Universidade Federal Fluminense

CAMILO CARDOSO FIGUEIRA
FELIPE KRAUS

Framework de Simulação de Escalonadores Dinâmicos Distribuídos

Niterói

Agosto de 2006

Agradecimentos

À Cristina Boeres - orientadora paciente que, com brilhantismo, conduziu-nos para o êxito deste trabalho.

À todos os colegas do Smart Grid Computing Lab, em especial à Daniela e Jacques - pessoas amigas e que, com abnegação, estiveram sempre dispostos a ajudar.

Ao Didiu e Ricão, amigos que compartilharam conosco todos os desafios nessa trajetória.

Resumo

Na busca incessante por minimizar o tempo de execução de aplicações paralelas em ambientes distribuídos, o estudo do problema do escalonamento de tarefas têm se tornado o foco de muitas pesquisas na área de Grades Computacionais. Nesse contexto, heurísticas de escalonamento têm papel fundamental para que os recursos da grade possam ser explorados de forma eficiente. Há duas abordagens básicas para as heurísticas de escalonamento, estática e dinâmica. No projeto *EasyGrid*, essas duas abordagens são unidas na proposta de escalonamento híbrido. A heurística híbrida realiza uma fase de escalonamento estático prévia à execução da aplicação e uma fase dinâmica, que avalia o comportamento do universo da grade ao longo da execução. O objetivo deste trabalho é simular uma hierarquia de escalonamento para execução de aplicações paralelas, possibilitando avaliar a implementação de um modelo hierárquico real e provendo um ambiente de análise de diversas políticas de escalonamento estático e dinâmico.

Sumário

Lista de Figuras

1 Introdução **1**

1.1 Organização . . . 3

2 Problema de Escalonamento **4**

2.1 Modelo da Aplicação . . . 5

2.1.1 Árvores Binárias Completas (*Out-tree* ou *Out*) . . . 5

2.1.2 Árvores Binárias Reversas Completas (*In-tree* ou *In*) . . . 5

2.1.3 Diamantes (Di) . . . 6

2.1.4 Randômicos (Ran) . . . 6

2.1.5 *Peer Set Graphs* (PSGs). . . . 7

2.2 Modelo da Comunicação (Latência) . . . 7

2.3 Modelo da Arquitetura . . . 8

2.4 O Portal EasyGrid . . . 8

2.4.1 Framework EasyGrid . . . 8

2.4.2 Arquitetura EasyGrid . . . 9

3 Heurísticas de Escalonamento **11**

3.1 Heurísticas Híbridas . . . 12

3.1.1 Heurísticas Estáticas . . . 12

3.1.2 Heurísticas Dinâmicas . . . 13

3.1.2.1 *Min-Min* . . . 15

4 Modelo da Hierarquia 17

4.1 Escalonador Global - EG 18

4.2 Escalonadores do Site - ES 18

4.3 Escalonadores da Máquina - EM 19

4.4 Arquitetura do Simulador 19

4.4.1 Núcleo 19

4.4.2 Implementação 23

5 Ambiente de Análise 28

5.1 Estudo de Caso 29

5.1.1 Configurações Iniciais 30

5.1.1.1 GAD 31

5.1.1.2 Arquitetura e Hierarquia 31

5.1.2 Escalonamento Estático 31

5.1.3 Escalonamento Dinâmico 32

5.2 Outras Funcionalidades 33

5.2.1 Algoritmo × Grafo 34

5.2.2 Algoritmo Par-a-Par 35

5.2.3 Estatísticas 36

6 Trabalhos Relacionados 40

7 Conclusões e Trabalhos Futuros 43

Apêndice A 45

Usando o *Framework* 45

Apêndice B 49

Exemplos de arquivos de entrada e saída 49

Formato do arquivo contendo os parâmetros de entrada 49

Formato do arquivo de entrada oriundo do escalonador estático 50

Formato do arquivo de saída do escalonamento híbrido 51

Formato do arquivo de saída do escalonador dinâmico 52

Formato do arquivo de saída da divisão dos blocos 53

Apêndice C **54**

Exemplos de GAD's e arquiteturas 54

Formato dos arquivos de entrada GAD 54

Formato do arquivo de entrada de custos do GAD 55

Formato do arquivo de entrada de arquitetura 56

Formato do arquivo de entrada da hierarquia da arquitetura 57

Apêndice D **58**

Publicação 58

Referências **59**

Lista de Figuras

2.1 Grafo Acíclico Direcionado representando uma aplicação *Out-tree* 5

2.2 Grafo Acíclico Direcionado representando uma aplicação *In-tree* 6

2.3 Grafo Acíclico Direcionado representando uma aplicação *Diamante* 6

2.4 Grafo Acíclico Direcionado representando uma aplicação *Randômica* . . . 7

2.5 Estrutura em camadas do *Framework EasyGrid* 9

2.6 Hierarquia de processos gerenciadores do EasyGrid 10

4.1 Exemplo do modelo hierárquico de escalonadores dinâmicos 17

4.2 Implementação da topologia no simulador, onde h(p_i) é o fator de heterogeneidade do processador p_i 21

4.3 Implementação do modelo de tarefas no simulador 22

5.1 Tela da entrada de dados da aplicação 28

5.2 Tela da entrada de dados da arquitetura 29

5.3 Grafo Diamante 25 30

5.4 Hierarquia representativa da arquitetura utilizada 31

5.5 Configuração inicial do escalonador estático 32

5.6 Resultado final do escalonamento estático 33

5.7 Configuração inicial do escalonador híbrido 34

5.8 Primeiro evento de escalonamento 35

5.9 Segundo evento de escalonamento 36

5.10 Terceiro evento de escalonamento 37

5.11 Resultado de escalonamento híbrido 38

5.12 Resultados *Algoritmo* $\times$ *Grafo* 38

5.13 Resultados *Algoritmo Par-a-Par* 39

5.14 Resultados estatísticos 39

6.1 Arquitetura da plataforma e dos componentes do *GridSim*. 40

6.2 Arquitetura do simulador *MicroGrid*. 41

A.1 Tela de configuração do escalonador dinâmico na ferramenta *Easygrid* . . . 45

A.2 Tela de visualização dos resultados do escalonador dinâmico na ferramenta *Easygrid* 48

Capítulo 1

Introdução

Segundo [5], um *cluster* de processadores ou computadores é um tipo de sistema de processamento paralelo ou distribuído, que consiste em uma coleção de computadores interconectados trabalhando cooperativamente juntos como se fossem um único computador. Até então, os *clusters* de computadores eram contruídos em redes locais, com computadores homogêneos e conectados através de *switchs* e geralmente funcionavam em modo dedicado, ou seja, os computadores ficavam reservados para uso exclusivo de uma determinada aplicação ou conjunto de aplicações.

