Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Programa de Pós-Graduação em Engenharia de Eletricidade
Área de concentração: Ciência da Computação

Gilberto Cunha Filho

OGST (Opportunistic Grid Simulation Tool): uma ferramenta de simulação para avaliação de estratégias de escalonamento de aplicações em grades oportunistas

São Luís
2009

Gilberto Cunha Filho

OGST (Opportunistic Grid Simulation Tool): uma ferramenta de simulação para avaliação de estratégias de escalonamento de aplicações em grades oportunistas

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, como requisito parcial para a obtenção do grau de MESTRE em Engenharia de Eletricidade.

Orientador: Francisco José da Silva e Silva

Doutor em Ciência da Computação – UFMA

São Luís

2009

Cunha Filho, Gilberto

OGST (*Opportunistic Grid Simulation Tool*): uma ferramenta de simulação para avaliação de estratégias de escalonamento de aplicações em grades oportunistas / Gilberto Cunha Filho. – São Luís, 2009.

121 f.

Orientador: Francisco José da Silva e Silva.

Impresso por computador (fotocópia).

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-Graduação em Engenharia de Eletricidade. São Luís, 2009.

1. Grades oportunistas. 2. Simulação - Grades. 3. Grades - Escalonamento. I. Silva, Francisco José da Silva e, orient. II. Título.

CDU 004.413.2

OGST (OPPORTUNISTIC GRID SIMULATION TOOL): UMA FERRAMENTA DE SIMULAÇÃO PARA AVALIAÇÃO DE ESTRATÉGIAS DE ESCALONAMENTO DE APLICAÇÕES EM GRADES OPORTUNISTAS

Gilberto Cunha Filho

Dissertação aprovada em 13 de fevereiro de 2009.

Prof. Francisco José da Silva e Silva, Dr.
(Orientador)

Prof. Raphael Yokoingawa de Camargo, Dr.
(Membro da Banca Examinadora)

Prof. Zair Abdelouahab, Ph. D.
(Membro da Banca Examinadora)

Aos meus pais, irmão e minha família.

Agradecimentos

Em primeiro lugar, a Deus por ter me abençoado com o dom da vida, saúde e sabedoria. Aos meus pais, Gilberto Nonato e Antonia Guimarães, por estarem presentes na minha vida e sempre dispostos a me oferecer o maior e melhor suporte no exercício de minhas atividades. Ao meu irmão abençoado, Tonigil Guimarães, por sempre está proporcionando momentos de distrações.

A Dannielle P. Lima por ter aparecido na minha vida e proporcionado uma constante motivação pelo desejo de aprender e crescer, agradecendo a Deus por todas as maravilhas que ele nos faz.

Ao meu orientador Professor Francisco por seu tempo, dedicação, paciência, sabedoria e experiências de vida que pode me oferecer ao longo do desenvolvimento deste trabalho, proporcionando um aprendizado incrível não apenas na minha vida profissional, mas também na minha vida pessoal.

Aos professores Raphael Yokoingawa de Camargo e Zair Abdelouahab por terem aceito participar desta banca.

Aos membros do Laboratório de Sistemas Distribuídos (LSD), em especial a Bysmarck, Diego, Estevão, Eduardo Viana, Eduardo Devidson, Filipe, Jaciara, Jesseildo, Pablo, Paulo, Rafael, Regilaine e Vandecia, pelos momentos de trabalhos e distrações no dia-a-dia do laboratório.

A Helaine Cristina, Victor Hugo, Geraldo Braz, Euziel, turma 308 e ao pessoal da paróquia do São Cristovão pela amizade e apoio durante o desenvolvimento deste trabalho.

À CAPES pelo financiamento desta pesquisa e ao grupo de pesquisa do InteGrade, cujo projeto foi usado como base para o desenvolvimento deste trabalho.

A Lucinete (“Baixinha”) por proporcionar o café de todos os dias.

"O princípio da sabedoria é: adquire a sabedoria; sim, com tudo o que possuis, adquire o entendimento. Estima-a, e ela te exaltará; se a abraçares, ela te honrará;"

Provérbios (4:7,8)

Resumo

Durante o desenvolvimento de sistemas de *middleware* de grade, pesquisadores freqüentemente empregam técnicas e ferramentas de simulação para validação de novos conceitos e implementações. Ferramentas de simulação têm um papel fundamental no desenvolvimento de sistemas de *middleware* de grade uma vez que: (a) pesquisadores freqüentemente não têm acesso a grandes ambientes de grade para testes, limitando a capacidade para avaliar situações que demandam por uma grande quantidade de recursos; (b) é difícil explorar cenários com recursos e aplicações em larga escala envolvendo diversos usuários de forma repetitiva e controlada, devido à natureza dinâmica de ambientes de grade; (c) aplicações reais da grade geralmente consomem muito tempo, de poucas horas até mesmo a semanas.

Este trabalho descreve o OGST, um simulador de eventos discretos orientado a objetos cujo principal objetivo é auxiliar desenvolvedores de sistemas de *middleware* de grade oportunista na validação de novos conceitos e implementações. A motivação preliminar para o desenvolvimento do OGST foi prover um caminho para avaliar o comportamento de algoritmos de escalonamento comumente usados em ambientes de grade sob diferentes condições do ambiente de execução e a investigação de abordagens de escalonamento adaptativo. Ele foi cuidadosamente projetado para levar em consideração a dinâmica de grades oportunistas, provendo um conjunto de funcionalidades que agilizam o desenvolvimento de simulações que consideram o dinamismo do ambiente de execução. O simulador foi desenvolvido no contexto do projeto InteGrade, mas foi projetado para permitir a simulação de grades oportunistas de uma maneira em geral com o propósito de ser aplicado a outros projetos de pesquisa envolvendo *middleware* de grades.

Palavras-chaves: Grades oportunistas. Simulação de grades. Escalonamento em grades.

Abstract

During the development of Grid middleware systems, researchers often employ simulation tools and techniques for validating new concepts and implementations. Simulation tools play a fundamental role on the development of Grid middleware systems since: (a) researchers often do not have access to huge Grid testbed environments, limiting the capacity for evaluating situations that demand high amount of resources; (b) it is difficult to explore in large scale application and resources scenarios involving several users in a repetitive and controlled way, due to the dynamic nature of Grid environments; (c) real Grid applications usually consume great amount of time, ranging from a few hours to even weeks.

This work describes OGST, an object-oriented discrete event simulator whose main objective is to assist developers of opportunistic Grid middleware on validating new concepts and implementations. The preliminary motivation for OGST development was to provide a way for evaluating the behavior of scheduling algorithms commonly used on Grid environments under different execution environment conditions and the investigation of adaptive scheduling approaches. It was carefully designed to take into consideration the dynamics of opportunistic Grids, providing a set of features that hasten the development of simulations that takes into consideration the dynamism of the execution environment. The simulator was developed in the context of the InteGrade project, but was designed to allow the simulation of generic opportunistic Grids in order to be applied by other Grid middleware research projects.

Key-words: Opportunistic grids. Grid Simulation. Grid scheduling.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Uma grade computacional compreende uma infra-estrutura de *hardware* e *software* que permite a integração e compartilhamento de recursos distribuídos, tais como *software*, periféricos e dados, em redes corporativas e entre estas. Grades computacionais têm se tornado uma alternativa atraente para a execução de aplicações que demandam um grande poder computacional e têm sido usadas para resolver problemas em diversas áreas de atividades industriais, empresariais e científicas, tais como: computação biológica, processamento de imagem para diagnóstico médico, previsão meteorológica, simulações de mercado e prospecção de óleo [23].

O processo de escalonamento de aplicações em um sistema de grade consiste na atribuição das tarefas das aplicações para os recursos disponíveis segundo um objetivo específico, como o de minimizar o tempo de resposta das aplicações e/ou maximizar o uso dos recursos computacionais disponíveis. No entanto, a construção de uma estratégia de escalonamento voltada a ambientes de grades de computadores é uma tarefa desafiadora devido a diversas características presentes nestes ambientes computacionais, tais como [18]:

- Alta heterogeneidade: não somente os nós computacionais e de armazenamento, mas também os enlaces de rede que os conectam são heterogêneos. Os nós computacionais podem diferir com relação a aspectos como a arquitetura computacional, quantidade de processadores, tamanho da memória e velocidade de processamento. Os enlaces de rede podem apresentar diferenças em termos de velocidade de transmissão de dados e protocolos de comunicação. O processo de escalonamento deve, portanto, considerar as diferentes capacidades de processamento de tarefas e de acesso a dados existentes nos recursos.

- Alta escalabilidade: grades de computadores podem compreender milhares de recursos de processamento e armazenamento geograficamente distribuídos em múltiplos domínios administrativos. A alta escalabilidade torna inviável uma abordagem de escalonamento baseada no conhecimento global do estado dos recursos que compõem o ambiente.

- Autonomia de domínios administrativos: a estratégia de escalonamento de aplicações deve levar em consideração diferentes políticas de gerenciamento de recursos e de segurança definidas para cada domínio administrativo que compõe a grade. Aspectos como a priorização na execução de aplicações internas ao domínio ou restrições no uso de recursos são exemplos de regras que eventualmente podem estar presentes.

- Ambiente de execução dinâmico: a disponibilidade de recursos na grade está em constante mudança. Novos recursos com grande poder de processamento podem ser integrados e outros podem tornar-se indisponíveis devido a diversos fatores, como eventuais falhas dos enlaces de rede. Além disso, enlaces de rede podem apresentar alta variação em sua largura de banda devido ao tráfego de dados de aplicações externas aos serviços da grade. Este dinamismo deve ser levado em consideração pela estratégia de escalonamento adotada e é particularmente acentuado em ambientes de grades oportunistas, onde o foco é no aproveitamento de ciclos ociosos de máquinas interconectadas em rede, tanto em períodos nos quais as máquinas não estão sendo utilizadas por seus usuários (como períodos noturnos e de almoço) quanto em ocasiões em que tarefas de baixo consumo de recursos são executadas, como edição de texto e navegação na WWW [27].

Um aspecto fundamental no processo de desenvolvimento de novas estratégias de escalonamento para ambientes de grades de computadores é a sua avaliação. Neste contexto, ferramentas de simulação usualmente desempenham um importante papel, dado as dificuldades em se obter um ambiente adequado de testes, entre as quais destacamos:

- A dificuldade em se ter acesso a um ambiente de grade com recursos em grande escala, visto que freqüentemente pesquisadores apenas têm acesso a uma quantidade restrita de recursos, limitando a capacidade para avaliar situações que demandam por uma grande quantidade de recursos computacionais;

- O custo econômico para adquirir ou pagar pelo uso de uma grande quantidade de recursos pode ser elevado, considerando que analisar novas abordagens e estratégias de escalonamento pode requerer uma quantidade elevada de experimentos com o maior número de recursos possíveis;

- A dificuldade em se explorar cenários com aplicações em grande escala. Uma vez que aplicações reais da grade geralmente consomem muito tempo, de poucas horas até mesmo a semanas, pode ser inviável executar vários experimentos quando existem restrições de tempo;

- Pode ser uma árdua tarefa realizar experimentos que envolvam falhas de recursos, visto que o dinamismo de recursos da grade pode gerar falhas que não foram previstas para um experimento;

- É difícil explorar cenários com recursos e aplicações envolvendo diversos usuários de forma repetitiva e controlada, devido à natureza dinâmica dos ambientes de grade e à dificuldade de coordenar os usuários.

Apesar da importância do uso de simuladores no desenvolvimento de sistemas de middleware para computação em grade, observamos que em algumas situações, sistemas distribuídos não podem ser facilmente modelados por simuladores. Comportamentos e interações complexas de nós de sistemas distribuídos podem não ser simulados, devido à dificuldade de capturar e extrair fatores que influenciam os comportamentos de sistemas distribuídos [43].

1.1 Objetivos

Esta dissertação de mestrado tem como principal objetivo o desenvolvimento de uma ferramenta de simulação que permita avaliar o comportamento de estratégias de escalonamento em diversos cenários comuns aos ambientes de execução de grades oportunistas. Os objetivos específicos deste trabalho são:

- Estudo do estado da arte de algoritmos de escalonamento em grades computacionais;

- Estudo do estado da arte de ferramentas de simulação em grades computacionais;

- Definição dos requisitos de projeto e arquitetura da ferramenta de simulação de uma grade oportunista;

- Implementação e avaliação da ferramenta de simulação de uma grade oportunista;
- Implementação e avaliação de algoritmos de escalonamento em grades oportunistas.

1.2 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta a base teórica necessária para o entendimento do escalonamento de aplicações em grades de computadores, a descrição de importantes algoritmos de escalonamento voltados a ambientes distribuídos e as abordagens de escalonamento utilizadas pelos principais sistemas de *middleware* de grade. O Capítulo 3 descreve a ferramenta de simulação desenvolvida ao longo deste trabalho denominada OGST (*Opportunistic Grid Simulation Tool*). São apresentados os requisitos de seu projeto, sua arquitetura e detalhes de implementação. O Capítulo 4 descreve avaliações realizadas do OGST que utilizam as abordagens de mapeamento de funcionalidades e validação de resultados de simulações. O Capítulo 5 traz uma análise de desempenho das estratégias de escalonamento de aplicações em grades oportunistas. O Capítulo 6 discute relevantes trabalhos relacionados, enquanto o Capítulo 7 apresenta as conclusões obtidas a partir deste trabalho e apresenta os trabalhos futuros que podem ser desenvolvidos a partir deste esforço inicial.

2 Escalonamento em Grades de Computadores

Neste capítulo descreveremos uma visão geral do processo de escalonamento em grades de computadores. Apresentaremos uma arquitetura básica de um escalonador de grade e esquemas de organização de escalonadores. Além disso, descrevemos alguns dos principais algoritmos de escalonamento empregados pelos principais sistemas de *middleware* de grade.

2.1 Visão Geral do Processo de Escalonamento de Aplicações em Grades de Computadores

O escalonamento em grades é um processo de tomada de decisões envolvendo recursos pertencentes a múltiplos domínios administrativos. Este processo inclui a pesquisa de recursos para a execução de aplicações na grade. No entanto, diferentemente dos escalonadores tradicionais de sistemas distribuídos e paralelos (ex: MPP e SMP), os escalonadores de grades não possuem controle sobre os recursos e o conjunto de aplicações no sistema. Assim, torna-se importante a existência de componentes que permitam, entre outras funcionalidades, a descoberta de recursos, o monitoramento e armazenamento de informações sobre recursos e aplicações, a permissão de acesso a diferentes domínios administrativos e, dependendo da estratégia de escalonamento adotada, a estimativa do desempenho dos recursos e do tempo de execução das aplicações nos mesmos.

A Figura 2.1 ilustra os passos básicos do processo de escalonamento em grades de computadores, que foi estendido a partir da arquitetura exposta por Zhu, em [53]. O processo de escalonamento em grades, segundo Schopf et al [46], pode ser dividido em três estágios: filtro e descoberta de recursos, seleção dos recursos e envio da aplicação com preparação do ambiente de execução.

O Escalonador da Grade (EG), no primeiro estágio, cria um filtro para selecionar os recursos a partir das restrições e preferências providas pelos usuários. As restrições definem requisitos mínimos para a seleção de máquinas, como por exemplo,

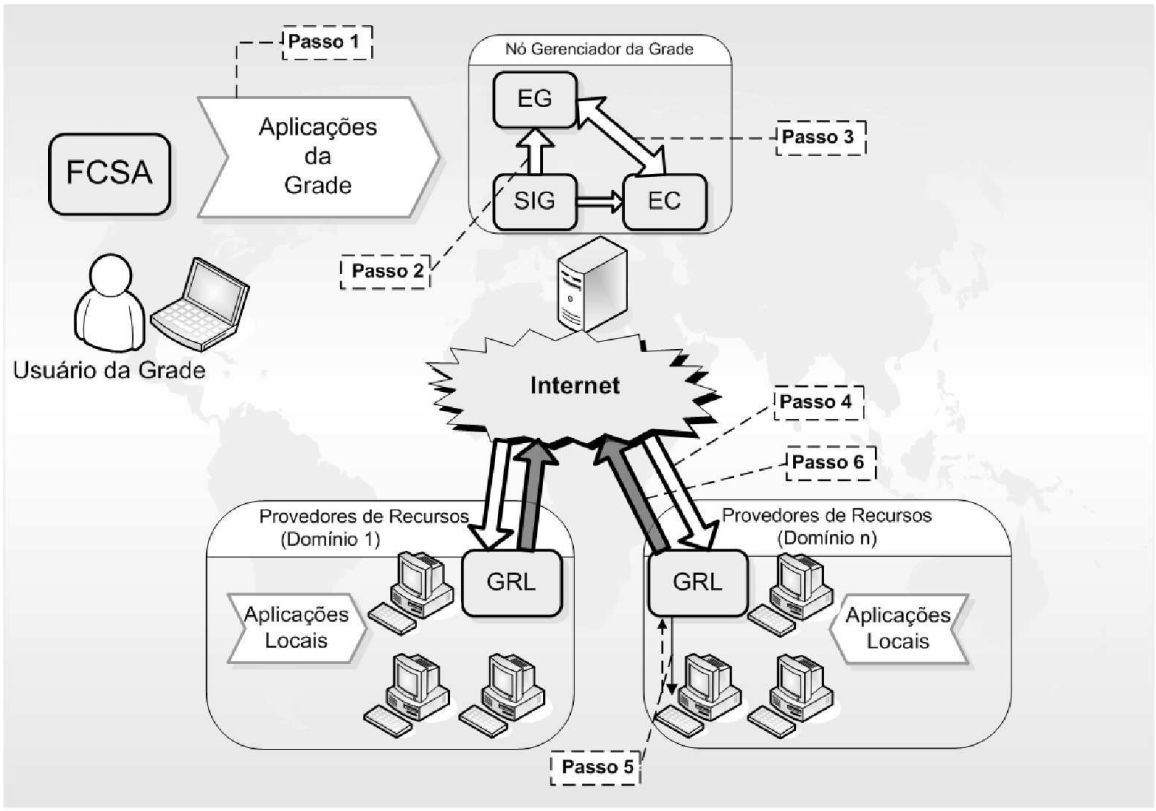

Figura 2.1: Arquitetura básica do escalonador de aplicações em grades de computadores

a utilização de máquinas que tenham pelo menos 4 GB de memória RAM. Já as preferências definem a ordem na escolha dos recursos como, por exemplo, ordenar as máquinas pela maior quantidade de memória disponível.

As informações sobre os recursos disponíveis são muito importantes para um escalonamento, especialmente devido à natureza heterogênea e dinâmica da grade. O papel do serviço de informação da grade (SIG) é o de prover um conjunto de informações dinâmicas e estáticas para os escalonadores da grade. As informações dinâmicas de cada recurso são constantemente monitoradas pelo Gerenciador de Recursos Locais (GRL) e enviadas periodicamente para o SIG. Alguns exemplos de informações dinâmicas são: a capacidade livre de CPU, a quantidade da memória em uso e o atraso para entrega de pacotes nos enlaces da rede. Por outro lado, as informações estáticas obtidas pelo SIG são enviadas pelo GRL, geralmente, no momento de registro de cada recurso na grade. Alguns exemplos de informações estáticas são: a capacidade da CPU e o tamanho da memória de nós da grade e a configuração e capacidade dos enlaces de rede.

O Estimador de Custo (EC) provê uma medida de como um recurso compu-

tacional executa um dado tipo de aplicação. Um *benchmark* analítico poderia ser usado como forma de ordenar os recursos disponíveis de acordo com a sua eficiência para executar um determinado tipo de código computacional.

Os usuários submetem aplicações para a grade, através da Ferramenta de Controle e Submissão de Aplicações (FCSA) (passo 1). Após filtrar os recursos, no segundo estágio do processo de escalonamento, o EG irá gerar o mapeamento das aplicações para os recursos de acordo com o objetivo do sistema como, por exemplo, minimizar o tempo de resposta das aplicações ou maximizar o número de aplicações concluídas por unidade de tempo (*Throughput*) [53, 18]. Este mapeamento irá ocorrer de acordo com informações sobre o estado dos recursos disponíveis e a estimativa do desempenho de recursos para tipos específicos de aplicações fornecidas, respectivamente, pelo Serviço de Informação da Grade (SIG) (passo 2) e um Estimador de Custo (EC) (passo 3).

No terceiro estágio do processo de escalonamento, o GRL é responsável por receber as aplicações enviadas pelo escalonador da grade (passo 4) e preparar o ambiente para sua execução como, por exemplo, a transferência de arquivos contendo dados de entrada para a aplicação. Após a preparação do ambiente, o GRL inicia a execução das aplicações (passo 5). Quando as aplicações terminam sua execução, ele envia os resultados da execução das aplicações para os clientes da grade (passo 6).

2.2 Esquemas de Escalonamento em Grades

A Figura 2.1 ilustra somente um escalonador na grade com um esquema centralizado. No entanto, de acordo com diferentes interesses de desempenho ou escalabilidade, os sistemas de escalonamentos em grade podem ser formados por vários escalonadores e organizados em diferentes esquemas [47], como os seguintes:

- **Centralizado**: neste modelo, o escalonador mantém informações sobre todos os domínios administrativos. Todas as aplicações são submetidas para o escalonador. Baseado na fila de aplicações submetidas e nas informações sobre todos os domínios administrativos, ele toma as decisões de escalonamento.

- **Hierárquico**: neste esquema, cada domínio possui seu próprio escalonador local

e os escalonadores da grade são interligados em uma estrutura hierárquica. Uma solicitação de execução de aplicação que não possa ser atendida com os recursos disponíveis localmente é repassada ao nível acima da hierarquia, cujo escalonador possui uma visão mais abrangente dos recursos da grade.

- **Distribuído**: neste modelo, cada domínio possui seu próprio escalonador e os escalonadores consultam periodicamente uns aos outros para coletar informações atualizadas sobre cargas locais. Uma aplicação submetida a um dado domínio pode ser transferida para execução em outro domínio que possua carga menor.

2.3 Algoritmos de Escalonamento

Um algoritmo de escalonamento determina como as aplicações devem ser escalonadas e como os recursos devem ser utilizados, de acordo com as metas de desempenho da grade e as informações disponibilizadas. No entanto, o mapeamento de um conjunto de tarefas em um conjunto de recursos heterogêneos é um problema NP-completo bem conhecido [32, 20].

Os algoritmos de escalonamento podem ser agrupados em dois modos: modo *batch* e modo *on-line* [39]. O modo de escalonamento *on-line* define que uma tarefa será mapeada para uma máquina assim que ela chega ao escalonador. Já no modo *batch*, as tarefas não são mapeadas assim que elas chegam. Ao invés disso, elas são colocadas dentro de um conjunto de tarefas, chamado de metatarefa e são mapeadas em tempos pré-escalonados, chamados eventos de mapeamento. Algoritmos *batch* comparam os requisitos de recursos das tarefas para tomar uma decisão de mapeamento.

Dong e Akl [18] descrevem algoritmos de escalonamento usualmente utilizados em ambientes distribuídos de grades de computadores, entre os quais destacam-se:

- *Work queue* (WQ): é um algoritmo *on-line* que atribui cada tarefa para a próxima máquina que se torna disponível. Se múltiplas máquinas tornam-se disponíveis simultaneamente uma delas é escolhida aleatoriamente. Este algoritmo mapeia apenas uma tarefa por recurso. Ele não faz uso de qualquer informação sobre a

aplicação e o recurso e é particularmente útil para sistemas onde o objetivo principal é maximizar a utilização dos recursos, ao invés de minimizar o tempo de execução de tarefas individuais. Dependendo da implementação, este algoritmo pode necessitar consultar o estado de todas as m máquinas para encontrar as máquinas disponíveis. Sendo assim, a ordem de complexidade deste algoritmo para realizar um escalonamento é $O(m)$. Uma versão deste algoritmo incrementada com o mecanismo de replicação é utilizada pelo *middleware* OurGrid [13];

- MCT (*Minimum Conclusion Time*): é uma heurística *on-line* que atribui cada tarefa para a máquina com o menor tempo de conclusão esperado para realizá-la. O tempo de conclusão corresponde à soma do tempo que levará para a máquina estar disponível para executar a tarefa mais o tempo que a tarefa demora para ser executada. Este algoritmo mapeia mais de uma tarefa por recurso. A complexidade de mapeamento é $O(m)$ pois, assim que uma tarefa chega, todas as máquinas na grade são examinadas para determinar a máquina que possui o menor tempo de conclusão para a mesma.

- Min-min: é uma heurística do tipo *batch* baseada no MCT, cujo pseudo-código é apresentado na Figura 2.2. O Min-min recebe como parâmetros de entrada um conjunto de tarefas não mapeadas M (metatarefa) e um conjunto de máquinas R constantes da grade. Ao iniciar, o algoritmo calcula o tempo de conclusão que cada tarefa de M levaria para ser concluída em cada máquina constante em R (linhas de 2 até 6). $Tc_{i,j}$ representa o tempo de conclusão da tarefa t_i na máquina m_j, calculado pela soma de Tp_j (tempo esperado para que a máquina m_j esteja pronta para executar uma tarefa) e $Te_{i,j}$ (tempo que o recurso m_j levaria para executar a tarefa t_i).

  A seguir, o algoritmo procura o menor tempo de conclusão para cada tarefa obtendo, também, a máquina que alcançaria este valor (linhas de 8 a 10). O Min-min, então, procura a tarefa t_k com o tempo de conclusão mínimo entre todas as tarefas de M e a atribui para a máquina m_x em cuja execução este tempo seria obtido (linhas 11 e 12). A tarefa mapeada é removida da metatarefa M (linha 13). Uma vez que o mapeamento da tarefa t_k para a máquina m_x incrementa o tempo que esta máquina ficará ocupada (Tp_x) (linha 14), isto requer a atualização do tempo de conclusão ($Tc_{i,x}$) de m_x para o restante das tarefas t_i de M (linha

15). As operações das linhas 7 até 16 são repetidas enquanto houver tarefa para ser mapeada na grade. Assim como o MCT, este algoritmo mapeia mais de uma tarefa por recurso.

```
1  início
       Entrada: M = conjunto de tarefas não mapeadas; R = conjunto de
                máquinas na grade;
2      para cada Tarefa t_i contida em M faça
3          para cada Máquina m_j contida em R faça
4              Tc_{i,j} = Tp_j + Te_{i,j};
5          fim
6      fim
7      enquanto existir tarefa não mapeada faça
8          para cada Tarefa t_i contida na MetaTarefa M faça
9              Procura seu menor tempo de conclusão e a máquina que obtém ele;
10         fim
11         Procura a tarefa t_k com o tempo de conclusão mínimo entre todas as
           tarefas de M;
12         Atribui a tarefa t_k para a máquina m_x que obtém o menor tempo de
           conclusão;
13         Remove a tarefa t_k de M;
14         Atualiza Tp_x;
15         Atualiza o tempo de conclusão das tarefas Tc_{i,x} para a máquina m_x;
16     fim
17 fim
```

Figura 2.2: Min-min

Sendo m a quantidade de tarefas de M e r a quantidade de recursos constante em R, o cálculo do tempo de conclusão de cada tarefa em cada máquina constante de R (linhas 2 a 6) custa $O(mr)$. O laço nas linhas 7 a 16 é repetido m vezes e cada interação custa $O(mr)$. Logo, o custo total do algoritmo é $O(m^2r)$.

- *Task Grouping*: o algoritmo *Task Grouping* é voltado para o escalonamento de aplicações constituídas por uma grande quantidade de tarefas de curta duração. Neste caso, o escalonamento e distribuição de cada tarefa individualmente levaria a uma sobrecarga do sistema durante a transmissão das tarefas para os recursos da grade. O algoritmo agrupa as tarefas da aplicação de acordo com seus

tamanhos de computação (medido em milhões de instruções) e a capacidade de processamento dos recursos da grade. Cada grupo é enviado a um único recurso, reduzindo-se a carga de transmissão das tarefas.

2.4 Predição de Desempenho de Aplicações de Grade

Algoritmos como o Min-min e o MCT necessitam conhecer o tempo que as aplicações levariam para ser executadas nos recursos da grade. Esta informação pode ser estimada através de algoritmos de predição [38] que seguem basicamente duas abordagens. Na primeira abordagem calcula-se a estimativa do tempo de execução da aplicação baseado no registro de execuções anteriores da mesma ou de aplicações semelhantes. A segunda abordagem é baseada no conhecimento do modelo de execução da aplicação, usualmente aplicações paralelas acopladas com cargas de trabalho divisíveis (ex: MPI ou PVM). O código da aplicação é analisado, estimando-se o tempo de execução de cada tarefa de acordo com a capacidade dos recursos da grade.

Um exemplo desta última abordagem é o sistema PACE (*Performance Analysis and Characterisation Environment*) [34]. Sua arquitetura, ilustrada na Figura 2.3, é composta por ferramentas de aplicação e ferramentas dos recursos. As ferramentas da aplicação realizam uma análise do código fonte das aplicações através da qual se obtêm a freqüência com que as operações são executadas e a estrutura da comunicação entre as tarefas. As ferramentas de recursos analisam características dos recursos da grade como capacidade de CPU, *cache* e enlaces de rede. A análise do código permite gerar um modelo da aplicação que descreve suas necessidades computacionais, já o modelo do recurso descreve capacidades de *hardware*. Através do cruzamento dos mesmos pode-se predizer o tempo de execução da aplicação e encontrar mapeamentos para as tarefas que resultem em um tempo menor de execução para as aplicações.

2.5 Escalonamento em Sistemas de *Middleware* de Grade

Esta seção descreve e compara a política de escalonamento de aplicações em alguns dos principais projetos de grades computacionais.

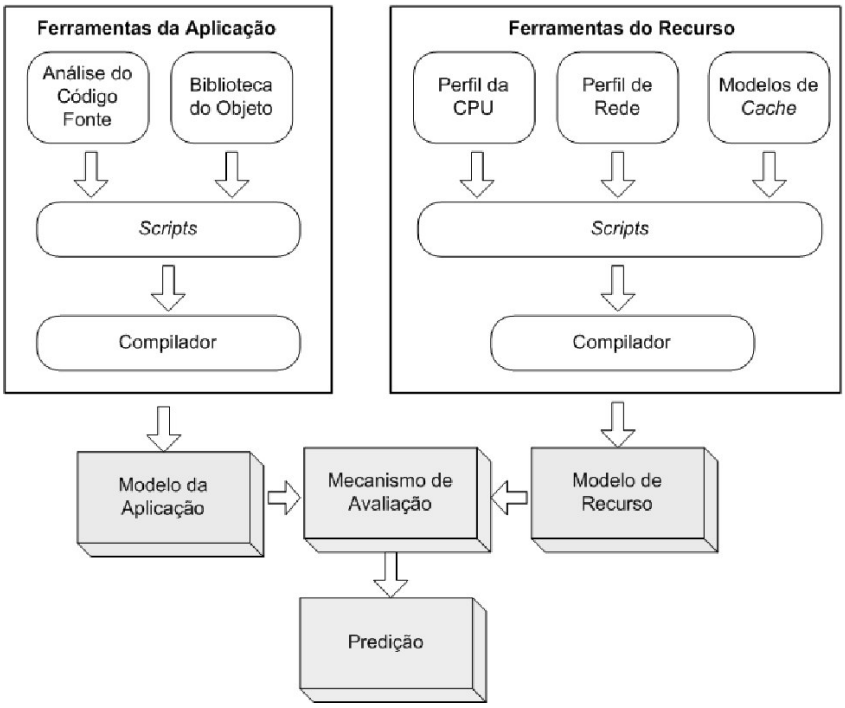

Figura 2.3: Arquitetura do PACE

2.5.1 Globus

O Globus[1] [23, 24] é o projeto de computação em grade com maior relevância na atualidade. Ele é desenvolvido pela *Globus Alliance*, um grupo formado por inúmeras instituições ao redor de todo o mundo, cujos principais responsáveis são: *Argonne National Laboratory*, *University of Southern California*, *University of Chicago*, *University of Edinburgh*, *Swedish Center for Parallel Computers* e o *National Center for Supercomputing Applications* (NCSA). Além disso, o projeto Globus conta também com o apoio de várias empresas como a Microsoft e a IBM.

