3* e *SRC6*, a fim de evitar a concorrência do tráfego da aplicação com os tráfegos de interferência.

**Passos 6, 7 e 8:** No `NS-2` as transferências de dados devido às migrações são realizadas no instante 120min. A Figura 6.11 exibe a interrupção do uso da CPU pelas Tarefas 1 e 2 entre o instante 120min e o instante 143min. O intervalo de tempo em que o uso da CPU é interrompido coincide com o intervalo em que a migração da tarefa 1 é realizada do host *SRC1* para o host *SRC3*, e em que a migração da tarefa 2 é realizada do host *SRC6* para o host *SRC5* como pode ser notado no gráfico da Figura 6.12 que exibe o RTT medido entre os hosts *SRC1* e *SRC3* e entre os hosts *SRC5* e *SRC6*. Na Figura 6.12, nota-se o aumento no RTT no intervalo entre 120min e 143min, causado pela transferência dos dados das migrações.

Nesse experimento, o tempo de finalização da aplicação foi de 281min. Comparando esse tempo com o que foi encontrado no experimento anterior (358min), quando a engenharia de tráfego não foi utilizada, nota-se que houve um ganho de $\cong 21,51\%$ no tempo de execução da aplicação. Se a comparação for feita com o experimento que considerou o ambiente ideal, nota-se que, a aplicação executou em um tempo somente $\cong 9,34\%$ pior.

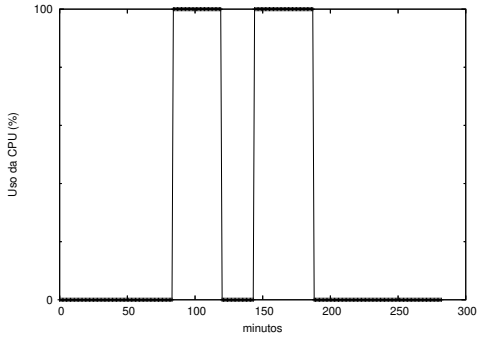



Figura 6.11: Uso da CPU pelas tarefas 1 e 2

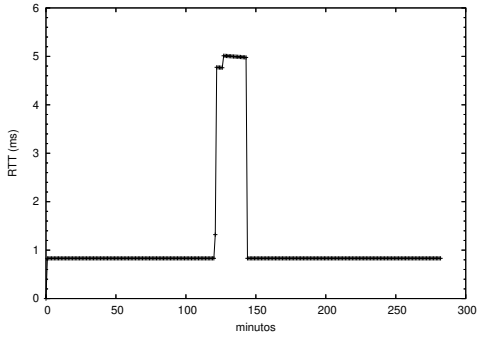



Figura 6.12: RTT entre os hosts *SRC1* e *SRC2* e entre *SRC5* e *SRC6*

### 6.3.11 Inclusão de hosts com mais banda disponível

Nesta subseção, são relatados os experimentos realizados com o objetivo de avaliar o desempenho da engenharia de tráfego quando são incluídos novos hosts na grade. Nos experimentos, não houve modificação na capacidade de transmissão dos enlaces e nem na taxa de processamento dos hosts iniciais durante toda a execução da aplicação. A única modificação na grade ocorre no instante 90min, quando foram incluídos 10 novos hosts. Cada um deles tem um enlace ligado a um dos hosts *SRC1* a *SRC10* e outro enlace ligado ao host *SRC0*. A Figura 6.13 exibe a inclusão do host *SRC16* ligado a *SRC6* (a topologia final completa não é exibida a fim de facilitar a visualização). A taxa de processamento disponível nos novos hosts é de 8000MIPS e a capacidade dos seus enlaces é de 1Gbps.

Nesse experimento, a engenharia de tráfego foi utilizada (o monitoramento foi realizado de 40 em 40 minutos) e, no instante 120min, as tarefas 1 a 7 migraram dos hosts em que estavam executando para os que foram incluídos na grade. A aplicação finalizou a sua execução no instante 247min, um tempo inferior ao alcançado sem a inclusão dos novos hosts, que foi 257min, o que comprova que a engenharia de tráfego otimizou a execução da aplicação quando surgiram mais hosts disponíveis na grade.