Há alguns anos, esse conceito de *cluster* foi expandido. Graças a ampla disseminação de redes de alta velocidade, tornou-se possível criar malhas computacionais com recursos geograficamente distribuídos. Uma malha consiste em um conjunto de recursos computacionais heterogêneos interconectados através de uma rede, por exemplo a *internet*, para resolver um problema comum à todos esses recursos. A esse sistema deu-se o nome de Grade Computacional ou comumente chamado *Grid* [11].

Através de um *Grid* é possível prover alto poder computacional para usuários que necessitam resolver problemas de grande complexidade e que são incapazes de fazê-lo em um computador seqüencial tradicional devido a dois fatores: inviabilidade de tempo e custo de adquirir melhores ou maiores sistemas computacionais.

Ao contrário dos altos custos de aquisição dos supercomputadores, as grades computacionais podem ser constituídas de recursos heterogêneos como computadores domésticos, clusters de computadores, ou até mesmo supercomputadores já existentes, tornando-as versáteis e consequentemente, a um custo mais acessível.

Diversas áreas da pesquisa científica ou mesmo aplicações comerciais que necessitam de processamento pesado podem se beneficiar dessa abordagem. Grades têm se tornado cada vez mais importantes para aplicações em áreas como previsão do tempo, simulações

de física de alta energia, biologia molecular, dinâmica molecular e no processamento de imagens [17] [16] [1].

Para se beneficiar de todo esse potencial, é preciso dividir o problema em partes. Para isso, utiliza-se o conceito de aplicação paralela que consiste na construção de códigos paralelos, distribuídos em processos visando reduzir o tempo final de execução para solucionar um dado problema. Esses processos ou *tarefas* podem ser executadas em paralelo em processadores distintos de tal forma que o tempo de execução seja o menor possível. Entretanto, utilizar e gerenciar essa execução eficiente em um conjunto de processadores não é trivial, mais ainda, em ambientes computacionais como o de grades. Desta forma, o estudo de algoritmos de escalonamento de aplicações paralelas têm se tornado o foco de pesquisas nessa área. Diferentes algoritmos de escalonamento vêm sendo desenvolvidos na tentativa de alocar eficientemente um conjunto de tarefas a um conjunto de recursos computacionais, segundo alguma métrica. Uma métrica muito utilizada é o *makespan* que é o tempo compreendido entre o início de execução da primeira tarefa de uma aplicação até o término da última tarefa. Heurísticas de escalonamento de aplicações em ambientes de grades devem considerar duas características básicas: os recursos são heterogêneos e compartilhados.

Existem dois tipos básicos de abordagem de escalonamento: estático e dinâmico. As heurísticas estáticas realizam um escalonamento prévio à execução enquanto que o escalonamento nas heurísticas dinâmicas, se dá ao longo da execução da aplicação sendo escalonada. Há ainda um terceiro tipo de heurística: a híbrida. Ela realiza um escalonamento prévio (estático) e, ao longo da execução, avalia as decisões a serem tomadas, de acordo com a sobrecarga nos processadores ou o congestionamento dos links, visando otimizar o escalonamento como um todo.

O escalonamento dinâmico pode ser realizado de forma centralizada. Com isso, o escalonador deve gerenciar o andamento do escalonamento, observando a execução das tarefas de todas as máquinas designadas ao escalonamento. No entanto, esta centralização pode significar um gargalo na execução da aplicação considerando principalmente ambientes de larga escala. O gargalo se deve principalmente à necessidade do escalonador ter que coletar informações sobre a aplicação e configuração atualizada do ambiente de execução.

A hierarquia e distribuição do escalonador dinâmico proposta pelo *Framework EasyGrid* [4], tende a otimizar o escalonamento dinâmico, pois o gerenciamento do escalonamento é feito em níveis, com procedimentos de escalonamento definidos para cada nível. Assim, o escalonador designa alguns processadores para executarem gerenciadores asso-

ciados à cada nível, que reportarão informações ao escalonador ou gerenciador global.

Este trabalho contribuiu para o Portal *EasyGrid* [2, 4] através da implementação de um simulador da hierarquia de escalonamento dinâmico para aplicações paralelas e distribuídas, de acordo com o *Framework EasyGrid* [2, 4]. A simulação possibilita avaliar a implementação de um modelo hierárquico real, proposto em [4] e validado em [22], provendo um ambiente de análise de diversas políticas de escalonamento.

Em [26], o simulador de escalonamento dinâmico era centralizado. Para simular a realidade do ambiente *EasyGrid*, este trabalho aqui proposto vêm de encontro com a necessidade de adicionar ao ambiente de simulação a característica hieráquica.

Tal estudo é de vital importância visto que a simulação é o melhor meio de se comparar eficientemente políticas de escalonamento, pois através de uma simulação é possível realizar experimentos com uma vasta variedade de configurações dos recursos. Além disso, devido as variações na rede e nas taxas de ocupação dos processadores, torna-se difícil obter resultados repetidos ao longo de execuções reais para um mesmo experimento.

1.1 Organização

Esta monografia de projeto final de curso está organizado da seguinte forma: no Capítulo 1, foi apresentado uma breve descrição sobre grades computacionais, aplicações paralelas e o objetivo deste trabalho. No Capítulo 2, será descrito o problema de escalonamento que define o modelo de aplicação, comunicação e arquitetura utilizados no simulador. No Capítulo 3, serão descritas as heurísticas de escalonamento adotadas pelo *framework* de simulação. No Capítulo 4, há um detalhamento do modo como foi implementado o *framework* como um todo, esmiuçando as principais funções além de seus respectivos pseudo-códigos. No Capítulo 5, serão apresentados os experimentos computacionais assim como os resultados obtidos. No Capítulo 6, há um paralelo com trabalhos semelhantes de outros projetos e no Capítulo 7, a conclusão e trabalhos futuros. No Apêndice A, há uma descrição de como utilizar o *framework* na ferramenta de simulação, explicando todas as opções de parâmetros. No Apêndice B, serão mostrados exemplos dos formatos dos arquivos de entrada e saída utilizados pelo simulador. No Apêndice C, exemplos dos formatos de arquivos GAD's (Grafo Acíclico Direcionado) e da arquitetura. No Apêndice D, há uma referência ao artigo publicado no evento WSCAD 2005, fruto deste trabalho.

Capítulo 2

Problema de Escalonamento

A motivação principal de se desenvolver aplicações paralelas é o de conseguir obter resultados mais rapidamente em relação a respectiva aplicação sequencial tradicional e com a mesma confiabilidade. Em segunda instância, poderia-se utilizá-las com o intuito de resolver problemas mais complexos no que diz respeito às dimensões da entrada de dados. Porém, conseguir desenvolver tais aplicações de maneira eficiente e executá-las em um sistema de processadores paralelos e/ou distribuídos de tal forma que o alto desempenho seja atingido não é uma tarefa fácil.