O projeto de computação em grade desenvolvido pela aliança chama-se *Globus Tookit*. O *Globus Tookit* fornece um conjunto de serviços e bibliotecas de *software* de arquitetura e código abertos para o suporte às grades e às aplicações de grades. O *toolkit* possui componentes responsáveis pelo tratamento de vários aspectos relativos aos ambientes de grades, como segurança, descoberta de informação, gerenciamento de recursos, gerenciamento de dados, comunicação, detecção de falhas e portabilidade.

O *Globus Toolkit* não provê um componente responsável pelo escalonamento de aplicações. No entanto, ele possui um componente responsável pelo gerenciamento

[1]Página Inicial: `http://www.globus.org`

de recursos na grade chamado de GRAM (*Globus Resource Allocation Manager*), a fim de prover uma base para que sistemas de escalonamento de aplicações independentes sejam agregados ao *Globus Toolkit*. Um exemplo de sistema de escalonamento escrito para o Globus é o Nimrod-G, descrito a seguir.

Nimrod-G

O Nimrod-G [8] é um sistema de escalonamento de grade que usa uma arquitetura orientada à economia computacional para o escalonamento de aplicações paralelas não acopladas. Atualmente ele possui módulos que permitem a integração com o *Globus Toolkit* e o Legion [36]. O Nimrod-G utiliza os serviços da GRACE (*Grid Architecture for Computational Economy*) para selecionar recursos apropriados. A GRACE é uma arquitetura para gerenciamento de recursos em uma grade computacional baseada na utilização de modelos econômicos.

Em modelos computacionais econômicos, os algoritmos de escalonamento necessitam se adaptar à variação das cargas de trabalho e das disponibilidades de recursos na grade e ao mesmo tempo atender às restrições de prazo de conclusão e orçamento das aplicações. O prazo representa o tempo que o usuário está disposto a esperar pelo resultado da aplicação. O orçamento é definido em termos de quanto dinheiro o usuário tem disponível para pagar ao dono do recurso que deseja usar.

O proprietário de cada recurso da grade pode, sempre que quiser, redefinir o valor cobrado pelo uso do recurso. O valor a ser cobrado para a execução de uma aplicação pode ser definido em função de milhões de instruções a serem executadas, definindo-se um valor para cada tarefa ou de acordo com o tempo de CPU utilizado.

O Nimrod-G é um escalonador no nível de aplicação. Quando um usuário da grade vai submeter uma aplicação para a grade ele deve especificar o prazo de conclusão da aplicação, o orçamento disponível para sua execução e o algoritmo de escalonamento que será utilizado. São disponibilizadas três opções para o algoritmo de escalonamento:

1. **Otimização de custo, com restrições de prazo e orçamento**: este algoritmo tenta finalizar a aplicação com o menor custo possível dentro de um determinado prazo. O funcionamento deste algoritmo é o seguinte:

(a) Ordena os recursos em ordem crescente de custo (dinheiro cobrado por uso de CPU a cada segundo);

(b) Atribui-se para cada recurso o maior número de tarefas possíveis, desde que não exceda o prazo de conclusão da aplicação.

2. **Otimização de tempo, com restrições de prazo e orçamento**: este algoritmo tenta finalizar a aplicação o mais rápido possível, dentro de um orçamento disponível. O funcionamento deste algoritmo é o seguinte:

   (a) Para cada recurso, é calculado o tempo de conclusão da tarefa, levando-se em conta as tarefas já atribuídas ao recurso;

   (b) Ordena os recursos pelo menor tempo de conclusão da tarefa;

   (c) Escalona a tarefa para o primeiro recurso cujo custo por tarefa seja menor ou igual ao orçamento restante por tarefa;

   (d) Repete os passos até que todas as tarefas sejam escalonadas.

3. **Otimização de tempo, com restrições no prazo e com orçamento ilimitado**: este algoritmo é uma variação do algoritmo anterior no qual são escolhidos os recursos que executarão a aplicação de forma mais rápida, independentemente do custo gerado.

2.5.2 OurGrid

O OurGrid[2] [5, 14] é um projeto de grade computacional desenvolvido em conjunto pela Universidade Federal de Campina Grande e Hewlett-Packard (HP). O OurGrid é um sistema de grade cooperativo, aberto e gratuito, no qual laboratórios podem doar poder computacional em troca de acessar, quando necessário, recursos ociosos pertencentes a outros laboratórios. Atualmente, o OurGrid é voltado apenas para a execução de aplicações paralelas não acopladas.

A arquitetura da comunidade OurGrid foi projetada para ser escalável e permitir que milhares de aglomerados possam estar conectados ao sistema. Esta escalabilidade provém do fato de basear-se em redes P2P (*peer-to-peer*), em que cada

[2]Página Inicial: `http://www.ourgrid.org`

aglomerado representa um *peer*, criando um esquema de grade descentralizado. Entretanto, redes P2P podem ter o seu desempenho comprometido por causa de *peers* chamados *freeriders* [31]. Um *freerider* é um *peer* que somente consome recursos, nunca contribuindo com a comunidade. Este comportamento poderia ter um impacto negativo no OurGrid, dado que muitos usuários apresentam uma insaciável demanda por recursos computacionais. Para lidar com este problema foi criado um mecanismo de *rede de favores* (*network of favors*), um mecanismo de alocação de recursos totalmente descentralizado e autônomo que marginaliza os *freeriders*.

No OurGrid, o algoritmo de escalonamento é implementado na própria ferramenta utilizada pelo usuário para a submissão de aplicações adotando-se, portanto, uma abordagem distribuída. As aplicações submetidas para a grade são formadas por três tipos de tarefas: *inicial*, *grade* e *final*, que são executadas em ordem seqüencial. As tarefas *inicial* e *final* são executadas localmente na máquina do usuário. A tarefa *inicial* é utilizada para configurar o ambiente da tarefa como, por exemplo, realizar a transferência de dados de entrada para máquina da grade selecionada para executar a tarefa. A tarefa *final* é tipicamente usada para coletar os resultados das tarefas de volta para a máquina do usuário. A tarefa *grade* executa em uma máquina da grade e realiza a computação em si. O escalonamento das tarefas do tipo *grade* para os recursos computacionais pode ser feito através de dois algoritmos: o *Work Queue with Replication* (WQR), que visa escalonar aplicações computacionalmente intensivas e o *Storage Affinity*, que leva em conta o uso intensivo de dados pelas aplicações.

O algoritmo WQR basicamente adiciona o mecanismo de replicação ao algoritmo *Workqueue*, apresentado em seção anterior. Sua execução inicia da mesma maneira que o *Workqueue* até que todas as tarefas sejam escalonadas. Cada recurso da grade executa apenas uma tarefa por vez. A partir deste ponto, o algoritmo cria réplicas das tarefas que estão executando. O mecanismo de replicação cria réplicas apenas quando existir algum recurso disponível e não existir mais tarefas para serem escalonadas. O WQR estabelece um limite para a replicação, ou seja, uma tarefa é replicada somente se um número máximo pré-definido de réplicas não tiver sido alcançado. Por exemplo, considere que a quantidade de réplicas pré-definida seja 2. O WQR escolhe aleatoriamente as tarefas que ainda estão executando e cria uma réplica para cada tarefa. As réplicas são escalonadas da mesma maneira que as tarefas originais. Caso todas as réplicas já tenham sido escalonadas, as tarefas que ainda estão

executando são replicadas novamente, até que todas as tarefas em execução possuam duas réplicas. O limite da replicação é alcançado quando todas as tarefas em execução possuem duas réplicas. Quando uma tarefa é finalizada, suas réplicas são canceladas.

O algoritmo *Storage Affinity* [44] foi desenvolvido com o objetivo de otimizar o desempenho de aplicações que fazem uso intensivo de grandes quantidades de dados. O valor de afinidade entre uma tarefa e um *peer* é definido pela quantidade de *bytes* dos dados de entrada de uma tarefa que já está armazenada neste *peer*. Quanto maior a quantidade de *bytes* armazenados, maior será o valor da afinidade entre a tarefa e o *peer*.

O algoritmo *Storage Affinity* pode ser dividido em duas fases: na primeira fase, o algoritmo calcula para cada tarefa que compõe a aplicação submetida seu valor de afinidade com relação a cada um dos *peers* da grade. A seguir, o algoritmo ordena todos os valores de afinidade calculados na etapa anterior e os percorre do maior para o menor valor, até que todas as tarefas da aplicação tenham sido mapeadas para os *peers*, de forma a minimizar o volume de dados de entrada que devem ser transferidos. A segunda etapa refere-se a replicação das tarefas. As tarefas são replicadas seguindo-se os seguintes critérios:

1. A tarefa deve possuir um valor de afinidade positivo com algum *peer* que possua recursos disponíveis;
2. O grau de replicação da tarefa deve ser o menor entre todas as tarefas que estão executando;
3. A tarefa deve possuir o maior valor de afinidade entre todos as tarefas candidatas restantes.

Assim como o WQR, o *Storage Affinity* permite pré-definir um limite de réplicas.

2.5.3 Condor

O projeto Condor[3] [37, 51] é um projeto de computação em grade desenvolvido na Universidade de Wisconsin-Madison. Trata-se um projeto bastante maduro,

[3]Página Inicial: `http://www.cs.wisc.edu/condor`

dado que está em produção desde a década de 80, para uso acadêmico e empresarial. Este projeto está focado em desenvolver uma arquitetura que integre recursos computacionais para a execução de computações intensivas. Este sistema permite dois diferentes modos de operação: grade dedicada e grade oportunista. O objetivo do ambiente de grade dedicada é prover aos seus usuários grandes quantidades de poder computacional por longos períodos de tempo, utilizando todos os recursos disponíveis na mesma. Já o objetivo da grade oportunista é utilizar somente os recursos que estiverem disponíveis em um dado momento, sem requerer 100% dos recursos das máquinas.

O Condor permite a execução de aplicações seqüenciais, paralelas não acopladas e paralelas acopladas. Entre as aplicações paralelas acopladas, podem ser executadas aplicações do tipo MPI e PVM. O suporte às aplicações MPI é limitado [52]. Estas aplicações só podem ser executadas em recursos reservados, ou seja, máquinas que: (1) quando estão executando uma aplicação MPI não interrompem ou suspendem a execução em nenhuma hipótese, e (2) caso tais máquinas estejam executando uma aplicação seqüencial e recebam uma requisição para executar uma aplicação paralela, a aplicação seqüencial deve ser interrompida imediatamente, podendo ser migrada conforme o caso. Ou seja, tais limitações praticamente impedem a execução de aplicações MPI sobre recursos compartilhados. Por outro lado, aplicações do tipo PVM podem ser executadas em recursos compartilhados, uma vez que o próprio modelo PVM permite que nós sejam adicionados ou removidos de uma aplicação em execução. Porém, aplicações PVM a serem executadas sobre o Condor devem seguir o paradigma mestre-escravo. Um nó especial da aplicação, o mestre, é executado na própria máquina que submete a aplicação para execução. Tal nó é responsável por distribuir a computação aos escravos e é informado caso algum escravo falhe.

Uma grade Condor é formada por aglomerados de máquinas chamados *Condor Pool*. Cada aglomerado, em geral, pertence a um domínio administrativo distinto (apesar de isto não ser obrigatório). Para cada aglomerado existe um escalonador central do tipo *batch*, que periodicamente escalona tarefas dentro de um mesmo ou em diferentes aglomerados.

O Condor calcula continuamente a quantidade de máquinas disponíveis que poderá ser alocada para cada usuário, como descrito em [1]. Esta quantidade está inversamente relacionada com a razão entre as prioridades dos usuários. Por

exemplo, um usuário com prioridade 10 tem direito a duas vezes mais máquinas do que um usuário com prioridade 20. A prioridade de cada usuário muda de acordo com o número de recursos que ele está usando. Cada usuário começa com a prioridade de 0.5 (a melhor prioridade permitida). Se o número de máquinas que um usuário está usando atualmente é maior do que a sua prioridade, a prioridade vai aumentar numericamente (piora) ao longo do tempo, e se ela for menor, então a prioridade vai diminuir numericamente (melhorar) ao longo do tempo. A prioridade do usuário é avaliada diariamente e é ajustada de forma exponencial da seguinte forma: se um usuário com a prioridade 100 está utilizando 100 máquinas e então termina todas suas tarefas, um dia depois sua prioridade passa a ser 50, dois dias depois a prioridade será 25, etc. O Administrador Condor também poderá elevar ou reduzir a prioridade de determinado usuário diretamente.

No processo de mapeamento, as aplicações dos usuários com maior prioridade são atendidas primeiras, selecionando-se apenas os recursos que atendam aos requisitos definidos para sua execução (ex: máquinas com ao menos 1Gb de memória) e ordenando-os de acordo com as preferências estabelecidas, tais como maior quantidade de memória livre.

O Condor utiliza uma abordagem preemptiva: tarefas de usuários de baixa prioridade iniciadas recentemente (a menos de uma hora) são interrompidas, cedendo seus recursos para usuários de maior prioridade. O Condor realiza o *checkpoint* destas tarefas que são reiniciadas posteriormente assim que recursos tornarem-se disponíveis.

2.5.4 InteGrade

O *middleware* InteGrade[4] é um sistema orientado a objetos que provê uma infra-estrutura de *software* robusta e flexível para computação em grade oportunista. A unidade arquitetural básica de uma grade InteGrade é um aglomerado, uma coleção de máquinas geralmente conectadas por uma rede local. Aglomerados podem ser organizados de forma hierárquica, podendo abranger um grande número de máquinas. Cada aglomerado contém um nó gerenciador que executa componentes do InteGrade responsáveis pelo gerenciamento de recursos computacionais e pela comunicação entre aglomerados. Os outros nós do aglomerado são estações de trabalho que exportam

[4]Página Inicial: `http://integrade.incubadora.fapesp.br/portal`

parte dos seus recursos para usuários da grade. Os nós que provêem recursos podem ser máquinas compartilhadas ou dedicadas.

Cada aglomerado do InteGrade possui um escalonador central, responsável por realizar o escalonamento das aplicações nos recursos locais a este aglomerado. No entanto, caso não haja recursos suficientes para realizar o escalonamento de um conjunto de tarefas pertencentes a uma mesma aplicação paralela, o escalonador encaminha a aplicação para que seja executada em outro aglomerado da hierarquia.

Atualmente o InteGrade permite a execução de três classes de aplicações:

a) Aplicações regulares, onde o código executável é atribuído para um único nó da grade;

b) Aplicações paralelas não acopladas, onde diversas cópias do código executável (tarefas) são atribuídas para diferentes nós da grade e cada uma delas processa um subconjunto de dados de entrada independentemente e sem troca de dados entre si. Estas aplicações são também conhecidas como aplicações paramétricas ou *Bag-of-Tasks*;

c) Aplicações paralelas acopladas, seguem modelos de aplicações BSP ou MPI, cujos processos ocasionalmente trocam dados entre si.

O usuário do InteGrade pode, opcionalmente, especificar os requisitos indispensáveis da aplicação como, por exemplo, a arquitetura para a qual a aplicação foi compilada e as preferências que guiam a ordenação para a escolha dos recursos candidatos (e.g. recursos que possuem a maior quantidade de memória livre). De posse dessas informações sobre as exigências da aplicação, o escalonador do aglomerado seleciona e ordena um conjunto de recursos candidatos. As tarefas das aplicações são então escalonadas para os recursos de acordo com a lista de recursos ordenados.

2.5.5 Comparação entre o Escalonamento nos Principais Sistemas de *Middleware* de Grade

Esta subsecção apresenta uma análise comparativa entre as abordagens de escalonamento dos principais sistemas de *middleware* de grade. Tomamos como base os seguintes critérios de comparação: esquema de escalonamento (centralizado,

distribuído ou hierárquico), número de escalonadores por aglomerado, modo de escalonamento (*batch* ou *on-line*), tipo de aplicações (regulares, paralelas não acopladas, paralelas acopladas, etc), uso de informações sobre a aplicação (como o tempo de execução estimado), uso de informações sobre as características das máquinas (CPU, memória, etc) e uso do mecanismo de replicação.

Podemos observar na tabela 2.1 que todos os projetos de grade possuem um esquema de escalonamento não centralizado. Isto é motivado por questões de escalabilidade e desempenho, uma vez que pode ser solicitado ao escalonador da grade tratar centenas de submissões de aplicações ao mesmo tempo e gerenciar milhares de informações sobre os provedores de recursos. Esta abordagem é também reforçada pela autonomia dos domínios administrativos. InteGrade, Condor e Globus/Nimrod-G utilizam um escalonador por aglomerado. O OurGrid utiliza um escalonador por usuário. Desta forma podem ser definidas diferentes políticas de escalonamento de aplicações para cada domínio administrativo.

É interessante observar que a maioria dos projetos utilizam o modo de escalonamento *on-line*, com exceção do Condor que funciona como um sistema *batch*. Os projetos que usam o modo *on-line* não se preocupam em comparar propriedades das aplicações entre si, geralmente devido à falta de atributos comparáveis ou motivados por atender imediatamente aos donos das aplicações. Através do modo de escalonamento *batch*, o Condor pode comparar as prioridades dos usuários referentes a um conjunto de submissões realizadas em um determinado intervalo de tempo e tentar balancear a utilização dos recursos para diferentes usuários.

Todos os projetos se concentraram pelo menos no suporte a dois tipos de aplicações: regulares e paralelas não acopladas. InteGrade e Condor permitem também a execução de aplicações paralelas acopladas.

Devido à não integração, até o momento, de técnicas para estimar o tempo de execução das tarefas, a maior parte dos sistemas de *middleware* apresentados não fazem uso desta informação. No entanto, o Niromd-G utiliza esta informação para tentar otimizar o tempo que as aplicações levam para serem concluídas. Por outro lado, o uso de informações estáticas e dinâmicas sobre o *hardware* do provedor de recurso é geralmente utilizado pelas estratégias de escalonamento, a fim de atender requisitos e preferências das aplicações (ex: Nimrod-G, Condor, InteGrade).

Finalmente, o mecanismo de replicação é utilizado pela estratégia de escalonamento do OurGrid e do InteGrade [16], tanto para prover um mecanismo de tolerância a falhas na execução de aplicações quanto para aumentar a probabilidade de que sejam enviadas para processadores mais rápidos do que os relacionados às tarefas originais, implicando em uma redução no tempo total de execução da aplicação.

Projeto de grade	Esquema de escalonamento	Número de escalonadores por aglomerado	Modo de escalonamento	Tipo de aplicações	Informações requeridas para o escalonamento	Uso de replicação	Algoritmo
Condor.	Distribuído.	1.	*Batch*.	Regulares, paralelas não acopladas, MPI e PVM.	Prioridades de usuário; características dos recursos; e requisitos e preferências das aplicações.	Não.	Prioridade de usuários com requisitos e preferências de aplicações.
InteGrade.	Hierárquico.	1.	*On-line*.	Regulares, paralelas não acopladas, MPI e BSP.	Características dos recursos; e requisitos e preferências das aplicações.	Sim.	Preferências e requisitos de aplicações.
OurGrid.	Distribuído.	Vários.	*On-line*.	Regulares e paralelas não acopladas.	Afinidade da tarefa com o *peer*.	Sim.	*Work queue with Replication* e *Storage Affinity*.
Globus / Nimrod-G.	Distribuído.	1.	*On-line*.	Regulares e paralelas não acopladas.	Tempo de execução estimado da tarefa da aplicação.	Não.	Otimização de custo e de tempo, com restrições de prazo e orçamento.

Tabela 2.1: Resumo comparativo do escalonamento nos principais sistemas de *middleware* de grade

2.6 Conclusões

Este capítulo apresentou os fundamentos do processo de escalonamento de aplicações em grades de computadores utilizado em nosso trabalho. Foram descritas as principais características de uma arquitetura básica de escalonamento em grades como a submissão de aplicações, seleção de recursos e execução da aplicação. Além disso, apresentamos os esquemas de escalonadores de grades.

Apresentamos ainda importantes algoritmos de escalonamento utilizados no contexto de grades e as estratégias de escalonamento dos principais projetos de grade. Diferenciaram-se os algoritmos pelo modo de escalonamento (*on-line* e *batch*) e pelo uso ou não das informações sobre o tempo estimado de execução das tarefas e sobre as características de *hardware* dos provedores de recursos. Estes aspectos, aliado às características de grades oportunistas, como a inexistência de um conjunto de recursos dedicados para escalonar aplicações, foram levados em consideração no projeto OGST.

3 OGST: Uma Ferramenta de Simulação para Grades Oportunistas

O OGST (*Opportunistic Grid Simulation Tool*) [22] é uma ferramenta de simulação cujo principal objetivo é ajudar desenvolvedores de sistemas de *middleware* de grades oportunistas na avaliação de novos conceitos e em suas implementações, considerando diferentes condições do ambiente de execução e cenários diversos. Nossa motivação inicial para o desenvolvimento do OGST foi prover uma ferramenta para avaliar o comportamento de algoritmos de escalonamento comumente usados em ambientes de grades quando sujeitos a diferentes condições do ambiente de execução, permitindo a investigação de abordagens de escalonamento adaptativo e reescalonamento dinâmico de aplicações.

Considerando a taxonomia do mecanismo de simulação descrito por Sulistio et al [48], o OGST é um simulador de eventos discretos (DES) orientado a objetos. Na simulação de eventos discretos, a passagem do tempo é percebida de forma discreta. A ação que provoca uma mudança de estado em um determinado instante é um evento, como por exemplo, se uma máquina está no estado "funcionando" e acontece uma "falha", a máquina passa a não fazer mas parte da grade. Logo, a "falha" é um evento. Cada evento deve estar associado a um instante de tempo, que é o instante que o evento ocorre e provoca uma mudança de estado. Os eventos devem ocorrer obedecendo a uma ordem cronológica, ou seja, os eventos de menor tempo devem ser tratados primeiros. Na simulação o tempo é uma variável global chamada de tempo de simulação e deve ser sempre igual ao tempo associado ao evento que está acontecendo no momento, ou o último evento que ocorreu. O intervalo de tempo entre cada evento não é necessariamente igual, uma vez que a simulação é orientada a eventos e não a tempo. Um evento deve simplesmente anunciar que está ocorrendo e todo o sistema deve reagir a essa ocorrência [3].

O OGST permite a simulação de aplicações e recursos em grande escala, além de permitir a criação de cenários envolvendo diversos usuários de uma maneira

controlada e repetitiva. Ele foi construído tendo por base o GridSim[1] [9], um conjunto de ferramentas para simulação de grades de computadores. O GridSim foi escolhido como base para este trabalho uma vez que fornece um conjunto de entidades que funcionam como base para a modelagem de aplicações, recursos, usuários e comunicação entre componentes de uma simulação de grade computacional. Sua interface de programação é organizada, possui um bom nível de documentação, trata-se de um projeto em constante atualização e os seus desenvolvedores oferecem um bom suporte no esclarecimento de questões relativas ao uso de suas funcionalidades.

Na seção seguinte descrevemos as grades oportunistas. No restante do capítulo apresentamos os requisitos, a arquitetura e os detalhes de implementação do OGST.

3.1 Grades Oportunistas

Atualmente, as instituições privadas e públicas têm um grande número de recursos de computação, tais como computadores pessoais e estações de trabalho, com grande capacidade de processamento e armazenamento de dados. Os computadores estão ociosos na maior parte do tempo e, mesmo quando estão em uso, normalmente apenas uma pequena percentagem de sua capacidade de computação é efetivamente utilizada [40, 17, 41]. Grades oportunistas são sistemas computacionais que provêem meios para usar uma base instalada de computadores pessoais para execução de aplicações computacionais de alto desempenho, aproveitando o poder de computação ocioso disponível [27].

O foco de um *middleware* de grade oportunista não é a integração de aglomerados de computadores dedicados (ex: Beowulf) ou recursos de supercomputação, mas é aproveitar ciclos de computação ociosa de computadores regulares e estações de trabalho que podem está espalhados através de diversos domínios administrativos.

Desenvolvedores de um *middleware* de grade oportunista devem tratar diversos desafios tais como:

a) Grande instabilidade, uma vez que nós são geralmente não dedicados e aplicações não executam sobre um ambiente controlado;

[1] Página Inicial: `http://www.gridbus.org/gridsim/`

b) Alta heterogeneidade de recursos computacionais e de enlaces de rede, uma vez que computadores estão geralmente espalhados através de diferentes domínios administrativos;

c) O *middleware* não deveria interferir no uso regular dos recursos computacionais ou, pelo menos, deveria prover o mínimo de impacto sobre o desempenho percebido por seus usuários, caso contrário será difícil obter o seu consentimento para o uso do recurso;

d) É desejável o fornecimento de mecanismos para prever o período de tempo, no futuro, em que uma máquina estará ociosa, minimizando a necessidade de migração de código.

3.2 Requisitos de Projeto

O projeto do OGST levou em consideração características das grades oportunistas [26] estabelecendo-se, assim, os seguintes requisitos para seu desenvolvimento:

1. Deve-se prover suporte à definição de ambientes de grades que exibem alta heterogeneidade de máquinas e enlaces de rede;

2. Deve-se permitir a definição de aplicações heterogêneas, desde aplicações regulares até aplicações paralelas não acopladas e acopladas, que consomem uma grande quantidade de tempo. O tempo de execução destas aplicações pode variar de poucas horas até semanas;

3. Deve-se permitir a simulação de nós sendo agregados e deixando a grade frequentemente, uma vez que nós geralmente não são dedicados em grades oportunistas, o que torna o ambiente da grade bastante volátil;

4. Deve-se permitir a injeção de falhas de nós e de enlaces de rede, devido à instabilidade comum em grades oportunistas;

5. Deve-se prover suporte para simulação de mecanismos de tolerância a falhas comumente aplicados em grades oportunistas (como, por exemplo, reinício, *check-*

pointing e replicação [16]) a fim de garantir o progresso de execução da aplicação mesmo em um ambiente de execução instável;

6. Deve-se permitir a simulação da variação de disponibilidade de cada nó da grade, considerando diferentes períodos de uso;

7. Deve-se prover os meios que permitam aos algoritmos de escalonamento obterem diversas informações relativas às aplicações e ao ambiente de execução, tais como o tempo de conclusão estimado para cada tarefa, o tamanho dos arquivos de entrada, a capacidade de processamento e a carga dos nós da grade. Desta forma, o simulador permitirá o desenvolvimento de simulações envolvendo os mais diversos algoritmos de escalonamento;

8. Deve-se permitir a substituição dinâmica do algoritmo de escalonamento e/ou ajuste dinâmico dos parâmetros de escalonamento durante a simulação a fim de permitir a avaliação de abordagens de escalonamento adaptativo;

9. Deve-se disponibilizar os meios que permitam o uso de rastreamentos (*traces*) coletados de ambientes reais (tais como disponibilidade de nós) além do uso de dados sintéticos;

10. Deve-se prover o armazenamento de dados relevantes sobre a simulação, tais como tempos referentes à submissão e conclusão de cada tarefa que compõe uma aplicação.

3.3 Arquitetura do OGST

A arquitetura do OGST segue uma abordagem orientada a objetos. O OGST foi desenvolvido no contexto do projeto InteGrade mas foi projetado para permitir a simulação de grades oportunistas de uma maneira em geral com o propósito de ser aplicado a outros projetos de pesquisa envolvendo sistemas de *middleware* de grade.

A arquitetura do OGST é baseada na arquitetura básica do escalonador de aplicações em grades de computadores descrita na seção 2.1. A Figura 3.1 ilustra os principais componentes do OGST. O `Grid Feature Generator` (GFG) é um componente usado para definir um ambiente de grade simulado (nós da grade) e

as aplicações a serem executadas com suas respectivas taxas de chegada. O OGST atualmente permite a simulação de aplicações regulares e paralelas não acopladas. Para cada tarefa de uma aplicação deve-se prover seu tamanho, definido em milhões de instruções. O `User Application Submission Tool` (UAST) representa o usuário da grade e é o componente responsável pela submissão da aplicação, recebendo uma notificação sobre sua conclusão.

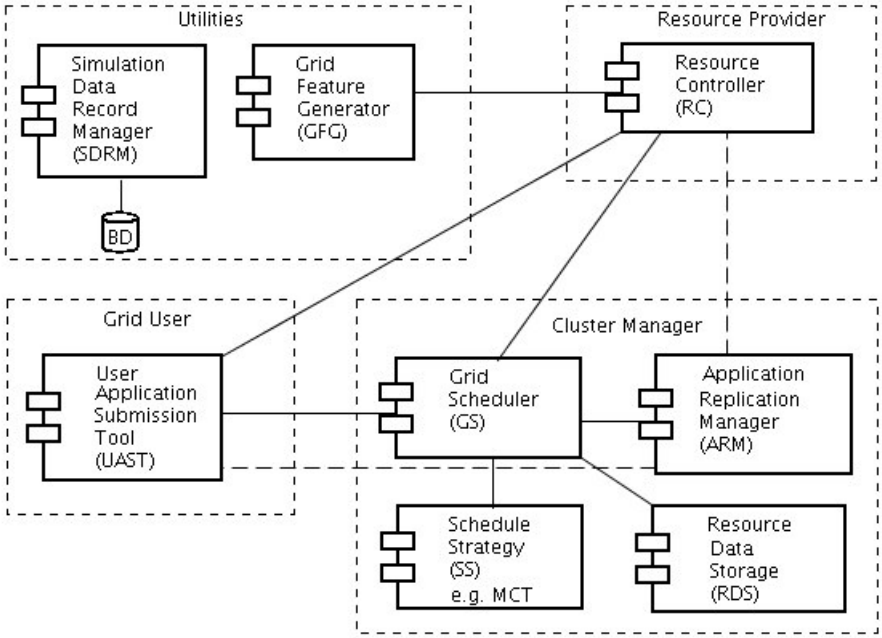

Figura 3.1: Principais componentes do OGST

O `Grid Scheduler` (GS) recebe submissões para execução de aplicações do UAST e executa o algoritmo de escalonamento, encapsulado no componente `Scheduling Strategy` (SS). A estratégia de escalonamento usa dados sobre a disponibilidade dos recursos da grade providos pelo componente `Resource Data Storage` (RDS). Cada tarefa da aplicação é então mapeada para a execução em um nó específico da grade. Cada nó da grade executa um `Resource Controller` (RC), responsável pela instanciação e execução das tarefas das aplicações escalonadas para o nó, mantendo uma lista de tarefas esperando pela execução. Ele também é responsável pela simulação da variação da carga do recurso local. O `Simulation Data Record Manager` (SDRM) usa uma base de dados relacional para o armazenamento dos dados da simulação coletados, tais como os tempos de inicio e conclusão das tarefas. O componente RDS é também responsável pela simulação da ocorrência de falhas e recuperação de nós.

O OGST permite ainda a simulação da execução de aplicações com replicação, técnica comumente usada em ambientes de grades oportunistas a fim de contornar eventuais falhas de nós. O `Application Replication Manager` (ARM) é o componente responsável pelo gerenciamento da execução de réplicas.

A versão atual do OGST organiza os recursos pertencentes à grade em único aglomerado. Como trabalho futuro pretende-se desenvolver o suporte a outras formas de organização, como agrupamento de aglomerados estruturados de forma hierárquica.