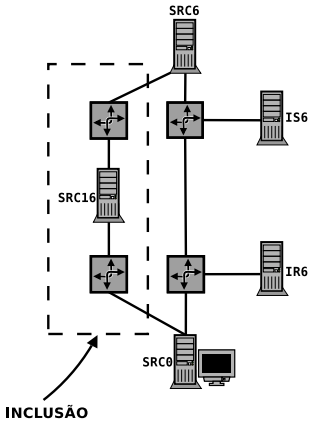
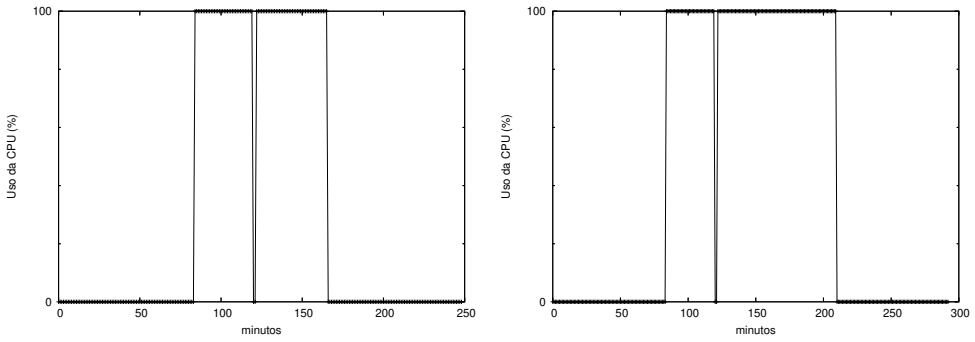



Figura 6.13: Inclusão do novo host ligado a *SRC6*

É importante observar que na verificação dos ganhos alcançados com a migração, os escalonadores avaliam tanto o estado dos enlaces quanto dos hosts. Para ilustrar esse ponto, o seguinte experimento foi realizado: com as mesmas inclusões descritas na Figura 6.13, mas com as taxas de processamento disponível dos novos hosts iguais a 4000MIPS, ou seja, uma taxa que vale a metade da considerada anteriormente, os escalonadores não propuseram migrações. Mesmo assim, as migrações foram realizadas e o tempo de execução da aplicação foi de 291min, pior do que o alcançado caso a migração não tivesse sido realizada (257min), o que comprova que os escalonadores forneceram a melhor opção ao não propor as migrações das tarefas. A Figura 6.14 apresenta dois gráficos que servem para comparar o tempo de execução da tarefa 1 quando ela migrou para o novo host com taxa de 8000MIPS e quando ela não seguiu a recomendação do escalonador e migrou para o novo host com taxa de 4000MIPS.




(a) Uso da CPU pela tarefa 1 no host com 8000MIPS

(b) Uso da CPU pela tarefa 1 no host com 4000MIPS

Figura 6.14: Uso da CPU pela tarefa 1 ao migrar para hosts com diferentes taxas de processamento

Quando a migração é feita para o host com taxa de 8000MIPS, a execução da Tarefa 1 finaliza no instante 165min (gráfico da Figura 6.14(a)). Quando a mesma tarefa é migrada para o host com taxa de 4000MIPS, a sua execução finaliza no instante 209min (gráfico da Figura6.14(b)), o que faz com que a tarefa de saída do DAG atrase o início de sua execução e aumente o tempo de execução da aplicação.

### 6.3.12 Mudanças no instante de monitoramento

Os últimos experimentos realizados ilustram a importância que os instantes de monitoramento têm no ganho obtido com a utilização da engenharia de tráfego. Isso se deve ao fato da consideração de que a quantidade de bytes a serem transmitidos para uma tarefa migrar é diretamente proporcional ao tempo de execução da aplicação. Assim, quanto mais tempo leva-se para notar a mudança no estado de algum enlace, menores são as chances de se conseguir algum ganho com a migração das tarefas. Além da importância dos instantes de monitoramento, os experimentos também avaliam os escalonamentos propostos por dois escalonadores diferentes.