Este problema de escalonamento de tarefas de uma aplicação em um conjunto de processadores considerando a comunicação entre eles é, em sua forma geral, *NP-Completo* [27]. Por isso, é necessário utilizar boas heurísticas de escalonamento de forma a obter bons resultados em um tempo polinomial. Para isso, é necessário modelar as características da aplicação assim como da arquitetura, para que essas peculiaridades sejam consideradas no momento do escalonamento. Esses modelos podem ser simples e teóricos à complexos e realistas, sabendo-se que quanto mais real, mais difícil e complexo será o modelo.

O objetivo das heurísticas de escalonamento adotadas nesse trabalho é o de alocar as tarefas da aplicação nos processadores de um sistema de memória distribuída, de modo a minimizar o tempo de execução da aplicação, levando-se em consideração as restrições impostas pelo sistema alvo, tais como a largura de banda nos canais de comunicação e a carga em cada processador, e, se possível, utilizando um número pequeno de processadores.

Na sequência, serão descritos o modelo de aplicação, o modelo da comunicação e o modelo da arquitetura utilizados nesse trabalho.

2.1 Modelo da Aplicação

As aplicações deste trabalho são representadas por um grafo acíclico direcionado (GAD). Um GAD é denotado por $G = (V, E, \varepsilon, \omega)$, onde $V = \{v_0, v_1, ..., v_{n-1}\}$ é o conjunto de vértices que representam as tarefas ou processos da aplicação e $E = \{(v_i, v_j)|v_i, v_j \in V\}$ é o conjunto de arcos que representam as relações de precedência entre as tarefas. Denota-se por $\varepsilon(v_i)$, o peso de execução de uma tarefa $v_i \in V$ e por $\omega(v_i, v_j)$, o peso de dados associado ao arco $(v_i, v_j) \in E$, ou seja, a quantidade de dados transmitida de v_i para v_j. O conjunto de predecessores imediatos de v_i é definido como $pred(v_i) = \{v_j|(v_j, v_i) \in E\}$ e o conjunto dos sucessores imediatos, como $succ(v_i) = \{v_j|(v_i, v_j) \in E\}$.

A seguir, serão descritas as classes de aplicações que foram disponibilizadas na *Ferramenta de Simulação EasyGrid* [9].

2.1.1 Árvores Binárias Completas (*Out-tree* ou *Out*)

São definidas como grafos, onde cada vértice tem apenas um predecessor imediato, com exceção do vértice origem (raiz), e ainda cada vértice possui dois sucessores imediatos, com exceção dos vértices terminais (folhas). Algoritmos de difusão (*broadcast*) e divisão e conquista, onde os dados migram da raiz até as folhas, são bons exemplos destes grafos.

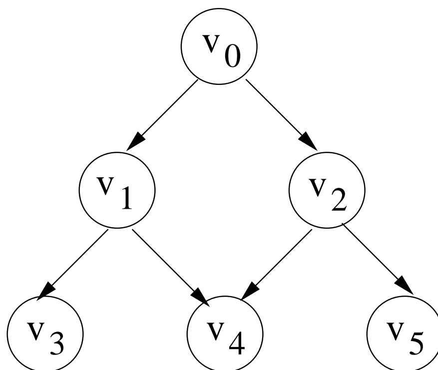

Figura 2.1: Grafo Acíclico Direcionado representando uma aplicação *Out-tree*

2.1.2 Árvores Binárias Reversas Completas (*In-tree* ou *In*)

Nestes tipos de árvores, cada vértice tem dois predecessores imediatos, com exceção dos vértices origens, e apenas um sucessor, com exceção do vértice terminal. Os algoritmos em que os dados partem das folhas em direção à raiz havendo uma redução nas operações, como na soma em paralelo, são exemplos destes grafos.

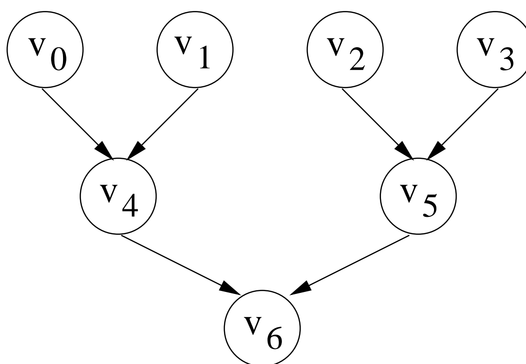

Figura 2.2: Grafo Acíclico Direcionado representando uma aplicação *In-tree*

2.1.3 Diamantes (Di)

Nos grafos diamantes, todas as tarefas são relacionadas por predecessores e sucessores comuns. O grafo diamante é um bom representante de aplicações como decomposição LU e multiplicação de matrizes.

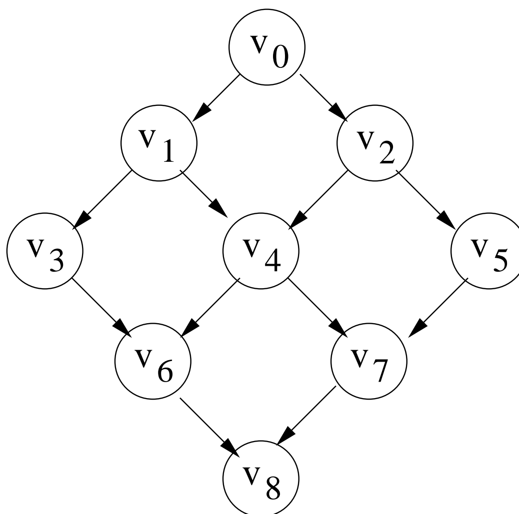

Figura 2.3: Grafo Acíclico Direcionado representando uma aplicação *Diamante*

2.1.4 Randômicos (Ran)

Estes grafos são gerados aleatoriamente e têm como objetivo representar uma aplicação paralela qualquer para que se possa avaliar o desempenho de uma heurística de escalonamento sobre grafos com estruturas irregulares.

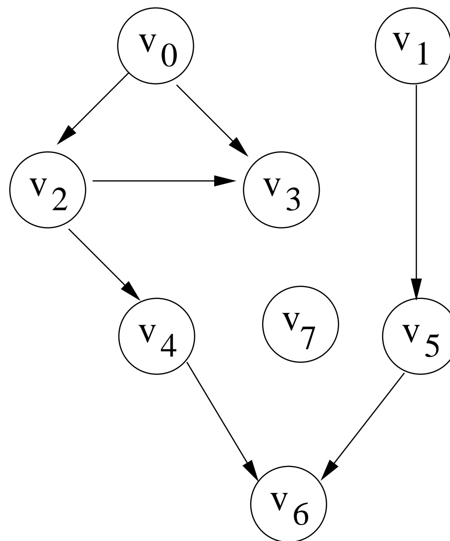

Figura 2.4: Grafo Acíclico Direcionado representando uma aplicação *Randômica*

2.1.5 *Peer Set Graphs* (PSGs).

