

Denison Menezes

GERAÇÃO DE ALGORITMOS DE ESCALONAMENTO PARA SIMULAÇÃO DE GRADES COMPUTACIONAIS

São José do Rio Preto
2012

Denison Menezes

GERAÇÃO DE ALGORITMOS DE ESCALONAMENTO PARA SIMULAÇÃO DE GRADES COMPUTACIONAIS

Dissertação de Mestrado elaborada junto
ao Programa de PósGraduação em Ciência
da Computação – Área de Concentração
em Sistemas de Computação, como parte
dos requisitos para a obtenção do título de
Mestre em Ciência da Computação.

Orientador: Aleardo Manacero Junior

São José do Rio Preto
2012

Denison Menezes

GERAÇÃO DE ALGORITMOS DE ESCALONAMENTO PARA SIMULAÇÃO DE
GRADES COMPUTACIONAIS

Dissertação apresentada para obtenção do título de Mestre em Ciência da Computação, área de Concentração em Sistemas de Computação junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociência, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Campus de São José do Rio Preto.

Banca Examinadora:

Prof^o. Dr^o. 
Aleardo Manacero Júnior
UNESP – São José do Rio Preto
Orientador

Prof^o. Dr^o. 
Mario Antonio Ribeiro Dantas
Universidade Federal de Santa Catarina

Prof^o. Dr^o. 
Marcos Antônio Cavalcanti
UNESP – Bauru

A liberdade não tem preço, a mera possibilidade de obtê-la já vale a pena.

Isaac Asimov

Agradecimentos

Neste momento gostaria de agradecer a todos que estiveram envolvidos, diretamente ou indiretamente com o desenvolvimento desse trabalho, aos amigos de longa data, as amizades construídas durante o mestrado, e os familiares que me apoiaram e incentivaram durante todo o trajeto até aqui trilhado.

Não é possível mencionar todos que de alguma forma foram fundamentais para realização deste trabalho, mas devo agradecer meu orientador, Professor Doutor Aleardo Manacero Junior por toda atenção e suporte dado sempre que necessitei. Por sua paciência em solucionar as dificuldades encontradas e meus erros, conduzindo para uma saída sempre que me deparava com um obstáculo. Devo essa dissertação a ele pela amizade e atenção.

Agradeço também de forma muito especial a Professora Doutora Renata Spolon Lobato. A ela devo, não apenas incontáveis indicações de preciosas leituras, mas, sobretudo por todo apoio fornecido, fazendo às vezes de orientador durante a ausência do professor Aleardo.

Agradeço ao aporte financeiro concedido pela Pró-Reitoria de Pós-Graduação (PROPG) da UNESP, e pela infraestrutura necessária presente no laboratório de Grupo de Sistemas Paralelos e Distribuídos (GSPD) da UNESP campus de São José do Rio Preto.

Agradeço aos amigos do GSPD pela contribuição no projeto, ajudas e sugestões especialmente ao Diogo Tavares, Aldo Ianelo Guerra, Silas Fernandes, Paulo Henrique Oliveira, Gabriel Covello Furlanetto, Rafael Souza Stabile, Gabriel Saraiva, e o Matheus Della Croce Oliveira. Aos funcionários da UNESP pela atenção e esmero com que me atenderam durante a realização da pesquisa.

Agradeço aos meus pais, José Menezes e Maria Benedita de Menezes, e meus irmãos pelo afeto, solidariedade e compreensão. A namorada e companheira, Aline Pereira Paes, pelo incentivo e por suportar a distância e ausência devido à dedicação na pesquisa.

Resumo

A crescente necessidade por poder computacional, unida com o progresso atingido nos computadores pessoais e redes de interconexão, fez surgir diversas propostas, tais como grades computacionais, para tornar a computação de alto desempenho mais barata e acessível. Como contraponto, a maior acessibilidade aos recursos para computação de alto desempenho oferecida pelas grades, criou um universo de usuários tipicamente não especialistas em computação paralela, aumentando a demanda por ferramentas de avaliação de desempenho e de apoio ao desenvolvimento de sistemas. Visando criar uma ferramenta de simulação de grades com facilidade de uso, mesmo para usuários não especialistas em programação, vem sendo desenvolvido o simulador de grades computacionais iSPD (*iconic Simulator of Parallel and Distributed systems*). Como o escalonamento de tarefas é essencial na computação distribuída, o iSPD necessitava de uma interface para a especificação de escalonadores no ambiente simulado que mantivesse os conceitos de fácil modelagem. Este trabalho de pesquisa apresenta a proposta e desenvolvimento de técnicas que permitam que o usuário do iSPD modele novas políticas de escalonamento de forma automatizada e simples. Estas técnicas foram aplicadas em um novo componente capaz de interpretar algoritmos de escalonamento especificados pelo usuário adicionando-os a um banco de algoritmos pré-disponibilizados.

Palavras-chave: Simulação, escalonamento, computação em grade

Abstract

The increasing demand for more computing power, associated with the progress in personal computers and interconnection networks, culminated in proposals to make high performance computing cheaper and more accessible such as computer grids. The greater accessibility to resources for high performance computing offered by grids created a universe of users lacking of parallel programming expertise, increasing the demand for tools for performance evaluation and systems development support. Aiming for the development of a grid performance evaluation tool that could be easy to use, even for people not expert in parallel programming, iSPD (iconic Simulator of Parallel and Distributed systems) has been developed. Since task scheduling in distributed systems is a critical process, iSPD needed an easy approach to specify scheduling policies for a grid. This work presents the development, and its associated results, of a set of techniques that allow the iSPD's user to model scheduling policies in an automated and simple way. These techniques were applied to a new component capable of interpreting scheduling algorithms specified by a user, adding them to a pre-built algorithms database. Results achieved with this component show that the used approach is right.

Keywords: Simulation, scheduling, grid computing

Sumário

Lista de Figuras	x
Lista de Tabelas	xii
1 Introdução	13
1.1 Motivação	14
1.2 Objetivo	16
1.3 Organização do texto	16
2 Simuladores de grades computacionais	18
2.1 Grade computacional	18
2.1.1 Tipos de Grades Computacionais	21
2.1.2 Escalonadores	22
2.2 Simuladores de grades computacionais	24
2.2.1 SimGrid	24
2.2.2 GridSim	27
2.2.3 OptorSim	30
2.2.4 GangSim	32
2.3 iSPD	33
2.3.1 Criação de modelos	34
2.4 Comparação entre os simuladores de grades computacionais	36

2.5	Considerações finais	39
3	iSPD	40
3.1	Arquitetura do iSPD	40
3.1.1	Interface gráfica	41
3.1.2	Interpretador de modelos	42
3.1.3	Motor de simulação	44
3.2	Considerações finais	46
4	Geração e gerenciamento de escalonadores	47
4.1	Módulo de escalonadores	47
4.1.1	Caso de uso	48
4.1.2	Atividades	50
4.2	Geração automatizada de escalonadores	51
4.2.1	Interface gráfica do gerador de Escalonadores	52
4.2.2	Gramática de geração de algoritmos de escalonamento	53
4.2.3	Gerenciamento dos escalonadores no iSPD	57
4.3	Especificação do Motor de simulação	59
4.3.1	Simulador de eventos discretos	61
4.3.2	Modelo de filas	62
4.3.3	Mestre-escravo e o escalonamento	64
4.4	Implementação do módulo de escalonadores	65
4.5	Implementação do Motor de simulação	67
4.5.1	Simulador de eventos discretos	68
4.5.2	Modelo de filas	69
4.5.3	Escalonamento	69
4.6	Considerações finais	71

5	Ambiente e resultados experimentais	73
5.1	Validação do motor de simulação	73
5.1.1	Ambiente computacional	74
5.1.2	Algoritmos de escalonamento	75
5.1.3	Resultados	77
5.2	Verificação da geração de escalonadores	79
5.2.1	Modelo simulado	80
5.2.2	Algoritmos de escalonamento	80
5.2.3	Resultados	81
5.3	Avaliação entre política estática e dinâmica	83
5.3.1	Ambiente computacional e cargas de trabalho	84
5.3.2	Algoritmo de escalonamento	85
5.3.3	Resultados	86
5.4	Eficiência do simulador	88
5.4.1	Resultados	89
5.5	Considerações finais	91
6	Considerações Finais	92
6.1	Conclusões	92
6.2	Direções futuras	94
6.3	Publicações	95
	Referências Bibliográficas	96

Lista de Figuras

2.1	Grade computacional (Anglano et al., 2008)	20
2.2	GRAS (Quinson, 2006)	26
2.4	Visão de um modulo ainda não totalmente configurado	35
2.5	Janela de configuração das cargas de trabalho	36
3.1	Diagrama conceitual da plataforma de simulação (Guerra et al., 2010) .	41
3.2	Interface principal do iSPD	42
3.3	Interpretadores internos do iSPD	43
3.4	Interpretador externo	44
3.5	Primeiro protótipo do motor de simulação	45
4.1	Diagrama caso de uso UML	49
4.2	Diagrama de atividades UML	50
4.3	Interface do gerador de escalonadores	53
4.4	Gramática do gerador de escalonadores	55
4.5	Algoritmo Workqueue	57
4.6	Etapas para adição de escalonadores no iSPD	58
4.7	Versão atual do motor de simulação	60
4.8	Processo de simulação	62
4.9	Paradigma mestre-escravo	64

4.10	Diagrama de pacotes UML	65
4.11	Interfaces do pacote "escalonador"	66
4.12	Diagrama de Classe Gerenciador de Escalonadores	68
4.13	Diagrama UML do Simulador de eventos discretos	69
4.14	Diagrama UML dos servidores do modelo de filas	70
4.15	Diagrama UML dos clientes do modelo de filas	71
4.16	Escalonamento no motor de simulação	72
5.1	Cluster GSPD (GSPD, 2012)	75
5.2	Aplicação para o cluster	76
5.5	Segundo ambiente computacional	80
5.6	Algoritmo Round-Robin	81
5.7	<i>Dynamic</i> FPLTF	81
5.9	Terceiro ambiente computacional	84
5.11	Utilização dos recursos da grade pelos usuários ao longo da simulação . .	87

Lista de Tabelas

2.1	Comparação entre simuladores de grades computacionais.	37
5.1	Resultados da simulação variando número de tarefas	78
5.2	Total de tarefas executadas em cada recurso	83
5.3	Total de tarefas executadas em cada recurso (a) Dinâmico. (b) Estático.	88
5.4	Tempo de execução da simulação (b) Dinâmico. (a) Estático.	90
5.5	Tempo de execução da simulação no SimGrid (Estático)	90

Capítulo 1

Introdução

Há uma crescente necessidade por sistemas computacionais de alto desempenho. Notoriamente tal tendência decorre do anseio em solucionar problemas de grande complexidade e com volumes de dados cada vez maiores. Muitas aplicações do meio científico (como simulações da astrofísica, genoma, bioinformática, entre outras), e da indústria (por exemplo animação e comércio eletrônico), necessitam de alto poder computacional, e geram grande volume de dados que precisam ser armazenados e analisados.

Devido as características e limitações das arquiteturas sequenciais surgiram sistemas que utilizam a computação paralela e distribuída, tendo como meta aumentar o poder de processamento para atender a necessidade por computação de alto desempenho (Branco, 2004). Contudo, sistemas como supercomputadores são extremamente caros, com valores na casa dos milhões de dólares, como o adquirido pelo INPE (Instituto Nacional de Pesquisas Espaciais), por cerca de R\$ 50 milhões (FAPESP, 2010). Agravando este quadro, mesmo estes grandes sistemas podem se tornar obsoletos à medida que a tecnologia evolui, como o primeiro supercomputador Cray-1TM, que custava \$8.8 milhões com desempenho de 160 megaflops (Cray, 2012), que é um poder computacional baixo comparado a um processador atual, como o Intel Xeon E5620, que atinge 38,4 gigaflops (Intel, 2012).

O alto valor investido nos equipamentos existentes nas instituições, em conjunto com o aumento da velocidade das redes e do poder de processamento dos computadores pessoais, culminou no surgimento de sistemas distribuídos com objetivo de aumentar o poder de processamento de forma mais econômica. A partir da computação distribuída foram criados os conceitos e tecnologias utilizados nas grades computacionais. Dentro desses conceitos pode-se citar projetos como o Condor (Litzkow et al., 1988), desenvolvido para utilizar os recursos subutilizados de uma instituição, e o I-WAY (Foster et al., 1996), que cria um sistema unindo supercomputadores de diversas organizações. Essas evoluções levaram a um aumento na disponibilidade de computação de alto desempenho.

Por outro lado, a maior acessibilidade aos recursos necessários para prover computação de alto desempenho acaba criando um espectro de usuários que, em sua maioria, não podem ser considerados especialistas em programação de alto desempenho. Estes usuários não especialistas acabam por criar uma demanda por ferramentas de auxílio ao desenvolvimento e para a avaliação de desempenho de sistemas paralelos e distribuídos.

Neste cenário a simulação é uma ferramenta importante, tanto para auxiliar no desenvolvimento de sistemas paralelos, como no desenvolvimento de aplicações, ou ainda avaliar uma configuração ou política de escalonamento. Este trabalho é baseado em simulação visando avaliação de desempenho de sistemas distribuídos, com foco no escalonamento de tarefas em grades e como esses escalonadores podem ser modelados de modo simples e fácil.

1.1 Motivação

Desempenho é um tema importante durante o projeto, desenvolvimento, e montagem de um sistema computacional. Dessa forma é necessário realizar avaliações quantitativas de desempenho do sistema computacional durante várias fases da sua vida útil.

Estas avaliações podem ter diversos objetivos, como indicar a vantagem de um novo sistema, ou verificar se o sistema atende os requisitos de uma aplicação. Ao analisar o desempenho de uma grade computacional tem-se que o escalonador de tarefas é um elemento crítico, visto que ele influencia fortemente o desempenho de todo o sistema, assim como na satisfação do usuário. No entanto, realizar o escalonamento nestes sistemas não é uma tarefa trivial. Paula (2009) ressalta que apesar do progresso nas ferramentas e infraestrutura para uso transparente de recursos de grades computacionais, ainda há muita dependência do usuário na escolha dos recursos para a execução de suas aplicações.

O desempenho de um sistema pode ser avaliado de três formas: modelagem analítica, simulação e *benchmarking* (Jain, 1991). A modelagem analítica apresenta limitações quanto a confiabilidade do modelo, e devido às características da computação em grade este modelo se torna inviável. O *benchmarking* apresenta limitações de aplicabilidade e custos associados, pois necessita do sistema real. Dessa forma a simulação apresenta atrativos para realizar a avaliação de desempenho de um sistema distribuído heterogêneo, e se adéqua melhor as necessidades do estudo de escalonadores, como a modificação de forma rápida da plataforma computacional e cargas de trabalhos.

Portanto a simulação é uma importante ferramenta para realizar estudos de políticas para escalonamento de tarefas em grades computacionais, com diversos simuladores propostos e desenvolvidos inicialmente para esta finalidade, tais como SimGrid (Casanova et al., 2008), GridSim (Buyya and Murshed, 2002), GangSim (GangSim, 2012), OptorSim (OptorSim, 2012) e Bricks (Takefusa and Matsuoka, 2000). Contudo, estes simuladores apresentam dificuldades quanto à criação de modelos, e implementação de escalonadores, fazendo com que o usuário necessite conhecer muitos detalhes da implementação da ferramenta que será utilizada, para então poder realizar a codificação do escalonador que deseja analisar.

1.2 Objetivo

Como pode ser inferido da discussão anterior, um dos componentes essenciais em grades computacionais é o escalonador de tarefas. Felizmente, o estudo de políticas de escalonamento e seu impacto numa grade é uma área que se ajusta bem com simulação. Isso porque através da simulação é possível realizar mudanças na topologia do sistema, carga processada, e outros parâmetros, e averiguar os resultados de forma muito rápida e prática.

Infelizmente, boa parte dos simuladores de grade existentes não apresenta uma interface de fácil uso para a modelagem, tanto do sistema, como de seus escalonadores. Para reverter esse cenário o iSPD (*iconic Simulator of Parallel and Distributed systems*), simulador desenvolvido no Grupo de Sistemas Paralelos e Distribuídos da UNESP (GSPD, 2012), é baseado em modelagem por ícones. O trabalho aqui apresentado tem como objetivo permitir a inserção de políticas de escalonamento de forma simples e automatizada no iSPD.

Com esse objetivo se fez a modelagem e desenvolvimento de um agente capaz de interpretar algoritmos de escalonamento modelados pelo usuário, fazendo a sua adição aos escalonadores disponíveis no iSPD.

1.3 Organização do texto

Esta dissertação está dividida em seis capítulos, incluindo este que se encerra. No capítulo 2 são discutidos os conceitos essenciais para o projeto, como a definição das Grades Computacionais, também são apresentados os principais simuladores de grade computacional, concluindo com uma comparação entre eles e o simulador alvo do trabalho. O capítulo 3 apresenta a arquitetura da primeira versão do iSPD.

No capítulo 4 é apresentada a modelagem e implementação do módulo responsável por escalonamentos dentro do iSPD, que é o foco do presente trabalho.

O capítulo 5 apresenta alguns testes realizados para validar o módulo de escalonamento e o gerador de escalonadores. Concluindo com o capítulo 6, que apresenta uma análise final sobre o trabalho de pesquisa realizado e propostas de trabalhos futuros.

Capítulo 2

Simuladores de grades computacionais

Este capítulo apresenta uma revisão sobre os temas essenciais para o desenvolvimento deste trabalho de pesquisa. Inicialmente é discutida a definição e histórico da computação em grade, visando esclarecer algumas de suas principais características, e o tipo de usuário que se beneficia destes sistemas. Posteriormente apresenta-se o estado da arte no campo de simulação de grades computacionais, definindo as principais características das ferramentas de simulação de grades mais representativas. Dessa forma foi possível identificar as virtudes e deficiências de cada ferramenta, contribuindo para identificar as funcionalidades que devem estar presentes em um sistema de simulação de grade computacional, como o iSPD. O capítulo é concluído com uma descrição do iSPD e uma comparação entre ele e os demais simuladores.

2.1 Grade computacional

As grades computacionais surgiram na década de 90, desenvolvidas inicialmente em universidades por grupos de pesquisa em computação de alto desempenho. Elas foram

criadas para serem plataformas mais baratas que supercomputadores, e possibilitarem a execução de aplicações paralelas em escalas maiores que as possíveis em um único supercomputador, através da aglomeração de recursos (Foster et al., 2001).

O termo computação em grade surgiu como uma analogia com a rede elétrica (*The Electric Grid*), pois esta disponibiliza eletricidade sob demanda, e o cliente não precisa saber da sua origem, e da complexidade na sua transmissão e distribuição. Esse conceito revela a intenção de simplificar a utilização de grades computacionais (Foster and Kesselman, 2003).

Uma grade computacional é um tipo específico de sistema distribuído, composto por conjuntos de unidades de armazenamento, de processamento, e diversos outros tipos de recursos, conectados por uma rede. Este sistema é tipicamente heterogêneo, compartilhando serviços mantidos para suprir a necessidade de uma ou mais comunidades. Esse compartilhamento é gerenciado por uma camada de *software*, o *middleware*, que define políticas de uso e controle dos recursos, que podem pertencer a mais de uma organização (Foster and Kesselman, 2003) (Tanenbaum and Steen, 2007). A figura 2.1 representa uma grade computacional, construída sobre a Internet conectando recursos heterogêneos.

A computação em grade permite criar organizações virtuais, que compartilham recursos distribuídos, para alcançar um objetivo comum, sem controle centralizado. Tais organizações virtuais podem ocupar o mesmo espaço físico ou serem grupos espalhados pelo mundo; serem estáticas ou dinâmicas; grandes ou pequenas; e ainda serem formadas para um evento e dissolvidas após o fim do mesmo (Abbas, 2003).

Apesar de ser possível definir diversas características atribuídas a uma grade computacional, estabelecer uma estrutura formal para esta tecnologia é uma tarefa tão complexa quanto definir um sistema distribuído. Neste texto será usada a definição de grade dada por Foster and Kesselman (2003), em que a grade pode ser concebida de acordo com três requisitos:

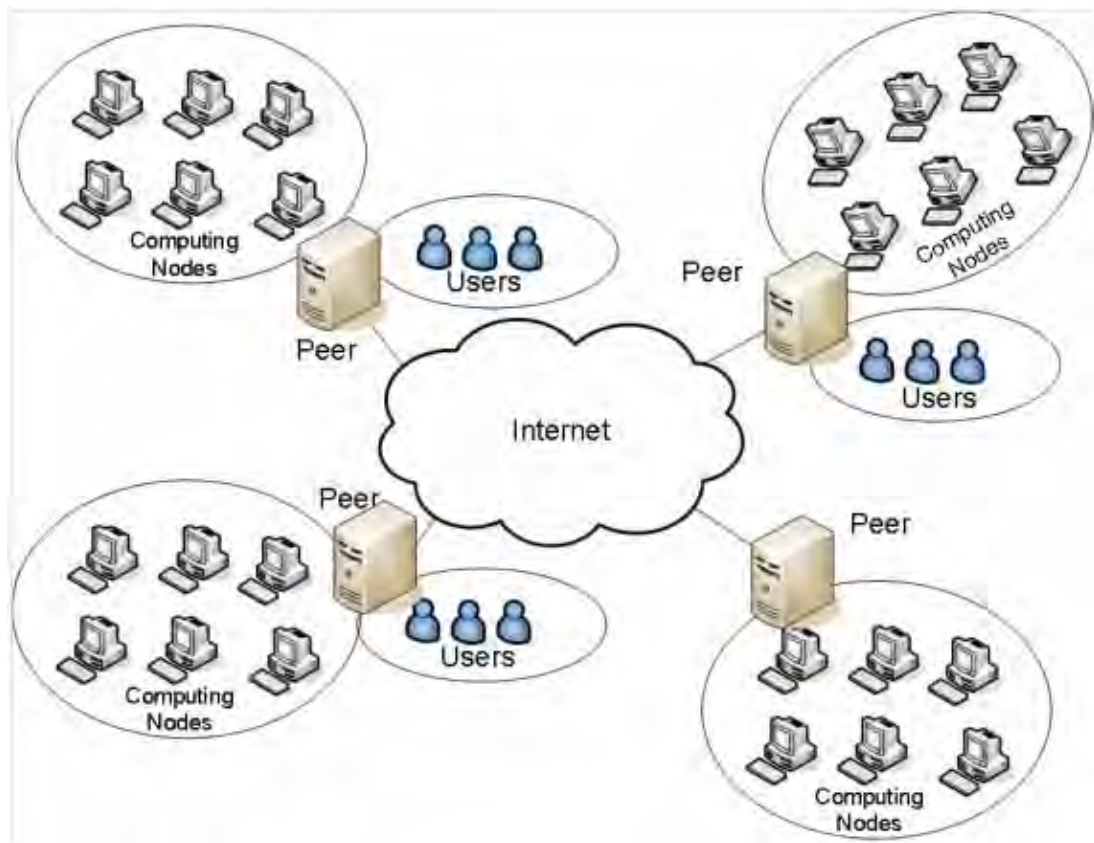


Figura 2.1: Grade computacional (Anglano et al., 2008)

- **Coordenação de recursos distribuídos:** Em uma grade deve haver integração coordenada dos recursos e usuários, em áreas de controle diferentes, abordando questões de política de uso e segurança, entre outras. Este controle não é centralizado.
- **Protocolos e interfaces padronizados ou abertos:** Uma grade é construída a partir de diversos protocolos e interfaces, atendendo problemas como autenticação e autorização. É importante que os protocolos e interfaces sejam padronizados ou abertos, caso contrário, seria um sistema de aplicação específica.
- **Qualidade de serviços não-triviais:** Os recursos da grade devem ser utilizados de forma coordenada, oferecendo vários níveis de qualidades de serviços QoS,

relativos a tempo de resposta, disponibilidade e segurança. Essa utilização coordenada busca tornar o sistema combinado maior que a simples soma de suas partes.