Foram realizados 12 experimentos com a utilização da engenharia de tráfego para grades. Para metade dos experimentos, o Passo 5 da engenharia de tráfego foi executado com o escalonador PLMTC. Na outra metade, foi utilizado o PLITD. Para cada um dos escalonadores utilizados, o monitoramento do estado da grade foi realizado uma única vez nos instantes 100min, 110min, 120min, 130min, 140min e 150min, totalizando assim os 12 experimentos. Para todos os experimentos, houve a criação de tráfegos de interferência no instante 90min entre os mesmos hosts dos experimentos relatados na Subseção 6.3.10 (um entre os hosts *IR2* e *IS2*, e outro entre os hosts *IR5* e *IS5*), o que tornou as tarefas 1 e 2 passíveis de migração. A taxa dos tráfegos de interferência foi de 60Mbps.

A fim de ter um parâmetro para comparação, a aplicação foi executada inicialmente sem a utilização da engenharia de tráfego. Nesse caso, o primeiro escalonamento foi devolvido pelo escalonador TD-R1, nenhuma tarefa migrou e o tempo de execução da aplicação foi de 276min.

A Tabela 6.9 apresenta os tempos de execução da aplicação para cada um dos experimentos realizados com a utilização da engenharia de tráfego. Para cada instante do monitoramento, é apresentado o tempo de execução da aplicação de acordo com as sugestões de reescalonamento dadas pelos escalonadores PLMTC e PLITD. Para cada tempo de execução, é apresentada entre parênteses a informação sobre a realização ou não de migração das tarefas 1 e 2 de acordo com o escalonamento devolvido pelos escalonadores.

Pelos resultados, nota-se que a migração (das tarefas 1 e 2) somente é sugerida pelo PLMTC quando o instante de medição é menor que 120min. A partir desse valor, o escalonador não sugeriu migração e a aplicação terminou sua execução nos hosts inicialmente

| Instante | PLMTC | PLITD |
|---|---|---|
| 100 | 269 (com migração) | 269 (com migração) |
| 110 | 275 (com migração) | 275 (com migração) |
| 120 | 276 (sem migração) | 281 (com migração) |
| 130 | 276 (sem migração) | 287 (com migração) |
| 140 | 276 (sem migração) | 276 (sem migração) |
| 150 | 276 (sem migração) | 276 (sem migração) |

Tabela 6.9: Tempos de execução para mudanças no instante de monitoramento (minutos)

mapeados. Já o PLITD só não sugeriu migração quando o instante de monitoramento foi maior que 130min, o que foi uma decisão ruim, já que a aplicação levou mais tempo para executar ao seguir as sugestões de migração nos instantes 120min e 130min. Essa decisão ruim deve-se às aproximações do modelo implementado pelo PLITD, que considera a linha do tempo inteira.

A fim de comparar a vantagem alcançada ao não serem realizadas as migrações de tarefas nos instantes de monitoramento 140 e 150, a aplicação foi executada e a migração das tarefas foi realizada, apesar dos escalonadores não as terem sugerido. Nesses casos, o tempo de execução da aplicação foi, respectivamente, 293min e 299min, o que mostra que as decisões dos escalonadores de não migrar as tarefas foi correta, já que o tempo de execução da aplicação manteve-se em 276min nos dois casos.

### 6.3.13 Resumo conclusivo dos resultados encontrados

Os resultados alcançados com os experimentos que avaliaram o uso dos escalonadores para reescalonar e migrar as tarefas de DAGs sob mudanças no estado da grade apresentaram resultados que comprovam a eficácia dos algoritmos de escalonamento e do algoritmo que modifica o DAG original para que haja a verificação de necessidade de migração. Independente do tipo de DAG utilizado, os escalonadores sugeriram migrações que trouxeram ganhos para a execução da aplicação com relação ao comprimento do escalonamento. Os escalonadores mais presentes na lista dos melhores para este grupo de experimentos foram os escalonadores TD-R1 e SC, nessa ordem, sendo que os resultados do escalonador PLITD ficaram próximos dos alcançados por esses dois escalonadores.