Os PSGs são exemplos de grafos documentados na literatura [14], e muito utilizados por pesquisadores. Uma das vantagens deste conjunto de grafos é seu número reduzido de tarefas, o que permite trilhar o comportamento do algoritmo, além de acompanhar as variações topológicas e de granularidade (relação entre o grau de computação e comunicação do grafo).

2.2 Modelo da Comunicação (Latência)

O modelo de comunicação Latência [20] é o modelo padrão de comunicação utilizado atualmente. A latência é definida como sendo o custo da comunicação entre dois processadores, ou seja, o atraso no tempo de transmissão para enviar uma unidade de dado no canal de comunicação entre dois processadores, e é denotada por L. Esse modelo não considera o tempo gasto no preparo para o envio ou o recebimento de informações, assumido-os como instantâneo. Assim, esses processadores se tornam *multicast*, isto é, podem enviar e receber mensagens distintas simultaneamente. No caso da *Ferramenta de Simulação EasyGrid* [9], a sobrecarga de envio e recebimento de informação é negligível.

2.3 Modelo da Arquitetura

Define-se como arquitetura $P = \{p_0, p_1, ..., p_{q-1}\}$ o conjunto de q processadores, sendo $h(p_i)$ o fator de heterogeneidade do processador p_i. Assim, o tempo de execução de uma tarefa v_j em um processador p_i é dada por $\varepsilon(v_j) \times h(p_i)$. Para duas tarefas u e v adjacentes que foram alocadas em processadores distintos p_u e p_v respectivamente, o custo de comunicação entre u e v será $\omega(u, v) \times L$, onde a latência L é o fator multiplicativo de comunicação em qualquer canal da rede de processadores. Esse atraso corresponde ao tempo de transmissão por *byte* em um canal de comunicação (unidade de dado aqui adotada).

2.4 O Portal EasyGrid

O Portal *EasyGrid* [2, 4] tem por objetivo transformar aplicações MPI [21] em aplicações *system-aware*, ou seja, aplicações cientes do comportamento do ambiente onde estão sendo executadas, através do *Framework EasyGrid*. O *Framework EasyGrid* deve escolher a política de escalonamento e de tolerância a falhas adequada para a aplicação, determinar a alocação inicial de tarefas, compilar a aplicação, criar o ambiente MPI e garantir uma execução segura e eficiente da aplicação através de mecanismos de escalonamento e de tolerância a falhas.

2.4.1 Framework EasyGrid

O *Framework EasyGrid* é um sistema gerenciador de aplicações MPI, que trabalha com a criação dinâmica de tarefas, usando a biblioteca MPI/LAM. O *framework* utiliza uma hierarquia de processos gerenciadores, onde cada processo gerenciador possui um conjunto de funcionalidades que são essenciais para a execução eficiente de processos MPI na grade computacional. A Figura 2.5 apresenta a estrutura em camadas de cada processo do *Framework EasyGrid*.

A *camada de Abstração MPI* está presente somente nos processos da aplicação, e tem por objetivo mascarar as funções da biblioteca MPI para tornar possível a comunicação com os processos gerenciadores. Essa comunicação é realizada através de mensagens enviadas à camada de *Monitoramento da Aplicação*. A camada de *Monitoramento* por sua vez fornece dados de monitoramento às camadas de *Escalonamento Dinâmico* e de *Tolerâcia a Falhas*. A camada de *Tolerâcia a Falhas* informa à camada de *Escalonamento*

Dinâmico caso exista a necessidade de recriação de algum processo. E, por fim, a camada de *Escalonamento Dinâmico* controla a camada de *Gerenciamento de Processos*, que é responsável por criar e comunicar com os processos da aplicação.

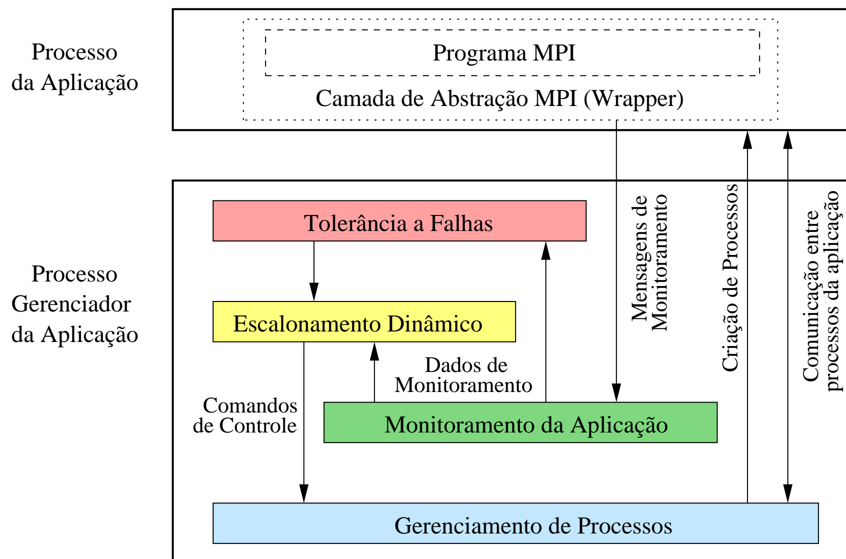

Figura 2.5: Estrutura em camadas do *Framework EasyGrid*

2.4.2 Arquitetura EasyGrid

Processos gerenciadores oferecem serviços de acordo com a sua posição na hierarquia *EasyGrid*. A Figura 2.6 mostra um exemplo de grade computacional com três *sites*, onde é possível observar os três níveis da hierarquia de gerenciadores, onde:

- EG (*Escalonador Global*): é o gerenciador global que fica no topo da hierarquia supervisionando os sites da grade onde a aplicação do usuário pode executar;
- ES (*Escalonador do Site*): é um gerenciador existente em cada um dos *sites*, e é responsável pela alocação de tarefas da aplicação no seu respectivo *site*;
- EM (*Escalonador da Máquina*): é um gerenciador presente em cada um dos recursos existentes na grade, e é responsável pelo escalonamento, criação e execução dos processos do usuário.

Neste Capítulo foi apresentado o problema de escalonamento de tarefas e os modelos de computação utilizados. Ainda neste Capítulo, o Portal *EasyGrid* foi brevemente descrito. No próximo Capítulo serão apresentadas heurísticas de escalonamento de tarefas.