3.4 Implementação do OGST

O OGST foi escrito em Java e é uma extensão do conjunto de ferramentas chamado GridSim [9] que, por sua vez, herda características de gerenciamento de eventos e de *threads* do SimJava [30]. O GridSim acrescenta características de redes de comunicação e de entregas de eventos ao SimJava que permitem comunicação síncrona ou assíncrona para o acesso e liberação de serviços. Os componentes do OGST estendem uma classe base do GridSim chamada também de `GridSim` que disponibiliza os mecanismos para a comunicação entre os mesmos. A troca de mensagens entre os componentes do OGST se baseia em uma abordagem dirigida a eventos e é implementada no método `body()` provido pela classe `GridSim`. As extensões providas pelo OGST para o GridSim são apresentadas pelo diagrama de classe mostrado na Figura 3.2 e são descritas ao longo desta seção.

3.4.1 Geração de Recursos Computacionais

O OGST permite a criação automática de ambientes de grades simulados compostos por uma grande quantidade de nós heterogêneos. Esta característica é provida pelo método `generateMachines (simulationDataPath, totalOfMachines, slowCapacity, fastCapacity)` da classe `utilities.GridFeatureGenerator`. Este método recebe como parâmetros de entrada o caminho onde um arquivo texto contendo a descrição das máquinas geradas pelo OGST será armazenado (`simulationDataPath`), a quantidade de máquinas desejada (`totalOfMachines`) e a capacidade de processamento do nó da grade mais lento (`slowCapa-`

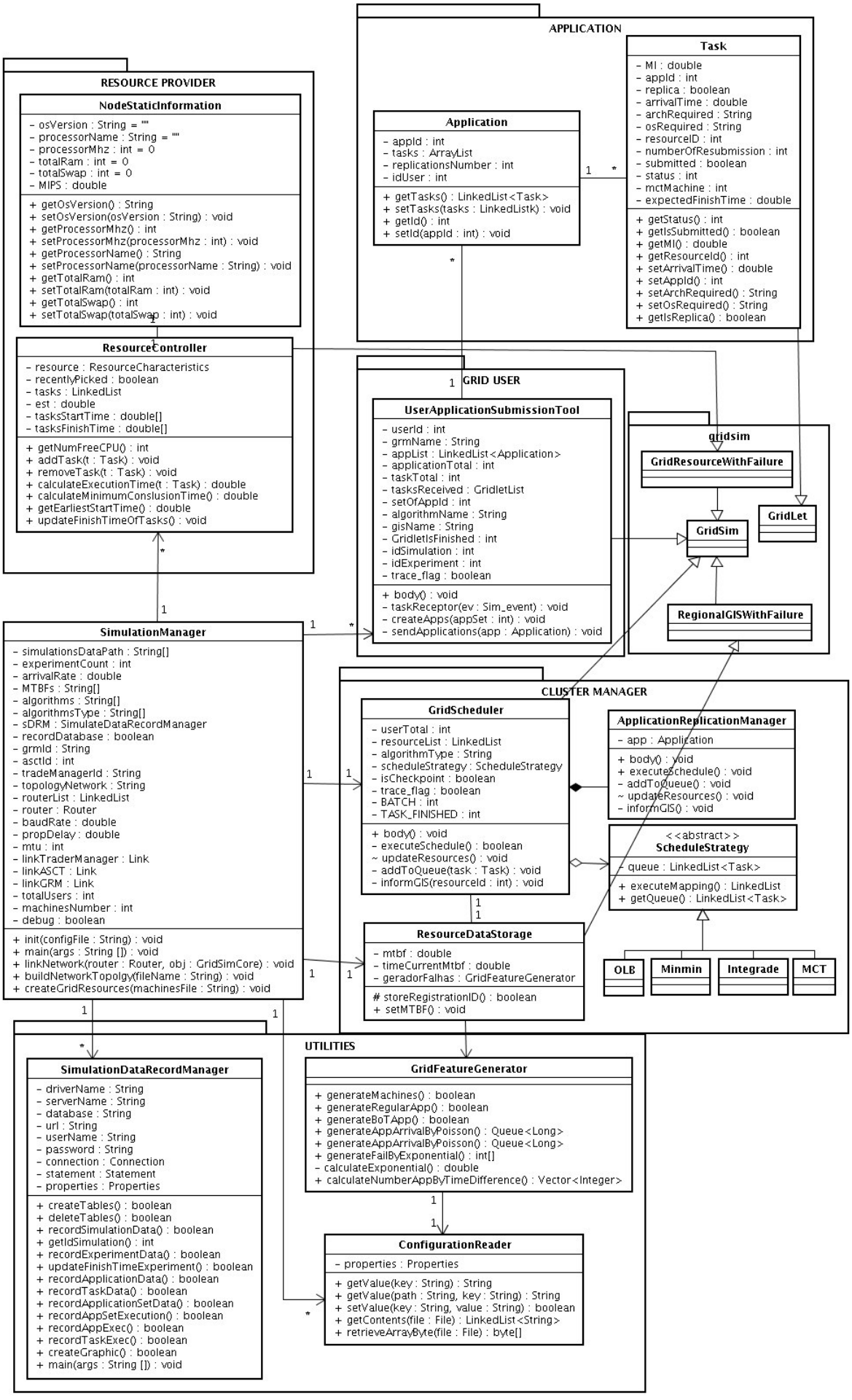

Figura 3.2: Diagrama de classe do OGST

city) e do mais rápido (fastCapacity). A capacidade de processamento de um nó é definido em MIPS (*Millions Instruction Per Second*) em função da avaliação do *benchmark* SPEC (*Standard Performance Evaluation Corporation*) CPU (INT) 2000 [2]. As capacidades de processamento dos nós são geradas de acordo com uma distribuição uniforme, onde os valores gerados irão variar entre o valor especificado da capacidade de processamento do nó da grade mais lento até o mais rápido.

3.4.2 Geração de Aplicações

O GridSim não define explicitamente um modelo de execução de aplicação. Os exemplos fornecidos assumem que todas as tarefas de uma dada aplicação executam em uma única máquina paralela em um tempo de chegada provido pelo usuário. O OGST define explicitamente três modelos de execução: regular, paralela não acoplada e paralela acoplada. Nossa atual implementação permite somente a definição de aplicações regulares e paralelas não acopladas. Aplicações regulares são compostas por uma única tarefa e executam em um único nó. Aplicações paralelas não acopladas são compostas por duas ou mais tarefas, cada uma executando em um nó diferente da grade sem a troca de dados entre si. A versão atual do OGST é voltada para a avaliação do comportamento de aplicações computacionalmente intensivas. Seus recursos ainda não levam em consideração aplicações que trabalham intensivamente sobre dados.

O OGST provê a geração automática de aplicações sintéticas através da classe utilities.GridFeatureGenerator. As aplicações regulares e paralelas não acopladas são geradas a partir dos seguintes métodos:

- generateRegularApp(simulationDataPath, numSetOfApps, numberOfApps, arrivalRate, shortestApp, longestApp): responsável por gerar aplicações regulares, este método recebe como parâmetros de entrada o caminho onde um arquivo texto contendo a descrição das aplicações geradas pelo OGST será armazenado (simulationDataPath), a quantidade de aplicações desejadas (numberOfApps), a taxa de chegada de aplicações por segundo (arrivalRate) e o tamanho da menor e da maior aplicação a serem geradas (shortestApp e longestApp). O tamanho das aplicações geradas é definido em

[2] http://www.spec.org/osg/cpu2000/results

milhões de instruções (MI) e seguem uma distribuição uniforme. Por exemplo, poderiam ser geradas aplicações regulares com a distribuição de $U(3015 \times 10^4, 30150 \times 10^4)$ MI, correspondendo a um tempo de execução entre 5 e 50 horas, aproximadamente no caso de execução em um Pentium 4 com 2.8 GHz.

- `generateBoTApp(simulationDataPath, numSetOfApps, numberOfApps, arrivalRate, numberOfTask, averageTaskLength, varLength)`: responsável por gerar aplicações paralelas não acopladas, este método recebe os quatro primeiros parâmetros do método anterior e, além destes, um valor que define a quantidade de tarefas a ser gerada para cada aplicação (`numOfTask`), o tamanho médio das tarefas (`averageTaskLength`) e um valor que define um percentual de variação do tamanho médio da tarefa provido (`varLength`). Este último parâmetro é utilizado para criar tarefas de uma mesma aplicação com diferentes tamanhos, a fim de simular a heterogeneidade das tarefas. Por exemplo, dado um tamanho médio de $5000MI$ e variação de 50% os tamanhos das tarefas serão gerados de acordo a distribuição uniforme de $U(3750, 6250)MI$ (com 25% para tarefas menores que $5000MI$ e 25% para tarefas maiores que $5000MI$).

O tempo de chegada das aplicações segue uma distribuição de Poisson, uma vez que a sobreposição de um grande número de processos ("aplicações") que se renovam de forma independente é aproximadamente um processo de Poisson, segundo o teorema de Palm-Khintchine [29].

3.4.3 Implementação das Estratégias de Escalonamento

A estratégia de escalonamento a ser utilizada durante a simulação deve estender a classe abstrata `SS`. O OGST fornece uma biblioteca de estratégias de escalonamento, composta atualmente por quatro heurísticas: InteGrade, WQ, MCT e o Min-min [39, 7].

Com o intuito de prover um ambiente para o desenvolvimento e avaliação de estratégias de escalonamento adaptativas, o OGST utiliza o padrão *Strategy* descrito por Gamma et al em [25], conforme apresentado na Figura 3.3. Adotando este padrão, o OGST define uma família de estratégias de escalonamentos, o que permitiria a

troca entre os algoritmos de escalonamento em tempo de execução. No entanto, esta substituição requer a transferência do estado de execução de um algoritmo para o outro, o que ainda não foi implementado no OGST.

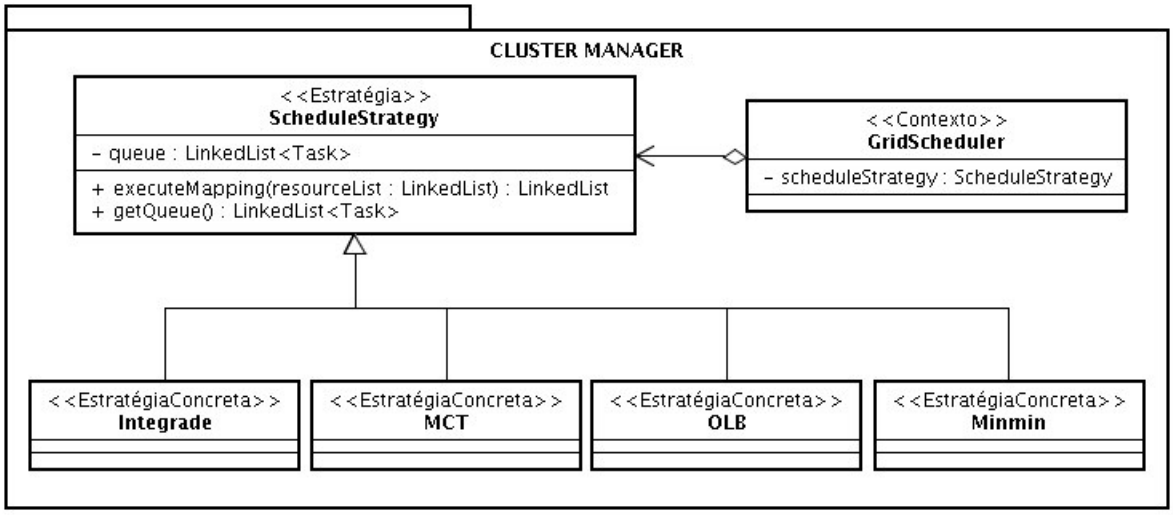

Figura 3.3: Padrão comportamental das estratégias de escalonamento do OGST

Estratégias de escalonamento requerem diversas informações a respeito das aplicações e do estado dos recursos da grade. A classe `Application` do OGST fornece a identificação do usuário que submeteu a aplicação, a quantidade de tarefas que a compõem e a quantidade de réplicas da mesma a serem geradas. A classe `Task` representa as tarefas que compõem uma dada aplicação e armazena seu tamanho especificado em milhões de instruções. A classe `NodeStaticInformation` provê, para cada recurso da grade, sua arquitetura de *hardware* (ex: i686), o nome e versão do sistema operacional, nome e freqüência do processador, tamanho da memória RAM, tamanho da memória *Swap* e a capacidade de processamento da estação de trabalho medida em milhões de instruções por segundo - MIPS. Além disso, a classe `Resource Controller` disponibiliza o tempo de execução previsto para cada tarefa em um dado recurso e o tempo previsto para que um recurso termine a execução de todas as tarefas a ele atribuídas. A partir destas informações, as heurísticas de escalonamento realizam o mapeamento de tarefas a recursos da grade que as executarão. Diferentes estratégias de escalonamento requerem informações distintas. Por exemplo, o MCT escalona as tarefas baseado no tempo de conclusão previsto, calculado pela soma do tempo de execução da tarefa com o tempo para o recurso concluir todas as tarefas que lhe foram atribuídas.

No OGST, os usuários da grade podem definir requisitos (restrições) e preferências de *hardware* e *software* para a seleção dos recursos a serem utilizados na execução das tarefas que compõe a aplicação. Por exemplo, um usuário pode ter como requisito que a aplicação só execute em máquinas com o sistema operacional *Unix*. As preferências especificadas definem a ordem da escolha destas máquinas de acordo com algum critério, por exemplo, pela maior capacidade de processamento (MIPS).

3.4.4 Mecanismo de Injeção de Falhas

Uma falha na grade pode ser percebida como uma interrupção que pára ou mesmo aborta a execução de uma aplicação, possivelmente comprometendo a meta do sistema que pode ser a minimização do tempo de resposta das aplicações. Choi et al [12] classifica as falhas de recursos voluntários em duas classes principais: falhas de instabilidade e falhas de interferência. A primeira classe compreende as falhas de enlaces de rede e de problemas de *hardware* das máquinas. A outra classe é uma conseqüência da natureza dos recursos compartilhados, onde os proprietários das máquinas têm prioridade em relação às computações da grade no acesso aos recursos da máquina.

A arquitetura de injeção de falhas no GridSim assume que a quantidade média de recursos que devem falhar, o instante da falha e o tempo de duração da mesma são geradas antes da simulação iniciar [10] e seguem uma distribuição hiper-exponencial. Uma vez que nosso objetivo é avaliar estratégias de escalonamento considerando condições variáveis do ambiente de execução (tais como diferentes taxas de falhas), modificou-se o comportamento do GridSim para permitir a alteração da definição do tempo médio entre falhas de nós durante a execução da simulação.

A classe `RegionalGISWithFailure` do GridSim, responsável pela criação de falhas e gerenciamento dos recursos da grade, foi estendida pela classe `ResourceDataStorage`, que gera falhas de recursos durante a execução da simulação. A média do intervalo entre as falhas dos recursos é um atributo da classe `ResourceDataStorage` que pode ser alterada a qualquer momento através do método `setMTBF()`. A partir desta média, o OGST gera o tempo da próxima falha seguindo uma distribuição exponencial [33].

A Figura 3.4 ilustra o processo de injeção de falhas de recursos. Uma

vez que os recursos se registram na grade (passos 1 e 2), o `ResourceDataStorage` (RDS) envia o evento `GRIDRESOURCE_FAILURE` (passo 3) para que seja tratado por ele mesmo depois de transcorrido o tempo especificado para a ocorrência da falha de algum recurso. Quando o evento `GRIDRESOURCE_FAILURE` é recebido pelo RDS, este escolhe de maneira aleatória o recurso que vai falhar e envia o evento de falha para que o recurso se torne indisponível na grade (passo 4). Em seguida, o RDS gera o evento `GRIDRESOURCE_RECOVERY` e envia este evento para ele próprio (passo 5). Este evento determina o tempo que o recurso que falhou permanecerá indisponível até que seja recuperado pelo RDS de acordo com uma distribuição uniforme. Em seguida, o RDS gera um novo evento `GRIDRESOURCE_FAILURE` especificando o tempo para a ocorrência da próxima falha (passo 6). Eventualmente, este tempo pode está relacionado a uma nova média do intervalo entre falhas de recursos.

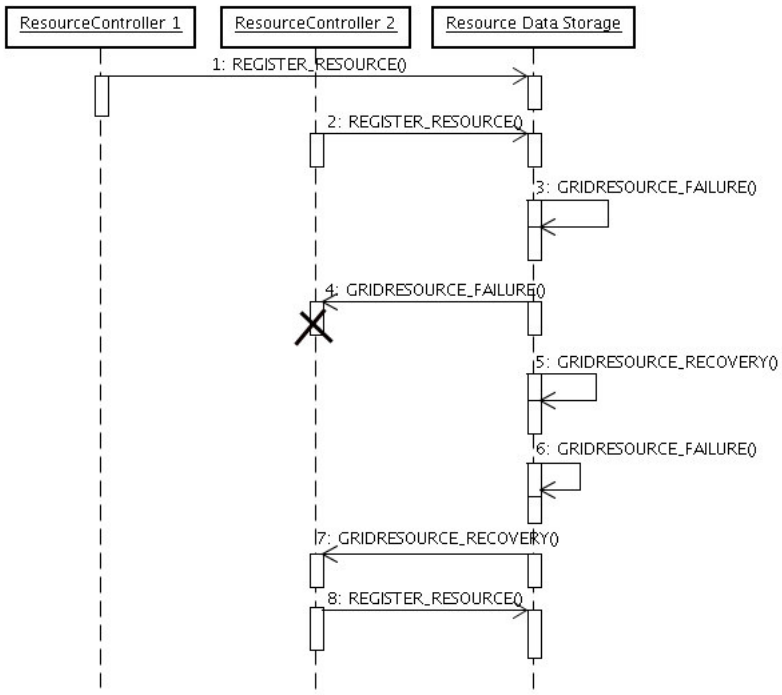

Figura 3.4: Diagrama de seqüência de injeção de falhas

Após esperar o tempo que o recurso deve ficar indisponível, o RDS deve tratar o evento de recuperação do recurso enviando para este uma notificação para que se registre novamente na grade (passo 7). Ao receber esta notificação, o recurso da grade (`ResourceController`) deve se registrar novamente junto ao RDS (passo 8), tornando-o disponível para executar aplicações.

3.4.5 Mecanismos de Tolerância a Falhas

O gerente de recursos da grade (RDS) identifica que um recurso falhou de duas maneiras: quando o recurso não responde a uma mensagem de *polling* que é periodicamente enviada ao mesmo; quando ele é informado pelo UAST sobre a falha de um recurso, dado que a falha na execução de uma tarefa gera uma notificação ao UAST.

O GridSim permite a simulação do reinício automático de tarefas que falham, fazendo com que elas sejam executadas novamente desde o início. O OGST estende esta característica provendo novos mecanismos de recuperação de aplicações em caso de falha: o uso de *checkpointing* e a replicação, técnicas comumente usadas em grades oportunistas.

A Figura 3.5 ilustra o mecanismo de *checkpointing* do OGST. O mecanismo de *checkpointing* simulado é baseado em um armazém estável globalmente disponível, permitindo um rápido reinício de tarefas em caso de falhas de nós. Quando um recurso da grade falha (passo 1), todas as tarefas que estavam esperando para serem executadas bem como as que estavam em execução são enviadas de volta para o `User Application Submission Tool` (passo 2). Em seguida, o UAST encaminha estas tarefas (pertencentes à classe `Task`) para o `GridScheduler` (passo 3). A classe `Task` armazena o estado de execução atual da tarefa contendo a quantidade de instruções já executadas e quantidade total de instruções que compõem a tarefa. O `GridScheduler` leva em consideração apenas a quantidade restante de instruções da tarefa a serem executadas no processo de escalonamento, enviando a tarefa para um novo recurso (passo 4).

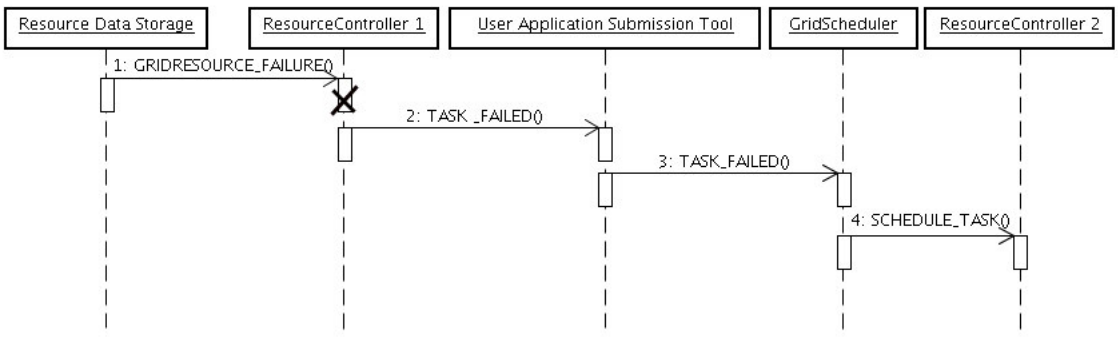

Figura 3.5: Diagrama de seqüência do mecanismo de *checkpointing*

A replicação é outro mecanismo de tolerância a falhas bem conhecido e desenvolvido no OGST. A Figura 3.6 ilustra as interações para a criação e gerenciamento das réplicas, sem considerar a etapa de registro dos recursos. As interações iniciam com a submissão de uma aplicação feita pelo usuário da grade (`UAST`) (passo 1), onde o escalonador (`GridScheduler`) irá criar a quantidade de réplicas pré-definida para a aplicação (passo 2). Em seguida, o `GridScheduler` irá realizar o mapeamento das tarefas encaminhando-o para o `Application Replication Manager` (`ARM`) (passo 3) que, por sua vez, irá enviar as réplicas para os recursos e passará a monitorar suas execuções (passos 5 e 6). O `ARM` é informado sempre que uma tarefa falha (passo 7). Caso todas as réplicas falhem, ele informa ao UAST responsável pela submissão da aplicação que não foi possível concluir a tarefa (passo 7). Uma vez finalizada uma tarefa, o `ARM` irá cancelar a execução de todas as outras réplicas restantes (passo 9) e comunicará tanto ao `UAST` quanto ao `GridScheduler` que a tarefa finalizou (passos 10 e 11).

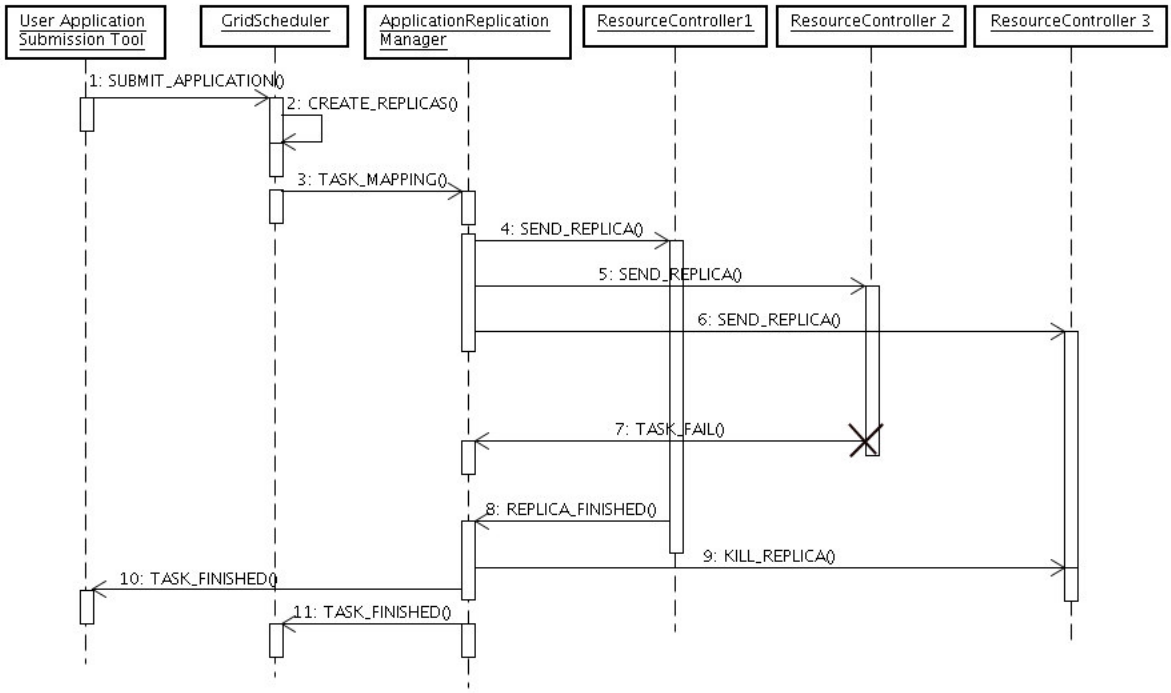

Figura 3.6: Diagrama de seqüência do mecanismo de replicação

3.4.6 Gerenciamento dos Dados das Simulações

O GridSim pode coletar dados estatísticos relativos à execução de uma simulação que são armazenados em um arquivo texto. A partir deste arquivo, pode-

se gerar um relatório de execução. No entanto, consultar e procurar relacionamentos entre dados em uma grande quantidade de dados estatísticos através da manipulação de um arquivo texto pode se tornar uma árdua tarefa. Para simplificar a consulta e análise de uma grande quantidade de dados gerados, o componente SDRM do OGST utiliza um banco de dados relacional.

O OGST define o modelo de dados ilustrado na Figura 3.7. A entidade `Simulation` é responsável por armazenar a definição dos recursos da grade, o intervalo entre falhas e mecanismo de tolerância a falhas utilizados na simulação. Cada simulação executa um dado experimento (`Experiment`). Cada experimento emprega uma estratégia de escalonamento e executa um conjunto de aplicações (`ApplicationSet`), formado por uma coleção de aplicações (`Application`). De acordo com o modelo de aplicação utilizado (regular ou paralela não acoplada), uma aplicação pode possuir uma ou mais tarefas (`Task`). Os dados da execução simulada de cada conjunto de aplicações, aplicações e tarefas são armanezados nas tabelas (`AppSetExecution`), (`AppExecution`) e (`TaskExecution`), respectivamente.

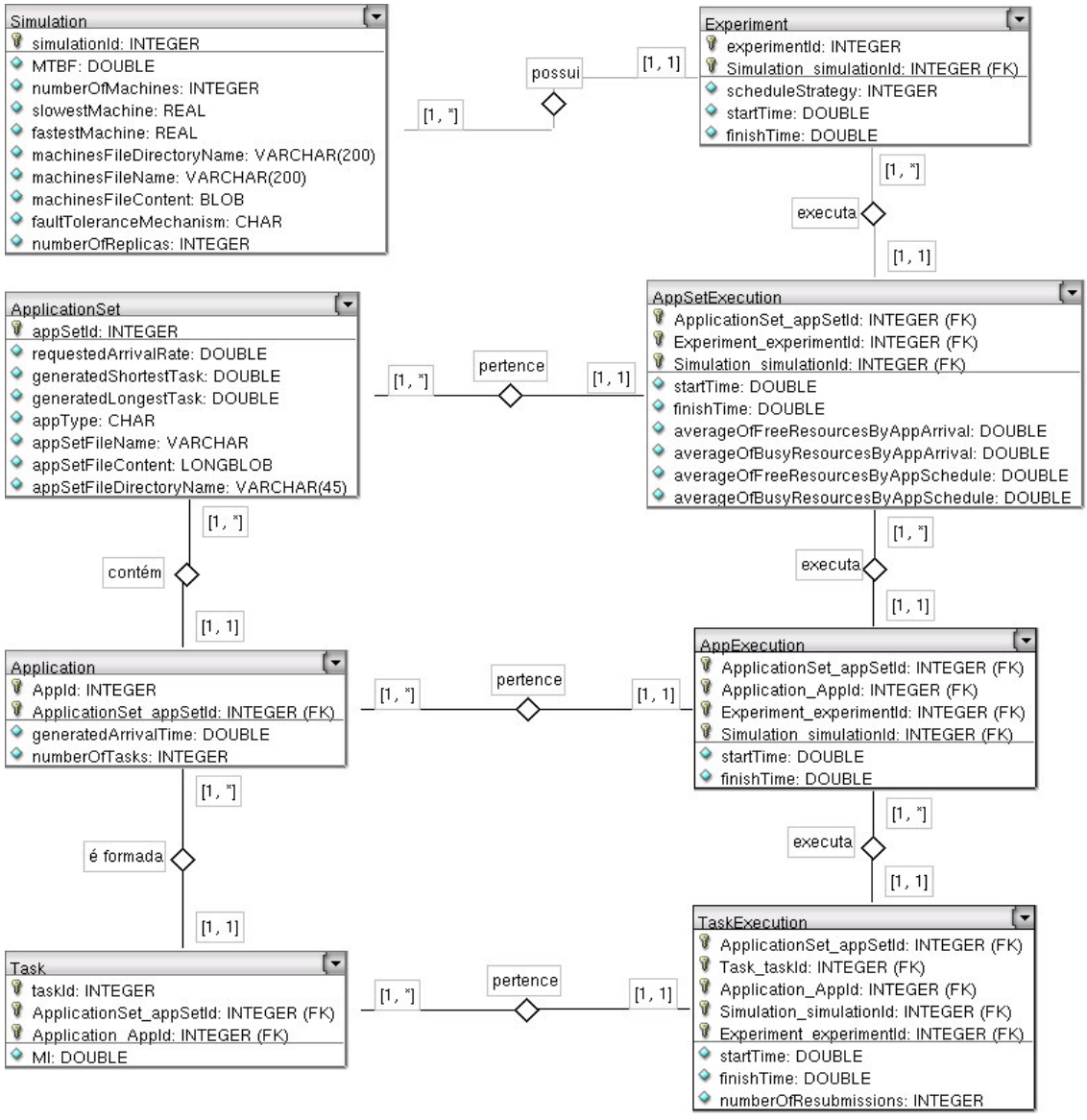

Figura 3.7: Modelo de dados do OGST

O SDRM também permite a geração automática de gráficos de dados coletados através da execução de simulações, tais como o tempo médio de conclusão das aplicações em função do tempo médio entre falhas de nós (Figura 3.8) ou em função das taxas de chegada das aplicações (Figura 3.9). Além disso, o OGST permite a geração de um gráfico de três dimensões que relacione a diferença do tempo médio de conclusão das aplicações obtido por duas estratégias de escalonamento diferentes, o tempo médio entre falhas de nós e a taxa de chegada das aplicações (Figura 3.10). Estes gráficos são gerados usando a ferramenta *gnuplot* [3].

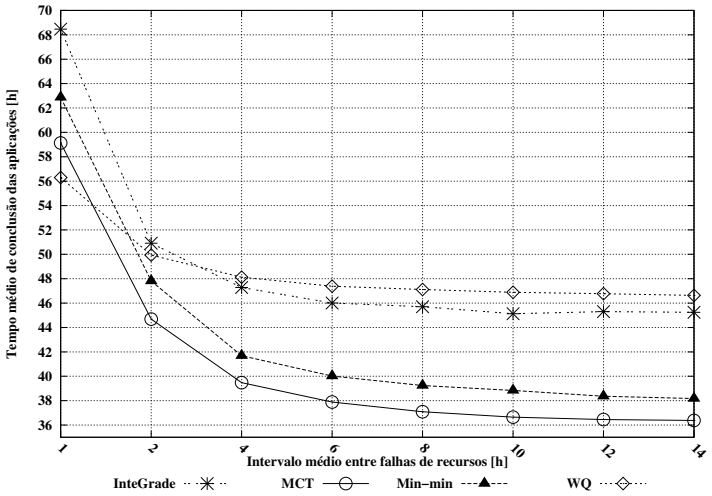

Figura 3.8: Exemplo de gráfico do tempo médio de conclusão das aplicações em função do tempo médio entre falhas de nós.