Pelos resultados alcançados com os experimentos que avaliaram o desempenho da engenharia de tráfego para grades, nota-se a importância de ser utilizado um escalonador que dê resultados os mais precisos possíveis, sendo que o PLMTC é uma boa opção de escalonador a ser usado no reescalonamento e migração caso hajam poucas tarefas

passíveis de migração. Nota-se, também, que o instante em que as medições são realizadas influencia nas chances de que migrações sejam sugeridas e de que a aplicação execute mais rápida, já que a partir de um certo valor, o tempo gasto com a migração da tarefa deixa a aplicação mais lenta do que se não fosse feita migração.

# Capítulo 7

# Conclusões

A presente dissertação apresentou uma proposta para otimizar o tempo de execução de aplicações em grades através da utilização dos mesmos princípios aplicados pela engenharia de tráfego na Internet. A principal contribuição desta proposta, e o que a diferencia das demais encontradas na literatura, é o fato dela considerar o estado dos enlaces da rede e o custo causado pelas transferências de dados tanto no escalonamento quanto na migração das tarefas, sendo que o custo para este último não é fixo e depende do progresso de execução das tarefas.

A aplicação da proposta em um cenário de simulação, executado no simulador para grades `GridSim-NS`, um simulador desenvolvido como extensão para o `NS-2`, mostrou que a mesma consegue ganhos ao migrar as tarefas em situações em que há diminuição da capacidade dos enlaces ou inclusão de recursos com melhor capacidade de transmissão do que os recursos atuais da grade. Os resultados das simulações também mostram a importância do monitoramento freqüente dos hosts e da rede, bem como a necessidade de se adotar escalonadores de tarefas que forneçam escalonamentos próximos do ótimo em um curto intervalo de tempo.

Para a tarefa de escalonamento, foram propostos oito novos escalonadores. Uma hipótese freqüentemente assumida na literatura é que os hosts na grade são completamente conectados, o que restringe a validade dessas propostas. Os escalonadores, apresentados nesta dissertação, não consideram essa hipótese e são experimentados para uma variedade de configurações de grades.

Seis escalonadores são caracterizados por serem baseados na modelagem do problema de escalonamento como problemas de programação linear inteira e mista. Os outros dois escalonadores são caracterizados por serem baseados em algoritmos iterativos que realizam seqüências de sorteio em busca do melhor escalonamento possível. A diferença entre as várias propostas diz respeito à qualidade do escalonamento devolvido e ao tempo de processamento necessário. De um modo geral, as propostas que exigem mais tempo de

processamento são as que devolvem os melhores escalonamentos. Essas várias propostas de escalonadores podem ser executadas em sequência dentro de um *deadline* específico. Aquele escalonador que fornecer o melhor escalonamento dentro do *deadline* será o utilizado pela aplicação.

Pelos resultados numéricos, comprovou-se a diferença no tempo de execução entre as propostas bem como na qualidade dos escalonamentos encontrados. O escalonador baseado em PL com linha do tempo discreta, sem relaxamento das variáveis inteiras e com relaxamento das variáveis inteiras utilizando o algoritmo de arredondamento aleatório, foram os que apresentaram melhores resultados, sendo que o segundo apresentou escalonamentos com uma qualidade bem próxima do primeiro e com um tempo de execução bem menor. Os escalonadores baseados em sorteio executaram bem mais rápido do que os demais, no entanto, de modo geral, não obtiveram bons escalonamentos. Notou-se que a distribuição "consciente" das probabilidades fez diferença, dado que o escalonador só não apresentou resultado melhor do que aquele que distribui as probabilidades uniformemente em 3 dos 42 diferentes experimentos realizados (cada experimento é composto de uma topologia de grade e um DAG específico).