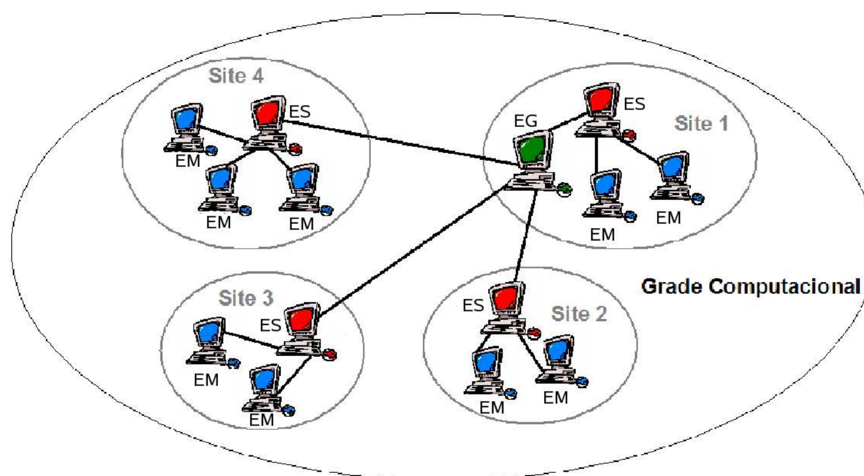

Figura 2.6: Hierarquia de processos gerenciadores do EasyGrid

Capítulo 3

Heurísticas de Escalonamento

Podemos classificar as heurísticas de escalonamento em duas abordagens básicas: estáticas e dinâmicas. A diferença entre elas se dá no momento em que as decisões são tomadas. Na heurística estática, todas as decisões são tomadas antes de começar a execução das tarefas, ou seja, todas as tarefas são escalonadas antes que a primeira comece a ser executada. Na heurística dinâmica, as decisões são tomadas ao longo da execução através de eventos de escalonamento. Em um *evento de escalonamento*, o escalonador dinâmico é acionado para avaliar o andamento das execuções das tarefas, distribuindo tarefas ainda não escalonadas. Se o escalonador detectar alguma anormalidade como sobrecarga em um dos processadores ou congestionamento em algum dos links, ele poderá reescalonar tarefas ainda não executadas para processadores subcarregados.

Na *ferramenta de simulação EasyGrid* [9], as vantagens de cada uma das abordagens é unida na proposta de escalonamento híbrido: tanto uma fase de escalonamento estático é executada, e no momento da execução da aplicação de acordo com o escalonamento já definido, o escalonador dinâmico pode re-escalonar tarefas ainda a serem executadas.

No *framework EasyGrid* adotou-se o modelo de não preempção, ou seja, a partir do momento em que uma determinada tarefa inicie sua execução, ela não é interrompida até que termine. A migração de tarefas vêm sendo estudada atualmente para ser implementada no futuro, sendo integrada ao *framework* existente. A migração pode muito útil em casos de escalonamentos com tarefas grandes, pois caso um processador seja sobrecarregado após o início da execução de uma tarefa, poderá realizar a migração desta tarefa para um outro processador ocioso. Até o momento, não se têm informações para avaliar se a migração obterá um ganho para um escalonamento real visto que será necessário realizar o salvamento do contexto de execução das tarefas assim como processos para realizar o gerenciamento da migração. Esse tipo de análise está fora do escopo deste trabalho.

3.1 Heurísticas Híbridas

Os escalonadores estáticos e dinâmicos se diferem principalmente no momento da tomada de decisão. Um escalonador híbrido é composto por essas duas etapas. Na primeira etapa, um escalonamento estático é realizado escalonando todas as tarefas da aplicação previamente à execução. Dado esse escalonamento inicial, o escalonamento dinâmico é feito ao longo da execução da aplicação.

No escalonamento dinâmico, as decisões são realizadas através de eventos de escalonamento. Em um *evento de escalonamento*, o escalonador dinâmico é acionado para avaliar o andamento das execuções das tarefas, distribuindo aos processadores tarefas ainda não escalonadas. O escalonador pode reescalonar tarefas ainda não executadas se detectar alguma anormalidade como sobrecarga em um dos processadores ou congestionamento em algum dos links.

Na definição de heurística híbrida, pode-se utilizar ou não o escalonamento estático como um pré-escalonamento à execução da aplicação. No caso de não utilizá-lo, o escalonador híbrido se comportaria como um escalonador dinâmico no início da execução, porém ele utilizaria as informações do escalonamento estático ao longo de toda a execução.

3.1.1 Heurísticas Estáticas

Existem diversos tipos de heurísticas estáticas, sendo a maioria delas derivadas da classe de heurísticas *List-Scheduling*, pois esta classe fornece um bom escalonamento de tarefas com uma baixa complexidade algorítmica. Essas heurísticas já se encontram disponíveis para utilização na ferramenta de simulação [9] do Portal *EasyGrid* [2, 4]. Dentre elas destacam-se:

- o List-Scheduling Configurável: a idéia básica do List-Scheduling tradicional é o de ordenar as tarefas de uma aplicação paralela em uma lista, de acordo com alguma prioridade pré-estabelecida e, posteriormente, associar cada tarefa a um processador, de forma a minimizar o tempo de início das tarefas. A cada iteração, a tarefa livre de maior prioridade, seguindo a ordenação da lista, é associada a um processador ocioso tal que o seu tempo de início seja o mais cedo possível. No List-Scheduling Configurável, há três níveis de prioridades para a escolha das tarefas, onde para cada nível é definido uma política de prioridade. Quando ocorre um empate de tarefas na escolha da tarefa de maior prioridade utilizando-se a política de nível primário,

empregase a política secundária e, se necessáio, a terciária para resolver qual terá a precedência de execução. As políticas de prioridades existentes atualmente na ferramenta *EasyGrid* são: nível, co-nível, nível + co-nível, ALAP, número de sucessores, LBNível, largura, ALAP menos maior aresta incidente, nível dinâmico, co-nível dinâmico e ALAP dinâmico. Seja $V = \{v_0, v_1, ..., v_{n-1}\}$, o conjunto de vértices que representam as tarefas e $proc(v_i)$, o processador onde uma tarefa v_i foi escalonada. Considera-se tarefa livre para ser escalonada toda tarefa $v_i \in V$ onde o $proc(v_i) = \varnothing$ (ou seja, v_i ainda não foi escalonada), onde para todo $v_j \in pred(v_i)$, $proc(v_j) \neq \varnothing$ (ou seja, todos os predecessores de v_i já foram escalonados).