2.1.1 Tipos de Grades Computacionais

Abbas (2003) separa as grades computacionais em sete categorias, através de dois critérios distintos de classificação. Partindo de sua estrutura de organização as grades podem ser:

- **Departamentais:** São feitas para solucionar os problemas de um grupo particular de pessoas dentro de uma empresa. Os recursos não são divididos com outros grupos dentro da empresa. Este tipo de grade otimiza recursos em uma empresa sem envolver outros parceiros internos.
- **Empresariais:** Consistem em grades com recursos disponíveis em vários setores da empresa, provendo serviço para todos os usuários desta instituição. Múltiplos projetos e departamentos compartilham seus recursos de modo cooperativo.
- **Extra-empresariais:** São estabelecidas entre companhias, parceiros de negócios e consumidores. Os recursos normalmente são disponibilizados por meio de uma rede virtual privada (VPN).
- **Globais:** São grades formadas sobre a internet pública, elas podem ser estabelecidas por organizações distintas para facilitar os negócios ou compras entre as partes, ou prover serviços. Sua utilização pode ser feita por meio de compra, troca ou venda do excesso de capacidade de recursos.

O segundo critério de classificação está relacionado ao sistema e aos principais serviços fornecidos, podendo ser grades dos seguintes tipos:

- **Grades de Computação ou alto desempenho:** Este modelo de grade criou-se com o simples objetivo de prover acesso a recursos computacionais. Elas são classificadas de acordo com seu *hardware* computacional: Desktop Grids compartilham recursos de computadores pessoais; Server Grids, formado por servidores; e High-Performance/Cluster Grids são formados por supercomputadores e clusters.
- **Grades de Dados:** São Grades que requerem armazenamento, acesso ou processamento de grandes volumes de dados. Elas são otimizadas para operações orientadas a dados, e consomem uma grande capacidade de armazenamento.
- **Grades de Utilidades:** As grades de utilidades são recursos computacionais mantidos e gerenciados por provedores de serviços. Este tipo de grade deve apresentar grande disponibilidade. Consumidores com necessidades computacionais podem comprar “ciclos” de uma grade de utilidades. Estes recursos podem também ser comprados por uma empresa para aumentar os seus próprios.

2.1.2 Escalonadores

Como indicado no capítulo anterior, o escalonamento é uma área de grande interesse em grades computacionais. Se a escolha da política de escalonamento não for adequada pode levar a um baixo desempenho na aplicação, e consequentemente influenciar o sistema inteiro. Essa escolha é complexa pois para grades existem os complicadores da dispersão dos recursos e domínios organizacionais, o que demanda estratégias diferenciadas.

O escalonador utiliza regras definidas pela política de escalonamento para fornecer os recursos disponíveis aos consumidores que os desejam. As regras da política de escalonamento são estabelecidas de acordo com objetivos que se deseja atingir. Como objetivos típicos pode-se citar maximizar a vazão (*throughput*), minimizar o tempo de resposta, ou maximizar a utilização dos recursos. Tais objetivos podem ser conflitantes,

como por exemplo, para maximizar a vazão o ideal é alocar as menores tarefas primeiro, enquanto para maximizar a utilização dos recursos o ideal é alocar as tarefas maiores primeiro, não sendo possível utilizar ambos objetivos ao mesmo tempo (Paula, 2009).

Numa grade computacional o escalonamento ganha outros parâmetros. Por exemplo, é possível escalonar tarefas a um único recurso ou a um conjunto deles, ou ainda dentro de uma organização ou em várias delas. Além disso, os recursos podem ser alocados localmente ou de forma distribuída. Todas essas características, assim como outras características da grade, devem ser consideradas de modo à melhor distribuir a carga de trabalho e o desempenho do sistema.

Do ponto de vista taxonômico os escalonadores para grades podem ser classificados basicamente em três categorias (Paula, 2009) (Falavinha et al., 2009):

- **Escalonador Centralizado:** Um único escalonador controla todos os recursos do sistema e é responsável por alocar todas as tarefas, independente do usuário que a submeteu.
- **Escalonamento Hierárquico:** Neste modelo o escalonamento é dividido em dois ou mais níveis de escalonadores. Este tipo de estrutura melhora o tratamento de problemas de escalabilidade, permitindo acréscimo de recursos sem interferir com a gerência de outros, e na tolerância a falhas, com cada domínio podendo ter sua solução. Apesar dessas vantagens, ele não permite um escalonamento totalmente autônomo e com várias políticas distintas.
- **Escalonador Distribuído ou Descentralizado:** Neste tipo de escalonador, o escalonamento é realizado de forma distribuída, portanto não há visão global do sistema. Este escalonador possui maior tolerância a falha, e permite que cada administrador mantenha controle sobre seus recursos. O escalonamento distribuído é dividido em dois tipos de escalonadores:

- **Escalonador de aplicações:** Procura melhorar o desempenho de aplicações do lado de quem as submete, escalonando as tarefas para *sites* de diferentes domínios administrativos. Ela não controla os recursos, mas obtém acesso submetendo solicitações para outros escalonadores
- **Escalonador de recursos:** Trabalha para maximizar a utilização dos recursos do seu *site*.

2.2 Simuladores de grades computacionais

A discussão na seção anterior aponta para a necessidade de ferramentas para avaliação de desempenho. Os simuladores de grades computacionais permitem analisar determinados componentes de um sistema distribuído heterogêneo, abstraindo os componentes de menor interesse na análise realizada. Isto é possível visto que a simulação não utiliza o sistema real mas sim um modelo do mesmo. Para grades computacionais os simuladores são de grande utilidade no estudo de algoritmos de escalonamento de tarefas, permitindo avaliar o desempenho de diferentes algoritmos em diferentes cenários, de forma muito mais simples do que seria possível em um ambiente real. Por este motivo diversas ferramentas de simulação foram concebidas visando a investigação de estratégias de escalonamento, tais como SimGrid (Casanova et al., 2008), GridSim (Buyya and Murshed, 2002), GangSim (GangSim, 2012), OptorSim (OptorSim, 2012) e Bricks (Takefusa and Matsuoka, 2000). A seguir são descritas as principais ferramentas de simulação de grades computacionais.

2.2.1 SimGrid

Idealizado em 1999, por Henri Casanova, sua motivação veio da necessidade de usar simulação, em vez de experimentos reais, no estudo de algoritmos de escalonamento centralizados em plataformas computacionais distribuídas e heterogêneas (Casanova,

2001). Atualmente esta ferramenta fornece funcionalidades para pesquisas na área de computação paralela e distribuída em sistemas de larga escala, como grades computacionais, sistemas P2P e computação em nuvem. O SimGrid, está na versão 3.7 e é uma ferramenta open-source orientada a eventos, desenvolvida na linguagem C, disponível para os ambientes Windows, Linux e MacOS (SimGrid, 2012) (Casanova et al., 2008).

Ele é dividido em vários componentes, fornecendo quatro interfaces para os usuários, escolhidas de acordo com a aplicação que se deseja avaliar. As interfaces são:

- **SimDag:** Módulo projetado para investigação da política de escalonamento. Fornece funcionalidades para simulação de aplicações paralelas por meio do modelo DAG (*Direct Acyclic Graphs*). Com ele pode-se descrever os programas paralelos, por meio das suas tarefas e as dependências delas.
- **MSG:** A interface que permite o estudo de processos concorrentes, para estudar algoritmos de escalonamento. Revelou-se perfeitamente utilizável em outros contextos, se tornando a mais popular API do SimGrid. A utilização deste módulo é feita pela codificação na linguagem C ou nas versão mais recentes em Java (jMSG) Lua (lMSG) e Ruby.
- **GRAS** (*Grid Reality and Simulation*): é um ambiente feito para facilitar o desenvolvimento de aplicações (Quinson, 2006). Através do GRAS é possível desenvolver um único código que será executado tanto no ambiente de simulação quanto em uma plataforma real. Este processo pode ser observado na figura 2.2. Este ambiente faz uso da API fornecida pelo MSG (na simulação) e de sockets (na aplicação real).
- **SMPI:** O ambiente SMPI (Clauss et al., 2011) é um *framework* que permite simular aplicações MPI, permitindo o estudo de aplicações MPI sem ter de modificá-las, atuando como um GRAS para MPI.



Figura 2.2: GRAS (Quinson, 2006)

Criação de modelo

A criação do modelo a ser simulado no SimGrid é dividida em dois esforços: a especificação da plataforma computacional, e a implementação dos processos que serão executados em cada recurso. A especificação da plataforma computacional é feita por meio de arquivos formatados em XML (*Extensible Markup Language*), permitindo alterar a plataforma, e reutilizar os modelos computacionais, de maneira simples e facilitando a simulação, tornando o código da aplicação independente da grade simulada. Desta forma são envolvidos dois arquivos XML:

- Arquivo de plataforma: neste arquivo é indicado os recursos, o poder computacional das máquinas, a banda e latência das redes. Mais de um link pode ser usado para conectar dois recursos, e um link pode ser usado para enviar e receber tarefas.
- Arquivo de aplicação: neste arquivo é especificado qual processo é executado em cada máquina e seus argumentos, cada recurso pode executar n processos, sendo cada um dos processos associados a uma porta diferente. Todas as funções especificadas no arquivo XML devem ser implementadas no código da aplicação.

A codificação dos processos, referenciados no arquivo XML de aplicação, pode ser realizada com o módulo MSG, e suas variações, portanto para implementar estas funções deve ser desenvolvido um programa na linguagem C, Java ou Ruby.

O SimGrid também permite inserir dinamismo a grade simulada, indicando no XML de plataforma por meio de *traces*. Os *traces* são formados por arquivos que indicam variações nas cargas de trabalho dos *hosts*, na disponibilidade dos *links*, e a ocorrência de falhas permanentes e temporárias dos recursos. Este recurso permite que o modelo de simulação seja mais fiel a um sistema real, inclusive permitindo a utilização de cargas de trabalhos obtidas em sistemas reais.

Aplicações

O SimGrid é um dos maiores projetos na área de simulação de grades computacionais, por isto ele está em constante melhoria, recebendo novas funcionalidades, como o módulo SMPI (Clauss et al., 2011), a ferramenta para visualização de escalonamento em sistema distribuídos Jecure (Hunold et al., 2010), o componente SimGrid MC que realiza a checagem de modelos desenvolvidos para esta plataforma (Merz et al., 2011), entre outros trabalhos do grupo de pesquisa responsável pela ferramenta (inclusive utilizando resultados obtidos com a utilização da ferramenta).

Outros trabalhos que utilizam o SimGrid se concentram na área de escalonamento em sistemas distribuídos. Por exemplo, Falavinha Jr. et al. (2007) realiza avaliação de diversos algoritmos de escalonamento com este simulador, enquanto Gao and Zhou (2008) propõe e avaliam uma política de escalonamento baseado em algoritmos genéticos, e Benoit et al. (2010) estudam o escalonamento de tarefas *Bag-of-Task* (BoT) em ambiente heterogêneo.

2.2.2 GridSim

O GridSim (Buyya and Murshed, 2002), atualmente na versão 5.0, é um simulador que permite modelar: diferentes classes de recursos heterogêneos; usuários; aplicações; e escalonadores. Ele pode ser usado para simular escalonadores de tarefas para sistemas distribuídos como clusters (único domínio) e grades (múltiplos domínios). Ele é baseado

no SimJava, um simulador de eventos desenvolvido em Java (GridSim, 2012).

Entre suas características, o GridSim permite reserva avançada dos recursos, e compartilhamento de recursos no tempo (como sistemas multitarefas), ou no espaço (como sistemas em lote). As tarefas podem ser heterogêneas, e cada domínio pode ter uma política de alocação de recursos diferente, permitindo que o usuário desenvolva sua própria política. Cada componente modelado no GridSim é executado por uma thread distinta, eles são conectados utilizando portas, podendo se comunicar por meio de envio e recebimento de eventos, o que possibilita sua execução em arquiteturas paralelas.

Para modelagem da grade computacional desenvolveu-se uma interface gráfica, que permitia salvar os modelos em um arquivo XML para futuras alterações, (Sulistio et al., 2003), contudo esta ferramenta foi descontinuada não sendo mais suportada pelo simulador.

Diversas modificações foram introduzidas desde sua proposta inicial, visando tornar a simulação mais real, como a adição e detecção de falhas (Caminero et al., 2007) ou a simulação de grades de dados (Sulistio et al., 2008), o que permite avaliar políticas de replicação de arquivos e políticas de escalonamento com dependência de grandes arquivos.

Criação de modelos

No GridSim a modelagem da grade computacional, assim como das cargas de trabalho, é feita por meio de implementação, em Java, de um programa que utilize as classes e métodos fornecidos pelo simulador. A interação entre as entidades presentes no modelo é feita por métodos específicos, que abstraem os conceitos da simulação orientada a eventos.

A grade computacional é modelada construindo vários elementos de processamento (PEs) e combinando eles para formar as máquinas da grade. A máquina pode ser monoprocessada (com um único PE) ou multiprocessada (com vários PEs), e combiná-las

para formar clusters. Dessa forma podem-se modelar diversos tipos de recursos, indo desde um computador de um único processador até multiprocessadores de memória compartilhada (SMP), e cluster de computadores, com memória distribuída. Cada recurso possui um escalonador interno, definido pela classe `AllocPolicy`, permitindo ao usuário utilizar uma das duas políticas já implementada (de espaço ou tempo compartilhado), ou desenvolver sua própria política para cada recurso.

A criação e manipulação das tarefas simuladas são feitas pelo pacote `Gridlet`, ele mantém informações sobre operações de E/S, tamanho do arquivo de entrada e saída, tamanho computacional em milhões de instruções, e o usuário que submeteu a tarefa. Os atributos relacionados as tarefas podem ser criados aleatoriamente pela classe `GridSimRandom`, ou utilizando *traces* com a classe `Workload`.

Aplicações

Desde o projeto inicial diversas melhorias foram adicionadas ao `GridSim`, por exemplo Sulistio et al. (2007) acrescenta serviços adicionais a rede (tráfego de fundo, roteamento, coleta de informações, etc) e permitindo maior realismo e a análise da influência da rede na simulação. Além destes são encontrados diversos trabalhos que utilizam o `GridSim` com foco no estudo de políticas de escalonamento, como Ashraf and Erlebach (2011) que propõe uma política de escalonamento híbrida (com diversos algoritmos que são selecionados de acordo com a necessidade) e utiliza o `GridSim` para simular um ambiente com reserva avançada de recurso. Kumar et al. (2011) utilizam o simulador para analisar um algoritmo genético voltado para tarefas com uso intensivo de computação e armazenamento. Outras aplicações envolvem o estudo da replicação de dados (Nukarapu et al., 2011) e da provisão de qualidade de serviço (QoS) Caminero et al. (2006), por exemplo.

Também há trabalhos que utilizam esta plataforma visando ampliá-la para criar novas ferramentas de simulação, como Albodour et al. (2010) que desenvolveu uma

extensão que fornece diversas Qualidades de Serviços. Alea (Klusáček et al., 2008) um ambiente desenvolvido para simular diversos problemas de escalonamento relacionado a computação em Grade. O *Grid Scheduling Simulator* (GSSIM) (Kurowski et al., 2007) também tem como base o GridSim, buscando solucionar algumas dificuldades relacionadas a geração de cargas de trabalhos e níveis de escalonamento.

2.2.3 OptorSim

O OptorSim (Bell et al., 2002) (Bell et al., 2003) é um simulador de grades computacionais projetado para testar diversos algoritmos de replicação dinâmica usados na otimização de dados contidos na grade. A replicação envolve a criação de cópias dos arquivos em recursos geograficamente distribuídos. Desenvolvido em Java, visando imitar estruturas de arquivos de grades reais. Durante a simulação a execução de uma tarefa, ou o acesso aos arquivos, pode desencadear a criação de réplicas dos arquivos envolvidos. Como pode ser observado este simulado é centrado nas grades de dados, nas quais há grandes transferências de dados, se tornando um fator de limitação no desempenho do sistema. Atualmente está na versão 2.1 (OptorSim, 2012).

Uma das principais considerações no projeto do OptorSim foi modelar as interações dos componentes individuais da grade de dados da forma mais realista possível. Por isto, a simulação foi baseada na arquitetura do EU DataGrid project (Cancio et al., 2001). O simulador assume que a grade é formada por vários *sites*, cada um deles fornece recursos computacionais e de armazenamento para as tarefas submetidas. Seu principal componente é o algoritmo de otimização de réplicas. O OptorSim traz alguns exemplos de algoritmos de replicação simples e um mais complexo, de abordagem econômica (Ernemann et al., 2002).

Criação de modelos

A modelagem da grade computacional, assim como do conjunto de tarefas, é feito por meio da criação de arquivos de configurações. Já o algoritmo para buscar e criação de replicas é feito com a implementação de uma classe que utilize a interface *Optimisable*, sendo que alguns algoritmos de replicação já vem implementados.

No arquivo de especificação da grade é descrita a topologia de cada *site* e seu comportamento, indicando o número de elementos computacionais, de armazenamento (com o tamanho em megabytes) e a banda de conexão entre os *sites*.

Em outro arquivo é feita a especificação das tarefas que serão atendidas durante a simulação. Neste arquivo também é especificada a localização e tamanho dos arquivos utilizados pelas tarefas.

Um terceiro arquivo é utilizado para passar vários parâmetros para a simulação e especificar características da aplicação, como tipo de usuário, estratégia de escalonamento do *Resource Broker*, tamanho máximo da fila na CE, entre outras características.

Aplicações

As pesquisas que normalmente utilizam o OptorSim estão ligadas com a análise de técnicas para replicação de dados em grades computacionais, abrangendo transferências de enormes quantidades de dados, no qual é justificado o interesse na otimização destas transferência para reduzir o tempo para atender as tarefas.

Em Shorfuzzaman et al. (2010) é apresentado um algoritmo para otimizar a criação e localização de réplicas em grades de dados hierárquica através da identificação da popularidade dos arquivos. Abdurrah and Xie (2010) propões uma estratégia de replicação baseada na observação que conjuntos de tarefas tendem a utilizar grupos comuns de arquivos, dessa forma mantendo os arquivos próximos das tarefas que irão utilizá-los. Zhong et al. (2010) apresenta uma estratégia de gerenciamento de réplica dinâmica que aumenta automaticamente o número de réplicas de acordo com a frequência de acesso

do arquivo.

2.2.4 GangSim

O GangSim (Dumitrescu and Foster, 2005) é uma ferramenta desenvolvida para o estudo de políticas de escalonamento em grades computacionais, ela permite a análise da interação entre as políticas de escalonamento locais (de um *site*) com as políticas distribuídas (pertencentes a Organizações virtuais). A ferramenta pode ser utilizada para simular grandes grupos, ou *gangs*, de *sites* e usuários (GangSim, 2012).

O GangSim é derivado, em parte, do sistema de monitoramento Ganglia (Ganglia, 2012) utilizado para o monitoramento de sistemas de computação de alto desempenho, como clusters e grades computacionais. Esta abordagem de implementação permite a combinação de componentes de grade simulada com componentes reais.

A inovação deste simulador é a possibilidade de modelar usuários e organizações virtuais (VO), e sua capacidade para modelar as políticas de uso, tanto no nível de *site* como no de VO. Dessa forma o sistema permite avaliar o comportamento que uma grade computacional, existente, teria ao aumentar o número de *sites* e usuários (Dumitrescu and Foster, 2005).

O escalonamento das tarefas é feito de forma descentralizada, com cada VO adota uma política de escalonamento, e contribui obtenção das informações de monitoramento.

Criação de modelos

A modelagem da grade computacional é realizada por meio de um arquivo de configuração, mesmo procedimento do VO-Ganglia. No arquivo é especificado o número de VOs e *sites*, o poder computacional das máquinas presentes nos *sites*, e as capacidades de armazenamento dos elementos presentes neles. A modelagem das cargas de trabalhos é feita por ferramentas específicas do GangSim, enquanto que a especificação das políticas de escalonamento dos *sites* e VOs é feita por arquivo de configuração ou por

uma interface Web presente no GangSim.

Aplicações

Não são encontradas muitas publicações utilizando este simulador, excetuando-se trabalhos com resultados obtidos pelo grupo de desenvolvimento deste projeto, como Dumitrescu et al. (2005), Dumitrescu et al. (2006) e Dumitrescu et al. (2007), que realizam estudos sobre escalonamento em sistemas com várias organizações virtuais e políticas de uso para os recursos.

2.3 iSPD

O iSPD (*iconic Simulator of Parallel and Distributed systems*) (Manacero et al., 2012) é o simulador alvo desta pesquisa. Ele é uma plataforma de simulação de grades computacionais desenvolvida, mantida e distribuída pelo Grupo de Sistemas Paralelos e Distribuídos da UNESP (GSPD, 2012). Desenvolvido buscando ser uma ferramenta fácil de usar, permitindo aos usuários simularem sistemas complexos sem a necessidade de conhecerem programação, como é o perfil de grande parte dos usuários de grades computacionais.

Desenvolvido na plataforma Java, sendo dividido em vários módulos que serão abordados na próxima seção. O simulador trabalha com tarefas do tipo *Bag-of-Task* (BoT) e o modelo de programa paralelo utilizado na grade é o mestre-escravo. Uma característica diferencial no iSPD é a possibilidade de importar modelos de outros simuladores. Tal capacidade não foi identificada em nenhum dos simuladores estudados (Guerra et al., 2010).

2.3.1 Criação de modelos

Um dos principais objetivos do projeto do iSPD foi simplificar a criação e modelagem dos componentes presentes na simulação. A criação de um modelo simulável no iSPD pode ser dividida nas etapas de especificação da grade e das cargas de trabalhos. Um terceiro passo no desenvolvimento de um modelo é a criação de uma aplicação para executar sobre a grade, o que no momento é limitado ao modelo mestre-escravo para tarefas BoT, cada mestre podendo realizar o escalonamento de forma distinta, o que é o foco deste trabalho. Após um modelo ser construído, ele pode ser salvo, seguindo um arquivo XML (*Extensible Markup Language*). Na figura 2.3 é apresentado um exemplo de uma grade no modelo icônico e sua representação em um arquivo XML, omitindo-se o conteúdo das tags.

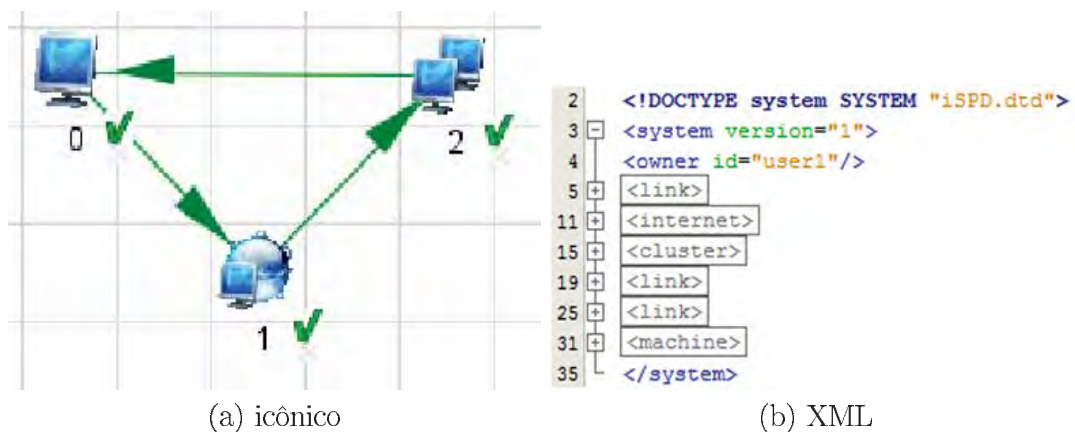


Figura 2.3: (a) icônico. (b) XML.