Os escalonadores também são utilizados pela engenharia de tráfego nos passos referentes ao reescalonamento e à migração das tarefas. Introduziu-se um algoritmo que modifica o DAG da aplicação sendo executada, de modo a considerar as instruções que já foram executadas, o mapeamento anterior e a quantidade de dados necessários para que uma tarefa interrompa a sua execução e a continue em um outro host, do ponto onde foi interrompida. Esse DAG modificado é então passado como entrada para algum dos escalonadores apresentados que avaliam se há ou não ganho em se realizar alguma migração. De modo geral, escalonadores usados para migração tendem a executar em um tempo menor que os escalonadores usados para o escalonamento inicial, já que uma boa parte das tarefas é mantida fixa e não precisa ter seu mapeamento encontrado.

Pelos experimentos realizados para comprovar a eficácia do algoritmo de modificação do DAG e a eficácia de utilizar os mesmos algoritmos voltados para escalonar as tarefas, notou-se ganhos em praticamente todos os experimentos realizados. Somente em 3 dos 56 experimentos realizados nenhum dos escalonadores sugeriram migrações de tarefas que trouxessem ganhos no tempo de execução da aplicação.

Como trabalhos futuros para o passo de escalonamento da engenharia de tráfego, pretende-se analisar o desempenho dos escalonadores com DAGs gerados aleatoriamente, como feito em [50], e com a verificação de outros requisitos das tarefas, como espaço para armazenar dados na memória principal e na memória secundária dos hosts e restrições relacionadas ao instante em que a migração pode ocorrer, ou seja, tarefas que tenham *checkpoints* definidos. Também serão estudadas modificações que devem ser implementadas nos escalonadores com o objetivo de permitir que os mesmos tratem a incerteza

das informações nos DAGs, tanto na quantidade de instruções, quanto na quantidade de bytes das dependências.

Para os demais passos de engenharia de tráfego, pretende-se realizar a análise de características relacionadas com o tráfego gerado por aplicações para grades, a fim de se propor modelos que melhor representem as transferências de dados nesses ambientes. Pretende-se ainda, realizar um estudo com o objetivo de definir dinamicamente o melhor intervalo a ser utilizado no monitoramento do estado da grade de modo a detectar o maior número de mudanças relevantes e sem gerar consumo elevado de processamento nos hosts e de banda nos enlaces da grade.

# Referências Bibliográficas


[1] Gabrielle Allen, David Angulo, Ian Foster, Gerd Lanfermann, Chuang Liu, Thomas Radke, Ed Seidel, and John Shalf. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *International Journal of High Performance Computing Applications*, 15(4):345–358, November 2001.

[2] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. RFC 3272: Overview and Principles of Internet Traffic Engineering, 2002.

[3] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. RFC 2702: Requirements for Traffic Engineering Over MPLS, 1999.

[4] Daniel M. Batista, Nelson L. S. da Fonseca, Fabrizio Granelli, and Dzmitry Kliazovich. Self-Adjusting Grid Networks. In *Proceedings of the IEEE International Conference on Communications – ICC 2007 (Aceito para publicação)*, Jun 2007.

[5] Daniel M. Batista, Nelson L. S. da Fonseca, and Flávio K. Miyazawa. Escalonadores de Tarefas em Grades. In *Anais do XXVI Congresso da Sociedade Brasileira de Computação – V Wperformance*, pages 73–92, Jul 2006.

[6] Daniel M. Batista, Nelson L. S. da Fonseca, and Flávio K. Miyazawa. A Set of Schedulers for Grid Networks. In *SAC'07: Proceedings of the 2007 ACM Symposium on Applied Computing*, Mar 2007.

[7] Philip A. Bernstein. Middleware: A Model for Distributed System Services. *Commun. ACM*, 39(2):86–98, 1996.

[8] Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. In *IEEE International Symposium on Cluster Computing and Grids (CCGRID'05)*, volume 2, pages 759–767, May 2005.