- o DCP(*Dynamic Critical Path*) define um caminho crítico de escalonamento de tarefas de modo que as demais tarefas que ainda serão escalonadas não aumentem o tempo de fim definida pelo caminho crítico. Essa heurística utiliza a estratégia de *look-ahead* [15] para decidir a próxima tarefa a ser escalonada. [15];

- o ETF(*Earliest Task First*) considera o modelo de latência [20] e funciona da seguinte forma: a cada escalonamento de uma nova tarefa, o ETF calcula o tempo de início mais cedo para todas as tarefas livres em todos os processadores ociosos naquele momento e é escolhido o par tarefa-processador que possibilite o menor tempo de início para uma tarefa livre. Considera-se tarefa livre uma tarefa onde todas as suas predecessoras já foram escalonadas. [13];

- o HEFT(*Heterogeneous Earliest Finish Time*) possui uma política de inserção de tarefas nos tempos ociosos dos processadores proporcionando a minimização dos processadores utilizados e o uso mais eficiente dos mesmos. [25];

3.1.2 Heurísticas Dinâmicas

As heurísticas dinâmicas têm por objetivo escalonar aplicações durante suas respectivas execuções. Para isso, o funcionamento dessas heurísticas deve ser simples para não comprometer o desempenho da execução uma vez que são chamadas em tempo de execução, e oferecendo uma resposta mais imediata ao problema de alocação das tarefas aos processadores. As tarefas da aplicação são divididas em subconjuntos de tarefas. Durante a execução da aplicação, de tempos em tempos, um dos subconjuntos terá as suas tarefas associadas aos processadores, de acordo com algum critério. A frequência com a qual a heurística irá executar dependerá do *evento de escalonamento* [19] definido.

Nesse trabalho, a implementação do escalonamento dinâmico foi feita da seguinte

forma: o GAD (Grafo Acíclico Direcionado) da aplicação será dividido em blocos de tarefas que estejam em um mesmo nível no GAD, através de *eventos de escalonamento*. Um *evento de escalonamento* ocorrerá quando:

- a primeira tarefa do bloco corrente for executada;
- a última tarefa do bloco corrente for executada;
- todas as tarefas do bloco corrente forem executadas.

Nesse trabalho considerou-se também que o tempo gasto no processamento da heurística, a cada *evento de escalonamento*, é instantâneo.

Critérios distintos são utilizados para a escolha da tarefa da aplicação a ser escalonada e a qual processador essa tarefa será associada. A esses critérios daremos o nome de *políticas* e uma descrição detalhada de cada uma delas pode ser obtida em [19]. A seguir serão descritas cinco heurísticas de escalonamento dinâmico, sendo quatro delas implementadas pelo Projeto *EasyGrid* e uma, a heurística *Min-Min*, implementada nesse trabalho. Todas elas estão disponibilizadas no *framework* de simulação.

- A heurística PS (*Minimal Partial Completion Time Static Priority*) seleciona a tarefa v_i do bloco com o maior *nível escalonado*. A seguir PS associa a tarefa v_i ao processador que minimizar o *tempo de fim* da execução de v_i.
- A heurística CS (*Minimal Completion Time Static Priority*) também escolhe a tarefa v_i do bloco em função do seu *nível escalonado*. Porém, a escolha do processador é feita de forma que minimize o *caminho crítico* que passa por v_i.
- Na heurística CD (*Minimal Completion Time Dynamic Priority*), as prioridades das tarefas são recalculadas a cada escolha de tarefa em todos os *eventos de escalonamentos*, com base no seu *caminho crítico*. A escolha do processador é feita de forma que minimize o *caminho crítico* que passa por v_i assim como ocorre na heurística CS.
- E a heurística CB (*Minimal Completion Time Botton Level Priority*) em que a escolha da tarefa v_i é defina pelo *nível* de v_i. A escolha do processador é feita de forma que minimize o *caminho crítico* que passa por v_i assim como na CS.

3.1.2.1 *Min-Min*

Implementada nesse trabalho, essa heurística puramente dinâmica, isto é, que não se utiliza de nenhuma informação do escalonamento estático para realizar suas associações, visa escalonar primeiro as tarefas com o menor tempo de fim dentre todas as outras tarefas da aplicação. Com isso, tarefas com baixo custo computacional serão executadas primeiro em detrimento de tarefas com alto custo computacional, pois estas últimas terão baixa prioridade.

Seu funcionamento pode ser observado no esquema a seguir. Dado um conjunto de tarefas armazenadas em uma lista L:

1. calcular o menor tempo de fim de cada tarefa em relação a todos os processadores presentes no ambiente computacional;
2. ordenar L em ordem crescente com base no tempo de fim;
3. associar a primeira tarefa vi de L ao seu respectivo processador pj;
4. remover vi de L;
5. se L não estiver vazia, voltar ao passo 1.

Essa é uma heurística muito básica e trivial, porém seu conceito pode ser extendido a fim de se obter melhores resultados. Uma possível melhoria dessa heurística, a *Segmented Min-Min* [28], se propõe a otimizar os resultados do *Min-Min* convencional da seguinte maneira:

1. calcular o menor tempo de fim de cada tarefa em relação a todos os processadores presentes no ambiente computacional;
2. ordenar L em ordem crescente, decrescente ou pela média com base no tempo de fim;
3. particionar L em n segmentos s de mesmo tamanho;
4. ordernar os segmentos em ordem decrescente;
5. para cada segmento si:

    escalonar si utilizando *Min-Min* convencional;

Através dessa otimização, a heurística *Segmented Min-Min* elimina o desbalanceamento das cargas nos processadores.

Esse Capítulo apresentou as heurísticas de escalonamento de tarefas, abordando heurísticas estáticas, dinâmicas e híbridas. No próximo Capítulo, será apresentado o modelo da hierarquia utilizado neste trabalho e os detalhes de implementação do mesmo.

Capítulo 4

Modelo da Hierarquia

Este trabalho descreve um simulador de *Framework* de Escalonamento Hierárquico, *framework* este proposto em [3] e que objetiva viabilizar o escalonamento de aplicações paralelas em grades computacionais tentando minimizar os custos de comunicação entre os processadores, tal que o tempo de execução da aplicação seja o menor possível.

A hierarquia possui três níveis distintos, cada um com sua respectiva classe de escalonador. Assim, como mostrado na Figura 4.1, no primeiro nível da hierarquia de escalonamento existe apenas um processo, que será chamado de escalonador global (EG). No segundo nível da hierarquia encontram-se os escalonadores do *site* (ES) e, por último, no terceiro nível encontram-se os escalonadores da máquina (EM). Cada escalonador possui uma função distinta dentro da hierarquia de escalonamento. Os escalonadores e suas respectivas funções implementadas dentro deste *Framework* de simulação serão descritos a seguir, de acordo com as características implementadas no ambiente de simulação aqui implementado.