Especificação da plataforma computacional

O sistema provê ao usuário uma interface gráfica que fornece funcionalidades básicas para a especificação da plataforma computacional. A interface segue um modelo icônico, no qual o usuário modela uma arquitetura de grade computacional adicionando ícones que representam os recursos da grade e interligando eles por meio de conexões de rede.

Esta interface é intuitiva e visa simplificar a experiência do usuário com o simulador.

Ao adicionar um ícone no campo de desenho ele vem desconfigurado, o que é indicado por um **x** nos ícones para máquinas, cluster e conexão com internet e pela cor vermelha nos ícones para conexão de rede ponto-a-ponto, conforme pode ser observado na figura 2.4. Após a configuração de um elemento o **x** é substituído por uma marca de verificado e a conexão de rede fica verde. Ícones como máquinas ficam com um quadrado em volta quando selecionados, enquanto a conexão de rede fica com tom preto.



Figura 2.4: Visão de um modulo ainda não totalmente configurado

Os atributos de um ícone podem ser configurados diretamente na área de *Settings* da janela principal do iSPD, ou pressionando-se duas vezes o botão esquerdo do mouse sobre o ícone. Com o uso do mouse o simulador irá abrir uma janela solicitando informações específicas, conforme o ícone selecionado. Entre as características dos elementos da grade pode ser configurado um fator de carga, indicando computação ou comunicação de fundo que é mantida durante toda a simulação. Após realizar a configuração de todos os ícones presentes no modelo, o ambiente computacional estará pronto para realizar a simulação.

Especificação das cargas de trabalhos

Para concluir a construção de um modelo é necessário configurar as cargas de trabalho, o que é feita pela interface apresentada na figura 2.5. Na janela de configuração de tarefas é permitido configurar tarefas de forma randômica com os parâmetros de distribuição solicitados, ou adicionando cargas para cada mestre, que criará as tarefas, possivelmente como aplicações pertencentes a usuários distintos.

Com todo o modelo configurado, incluindo as tarefas a serem executadas, a simula-

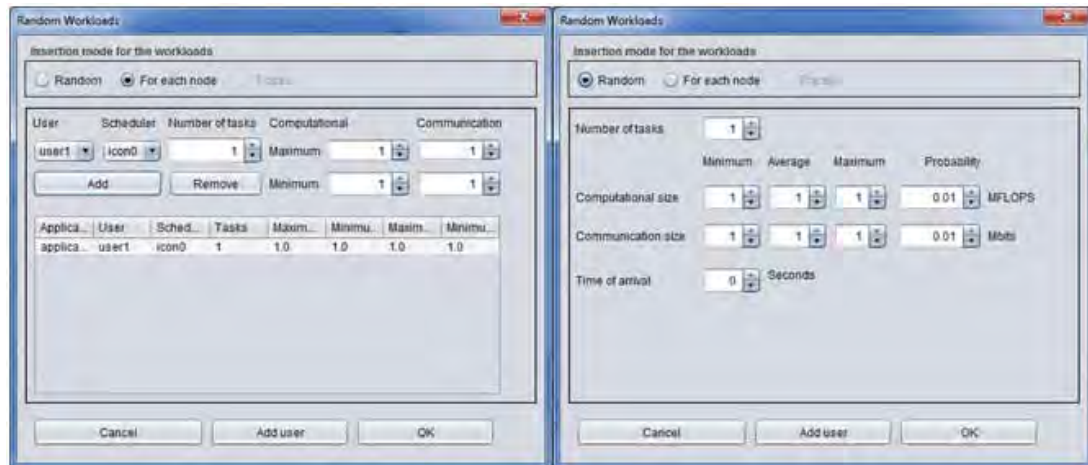


Figura 2.5: Janela de configuração das cargas de trabalho

ção pode ser iniciada. Isso implica na conversão do modelo icônico para o simulável, e sua execução na forma de sistema de filas pelo motor de simulação.

2.4 Comparação entre os simuladores de grades computacionais

Para concluir esse trabalho de revisão é interessante comparar mais objetivamente os simuladores aqui apresentados. Assim, a tabela 2.1 apresenta diversas características de cada um das plataformas de simulação estudadas. Foram selecionadas e analisadas diversas características importantes dos simuladores, como a linguagem de desenvolvimento, que implica em portabilidade, os tipos de sistemas suportados, indicando a disponibilidade para uso assim como o tipo de licença, que possibilita ainda modifica o simulador de acordo com a necessidade do usuário. A documentação facilita o uso e modificação de uma aplicação, enquanto o mecanismo de simulação está ligado ao desempenho e escalabilidade. Finalizando com a análise da computação e comunicação de fundo utilizada para tornar os modelos mais próximos de sistemas reais.

Observando as linguagens de desenvolvimento pode-se observar a predominância

Tabela 2.1: Comparação entre simuladores de grades computacionais.

	SimGrid	GridSim	GangSim	OptorSim	Beosim	iSPD
Lingua-gem	C	Java	Perl	Java	C	Java
Plataformas suportadas	Linux, Mac Os, Windows	S.O. com suporte a JVM	Linux	S.O. com suporte a JVM	Linux	S.O. com suporte a JVM
Interface com usuário	Modo texto	Modo texto	Modo texto	Modo texto GUI	Modo texto GUI	GUI
Licença	<i>Software</i> livre	<i>Software</i> livre	<i>Software</i> livre	<i>Software</i> livre	<i>Software</i> proprietário	<i>Software</i> livre
Documentação	Muito boa	Muito boa	Fraca	Boa	—	Regular
Mecanismo de simulação	Sequencial	Multi-threaded	Sequencial	Multi-threaded	Sequencial	Sequencial
Computação de fundo	SIM	NÃO	NÃO	NÃO	—	SIM
Comunicação de fundo	SIM	SIM	NÃO	SIM	—	SIM

da plataforma Java. A plataforma de desenvolvimento exerce influência em diversas áreas como no desempenho, portabilidade, facilidade de aprendizagem e expansão da ferramenta, contudo não há superioridade na escolha de uma, cada uma delas possui suas vantagens e desvantagens. A linguagem de desenvolvimento possui ligação direta com a facilidade de portar o simulador para outras plataformas, com uma linguagem interpretada como Java é necessário que o sistema operacional alvo tenha uma máquina virtual compatível com a implementação realizado, algo simples, enquanto que na linguagem C o código deve ser recompilado para cada sistema, muitas vezes sendo necessário realizar modificações na implementação para realizar tal conversão. A plataforma de desenvolvimento também está intimamente ligada com a interface gráfica e

desempenho, por exemplo, na linguagem C uma interface estaria limitada ao sistema operacional desenvolvido, enquanto que em Java depende da máquina virtual Java, e na plataforma Web de um navegador, enquanto linguagens interpretadas, como Java e Perl, comumente possuem menos desempenho que uma diretamente executada.

A interface de interação com o usuário está associada diretamente à facilidade de uso da ferramenta. Nos simuladores estudados a modelagem da grade computacional e das cargas de trabalho é feita por meio de algum tipo de arquivo de configuração. Enquanto os resultados normalmente são mensagens e métricas obtidas durante a simulação, que são apresentadas aos usuários por meio de uma interface gráfica ou de um terminal de texto. Quase todos os simuladores tiveram projetos de interfaces gráficas para apresentação dos resultados, o OptorSim possui uma das melhores interfaces de resultados. No entanto poucas interfaces foram feitas para modelagem da grade, como a interface que o GridSim possuía, e a presente no iSPD. Obviamente pelo foco do iSPD, este projeto possui a interface de modelagem com maiores funcionalidades, contudo a interface de resultados é muito simples, e por este motivo está recebendo melhorias.

A documentação também é um importante fator para facilitar o uso dos simuladores e estimular sua utilização, tanto para ajudar o usuário a compreender e utilizar a ferramenta, quanto para auxiliar o desenvolvedor na ampliação ou modificação das suas funcionalidades. Neste quesito um destaque negativo é o GangSim, que não oferece uma documentação detalhada para sua instalação e utilização. O iSPD está melhorando sua documentação, acrescentando tutoriais de auxílio para o uso na própria ferramenta, e documentação nas classes para auxiliar desenvolvedores.

A licença de uso indica a acessibilidade da ferramenta aos usuários, incluindo a possibilidade de modificá-la e adicionar novas funcionalidades. Todos os projetos estudados são de código aberto, destacando-se o GridSim e o SimGrid, que possuem projetos de simuladores e extensões desenvolvidos sobre estas plataformas. O único simulador de código fechado, e acesso restrito, encontrado foi o Beosim (Jones et al., 2005) (Jones

et al., 2010). Por este fato não obteve-se muitas informações sobre esta ferramenta.

Já o mecanismo de simulação tem relação com a escalabilidade da ferramenta. O destaque nesta área é o GridSim, que utiliza o pacote de simulação SimJava, que possui versão para execução distribuída. O uso do Distributed SimJava em conjunto com uma máquina virtual Java também distribuída (como o Jessica2 (Zhu et al., 2002)), permite a execução paralela da simulação.

Como a computação e comunicação de fundo indicam utilização externa da grade computacional, sua disponibilidade torna o ambiente mais real. Em alguns simuladores, como SimGrid e OptorSim, estes parâmetros são definidos por meio de *traces*, que podem ser obtidos em ambientes reais por meio de algumas ferramentas de monitoração, como NetPerf (Netperf, 2012), suportadas pelos simuladores e alguns padrões específicos, como *Standard Workload Format* (SWF) (Iosup et al., 2006). O iSPD possui a funcionalidade de computação e comunicação de fundo, contudo o valor é fixo, mantido durante toda a simulação, o que implica que em alguns casos não representará exatamente o comportamento que o usuário deseja.

2.5 Considerações finais

Neste capítulo realizou-se uma revisão bibliográfica sobre os temas fundamentais para desenvolvimento desta pesquisa. Apresentaram-se os principais conceitos sobre grades computacionais, e simulação destas plataformas. Foram apresentados os principais simuladores de grades computacionais, com destaque ao simulador alvo desta pesquisa, encerrando com uma comparação entre esses simuladores, que é o ponto de partida para justificar o projeto apresentado no próximo capítulo.

Capítulo 3

iSPD

Como este trabalho de pesquisa visa ampliar as funcionalidades do simulador iSPD é necessário realizar uma apresentação mais detalhada desta ferramenta. Este capítulo apresenta arquitetura original do iSPD, permitindo um melhor entendimento das contribuições realizadas nesse trabalho, descritas no próximo capítulo.

3.1 Arquitetura do iSPD

O simulador de grades computacionais iSPD foi apresentado no capítulo 2. Contudo é necessário realizar uma abordagem mais detalhada de alguns de seus componentes. O iSPD tem como base uma interface icônica para criação ou importação de modelos, ou ainda definição de cargas de trabalho. Na figura 3.1 apresenta-se um diagrama conceitual desta plataforma. Por meio dela o usuário criará um modelo icônico da grade computacional, que posteriormente é convertido em um modelo simulável baseado em filas, que é então simulado para disponibilizar as métricas de desempenho que sejam interessantes para o usuário.

Ao observar a figura 3.1 nota-se que o iSPD é dividido em três módulos ou subsistemas principais: a interface gráfica (icônica), o interpretador de modelos internos e

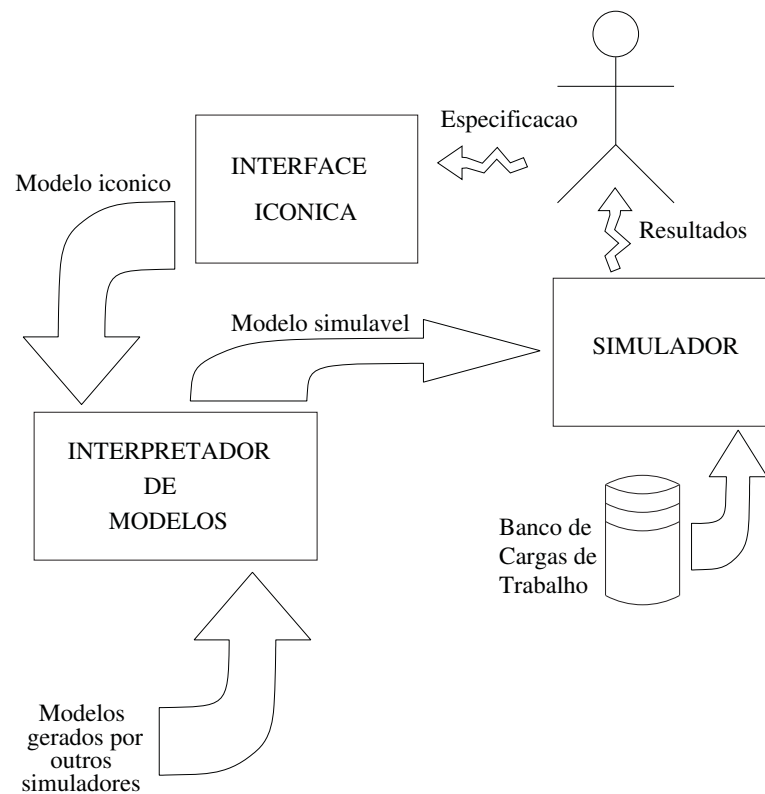


Figura 3.1: Diagrama conceitual da plataforma de simulação (Guerra et al., 2010)

externos e o motor de simulação. Essa modularização, além de auxiliar no desenvolvimento e na implementação, delimitou o papel de cada um desses sub-sistemas com precisão. A seguir são descritos cada um dos módulos presentes no iSPD.

3.1.1 Interface gráfica

Toda interação ocorre por meio da interface gráfica, permitindo ao usuário modelar uma grade e as cargas de trabalhos; carregar modelos criados anteriormente ou para outros simuladores; e apresentar os resultados após uma simulação. Para tal este módulo utiliza das funcionalidades disponibilizadas pelos demais módulos. A figura 3.2 apresenta a interface principal do iSPD, através desta interface é realizada a modelagem apresentada no capítulo 2.

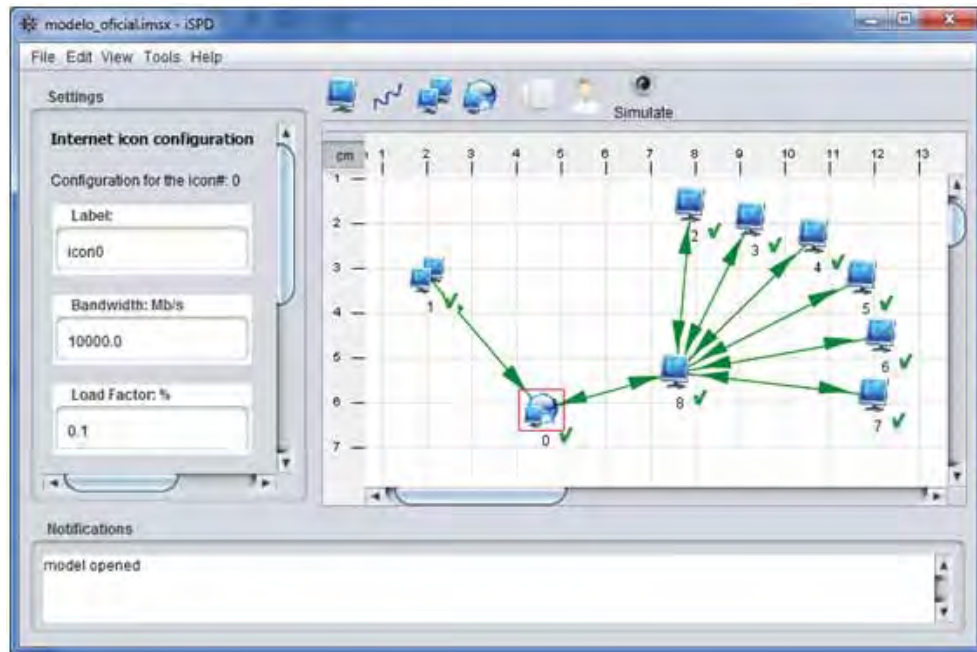


Figura 3.2: Interface principal do iSPD

3.1.2 Interpretador de modelos

É o módulo responsável pelas operações de conversão de modelos, sendo dividido em conversores para modelos internos e externos. Detalhes do desenvolvimento deste módulo são apresentados em (Aoqui et al., 2010).

- **Interpretador de modelos internos:** O iSPD utiliza dois modelos internamente para realizar a simulação: um icônico, montado pelo usuário por meio da interface gráfica, e outro de filas, que é utilizado pelo motor de simulação. Ambos os modelos são representados por gramáticas específicas, portanto possuem interpretadores próprios. A figura 3.3 apresenta a interação entre os interpretadores internos durante o processo de simulação, pode-se observar que com a descrição de uma grade a partir do modelo icônico é realizada a análise e convertido em um modelo simulável que também passa por uma fase de interpretação para ser utilizado pelo motor de simulação.

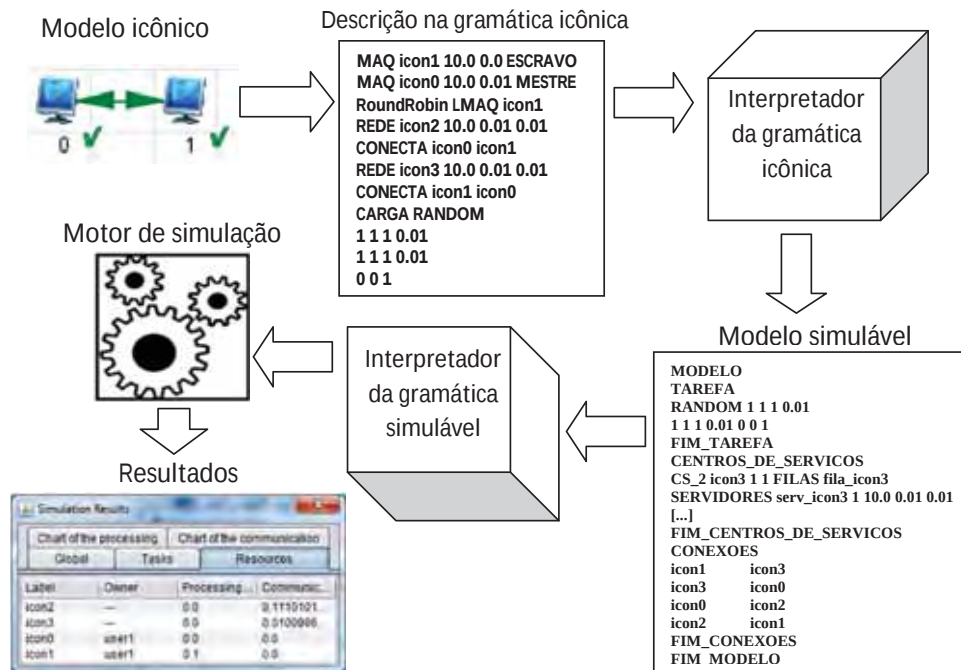


Figura 3.3: Interpretadores internos do iSPD

- Interpretador de modelos externos:** realiza a conversão de modelos desenvolvidos para outros simuladores para o modelo icônico do iSPD (Aoqui et al., 2010). Atualmente é possível converter modelos escritos para o SimGrid, estando em desenvolvimento um novo módulo para interpretação de modelos do GridSim. A figura 3.4 apresenta como é realizada a leitura de um modelo feito para o SimGrid no iSPD. São necessários dois arquivos, a partir dos quais um contendo a descrição da plataforma computacional e outro com os parâmetros da aplicação, o modelo feito para o SimGrid é convertido para a gramática do iSPD para então ser montado o modelo icônico na interface de usuário. Um processo semelhante pode ser realizado na conversão de modelos de outros simuladores.

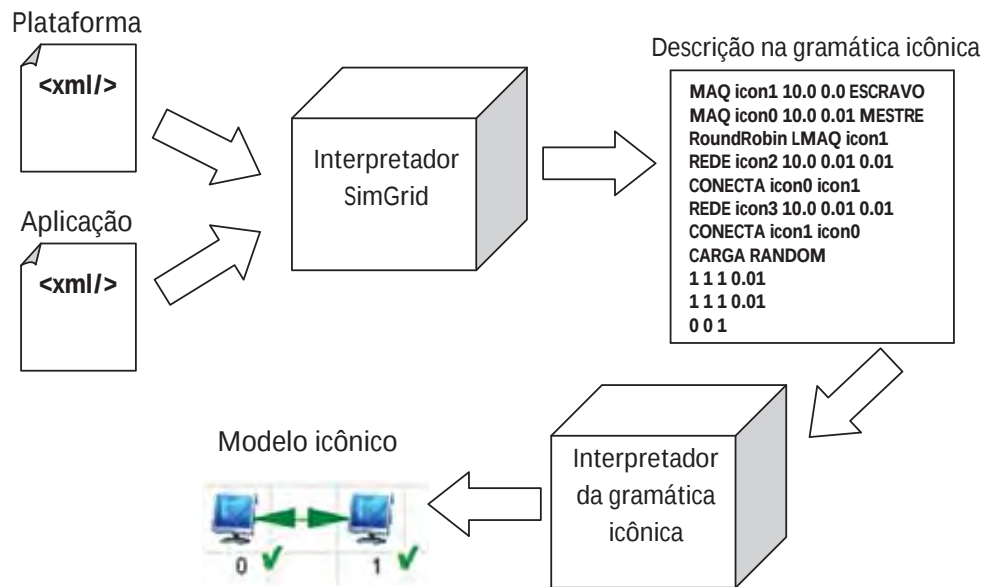


Figura 3.4: Interpretador externo

3.1.3 Motor de simulação

O motor de simulação é o módulo que realiza efetivamente a simulação e utiliza as métricas para calcular e apresentar os resultados, tendo sido desenvolvido e descrito em (Oliveira et al., 2010). Este módulo recebe um modelo escrito na gramática do modelo simulável, e utiliza o interpretador desta gramática para construir o modelo com objetos para realizar a simulação, instanciando o modelo simulável.

O motor realiza a simulação de eventos discretos com o modelo de teoria de filas, utilizando a modelagem orientada a evento. Informações detalhadas de sua implementação podem ser encontradas em (Oliveira et al., 2010).

A figura 3.5 apresenta um diagrama conceitual desse módulo. Durante a execução da simulação há uma lista de eventos que são atendidos pelo motor, de acordo com o tempo marcado pelo relógio de simulação. Os eventos são atendidos no gerenciador da rede de filas, que realiza o escalonamento, roteamento, e ainda detém o controle sobre os centros de serviços e as tarefas.