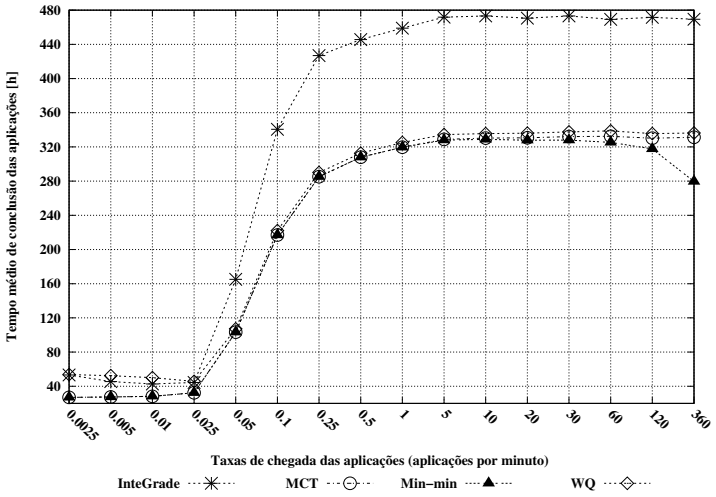

Figura 3.9: Exemplo de gráfico do tempo médio de conclusão das aplicações em função da taxa de chegada das aplicações por minuto.

[3]Página Inicial: `http://www.gnuplot.info`

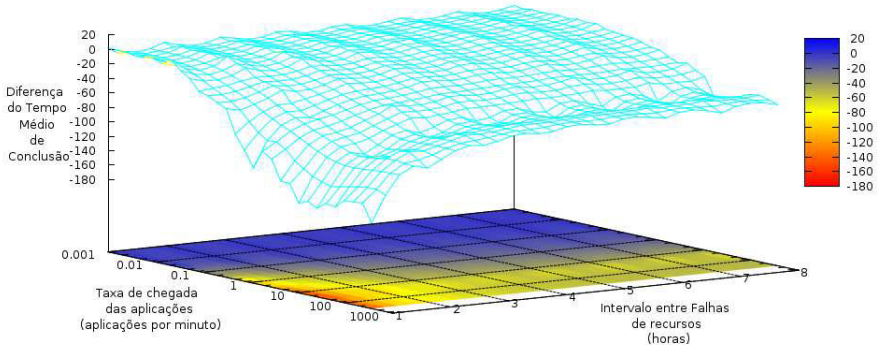

Figura 3.10: Exemplo de gráfico que relaciona a diferença do tempo médio de conclusão das aplicações obtido por duas estratégias de escalonamento diferentes, o tempo médio entre falhas de nós e a taxa de chegada das aplicações.

3.4.7 Variação na Carga Local dos Provedores de Recursos

Uma vez que uma grade oportunista faz uso de máquinas não dedicadas, seu ambiente de execução usualmente exibe uma variação na disponibilidade de cada nó da grade, considerando diferentes períodos de uso. Para simular este comportamento habitual, o GridSim permite a definição de carga de trabalho local para cada recurso de acordo com um calendário e um valor numérico que ajuda a definir a carga local do recurso (no horário de pico, fora do horário de pico, finais de semana e feriados).

A classe `ResourceCalendar` do GridSim implementa um mecanismo que permite a modelagem da carga local nos recursos. Para isso, deve-se criar um calendário para cada recurso que fará parte da grade a partir do seguinte construtor: `ResourceCalendar(timeZone, peakLoad, offPeakLoad, relativeHolidayLoad, weekendList, holidayList, seed)`. Os parâmetros definem respectivamente: a zona de tempo do recurso (`timeZone`) de acordo com o padrão GMT (`Greenwich Mean Time`), uma vez que recursos podem está em domínios administrativos com diferentes fusos horários; um fator de carga local no horário de pico (`peakLoad`) e fora do horário de pico (`offPeakLoad`) para os dias da semana; outro fator de carga local para fins de semana e feriados (`relativeHolidayLoad`); e duas listas que incluem dias de final de semana (sábado e domingo) e feriados. Os fatores da carga local (`peakLoad` e `offPeakLoad`) variam em uma faixa de 0 a 1. Quanto menor o valor, mais livre estará o recurso.

3.5 Conclusões

Este capítulo apresentou o OGST, cujos requisitos, arquitetura e implementação levaram em consideração características das grades oportunistas. O OGST apresenta-se como uma ferramenta auxiliar ao desenvolvimento de sistemas de *middleware* para grades. Ele permite a validação de novos conceitos e implementações considerando diferentes condições e cenários do ambiente de execução através da disponibilização de recursos como:

- Um gerador automático de ambientes de grades que exibem alta heterogeneidade de máquinas;
- Um gerador automático de aplicações heterogêneas, tanto regulares quanto paralelas não acopladas;
- Submissão das aplicações dinamicamente de acordo com a distribuição de poisson;
- Injeção de falhas de nós e recuperação dos mesmos, de acordo com diferentes distribuições;
- Suporte para simulação dos seguintes mecanismos de tolerância a falhas: reinício, *checkpointing* e replicação [16];
- Disponibilização de uma biblioteca de algoritmos de escalonamento atualmente composta pelos algoritmos InteGrade, MCT, Min-min e WQ;
- Armazenamento de dados coletados durante a execução da simulação e geração automática de gráficos a partir dos mesmos.

4 Avaliação do OGST

Ferramentas de simulação são cada vez mais utilizadas na solução de problemas e na tomada de decisões. Os desenvolvedores e usuários de tais ferramentas, que utilizam informações obtidas a partir dos resultados alcançados pelas simulações, estão sempre interessados se o modelo e seus resultados estão corretos. Desta forma, uma preocupação fundamental para um simulador de grades oportunistas, como o OGST, é o de avaliar seu grau de precisão, tanto com relação ao seu funcionamento, quanto dos resultados obtidos através de simulações em comparação com um sistema real.

Neste capítulo descreveremos abordagens existentes para avaliação de simuladores e apresentamos os resultados da avaliação do simulador OGST.

4.1 Avaliação de Simulações

Feinstein e Cannon em [21] apresentam três abordagens principais na construção da avaliação de ferramentas de simulação:

1. **Fidelidade:** a fidelidade é o nível de realismo que uma simulação apresenta para o usuário do simulador, definindo o grau de similariedade entre o ambiente do simulador e o ambiente real que está sendo simulado [28]. A fidelidade de um simulador se foca no equipamento que é usado para simular um ambiente de aprendizado particular, em termos de suas características físicas e funcionais. Por exemplo, na simulação de uma cabine de um piloto de avião, os elevados níveis de ruídos característicos destes ambientes deveriam apresentar uma boa fidelidade, a fim de avaliar um ambiente real no qual os pilotos devem manter comunicação com a torre de controle. Além disso, o sistema de navegação simulado do avião deveria fornecer informações similares a um sistema real de navegação. O termo fidelidade tem sido mais freqüentemente utilizado para o projeto de simuladores que são usados em treinamentos de pessoas [21];

2. **Verificação:** a verificação avalia se um modelo de simulação está funcionando

como previsto. O processo de verificação envolve depurar o modelo pelo isolamento e eliminação de tantos erros quanto possível. Isto pode ser feito através de:

- Uso de depuradores internos do *software* da simulação, avaliando passo a passo a execução de uma simulação;
- Visualização de relatórios de saída;

Muitos erros são simples problemas de depuração de sistemas, outros envolvem consertar erros de projeto, onde os componentes interagem de maneira imprevisível ou retornam respostas erradas para valores extremos;

3. **Validação:** a validação é o processo de avaliação que verifica se as conclusões obtidas a partir de uma simulação são semelhantes aos resultados obtidos no sistema real que está sendo simulado. Os resultados obtidos nas simulações geralmente referem-se às métricas avaliadas pelo sistema. Por exemplo, o tempo médio de execução e de conclusão das aplicações e a taxa de utilização de recursos na grade. Com o intuito de validar o modelo de simulação, os resultados do simulador podem ser comparados com resultados de sistemas reais ou de outros modelos de simulação válidos. Por exemplo, em [19] o simulador de estudo de escalonamento em grades (chamado GangSim) compara a proximidade do valor do tempo médio de resposta das aplicações obtido pelo simulador e por uma grade real conhecida como Grid3 [50].

Além destas três abordagens para avaliação de ferramentas de simulação, Taufer et al em [49] sugere que as funcionalidades de um simulador e de um ambiente real sejam mapeadas e comparadas entre si, a fim de mostrar a proximidade entre o comportamento de ambos. Taufer faz um mapeamento e comparação das funcionalidades de um simulador de projetos de computação voluntária (chamado SimBA) com o *middleware* BOINC, uma plataforma de *software* para computação de aplicações usando recursos voluntários. O funcionamento do SimBA e do BOINC são comparados em alto nível mostrando semelhanças e diferenças nos processos de criação, replicação, armazenamento e escalonamento das aplicações.

Escolhemos utilizar a abordagem sugerida por Taufer et al e a abordagem de validação para avaliar o OGST. Seguindo a primeira abordagem, avaliamos se o

modelo de simulação do OGST está funcionando como previsto, analisando a funcionalidade de cada componente da arquitetura do OGST, comparando-os com os componentes da arquitetura do middleware InteGrade. Além disso, a fim de validar as interações entre os componentes do OGST para a execução de aplicações, comparamos as ocorrências de eventos do modelo simulado com os eventos do sistema InteGrade. A abordagem de validação teve como objetivo mostrar a proximidade dos resultados obtidos pelo OGST e um *middleware* de grade real (InteGrade). A escolha do InteGrade para a comparação com o OGST foi motivada pela semelhança entre o protocolo de execução de aplicações do InteGrade e o protocolo de execução de aplicações apresentado na Seção 2.1, utilizada como base para este trabalho.

4.2 Comparação da Arquitetura do InteGrade com o OGST

A arquitetura de aglomerados do InteGrade, descrita na Subseção 2.5.4, é formada por vários componentes de *software*. Eles executam nos nós constantes no aglomerado de forma a prover as funcionalidades necessárias à submissão de aplicações, seleção de recursos e execução de aplicações. Os principais componentes quanto ao protocolo de execução de aplicações são descritos a seguir:

- GRM (*Global Resource Manager*): gerencia os recursos computacionais constantes de um aglomerado de máquinas conectadas por uma rede local e provê mecanismos de escalonamento baseado no conhecimento que ele possui sobre a disponibilidade dos recursos e os requisitos especificados pelo usuário no processo de submissão para execução de aplicações. O armazenamento e consulta das informações de estado dos recursos (LRMs) são realizados pelo serviço de negociação (*Trading*) definido em CORBA [42].

- AR (*Application Repository*): armazena binários de aplicações submetidas por usuários para execução na grade. Toda aplicação submetida à execução deve ser primeiramente registrada neste repositório.

- LRM (*Local Resource Manager*): é executado em cada nó que fornece recursos ao aglomerado. O LRM coleta informações sobre o estado do nó como memória, CPU, disco e utilização da rede, enviando estas informações periodicamente ao GRM de seu aglomerado;

- EM (*Execution Manager*): mantém um banco de dados com todas as aplicações que foram submetidas para a execução no aglomerado, incluindo a localização de cada processo da aplicação e o estado da requisição de execução.

- ASCT (*Application Submission and Control Tool*): ferramenta para submissão de aplicações para execução na grade. Os usuários podem especificar pré-requisitos de execução, como a plataforma de *hardware* e *software* nas quais a aplicação deve ser executada, requisitos de memória necessários à execução da aplicação, entre outros. Também é possível monitorar o progresso da aplicação em execução.

A maior parte das funcionalidades dos componentes do InteGrade foram implementadas no OGST. A Tabela 4.1 apresenta um mapeamento das funcionalidades apresentadas pelo InteGrade e implementadas pelo OGST.

Componentes do InteGrade	OGST
GRM.	Funcionalidades implementadas pelo componente `GridScheduler` (GS).
Trading	Funcionalidades implementadas pelo componente `Resource Data Storage` (RDS).
LRM.	Funcionalidades implementadas pelo componente `Resource Controller` (RC).
ASCT.	Funcionalidades implementadas pelo componente `User Application Submission Tool` (UAST).
EM.	O estado da requisição de execução da tarefa e local de execução é mantido em memória pelas `Tasks`. O `Simulation Data Record Manager` (SDRM) armazena o tempo que as tarefas levam para ser executadas.
AR.	Uma vez que as aplicações simuladas não possuem binários, não houve necessidade de implementar as funcionalidades deste componente.

Tabela 4.1: Funcionalidades dos componentes do InteGrade implementadas no OGST.

4.2.1 Comparação entre o Protocolo de Execução de Aplicações do InteGrade e do OGST

A Figura 4.1 ilustra os passos do protocolo de execução tanto do InteGrade quanto do OGST, com a finalidade de mostrar as interações comuns e distintas entre os componentes do *middleware* real e da ferramenta de simulação. Nos objetos da Figura 4.1, colocamos entre parênteses os componentes do OGST que implementam as funcionalidades do InteGrade. No InteGrade, assim que o usuário registra sua aplicação no Repositório de Aplicações através do ASCT (passo 1), ele pode solicitar a execução da aplicação (passo 2). O usuário pode, opcionalmente, especificar requisitos indispensáveis da aplicação como, por exemplo, a arquitetura para a qual a aplicação foi compilada. Ele pode ainda definir as preferências para a execução da aplicação como, por exemplo, os recursos que possuem a maior quantidade de memória RAM disponível.

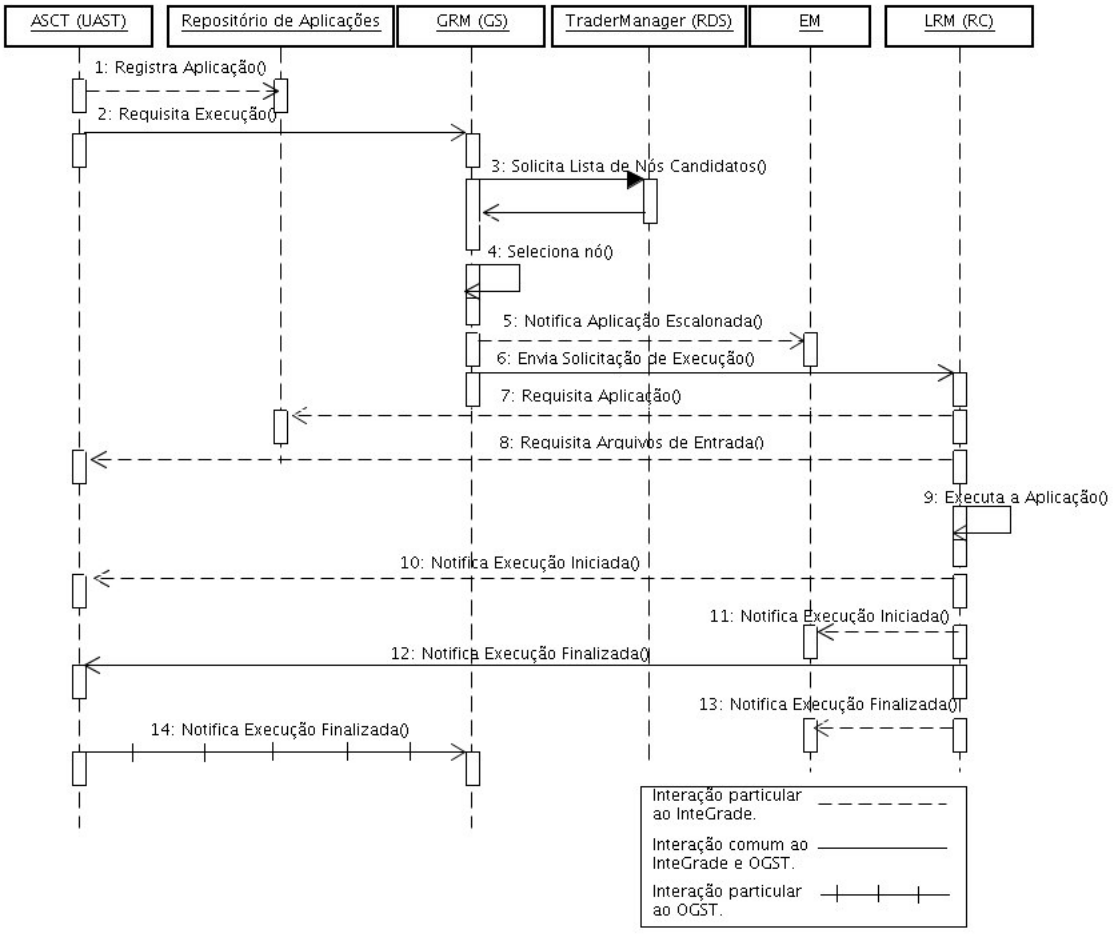

Figura 4.1: Protocolo de execução do InteGrade

Assim que a solicitação de execução é enviada ao GRM, este solicita e espera por uma lista de nós candidatos fornecida pelo serviço de negociação *Trading* (TraderManager) (passo 3), de acordo com os requisitos e preferências do usuário. A partir desta lista, o GRM procura um nó para executar a aplicação (passo 4) e, caso não encontre nenhum nó disponível que possa atender a requisição, será gerado uma notificação ao ASCT informando ao usuário que seu pedido de execução não pode ser atendido. Por outro lado, caso encontre um nó que atenda os requisitos da aplicação, será gerada uma notificação ao EM informando que a aplicação foi escalonada (passo 5) e, em seguida, o pedido de execução da aplicação será encaminhado para o LRM da máquina selecionada (passo 6).

Ao receber a solicitação de execução da aplicação, o LRM requisita o binário da mesma junto ao Repositório de Aplicações (passo 7) bem como os arquivos de entrada da aplicação junto ao ASCT requisitante (passo 8). Em seguida, ele inicia a execução da aplicação (passo 9), notificando o ASCT e o EM que a requisição foi atendida e, posteriormente, finalizada (passos de 10 a 13). O ASCT pode, eventualmente, cancelá-la remotamente.

Na Figura 4.1, as linhas tracejada, contínua e cruzada simbolizam, respectivamente, os passos exclusivos do protocolo de execução do InteGrade, os passos comuns tanto ao InteGrade quanto ao OGST e os passos exclusivos do OGST. Como pode ser observado, o protocolo para execução de aplicação no OGST está bem próximo do protocolo utilizado no InteGrade, omitindo apenas o repositório de aplicações (passos 1 e 7) e o gerente de execução das aplicações (EM) (passos 5, 11 e 13). O OGST possui apenas uma interação adicional, na qual o UAST notifica ao GS que uma tarefa finalizou (passo 14), uma vez que algumas heurísticas de escalonamento (ex: WQ) requerem, quando todos os recursos da grade estão ocupados, a informação de quando recursos tornam-se disponíveis a fim de escalonar uma nova tarefa.

4.3 Experimentos de Validação

Realizamos experimentos de validação com o objetivo de verificar se as conclusões obtidas a partir de uma simulação no OGST são semelhantes aos resultados obtidos no sistema real que está sendo simulado (o InteGrade). A métrica utilizada

para comparar os experimentos simulados e reais foi o tempo médio de execução de um conjunto de aplicações, escalonadas através do algoritmo de escalonamento do InteGrade. Sargent em [45] sugere que a validação de um simulador requer pelo menos dois diferentes conjuntos de condições experimentais. Consideramos a execução de dois conjuntos de aplicações: regulares e paralelas não acopladas.

No experimento real utilizando o InteGrade, o ambiente de grade utilizado foi composto por um conjunto de seis máquinas, conforme apresentado na Tabela 4.2. Na máquina **portinari** foram executados dois componentes do *middleware* InteGrade: o GRM e o ASCT. As cinco máquinas restantes funcionaram como provedores de recursos, executando o componente LRM.

Máquina	Processador	Memória	Kernel Linux	MIPS
portinari	Intel Pentium(R) D 2.80GHz	2 GB	2.6.20-17-generic	2487
debret	Intel Core(TM)2 Duo 2.40GHz	2 GB	2.6.24-21-generic	4099
frida	Intel Core(TM)2 Duo 2.40GHz	2 GB	2.6.24-21-generic	4102
gauguin	Intel(R) Core(TM)2 CPU 1.86GHz	3 GB	2.6.24-21-generic	3193
monet	Intel(R) Core(TM)2 CPU 1.86GHz	3 GB	2.6.24-21-generic	3190
cezanne	Intel(R) Core(TM)2 CPU 1.86GHz	2 GB	2.6.24-21-generic	3187

Tabela 4.2: Especificação dos recursos da grade.

Modelamos no OGST as mesmas seis máquinas que foram utilizadas no ambiente do InteGrade. A capacidade de processamento de cada máquina foi definida como a quantidade de milhões de instruções por segundo (MIPS) que uma CPU é capaz de processar. A capacidade em termos de MIPS para cada máquina foi estimada através do *benchmark* TSCP [1], como apresentado na coluna "MIPS" da Tabela 4.2.

A aplicação utilizada em nossos experimentos calcula os números da série de Fibonacci ($F(n)$), uma seqüência definida como recursiva conforme a fórmula abaixo:

$$F(n) = \begin{cases} 0 & \text{se } n = 0; \\ 1 & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

A partir desta aplicação, realizamos dois tipos de experimentos no Inte-

[1] `http://www.tckerrigan.com/Chess/TSCP`

Grade:

1. Execução de dez aplicações regulares: o argumento de entrada de cada aplicação corresponde a posição (n) de um termo da série de Fibonacci. Os argumentos das aplicações variaram entre $48, 49$ e 50, que correspondem respectivamente aos tempos de execução 95, 156 e 255 segundos, quando a aplicação é executada na máquina **frida**.

2. Execução de dez aplicações paralelas não acopladas: cada aplicação paralela não acoplada foi formada por cinco tarefas. Os argumentos das tarefas variaram entre $48, 49$ e 50.

No OGST, o tamanho de cada tarefa foi modelado em termos de milhões de instruções (MI). Para obter este valor utilizamos a máquina **frida**, calculando o produto do tempo que a tarefa demorou para executar (em segundos) com a capacidade de processamento desta máquina que é de 4102 MIPS. Por exemplo, a tarefa com o argumento de entrada 48 demorou 95 segundos para executar em **frida**. Logo, o tamanho da tarefa é de 389690 MI. Na Tabela 4.3, apresentamos os argumentos e tempo de execução das tarefas relacionados a quantidade de milhões de instruções.

Argumento	Tempo de Execução (s) (na máquina frida)	Milhões de Instruções (MI)
48	95	389690
49	156	639912
50	255	1046010

Tabela 4.3: Tarefas utilizadas nos experimentos.

As aplicações dos experimentos foram executadas no *middleware* InteGrade e no OGST em provedores de recursos livres das cargas locais dos usuários.

Tanto os experimentos reais no InteGrade quanto os experimentos simulados através do OGST foram executados 30 vezes. Para cada experimento, medimos o tempo de execução por aplicação e calculamos o tempo médio de execução das 10 aplicações de cada experimento. Então, com base nos tempos médios de execução das aplicações, considerando os 30 experimentos, calculamos os seguintes dados estatísticos: a média, o valor mínimo, o valor máximo, o desvio padrão, a variância e o intervalo de confiança de 95%.

A Tabela 4.4 apresenta os dados estatísticos obtidos para as aplicações regulares e paralelas não acopladas, tanto no ambiente real quanto no simulador. Como podemos observar, as diferenças percentuais entre os tempos médios de execução do ambiente real e do OGST são pequenas tanto para as aplicações regulares (diferença de 13.92%) quanto para as aplicações paralelas não acopladas (diferença de 16.66%). As pequenas variações de desvio padrão e intervalos de 95% de confiança apertados ratificam esta conclusão. Atribuímos as diferenças entre as medidas do ambiente real e do ambiente simulado a dois fatores:

1. Para cada experimento realizado, a ordem de escolha dos recursos para executar as aplicações muda aleatoriamente no algoritmo InteGrade, visto que não foram utilizados requisitos ou preferências de usuários na seleção de recursos;

2. O simulador não levou em conta a execução de processos do sistema operacional dos provedores de recursos, que estavam presentes nos LRMs que fizeram parte dos experimentos reais do InteGrade. Estes processos influem na variação dos tempos das execuções das tarefas que foram obtidos pelos recursos computacionais reais.

	Aplicações			
	Regulares		Paralelas não Acopladas	
	Real	Simulador	Real	Simulador
Média	4.001	3.512	6.041	5.178
Valor Mínimo	3.799	3.458	5.767	5.021
Valor Máximo	4.150	3.553	6.239	5.265
Variância	0.011	0.0009	0.023	0.006
Desvio Padrão	0.109	0.030	0.154	0.077
Intervalo de Confiança	[3.962, 4.040]	[3.501, 3.523]	[5.983, 6.099]	[5.150, 5.206]

Tabela 4.4: Resultado dos experimentos para as aplicações regulares e paralelas não acopladas. Valores em minutos.

4.4 Conclusões

Este capítulo apresentou a avaliação do simulador OGST. Foram descritas as principais formas de avaliação de ferramentas de simulação: fidelidade, verificação,

validação e mapeamento de funcionalidades. Com base no mapeamento de funcionalidades, apresentamos importantes comparações entre os componentes do simulador OGST e do *middleware* InteGrade. Descrevemos os procedimentos comuns e distintos a ambos os sistemas e mostramos a preocupação do OGST em representar os principais componentes e protocolos de uma grade oportunista.

Utilizamos ainda o processo de validação, realizando experimentos em um ambiente real do InteGrade e executando suas correspondentes simulações no OGST. Foram definidos conjuntos de aplicações tanto regulares quanto paralelas não acopladas, avaliando-se dados estatísticos sobre o tempo médio de execução das aplicações. Através da comparação entre os resultados obtidos pelo ambiente de execução do InteGrade e pelo OGST pudemos concluir que os resultados obtidos através do simulador são próximos aos que são obtidos em um ambiente real, tendo-se observado uma diferença entre o OGST e o InteGrade de apenas 13.92% em relação à média do tempo de execução das aplicações regulares e de 16.66% em relação à média do tempo de execução das aplicações paralelas não acopladas.

5 Avaliação de Estratégias de Escalonamento em Grades Oportunistas

Escalonamento em grades oportunistas é um problema desafiador devido a diversos fatores, tais como a existência de um ambiente de recursos dinâmico e não pré-determinado, alta heterogeneidade de nós e da interconexão entre eles, alta escalabilidade e um ambiente não controlável composto de nós computacionais não dedicados. Estas características justificam a importância de se investigar estratégias de escalonamento eficientes de aplicações que levem em consideração aspectos como tolerância a falhas de recursos, migração de tarefas, gerenciamento de informação dos recursos da grade, balanceamento de carga dos recursos, escalonamento adaptativo e reescalonamento de aplicações.

Este capítulo descreve as simulações que executamos usando o OGST visando avaliar diversas heurísticas de escalonamento (InteGrade, MCT, Min-min e WQ) sob diferentes condições do ambiente de execução. A escolha dos algoritmos MCT, Min-min e WQ foi baseada nos trabalhos Maheswaran et al em [39] e Braun [7]. MCT é uma estratégia de escalonamento representante do modo *on-line* que faz uso de informações sobre o tempo de execução das tarefas e a capacidade dos recursos computacionais. Maheswaran et al em [39] sugere que o MCT seja utilizado como um *benchmark* na avaliação de heurísticas de escalonamento de aplicações em sistemas distribuídos heterogêneos, enquanto em estudo apresentado por Braun em [7], o MCT foi a estratégia de escalonamento do tipo *on-line* que apresentou o menor *makespan* [1]. O Min-min é representante do modo de escalonamento *batch* e também faz uso de informações sobre o tempo de execução das tarefas e capacidade dos recursos computacionais que compõem o ambiente de execução. Em [7], esta estratégia apresentou o segundo menor *makespan* entre os algoritmos do tipo *batch*. Apesar do algoritmo genético GA ter apresentado menor *makespan*, ele não foi selecionado por demandar muito tempo de processamento em comparação com os demais algoritmos

[1] Diferença entre o instante de tempo em que a primeira tarefa da aplicação inicia sua execução e o instante de tempo no qual a última tarefa da mesma termina sua execução.

do tipo *batch* avaliados no estudo citado. Finalmente, o *Work Queue* (WQ), uma heurística *on-line*, foi selecionada por não requerer informações sobre o tempo de execução das tarefas nem sobre a capacidade dos recursos computacionais, enquanto a estratégia do InteGrade foi uma escolha natural, dado que este trabalho está inserido no contexto do desenvolvimento deste *middleware* de grade oportunista.

5.1 Simulações Realizadas

As avaliações de desempenho das estratégias de escalonamento foram realizadas através da execução de diversas simulações, considerando diferentes cenários do ambiente de execução da grade. Esta seção descreve as simulações realizadas e a análise de seus resultados.

5.1.1 Avaliação das Estratégias de Escalonamento em um Ambiente de Grade com Variação das Taxas de Chegada das Aplicações e sem Falhas de Recursos

O objetivo das primeiras simulações foi avaliar o desempenho dos algoritmos de escalonamento do InteGrade, MCT, Min-min e WQ, considerando uma variação das taxas de chegada das aplicações (baixa, média e alta) e assumindo que as máquinas da grade se mantinham disponíveis. A análise de desempenho das estratégias de escalonamento foi feita considerando o intuito de minimizar o tempo médio de conclusão das aplicações.

Foram simulados três ambientes de grade, compostos respectivamente por: 100, 200 e 400 máquinas. Nestes ambientes, o poder de processamento médio foi equivalente a um Pentium 4 com 2.8 GHz (1000 MIPS), considerado como um valor representativo para computadores pessoais usuais. A fim de levar em consideração a heterogeneidade do ambiente, os nós da grade foram gerados de acordo com uma distribuição uniforme $U(222, 1776)$ MIPS, onde o poder de processamento da máquina mais rápida é 8 vezes maior que o poder de processamento da máquina mais lenta.

Simulamos a execução de 1500 aplicações regulares, geradas sinteticamente com uma variação de tamanho (em termos de milhões de instruções) gerada através de

uma distribuição uniforme $U(3015 \times 10^4, 30150 \times 10^4)$ MI, com um tempo de execução entre 5 e 50 horas aproximadamente, quando executadas em um Pentium 4 com 2.8 GHz.

Avaliamos o comportamento das estratégias de escalonamento considerando variadas taxas de chegada de aplicações, o que permitiu analisar o comportamento destas estratégias em ambientes de grade com baixa, média e alta carga de trabalho. Foram simuladas as seguintes taxas de chegada de aplicações por minuto: 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 5, 10, 20, 30, 60, 120 e 360.

A combinação das estratégias de escalonamento com as taxas de chegada e os ambientes de grade resultaram em um total de 192 simulações diferentes, que foram executadas em um Intel Pentium 4 2.8 GHz com 2 GB de RAM rodando o kernel do Linux com a versão 2.6.24-19. Cada simulação foi repetida 30 vezes, resultando em 5760 experimentos. Para cada experimento, medimos o tempo de conclusão por aplicação e calculamos o tempo médio de conclusão das 1500 aplicações. Então, calculamos a média do tempo médio de conclusão das aplicações para cada 30 conjuntos de aplicações que, para o restante do capítulo, chamaremos apenas de tempo médio de conclusão por aplicação.

Para cada simulação realizada, além do tempo médio de conclusão por aplicação, calculamos também os seguintes dados estatísticos: o valor mínimo, o valor máximo, o desvio padrão, a variância e o intervalo de confiança de 95%. Devido ao grande volume de dados estatísticos gerados para as simulações, apresentadas ao longo deste capítulo, optamos colocar estes dados em apêndice nesta dissertação.