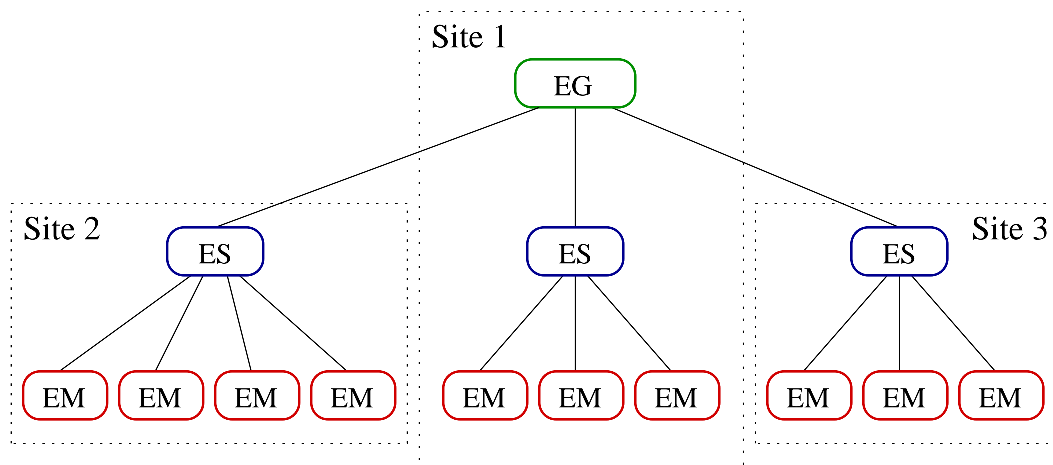

Figura 4.1: Exemplo do modelo hierárquico de escalonadores dinâmicos

4.1 Escalonador Global - EG

O escalonador global é um processo alocado em um único processador p_i que conterá todas as informações necessárias ao escalonamento, tais como: o escalonamento estático inicial que define os processadores onde cada uma das tarefas devam executar, os escalonadores de cada *site* da grade computacional e as características da aplicação como especificadas em seu modelo. O escalonador global comandará todo o escalonamento dinâmico, sendo posto em execução a cada evento de escalonamento (definido em [19]).

Há três eventos de escalonamento implementados pelo simulador:

- *TODAS_AS_TAREFAS*, definindo que todas as tarefas do bloco corrente devem ser executadas antes que possa ocorrer um próximo evento de escalonamento;
- *PRIMEIRO_NIVEL*, somente poderá ocorrer um próximo evento de escalonamento depois que alguma tarefa pertencente ao primeiro nível do bloco corrente for completamente executada;
- *ULTIMO_NIVEL*, semelhante ao evento anterior, porém uma tarefa pertencente ao último nível do bloco corrente precisa ser executada por completo para que um próximo evento de escalonamento possa ocorrer.

Durante um evento de escalonamento, o escalonador global decidirá quais tarefas da aplicação serão re-escalonadas e em que *sites* essas tarefas deverão ser alocadas. Após tomar essa decisão, o escalonador global irá disparar *threads* simulando os escalonadores locais em cada *site*, passando para eles as tarefas que deverão ser executadas dentro de seus *sites*.

4.2 Escalonadores do Site - ES

Os escalonadores do *site* são *threads* responsáveis pelo recebimento das tarefas a serem executadas no seu respectivo *site*, existindo somente um escalonador por *site*. Essas *threads* enviam mensagens contendo tarefas da aplicação através de *links* para os escalonadores da máquina. Os *links* e a maneira como as mensagens são enviadas são implementados pela biblioteca *SimGrid* [18, 7], descrita na Seção 4.4.1. Os escalonadores do *site* também são responsáveis pelo envio de informações ao escalonador global após a execução das tarefas nos Escalonadores da Máquina.

4.3 Escalonadores da Máquina - EM

Um escalonador da máquina é executado em cada processador de cada *site* da grade computacional, inclusive no processador em que o escalonador do *site* está sendo executado. Dessa forma, torna-se possível alocar tarefas também aos processadores que executam os escalonadores do *site*. O escalonador da máquina é implementado utilizando o *SimGrid* [18, 7].

Cada escalonador da máquina é responsável por executar as tarefas da aplicação designadas a ele e, ao término da execução dessas tarefas, reportar os resultados obtidos ao respectivo escalonador do *site*. As associações do escalonamento e a execução das tarefas realizadas por esses escalonadores são implementadas inteiramente através do uso de métodos da biblioteca *SimGrid* [18, 7].

4.4 Arquitetura do Simulador

O simulador foi desenvolvido utilizando a linguagem de programação C++, devido a robustez da linguagem e compatibilidade com os demais aplicativos do Portal EasyGrid [4, 2]. A biblioteca *Pthread* do Sistema Operacional Fedora Core Two [23] foi utilizada para tratar todas as ações de criação, destruição e gerenciamento das *threads* criadas pelo Escalonador Global. Uma outra biblioteca, o *SimGrid* [18, 7], foi escolhida para ser o coração do simulador, assim, ela comandará o relógio global que contará o tempo de escalonamento além de simular toda a estrutura física do *grid*. A motivação para utilização do *SimGrid* como base do simulador da hierarquia de escalonamento deste trabalho está descrita a seguir, assim como a implementação dos escalonadores e do simulador.

4.4.1 Núcleo

O principal objetivo do simulador proposto é a criação de um ambiente de análise para a execução de uma aplicação paralela em um ambiente distribuído segundo alguma métrica como o *makespan* associado à execução, ou o número de processadores utilizados, ou a quantidade de *sites* que foram utilizados durante a execução da aplicação paralela etc. O *makespan* é o tempo decorrido desde o início da execução da primeira tarefa da aplicação até o término da última tarefa. O núcleo do simulador deve ter a capacidade de armazenar esse tipo de informação para análises posteriores.

O simulador proposto foi estruturado de tal maneira que as tarefas pertencentes às

aplicações paralelas pudessem exibir dependências entre si. Nesse caso, essa dependência seria representada através de um GAD (Grafo Acíclico Direcionado) G. Também é necessário observar que o núcleo do simulador deve ser capaz de representar diversas topologias de rede.

A ferramenta para o estudo de simulações de algoritmos de escalonamento em ambientes distribuídos *SimGrid* [18, 7] foi escolhida como base para este trabalho, pois seu modelo de representação atende às exigências acima.

Apesar de atender as exigências do simulador proposto, algumas adaptações foram realizadas para o correto funcionamento do simulador. A primeira modificação foi devido ao modelo de rede topológico do *SimGrid*. Para uma topologia de processadores totalmente interconectados, um vetor foi utilizado para armazenar os processadores. Para a topologia baseada em *sites*, outra estrutura de dados teve de ser criada, o *Site*. Os *sites* também são mantidos em um vetor. A Figura 4.2 mostra o mapeamento descrito acima.

Outra pequena modificação foi devido ao modo como o *SimGrid* trata a eficiência dos processadores. Em seu modelo, um processador é descrito por sua velocidade relativa em relação a outro processador, isto é, um processador com velocidade relativa igual a dois é duas vezes mais rápido que um processador com velocidade relativa igual a um. O conceito de velocidade relativa é inversamente proporcional ao proposto pelo simulador, o fator de heterogeneidade. Assim, foi necessário inverter todos os fatores de heterogeneidade na alocação dos processadores. Na Figura 4.2 podemos observar que o processador p_0 é três vezes mais rápido do que o processador p_1.