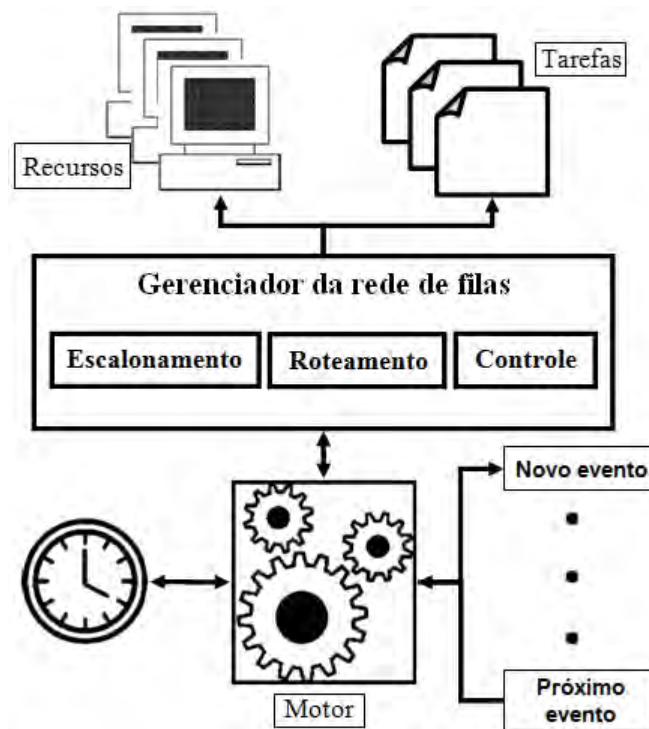


Figura 3.5: Primeiro protótipo do motor de simulação

O motor de simulação possui ainda um componente responsável por recolher e calcular todas as métricas, para posteriormente apresentar os resultados ao usuário, a seguir este componente será apresentado.

Métricas

Para a avaliação do desempenho da grade sendo simulada são oferecidas algumas métricas de desempenho. Essas métricas são agrupadas em três categorias:

- **Relativa à grade:** Apresenta as métricas referentes ao comportamento global da grade que está sendo simulada;
- **Relativa aos recursos:** Para cada um dos recursos definidos no modelo apresenta-se o tempo gasto por ele processando ou comunicando;

- **Relativa às tarefas:** Durante a simulação as tarefas são os clientes dos recursos, que são os centros de serviços, a partir desta relação pode-se obter métricas relativa ao tempo de espera e atendimento.

Escalonamento

O controle do escalonamento durante a simulação faz parte do motor, sendo um dos pontos de interesse desta pesquisa. No protótipo apresentado na figura 3.5 pode-se observar que o escalonamento é realizado pelo gerenciador de redes de filas. Contudo não projetou-se o protótipo para permitir a alteração do escalonador de forma simples. Para modificar a política de escalonamento seria necessário alterar alguns métodos da classe que faz o gerenciamento da rede de filas.

Dessa forma para viabilizar o estudo de escalonadores no iSPD seria necessário fazer algumas alterações, e como o princípio de projeto do iSPD é a simplicidade para modelagem dos sistemas a serem simulados, também a especificação dos algoritmos de escalonamento deve ser feita de forma simples e fácil. Este projeto de pesquisa desenvolveu técnicas para criação e inserção de novos escalonadores para o iSPD, por isto as alterações realizadas no iSPD são abordadas no capítulo 4, que apresenta o desenvolvimento realizado.

3.2 Considerações finais

Este capítulo permitiu observar algumas características da primeira versão do iSPD, incluindo suas vantagens e problemas para o tratamento de escalonadores. Esse estudo permite um melhor entendimento das contribuições realizadas neste trabalho, que serão apresentadas no próximo capítulo.

Capítulo 4

Geração e gerenciamento de escalonadores

Este capítulo é o corpo central desse trabalho. A parte inicial do capítulo trata da especificação do gerador de algoritmos, incluindo componentes do iSPD alterados para permitir a inclusão da metodologia para geração de escalonadores proposta. Após isso são apresentados detalhes da implementação dessas soluções, incluindo os diagramas UML mais relevantes do projeto.

4.1 Módulo de escalonadores

Conforme apresentado no capítulo 1, a avaliação do impacto causado por escalonadores de tarefa é de grande importância em sistemas distribuídos, justificando plenamente a inclusão de um módulo capaz de realizar essa atividade no iSPD. Como o princípio de projeto do iSPD é a simplicidade para modelagem dos sistemas a serem simulados, também a especificação dos algoritmos de escalonamento deve ser feita de forma simples e fácil. Com o objetivo de alterar as políticas de escalonamento utilizadas nos modelos criados no iSPD, e permitir ao usuário adicionar novos algoritmos de escalonamento de

forma simplificada, foram propostas algumas alterações e adições para esta plataforma.

As alterações propostas deram origem a um novo módulo responsável exclusivamente pelo processo de escalonamento. O módulo de escalonadores então seria o responsável pela geração novos escalonadores, por adicioná-los aos disponíveis no iSPD, e controlar o processo de remoção e edição dos escalonadores existentes. Além destas ações, ligadas diretamente à manipulação da entidade que realiza o escalonamento, o módulo de escalonadores deve fornecer uma interface para que o motor de simulação possa usar este novo elemento.

Para a construção do módulo de escalonadores necessitou-se definir como ele seria inserido no iSPD, e a forma que o usuário iria utilizá-lo. A seguir são apresentados os diagramas de caso de uso e atividades construídos com a adição do módulo de escalonadores.

4.1.1 Caso de uso

A figura 4.1 apresenta o diagrama de casos de uso do iSPD (Guerra; Aoqui, 2010) com a adição do módulo do escalonador, conforme pode ser observado os casos de uso destacados (Editar/Remover escalonador; Gerar escalonador; Controlar escalonadores) foram adicionados neste trabalho. Agora, através da interface gráfica do iSPD, além da opção de abrir ou criar um modelo, o usuário também poderá criar ou editar uma política de escalonamento para adicionar ao modelo. A seguir é apresentada uma breve descrição de cada caso de uso:

- **Criar Modelos Simuláveis:** Usuário acessa a interface para criar um novo modelo;
- **Abrir Modelos Simuláveis:** Usuário acessa interface para editar modelo existente;

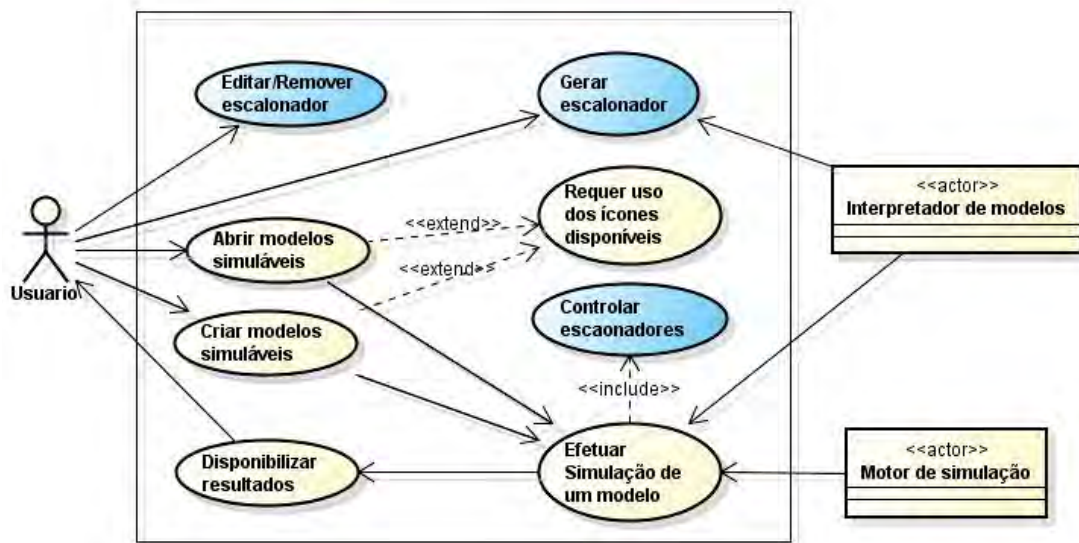


Figura 4.1: Diagrama caso de uso UML

- **Requer Uso dos Ícones disponíveis:** Extensão para auxiliar na criação, edição e parametrização do modelo;
- **Efetuar a simulação de um modelo:** Após ter criado o modelo o usuário inicia a simulação, o ator interpretador de modelos verifica o modelo e inicia a simulação, que é efetuada pelo ator Motor de simulação;
- **Controlar escalonadores:** caso de uso que auxilia no controle dos escalonadores utilizados pelo ator motor de simulação;
- **Disponibilizar resultados:** Depois de efetuar a simulação os resultados são gerados pelo ator Motor de simulação e entregues ao Usuário;
- **Editar/Remover escalonador:** Usuário pode editar o código, ou remover um escalonador já criado;
- **Gerar escalonador:** Usuário especifica a política de escalonamento, e o Agente gera o código para política descrita;

4.1.2 Atividades

A figura 4.2 apresenta o diagrama de atividades apenas do módulo de escalonadores, ele é dividido nas atividades dos atores Usuário e Interpretador de modelos, e as atividades realizadas pelo sistema representado pela Interface Gráfica. Este diagrama apresenta as atividades realizadas para adição e edição de um escalonador, mas não as que ocorrem durante a simulação.

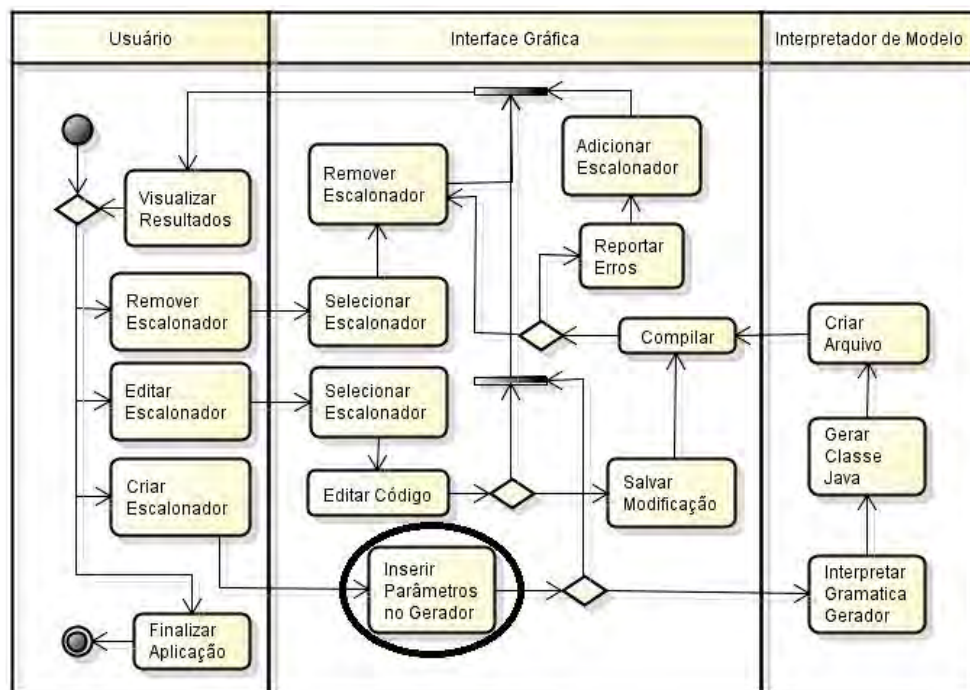


Figura 4.2: Diagrama de atividades UML

Observando a figura 4.2 nota-se que o ator interpretador de modelos, já existente no iSPD, ganhou novas atividades, que resultaram em um novo componente para o módulo homônimo a este ator. Também deve ser observada a atividade “inserir parâmetros no gerador”, que merece maior atenção pois representa a interação entre o simulador e o usuário, sendo abordada com maiores detalhes nas próximas seções.

4.2 Geração automatizada de escalonadores

O principal diferencial do iSPD para outros simuladores de grades computacionais é a simplicidade na especificação do modelo a ser simulado. A construção de escalonadores nos principais simuladores de grade disponíveis não é simples, conforme apresentado no capítulo 2. Estes simuladores usualmente não disponibilizam políticas de escalonamento previamente implementadas, ou quando o fazem possuem apenas algumas implementações básicas. Portanto, os algoritmos de escalonamento devem ser implementados pelo usuário, como no SimGrid, em que um algoritmo de escalonamento pode ser implementado na linguagem C com as funções do módulo MSG. Essa forma de especificação de políticas de escalonamento é mantida nos demais simuladores, cada um com suas peculiaridades e limitações, obrigando o usuário a ter conhecimentos avançados sobre cada ferramenta, e suas interfaces de programação, antes de realizar uma simulação.

Portanto o iSPD necessita de uma metodologia diferenciada para realizar a construção de escalonadores no iSPD. Contudo, como o processo deve ser diferente das abordagens apresentadas pelos outros simuladores, dessa forma este trabalho de pesquisa propõe uma nova abordagem, na qual no usuário iria interagir com um componente que realizaria a construção do escalonador de acordo com a especificação fornecida. Isso permitiria o estudo de escalonadores sem que o usuário necessite conhecer as interfaces de programação do simulador. Definiu-se que para construção de um componente que permita ao usuário inserir novos escalonadores sem dificuldades é preciso:

- **Linguagem de especificação de escalonadores:** Permitir a especificação de escalonadores apenas com a descrição de seu comportamento em uma linguagem de especificação adequada;
- **Converter especificação em componente:** A partir de um comportamento especificado deve ser convertido em um componente utilizável pelo motor de simulação, para realizar o escalonamento durante a simulação;

- **Gerenciamento de escalonadores:** Ter mecanismos para gerenciar os escalonadores criados pelos usuários, de forma que eles possam ser utilizados durante a modelagem da grade e da execução da simulação;

Para atender esses requisitos deve-se implementar um componente que:

1. **Tenha uma interface fácil de usar:** toda a comunicação com o usuário se daria por meio desta interface, na qual seriam obtidas todas as informações necessárias para criar uma nova política de escalonamento;
2. **Tenha um interpretador da linguagem de especificação:** A partir de uma especificação o interpretador poderia gerar um componente para ser utilizado durante a simulação;
3. **Tenha um gerenciador dos escalonadores modelados:** Um elemento iria controlar todos os escalonadores adicionados ao iSPD, de modo que o usuário possa utilizá-los na modelagem da grade e o motor durante a simulação;

Esses módulos são descritos a seguir.

4.2.1 Interface gráfica do gerador de Escalonadores

A interação entre usuário e ferramenta deve ser simples, preferencialmente seguindo uma abordagem na qual o usuário esteja habituado a lidar. Por este motivo, a forma escolhida para modelar a interface gráfica foi através de uma sequência de formulários, semelhante aos instaladores de *softwares*. Para descrição da ordem de atendimento das tarefas, e seleção dos recursos, utiliza-se de formulação matemática na qual os usuários podem selecionar uma série de variáveis e relações disponíveis no iSPD. A figura 4.3 apresenta a modelagem de uma sequência de formulários com os passos a serem seguidos pelo usuário durante a geração de seu escalonador. Na sequência de formulários tem-se que os marcados com os números 2, 3 e 4 definem parâmetros da

política de escalonamento, sendo o passo 5 uma alternativa ao passo 3 e 4. O passo 6 indica limitação referentes ao recurso, e o ultimo passo apresenta ao usuário o código resultante do escalonador modelado.

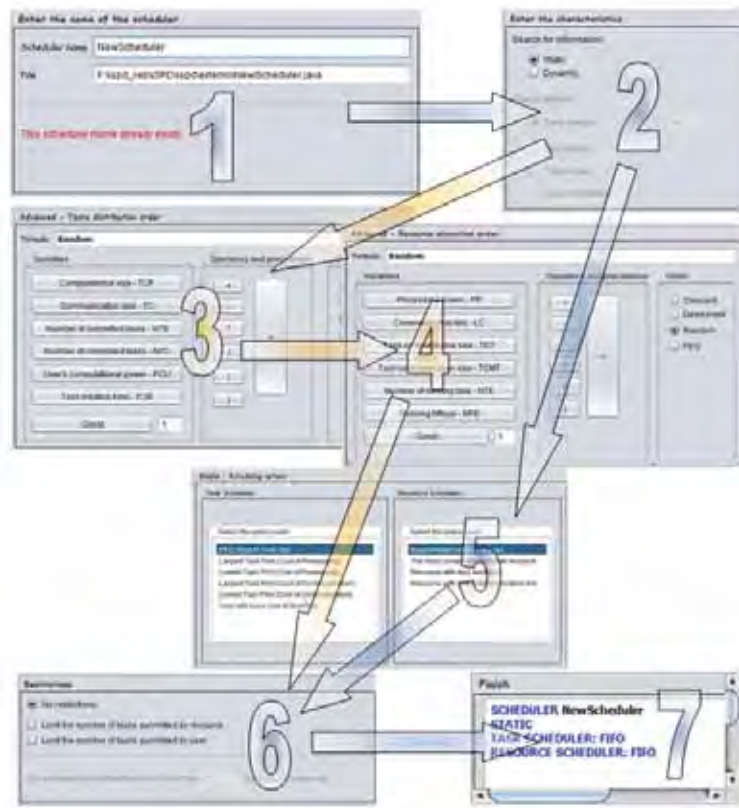


Figura 4.3: Interface do gerador de escalonadores

Todas as informações recolhidas pela interface foram definidas visando construir um código seguindo uma gramática desenvolvida especificamente para definição de políticas de escalonamento. Essa gramática será descrita a seguir.

4.2.2 Gramática de geração de algoritmos de escalonamento

Para representar um modelo de um escalonador devem estar presentes informações sobre como os recursos são alocados e como as tarefas são atribuídas a esses recursos. Isso implica em que a linguagem que os escalonadores serão modelados deve prever símbolos

para esses atributos, o que significa, por exemplo, ter regras para priorizar ou ordenar recursos e tarefas. Tais regras podem ser construídas através de modelos matemáticos para ordenação, os quais calculariam valores de referência para cada recurso ou tarefa.

Para definir as regras da linguagem para modelar a alocação de recursos é preciso considerar informações como:

- **Características do recurso:** Poder computacional, banda de comunicação.
- **Tarefa que será atendida:** Custo de computação, comunicação necessária e tempo de chegada da tarefa.
- **Disponibilidade do recurso:** Número de tarefas em execução e tamanho computacional delas.

Para definir as regras da linguagem para modelar a alocação de tarefas aos recursos é preciso considerar informações como:

- **Características da tarefa:** Custo de computação, comunicação necessária e tempo de chegada da tarefa.
- **Usuário que submeteu a tarefa:** Número de tarefas submetidas e atendidas, quantidade de recursos cedidos.

A gramática do gerador é simples, apesar de apresentar todas as funcionalidades utilizadas pela interface de geração de escalonadores, e consequentemente necessárias para conversão em uma classe Java. A definição desta gramática é apresentada na figura 4.4.

A construção inicial da gramática que descreve escalonadores é o símbolo `<inicial>`, cuja regra de derivação contém todos os componentes necessários para a geração do escalonador. Estes componentes são:

- **<nome>:** Deriva o nome do escalonador, e consequentemente da classe Java que será construída;

```

<var_tar> ::= "[TCP]" | "[NTS]" | "[NTC]" | "[NTA]" | "[TC]" | "[PCU]" | "[TCR]"
<variável_rec> ::= "[PP]" | "[LC]" | "[TCT]" | "[NTE]" | "[TCMT]"
<dígito> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<letra> ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"
<operador> ::= "/" | "*" | "+" | "-"
<inteiro> ::= { <dígito> }+
<real> ::= { <inteiro> }+ "." { <inteiro> }+
<identificador> ::= <letra> { <letra> | <dígito> }*
<constante> ::= <inteiro> | <real>
<variável> ::= <var_tar> | <variável_rec>
<operando> ::= [ "+" | "-" ] ( <variável> | <constante> | "(" <expressão> ")" )
<expressão> ::= <operando> ( <operador> <operando> )*
<formula> ::= ( "CRESCENT" | "DECREASING" ) "(" <expressão> ")" |
"FIFO" | "RANDOM"
<escalona_tar> ::= "TASK" "SCHEDULER:" <formula>
<escalona_rec> ::= "RESOURCE" "SCHEDULER:" <formula>
<limite_tar> ::= "RESTRICT" <inteiro> "TASKPER" ( "RESOURCE" | "USER" )
<tipo_atualiza> ::= "TASK" "ENTRY" | "TASK" "DISPACTH" |
"TASK" "COMPLETED" | "TIME" "INTERVAL" "(" <real> ")"
<caracteristica> ::= [ <limite_tar> ] ( "STATIC" | "DYNAMIC" <tipo_atualiza> )
[ <devolver_tarefas> ]
<nome> ::= "SCHEDULER" <identificador>
<inicial> ::= <nome> <caracteristica> <escalona_tar> <escalona_rec>
<devolver_tarefas> ::= "RETURN" ( "ALL" | <inteiro> ) "TASK" "OF"
( "ALL" | "SELECTED" | "OTHER" ) "RESOURCE" "WHEN"
( "TASK" "DISPACTH" | "TASK" "COMPLETED" )

```

Figura 4.4: Gramática do gerador de escalonadores

- **<caracteristica>**: Representa qual tipo de escalonador será construído, podendo ser estático ou dinâmico (com vários tipos de atualização);
- **<escalona_tar>**: Deriva a fórmula utilizada para selecionar a próxima tarefa a ser enviada para execução;
- **<escalona_rec>**: Semelhante ao **<escalona_tar>**, contudo a fórmula será utilizada para selecionar qual recurso receberá a tarefa;

A derivação da regra sobre o token **<caracteristica>** pode indicar a forma com

que os escalonadores dinâmicos adquirem informações sobre o ambiente. As alternativas para atualização das informações incluem sua execução sempre que chegar uma nova tarefa, após uma tarefa ser enviada do mestre para o escravo, ou após passar um intervalo de tempo fixo.

Com relação a ordem de atendimento das tarefas, e a seleção do recurso, a derivação de $\langle \text{formula} \rangle$ pode indicar uma regra constante, como escolha aleatória ou fixa, ou seguir uma ordem crescente ou decrescente de valores calculados a partir de uma expressão. A seguir são descritas as variáveis disponíveis para formulação da expressão:

Variáveis utilizadas para o escalonador de tarefas:

- **[TCP]** = Custo computacional da tarefa em megaflops;
- **[TC]** = Custo de comunicação da tarefa em megabits;
- **[NTS]** = Número de tarefas submetidas pelo usuário da grade;
- **[NTC]** = Número de tarefas atendidas deste usuário da grade;
- **[PCU]** = Poder computacional cedido pelo usuário em megaflops/s;
- **[TCR]** = Instante de criação da tarefa;

Variáveis utilizadas para o escalonador de recursos:

- **[PP]** = Poder computacional do recurso;
- **[LC]** = Rota de comunicação até o recurso em megabits/s;
- **[TCT]** = Custo computacional da tarefa selecionada;
- **[TCMT]** = Custo de comunicação da tarefa selecionada;
- **[NTE]** = Número de tarefas em execução no recurso;
- **[MFE]** = Total de megaflops sendo processado no recurso;

Na figura 4.5 é apresentado um exemplo de como seria a implementação do escalonador Workqueue definido pela linguagem do gerador. Conforme pode ser observado, a primeira linha indica o nome do escalonador (“WQ”). Ele seleciona uma tarefa aleatória, descrito na quarta linha. A tarefa é enviada para um recurso livre, pois a restrição indicada na segunda linha, em conjunto com a regra da última linha, limita o número de tarefas a uma por máquina.

```
SCHEDULER WQ
RESTRICT 1 TASKPER RESOURCE
STATIC
TASK SCHEDULER: RANDOM
RESOURCE SCHEDULER: CRESCENT ( [NTE] )
```

Figura 4.5: Algoritmo Workqueue

4.2.3 Gerenciamento dos escalonadores no iSPD

A possibilidade de gerar modelos para novas políticas de escalonamento demanda a existência de um processo de gerenciamento dos escalonadores no iSPD, de modo a controlar a inserção e remoção de novos escalonadores. A versão original do iSPD não previa esse componente, cuja descrição é feita nessa seção, evidenciando-se as dificuldades encontradas durante sua integração ao iSPD.

A figura 4.6 descreve a organização do componente de geração de escalonadores e as etapas realizadas para adição de um escalonador no iSPD, através da interface gráfica e conjunto de compiladores. Dessa forma é possível observar a interação entre os elementos existentes e os adicionados ao iSPD.