[9] Mario Cannataro, Carlo Mastroianni, Domenico Talia, and Paolo Trunfio. Evaluating and Enhancing the Use of the GridFTP Protocol for Efficient Data Transfer on the Grid. *Lecture Notes in Computer Science*, 2840:619 – 628, 2003.

[10] Henri Casanova. Distributed Computing Research Issues in Grid Computing. *SIGACT News*, 33(3):50–70, 2002.

[11] Henri Casanova. Modeling Large-Scale Platforms for the Analysis and the Simulation of Scheduling Strategies. In *Proceedings of the 6th Workshop on Advances in Parallel and Distributed Computational Models*, pages 170–177, Santa Fe, NM, April 2004.

[12] Daniel Paranhos da Silva, Walfredo Cirne, and Francisco Vilar Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. *Lecture Notes in Computer Science*, 2790:169–180, 2003.

[13] dash optimization. The Xpress-Optimizer. Disponível em `http://www.dashoptimization.com/home/products/products_optimizer.html`. Último acesso em 21/05/2006.

[14] Matthew Doar and Ian M. Leslie. How Bad is Naive Multicast Routing? In *Proceedings of IEEE INFOCOM'93*, pages 82–89, 1993.

[15] ESA Science & Technology: ESAGRID, 2004. `http://sci.esa.int/science-e/www/area/index.cfm?fareaid=85`. Último acesso em 21/05/2006.

[16] Kevin Fall and Kannan Varadhan. The ns Manual. Technical report, UC Berkeley, The VINT Project, Aug 2002. Disponível em `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf`. Último acesso em 21/05/2006.

[17] Wu-chun Feng and Peerapol Tinnakornsrisuphap. The Failure of TCP in High-Performance Computational Grids. In *Supercomputing*, pages 37–47, Nov 2000.

[18] I. Foster. What is the Grid? A Three Point Checklist. *GRIDToday*, 1(6), July 2002. Disponível em `http://www.gridtoday.com/02/0722/100136.html`. Último acesso em 21/05/2006.

[19] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal Supercomputer Applications*, 15(3):200–222, 2001.

[20] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1st edition, 1998.

[21] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edition, 2004.

[22] Noriyuki Fujimoto and Kenichi Hagihara. Near-Optimal Dynamic Task Scheduling of Precedence Constrained Coarse-Grained Tasks onto a Computational Grid. In *International Symposium on Parallel and Distributed Computing*, pages 80–87, 2003.

[23] Grid Common Services, 2002. `http://www.nas.nasa.gov/Research/Tasks/gridtasks.html`. Último acesso em 21/05/2006.

[24] GriPhyN - Grid Physics Network, 2005. `http://www.griphyn.org/`. Último acesso em 21/05/2006.

[25] Xiaoshan He, Xianhe Sun, and Gregor von Laszewski. QoS Guided Min-Min Heuristic for Grid Task Scheduling. *J. Comput. Sci. Technol.*, 18(4):442–451, 2003.

[26] Eduardo Huedo, Ruben S. Montero, and Ignacio M. Llrorent. An Experimental Framework for Executing Applications in Dynamic Grid Environments. Technical Report 2002-43, NASA Langley Research Center, 2002.

[27] Integrade: OO Grid Middleware, 2006. `http://integrade.incubadora.fapesp.br/portal/`. Último acesso em 04/07/2006.

[28] M. Iverson, F. Ozguner, and G. Follen. Parallelizing Existing Applications in a Distributed Heterogeneous Environment. In *Heterogeneous Computing Workshop*, pages 93–100, 1995.

[29] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[30] LCG - LHC Computing Grid Project, 2006. `http://lcg.web.cern.ch/LCG/`. Último acesso em 21/05/2006.

[31] Bruce B. Lowekamp. Combining Active and Passive Network Measurements to Build Scalable Monitoring Systems on the Grid. *SIGMETRICS Performation Evaluation Review*, 30(4):19–26, 2003.

[32] Dab Ma and Wei Zhang. A Static Task Scheduling Algorithm in Grid Computing. *Lecture Notes in Computer Science*, 3033:153–156, 2004.