As Figuras 5.1, 5.2 e 5.3 mostram os resultados obtidos nas simulações para os três diferentes ambientes de grade (100, 200 e 400 máquinas). Analisando os resultados destas simulações, podemos concluir que:

- Os algoritmos MCT e Min-min apresentaram os menores tempos médios de conclusão das aplicações devido ao uso de dados sobre o tempo de execução estimado para as aplicações nos recursos da grade. A diferença do MCT e do Min-min em relação ao WQ pode ser observada mais claramente para baixas taxas de chegada de aplicações, onde a grade permanecia com vários recursos disponíveis e o WQ escolhia um recurso aleatório o qual, provavelmente, não era o recurso mais rápido para executar a tarefa. Isto pode ser observado no Gráfico

5.1 em taxas de chegada abaixo de 0.05 aplicações por minuto;

- A diferença do tempo médio de conclusão das aplicações entre o WQ e o MCT tende a diminuir a partir do momento em que quase todos os nós da grade ficam ocupados. Isto acontece porque o WQ escalona as tarefas para as máquinas que se tornam disponíveis primeiro. A tendência é que as máquinas rápidas terminem suas tarefas primeiro, recebendo novas atribuições.

- Para várias taxas de chegada de aplicações, o tempo médio de conclusão por aplicação obtido pelo MCT e pelo Min-min ficaram próximos. No entanto, à medida que a taxa de chegada se torna maior, o algoritmo Min-min tende a apresentar um tempo médio de conclusão cada vez menor em relação ao MCT, já que a quantidade de aplicações consideradas por evento de mapeamento é maior do que 1, o que permite a comparação dos requisitos das aplicações entre si gerando um melhor mapeamento. O intervalo de tempo entre os eventos de mapeamento do Min-min foi de 1 minuto para todas as simulações realizadas. Isto acontece no Gráfico 5.1 a partir da taxa de chegada de 0.05 aplicações por minuto.

- Comparando os três gráficos, podemos observar que o aumento na disponibilidade de recursos da grade altera o ponto (taxa de chegada) a partir do qual o algoritmo Min-min passa a apresentar o menor tempo médio de conclusão das aplicações. Nos Gráficos 5.1, 5.2 e 5.3 este ponto corresponde, respectivamente, a 0.05, 0.1 e 0.5. Isto se deve ao fato de que o aumento da quantidade de máquinas disponíveis implica na diminuição do ganho de tempo advindo da comparação que o Min-min faz entre os requisitos das tarefas. Portanto, a escolha do algoritmo a ser utilizado em função da taxa de chegada das aplicações deve levar em consideração a quantidade de máquinas disponíveis na grade.

5.1.2 Avaliação das Estratégias de Escalonamento em um Ambiente de Grade com Variação das Taxas de Chegada das Aplicações e com Diferentes Intervalos entre Falhas de Recursos

O objetivo deste segundo conjunto de simulações foi avaliar os desempenhos obtidos para as mesmas estratégias de escalonamento utilizadas nas simulações

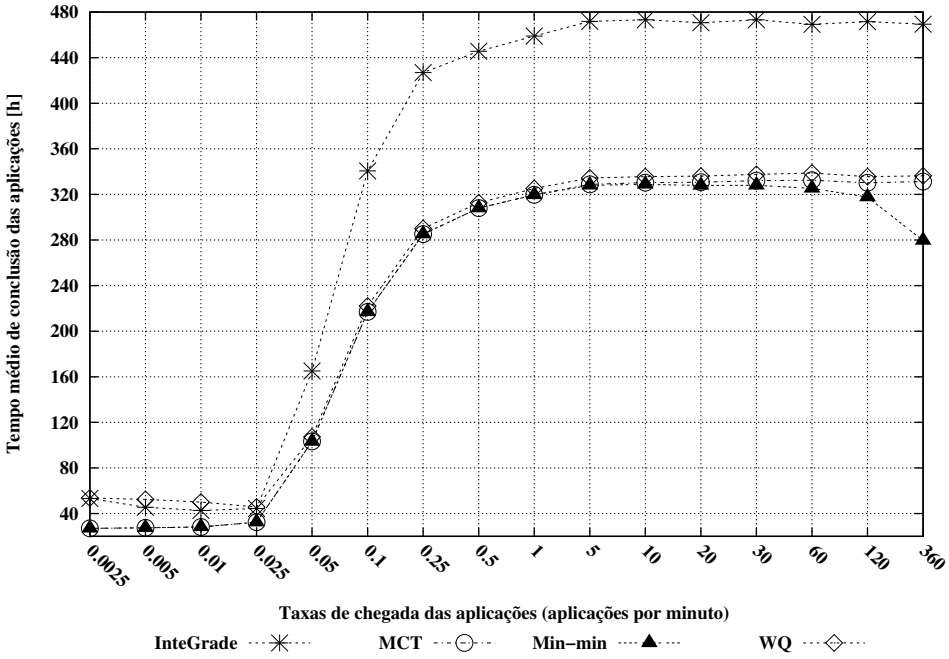

Figura 5.1: Tempo médio de conclusão por aplicações em função da variação da taxa de chegada de aplicações por minuto em uma grade composta por 100 máquinas.

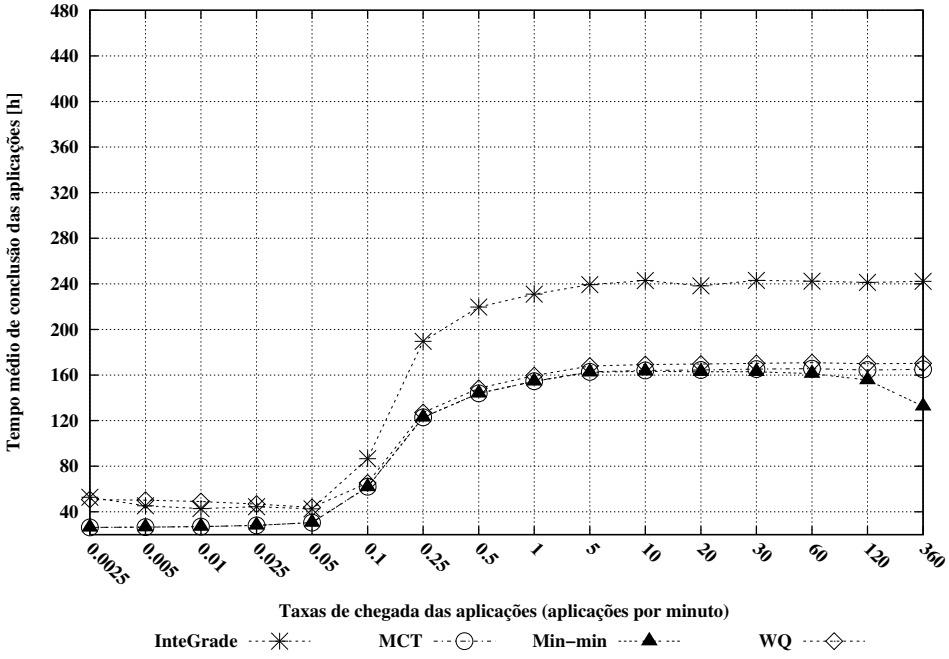

Figura 5.2: Tempo médio de conclusão por aplicações em função da variação da taxa de chegada de aplicações por minuto em uma grade composta por 200 máquinas.

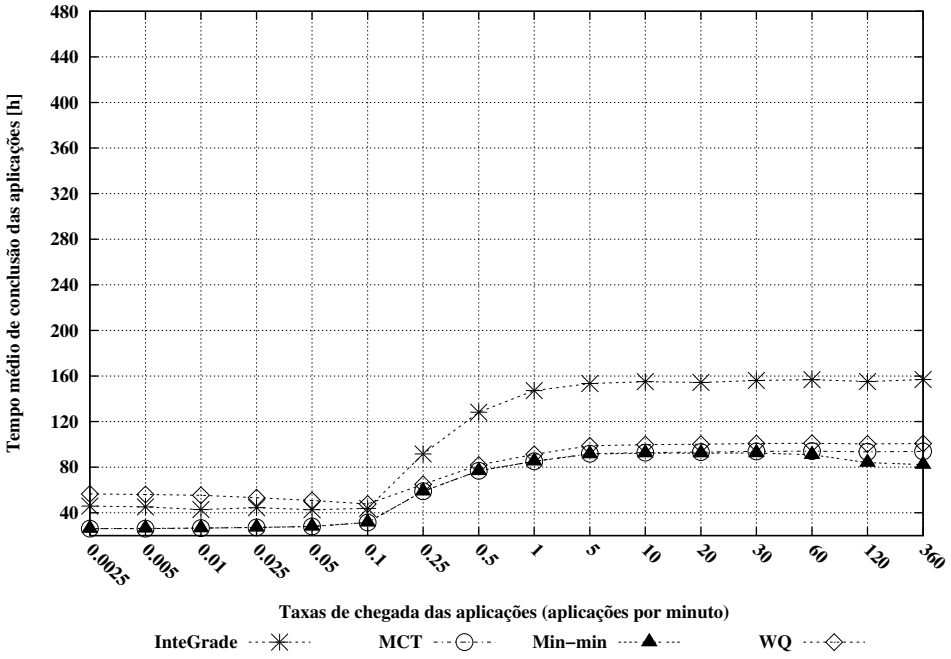

Figura 5.3: Tempo médio de conclusão por aplicações em função da variação da taxa de chegada de aplicações por minuto em uma grade composta por 400 máquinas.

anteriores, considerando uma variação das taxas de chegada das aplicações (baixa e alta) e inserindo falhas de recursos, com diferentes intervalos entre falhas (curto, médio e longo). Estas simulações também analisaram o desempenho dos algoritmos considerando o propósito de minimizar o tempo médio de conclusão por aplicação.

O ambiente de grade simulado foi composto por 100 máquinas. Este ambiente foi o mesmo utilizado nas simulações anteriores quando 100 máquinas foram utilizadas. As aplicações simuladas também foram modeladas da mesma maneira que as aplicações das simulações anteriores.

As simulações levaram em consideração a variação de dois parâmetros do ambiente de execução: a taxa de chegada das aplicações e o tempo médio entre falhas dos nós. Consideramos as seguintes taxas de chegada de aplicações por minuto: 0.025 (baixa) e 0.25 (alta). Os seguintes intervalos entre falhas de recursos foram simulados: 1, 2, 4, 6, 8, 10, 12 e 14 horas. A recuperação dos nós seguiu uma distribuição uniforme $U(6, 48)$ horas, a fim de caracterizar curtas e longas falhas de recursos computacionais, variando desde interferências do usuário local até falhas de *hardware* da máquina. Levamos ainda em consideração o uso de dois mecanismos de tolerância a falhas: reinício e *checkpointing*, resultando em um total de 128 simulações diferentes executadas no mesmo Pentium 4 descrito na seção anterior. Assim como

as simulações anteriores, cada simulação também foi composta por 30 experimentos, gerando um total de 3840 experimentos.

A Figura 5.4 mostra o tempo médio de conclusão das aplicações em função do intervalo entre falhas de recursos gerado quando foi utilizada uma taxa de chegada das aplicações de 0.025 aplicações por minuto e as aplicações tiveram suas execuções reiniciadas em um outro nó da grade quando o nó em que executavam falhava. A partir destas simulações, concluímos que:

- Com um pequeno intervalo entre falhas (1 hora), o WQ alcançou o melhor desempenho (baixo tempo de conclusão por aplicação), uma vez que o WQ somente escalona uma tarefa por máquina, reduzindo o número de tarefas submetidas novamente em caso de falha do nó, uma vantagem quando o ambiente exibe uma alta taxa de falhas.

- Assim que a taxa de falhas torna-se menor (2 até 14 horas do tempo médio entre falhas dos nós), MCT e Min-min apresentam um menor tempo de conclusão por aplicação do que o WQ, uma vez que eles podem executar um melhor mapeamento através do uso de informações sobre o tempo de conclusão esperado por aplicação.

- MCT apresentou um melhor desempenho em comparação ao Min-min, uma vez que nesta simulação a taxa de chegada de aplicações foi baixa (0.025 aplicação por minuto) e o MCT usa a abordagem *on-line* que escalona as tarefas assim que chegam, enquanto que o Min-min usa a abordagem *batch* que aguarda por eventos de mapeamento que, dado a baixa taxa de chegada de aplicações, será composto por uma pequena quantidade de tarefas.

Na Figura 5.5 consideramos a mesma taxa de 0.025 aplicação por minuto, porém levando em consideração o uso de *checkpointing*. Neste caso, mesmo quando a taxa de falhas foi alta (uma hora para o intervalo entre falhas), o MCT e o Min-min executaram melhor do que o WQ, pois o baixo número de reenvio de tarefas obtido com o WQ torna-se menos relevante, uma vez que as tarefas não necessitam reiniciar a computação desde o começo depois de uma falha.

A Figura 5.6 mostra os resultados das simulações quando o ambiente exibe um alta taxa de chegada de aplicações (0.25 aplicações por minuto) e o mecanismo

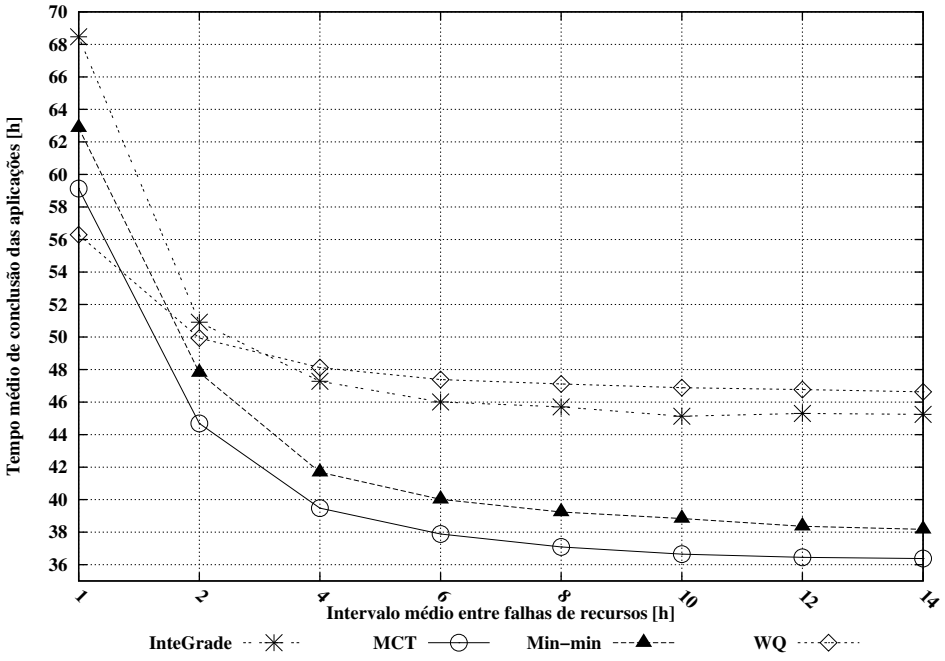

Figura 5.4: Tempo médio de conclusão das aplicações em função do intervalo médio entre falhas de recursos, considerando uma taxa de chegada de 0.025 aplicações por minuto e uso do mecanismo de reinício.

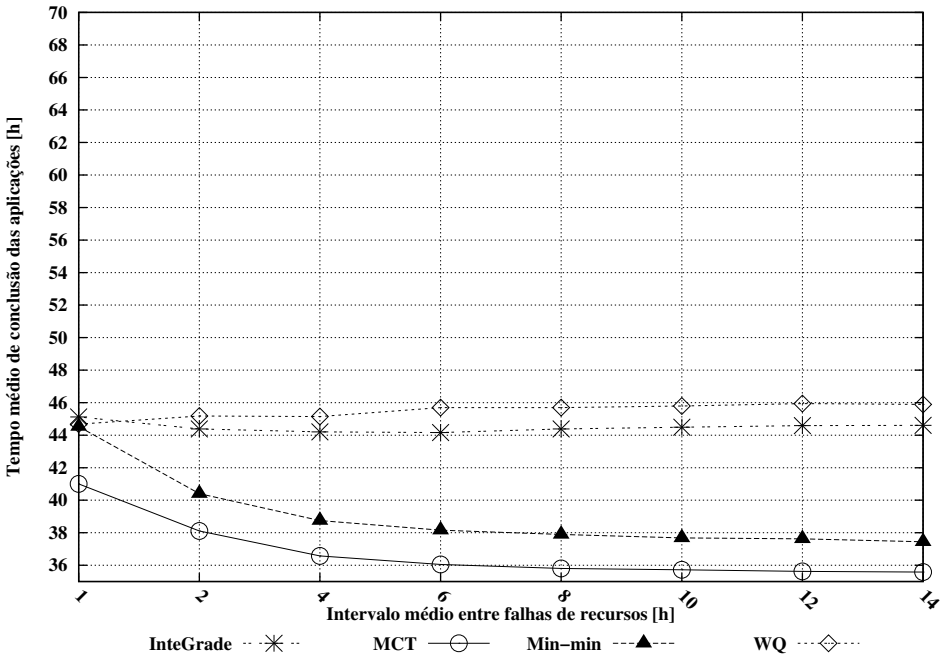

Figura 5.5: Tempo médio de conclusão das aplicações em função do intervalo médio entre falhas de recursos, considerando uma taxa de chegada de 0.025 aplicações por minuto e uso do mecanismo de *checkpointing*.

de reinício é usado para a recuperação de tarefas. Neste caso, o Min-min obteve os melhores resultados, uma vez que mais aplicações compõem o conjunto do evento de mapeamento, permitindo a comparação de seus tempos de conclusões estimados e isto conduz a um melhor mapeamento.

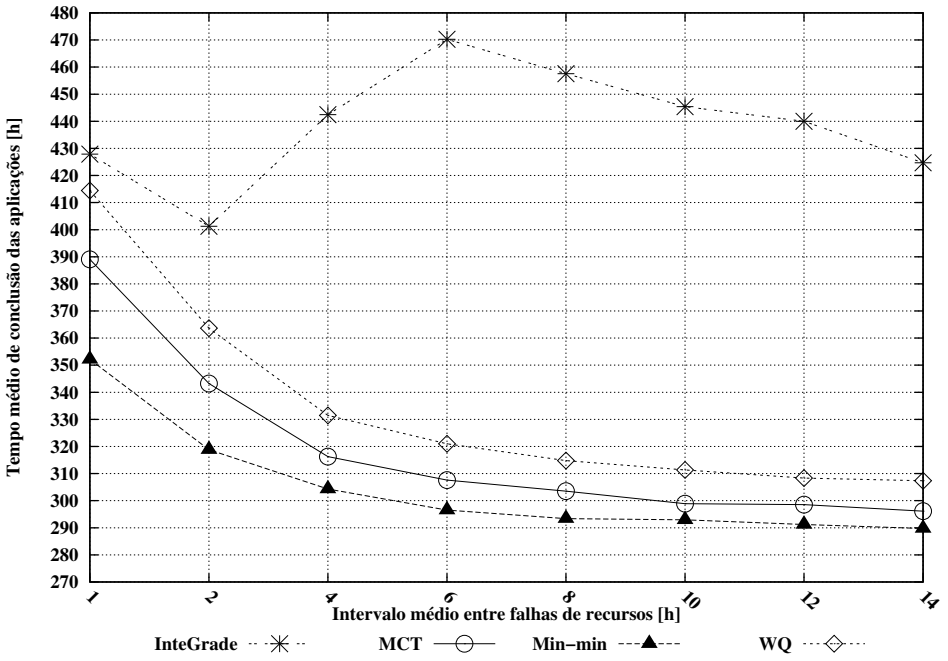

Figura 5.6: Tempo médio de conclusão das aplicações em função do intervalo médio entre falhas de recursos, considerando uma taxa de chegada de 0.25 aplicações por minuto e uso do mecanismo de reinício.

Na Figura 5.7, consideramos a mesma taxa de 0.25 aplicação por minuto e mais uma vez avaliamos o mecanismo de *checkpointing*. Os resultados obtidos mostram que a ordem do desempenho das heurísticas de escalonamento permanece igual aos resultados anteriores, com o Min-min apresentando o menor tempo médio de conclusão das aplicações. Isto enfatiza o melhor desempenho de algoritmos *batch* quando os recursos da grade estão sobrecarregados.

O algoritmo de escalonamento do InteGrade apresentou um tempo médio de conclusão das aplicações maior do que o tempo obtido pelos outros algoritmos nos resultados das simulações. Isto acontece porque nas simulações realizadas não foram utilizadas informações sobre os requisitos e preferências dos usuários para a seleção de recursos. Quando estas informações não estão disponíveis, o algoritmo do InteGrade escolhe recursos de forma aleatória para a execução das tarefas. Além disso, ele mapeia mais de uma tarefa por recurso.

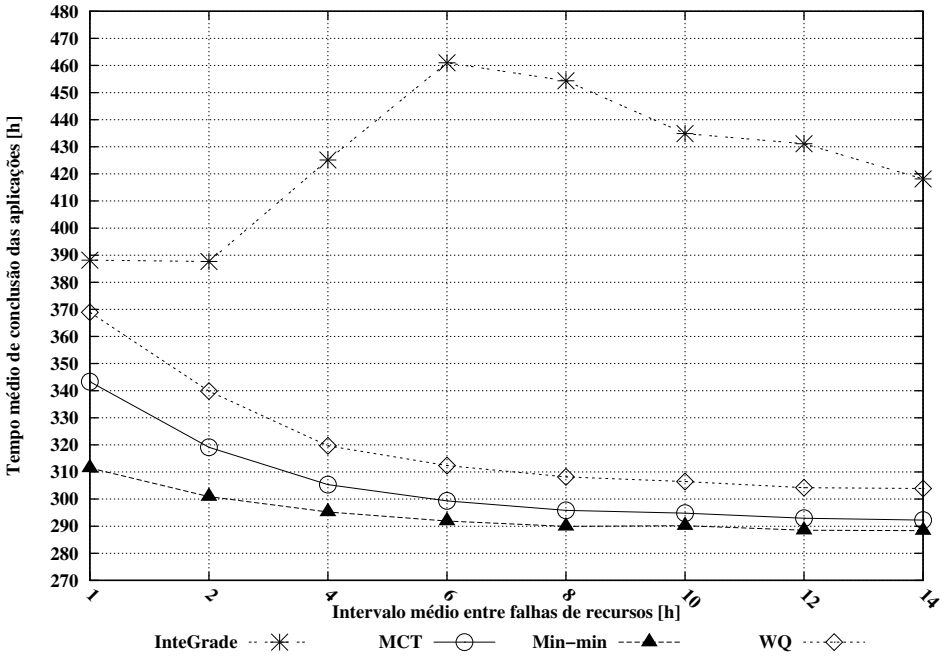

Figura 5.7: Tempo médio de conclusão das aplicações em função do intervalo médio entre falhas de recursos, considerando uma taxa de chegada de 0.25 aplicações por minuto e uso do mecanismo de *checkpointing*.

5.1.3 Investigação de uma Proposta de Abordagem Adaptativa

Após identificarmos que a heurística do MCT foi melhor para uma baixa taxa de chegada de aplicações e que a heurística do Min-min apresentou um bom desempenho para uma alta taxa de chegada de aplicações, realizamos mais um conjunto de simulações com o objetivo de investigar uma abordagem adaptativa que integre as duas heurísticas.

O ambiente de grade simulado composto por 100 máquinas e a modelagem das aplicações seguiram a mesma abordagem das simulações anteriores. Novamente, as simulações levaram em consideração a variação de dois parâmetros do ambiente de execução: a taxa de chegada das aplicações e o tempo médio entre falhas dos nós. Por outro lado, elas avaliaram o desempenho das heurísticas de escalonamento em função de um conjunto maior de taxa de chegada das aplicações, afim de investigar as heurísticas sob uma baixa, média e alta taxa de chegada. Foram simuladas as seguintes taxas de chegada de aplicações por minuto: 0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 5, 10, 20, 30, 60, 120 e 360. Os intervalos entre falhas de nós (1, 2, 4, 6, 8, 10, 12 e 14 horas) permaneceram iguais aos intervalos definidos para as simulações

anterirores, o que permite avaliarmos também ambientes com curto, médio e longo intervalo entre falhas de recursos. O mecanismo de tolerância a falhas utilizado foi o reinício de tarefas.

A combinação destes parâmetros gerou 272 simulações diferentes, que foram executadas em duas máquinas Intel Core 2 1.86 GHz com 1 GB de RAM rodando o *kernel* do Linux com a versão 2.6.24-21. Assim como as simulações anteriores, cada simulação também foi composta por 30 experimentos, obtendo-se um total de 8160 experimentos.

A Figura 5.8 apresenta os resultados das simulações através de um gráfico de superfície que relaciona: a diferença do tempo médio de conclusão das aplicações obtido pelas estratégias de escalonamento MCT e Min-min (k), com vários intervalos entre falhas de nós e com várias taxas de chegada de aplicações. Um valor k positivo significa que a heurística MCT obteve um tempo médio de conclusão das aplicações k horas menor do que o tempo obtido pelo Min-min, para um mesmo intervalo entre falhas e taxa de chegada. Um valor k negativo significa que, a heurística Min-min obteve um tempo médio de conclusão das aplicações k horas menor do que o tempo obtido pelo MCT, para um mesmo intervalo entre falhas de recursos e taxa de chegada das aplicações.

Os valores de k positivo no gráfico nos permitiu concluir que os desempenhos do MCT e do Min-min se mantém próximos um do outro, para uma variação de baixas taxas de chegada de aplicações e do intervalo entre falhas de recursos. Isto pode ser observado no gráfico para as taxas de chegada de 0.001 até 0.025 aplicações por minuto, nas quais o tempo médio de conclusão apresentado pelo MCT é média 5.83% menor do que o Min-min. Para os valores de k negativo podemos concluir que o Min-min apresenta uma relevante diferença de desempenho em relação ao MCT para uma variação de altas taxa de chegada de aplicações e torna-se cada vez mais acentuada à medida que os intervalos entre falhas de recursos ficam cada vez menores. Isto ocorre porque quanto maior a quantidade de falhas de recursos, maior será a quantidade de tarefas que retornam para um novo evento de mapeamento e os requisitos das tarefas podem ser comparados para um melhor escalonamento. Isto pode ser observado no gráfico a partir da taxa de chegada de 0.05 aplicações por minuto, onde o valor de k tende a ficar cada vez mais distante de 0 e o tempo médio de conclusão apresentado pelo Min-min é média 54.9% menor do que o MCT.

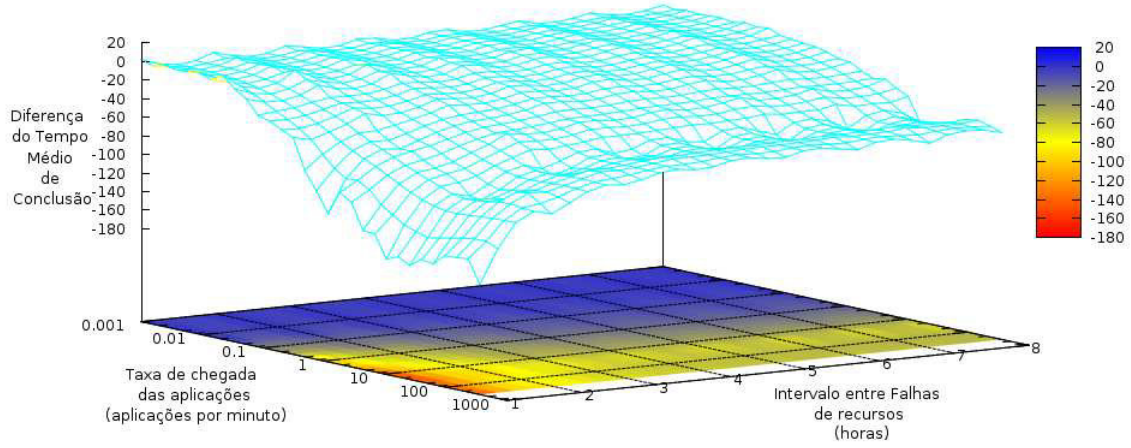

Figura 5.8: Diferença entre o tempo médio de conclusão das aplicações obtido pelo MCT e pelo Min-min, considerando uma variação no intervalo entre falhas de nós e nas taxas de chegada das aplicações. Utilizamos o mecanismo de reinício das tarefas para tolerância a falhas.

5.2 Conclusões

A análise dos resultados das simulações descritas neste capítulo enfatizaram que as heurísticas de escalonamento executam diferentemente sobre diversas condições do ambiente da grade. No entanto, o algoritmo Min-min merece destaque por ter apresentado um desempenho melhor que os demais em vários cenários investigados, considerando o objetivo de minimizar o tempo de conclusão das aplicações. Observamos que o Min-min requer informações sobre o tempo de execução estimado das tarefas nos recursos da grade com um bom grau de precisão. Uma possível abordagem para a obtenção destas informações é apresentada por Liu Y. [38] e é baseada no armazenamento de um histórico de execução das tarefas, a partir do qual a média do tempo de execução das mesmas é calculado. A precisão em estimar o tempo de execução aumenta à medida que são executadas mais tarefas da mesma aplicação na grade. O Nimrod-G é um exemplo de sistema de escalonamento que utiliza o histórico de execuções da tarefa para estimar seu tempo de execução [2].

Como pode ser observado nas simulações de um ambiente livre de falhas, a heurística Min-min obteve o melhor desempenho tendo-se como objetivo a minimização do tempo de conclusão das aplicações.

Para ambientes de grade com baixa taxa de chegada de aplicações (0.025

aplicações por minuto), alta taxa de falhas de recursos computacionais (intervalo médio de 1 hora entre falhas de recursos) e fazendo uso do mecanismo de reinício das tarefas que tiveram sua computação interrompida devido a falhas de recursos, o algoritmo *Work queue* obteve o menor tempo de conclusão das aplicações. No entanto, a diferença do tempo de conclusão por aplicação do WQ para o Min-min foi de 10.47%. Quando o intervalo entre falhas de recursos aumentou e consideramos a mesma taxa de chegada de aplicações, o MCT obteve o melhor desempenho, com um tempo médio de conclusão de aplicações 5.83% menor do que o Min-min. Considerando ainda uma baixa taxa de chegada e com o uso do mecanismo de *checkpointing*, o MCT apresentou um tempo de conclusão 4.24% menor do que o Min-min.

Em um ambiente com variação de altas taxas de chegada de aplicações, variação no intervalo entre falhas de recursos e uso do mecanismo de tolerância a falhas de reinício de tarefas, a heurística Min-min obteve o menor tempo médio de conclusão das aplicações (em média 54.9% menor do que o MCT).

6 Trabalhos Relacionados às Ferramentas de Simulação de Grades de Computadores

Ferramentas para simulação de grades e de computação voluntária estão surgindo como forma de prover um ambiente controlável e de baixo custo para a avaliação de novas arquiteturas e abordagens que lidam com os desafios de prover um ambiente, em larga escala, confiável e altamente distribuído para execução de aplicações computacionalmente intensivas a longo prazo.

Este capítulo descreve relevantes projetos cujo objetivo é o desenvolvimento de ferramentas para simulação de grades de computadores. Após um resumo de cada projeto é realizada uma análise comparativa com a ferramenta proposta nesta dissertação.

6.1 GridSim

O GridSim[1] [9] provê suporte à modelagem e simulação de recursos heterogêneos, usuários, aplicações e escalonadores de ambientes computacionais de grade. O GridSim é um simulador de eventos discretos escrito em Java e baseado em uma arquitetura *multithread*, onde cada componente do simulador é controlado individualmente por uma *thread*. Na Seção 3.4, apresentamos as características gerais do GridSim e as comparamos com as funcionalidades providas pelo OGST. Nesta Seção apresentamos uma visão geral da arquitetura do GridSim.

6.1.1 Arquitetura

Como ilustrado pela Figura 6.1, o GridSim adota uma arquitetura multicamadas. A primeira camada se refere ao ambiente de execução Java (*Java virtual machine* (JVM)) e às interfaces portáveis e escaláveis (cJava, RMI), cuja implementação está disponível para sistemas monoprocessados e multiprocessados. A segunda

[1] Página Inicial: `http://www.gridbus.org/gridsim`

camada se refere a uma infra-estrutura de eventos discretos construído sobre a JVM para dirigir a simulação do GridSim. Atualmente, a infra-estrutura utilizada é o SimJava [30]. A terceira camada compreende as principais ferramentas disponibilizadas especificamente pelo GridSim. Ela permite a modelagem e simulação das principais entidades da grade tais como recursos, serviço de informação dos recursos, primitivas de modelagem da aplicação e componentes para criar entidades de alto nível. A quarta camada provê a simulação de escalonadores de recursos. A última camada está focada na modelagem de aplicações e recursos com diferentes cenários usando os serviços providos pela terceira e quarta camada para avaliar algoritmos, heurísticas e políticas de gerenciamento de recursos e de escalonamento de tarefas.