Para a geração de um escalonador o usuário deve descrever o comportamento da política de escalonamento através da interface gráfica. Essa descrição é transformada inicialmente em código em uma linguagem intermediária. Este código é compilado e convertido para código Java utilizando as interfaces do iSPD. O código Java então é

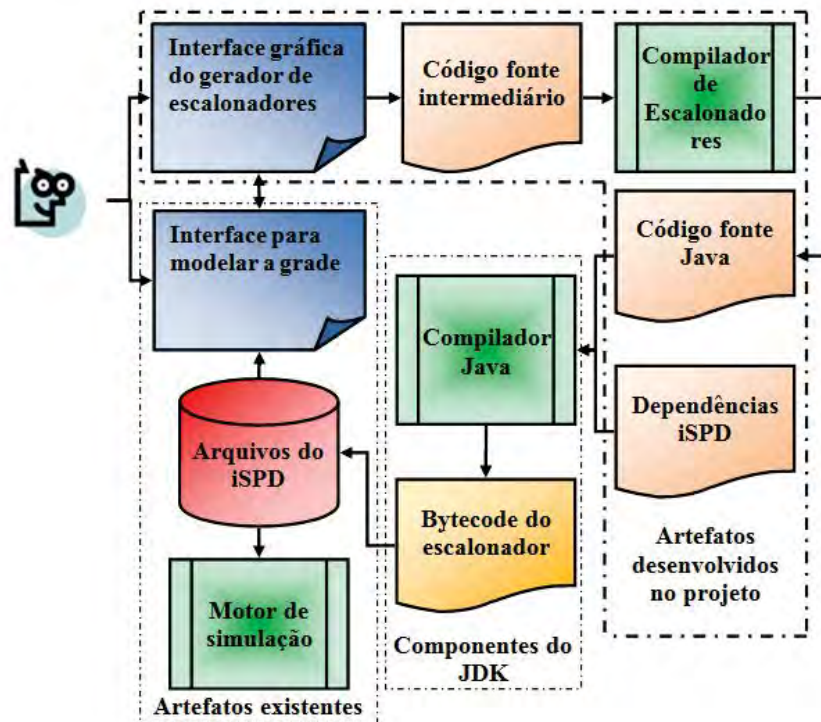


Figura 4.6: Etapas para adição de escalonadores no iSPD

compilado e os bytecodes da classe são adicionados ao sistema de arquivos do iSPD, para finalmente serem utilizados pela interface de modelagem da grade e pelo motor de simulação.

Para implementar a estrutura descrita na figura 4.6 necessitou-se alterar a implementação original de diversos componentes do iSPD, como por exemplo o módulo para tratamento de filas. O gerenciador da rede de filas retém diversas funções, incluindo a de escalonamento. Esta maneira de realizar o controle limitava o modelo, pois uma mesma política de escalonamento era aplicada em toda grade, não permitindo o estudo de escalonadores hierárquicos, ou distribuídos. Como a rede de filas possuía uma política de escalonamento fixa, seria necessário realizar alteração diretamente neste componente para inserir novas políticas, algo complexo e pouco eficiente para os objetivos do iSPD.

Como não existia um elemento com a função específica de realizar o escalonamento,

foi necessário criar um artefato escalonador, e uma forma de armazenar todos os escalonadores que seriam criados pelo usuário. Assim especificou-se uma interface para realizar o escalonamento, que seria utilizada pelo motor de simulação, e um sistema de arquivos que permitiria ao iSPD utilizar e armazenar todos os escalonadores criados pelos usuários no momento que for necessário.

4.3 Especificação do Motor de simulação

O motor de simulação é o módulo que realiza efetivamente a simulação do modelo da grade. Este modelo é obtido com a assistência do módulo interpretador de modelos, que converte o modelo icônico construído pelo usuário para o modelo de filas utilizado pelo núcleo de simulação.

Conforme indicado na seção 4.2.3, para realizar o estudo de escalonadores em grades computacionais é necessário que o simulador execute diversas políticas de escalonamento simultaneamente. Porém o protótipo inicial do motor, desenvolvido e descrito em (Oliveira et al., 2010), não permitia a alteração do algoritmo de escalonamento de forma simples. Visando permitir a execução de vários escalonadores distintos, o processo de escalonamento deve ser retirado do componente central, que realiza o gerenciamento da rede de filas, para ser realizado de forma local, em cada máquina mestre modelada na grade. O escalonador também necessitaria ser independente do mestre, para permitir a execução de novos escalonadores criados pelo usuário.

Portanto, o motor de simulação foi remodelado seguindo uma organização dividida em três elementos independentes, cada um realizando uma parte importante no processo de simulação e interagindo com os outros, conforme pode ser observado na figura 4.7. A base para o novo motor de simulação é o simulador de eventos discretos, herdado do protótipo apresentado na figura 3.5, e realiza o controle do tempo de simulação e a ordem de atendimento dos eventos. Sobre este primeiro componente foi construído o

modelo de filas, formado por uma rede de filas com diversos tipos de servidores. Neste modelo cada servidor é autônomo, realizando seu próprio controle, e atende as tarefas conforme a ocorrência de eventos. O último componente é responsável por realizar o escalonamento, estando presente em todo centro de serviço mestre. Portanto cada mestre possui uma implementação de alguma política de escalonamento, que pode ser distinta para cada mestre.

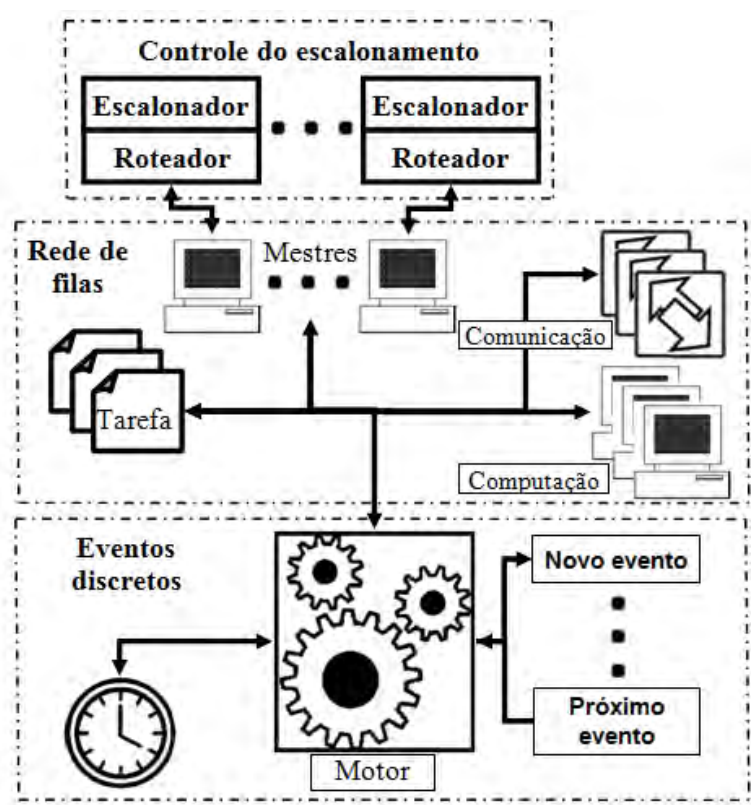


Figura 4.7: Versão atual do motor de simulação

As modificações foram definidas visando tornar o motor de simulação mais flexível, com elementos independentes, permitindo alterá-los sem modificar os demais componentes. No modelo de filas cada servidor teria um comportamento distinto e realizaria o próprio controle, porém com estrutura padronizada, de forma a facilitar a adição de novos tipos de servidores em projetos futuros. A seguir são detalhados os novos

componentes presentes no motor de simulação.

4.3.1 Simulador de eventos discretos

No iSPD o processo de simulação é realizado seguindo um modelo discreto orientado a eventos. O tempo é uma variável independente, e as demais variáveis de estado (que descrevem o modelo) são funções dependentes do tempo. Por ser um modelo discreto, os valores das variáveis dependentes são alterados em pontos específicos no tempo de simulação, a partir da ocorrência de eventos.

Como a simulação é orientada a eventos, a definição dos mesmos é uma etapa imprescindível no desenvolvimento do núcleo de simulação. Seguindo o modelo de filas criaram-se três eventos básicos, implementados de forma distinta em cada tipo de centro de serviço:

- **Chegada de cliente no servidor:** Evento no qual um cliente é adicionado à fila de atendimento do centro de serviço;
- **Atendimento do cliente:** Evento em que um cliente recebe o atendimento fornecido pelo centro de serviço;
- **Saída de cliente do servidor:** Evento no qual um cliente deixa o servidor, podendo ser encaminhado a outro centro de serviço se necessário.

Além destes eventos, também foram adicionadas mais duas classes de eventos:

- **Escalonamento:** Substitui o atendimento do cliente em um nó que represente um mestre;
- **Mensagens:** Alteram o estado dos clientes durante a execução da simulação, apenas nos centros de serviços de computação.

O funcionamento deste componente do iSPD é apresentado na figura 4.8. O processo de simulação segue o algoritmo Event Scheduling/Time Advance (Banks et al., 2001). O controle do processo de simulação é realizado pelo simulador de eventos discretos, que manipula a lista de eventos futuros (LEF), inserindo e retirando eventos da forma correta; atualiza o tempo de simulação a cada passo efetuado; e executa o evento mais recente da LEF até que todos os eventos da lista sejam atendidos. Durante o atendimento de um evento pode ser agendados novos eventos, adicionando estes novos eventos a LEF.

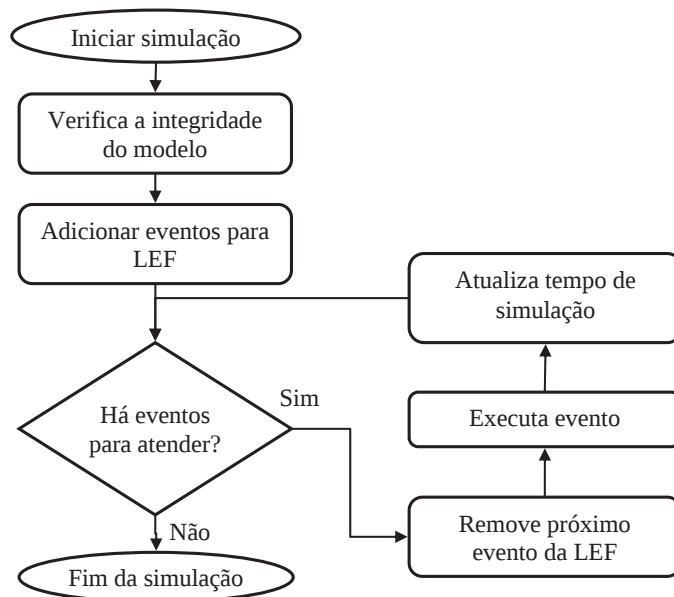


Figura 4.8: Processo de simulação

4.3.2 Modelo de filas

O modelo de filas é formado por um conjunto de centros de serviço sobre o qual o núcleo realiza a simulação. Cada ícone presente no modelo icônico construído pelo usuário é transformado em um ou mais centros de serviços no modelo de filas. A infraestrutura computacional da grade simulada é representada por uma rede de filas interconectando

os centros de serviços. As tarefas criadas como carga de trabalho compõem a população de clientes que utilizam os serviços providos pelos servidores.

Os centros de serviços no iSPD podem ser classificados entre os de comunicação ou de computação, como descritos a seguir:

- **Centro de serviços de comunicação**

- **Centro de serviços de conexão (link):** segue o modelo de uma fila com um servidor, com a disciplina de atendimento FIFO. Este centro de serviço realiza a conexão entre dois centros de serviços de comunicação ou processamento;
- **Centro de serviços de comutação (switch):** segue o modelo de múltiplas filas com um servidor, com a disciplina de atendimento FIFO. Este centro de serviço realiza a conexão entre diversos centros de serviços (comunicação ou processamento), como os nós de um cluster;
- **Centro de serviços de internet:** segue um modelo específico de uma fila com múltiplos servidores, no qual o número de servidores tende ao infinito. Dessa forma sempre há um servidor disponível para atender um cliente, mantendo vazia a fila. Este centro de serviço realiza a conexão entre diversos centros de serviços (comunicação ou processamento), utilizado para conectar os nós de uma grade;

- **Centro de serviços de processamento**

- **Centro de serviços de máquina:** segue o modelo de uma fila com múltiplos servidores, com a disciplina de atendimento FIFO. Este centro de serviço realiza o processamento das tarefas, podendo conter um ou mais servidores para representar máquinas com um único processador ou com vários processadores (e memória compartilhada);

- **Centro de serviços de mestre:** segue o modelo de uma fila com múltiplos servidores, com a disciplina de atendimento FIFO. Semelhante ao centro de serviços de máquina, porém sua função é realizar o escalonamento das tarefas, portanto encaminhando clientes para o local no qual serão atendidos;

A população de clientes do modelo de filas pode ser classificada em duas categorias, as tarefas e as mensagens. A tarefa é um consumidor de serviços de processamento e comunicação seguindo o modelo de execução BoT, enquanto as mensagens são consumidores de serviços de comunicação.

4.3.3 Mestre-escravo e o escalonamento

O paradigma de programação implementado no motor de simulação do iSPD é o mestre-escravo. Nele as tarefas são criadas e escalonadas pelo mestre, enquanto o escravo apenas realiza o processamento das tarefas.

A figura 4.9 apresenta o comportamento implementado na máquina mestre e na escrava. O mestre recebe as tarefas durante a simulação (conforme tempo indicado nas tarefas), e realiza o escalonamento para um de seus escravos. Enquanto isto o escravo executa o processamento de uma tarefa e devolve o resultado ao mestre que submeteu a tarefa.

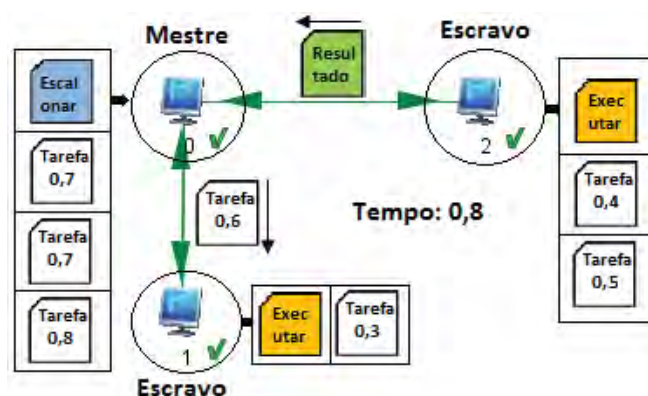


Figura 4.9: Paradigma mestre-escravo

Para tornar o modelo mais amplo, é permitido existir mestres independentes (dois ou mais mestres em um modelo sem nenhuma relação entre eles), assim como formar hierarquias entre eles (um mestre ser escravo de outro). Por último, ainda é possível um escravo pertencer a vários mestres. Estas características facilitam o estudo de diversos modelos de escalonadores.

4.4 Implementação do módulo de escalonadores

A partir das especificações apresentadas nas seções anteriores se implementou a versão atual do iSPD. Nesta seção e na seguinte se descreve tal implementação, começando com o módulo de escalonadores. Para proporcionar uma visão geral do sistema, a figura 4.10 apresenta o diagrama de pacotes do simulador após a adição do módulo de escalonadores.

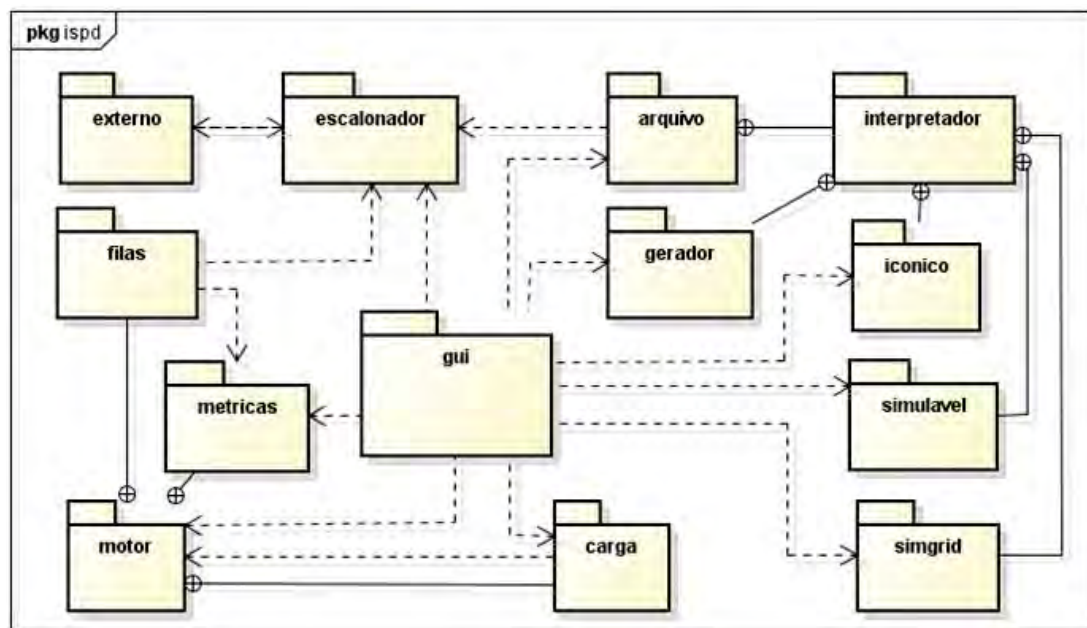


Figura 4.10: Diagrama de pacotes UML

Observe-se que cada módulo está separado em um pacote específico, sendo o pacote principal o de Interface gráfica (“gui”), responsável por todas as janelas construídas para

interação com o usuário, e apresentação de resultados. Este pacote se relaciona com os demais para utilizar as funcionalidades desenvolvidas.

Na figura 4.11 são apresentadas as classes que compõem o módulo de escalonadores, e a relação delas com outros componentes do iSPD. O principal componente do módulo de escalonadores é o pacote “escalador”. Neste pacote estão contidas as interfaces necessárias para a manipulação e utilização dos escalonadores pelo simulador. A interface “Mestre” e a classe abstrata “Escalador” realizam a conexão entre o modelo de filas e o módulo de escalonadores.

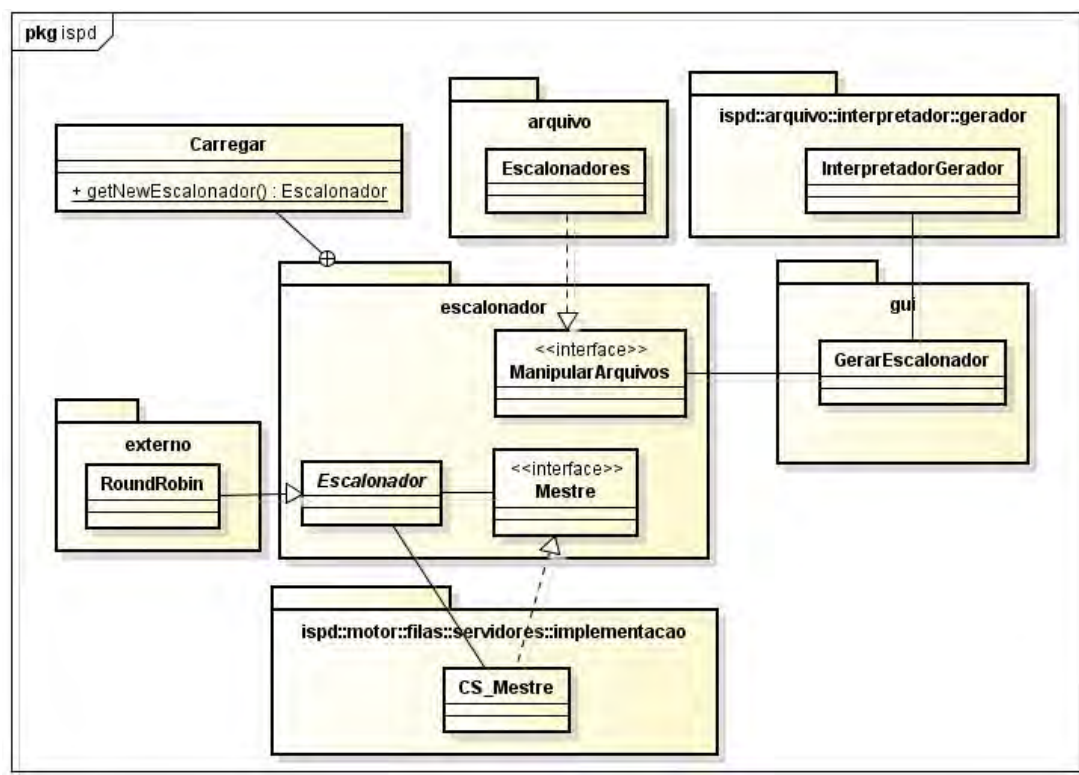


Figura 4.11: Interfaces do pacote "escalador"

A classe “Carregar” permite ao motor de simulação instanciar um escalonador dinamicamente, podendo ser um escalonador padrão do iSPD ou um construído pelo usuário. Enquanto a interface “ManipulaArquivo” fornece os métodos bases para criar, compilar e adicionar um escalonador ao iSPD.

O pacote “externo” contém um conjunto de escalonadores. Apesar de fazer parte do iSPD, seu nome é uma referência ao fato de ser criado um diretório externo ao iSPD (mantendo o mesmo nome), no qual são salvos todos os escalonadores modelados pelo usuário.

O pacote “arquivo” domina uma série de classes que manipulam arquivos utilizados pelo simulador. Neste pacote adicionou-se uma classe responsável por salvar classes Java, manter uma estrutura de arquivos e fazer chamada a um compilador Java. Também acrescentaram-se a este pacote todas as classes envolvidas na análise e geração de algoritmos, por meio da gramática do gerador. Por último, foi adicionado ao pacote “gui” a interface do gerador descrita anteriormente.

Na figura 4.12 é apresentado o diagrama de classe das classes utilizadas pelo Gerenciador de escalonamento. Na figura pode-se observar as relações entre o gerenciador e o gerador e a interface principal do iSPD. Lembrando-se que no momento, a inserção dos novos algoritmos de escalonamento é possível apenas com o uso de um compilador Java externo ao simulador.

O pacote “gerador” foi representado de forma simplificada, apenas com a classe de conexão entre a interface gráfica e o gerador de código implementado com a ferramenta JavaCC 5.0. A classe “InterpretadorGerador” recebe um arquivo e segue as etapas necessárias para validar a gramática e gerar um escalonador seguindo as interfaces do iSPD.

4.5 Implementação do Motor de simulação

Nesta seção são apresentadas as classes desenvolvidas para realizar as funcionalidades especificadas para o novo motor de simulação. A seção é dividida, entre o desenvolvimento dos três componentes do motor, anteriormente apresentados na figura 4.7.