[33] Teng Ma and Junzhou Luo. Optimizing Large File Transfer on Data Grid. *Lecture Notes in Computer Science*, 3795:455 – 460, 2005.

[34] Shawn McKee. LHC Scale Physics in 2008: Grids, Networks and Petabytes, 2005. Disponível em `http://ciara.fiu.edu/pasi/PASI_Shawn_May_18_2005.ppt`. Último acesso em 21/05/2006.

[35] Montage, 2005. `http://montage.ipac.caltech.edu/`. Último acesso em 21/05/2006.

[36] Harvey Newman. UltraLight Annual Report for 2004-2005. Technical report, California Institute of Technology, 2005. Disponível em `http://ultralight.caltech.edu/portal/documents/2005/02annualreport/UltraLightAnnualReport_V9022805.pdf`. Último acesso em 21/05/2006.

[37] OurGrid, 2006. `http://dsc.ufcg.edu.br/ourgrid/`. Último acesso em 04/07/2006.

[38] Christos H. Papadimitriou and Kennet Steiglitz. *Combinatorial Optimization – Algorithms and Complexity*, pages 363–366. Dover Publications, 1998.

[39] Radu Prodan and Thomas Fahringer. Dynamic Scheduling of Scientific Workflow Application on the Grid: a Case Study. In *SAC'05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694, New York, NY, USA, 2005. ACM Press.

[40] Luc Renambot, Tom van der Schaaf, Henri E. Bal, Desmond Germans, and Hans J. W. Spoelder. Griz: Experience with Remote Visualization over an Optical Grid. *Future Generation Computer Systems*, 19(6):871–882, 2003.

[41] Alain Roy and Miron Livny. Condor and Preemptive Resume Scheduling. In *Grid Resource Management: State of the Art and Future Trends (1st Edition)*, pages 135–144, 2003.

[42] Jennifer M. Schopf. Ten Actions when Grid Scheduling. In *Grid Resource Management: State of the Art and Future Trends (1st edition)*, pages 15–23, 2003.

[43] John A. Silvester. CalREN: Advanced Network(s) for Education in California, 2005. Disponível em `http://isd.usc.edu/~jsilvest/talks-dir/20051021-cudi-merida.pdf`. Último acesso em 21/05/2006.

[44] SINAPAD – Sistema Nacional de Alto Desempenho, 2006. `http://www.lncc.br/sinapad/`. Último acesso em 21/05/2006.

[45] Oliver Sinnen and Leonel A. Sousa. Communication Contention in Task Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):503–515, June 2005.

[46] D. B. Skillicorn. Motivating Computational Grids. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid(CCGRID'02)*, pages 401–406, May 2002.

[47] Jonathan M. Smith. A Survey of Process Migration Mechanisms. *SIGOPS Operating Systems Review*, 22(3):28–40, 1988.

[48] SPRACE – São Paulo Regional Analysis Center, 2005. `http://hep.ift.unesp.br/SPRACE/index.php`. Último acesso em 21/05/2006.

[49] Xian-He Sun and Ming Wu. GHS: A Performance System of Grid Computing. In *19th IEEE International Parallel and Distributed Processing Symposium*, 2005. Disponível em `http://doi.ieeecomputersociety.org/10.1109/IPDPS.2005.234`. Último acesso em 21/05/2006.

[50] Haluk Topcuouglu, Salim Hariri, and Min you Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002.

[51] UltraLight, 2005. `http://ultralight.caltech.edu/web-site/ultralight/html/index.html`. Último acesso em 21/05/2006.

[52] Sathish S. Vadhiyar and Jack J. Dongarra. A Performance Oriented Migration Framework for the Grid. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid(CCGRID'03)*, pages 130–137, 2003.

[53] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[54] Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: a Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[55] Kun Yang, Xin Guo, Alex Galis, Bo Yang, and Dayou Liu. Towards Efficient Resource on-Demand in Grid Computing. *SIGOPS Oper. Syst. Rev.*, 37(2):37–43, 2003.