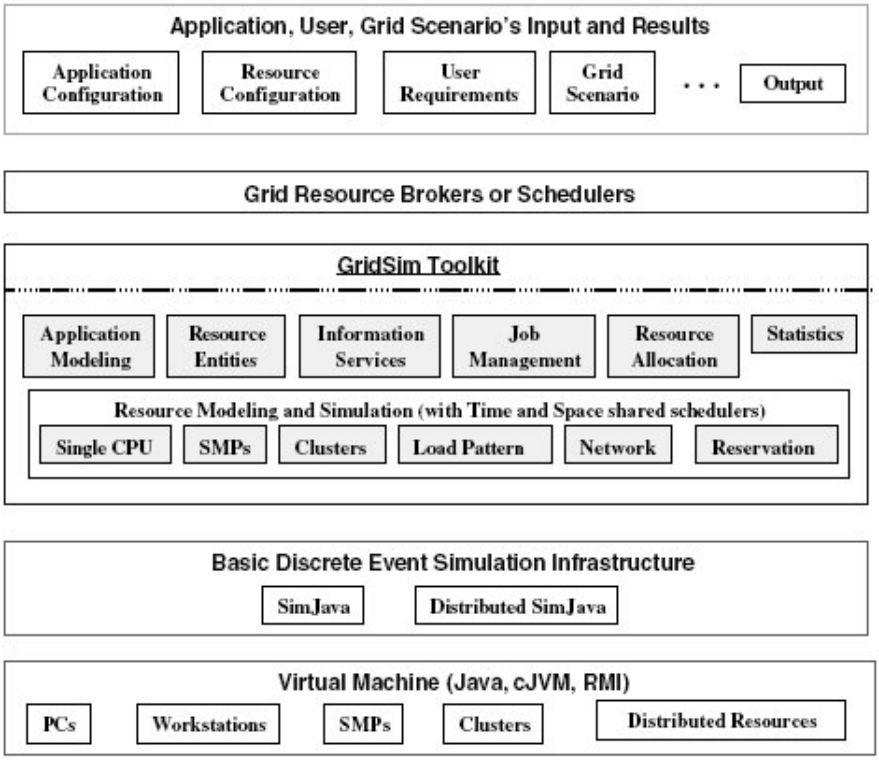

Figura 6.1: Arquitetura do GridSim. Fonte: [9].

6.2 SimGrid

Um simulador que permite incluir arquivos de *traces* trabalha lendo um conjunto de eventos que, provavelmente, foram coletados de algum ambiente real observado por algum período de tempo. Um simulador do tipo *serial* determina que a

execução de uma simulação pode usar somente um único processador. O SimGrid[2] [11] é um *framework* de simulação de eventos discretos, escrito em C, *serial* e que permite incluir *traces*, a fim de prover uma ferramenta de avaliação geral para computação distribuída em larga escala. Semelhante ao GridSim, ele permite a avaliação de algoritmos em *cluster*, grades e *peer-to-peer* através de componentes que modelam e simulam recursos da grade, aplicações, escalonadores e usuários.

6.2.1 Arquitetura

A Figura 6.2 ilustra a arquitetura do SimGrid. Esta arquitetura possui diversas camadas, divididas pelas funcionalidades específicas de cada conjunto de API.

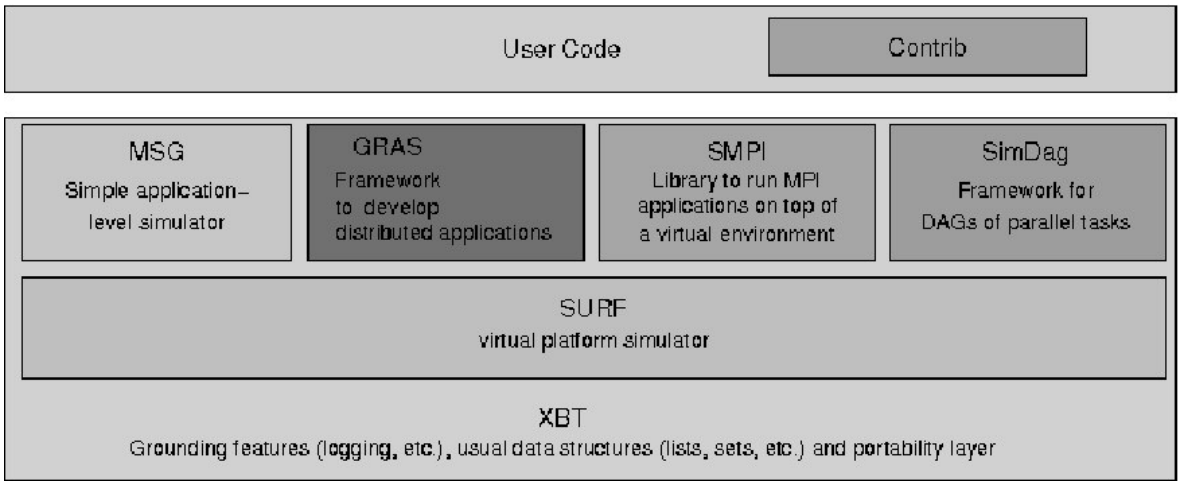

Figura 6.2: Arquitetura do SimGrid. Fonte: [11].

As suas camadas são:

- XBT (*eXtended Bundle of Tools*): implementa uma biblioteca de estruturas de dados utilizadas pela camada acima (SURF), mecanismos de exceção, um serviço de *logging* e suporte para configuração e portabilidade da ferramenta.
- SURF: disponibiliza uma API de baixo nível que, entre suas funcionalidades, é responsável por preparar o início da simulação, criar estruturas utilizadas durante a simulação (ex: recursos da grade), gerenciar o relógio da simulação, calcular o tempo de conclusão das ações (ex: execução de uma tarefa em um recurso) e notificar ações que são finalizadas à interface do usuário.

[2]Página Inicial: `http://simgrid.gforge.inria.fr/`

- MSG (*Meta-SimGrid*): o MSG foi o primeiro ambiente de programação provido no SimGrid e é a API mais utilizada para realizar a simulação e avaliação de heurísticas de escalonamento. Ela permite simulações no modelo de aplicações paralelas não acopladas (com mestre e escravos). O mestre envia tarefas aos escravos, que lhe devolvem resultados.

- GRAS (*Grid Reality And Simulation*): provê uma API para implementar aplicações distribuídas (ex: serviço de informação de recursos da grade) no topo de plataformas heterogêneas. Esta interface provê um suporte para o ciclo de desenvolvimento de aplicações através do simulador, permitindo que o código para plataformas reais seja automaticamente criado. Este ambiente faz uso da API fornecida pelo MSG para implementar a aplicação da simulação e de *sockets* para implementar a aplicação real.

- SMPI: visa permitir o desenvolvimento do mesmo código fonte para as aplicações MPI que serão executadas tanto no simulador, quanto em um ambiente real. Esta API ainda está em fase de desenvolvimento.

- SimDag : permite criar protótipos e simular heurísticas de escalonamento para aplicações paralelas acopladas estruturadas como grafos de tarefas (DAG - *Direct Acyclic Graphs*). Com esta API pode-se criar tarefas, adicionar dependências entre as tarefas, escalonar tarefas para execução nos recursos computacionais e computar o tempo de execução da aplicação DAG.

6.2.2 Aspectos de Configuração da Simulação

Para realizar uma simulação no SimGrid deve-se definir a plataforma da grade e as tarefas a serem executadas. A plataforma da grade consiste em um conjunto de recursos computacionais e enlaces de rede definidos pelo usuário do simulador em um arquivo XML (Extensible Markup Language) (*platform.xml*). Os recursos computacionais que executam as tarefas são modelados pelo seu nome e poder computacional, enquanto que os enlaces de rede são modelados pela sua largura de banda e latência.

Outro aspecto abordado pelo SimGrid, é a possibilidade de permitir o dinamismo na plataforma da grade, indicando arquivos *traces* que indicam as mudanças na disponibilidade dos recursos computacionais e ocorrências de falhas de recursos.

Em outro arquivo XML (*deployment.xml*) são especificados a quantidade de tarefas que serão executadas, o tamanho da computação das tarefas, o tamanho da comunicação das tarefas e os recursos que executam cada tarefa.

O SimGrid não faz distinção entre transferência de dados e computação: ambos são vistos como tarefas e é de responsabilidade do usuário garantir que tarefas de computação sejam escalonadas em processadores, e transferência de arquivos para enlaces de rede.

6.2.3 Apresentação dos Resultados

O SimGrid não define nenhum padrão para a exibição dos resultados de uma simulação. No entanto, ele oferece ferramentas que podem ajudar na visualização dos dados que são gerados durante e ao final da simulação. Através de um conjunto de funções do SimGrid são definidas mensagens pré-formatadas para acompanhamento da simulação, contendo o valor corrente do relógio da simulação, o recurso e o respectivo processo que origina uma determinada mensagem. Por exemplo, toda vez que o usuário envia uma tarefa para o escalonador, a mensagem pode identificar o horário em que a tarefa foi enviada e o identificador do usuário e o do escalonador.

6.3 SimBOINC

O SimBOINC[3] é um simulador de sistemas de computação voluntária e de grades formadas por computadores pessoais. O objetivo deste simulador é prover uma maneira de testar novas estratégias de escalonamento no BOINC, que executam localmente sobre cada máquina cliente [4]. Ele é baseado no conjunto de ferramentas de simulação disponibilizadas pelo SimGrid [11]. Assim como o BOINC [4], o simulador segue um modelo cliente-servidor para executar as aplicações paralelas não acopladas. O simulador usa uma abordagem *pull*, onde os clientes solicitam tarefas para os servidores que, por sua vez, executam as tarefas e retornam os resultados.

[3]Página Inicial: `http://simboinc.gforge.inria.fr`

6.3.1 Aspectos de Configuração da Simulação

O SimBOINC define um conjunto de arquivos de entrada para iniciarmos a simulação. Os principais arquivos são descritos a seguir:

- **Arquivo da plataforma**: este arquivo especifica os recursos computacionais e de enlaces de rede relacionados aos clientes e servidor BOINC. Para cada recurso devem ser especificados um conjunto de atributos, como apresentado na Figura 6.3. Por exemplo, a CPU de um recurso é definida em termos do poder de computação (*power*) (linhas 5 a 7). Além disso, para os clientes ainda podem ser especificados arquivos de *traces* (*availability_file* e *state_file*) que descrevem, respectivamente, a disponibilidade dos recursos e os momentos de falhas dos recursos (linhas 6 e 7).

```
1 <?xml version='1.0'?>
2 <!DOCTYPE platform_description SYSTEM "surfxml.dtd">
3 <platform_description>
4
5   <cpu name="Server" power="100"/>
6   <cpu name="Host_1" power="100" availability_file="mini_avail.txt" state_file="mini_fail.txt"/>
7   <cpu name="Host_1-SB-CPU1" power="100" availability_file="mini_avail.txt" state_file="mini_fail.txt"/>
8   <network_link name="0" bandwidth="100" latency=".001"/>
9   <network_link name="1" bandwidth="100" latency=".001"/>
10  <network_link name="loopback" bandwidth="100.00" latency="0.001"/>
11  <route src="Server" dst="Server"><route_element name="loopback"/></route>
12  <route src="Host_1" dst="Host_1"><route_element name="loopback"/></route>
13  <route src="Server" dst="Host_1">
14    <route_element name="0"/>
15  </route>
16  <route src="Host_1" dst="Server">
17    <route_element name="1"/>
18  </route>
19 </platform_description>
```

Figura 6.3: Arquivo de definição da plataforma do SimBOINC. Fonte: [35]

No BOINC, pelo menos três coisas podem causar uma falha na execução das tarefas:

1. A tarefa pode sofrer preempção pelo cliente BOINC devido à política de escalonamento local no recurso do cliente;
2. A tarefa pode sofrer preempção pelo cliente BOINC devido à atividade local do usuário;
3. O recurso pode falhar (quando o *hardware* tem problemas ou usuário desliga a máquina).

No SimBOINC, as falhas de um recurso são definidas em arquivos de *traces* que representam as falhas resultantes das duas últimas causas. Quando um recurso computacional falha todos os processos que estavam executando na CPU são finalizados. No entanto, o estado das tarefas é mantido e quando os recursos tornam-se novamente disponíveis os processos são reiniciados do último estado de execução (*checkpoint*). O *checkpoint* de uma tarefa é armazenado localmente em cada recurso.

Para os recursos de rede deve-se especificar a largura de banda e latência dos enlaces (linhas de 8 a 10). Em seguida, as rotas de conexão dos enlaces de rede entre cliente e servidor devem ser definidas (linhas 13 a 18).

- **Arquivo de carga de trabalho**: este arquivo especifica os projetos do BOINC que serão executados na plataforma, como ilustrado na Figura 6.4. Em particular, ele define para cada projeto o nome (<*name*>), a quantidade de tarefas para serem executadas (<*num_tasks*>), o tamanho da tarefa em termos de computação (<*task_comp_size*>), o tamanho da tarefa em termos de comunicação (<*task_comm_size*>) e a frequência de *checkpoint* (<*chkpt_freq*>) para cada tarefa.

```
1 <jobs>
2 <job>
3     <name>predictor</name>
4     <num_tasks>1000</num_tasks>
5     <task_comp_size>1000</task_comp_size>
6     <task_comm_size>2</task_comm_size>
7     <chkpt_freq>10</chkpt_freq>
8 </job>
9 <job>
10    <name>seti</name>
11    <num_tasks>200</num_tasks>
12    <task_comp_size>500</task_comp_size>
13    <task_comm_size>.0000000001</task_comm_size>
14    <chkpt_freq>5</chkpt_freq>
15 </job>
16 </jobs>
```

Figura 6.4: Arquivo de definição da carga de trabalho do SimBOINC. Fonte: [35].

6.3.2 Apresentação dos Resultados

Dado os arquivos de configuração como entradas, o simulador irá executar e produzir um arquivo de saída com os valores de uma série de métricas de desempenho do escalonador. O arquivo de saída do simulador gera para cada cliente e cada projeto que o cliente participa as seguintes métricas:

- Total de tarefas finalizadas;
- Quantidade e percentual de prazos de conclusões perdidos para as tarefas finalizadas;
- Quantidade e percentual de prazos de conclusões alcançados para as tarefas finalizadas.

6.4 OptorSim

O OptorSim[4] [6] permite a simulação de ambientes de grades de dados nos quais um grande volume de conjuntos de dados são processados de maneira distribuída. A arquitetura do simulador é baseada no EU DataGrid[5]. O foco do simulador está na replicação de dados, que envolve a criação e gerenciamento de réplicas em diferentes localizações geográficas com métodos para minimizar o custo de acesso a dados. O OptorSim é um simulador de eventos discretos, escrito em Java e *multithread*.

6.4.1 Arquitetura

A Figura 6.5 ilustra a arquitetura do simulador OptorSim. O simulador considera que a grade é formada por diversos domínios administrativos (*Job execution site*), os quais provêem recursos computacionais (CE - *Computing Element*) e de armazenamento (SE - *Storage Element*). Um CE executa tarefas que fazem uso de dados (arquivos) localizados nos SEs.

O *Resource Broker* controla o escalonamento de tarefas para os CEs. Entre as estratégias de escalonamento de tarefas disponíveis para o *Resource Broker*, podemos citar:

- *Random*: escalona a tarefa aleatoriamente para qualquer recurso computacional disponível;

[4]Página Inicial: `http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html`
[5]Página Inicial: `http://www.edg.org`

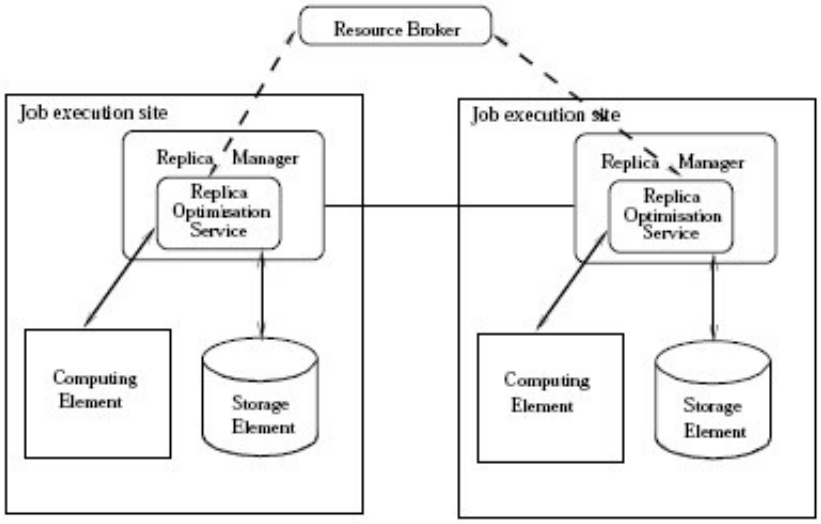

Figura 6.5: Arquitetura do OptorSim. Fonte: [6].

- *Shortest Queue*: escalona a tarefa para o recurso computacional com a menor fila de espera;
- *Access cost*: escalona a tarefa para o recurso computacional com o menor custo de acesso a todos os arquivos que a tarefa utilizará (considerando a latência da rede).

A movimentação de dados entre os domínios administrativos é realizada pelo *Replica Manager*, de acordo as decisões do *Replica Optimization Service*, que decide quando deve-se criar ou eliminar réplicas de dados de acordo com algum algoritmo de otimização.

As tarefas das aplicações podem solicitar vários arquivos durante suas execuções. As solicitações por arquivos são feitas através de referências. As referências podem ser de dois tipos:

1. Nome Lógico do Arquivo (LFN), que representa uma referência abstrata a um arquivo independente de sua localização;
2. Nome Físico do Arquivo (PFN), que referencia uma réplica de um LFN localizado em um dado domínio. Cada LFN possui um PFN distinto para cada réplica na grade.

A melhor réplica é aquela que pode ser transferida mais rapidamente para o recurso computacional, baseando-se na informação sobre a largura de banda dos enlaces da rede. Quando um arquivo é requisitado por uma tarefa, o arquivo LFN é usado para localizar a melhor réplica, através do método `getBestFile(LFN, destinationStorageElement)` do *Replica Optimization Service*, onde `destinationStorageElement` é o recurso de armazenamento para onde a réplica pode ser copiada, caso exista espaço no elemento de armazenamento.

Para localizar a melhor réplica, o método `getBestFile` verifica o catálogo de réplicas da grade, que mapeia os LFNs e seus correspondentes PFNs, examinando a largura de banda disponível entre o `destinationStorageElement` e todos os domínios onde a réplica do arquivo está armazenada.

O método `getBestFile` pode eventualmente criar réplicas dos dados no elemento de armazenamento de um domínio no qual a tarefa será executada. Após a replicação de um arquivo, o PFN da melhor réplica disponível é retornado à tarefa. Se não ocorre replicação, a melhor réplica é localizada em um domínio remoto e é acessado pela tarefa usando mecanismos de acesso remoto.

Quando não há mais espaço nos elementos de armazenamento de um dado domínio, um arquivo deve ser eliminado para que a replicação ocorra. A estratégia adotada para decidir qual o arquivo deve ser eliminado é o que diferencia os algoritmos de otimização de réplicas de dados. O método `getBestFile` contém o algoritmo de otimização de réplicas utilizado pelo *Replica Optimization Service*. O OptorSim implementa alguns algoritmos de otimização, como por exemplo:

- *Unconditional replication, oldest file deleted*: este algoritmo replica um arquivo para o domínio onde a tarefa está executando. Se não existe espaço para acomodar a replicação, o arquivo mais velho no recurso de armazenamento é apagado.
- *Unconditional replication, least accessed file deleted*: este algoritmo tem o mesmo comportamento que o método anterior, exceto que o arquivo menos acessado em um intervalo de tempo passado δt é apagado.

6.4.2 Aspectos de Configuração da Simulação

Os recursos da grade são definidos em um arquivo de configuração (grid.conf), no qual define-se:

- Zero ou mais recursos computacionais por domínio administrativo;
- Zero ou mais recursos de armazenamento por domínio administrativo;
- A capacidade dos recursos de armazenamento, em termos de *mega bytes*;
- Uma matriz que define a largura de banda entre os domínios administrativos da grade.

O OptorSim também permite ao usuário incluir arquivos de *traces* que definam o tráfego de rede de plano de fundo ao ambiente da grade, presente nos diferentes domínios administrativos.

As tarefas das aplicações são definidas em outro arquivo de configuração (jobs.conf). Para as tarefas são definidos os nomes dos arquivos que devem ser acessados durante suas execuções. Além do nome, os arquivos devem ter seus tamanhos e localizações definidas.

A submissão de tarefas do usuário para a grade ao longo do tempo pode ser feita de dois modos: o modo simples, que submete as tarefas em intervalos de tempos regulares e pré-definidos; e o modo aleatório, que submete tarefas em intervalos de tempo uniforme e aleatório.

O usuário da ferramenta pode ainda definir uma distribuição inicial dos arquivos, em que se define a localização de cada arquivo original entre os recursos de armazenamento disponíveis na grade ou ele pode distribuir aleatoriamente os arquivos pelos aglomerados da grade.

6.4.3 Apresentação dos Resultados

Durante a execução da simulação, o OptorSim permite que sejam exibidos em modo texto mensagens indicando a ocorrência de eventos. Por exemplo, o evento de submissão de uma tarefa do usuário para a grade pode apresentar uma mensagem

com informações sobre o nome da tarefa, o escalonador que recebeu a tarefa e o horário de submissão.

Após finalizada uma simulação, o OptorSim pode apresentar dados gerais e medidas estatísticas da simulação relativo a cada domínio administrativo. São apresentadas informações sobre o percentual de uso dos recursos computacionais, quantidade de arquivos acessados pelas tarefas, quantidade de réplicas criadas e o tempo total de execução das tarefas. Algumas destas informações podem ser visualizadas através de gráficos.

6.5 Comparação das Ferramentas de Simulação de Grades de Computadores

Nesta seção apresentamos uma análise comparativa entre as ferramentas de simulação, que foram descritas neste capítulo incluindo também o OGST. Tomamos como base para comparação os seguintes critérios:

1. Qual **mecanismo de simulação** utilizado: *multithread* ou *serial*;
2. Se a **simulação** é de eventos discretos ou é contínua;
3. Se o simulador permite ou não o **uso de arquivos de *trace***. Caso permita, qual a função destes arquivos (disponibilidade de recursos, injeção de falhas ou tráfego de rede de plano de fundo ao ambiente da grade);
4. Qual o **modelo de distribuição de tarefas**: a) o escalonador envia tarefas para os provedores de recurso (modelo *push*); b) é responsabilidade dos provedores de recurso requisitarem tarefas para execução (modelo *pull*); c) a ferramenta permite a implementação do modelo pelo usuário;
5. Quais **classes de aplicação** suportadas pela ferramenta de simulação (regular, paralelas não acopladas, paralelas acopladas);
6. Se o simulador permite a geração automática do tempo de **chegada das aplicações** de acordo com alguma distribuição de probabilidade especificada pelo usuário ou se é responsabilidade do usuário implementar a lógica da definição do tempo de chegada das aplicações;

7. Se a ferramenta de simulação permite ou não a **injeção de falhas** de recursos. Caso permita, qual a lógica utilizada para inserir falhas (ex: distribuição exponencial, arquivos de *trace*);

8. Quais mecanismos de **tolerância a falhas** que podem ser utilizados: reinício, *checkpointing*, replicação;

9. Como o **armazenamento** dos dados da simulação é realizado: através de arquivos ou utililzando-se sistemas gerenciadores de banco de dados.

10. Como a **visualização** dos resultados da simulação pode ser realizada: através de arquivos texto ou gráficos.

A Tabela 6.1 apresenta os resultados da comparação realizada. No OGST, GridSim e OptorSim o mecanismo de simulação é *multithread*. Cada recurso da grade é gerenciado por uma *thread* distinta. Máquinas multiprocessadas podem executar várias *threads* em paralelo, melhorando o desempenho da simulação. O SimGrid e o SimBOINC foram desenvolvidos com um mecanismo de simulação do tipo *serial*, determinando que a execução da simulação utiliza um único processador.

A simulação gerada por cada simulador apresentado neste capítulo é de eventos discretos, onde as mudanças de estado das entidades da simulação ocorre somente em pontos específicos do tempo. Os pontos são aqueles em que os eventos ocorrem. Em uma simulação contínua, o estado das entidades não mudam abruptamente como acontece com simuladores de eventos discretos, as mudanças ocorrem continuamente ao longo do tempo. As simulações de eventos discretos podem requerer um menor tempo para simulação de grandes grades computacionais, em comparação com simulações contínuas. Isto se deve ao fato de que o tempo da simulação avança irregularmente com a chegada de eventos.

O SimGrid e o SimBOINC usam arquivos de *traces* para indicar a variação de disponibilidade de CPU e o momento de falhas dos recursos. O OptorSim permite a inclusão de *traces* que simulam a variação do tráfego de rede que fica no plano de fundo do ambiente da grade. O OGST e o GridSim atualmente não permite a inclusão destes tipos de arquivos de *traces*. Obter estes arquivos de um ambiente real pode demorar bastante tempo e um ambiente de grade em grande escala de recursos nem sempre está disponível. No entanto, estes arquivos tornam a simulação mais realística.

O OGST, GridSim e o OptorSim são baseados no modelo de distribuição de tarefas do tipo *push*, onde o escalonador é responsável pelo mapeamento das tarefas para os provedores de recursos. Diferentemente dos outros, o SimBOINC é baseado no modelo *pull*, onde os clientes (provedores de recursos) requisitam trabalho. O SimGrid responsabiliza o usuário pela implementação do modelo de distribuição de tarefas.

Aplicações regulares e paralelas não acopladas estão presentes na maior parte das ferramentas de simulação (OGST, SimGrid, SimBOINC e OptorSim). O SimGrid também suporta a construção de aplicações paralelas acopladas. O GridSim não define explicitamente um modelo de aplicação. Ele delega a definição de modelos de aplicações ao desenvolvedor da simulação.

O OGST implementa a distribuição de poisson que pode ser utilizada para simular a submissão das aplicações. O OptorSim fornece dois mecanismos para submissão de aplicações: intervalos de tempos regulares e intervalos de tempos aleatórios. GridSim, SimGrid e SimBOINC definem como responsabilidade do usuário determinar o momento em que as aplicações são submetidas para a Grade.

No OGST, as falhas dos recursos são geradas automaticamente baseadas em uma distribuição exponencial. O intervalo de tempo entre as falhas dos recursos pode ser alterado dinamicamente no OGST, diferentemente do GridSim que define estaticamente no inicio da simulação as falhas de recursos de acordo com uma distribuição hiper-exponencial. No SimGrid e no SimBOINC, cada recurso possui um arquivo de *trace* que define os momentos em que recursos falham. O OptorSim pressupõe a avaliação de um ambiente de grade estável, no qual os recursos estão sempre disponíveis.

Com relação aos mecanismos de tolerância a falhas, o OGST foi projetado para prover três mecanismos comumente utilizados em grades: reinício, *checkpointing* e replicação. O GridSim permite o reinício das tarefas que falharam, mas não guarda o estado de execução do que já foi computado. O SimGrid atribui a responsabilidade de prover mecanismos de tolerância a falhas ao seu usuário. Assim, o SimBOINC que é baseado no SimGrid, desenvolve um mecanismo de *checkpointing* local a cada recurso, onde a tarefa pode reiniciar a sua computação a partir do último estado de execução salvo, caso o recurso que falhou se torne disponível novamente.

O armazenamento de dados das simulações da maior parte das ferramentas é realizado em arquivos. Estes dados geralmente são medidas estatísticas exibidas em linhas de texto. O OGST armazena os dados em um banco de dados relacional para permitir ao usuário relacionar grandes quantidades de dados de acordo com seus interesses específicos. A partir destes dados, o OGST gera gráficos que permitem ao usuário visualizar mais claramente algumas métricas que foram avaliadas durante a simulação (ex: taxa de chegada de aplicações, tempo médio de conclusão das aplicações, etc.). O OptorSim também possui ferramentas para geração automática de gráficos relativos aos resultados da simulação.

Ferramenta de simulação	Mecanismo de simulação	Simulação	Uso de arquivos de *trace*	Modelo de escalonamento	Classes de aplicação	Chegada de aplicações	Injeção de falhas	Tolerância a falhas	Armazenamento e Visualização de dados
OGST.	*Multithread*.	Eventos discretos.	-	Modelo *push*.	Regulares e Paralelas não Acopladas.	Distribuição de poisson .	Dinâmica, através da distribuição exponencial.	Reinício, *checkpointing* e replicação.	Banco de dados relacional e Gráficos.
GridSim.	*Multithread*.	Eventos discretos.	-	Modelo *push*.	Modelos de aplicações são definidos pelo usuário.	Definido pelo usuário.	Estática, através da distribuição hiperexponencial.	Reinício.	Arquivos e Texto.
SimGrid.	*Serial*.	Eventos discretos.	Disponibilidade de cpu e falhas de recursos.	Definido pelo usuário.	Regulares, Paralelas não Acopladas e Paralelas Acopladas;	Definido pelo usuário.	Arquivo de *Trace*.	-	Arquivos e Texto.
SimBOINC.	*Serial*.	Eventos discretos.	Disponibilidade de cpu e falhas de recursos.	Modelo *pull*.	Regulares e Paralelas não Acopladas.	Definido pelo usuário.	Arquivo de *Trace*.	*Checkpointing* local.	Arquivos e Texto.
OptorSim.	*Multithread*.	Eventos discretos.	Tráfego de rede de plano de fundo do ambiente da grade.	Modelo *push*.	Regulares e Paralelas não Acopladas.	Intervalos de tempo regulares ou aleatório.	-	-	Arquivos e Gráfico.

Tabela 6.1: Resumo comparativo das ferramentas de simulação de grades de computadores

7 Conclusões e Trabalhos Futuros

O processo de escalonamento de aplicações em um sistema de grade oportunista é uma tarefa desafiadora devido a diversas características presentes nestes ambientes computacionais, tais como a alta heterogeneidade e escalabilidade de recursos distribuídos, a autonomia de domínios administrativos e o dinamismo de recursos, tais como variações imprevisíveis na taxa de chegada de aplicações e de falhas de recursos. Estas características justificam a importância de investigar estratégias de escalonamento de aplicações, incluindo abordagens adaptativas e de reescalonamento de aplicações.

Este trabalho apresentou o OGST (*Opportunistic Grid Simulation Tool*), um simulador de eventos discretos orientado a objetos cujo principal objetivo é avaliar o comportamento de estratégias de escalonamento em diversos cenários comuns aos ambientes de execução de grades oportunistas. Ele foi desenvolvido no contexto do projeto InteGrade, mas foi projetado para permitir a simulação de grades oportunistas de uma maneira geral a fim de ser aplicado em outros projetos de pesquisa. O OGST foi cuidadosamente projetado para levar em consideração o dinamismo de grades oportunistas, provendo um conjunto de características que agilizam o desenvolvimento de simulações que levam em consideração o dinamismo do ambiente de execução.