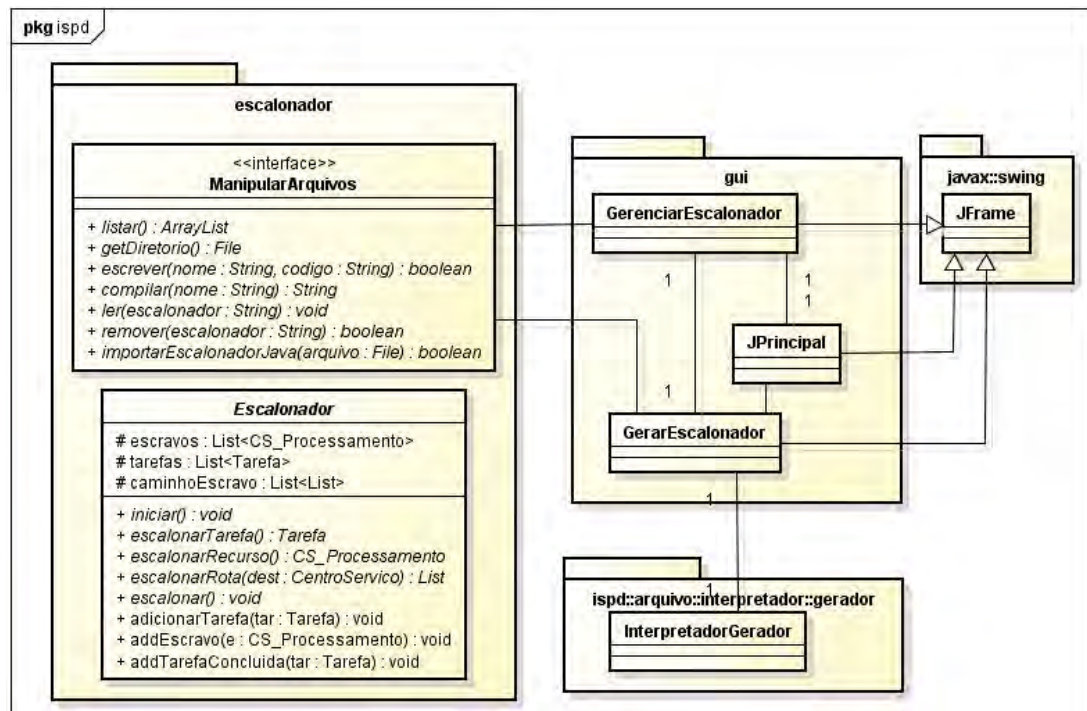


Figura 4.12: Diagrama de Classe Gerenciador de Escalonadores

4.5.1 Simulador de eventos discretos

O diagrama de classes UML das classes envolvidas com a simulação de eventos discretos é apresentado na figura 4.13. As classes “Tarefa” e “RedeDeFilas” foram representadas de forma simplificada, visando não poluir a imagem. A classe responsável por executar a simulação realizando os passos descritos na figura 4.8 é a “Simulacao”. Como pode ser observado por meio dos relacionamentos, mostrados na figura 4.13, a classe que realiza a simulação é composta por um conjunto de tarefas (implementando a interface “Cliente”), uma lista de eventos futuros, e uma rede de filas.

Os eventos futuros (classe “EventoFuturo”) são mantidos em uma lista ordenada pelo tempo de ocorrência do evento. O número de eventos existentes em uma simulação é volátil, iniciando sempre com o mesmo número de tarefas criadas. A classe “RedeDeFilas” contém o modelo da grade, criado por meio da interface gráfica, já convertido no

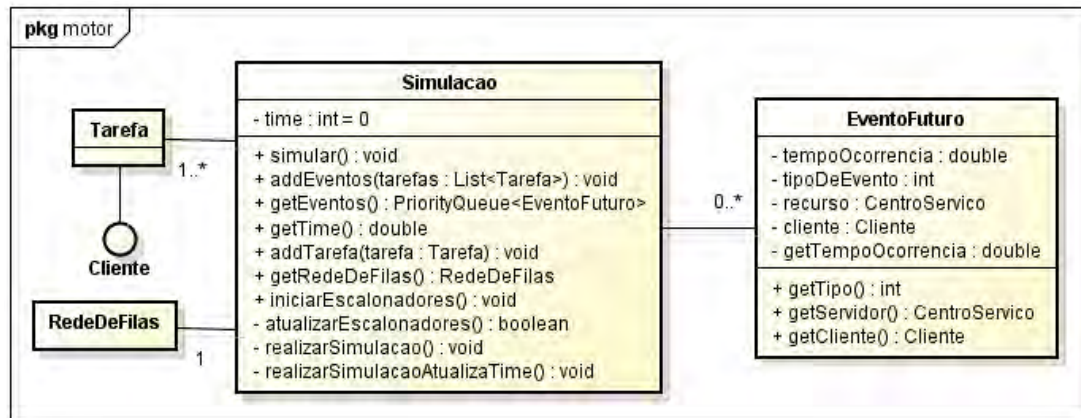


Figura 4.13: Diagrama UML do Simulador de eventos discretos

modelo de filas.

4.5.2 Modelo de filas

Para construir o modelo de filas devem ser instanciados diversos centros de serviços, e realizada a interconexão dos mesmos. Com o modelo montado, seus centros de serviços são agrupados na classe “RedeDeFilas” para então iniciar a simulação. As classes abstratas que compõem os servidores do modelo de filas são mostradas na figura 4.14.

A figura 4.15 apresenta as classes que representam os clientes do modelo de filas. Conforme pode ser observado, os dois tipos de clientes construídos são implementações da interface “Cliente”, sendo “Tarefa” um consumidor de serviços de processamento e comunicação, enquanto “Mensagem” é apenas um consumidor de serviços de comunicação.

4.5.3 Escalonamento

Conforme proposto, realizou-se a separação da política de escalonamento do processo servidor das redes de filas. Para fazer essa separação criou-se uma classe abstrata “Escalonador”, fazendo que as classes que realizam o escalonamento herdam os métodos abstratos desta classe, implementados conforme sua política de escalonamento. Já no

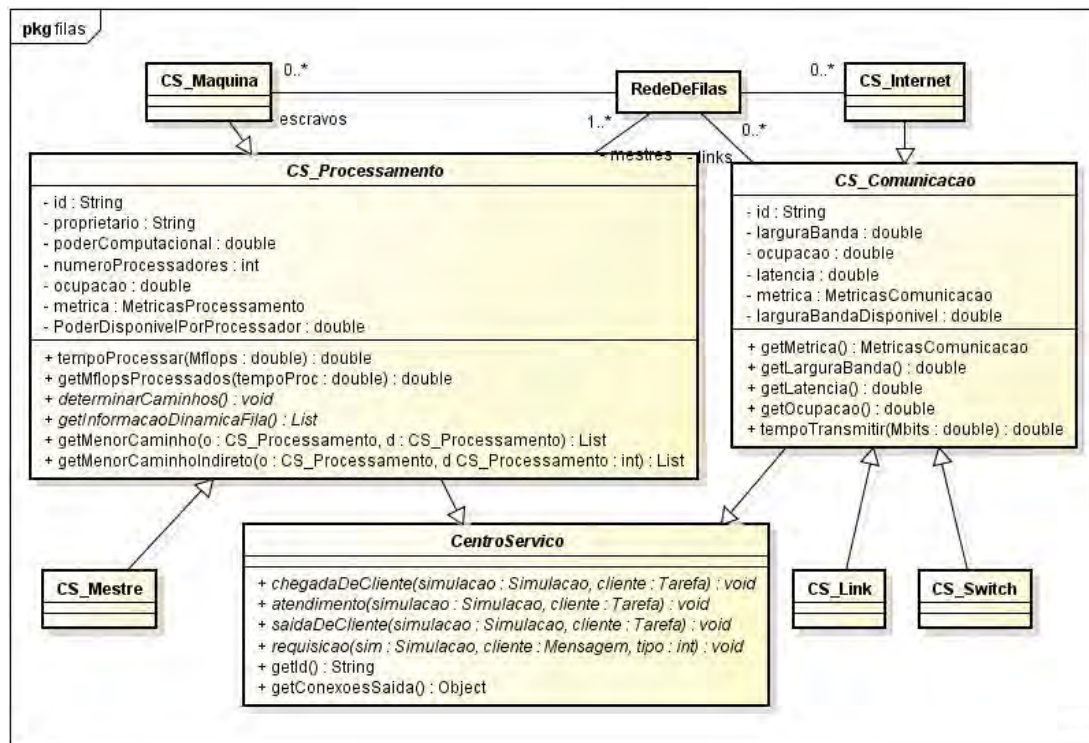


Figura 4.14: Diagrama UML dos servidores do modelo de filas

lado do modelo de filas, o centro de serviços deve implementar uma interface “Mestre” para se comunicar com o escalonador. A figura 4.16 apresenta o diagrama de classe UML das classes citadas. A classe “CSMestre” faz parte da rede de filas e possui um objeto escalonador, que acessa a rede de filas através da interface “Mestre”. No exemplo da figura a classe “Workqueue” implementa os métodos necessários do escalonador, que seriam usados pelo centro de serviços “CSMestre” para realizar o escalonamento através do algoritmo Workqueue.

As classes presentes na figura 4.16 foram projetadas para permitir a alteração das políticas de escalonamento (ou rede de filas) sem necessitar de alterar outras classes, permitindo assim utilizar novos objetos para realizar o escalonamento sem a necessidade de compilar ou alterar nada no iSPD, além da própria nova classe de escalonamento.

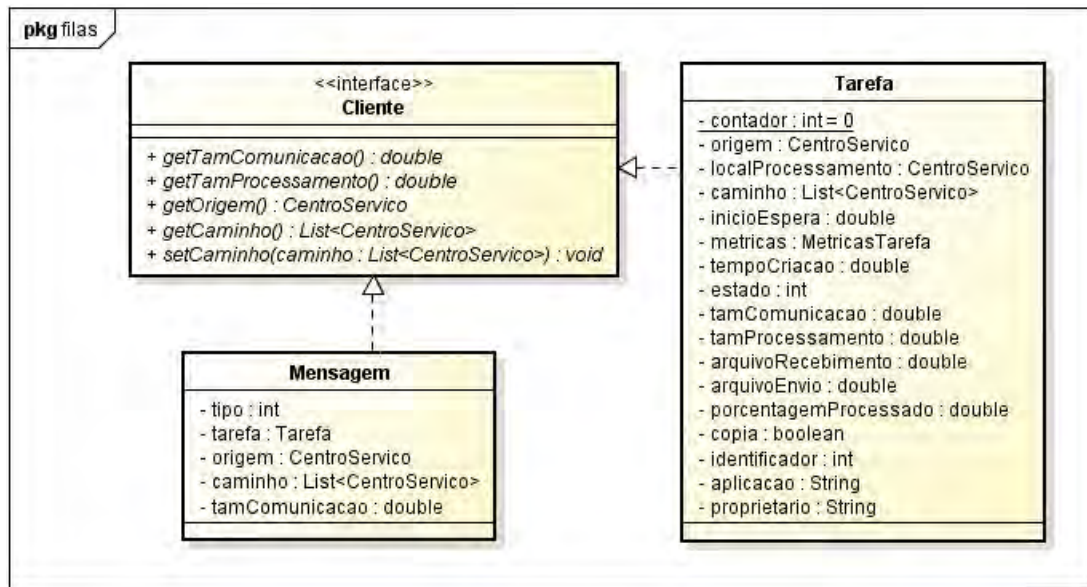


Figura 4.15: Diagrama UML dos clientes do modelo de filas

4.6 Considerações finais

Neste capítulo apresentou-se a modelagem e o desenvolvimento do módulo de escalonadores, assim como das modificações realizadas para inseri-lo no iSPD. O módulo apresentado é responsável por gerar algoritmos de escalonamento, organizar os arquivos dos escalonadores modelados no simulador, e carregar as classes durante a simulação de uma grade. As modificações propostas para o motor de simulação tiveram o objetivo de torná-lo mais flexível e facilitar o estudo de escalonadores de tarefas. Espera-se que com esse módulo a usabilidade deste simulador seja ampliada, permitindo experimentos mais significativos com o mesmo.

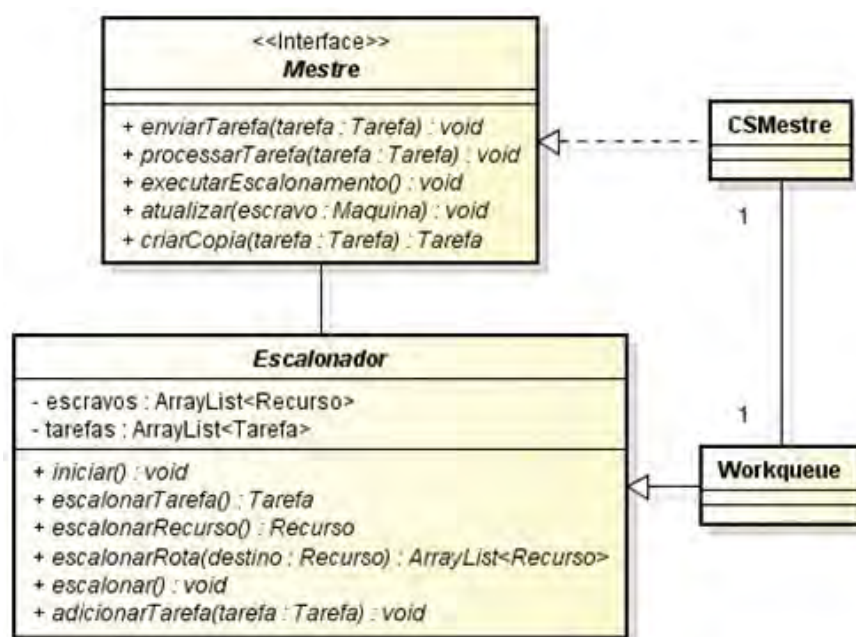


Figura 4.16: Escalonamento no motor de simulação

Capítulo 5

Ambiente e resultados experimentais

Neste capítulo se apresenta a validação da proposta de geração automatizada de escalonadores no iSPD. Foram realizados testes com quatro objetivos distintos: O primeiro busca a validação do motor de simulação através de sua comparação com outro simulador; O segundo objetivo é validar a geração de escalonadores comparando resultados entre um algoritmo implementado diretamente na classe Java, e sua versão construída pelo gerador de escalonadores; Outro objetivo foi analisar o comportamento obtido com uma política estática e sua contraparte dinâmica, ambas criadas com auxílio do gerador de escalonadores; O último objetivo nos testes foi verificar a eficiência do simulador no que diz respeito ao tempo para produção de resultado. Nas próximas seções os resultados obtidos nos testes para cada um desses objetivos são apresentados.

5.1 Validação do motor de simulação

Visando avaliar a precisão dos resultados obtidos pelo iSPD foram feitos diversos testes durante e após seu desenvolvimento. Como os testes apresentados aqui tem o objetivo

de validar os componentes de escalonamento, foram realizadas medições de um mesmo sistema em um ambiente real e de sua simulação em modelos para o iSPD e para o SimGrid.

5.1.1 Ambiente computacional

O ambiente computacional utilizado foi o cluster presente no laboratório do GSPD (financiado pela FAPESP, processo nº 2008/09312-7), para o qual foram criados modelos para as avaliações nos simuladores. A seguir são descritas as configurações de *software* e *hardware* desta plataforma.

Configuração de *Hardware*

As máquinas do cluster são conectadas por um switch 3Com® OfficeConnect® de 16 portas na velocidade de 100 Mb/s. O cluster, representado na figura 5.1, é composto por nove máquinas com a seguinte configuração:

- Processador: Intel(R) Pentium(R) Dual CPU E2160 @ 1.80GHz
- L2 Cache Size: 1024 KB
- Memória RAM: 2048 MB
- Disco Rígido: 30GB

Configuração de *Software*

O sistema operacional presente nas máquinas do cluster é o Debian GNU/Linux 5.0 com o kernel Linux versão 2.6.26-2-686. Para compilar os programas escritos na linguagem C foi utilizado o compilador GNU Compiler Collection (GCC) versão 4.3.2. Os principais programas construídos para os testes no cluster empregam MPI (*Message Passing Interface*) e para compilá-los foi utilizado o compilador Open MPI 1.2.7rc2.

A capacidade de processamento das máquinas do cluster foi determinada através de *benchmark* próprio. Através dele chegou-se a uma velocidade de processamento de

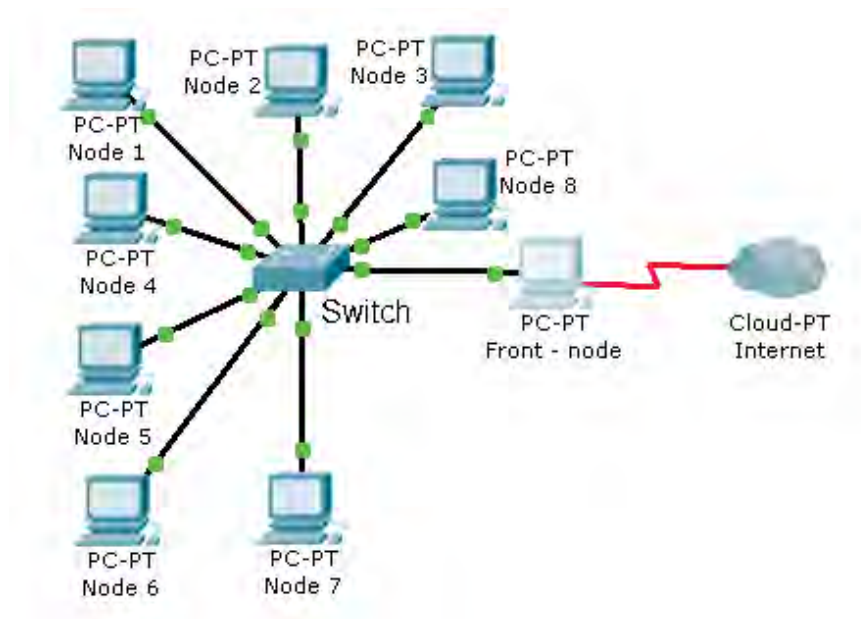


Figura 5.1: Cluster GSPD (GSPD, 2012)

768,9 milhões de instruções por segundo. Esse valor foi usado como capacidade de processamento nas simulações.

5.1.2 Algoritmos de escalonamento

Para os testes foram utilizados dois algoritmos de escalonamento diferentes, cujo comportamento é descrito a seguir:

- **Round-Robin:** as máquinas presentes no sistema são organizadas em uma lista circular, e as tarefas são distribuídas na ordem de chegada para o recurso atual da lista. Após este passo a lista é movida uma posição e atribui-se a próxima tarefa, continuando até enviar todas as tarefas.
- **Workqueue:** as tarefas são submetidas, na ordem de chegada, aos elementos de processamento aleatoriamente (apenas uma tarefa para cada processador), e à medida que um processador devolve o resultado de uma tarefa, ele recebe outra tarefa. Este processo se repete até terminarem todas as tarefas do escalonador.

Para realizar os testes no cluster, desenvolveu-se dois programas, um para cada escalonador, utilizando MPI e threads. Seu comportamento é descrito na figura 5.2, sendo composto por um processo mestre e oito processos escravos um para cada máquina do cluster. O processo mestre cria as tarefas, distribuindo com as políticas de escalonamento Round-Robin ou Workqueue, e espera os resultados. Já os processos escravos possuem três threads, a primeira recebe as tarefas, a segunda processa e a última retorna os resultados. No término do programa uma mensagem circula entre os processos para indicar que não há mais tarefas para serem executadas.

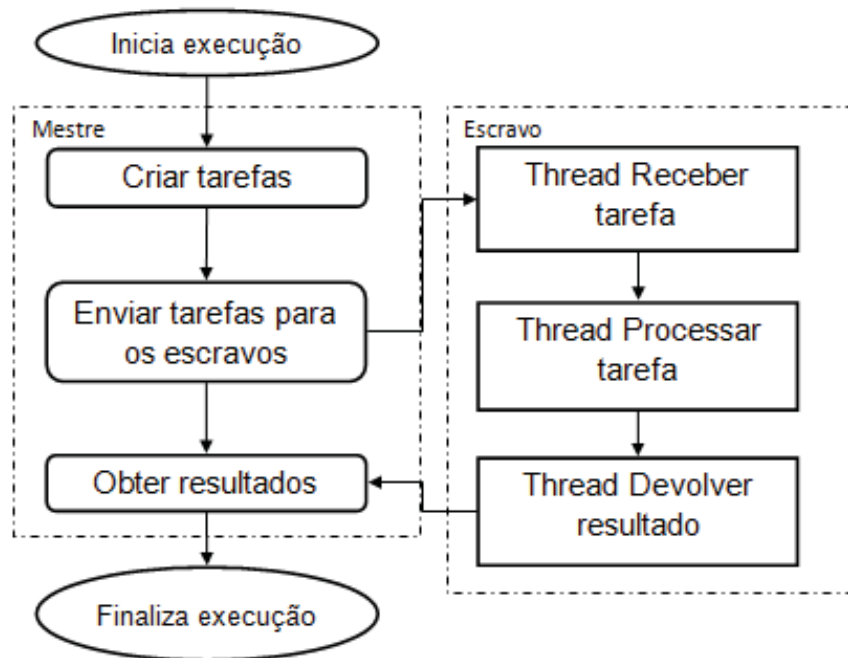


Figura 5.2: Aplicação para o cluster

Nos simuladores criaram-se modelos para o cluster, e o programa nele executado. Diferentes quantidades de tarefas, com tamanho de computação e comunicação variando são usados nos testes, buscando avaliar o comportamento do iSPD em vários cenários.

5.1.3 Resultados

Os valores apresentados nos gráficos desta seção consistem na média dos resultados obtidos nas simulações e no cluster, sendo que em cada caso apresentado foram dez iterações.

O gráfico da figura 5.3a apresenta a média dos resultados obtidos no SimGrid, iSPD e cluster-GSPD para o algoritmo Round-Robin, enquanto a figura 5.3b refere-se ao resultados do Workqueue. Nos testes realizados cada tarefa demandava 384,45 Mflops e transferia 1 kbits. Os testes foram executados com 100, 1000, e 10000 tarefas.

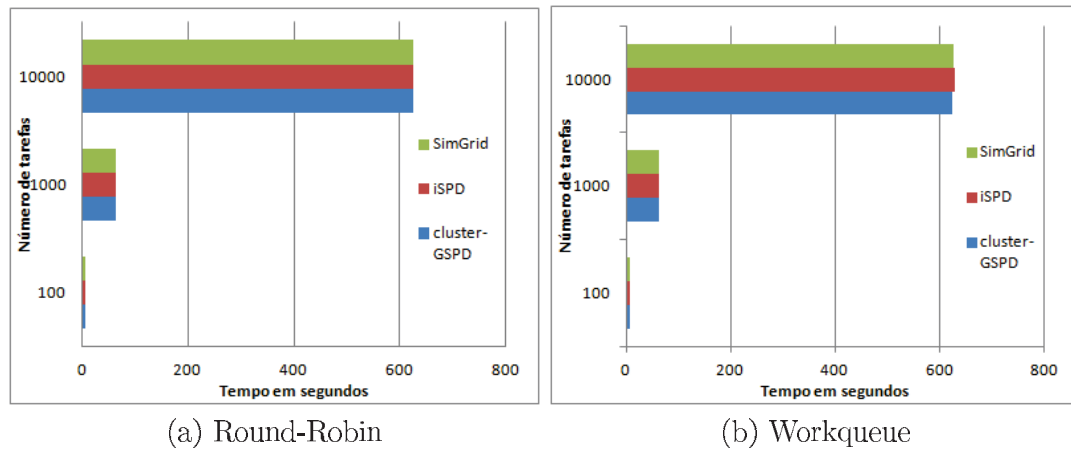


Figura 5.3: Gráficos dos testes variando o número de tarefas (a) Round-Robin. (b) Workqueue.

Como pode ser observado nos gráficos, os resultados obtidos nas simulações ficaram próximos dos obtidos com a aplicação real. Pode-se salientar que nos primeiros testes, com 1000 ou menos tarefas, o tempo simulado foi menor que o real, o que era esperado pois no programa real há mensagens de controle que não aparecem nos modelos simulados. Com um maior número de tarefas o tempo gasto com estas mensagens se torna menos significativo, diminuindo o erro percebido nos simuladores.

A tabela 5.1 apresenta os resultados obtidos com a simulação do algoritmo Round-

Robin e o Workqueue variando o número de tarefas, mantendo tamanho computacional fixo (384,45 Mflops). Dela se pode ver que os resultados entre os simuladores ficaram próximos com diferença média de 0,6 %. Comparando com o sistema real os resultados também foram bons, com erro médio de 2,2%, sendo que o erro diminuiu com um maior número de tarefas, chegando a menos de 1%.

Tabela 5.1: Resultados da simulação variando número de tarefas

Sistema	Algoritmo de escalonamento	Tarefas	Tempo (segundos)	precisão do resultado %
cluster-GSPD	Round-Robin	100	7,151	–
iSPD			6,55	91,58
Simgrid			6,521	91,18
cluster-GSPD	Workqueue	100	7,148	–
iSPD			6,85	95,82
Simgrid			6,54	91,48
cluster-GSPD	Round-Robin	1000	63,025	–
iSPD			62,59	99,3
Simgrid			62,633	99,37
cluster-GSPD	Workqueue	1000	63,086	–
iSPD			62,891	99,69
Simgrid			62,772	99,5
cluster-GSPD	Round-Robin	10000	624,42	–
iSPD			625,09	99,89
Simgrid			626,25	99,7
cluster-GSPD	Workqueue	10000	624,8	–
iSPD			628,79	99,36
Simgrid			627,52	99,56
cluster-GSPD	Round-Robin	100000	6237,25	–
iSPD			6250,09	99,79
Simgrid			6262,52	99,59
cluster-GSPD	Workqueue	100000	6242,2	–
iSPD			6287,77	99,27
Simgrid			6275,03	99,47

Os resultados apresentados no gráfico da figura 5.4 são referentes aos testes realizados variando-se a demanda de computação em cada tarefa. Foram criadas 50 tarefas com volume de comunicação variando de 1 kbits até 40 kbits, e processamento variando da seguinte forma:

- P: 384,45 – 1922,25 Mflops
- M: 1922,25 – 11533,5 Mflops
- G: 11533,5 – 38445 Mflops

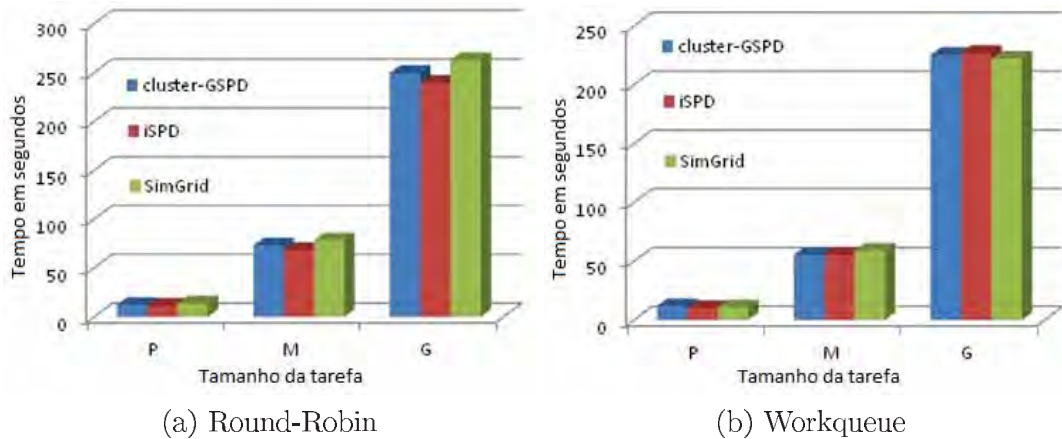


Figura 5.4: Gráficos dos testes variando o número de tarefas (a) Round-Robin. (b) Workqueue.

Comparando os resultados alcançados por cada um dos escalonadores pode-se observar que para tarefas com tamanho computacional fixo os resultados são muito próximos, enquanto que variando o tamanho das tarefas, o algoritmo Workqueue obteve melhor desempenho. Esse resultado era esperado pois o Workqueue evita que uma mesma máquina receba várias tarefas grandes, enquanto outra recebe várias tarefas pequenas.

5.2 Verificação da geração de escalonadores

Nesses testes buscou-se comparar políticas de escalonamento presentes por padrão no iSPD com versões construídas pelo gerador de escalonadores. O objetivo desta comparação é identificar a precisão obtida pelo gerador ao seguir uma regra específica,

comparada com um algoritmo desenvolvido manualmente para seguir a mesma regra.

5.2.1 Modelo simulado

O modelo de grade construído é apresentado na figura 5.5, sendo composto por um mestre (icon0) com uma conexão para internet, e conectado diretamente a um cluster (com dez máquinas, cada uma podendo processar 10 Mflops/s). No modelo também aparecem três máquinas escravas icon4, icon16 e icon6, conectadas ao mestre indiretamente, com os respectivos poderes computacionais: 1000, 100 e 500 Mflops/s.

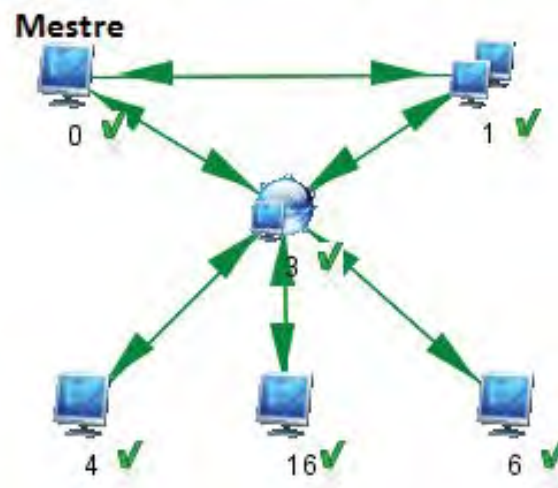


Figura 5.5: Segundo ambiente computacional

Na carga de trabalho configurada gerou-se dez mil tarefas, cada uma com 1000 megaflops para processar e 10 megabits de comunicação.

5.2.2 Algoritmos de escalonamento

Nestes testes foram utilizados três algoritmos de escalonamento, que são o Workqueue, Round-Robin e *Dynamic* FPLTF. Os dois primeiros foram descritos na seção 5.1. O código gerado pela interface para eles aparecem nas figuras 4.5 (Workqueue) e 5.6 (Round-Robin). Já o *Dynamic* FPLTF desenvolvido a partir do *Fastest Processor to*

Largest Task First (Menasce et al., 1995), tornando-o dinâmico. Seu objetivo está ligado ao dinamismo e à heterogeneidade dos recursos presentes em uma grade, buscando enviar as tarefas para os recursos mais rápidos e menos carregados, realizando inclusive a realocação das tarefas durante a execução (Silva, 2003). A figura 5.7 apresenta o código gerado para implementar este escalonador, em que pode ser observado que ele faz atualização na rede a cada segundo, e após receber o resultado de uma tarefa pede para serem devolvidas todas as tarefas que não estão sendo executadas para realizar o escalonamento delas novamente.

```
SCHEDULER R_R
STATIC
TASK SCHEDULER: FIFO
RESOURCE SCHEDULER: FIFO
```

Figura 5.6: Algoritmo Round-Robin

```
SCHEDULER DFPLTF
DYNAMIC TIME INTERVAL 1.0
TASK SCHEDULER: DECREASING ( [TCP] )
RESOURCE SCHEDULER: CRESCENT(( [TCT] / [PP] ) + ( [MFE] / [PP] ))
RETURN ALL TASK OF ALL RESOURCE WHEN TASK COMPLETED
```

Figura 5.7: *Dynamic* FPLTF

5.2.3 Resultados

A figura 5.8 apresenta os gráficos referentes ao total de megaflops processados em cada um dos recursos da grade para cada uma das políticas. Na figura 5.8a, referente ao Round-Robin, pode-se observar que o mesmo número de tarefas é distribuído para cada máquina. Já para o algoritmo Workqueue, figura 5.8b, as máquinas com maior poder computacional atendem mais tarefas que a máquina icon16 e o cluster (icon1), neste

caso o cluster atendeu uma quantidade menor porque ele possui um nó principal, e deste modo o mestre (icon0) o identifica como sendo apenas uma máquina, e pelas características do algoritmo de escalonamento, o cluster recebe somente uma tarefa por vez. Contudo, o cluster ainda atendeu mais tarefas que a máquina icon16 pois possui uma conexão direta com o mestre, estando mais próximo na rede. Por último, na figura 5.8c é apresentado o gráfico relativo ao algoritmo *Dynamic FPLTF*, em que a variação do número de tarefas executados em cada máquina fica mais evidente, processando-se mais na máquina mais potente.

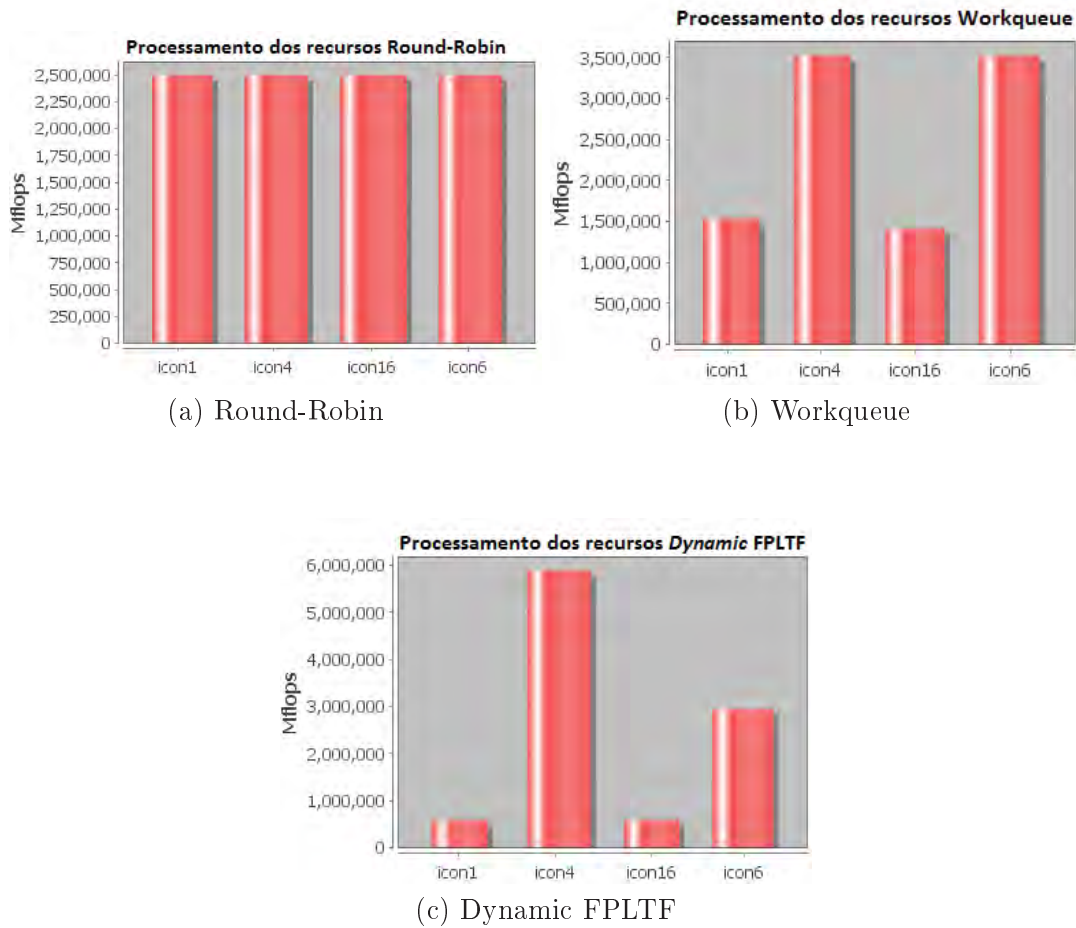


Figura 5.8: Megaflops executados em cada máquina (a) Round-Robin. (b) Workqueue. (c) Dynamic FPLTF.

A tabela 5.2 apresenta a quantidade exata de tarefas executadas em cada recurso da grade simulada, bem como tempo total simulado para cada algoritmo. Como não houve variação na quantidade de comunicação e processamento das tarefas nas simulações, os valores foram justamente os mesmos para cada par de algoritmos, exceto para o Dynamic FPLTF, pois esta política é mais complexa, e também dinâmica, ocorrendo mais variações durante os testes, porém com diferença máxima inferior a 4

Tabela 5.2: Total de tarefas executadas em cada recurso

	Tarefas Icon1	Tarefas Icon4	Tarefas Icon16	Tarefas Icon6	Tempo total de simulação
Workqueue padrão	1537	3526	1411	3526	18513,2
Workqueue gerado	1537	3526	1411	3526	18513,2
Round-Robin padrão	2500	2500	2500	2500	25003.1
Round-Robin gerado	2500	2500	2500	2500	25003.1
Dynamic FPLTF padrão	583	5889	587	2941	9530.41
Dynamic FPLTF gerado	600	5856	608	2936	9728.09

Conforme pode ser observado com os resultados apresentados, o gerador seguiu de forma precisa a política indicada. Como resultado ambos os escalonadores, os desenvolvidos diretamente na classe Java e os construídos pelo gerador, obtiveram o mesmo comportamento para cada política, apesar de possuírem implementações distintas.

5.3 Avaliação entre política estática e dinâmica

Esta seção busca analisar algumas características do gerador, verificando a distinção entre informações dinâmicas e estáticas. Desta forma as propriedades que se deseja avaliar estão ligadas ao tratamento dos usuários da grade simulada e da utilização de algoritmos dinâmicos e estáticos. A seguir é apresentado o modelo criado e em seguida a política de escalonamento.

5.3.1 Ambiente computacional e cargas de trabalho

A figura 5.9 apresenta o ambiente computacional utilizado neste teste, composto pelas grades “Grade1” e “Grade2”, delimitadas pelas elipses na figura. Cada grade possui uma máquina mestre, independente, sendo que o mestre presente na “Grade2” executa um escalonador estático, enquanto o mestre da “Grade1” utiliza um escalonador dinâmico. As máquinas contidas no retângulo central da figura, interligadas pelo ícone de internet, são compartilhadas por ambas as grades. Cada máquina tem velocidade de 500 Mflops/s, e todos os canais de comunicação possuem banda de 100 Mb/s.

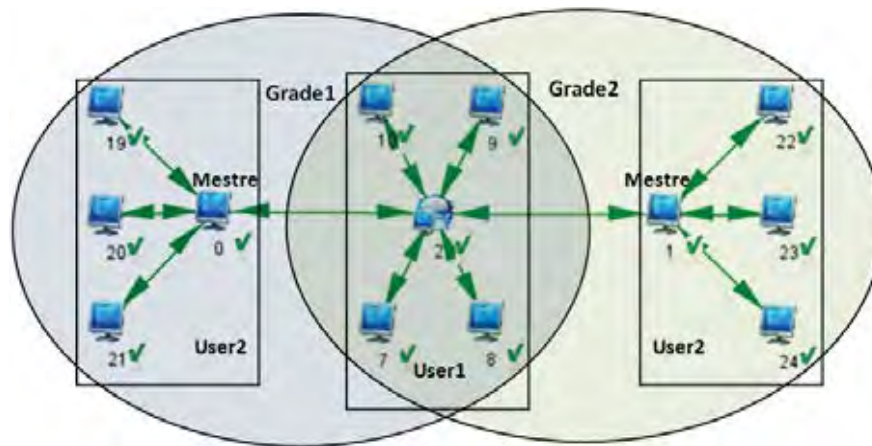


Figura 5.9: Terceiro ambiente computacional

O modelo construído possui três usuários. Na figura 5.9 as máquinas pertencentes aos usuários user1 e user2 são agrupadas em três retângulos. O terceiro usuário denominado user3 não oferece nenhum recurso, apenas utiliza a grade sem contribuir com recursos. O poder computacional oferecido por cada usuário é o seguinte:

- Grade1:
 - User1: 2000 Mflops/s;
 - User2: 1500 Mflops/s;
 - User3: Zero;

- Grade2:
 - User1: 2000 Mflops/s;
 - User2: 1500 Mflops/s;
 - User3: Zero;

A carga de trabalho submetida à grade, nos testes, é formada por 2000 tarefas com instante de ocorrência com distribuição exponencial, por usuário, totalizando 6000 tarefas, das quais cada usuário distribuiu mil em cada mestre. A quantidade de processamento de cada tarefa variava de cem mil até um milhão de megaflops, enquanto a carga de comunicação foi fixada em 10 megabits. Esta carga foi moldada visando ocupar cada processador durante um tempo considerável, e consequentemente aumentar a fila por este recurso.

5.3.2 Algoritmo de escalonamento

A figura 5.10 apresenta o código fonte dos algoritmos de escalonamento implementados. As políticas para distribuição de tarefas e seleção de recurso são idênticas, contudo o algoritmo “SchedulerE” é estático, enquanto o “SchedulerD” é dinâmico. Conforme já citado o algoritmo de escalonamento “SchedulerD” é utilizado na “Grade1”, e o “SchedulerE” na “Grade2”.

A política implementada nos escalonadores prioriza as tarefas do usuário que cedeu maior quantidade de processamento a grade, selecionando sempre que possível a tarefa deste usuário, e quando não existir mais tarefas dele na fila será selecionada a do próximo usuário, de acordo com a quantidade de processamento fornecida. Esta política envia tarefas sempre para o recurso com o menor número de tarefas em execução, até o limite de duzentas tarefas na fila de cada escravo. Este limite de tarefas enviadas foi inserido visando aumentar a quantidade de tarefas na fila do escalonador, permitindo que a política de ordenação das tarefas ficasse mais evidente.

```

SCHEDULER SchedulerE
RESTRICT 200 TASKPER RESOURCE
STATIC
TASK SCHEDULER: DECREASING ( [PCU] )
RESOURCE SCHEDULER: CRESCENT ( [NTE] )

```

(a) Estático

```

SCHEDULER SchedulerD
RESTRICT 200 TASKPER RESOURCE
DYNAMIC TASK ENTRY
TASK SCHEDULER: DECREASING([PCU])
RESOURCE SCHEDULER: CRESCENT([NTE])

```

(b) Dinâmico

Figura 5.10: Código dos escalonadores (a) Estático. (b) Dinâmico.

5.3.3 Resultados

Para formar o gráfico apresentado na figura 5.11 foram coletados dados a cada dez mil segundos durante a simulação, pois o tempo médio simulado era de quatrocentos e seis mil segundos e um intervalo menor de coleta poderia dificultar a compreensão do gráfico. Este gráfico apresenta a utilização dos recursos da grade pelos usuários ao longo do tempo simulado. Conforme pode ser observado, o usuário “user1” utilizou a maior parte dos recursos do sistema na primeira metade da simulação, portanto a maioria das tarefas dele foram executadas primeiro. Em seguida as tarefas do “user2” tiveram prioridade e por ultimo as tarefas do “user3” foram atendidas. Este comportamento era esperado, pois segue de acordo com o poder computacional de cada um dos usuários para cada grade, atendendo por último o usuário que não cedeu poder computacional a grade. No início há execução de tarefas de todos os usuários, em função da forma da chegada de novas tarefas ao sistema, de acordo com uma distribuição exponencial.

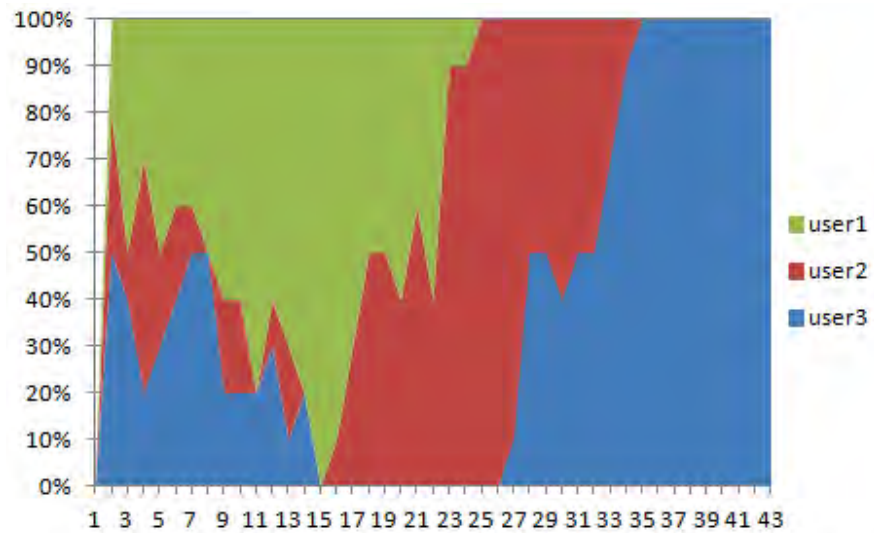


Figura 5.11: Utilização dos recursos da grade pelos usuários ao longo da simulação

A tabela 5.3 apresenta a quantidade de tarefas executadas em cada máquina do modelo simulado, contendo a quantidade de tarefas por usuário e o total atendido. Pode ser observado que o escalonador dinâmico distribuiu mais tarefas para os escravos exclusivos que o escalonador estático, na média de cinquenta tarefas a mais para cada máquina. Isto ocorreu porque o escalonador dinâmico possuía maior conhecimento sobre a carga em cada máquina. Como cada mestre submeteu três mil tarefas, o escalonador dinâmico submeteu aproximadamente 290,85 tarefas para cada máquina compartilhada, enquanto o escalonador estático submeteu 327,85.

Ambos os escalonadores enviaram um número menor de tarefas para os recursos mais sobrecarregados, sendo que o escalonador estático realizou isto por causa da restrição do número de tarefas que poderiam ser enviadas, já o escalonador dinâmico selecionava o recurso de acordo com as informações obtidas durante a execução da simulação.

Tabela 5.3: Total de tarefas executadas em cada recurso (a) Dinâmico. (b) Estático.

(a) Dinâmico

Escravos	Icon7	Icon8	Icon9	Icon10	Icon19	Icon20	Icon21
User1	122,6	121	122	121,4	172,4	174	166,6
User2	61,8	61	60	58,2	250	256,6	252,4
User3	107,2	107,4	110,6	110,2	188,6	185,4	190,6
Número total de tarefas	291,6	289,4	292,6	289,8	611	616	609,6

(b) Estático

Escravos	Icon7	Icon8	Icon9	Icon10	Icon22	Icon23	Icon24
User1	108,4	111,8	112	105,4	190,6	184,4	187,4
User2	109,6	107,8	111,2	112,4	185,8	188,2	185
User3	108,8	104,6	108,2	111,2	192,2	187	188
Número total de tarefas	326,8	324,2	331,4	329	568,6	559,6	560,4

5.4 Eficiência do simulador

O último teste avalia a eficiência do simulador do ponto de vista de velocidade, comparando-se o tempo decorrente da execução do processo de simulação em vários cenários. Para tal foram construídos três modelos de grades computacionais, e realizados diversos testes utilizando o algoritmo de escalonamento dinâmico Dynamic FPLTF (com intervalo de atualização de 1 segundo), e o estático Workqueue. Cada grade modelada é composta por um mestre e vários escravos conectados diretamente ao mestre, o número de escravos em cada modelo foi de 10, 50 e 100, visando observar o comportamento do simulador com o aumento dos recursos presentes na grade.