As principais contribuições deste trabalho foram:

- A definição da arquitetura e implementação de uma ferramenta de simulação de grades oportunistas através da qual é possível simular a execução de algoritmos de escalonamento em diferentes cenários e condições do ambiente de execução.
- Avaliação da ferramenta de simulação através das seguintes técnicas: (a) mapeamento de funcionalidades, onde foram comparadas as funcionalidades do simulador com as de uma implementação concreta de *middleware* para grades oportunistas, o InteGrade; e (b) validação dos resultados, onde foram comparados os resultados (tempo médio de execução das aplicações) obtidos por meio de simulação com os obtidos através da execução de experimentos reais utilizando-se o InteGrade;

- Implementação e avaliação de um conjunto de estratégias de escalonamento comumente utilizadas em cenários de grades de computadores. Estas estratégias de escalonamento foram avaliadas em diversas condições do ambiente de execução de grades oportunistas. Consideramos diferentes quantidades de recursos, diferentes taxas de chegada de aplicações e diferentes intervalos de falhas entre recursos.

  A partir dos resultados do nosso trabalho foi publicado o seguinte artigo:

- *OGST: An Opportunistic Grid Simulation Tool*: artigo completo publicado no *2nd International Workshop Latin American Grid (LAGrid 2008)*, que descreve a arquitetura do OGST, aspectos de implementação e avaliação de estratégias de escalonamento em grades oportunistas [22].

7.1 Trabalhos Futuros

Durante o desenvolvimento do OGST identificamos diversas possibilidades interessantes de extensão deste trabalho que por restrição de tempo não puderam ser desenvolvidas, bem como identificamos alguns possíveis desdobramentos que podem advir deste esforço inicial. Entre estes possíveis trabalhos futuros podemos destacar:

- A construção e avaliação de novas estratégias de escalonamento de aplicações (possivelmente adaptativas) em grades oportunistas que levem em consideração variações típicas do ambiente de execução;

- A avaliação do comportamento de estratégias de tolerância a falhas de aplicações, tais como reinício, *checkpointing* e replicação em ambientes dinâmicos de grades;

- A implementação de um mecanismo de predição sobre a disponibilidade e utilização futura dos recursos computacionais baseado em padrões de uso, como apresentado por Conde em [15]. Isto permitirá o desenvolvimento e avaliação de novas estratégias de escalonamento que levem em consideração informações sobre a disponibilidade futura dos recursos;

- A avaliação do uso de informações de predição da disponibilidade de recursos em estratégias de escalonamento de aplicações;

- A implementação do suporte à simulação de aplicações paralelas que utilizem modelos como MPI e BSP, permitindo a simulação da execução desta classe de aplicações que, assim como aplicações regulares e paralelas não acopladas, é também usualmente empregada em ambientes de grades computacionais;

- A avaliação do mecanismo de replicação de tarefas do OGST comparando-o com o trabalho apresentado por Araújo, em [16], que implementa um mecanismo de replicação para o *middleware* InteGrade. Isto permitirá a validação das simulações que utilizam o mecanismo de replicação no OGST;

- A implementação do suporte para substituição dinâmica das estratégias de escalonamento e/ou ajuste dinâmico dos parâmetros de escalonamento durante a simulação, o que permitirá a investigação de abordagens de escalonamento adaptativas que visam minimizar os custos e maximizar os benefícios da escolha de recursos na grade;

- A extensão do modelo de injeção de falhas para permitir a simulação de falhas de enlace de rede dado que o OGST atualmente permite apenas a simulação de falhas de recursos computacionais;

- A inclusão do suporte ao uso de arquivos com dados coletados de ambientes reais (*traces*) que podem conter dados que informem, entre outros, a chegada de aplicações na grade, a disponibilidade de recursos (ex: percentual de CPU livre), falhas dos nós e dos enlaces de rede da grade.

Referências Bibliográficas

[1] User priorities in the condor system. http://www.cs.wisc.edu/condor/manual/-v6.1/3_5User_Priorities.html. Acessado em: 22/01/2009.

[2] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: killer application for the global grid? In *IPDPS: Proceedings of 14th IEEE International Parallel and Distributed Processing Symposium*, pages 520–528, 2000.

[3] N. K. S. and. Simulação de sistemas de comunicação Óptica baseada em simulação a eventos discretos. Master's thesis, Universidade Estadual de Campinas, 2007. http://libdigi.unicamp.br/document/?code=vtls000418966, Acessado em: 27/12/2008.

[4] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[5] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. *Lecture Notes in Computer Science*, pages 61–86, 2003.

[6] W. Bell, D. Cameron, A. Millar, L. Capozza, K. Stockinger, and F. Zini. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4):403, 2003.

[7] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[8] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. *International Conference on High-Performance Computing in the Asia-Pacific Region*, 1:283, 2000.

[9] R. Buyya and M. Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[10] A. Caminero, A. Sulistio, B. Caminero, C. Carrion, and R. Buyya. Extending GridSim with an architecture for failure detection. *International Conference on Parallel and Distributed Systems*, 2:1–8, Dezembro 2007.

[11] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. *Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, April 2008.

[12] S. Choi, M. Baik, C. Hwang, J. Gil, and H. Yu. Volunteer availability based fault tolerant scheduling mechanism in DG computing environment. In *3rd IEEE International Symposium on Network Computing and Applications (NCA'04)*, 2004.

[13] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.

[14] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. A. B. Silva, C. O. Barros, and C. Silveira. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. In *Proceedings of International Conference on Parallel Processing (ICPP'03)*, pages 407–416, 2003.

[15] D. Conde. Análise de padrões de uso em grades computacionais. Master's thesis, Departamento de Ciência da Computação - Universidade de São Paulo, Brasil, SP, Janeiro 2008.

[16] S. A. de Sousa, F. J. da Silva e Silva, and R. F. Lopes. A flexible fault-tolerance mechanism for the integrade grid middleware. In *ICNS '07: Proceedings of the Third International Conference on Networking and Services*, page 26, Washington, DC, USA, 2007. IEEE Computer Society.

[17] P. Domingues, P. Marques, and L. Silva. Resource usage of windows computer laboratories. *ICPPW*, 00:469–476, 2005.

[18] F. Dong and S. Akl. Scheduling algorithms for grid computing: state of the art and open problems. Technical report, School of Computing, Queen's University, Kingston, Ontario, Janeiro 2006.

[19] C. L. Dumitrescu and I. Foster. Gangsim: a simulator for grid scheduling studies. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid*, volume 2, pages 1151–1158, Washington, DC, USA, Maio 2005. IEEE Computer Society.

[20] H. El-Rewini, T. G. Lewis, and H. H. Ali. *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[21] A. Feinstein and H. Cannon. Constructs of Simulation Evaluation. *Simulation & Gaming*, 33(4):425, 2002.

[22] G. C. Filho and F. J. da Silva e Silva. Ogst: An opportunistic grid simulation tool. In *LAGrid 2008: 2nd International Workshop Latin American Grid*, Mato Grosso do Sul, Campo Grande, Agosto 2008.

[23] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*, chapter Globus: A Toolkit-Based Grid Architecture, pages 259–278. Morgan Kaufmann Publisher, 1999.

[24] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, 15(3):200–222, 2001.

[25] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[26] A. Goldchleger. *InteGrade: Um Sistema de Middleware para Computacao em Grade Oportunista*. PhD thesis, Universidade de São Paulo, 2004.

[27] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. Bezerra. InteGrade object-oriented Grid middleware leveraging the idle computing power of desktop machines. *Concurrency and Computation: Practice & Experience*, 16(5):449–459, 2004.

[28] R. T. Hays and M. J. Singer. *Simulation fidelity in training system design: Bridging the gap between reality and training.* New York: Springer-Verlag, 1989.

[29] D. Heyman and M. Sobel. Stochastic models in operations research. Vol. I: Stochastic processes and operating characteristics. 1982.

[30] F. Howell and R. McNab. SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling. *Proceedings of the First International Conference on Web-based Modelling and Simulation*, 1998.

[31] D. Hughes, G. Coulson, and J. Walkerdine. Free riding on gnutella revisited: the bell tolls? *Distributed Systems Online, IEEE*, 6(6), June 2005.

[32] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)*, 24(2):280–289, 1977.

[33] R. Jain. *The art of computer systems performance analysis*. Wiley, 1991.

[34] S. A. Jarvis, D. P. Spooner, H. N. L. C. Keung, J. Cao, S. Saini, and G. R. Nudd. Performance prediction and its use in parallel and distributed computing systems. *Future Generation Computer Systems*, 22(7):745–754, 2006.

[35] D. Kondo. Simboinc: A simulator for desktop grids and volunteer computing systems. Disponível em: http://simboinc.gforge.inria.fr. Acessado em: 22/01/2009.

[36] M. Lewis, A. Nguyen-Tuong, J. Karpovich, M. Morgan, and A. Ferrari. From Legion to Avaki: the persistence of vision. *Grid Computing: Making the Global Infrastructure a Reality*, 2003.

[37] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[38] Y. Liu. Grid scheduling. Technical report, Department of Computer Science, University of Iowa, 2004. http://www.cs.uiowa.edu/ yanliu/QE/QEreview.pdf, Acessado em: 22/01/2009.

[39] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous

computing systems. In *HCW '99: Proceedings of the Eighth Heterogeneous Computing Workshop*, page 30, Washington, DC, USA, 1999. IEEE Computer Society.

[40] M. W. Mutka and M. Livny. Profiling workstations' available capacity for remote execution. In *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 529–544, Amsterdam, The Netherlands, 1988. North-Holland Publishing Co.

[41] M. W. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Perform. Eval.*, 12(4):269–284, 1991.

[42] Object Management Group. *Trading Object Service Specification, Junho de 2000*, 1.0 edition, June 2000. OMG document formal/00-06-27.

[43] B. Quetier and F. Cappello. A survey of Grid research tools: simulators, emulators and real life platforms. Technical report, University of Paris South, INRIA, LRI, 2005.

[44] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *JSSPP*, pages 210–232, 2004.

[45] R. G. Sargent. Verification and validation of simulation models. In *WSC '05: Proceedings of the 37th conference on Winter simulation*, pages 130–143. Winter Simulation Conference, 2005.

[46] J. M. Schopf. Ten actions when grid scheduling: the user as a grid scheduler. *Grid resource management: state of the art and future trends*, pages 15–23, 2004.

[47] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan. Distributed job scheduling on computational grids using multiple simultaneous requests. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 359, Washington, DC, USA, 2002. IEEE Computer Society.

[48] A. Sulistio, C. S. Yeo, and R. Buyya. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software-Practice and Experience*, 34(7):653–673, 2004.

[49] M. Taufer, A. Kerstens, T. Estrada, D. Flores, and P. J. Teller. Simba: A discrete event simulator for performance prediction of volunteer computing projects. In *PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, pages 189–197, Washington, DC, USA, 2007. IEEE Computer Society.

[50] G. Team. The grid2003 production grid: Principles and practice. *High-Performance Distributed Computing, International Symposium on*, 0:236–245, 2004.

[51] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[52] D. Wright. Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with condor. In *Proceedings of the Linux Clusters: The HPC Revolution Conference*, Champaign - Urbana, Junho 2001.

[53] Y. Zhu. A survey on grid scheduling systems. Technical report, Computer Science Department, University of Science and Technology, Hong Kong, 2003.

A Medidas Estatísticas das Simulações Realizadas

Este apêndice apresenta um conjunto de medidas estatísticas calculadas a partir da média do tempo de conclusão das aplicações, considerando os 30 experimentos (30 conjuntos distintos de aplicações), para todas as simulações descritas ao longo deste trabalho. As medidas estatísticas são: média, valor mínimo, valor máximo, variância, desvio padrão e intervalo de confiança de 95%.

Tabela A.1: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma variação na taxa de chegada de aplicações e um ambiente livres de falhas composto por 100 máquinas. Referente à Figura 5.1.

Taxa de Chegada	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
0,0025	InteGrade	53,18	51,39	55,68	0,60	0,77	52,90	53,46
	MCT	27,05	26,00	27,82	0,12	0,35	26,93	27,18
	Minmin	27,06	26,00	27,82	0,12	0,35	26,93	27,18
	WQ	53,75	50,91	55,83	1,28	1,13	53,35	54,16
0,005	InteGrade	45,47	44,00	47,81	0,86	0,93	45,14	45,80
	MCT	27,43	26,44	28,09	0,16	0,40	27,29	27,57
	Minmin	27,44	26,45	28,10	0,16	0,40	27,30	27,58
	WQ	52,42	49,38	54,58	1,28	1,13	52,01	52,82
0,01	InteGrade	42,72	40,81	44,02	0,51	0,71	42,46	42,97
	MCT	28,25	27,17	29,14	0,15	0,39	28,11	28,39
	Minmin	28,26	27,18	29,15	0,15	0,39	28,12	28,40
	WQ	50,10	48,24	51,19	0,54	0,73	49,84	50,36
0,025	InteGrade	44,47	43,08	45,57	0,39	0,63	44,24	44,69
	MCT	32,42	31,36	33,23	0,29	0,53	32,23	32,61
	Minmin	32,45	31,39	33,30	0,28	0,53	32,26	32,64
	WQ	45,72	44,93	46,80	0,25	0,50	45,54	45,90
0,05	InteGrade	165,13	150,04	180,58	83,84	9,16	161,85	168,40
	MCT	103,27	95,19	113,80	22,58	4,75	101,57	104,97
	Minmin	103,24	94,94	113,86	22,48	4,74	101,54	104,94
	WQ	107,15	96,84	118,80	26,30	5,13	105,32	108,98
0,1	InteGrade	340,61	302,90	375,67	247,50	15,73	334,98	346,23
	MCT	216,97	209,42	227,99	22,32	4,72	215,28	218,66
	Minmin	216,96	209,07	228,47	22,60	4,75	215,26	218,66
	WQ	222,08	214,21	231,93	17,05	4,13	220,60	223,56
0,25	InteGrade	426,88	387,12	468,81	456,55	21,37	419,23	434,52
	MCT	285,20	276,80	298,65	24,39	4,94	283,43	286,96
	Minmin	285,14	276,61	298,63	24,51	4,95	283,36	286,91
	WQ	290,04	281,08	305,28	23,90	4,89	288,29	291,79
0,5	InteGrade	445,65	422,93	481,13	234,11	15,30	440,18	451,13
	MCT	307,99	295,56	318,26	29,80	5,46	306,04	309,94
	Minmin	307,94	295,50	318,28	30,07	5,48	305,97	309,90

Continua na próxima página

Tabela A.1 – continuação da página anterior

Taxa de Chegada	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
	WQ	312,64	298,43	320,70	26,46	5,14	310,80	314,48
1	InteGrade	458,98	434,62	490,86	257,36	16,04	453,24	464,72
	MCT	319,74	311,76	330,90	17,56	4,19	318,24	321,23
	Minmin	319,56	311,51	330,64	17,49	4,18	318,06	321,05
	WQ	325,24	316,48	334,95	18,24	4,27	323,71	326,77
5	InteGrade	471,81	419,95	500,79	277,85	16,67	465,85	477,78
	MCT	328,78	319,79	338,78	25,21	5,02	326,98	330,57
	Minmin	328,01	319,05	338,17	25,26	5,03	326,21	329,80
	WQ	334,37	322,95	347,17	25,93	5,09	332,55	336,19
10	InteGrade	473,20	436,35	502,64	270,49	16,45	467,31	479,08
	MCT	330,31	316,80	342,10	30,93	5,56	328,32	332,30
	Minmin	328,78	315,10	340,36	31,00	5,57	326,79	330,78
	WQ	335,54	326,69	346,68	24,90	4,99	333,76	337,33
20	InteGrade	470,61	445,48	505,87	272,68	16,51	464,71	476,52
	MCT	330,71	320,69	342,44	28,25	5,31	328,81	332,61
	Minmin	327,85	317,93	339,54	28,22	5,31	325,95	329,75
	WQ	336,08	327,27	349,47	29,53	5,43	334,14	338,03
30	InteGrade	473,16	422,38	506,03	296,54	17,22	467,00	479,32
	MCT	332,08	324,60	343,20	23,51	4,85	330,35	333,82
	Minmin	328,00	320,49	339,16	23,58	4,86	326,27	329,74
	WQ	337,55	326,19	350,48	34,24	5,85	335,45	339,64
60	InteGrade	469,21	430,48	504,71	305,72	17,48	462,96	475,47
	MCT	332,39	322,94	344,48	23,82	4,88	330,64	334,14
	Minmin	325,28	315,67	337,63	24,52	4,95	323,51	327,05
	WQ	338,82	328,13	347,60	18,96	4,35	337,26	340,37
120	InteGrade	471,56	450,61	504,03	207,78	14,41	466,40	476,71
	MCT	330,17	322,04	339,97	17,46	4,18	328,68	331,67
	Minmin	317,64	309,51	328,72	19,10	4,37	316,07	319,20
	WQ	335,65	324,24	344,33	22,63	4,76	333,94	337,35
360	InteGrade	469,37	441,10	499,35	219,37	14,81	464,07	474,67
	MCT	331,22	320,74	341,05	29,13	5,40	329,29	333,15
	Minmin	279,55	270,42	288,49	27,28	5,22	277,68	281,42
	WQ	336,30	324,76	346,32	24,72	4,97	334,52	338,08

Tabela A.2: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma variação na taxa de chegada de aplicações e um ambiente livres de falhas composto por 200 máquinas. Referente à Figura 5.2.

Taxa de Chegada	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
0,0025	InteGrade	52,71	48,68	54,15	1,74	1,32	52,24	53,18
	MCT	26,26	25,32	26,87	0,14	0,37	26,13	26,40
	Minmin	26,34	25,31	27,09	0,12	0,34	26,22	26,46
	WQ	50,78	48,90	52,06	0,54	0,73	50,52	51,04
0,005	InteGrade	45,32	43,85	46,26	0,27	0,52	45,13	45,50
	MCT	26,59	25,66	27,23	0,15	0,39	26,45	26,73
	Minmin	26,60	25,65	27,22	0,14	0,38	26,47	26,74
	WQ	50,25	48,31	52,19	1,03	1,02	49,89	50,61

Continua na próxima página

Tabela A.2 – continuação da página anterior

Taxa de Chegada	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
0,01	InteGrade	42,81	40,87	44,44	0,42	0,65	42,58	43,04
	MCT	26,99	25,98	27,80	0,13	0,36	26,86	27,12
	Minmin	27,00	25,99	27,81	0,13	0,36	26,87	27,13
	WQ	49,07	47,48	51,09	0,84	0,92	48,75	49,40
0,025	InteGrade	44,45	43,44	45,44	0,32	0,57	44,24	44,65
	MCT	28,02	27,39	28,60	0,12	0,35	27,90	28,14
	Minmin	28,03	27,39	28,60	0,12	0,34	27,90	28,15
	WQ	46,80	45,86	48,07	0,33	0,57	46,60	47,01
0,05	InteGrade	42,64	41,72	43,82	0,28	0,53	42,45	42,83
	MCT	30,55	29,73	31,46	0,21	0,46	30,39	30,72
	Minmin	30,57	29,76	31,48	0,21	0,46	30,41	30,73
	WQ	43,91	42,93	45,08	0,28	0,53	43,72	44,10
0,1	InteGrade	86,57	77,45	95,91	25,78	5,08	84,75	88,39
	MCT	62,01	58,91	65,94	2,86	1,69	61,41	62,62
	Minmin	61,76	57,91	66,10	4,06	2,02	61,04	62,48
	WQ	65,33	60,51	71,07	5,63	2,37	64,48	66,18
0,25	InteGrade	189,51	172,11	207,91	84,30	9,18	186,22	192,80
	MCT	122,84	118,22	129,84	6,91	2,63	121,90	123,78
	Minmin	122,82	118,23	129,75	6,78	2,60	121,89	123,75
	WQ	127,03	123,22	132,62	6,21	2,49	126,14	127,92
0,5	InteGrade	219,67	204,42	235,94	56,73	7,53	216,97	222,36
	MCT	143,91	137,53	148,74	7,84	2,80	142,91	144,91
	Minmin	143,88	137,61	148,79	7,71	2,78	142,89	144,88
	WQ	148,33	142,35	153,43	8,06	2,84	147,32	149,35
1	InteGrade	231,02	218,61	245,07	49,71	7,05	228,50	233,55
	MCT	154,57	150,88	159,82	4,16	2,04	153,84	155,30
	Minmin	154,49	150,86	159,66	4,12	2,03	153,76	155,21
	WQ	159,51	156,06	164,74	3,37	1,84	158,85	160,16
5	InteGrade	239,31	223,73	258,93	77,56	8,81	236,16	242,46
	MCT	162,76	158,26	167,72	6,14	2,48	161,87	163,64
	Minmin	162,36	157,79	167,34	6,28	2,51	161,46	163,26
	WQ	167,92	163,12	173,35	5,79	2,41	167,06	168,78
10	InteGrade	242,93	225,23	264,85	82,61	9,09	239,68	246,19
	MCT	164,01	157,46	169,66	7,76	2,79	163,01	165,00
	Minmin	163,19	156,34	169,00	7,91	2,81	162,18	164,20
	WQ	169,20	164,59	175,32	6,72	2,59	168,27	170,13
20	InteGrade	238,18	224,61	251,36	56,16	7,49	235,50	240,87
	MCT	164,44	159,39	170,35	7,18	2,68	163,48	165,40
	Minmin	162,84	157,67	168,65	7,11	2,67	161,88	163,79
	WQ	169,67	165,84	176,21	6,96	2,64	168,73	170,61
30	InteGrade	243,08	224,07	262,78	103,59	10,18	239,44	246,72
	MCT	165,27	161,46	171,14	6,02	2,45	164,39	166,15
	Minmin	162,92	159,35	168,63	5,75	2,40	162,06	163,78
	WQ	170,30	166,13	176,87	6,35	2,52	169,40	171,21
60	InteGrade	242,32	225,11	255,73	66,93	8,18	239,39	245,24
	MCT	165,54	160,80	171,57	5,96	2,44	164,67	166,42
	Minmin	161,23	156,33	167,49	6,22	2,49	160,34	162,12
	WQ	170,87	166,43	174,94	4,58	2,14	170,10	171,63
120	InteGrade	241,29	226,76	266,83	81,95	9,05	238,05	244,53
	MCT	164,44	160,54	169,34	4,48	2,12	163,68	165,19
	Minmin	155,60	151,31	161,18	4,97	2,23	154,80	156,40
	WQ	169,95	163,40	174,77	7,05	2,65	169,00	170,90
360	InteGrade	242,23	226,08	261,26	56,04	7,49	239,55	244,91
	MCT	164,98	159,64	169,96	7,52	2,74	164,00	165,96
	Minmin	132,61	127,86	137,51	6,08	2,47	131,72	133,49

Continua na próxima página

Tabela A.2 – continuação da página anterior

Taxa de Chegada	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
	WQ	170,31	164,79	176,80	8,22	2,87	169,28	171,33

Tabela A.3: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma variação na taxa de chegada de aplicações e um ambiente livres de falhas composto por 400 máquinas. Referente à Figura 5.3.

Taxa de Chegada	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
0,0025	InteGrade	45,82	43,73	50,86	1,77	1,33	45,34	46,29
	MCT	26,11	25,09	26,84	0,11	0,34	25,99	26,23
	Minmin	26,11	25,10	26,85	0,11	0,34	25,99	26,23
	WQ	56,51	53,42	58,62	1,45	1,20	56,08	56,94
0,005	InteGrade	45,15	41,75	46,29	0,77	0,88	44,84	45,47
	MCT	26,09	25,34	26,75	0,15	0,38	25,95	26,23
	Minmin	26,26	25,33	26,87	0,14	0,37	26,13	26,39
	WQ	56,00	53,45	58,19	1,55	1,24	55,56	56,45
0,01	InteGrade	42,83	41,09	44,12	0,37	0,61	42,61	43,04
	MCT	26,50	25,52	27,28	0,12	0,35	26,37	26,62
	Minmin	26,50	25,53	27,29	0,13	0,36	26,37	26,63
	WQ	55,35	52,73	58,23	1,80	1,34	54,87	55,83
0,025	InteGrade	44,47	43,39	45,40	0,31	0,56	44,27	44,67
	MCT	27,15	26,55	27,69	0,10	0,32	27,04	27,27
	Minmin	27,16	26,56	27,69	0,10	0,32	27,05	27,28
	WQ	53,19	51,43	55,30	0,71	0,84	52,89	53,49
0,05	InteGrade	42,67	41,73	43,74	0,26	0,51	42,49	42,85
	MCT	28,01	27,41	28,77	0,13	0,36	27,88	28,14
	Minmin	28,01	27,50	28,72	0,11	0,33	27,90	28,13
	WQ	50,83	49,26	52,38	0,49	0,70	50,58	51,08
0,1	InteGrade	43,86	42,68	45,07	0,32	0,57	43,66	44,06
	MCT	31,52	30,59	32,47	0,21	0,46	31,36	31,69
	Minmin	31,55	30,64	32,51	0,22	0,47	31,39	31,72
	WQ	47,71	46,67	48,91	0,23	0,48	47,53	47,88
0,25	InteGrade	91,63	84,95	100,52	13,68	3,70	90,31	92,95
	MCT	58,88	56,15	63,06	2,48	1,57	58,32	59,44
	Minmin	58,90	55,95	63,13	2,56	1,60	58,33	59,48
	WQ	64,89	62,01	69,41	2,88	1,70	64,28	65,50
0,5	InteGrade	128,26	119,58	138,54	19,33	4,40	126,68	129,83
	MCT	76,81	72,98	79,27	2,61	1,61	76,24	77,39
	Minmin	76,79	73,06	79,28	2,56	1,60	76,22	77,36
	WQ	81,93	78,65	84,86	2,60	1,61	81,36	82,51
1	InteGrade	147,16	142,18	153,06	9,37	3,06	146,07	148,26
	MCT	85,23	83,23	88,04	1,31	1,14	84,82	85,64
	Minmin	85,19	83,12	87,99	1,31	1,15	84,78	85,60
	WQ	91,24	89,32	93,36	1,14	1,07	90,85	91,62
5	InteGrade	153,45	146,94	161,94	20,54	4,53	151,82	155,07
	MCT	91,80	89,06	94,83	1,91	1,38	91,31	92,29
	Minmin	91,54	88,73	94,59	1,93	1,39	91,05	92,04
	WQ	98,75	96,16	101,62	1,87	1,37	98,26	99,24

Continua na próxima página

Tabela A.3 – continuação da página anterior

Taxa de Chegada	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
10	InteGrade	155,05	140,85	172,22	32,53	5,70	153,01	157,09
	MCT	92,80	88,99	96,02	2,63	1,62	92,22	93,38
	Minmin	92,28	88,50	95,48	2,57	1,60	91,71	92,85
	WQ	99,82	96,57	103,62	2,94	1,72	99,21	100,43
20	InteGrade	154,37	146,83	165,46	16,90	4,11	152,89	155,84
	MCT	93,27	90,12	96,61	2,40	1,55	92,72	93,83
	Minmin	92,28	89,13	95,54	2,37	1,54	91,73	92,83
	WQ	100,13	97,34	103,88	2,50	1,58	99,56	100,70
30	InteGrade	156,12	148,31	165,33	18,76	4,33	154,57	157,67
	MCT	93,87	91,70	97,37	1,99	1,41	93,37	94,38
	Minmin	92,38	90,16	95,74	1,88	1,37	91,89	92,87
	WQ	100,77	97,64	104,26	2,04	1,43	100,26	101,28
60	InteGrade	156,83	149,87	166,46	20,81	4,56	155,20	158,47
	MCT	94,11	91,23	97,43	1,95	1,40	93,61	94,61
	Minmin	90,90	88,11	94,19	1,96	1,40	90,39	91,40
	WQ	100,94	98,05	103,97	1,82	1,35	100,46	101,42
120	InteGrade	155,15	142,93	164,59	24,48	4,95	153,38	156,92
	MCT	93,49	91,29	96,45	1,52	1,23	93,05	93,93
	Minmin	84,03	81,38	87,34	2,10	1,45	83,51	84,55
	WQ	100,45	97,66	103,48	2,38	1,54	99,90	101,00
360	InteGrade	156,86	146,30	170,24	31,64	5,63	154,85	158,88
	MCT	93,80	90,77	96,65	2,43	1,56	93,25	94,36
	Minmin	82,29	79,53	85,19	2,10	1,45	81,77	82,81
	WQ	100,71	97,82	103,54	2,43	1,56	100,16	101,27

Tabela A.4: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma taxa de chegada de 0.025 aplicações por minuto, uma variação no intervalo ente falhas e o uso do mecanismo de reinício. Referente à Figura 5.4.

Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
1	InteGrade	68,46	58,43	85,78	41,60	6,45	66,16	70,77
	MCT	59,13	51,68	69,86	16,24	4,03	57,69	60,57
	Min-min	62,88	57,77	70,48	7,62	2,76	61,89	63,86
	WQ	56,29	51,87	59,13	3,64	1,91	55,61	56,97
2	InteGrade	50,91	47,86	54,53	2,95	1,72	50,30	51,53
	MCT	44,69	40,57	49,95	4,84	2,20	43,90	45,48
	Min-min	47,81	43,82	53,77	3,82	1,95	47,11	48,51
	WQ	49,94	48,64	51,49	0,53	0,73	49,68	50,19
4	InteGrade	47,28	45,39	51,69	1,37	1,17	46,86	47,70
	MCT	39,48	37,54	41,62	0,93	0,96	39,14	39,83
	Min-min	41,68	39,66	42,96	1,16	1,08	41,30	42,07
	WQ	48,12	46,79	49,09	0,29	0,54	47,93	48,32
6	InteGrade	46,01	43,97	48,43	1,28	1,13	45,60	46,41
	MCT	37,89	35,72	39,93	0,77	0,88	37,58	38,20
	Min-min	40,02	38,25	42,90	1,12	1,06	39,64	40,40
	WQ	47,39	46,62	48,61	0,19	0,44	47,23	47,54
8	InteGrade	45,70	44,11	47,50	0,77	0,88	45,39	46,02
	MCT	37,09	35,18	38,64	0,64	0,80	36,80	37,37
	Min-min	39,24	37,75	40,65	0,68	0,83	38,95	39,54
	WQ	47,12	46,08	48,24	0,33	0,57	46,91	47,32
10	InteGrade	45,13	42,73	47,05	0,74	0,86	44,82	45,44
	MCT	36,65	35,09	38,53	0,47	0,69	36,41	36,90
	Min-min	38,84	37,27	40,15	0,57	0,76	38,57	39,11
	WQ	46,89	46,10	47,56	0,20	0,45	46,73	47,05
12	InteGrade	45,30	43,73	46,61	0,59	0,77	45,02	45,58
	MCT	36,46	34,99	37,63	0,49	0,70	36,21	36,71
	Min-min	38,37	36,99	39,87	0,62	0,79	38,08	38,65
	WQ	46,77	45,68	47,84	0,29	0,53	46,58	46,96
14	InteGrade	45,25	43,65	46,92	0,74	0,86	44,94	45,56
	MCT	36,38	34,65	37,84	0,66	0,81	36,09	36,67
	Min-min	38,17	36,49	39,41	0,49	0,70	37,92	38,42
	WQ	46,63	45,82	47,39	0,19	0,44	46,47	46,79

Tabela A.5: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma taxa de chegada de 0.025 aplicações por minuto, uma variação no intervalo ente falhas e o uso do mecanismo de *checkpointing*. Referente à Figura 5.5.

Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
1	InteGrade	45,12	42,86	47,57	1,51	1,23	44,68	45,56
	MCT	41,00	38,31	43,15	1,49	1,22	40,57	41,44
	Min-min	44,53	41,52	47,90	2,17	1,47	44,00	45,06
	WQ	44,68	43,87	45,83	0,24	0,49	44,51	44,85
2	InteGrade	44,38	43,25	46,35	0,64	0,80	44,09	44,67
	MCT	38,11	36,38	39,61	0,90	0,95	37,77	38,45
	Min-min	40,41	38,52	42,41	1,18	1,09	40,02	40,80
	WQ	45,18	44,13	46,17	0,26	0,51	45,00	45,36
4	InteGrade	44,20	42,50	45,59	0,59	0,77	43,93	44,48
	MCT	36,57	34,91	38,05	0,68	0,82	36,28	36,87
	Min-min	38,75	37,15	41,02	0,63	0,79	38,46	39,03
	WQ	45,48	44,42	46,36	0,23	0,48	45,30	45,65
6	InteGrade	44,16	42,30	45,88	0,72	0,85	43,86	44,47
	MCT	36,05	34,76	37,61	0,55	0,74	35,78	36,31
	Min-min	38,16	36,49	39,35	0,50	0,71	37,91	38,42
	WQ	45,70	44,66	46,58	0,23	0,48	45,53	45,87
8	InteGrade	44,39	43,09	46,52	0,71	0,84	44,08	44,69
	MCT	35,80	34,52	36,67	0,38	0,62	35,58	36,02
	Min-min	37,89	36,24	39,09	0,49	0,70	37,64	38,14
	WQ	45,84	44,85	46,74	0,22	0,47	45,67	46,01
10	InteGrade	44,49	43,07	46,27	0,63	0,80	44,21	44,78
	MCT	35,72	34,42	36,90	0,46	0,68	35,48	35,96
	Min-min	37,68	36,45	38,84	0,44	0,66	37,44	37,92
	WQ	45,79	44,90	46,81	0,28	0,53	45,61	45,98
12	InteGrade	44,59	42,23	46,01	0,76	0,87	44,27	44,90
	MCT	35,62	34,30	36,76	0,39	0,62	35,40	35,84
	Min-min	37,62	36,12	38,88	0,54	0,73	37,36	37,88
	WQ	45,94	44,95	46,79	0,25	0,50	45,76	46,12
14	InteGrade	44,60	42,35	46,62	0,69	0,83	44,31	44,90
	MCT	35,58	34,36	36,88	0,43	0,66	35,34	35,81
	Min-min	37,44	36,15	38,66	0,44	0,66	37,21	37,68
	WQ	45,89	45,01	46,71	0,24	0,48	45,72	46,06

Tabela A.6: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma taxa de chegada de 0.25 aplicações por minuto, uma variação no intervalo ente falhas e o uso do mecanismo de reinício. Referente à Figura 5.6.

Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
1	InteGrade	427,90	407,31	451,56	137,89	11,74	423,70	432,10
	MCT	389,05	367,76	411,04	120,99	11,00	385,12	392,99
	Min-min	352,12	330,88	367,78	82,91	9,11	348,87	355,38
	WQ	414,47	393,01	435,00	103,29	10,16	410,83	418,11
2	InteGrade	401,27	369,68	435,63	245,82	15,68	395,66	406,88
	MCT	343,22	326,38	368,03	79,99	8,94	340,02	346,42
	Min-min	318,81	303,81	331,57	47,00	6,86	316,36	321,26
	WQ	363,65	346,19	386,24	88,93	9,43	360,27	367,02
4	InteGrade	442,45	425,50	497,02	284,82	16,88	436,41	448,49
	MCT	316,24	302,97	332,14	64,61	8,04	313,37	319,12
	Min-min	304,30	289,65	317,08	43,58	6,60	301,94	306,66
	WQ	331,47	319,75	345,74	33,88	5,82	329,39	333,55
6	InteGrade	470,27	431,55	501,84	394,67	19,87	463,16	477,38
	MCT	307,59	293,52	325,60	56,30	7,50	304,90	310,27
	Min-min	296,51	280,70	308,72	39,34	6,27	294,27	298,76
	WQ	321,00	304,30	337,54	46,43	6,81	318,56	323,43
8	InteGrade	457,54	401,82	512,47	657,48	25,64	448,37	466,72
	MCT	303,53	291,23	319,89	49,70	7,05	301,01	306,05
	Min-min	293,40	282,51	306,45	30,70	5,54	291,42	295,38
	WQ	314,78	305,72	328,96	25,20	5,02	312,98	316,57
10	InteGrade	445,41	414,06	478,60	309,19	17,58	439,12	451,70
	MCT	298,87	285,81	313,87	44,02	6,63	296,50	301,25
	Min-min	292,95	281,69	308,93	39,67	6,30	290,70	295,21
	WQ	311,35	298,96	328,61	54,51	7,38	308,71	313,99
12	InteGrade	439,97	380,35	491,72	602,60	24,55	431,19	448,76
	MCT	298,53	290,00	314,20	35,61	5,97	296,39	300,66
	Min-min	291,23	280,63	299,97	33,88	5,82	289,15	293,32
	WQ	308,32	293,90	326,42	40,87	6,39	306,03	310,60
14	InteGrade	424,69	387,81	472,04	382,16	19,55	417,69	431,68
	MCT	296,14	285,08	308,21	32,17	5,67	294,11	298,17
	Min-min	289,71	277,33	301,91	43,25	6,58	287,36	292,07
	WQ	307,33	295,47	316,81	27,93	5,28	305,44	309,23

Tabela A.7: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma taxa de chegada de 0.25 aplicações por minuto, uma variação no intervalo ente falhas e o uso do mecanismo de *checkpointing*. Referente à Figura 5.7.

Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
1	InteGrade	388,17	372,37	413,86	121,64	11,03	384,22	392,11
	MCT	343,32	326,09	366,35	125,69	11,21	339,31	347,33
	Min-min	311,50	294,31	332,42	62,39	7,90	308,67	314,33
	WQ	368,92	348,02	392,95	135,89	11,66	364,74	373,09
2	InteGrade	387,69	356,33	408,67	161,61	12,71	383,14	392,24
	MCT	319,14	306,03	338,32	62,32	7,89	316,32	321,97
	Min-min	300,89	288,15	311,26	34,95	5,91	298,77	303,00
	WQ	339,85	328,01	353,90	47,31	6,88	337,39	342,31
4	InteGrade	425,13	399,10	459,06	211,66	14,55	419,92	430,33
	MCT	305,36	292,97	316,46	43,95	6,63	302,98	307,73
	Min-min	295,26	285,18	306,27	45,20	6,72	292,86	297,67
	WQ	319,66	305,59	332,50	39,44	6,28	317,41	321,90
6	InteGrade	461,02	418,42	494,80	414,66	20,36	453,74	468,31
	MCT	299,32	285,24	309,19	31,93	5,65	297,30	301,34
	Min-min	291,90	281,32	303,89	43,19	6,57	289,55	294,25
	WQ	312,38	301,43	330,89	41,99	6,48	310,06	314,70
8	InteGrade	454,37	414,54	508,65	479,60	21,90	446,53	462,20
	MCT	295,82	286,87	308,11	30,32	5,51	293,85	297,79
	Min-min	289,94	274,63	304,79	42,32	6,51	287,61	292,27
	WQ	308,18	297,05	321,15	35,14	5,93	306,06	310,31
10	InteGrade	434,86	374,00	477,33	505,20	22,48	426,82	442,90
	MCT	294,79	284,55	308,41	37,74	6,14	292,59	296,98
	Min-min	290,20	278,94	304,32	41,16	6,42	287,91	292,50
	WQ	306,48	294,47	318,73	27,17	5,21	304,62	308,35
12	InteGrade	431,11	364,44	501,54	623,31	24,97	422,17	440,04
	MCT	292,93	284,68	302,16	28,39	5,33	291,02	294,83
	Min-min	288,50	278,51	299,59	30,18	5,49	286,53	290,46
	WQ	304,21	292,09	316,01	35,64	5,97	302,07	306,34
14	InteGrade	418,14	373,41	462,55	360,64	18,99	411,34	424,93
	MCT	292,23	283,05	305,64	34,70	5,89	290,12	294,34
	Min-min	288,38	275,44	297,86	39,36	6,27	286,14	290,63
	WQ	303,93	291,94	312,98	28,17	5,31	302,03	305,83

Tabela A.8: Medidas estatísticas geradas a partir da média do tempo de conclusão das aplicações, que foi obtida na avaliação das heurísticas de escalonamento, considerando: uma variação na taxa de chegada, uma variação no intervalo ente falhas e o uso do mecanismo de reinício. Referente à Figura 5.8.

Taxa de Chegada	Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
0,001	3600	MCT	31,14	30,26	32,36	0,27	0,52	30,96	31,32
		Min-min	32,99	32,07	34,10	0,30	0,55	32,80	33,19
	7200	MCT	29,49	28,83	30,20	0,16	0,40	29,35	29,64
		Min-min	31,20	30,43	32,22	0,19	0,44	31,04	31,36
	14400	MCT	28,61	28,03	29,36	0,12	0,35	28,49	28,74
		Min-min	30,13	29,32	31,03	0,14	0,37	30,00	30,27
	21600	MCT	28,25	27,67	28,80	0,11	0,33	28,13	28,37
		Min-min	29,75	29,11	30,45	0,12	0,34	29,63	29,87
	28800	MCT	28,11	27,65	28,77	0,09	0,30	28,01	28,22
		Min-min	29,54	29,08	30,50	0,10	0,31	29,43	29,65
	36000	MCT	28,01	27,53	28,51	0,08	0,28	27,91	28,11
		Min-min	29,42	28,93	29,96	0,09	0,30	29,31	29,53
	43200	MCT	27,89	27,35	28,59	0,09	0,30	27,78	27,99
		Min-min	29,33	28,76	29,95	0,10	0,32	29,22	29,45
	50400	MCT	27,86	27,39	28,41	0,08	0,29	27,76	27,97
		Min-min	29,29	28,71	30,05	0,10	0,31	29,18	29,40
0,0025	3600	MCT	31,44	30,08	32,64	0,30	0,55	31,24	31,63
		Min-min	33,15	31,43	34,18	0,33	0,57	32,94	33,35
	7200	MCT	29,76	28,45	30,50	0,19	0,44	29,60	29,92
		Min-min	31,36	30,48	32,16	0,19	0,44	31,21	31,52
	14400	MCT	28,85	27,78	29,69	0,16	0,40	28,71	28,99
		Min-min	30,33	28,99	30,94	0,17	0,41	30,19	30,48
	21600	MCT	28,51	27,38	29,14	0,15	0,38	28,37	28,64
		Min-min	29,92	28,84	30,95	0,16	0,39	29,78	30,07
	28800	MCT	28,29	27,14	29,21	0,15	0,39	28,15	28,43
		Min-min	29,78	28,79	30,47	0,13	0,36	29,65	29,91
	36000	MCT	28,20	27,21	29,05	0,14	0,38	28,07	28,34
		Min-min	29,65	28,55	30,41	0,14	0,37	29,52	29,78
	43200	MCT	28,14	27,10	28,98	0,14	0,37	28,00	28,27
		Min-min	29,60	28,44	30,27	0,12	0,35	29,48	29,73
	50400	MCT	28,08	26,98	28,81	0,15	0,38	27,95	28,22
		Min-min	29,51	28,71	30,27	0,11	0,33	29,39	29,63
0,005	3600	MCT	32,17	30,58	33,52	0,40	0,63	31,95	32,40
		Min-min	33,68	31,81	34,70	0,37	0,61	33,46	33,90
	7200	MCT	30,33	29,15	31,10	0,21	0,45	30,16	30,49
		Min-min	31,87	30,44	32,94	0,28	0,53	31,68	32,06
	14400	MCT	29,32	28,42	30,07	0,20	0,45	29,16	29,48
		Min-min	30,77	29,48	31,56	0,22	0,47	30,60	30,94
	21600	MCT	28,93	27,81	29,60	0,18	0,42	28,78	29,08
		Min-min	30,38	29,38	31,10	0,18	0,42	30,22	30,53
	28800	MCT	28,71	27,67	29,43	0,20	0,44	28,55	28,87
		Min-min	30,18	29,16	30,88	0,19	0,43	30,03	30,34
	36000	MCT	28,58	27,46	29,30	0,15	0,39	28,44	28,72
		Min-min	30,05	29,07	30,90	0,18	0,42	29,90	30,20
	43200	MCT	28,54	27,58	29,19	0,17	0,41	28,39	28,69
		Min-min	29,98	28,90	30,76	0,19	0,43	29,82	30,13
	50400	MCT	28,50	27,50	29,11	0,16	0,40	28,36	28,64
		Min-min	29,91	28,89	30,61	0,18	0,42	29,76	30,06

Continua na próxima página

Tabela A.8 – continuação da página anterior

Taxa de Chegada	Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
0,01	3600	MCT	33,71	31,72	35,13	0,36	0,60	33,50	33,92
		Min-min	35,50	33,99	37,41	0,50	0,71	35,25	35,75
	7200	MCT	31,57	30,44	32,54	0,23	0,48	31,40	31,75
		Min-min	33,25	32,35	34,67	0,37	0,60	33,04	33,47
	14400	MCT	30,33	29,16	31,38	0,22	0,47	30,17	30,50
		Min-min	31,90	30,68	32,92	0,21	0,45	31,74	32,07
	21600	MCT	29,87	28,63	30,77	0,18	0,42	29,72	30,02
		Min-min	31,44	30,24	32,46	0,22	0,47	31,27	31,61
	28800	MCT	29,70	28,58	30,47	0,16	0,40	29,55	29,84
		Min-min	31,20	30,05	32,14	0,18	0,43	31,04	31,35
	36000	MCT	29,52	28,36	30,35	0,17	0,41	29,37	29,67
		Min-min	31,04	29,99	32,10	0,20	0,44	30,88	31,20
	43200	MCT	29,43	28,48	30,49	0,17	0,41	29,29	29,58
		Min-min	30,94	29,80	31,83	0,18	0,43	30,78	31,09
	50400	MCT	29,40	28,24	30,33	0,19	0,44	29,25	29,56
		Min-min	30,87	29,92	31,88	0,18	0,42	30,72	31,02
0,025	3600	MCT	55,16	46,82	65,40	14,84	3,85	53,78	56,54
		Min-min	61,63	51,32	77,71	29,07	5,39	59,70	63,56
	7200	MCT	41,32	38,95	45,14	2,26	1,50	40,78	41,86
		Min-min	45,62	42,66	50,37	2,93	1,71	45,01	46,24
	14400	MCT	36,91	34,96	38,94	0,85	0,92	36,58	37,24
		Min-min	39,84	37,11	43,13	1,42	1,19	39,42	40,27
	21600	MCT	35,55	34,16	37,00	0,64	0,80	35,26	35,83
		Min-min	38,42	36,63	40,60	0,89	0,94	38,08	38,75
	28800	MCT	35,03	33,76	36,70	0,60	0,77	34,76	35,31
		Min-min	37,69	36,29	39,36	0,77	0,88	37,38	38,00
	36000	MCT	34,67	33,13	35,92	0,43	0,66	34,43	34,90
		Min-min	37,22	35,63	39,94	0,92	0,96	36,87	37,56
	43200	MCT	34,51	33,11	35,70	0,42	0,65	34,27	34,74
		Min-min	36,91	35,58	38,31	0,47	0,69	36,67	37,16
	50400	MCT	34,31	33,21	35,16	0,36	0,60	34,10	34,52
		Min-min	36,79	35,68	38,35	0,47	0,68	36,55	37,03
0,05	3600	MCT	212,57	185,11	242,41	143,15	11,96	208,29	216,85
		Min-min	196,53	182,75	219,03	66,37	8,15	193,61	199,44
	7200	MCT	166,17	150,02	178,39	43,59	6,60	163,80	168,53
		Min-min	157,95	145,75	176,73	61,54	7,84	155,15	160,76
	14400	MCT	137,26	127,94	153,23	49,23	7,02	134,75	139,77
		Min-min	135,11	121,82	150,63	51,20	7,16	132,55	137,67
	21600	MCT	126,71	116,70	143,99	36,04	6,00	124,57	128,86
		Min-min	126,26	113,43	138,53	50,13	7,08	123,73	128,80
	28800	MCT	121,67	111,10	132,22	34,80	5,90	119,56	123,79
		Min-min	120,19	110,98	132,30	28,04	5,30	118,29	122,08
	36000	MCT	117,92	106,99	130,38	29,23	5,41	115,99	119,85
		Min-min	117,26	106,30	129,84	31,43	5,61	115,26	119,27
	43200	MCT	115,21	107,44	127,59	30,55	5,53	113,23	117,19
		Min-min	115,31	107,39	125,38	19,17	4,38	113,75	116,88
	50400	MCT	113,60	102,94	124,57	30,19	5,49	111,63	115,56
		Min-min	115,01	107,26	127,59	25,84	5,08	113,19	116,82
0,1	3600	MCT	324,95	303,66	351,48	98,54	9,93	321,40	328,50
		Min-min	291,87	277,17	312,58	68,66	8,29	288,90	294,83
	7200	MCT	280,25	266,18	300,32	51,52	7,18	277,69	282,82
		Min-min	259,61	246,48	269,13	31,96	5,65	257,58	261,63
	14400	MCT	251,89	236,55	268,80	64,26	8,02	249,02	254,76

Continua na próxima página

Tabela A.8 – continuação da página anterior

Taxa de Chegada	Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
		Min-min	240,31	230,30	255,14	32,45	5,70	238,27	242,35
	21600	MCT	241,24	231,21	254,98	32,01	5,66	239,22	243,27
		Min-min	232,81	221,65	244,64	25,32	5,03	231,00	234,61
	28800	MCT	235,96	224,00	248,34	32,23	5,68	233,93	237,99
		Min-min	229,84	219,99	242,76	34,76	5,90	227,73	231,95
	36000	MCT	231,77	222,48	244,93	29,53	5,43	229,82	233,71
		Min-min	227,10	216,63	242,17	31,47	5,61	225,09	229,10
	43200	MCT	230,51	220,24	245,00	32,55	5,70	228,47	232,56
		Min-min	226,20	214,72	239,23	31,48	5,61	224,20	228,21
	50400	MCT	228,93	216,68	240,30	25,28	5,03	227,13	230,73
		Min-min	225,58	217,02	238,81	24,40	4,94	223,82	227,35
0,25	3600	MCT	391,34	370,35	419,12	108,35	10,41	387,62	395,07
		Min-min	352,11	335,86	371,40	97,50	9,87	348,57	355,64
	7200	MCT	345,48	324,95	363,62	97,29	9,86	341,95	349,01
		Min-min	323,11	308,31	335,03	46,95	6,85	320,66	325,56
	14400	MCT	318,88	308,03	337,65	48,62	6,97	316,39	321,38
		Min-min	301,73	291,68	318,45	46,80	6,84	299,28	304,18
	21600	MCT	307,77	296,30	321,68	36,77	6,06	305,60	309,94
		Min-min	297,89	287,55	311,49	33,84	5,82	295,80	299,97
	28800	MCT	303,40	293,31	320,59	43,12	6,57	301,05	305,75
		Min-min	294,38	282,33	309,87	44,80	6,69	291,99	296,78
	36000	MCT	301,14	289,52	317,65	44,53	6,67	298,76	303,53
		Min-min	294,11	285,55	308,13	25,25	5,03	292,31	295,91
	43200	MCT	297,88	287,64	319,28	53,85	7,34	295,26	300,51
		Min-min	293,11	283,45	305,46	21,41	4,63	291,45	294,76
	50400	MCT	297,67	287,66	309,72	32,41	5,69	295,64	299,71
		Min-min	292,26	282,68	306,85	27,51	5,25	290,38	294,14
0,5	3600	MCT	454,88	415,66	541,38	637,09	25,24	445,85	463,91
		Min-min	375,55	349,96	394,33	164,94	12,84	370,96	380,15
	7200	MCT	388,21	371,65	420,07	144,79	12,03	383,91	392,52
		Min-min	343,41	326,12	360,05	79,15	8,90	340,23	346,60
	14400	MCT	350,65	333,08	373,10	90,18	9,50	347,25	354,05
		Min-min	326,51	309,57	340,56	47,09	6,86	324,06	328,97
	21600	MCT	338,69	322,07	353,42	70,75	8,41	335,68	341,70
		Min-min	320,28	307,18	331,83	42,93	6,55	317,94	322,62
	28800	MCT	331,09	314,78	344,97	65,87	8,12	328,19	334,00
		Min-min	317,92	305,17	332,20	45,20	6,72	315,51	320,32
	36000	MCT	327,53	309,15	341,78	59,07	7,69	324,78	330,28
		Min-min	315,07	301,58	329,36	44,03	6,64	312,70	317,45
	43200	MCT	323,79	308,28	336,57	54,39	7,38	321,15	326,43
		Min-min	314,16	301,16	325,31	44,88	6,70	311,76	316,56
	50400	MCT	322,85	305,50	333,08	51,76	7,19	320,28	325,43
		Min-min	311,96	294,60	325,02	37,04	6,09	309,78	314,14
1	3600	MCT	515,14	462,88	576,53	870,29	29,50	504,58	525,70
		Min-min	387,52	368,81	408,55	78,28	8,85	384,35	390,68
	7200	MCT	417,81	396,10	452,66	257,46	16,05	412,07	423,55
		Min-min	357,36	342,93	368,96	47,23	6,87	354,90	359,81
	14400	MCT	371,93	353,56	388,06	78,18	8,84	368,77	375,10
		Min-min	334,54	325,31	347,41	35,09	5,92	332,42	336,66
	21600	MCT	354,33	332,68	372,94	77,03	8,78	351,19	357,47
		Min-min	331,10	320,77	345,68	32,02	5,66	329,08	333,13
	28800	MCT	348,48	335,70	358,91	39,85	6,31	346,22	350,74
		Min-min	326,19	312,64	341,85	34,27	5,85	324,09	328,28

Continua na próxima página

Tabela A.8 – continuação da página anterior

Taxa de Chegada	Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
	36000	MCT	343,26	330,82	360,95	55,66	7,46	340,59	345,93
		Min-min	324,86	314,82	335,35	30,22	5,50	322,90	326,83
	43200	MCT	338,80	324,17	349,74	41,86	6,47	336,49	341,12
		Min-min	323,49	315,03	335,74	21,36	4,62	321,84	325,15
	50400	MCT	337,93	324,97	359,10	57,00	7,55	335,23	340,64
		Min-min	322,15	313,21	334,05	21,82	4,67	320,48	323,82
5	3600	MCT	524,24	469,57	581,30	1191,27	34,51	511,89	536,59
		Min-min	390,21	372,45	401,50	50,23	7,09	387,68	392,75
	7200	MCT	426,29	395,90	467,63	230,61	15,19	420,85	431,72
		Min-min	354,39	339,85	371,86	56,80	7,54	351,69	357,09
	14400	MCT	375,73	354,20	407,10	83,00	9,11	372,47	378,99
		Min-min	336,60	323,35	350,01	46,34	6,81	334,17	339,04
	21600	MCT	364,50	343,47	389,38	87,88	9,37	361,14	367,85
		Min-min	329,38	318,78	341,61	35,27	5,94	327,25	331,50
	28800	MCT	356,56	341,40	373,95	75,32	8,68	353,45	359,66
		Min-min	324,52	314,88	337,06	30,56	5,53	322,54	326,50
	36000	MCT	351,56	338,28	364,87	53,89	7,34	348,94	354,19
		Min-min	323,52	315,71	337,22	29,05	5,39	321,59	325,45
	43200	MCT	348,96	337,24	365,55	53,51	7,31	346,34	351,58
		Min-min	321,40	310,91	334,01	24,58	4,96	319,63	323,17
	50400	MCT	345,81	333,31	363,68	47,27	6,88	343,35	348,27
		Min-min	320,14	312,12	332,29	21,87	4,68	318,46	321,81
10	3600	MCT	536,96	481,78	577,46	632,49	25,15	527,97	545,96
		Min-min	379,96	361,29	398,49	84,64	9,20	376,67	383,26
	7200	MCT	427,92	398,87	459,80	315,02	17,75	421,57	434,28
		Min-min	346,44	334,15	360,41	53,62	7,32	343,82	349,06
	14400	MCT	377,15	357,90	390,89	52,78	7,27	374,55	379,75
		Min-min	324,42	312,17	336,03	43,04	6,56	322,08	326,77
	21600	MCT	365,67	338,93	382,17	95,55	9,77	362,17	369,17
		Min-min	319,17	308,33	332,65	49,50	7,04	316,65	321,69
	28800	MCT	358,12	334,97	374,74	89,61	9,47	354,73	361,50
		Min-min	314,41	300,62	326,40	37,82	6,15	312,21	316,61
	36000	MCT	352,51	337,48	365,78	57,54	7,59	349,80	355,23
		Min-min	311,53	301,77	322,16	29,56	5,44	309,59	313,48
	43200	MCT	348,62	336,98	367,00	60,48	7,78	345,84	351,41
		Min-min	309,05	295,36	319,12	32,07	5,66	307,03	311,08
	50400	MCT	347,73	336,14	361,50	45,03	6,71	345,33	350,13
		Min-min	307,76	294,96	320,24	36,82	6,07	305,59	309,94
20	3600	MCT	525,73	458,29	597,63	1359,41	36,87	512,54	538,92
		Min-min	366,64	346,50	393,93	147,67	12,15	362,30	370,99
	7200	MCT	426,62	390,96	470,53	350,66	18,73	419,92	433,32
		Min-min	326,01	301,75	341,78	65,15	8,07	323,12	328,89
	14400	MCT	382,63	356,71	402,64	123,08	11,09	378,66	386,60
		Min-min	301,04	287,21	316,27	53,41	7,31	298,43	303,66
	21600	MCT	364,48	345,20	383,29	86,22	9,29	361,16	367,80
		Min-min	287,93	278,85	301,81	27,41	5,24	286,05	289,80
	28800	MCT	358,51	338,46	371,03	58,20	7,63	355,78	361,24
		Min-min	282,87	271,16	303,25	44,69	6,69	280,47	285,26
	36000	MCT	353,73	340,52	375,92	71,61	8,46	350,70	356,76
		Min-min	280,24	265,61	297,25	47,39	6,88	277,78	282,70
	43200	MCT	351,40	338,19	371,19	48,70	6,98	348,90	353,90
		Min-min	277,77	267,42	293,00	43,18	6,57	275,42	280,12
	50400	MCT	348,83	332,78	362,55	50,25	7,09	346,29	351,36

Continua na próxima página

Tabela A.8 – continuação da página anterior

Taxa de Chegada	Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
		Min-min	275,89	261,64	291,85	37,80	6,15	273,69	278,09
30	3600	MCT	535,52	492,38	640,46	1361,49	36,90	522,32	548,72
		Min-min	371,12	342,92	402,74	135,47	11,64	366,95	375,28
	7200	MCT	426,92	396,83	455,01	173,61	13,18	422,21	431,64
		Min-min	330,90	308,83	344,73	68,49	8,28	327,94	333,86
	14400	MCT	382,35	361,18	405,88	89,87	9,48	378,96	385,75
		Min-min	302,29	285,48	319,86	64,29	8,02	299,42	305,16
	21600	MCT	365,82	352,64	378,85	41,72	6,46	363,51	368,14
		Min-min	293,86	281,59	309,04	45,62	6,75	291,44	296,28
	28800	MCT	359,83	343,64	370,66	41,13	6,41	357,53	362,12
		Min-min	287,23	273,32	303,65	48,16	6,94	284,74	289,71
	36000	MCT	355,43	338,20	373,44	62,40	7,90	352,60	358,26
		Min-min	283,57	273,25	297,56	35,59	5,97	281,43	285,70
	43200	MCT	352,38	341,46	365,01	50,37	7,10	349,84	354,92
		Min-min	281,76	270,39	302,86	47,21	6,87	279,30	284,21
	50400	MCT	350,26	338,04	364,72	49,74	7,05	347,74	352,78
		Min-min	279,85	262,94	293,91	40,55	6,37	277,57	282,13
60	3600	MCT	530,93	478,75	597,81	623,64	24,97	522,00	539,87
		Min-min	380,58	359,86	398,79	72,13	8,49	377,54	383,62
	7200	MCT	428,77	403,60	461,79	233,04	15,27	423,30	434,23
		Min-min	340,01	326,38	364,02	77,25	8,79	336,87	343,16
	14400	MCT	383,55	365,88	409,44	100,96	10,05	379,95	387,14
		Min-min	313,24	303,38	329,92	43,98	6,63	310,86	315,61
	21600	MCT	367,29	345,62	386,92	90,96	9,54	363,87	370,70
		Min-min	302,60	290,05	315,90	47,49	6,89	300,14	305,07
	28800	MCT	358,14	337,47	380,06	91,16	9,55	354,72	361,56
		Min-min	296,31	286,70	308,14	29,66	5,45	294,36	298,26
	36000	MCT	355,83	343,86	373,53	52,77	7,26	353,23	358,43
		Min-min	292,32	282,53	301,97	34,82	5,90	290,21	294,44
	43200	MCT	352,05	340,76	366,66	45,74	6,76	349,63	354,47
		Min-min	291,75	281,95	304,96	30,08	5,48	289,79	293,71
	50400	MCT	351,60	339,69	366,65	54,50	7,38	348,96	354,24
		Min-min	288,59	278,57	296,98	24,98	5,00	286,81	290,38
120	3600	MCT	526,20	489,76	593,41	489,93	22,13	518,28	534,12
		Min-min	382,80	360,02	407,09	121,54	11,02	378,85	386,74
	7200	MCT	419,64	394,40	440,94	165,20	12,85	415,04	424,24
		Min-min	345,59	331,48	358,54	54,19	7,36	342,95	348,22
	14400	MCT	379,77	360,86	407,65	67,20	8,20	376,83	382,70
		Min-min	320,48	306,19	333,66	46,97	6,85	318,03	322,93
	21600	MCT	365,64	351,26	378,01	48,17	6,94	363,15	368,12
		Min-min	311,52	299,79	324,64	40,08	6,33	309,26	313,79
	28800	MCT	358,23	345,18	374,10	54,70	7,40	355,58	360,87
		Min-min	305,88	294,76	315,25	27,60	5,25	304,00	307,76
	36000	MCT	353,47	343,09	372,45	61,46	7,84	350,67	356,28
		Min-min	303,55	294,11	315,28	25,28	5,03	301,75	305,35
	43200	MCT	349,17	338,53	363,19	34,40	5,86	347,07	351,27
		Min-min	301,19	291,49	316,03	34,75	5,90	299,08	303,30
	50400	MCT	347,72	338,49	361,30	32,33	5,69	345,68	349,75
		Min-min	299,20	288,01	312,34	34,47	5,87	297,10	301,30
360	3600	MCT	529,39	483,40	597,66	849,51	29,15	518,96	539,82
		Min-min	367,91	344,74	386,75	103,86	10,19	364,26	371,56
	7200	MCT	424,59	392,20	464,43	219,74	14,82	419,29	429,90
		Min-min	330,47	317,71	345,59	54,77	7,40	327,82	333,12

Continua na próxima página

Tabela A.8 – continuação da página anterior

Taxa de Chegada	Intervalo entre Falhas	Algoritmo	Média	Mínimo	Máximo	Variância	Desvio Padrão	Intervalo de Confiança	
	14400	MCT	379,68	360,49	414,44	187,97	13,71	374,77	384,58
		Min-min	308,48	292,55	319,71	38,41	6,20	306,26	310,69
	21600	MCT	364,04	341,02	393,98	121,48	11,02	360,10	367,99
		Min-min	299,72	288,38	312,41	39,12	6,25	297,48	301,96
	28800	MCT	357,93	337,30	378,26	66,71	8,17	355,01	360,85
		Min-min	296,62	282,31	313,73	50,08	7,08	294,09	299,15
	36000	MCT	353,85	338,29	364,61	47,76	6,91	351,38	356,32
		Min-min	293,18	280,38	302,15	29,74	5,45	291,23	295,13
	43200	MCT	352,31	338,40	369,00	58,11	7,62	349,59	355,04
		Min-min	291,33	280,65	301,76	35,54	5,96	289,20	293,46
	50400	MCT	348,90	337,63	363,54	49,10	7,01	346,39	351,40
		Min-min	291,71	280,94	301,16	31,41	5,60	289,70	293,71