Optou-se por utilizar uma configuração homogênea para as máquinas e conexão de rede presentes no modelo, com a seguinte configuração:

- Máquinas
 - Capacidade de Processamento: 33000 Mflops (Intel Core i5)
 - Fator de carga: 0,05
- Rede
 - Capacidade de Comunicação: 300 Mbps
 - Fator de carga: 0,05
 - Latência 0,005 segundos

Foram modeladas quatro configurações de cargas de trabalho contendo 100, 1000, 10000 e 100000 tarefas, cada tarefa com a computação variando entre 33000 e 33000000 Mflops, e a demanda de comunicação entre 300 e 3000 Mbits. Desta forma foram realizadas várias simulações com cada uma das quatro configurações de carga de trabalho para cada modelo de grade. Para realizar os testes utilizou-se o computador “<GSPD_Tempero>” do laboratório do GSPD, que possui um processador Intel Core i5 de 3,2 GHz, e 8 GB de memória RAM.

5.4.1 Resultados

Os resultados obtidos nos testes com o escalonador estático são apresentados na tabela 5.4a, enquanto os resultados alcançados com o escalonador dinâmico aparecem na tabela 5.4b. Nestas tabelas é apresentado o tempo que o iSPD gastou para realizar a simulação de cada um dos modelos criados. Conforme é possível observar, o tempo de execução da simulação aumenta um pouco à medida que o número de máquinas aumenta, contudo este tempo é mais dependente do número de tarefas submetidas.

Para realizar uma comparação pode ser observado na tabela 5.5 os resultados referente ao tempo gasto para simular o algoritmo estático Workqueue no SimGrid. Inici-

Tabela 5.4: Tempo de execução da simulação (b) Dinâmico. (a) Estático.

(a) Escalonador Estático

Número de tarefas	10 escravos	50 escravos	100 escravos
100	0,003	0,002	0,012
1000	0,02	0,017	0,024
10000	0,998	1,004	1,193
100000	109,924	112,005	130,297

(b) Escalonador Dinâmico

Número de tarefas	10 escravos	50 escravos	100 escravos
100	0,135	0,244	0,384
1000	1,21	1,592	2,58
10000	26,491	30,733	32,202
100000	1726,594	1799,453	1861,148

almente o iSPD realiza a simulação de modelos pequenos mais rapidamente, contudo com modelos maiores o SimGrid é mais eficiente. O mais preocupante aqui é que a curva do tempo gasto para concluir a simulação no iSPD cresce muito a medida que aumenta o modelo, principalmente com algoritmos de escalonamento dinâmicos.

Tabela 5.5: Tempo de execução da simulação no SimGrid (Estático)

Número de tarefas	10 escravos	50 escravos	100 escravos
100	0,084	0,107	0,133
1000	0,369	0,439	0,495
10000	3,202	3,664	4,405
100000	28,148	31,968	40,506

A figura 5.12 apresenta os gráficos dos resultados obtidos nos modelos com 10 e 100 escravos. Conforme pode ser observado, o tempo de execução da simulação cresce exponencialmente à medida que aumenta o número de tarefas. As características presentes em diversos escalonadores dinâmicos torna o tempo de execução de sua simulação maior que o de escalonadores estáticos.

Portanto, apesar do tempo de execução ser aceitável, é notório que o desempenho possui grande queda à medida que o modelo simulado cresce tornando interessante a

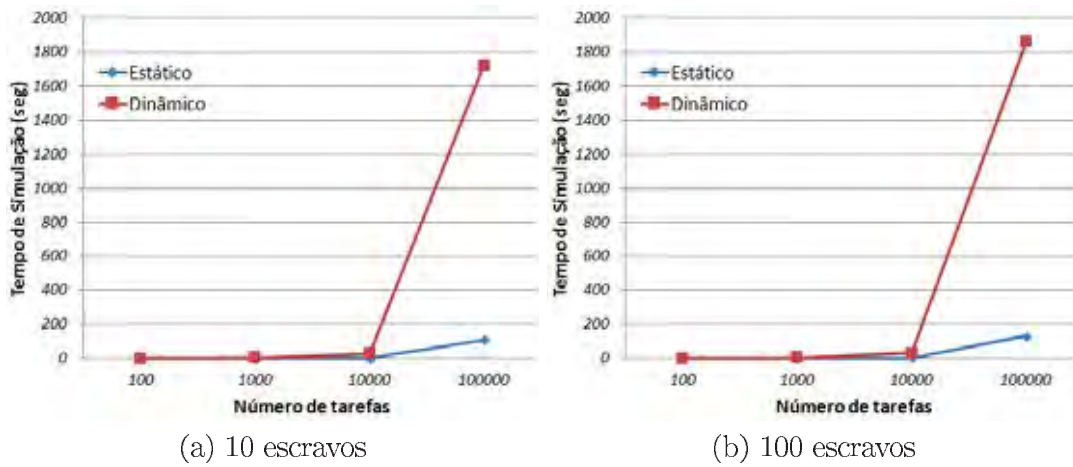


Figura 5.12: Gráfico do tempo de execução da simulação (a) 10 escravos. (b) 100 escravos.

paralelização da execução do motor de simulação, principalmente para modelos que utilizem muitas mensagens.

5.5 Considerações finais

Neste capítulo foram apresentados alguns resultados obtidos na análise do iSPD. Cada teste apresentado tem uma finalidade distinta, e conforme apresentado os resultados obtidos podem ser considerados satisfatórios, pois possuíram o comportamento previsto com a precisão esperada.

Capítulo 6

Considerações Finais

Esta dissertação apresentou a modelagem e implementação de um componente capaz de interagir de forma simples com o usuário na construção de algoritmos de escalonamento para o simulador de grades iSPD. Seu intuito é facilitar a criação de novas políticas de escalonamento para grades computacionais, tornando rápido o processo para avaliação de modelos numa área de grande interesse como é a de computação em grades. A seguir são apresentadas as principais conclusões sobre o trabalho, bem como indicados novos desdobramentos a partir do que foi realizado.

6.1 Conclusões

O iSPD busca ser uma ferramenta de simulação estável e precisa, focalizada em facilitar a interação com o usuário, através da modelagem da grade de forma simples e rápida. Esta pesquisa ampliou esta funcionalidade para a modelagem de escalonadores, definindo técnicas para realizar tanto a inserção automatizada de modelos para os escalonadores, como o seu gerenciamento na construção dos modelos para as grades.

A principal contribuição da simplificação desse processo é permitir o estudo do impacto de políticas de escalonamento utilizando o iSPD. Esse estudo é importante pois

o escalonador tem grande influência no desempenho de sistemas distribuídos. Além disso, a análise de algoritmos de escalonamento é um campo que se adapta bem à avaliação por meio de simulação.

A partir dos resultados obtidos através deste trabalho podem-se enfatizar as seguintes características:

- A separação do escalonador da estrutura de servidores da rede de filas desenvolvida no motor de simulação implica em diversas vantagens, como simplificar a alteração de um componente sem a necessidade de modificar o outro, permitindo desenvolver novos escalonadores para uso nos servidores de filas já implementados.
- A reformulação da rede de filas distribuiu o controle dos servidores, cada centro de serviço possui gerenciamento próprio, o que possibilita atribuir um escalonador diferente para cada servidor.
- Os testes feitos com intuito de verificar a precisão do simulador obtiveram resultados bem próximos dos apresentados no ambiente real e em outro simulador, confirmando que o iSPD além de ser é uma ferramenta fácil de usar também é precisa.
- Os testes utilizados para validar o gerador de escalonadores comprovam a importância deste componente, pois através de um processo simples é possível desenvolver políticas de escalonamento com o mesmo comportamento alcançado com a codificação direta da classe Java do escalonador.
- A geração de algoritmos de escalonamento de forma automática permite ao usuário do simulador especificar e avaliar uma política de escalonamento sem a necessidade de aprender uma nova linguagem de programação, ou a forma que o simulador realiza esta operação.

6.2 Direções futuras

Atualmente estão em andamento algumas pesquisas dentro do GSPD para fazer melhorias no iSPD. Os trabalhos possuem os seguintes objetivos:

- Permitir a importação e exportação de modelos criados para outros simuladores, sendo que já existe a importação de modelos criados para o SimGrid. O trabalho atual se concentra na conversão de modelos do GridSim para o iSPD, e do iSPD para o GridSim.
- Estender o tratamento de cargas de trabalho, melhorando o modelo de definição e permitindo utilizar arquivos de traços padronizados e bancos de cargas.
- Aprimoramento das métricas de desempenho e formas de apresentação de resultados ao concluir a simulação, com adição de gráficos e organização das métricas em várias categorias.

Além dos trabalhos em andamento, o iSPD ainda permite diversas novas frente de trabalhos, como:

- Ampliação do tipo de aplicação simulada sobre a grade, atualmente limitado na execução de aplicações BoT no modelo mestre-escravo, pode-se permitir a simulação de tarefas com comunicação e dependências semelhante ao MPI.
- Viabilizar o estudo de grades de dados, com a definição de arquivos distribuídos seria possível a análise de técnicas de replicação de dados e simulação de sistemas de arquivos distribuídos.
- Permitir a análise da computação em nuvem, por meio da definição de grades orientadas a serviços.
- Desenvolver os conceitos necessários para criação de organizações virtuais, permitindo estudar o comportamento de seus usuários na grade, criar políticas para

limitar acesso aos recursos, e adicionar novas informação para usar nas políticas de escalonamento.

- Paralelização do motor de simulação, conforme apresentado nos testes é necessário melhorar o desempenho na simulação de sistemas grades (contendo diversos recursos, tarefas e troca de mensagens). Uma forma de melhorar o desempenho nestes casos seria através da execução paralela da simulação.
- Melhorias no gerador, que deve receber atualizações constantes, à medida que o motor de simulação recebe atualizações, visando acompanhar todas as funcionalidades adicionadas e ampliar os tipos de escalonadores que ele possa modelar.

6.3 Publicações

Com a finalidade de expor os resultados parciais obtidos com o desenvolvimento deste trabalho, foram publicados e apresentados artigos nos seguintes eventos:

- *Interpretador de algoritmos de escalonamento para inserção de escalonadores em simulador de computação em grade.* Apresentado na II Escola Regional de Alto Desempenho de São Paulo (ERAD-SP 2011) (Menezes et al., 2011).
- *Interpretador de algoritmos de escalonamento para inserção de escalonadores em simulador de computação em grade.* Apresentado no I Workshop do Programa de Pós-Graduação em Ciência da Computação da UNESP (WPPGCC 2011) (Menezes and Manacero, 2011).
- *iSPD: an iconic-based modeling simulator for distributed grids.* Apresentado no Annual Simulation Symposium (ANSS) (Manacero et al., 2012)

Referências Bibliográficas

Abbas, A. *GRID COMPUTING: A Practical Guide to Technology and Applications*. Charles River Media, Inc., Rockland, MA, USA, 2003.

Abdurrab, A. R. and Xie, T. Fire: A file reunion based data replication strategy for data grids. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 215–223, Washington, DC, USA. IEEE Computer Society, 2010.

Albodour, R., James, A., and Yaacob, N. An extension of gridsim for quality of service. In *Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on*, pages 361 –366, 2010.

Anglano, C., Canonico, M., Guazzone, M., Botta, M., Rabellino, S., Arena, S., and Girardi, G. Peer-to-peer desktop grids in the real world: The sharegrid project. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 609 –614, 2008.

Aoqui, V., Guerra, A. I., Garcia, M. A. B. A., Oliveira, P. H. M. A., Lobato, R. S., and JR, A. M. Interpretador de modelos externos para simulador de grades computacionais. In *I Escola Regional de Alto Desempenho de São Paulo*, volume CD-ROM, pages 1–2, São Carlos. Universidade Presbiteriana Mackenzie, 2010.

Ashraf, J. and Erlebach, T. A hybrid scheduling technique for grid workflows in advance

- reservation environments. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 98 –106, 2011.
- Banks, J., Carson, J. S., Nicol, D. M., and L., N. B. *Discrete-Event System Simulation*. Prentice-Hall, 3rd edition edition, 2001.
- Bell, W., Cameron, D., Capozza, L., Millar, A., Stockinger, K., and Zini, F. Simulation of dynamic grid replication strategies in optorsim. In *Proc. of the ACM/IEEE Workshop on Grid Computing*. Springer-Verlag, 2002.
- Bell, W. H., Cameron, D. G., Capozza, L., Millar, A. P., Stockinger, K., and Zini, F. Optorsim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, 2003.
- Benoit, A., Marchal, L., Pineau, J.-F., Robert, Y., and Vivien, F. Scheduling concurrent bag-of-tasks applications on heterogeneous platforms. *IEEE Trans. Comput.*, 59:202–217, 2010.
- Branco, K. R. L. J. C. *Índices de Carga e Desempenho em Ambientes Paralelos / Distribuídos - Modelagem e Métricas*. PhD thesis, Universidade de São Paulo (USP), São Carlos, 2004.
- Buyya, R. and Murshed, M. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Pract. and Exper.*, 14:1175–1220, 2002.
- Caminero, A., Carrión, C., and Caminero, B. On the improvement of the network qos in a grid environment. In *Proceedings of the 4th international workshop on Middleware for grid computing*, MCG '06, pages 18–, New York, NY, USA. ACM, 2006.
- Caminero, A., Sulistio, A., Caminero, B., Carrion, C., and Buyya, R. Extending gridsim with an architecture for failure detection. In *Proceedings of the 13th International*

- Conference on Parallel and Distributed Systems - Volume 01*, pages 1–8, Washington, DC, USA. IEEE Computer Society, 2007.
- Cancio, G., Steve, C., Folkes, T., Fisher, S. M., Hoschek, W., Kelsey, D., Folkes, R. T., Giacomini, F., Hoschek, I. W., Kelsey, C. D., and Tierney, B. L. The datagrid architecture version 2. Technical report, CERN, 2001.
- Casanova, H. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, pages 430–437, 2001.
- Casanova, H., Legrand, A., and Quinson, M. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, 2008.
- Clauss, P.-N., Stillwell, M., Genaud, S., Suter, F., Casanova, H., and Quinson, M. Single Node On-Line Simulation of MPI Applications with SMPI. In *International Parallel & Distributed Processing Symposium*, Anchorage (AK), États-Unis. IEEE, 2011.
- Cray. Cray history. Disponível em <<http://www.cray.com/About/History.aspx>>. Acesso em: 1 mar. 2012.
- Dumitrescu, C., Dumitrescu, C., Epema, D. H. J., Epema, D. H. J., Dünneberger, J., Dünneberger, J., Gorlatch, S., Gorlatch, S., and Tr, C. User transparent scheduling of structured parallel applications in grid environments. *HPC-GECO/CompFrame Workshop*, 2006.
- Dumitrescu, C., Iosup, A., Sonmez, O. O., Mohamed, H. H., and Epema, D. H. J. Virtual domain sharing in e-science based on usage service level agreements. In *CoreGRID'07*, pages 15–25, 2007.

- Dumitrescu, C. L. and Foster, I. Gangsim: a simulator for grid scheduling studies. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2 - Volume 02*, CCGRID '05, pages 1151–1158, Washington, DC, USA. IEEE Computer Society, 2005.
- Dumitrescu, C. L., Wilde, M., and Foster, I. A model for usage policy-based resource allocation in grids. In *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 191–200, Washington, DC, USA. IEEE Computer Society, 2005.
- Ernemann, C., Hamscher, V., and Yahyapour, R. Economic scheduling in grid computing. In *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '02, pages 128–152, London, UK, UK. Springer-Verlag, 2002.
- Falavinha, J., Manacero, A., Livny, M., and Bradley, D. The owner share scheduler for a distributed system. In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, pages 298 –305, 2009.
- Falavinha Jr., J., Manacero Jr., A., Boccardo, D., and de Oliveira, L. Avaliação de algoritmos de escalonamento em grids para diferentes configurações de ambiente. In *Anais do XXVII Congresso Anual da SBC - Wperformance'2007*, pages 505–524, 2007.
- FAPESP Inpe recebe supercomputador climático. Disponível em <<http://agencia.fapesp.br/12874>>. 2010.
- Foster, I., Geisler, J., Nickless, B., Smith, W., and Tuecke, S. Software infrastructure for the i-way high-performance distributed computing experiment. In *High Performance Distributed Computing, 1996., Proceedings of 5th IEEE International Symposium on*, pages 562 –571, 1996.

- Foster, I. and Kesselman, C. *The Grid 2: blueprint for a new computing infrastructure*. Morgan Kaufmann, 2003.
- Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of Supercomputer Applications*, 15(3):200–222, 2001.
- Ganglia. Ganglia monitoring system. Disponível em <<http://ganglia.sourceforge.net/>>. Acesso em: 1 mar. 2012.
- GangSim. Gangsim: A simulator for grid scheduling studies with support for uslas. Disponível em <<http://people.cs.uchicago.edu/~cldumitr/GangSim/>>. Acesso em: 1 mar. 2012.
- Gao, R. and Zhou, H. Research on scheduling strategy in parallel applications based on a hybrid genetic algorithm. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1–4, 2008.
- GridSim. Gridsim: A grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing. Disponível em <<http://www.buyya.com/GridSim/>>. Acesso em: 1 mar. 2012.
- GSPD. Gspd's homepage. Disponível em <<http://www.dcce.ibilce.unesp.br/spd/>>. Acesso em: 1 mar. 2012.
- Guerra, A. I., Garcia, M. A. B. A., Oliveira, P. H. M. A., Aoqui, V., Lobato, R. S., and JR, A. M. Plataforma de simulação de grades computacionais: Interface icônica. In *I Escola Regional de Alto Desempenho de São Paulo*, volume CD-ROM, pages 1–2. Universidade Presbiteriana Mackenzie, 2010.
- Hunold, S., Hoffmann, R., and Suter, F. Jecure: A tool for visualizing schedules of parallel applications. In *Proceedings of the 2010 39th International Conference on*

- Parallel Processing Workshops*, ICPPW '10, pages 169–178, Washington, DC, USA. IEEE Computer Society, 2010.
- Intel. Intel® xeon® processor - microprocessor export compliance metrics. Disponível em <<http://www.intel.com/support/processors/xeon/sb/CS-020863.htm>>. Acesso em: 1 mar. 2012.
- Iosup, A., Li, H., Dumitrescu, C., Wolters, L., and Epema, D. H. J. The grid workload format. Technical report, 2006.
- Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 2nd edition edition, 1991.
- Jones, W. M., Daly, J. T., and DeBardleben, N. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 276–279, New York, NY, USA. ACM, 2010.
- Jones, W. M., Ligon, III, W. B., Pang, L. W., and Stanzione, D. Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. *J. Supercomput.*, 34(2):135–163, 2005.
- Klusáček, D., Matyska, L., and Rudová, H. Alea: grid scheduling simulation environment. In *Proceedings of the 7th international conference on Parallel processing and applied mathematics*, PPAM'07, pages 1029–1038, Berlin, Heidelberg. Springer-Verlag, 2008.
- Kumar, S., Kumar, N., and Kumar, P. Genetic algorithm for network-aware job scheduling in grid environment. In *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE*, pages 615 –620, 2011.

- Kurowski, K., Nabrzyski, J., Oleksiak, A., and Weglarz, J. Grid scheduling simulations with gssim. In *International Conference on Parallel and Distributed Systems*, pages 1–8, 2007.
- Litzkow, M., Livny, M., and Mutka, M. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111, 1988.
- Manacero, A., Lobato, R., Guerra, A., Garcia, M., Oliveira, P., Aoqui, V., Menezes, D., and Silva, D. ispd: an iconic-based modeling simulator for distributed grids. In *Annals of 45th Annual Simulation Symposium*, volume CDROM of *ANSS12*, pages 1–8, Orlando, USA, 2012.
- Menascé, D. A., Saha, D., Porto, S. C. d. S., Almeida, V. A. F., and Tripathi, S. K. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *J. Parallel Distrib. Comput.*, 28(1):1–18, 1995.
- Menezes, D. and Manacero, A. Interpretador de algoritmos de escalonamento para inserção de escalonadores em simulador de computação em grade. In *I Workshop do Programa de Pós-Graduação em Ciência da Computação da UNESP*, volume CD-ROM, pages 1–2, Bauru, 2011.
- Menezes, D., Silva, D. T., and Manacero, A. Interpretador de algoritmos de escalonamento para inserção de escalonadores em simulador de computação em grade. In *II Escola Regional de Alto Desempenho de São Paulo*, volume CD-ROM, pages 1–4, São José dos Campos, 2011.
- Merz, S., Quinson, M., and Rosa, C. Simgrid mc: verification support for a multi-api simulation platform. In *Proceedings of the joint 13th IFIP WG 6.1 and 30th IFIP WG 6.1 international conference on Formal techniques for distributed systems, FMOODS’11/FORTE’11*, pages 274–288, Berlin, Heidelberg. Springer-Verlag, 2011.

- Netperf. Netperf homepage. Disponível em <<http://www.netperf.org/netperf/>>. Acesso em: 1 mar. 2012.
- Nukarapu, D., Tang, B., Wang, L., and Lu, S. Data replication in data intensive scientific applications with performance guarantee. *Parallel and Distributed Systems, IEEE Transactions on*, 22(8):1299–1306, 2011.
- Oliveira, P. H. M. A., Guerra, A. I., Garcia, M. A. B. A., Aoqui, V., JR, A. M., and Lobato, R. S. O motor de uma plataforma de simulação de grades computacionais. In *I Escola Regional de Alto Desempenho de São Paulo*, volume CD-ROM, pages 1–2. Universidade Presbiteriana Mackenzie, 2010.
- OptorSim. Simulating data access optimization algorithms - optorsim. Disponível em <<http://optorsim.sourceforge.net/>>. Acesso em: 1 mar. 2012.
- Paula, N. C. *Um ambiente de monitoramento de recursos e escalonamento cooperativo de aplicações paralelas em Grades Computacionais*. PhD thesis, Universidade de São Paulo (USP), São Paulo, 2009.
- Quinson, M. Gras: A research and development framework for grid and p2p infrastructures. In *In Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'06)*, Dallas, TX, 2006.
- Shorfuzzaman, M., Graham, P., and Eskicioglu, R. Distributed popularity based replica placement in data grid environments. In *Proceedings of the 2010 International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT '10*, pages 66–77, Washington, DC, USA. IEEE Computer Society, 2010.
- Silva, D. P. Usando replicação para escalonar aplicações bag-of-tasks em grids computacionais. Master's thesis, Universidade Federal de Campina Grande, Campina Grande, 2003.

- SimGrid. Welcome to the simgrid project! Disponível em <http://SimGrid.gforge.inria.fr/>. Acesso em: 1 mar. 2012.
- Sulistio, A., Cibej, U., Venugopal, S., Robic, B., and Buyya, R. A toolkit for modelling and simulating data grids: an extension to gridsim. *Concurr. Comput. : Pract. Exper.*, 20:1591–1609, 2008.
- Sulistio, A., Poduval, G., Buyya, R., and Tham, C.-K. On incorporating differentiated levels of network service into gridsim. *Future Gener. Comput. Syst.*, 23(4):606–615, 2007.
- Sulistio, A., Yeo, C., and Buyya, R. Visual modeler for grid modeling and simulation (gridsim) toolkit. In *Lecture Notes on Computer Science*, pages 1123–1132, 2003.
- Takefusa, A. and Matsuoka, S. Performance issues in client-server global computing. *International Workshop on Global and Cluster Computing (WGCC'2000)*, 2000.
- Tanenbaum, A. S. and Steen, M. V. *Sistemas Distribuídos Princípios e Paradigmas*. Ed. Pearson, 2nd edition, 2007.
- Zhong, H., Zhang, Z., and Zhang, X. A dynamic replica management strategy based on data grid. In *Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing*, GCC '10, pages 18–23, Washington, DC, USA. IEEE Computer Society, 2010.
- Zhu, W., Wang, C.-L., and Lau, F. C. M. Jessica2: A distributed java virtual machine with transparent thread migration support. In *Proceedings of the IEEE International Conference on Cluster Computing*, CLUSTER '02, pages 381–, Washington, DC, USA. IEEE Computer Society, 2002.