Uma outra adaptação necessária foi a adequação do simulador ao modelo de tarefas do *SimGrid*. No modelo convencional de um GAD, o nó significa uma tarefa computacional e a aresta significa transferência de dados entre duas tarefas computacionais, logo uma aresta representa uma comunicação. No modelo implementado pelo *SimGrid*, o nó do GAD representa uma tarefa computacional e também uma comunicação, enquanto que a aresta simboliza apenas a precedência entre essas tarefas. A Figura 4.3 mostra o mapeamento desse modelo. É necessário ressaltar que é responsabilidade do simulador estabelecer a ordem de precedência correta entre as tarefas.

O *SimGrid* é uma API desenvolvida para a linguagem C que permite aos usuários manipularem duas estruturas de dados: uma para recursos, a *SG_Resource*, e outra para tarefas, a *SG_Task*. Recursos são descritos por um nome (uma identificação) e um conjunto de atributos específicos, tais como a velocidade relativa de um processador em relação a outro processador, a latência de um canal de comunicação ou largura de banda

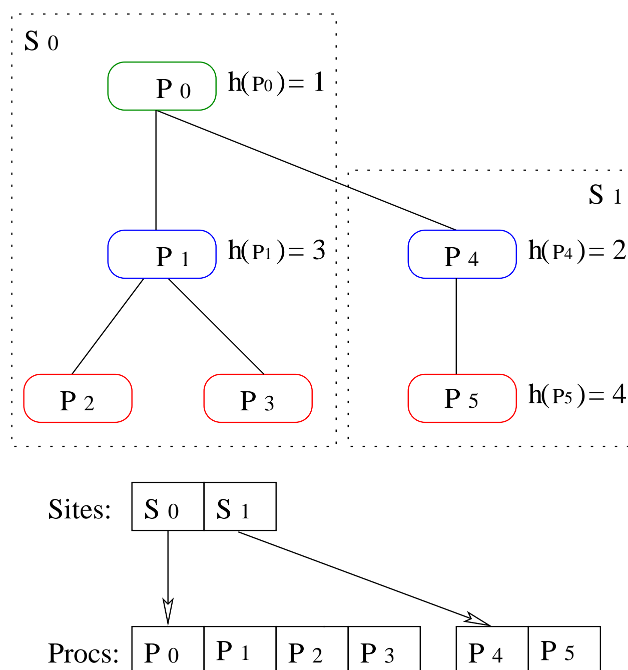

Figura 4.2: Implementação da topologia no simulador, onde h(p_i) é o fator de heterogeneidade do processador p_i

de um *link*. Tarefas são descritas por um nome (assim como os recursos), por um custo e um estado. O custo pode ser computacional (quantidade de unidades de tempo requerida para o processamento da tarefa) ou pode ser o custo de uma comunicação por entre um *link* (quantidade de unidades de tempo requerida para que a informação trafegue pelo *link*). O estado de uma tarefa define seu ciclo de vida. Uma tarefa, seja uma computação ou uma comunicação, pode estar em um, e apenas um, dos seguintes estados:

- *SG_NOT_SCHEDULED*, significando que a tarefa ainda não foi associada a nenhum recurso;
- *SG_SCHEDULED*, significando que a tarefa já está alocada a um recurso porém ainda não está pronta para ser executada devido as suas precedências;
- *SG_READY*, significando que a tarefa já pode ser executada, porém ainda não começou a sua execução;
- *SG_RUNNING*, significando que a tarefa está sendo executada em algum recurso;
- *SG_DONE*, significando que a tarefa já foi executada completamente;
- *SG_FAILED*, que significa que a tarefa terminou a execução com alguma falha.

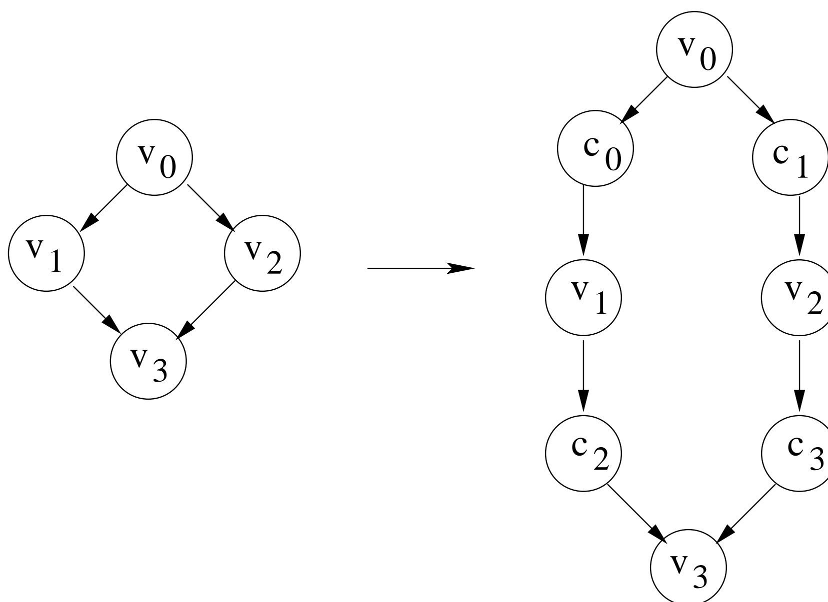

Figura 4.3: Implementação do modelo de tarefas no simulador

A API do *SimGrid* fornece funções para criação, destruição e monitoramento de recursos e tarefas, além de um conjunto específico de funções para descrever e aplicar dependências entre tarefas e comportamentos de erros. Ela possui também funções para associar tarefas a recursos de acordo com um dado escalonamento e também desfazer uma associação previamente realizada. A mais importante das funções, a *SG_simulate()*, tem por objetivo realizar efetivamente a simulação da execução das tarefas previamente alocadas aos recursos. A *SG_simulate()* leva as tarefas através de seus respectivos ciclos de vida e simula a execução das tarefas dependendo dos seguintes parâmetros:

- *time*: Número de segundos no qual essa função permanecerá executando. Pode assumir qualquer valor real. Caso o valor de *time* seja um número menor ou igual a zero, então a *SG_simulate()* não levará em conta esse parâmetro;

- *task_condition*: Condição de parada indicando quando a simulação estará completa. As possíveis condições são:

  \- *SG_ALL_TASK*, indicando que a *SG_simulate()* deve parar de executar quando todas as tarefas escalonadas forem executadas;

  \- *SG_ANY_TASK*, indicando que a *SG_simulate()* deve parar de executar assim que qualquer uma das tarefas escalonadas for executada;

  \- *NULL*, indicando que esse parâmetro não deve ser levado em consideração;

No caso de ambos os parâmetros estarem ativados, a *SG_simulate()* retorna para o fluxo de execução do programa principal assim que a primeira condição for satisfeita.

Ao terminar, a *SG_simulate()* retorna uma lista com todas as tarefas que foram completamente executadas. Após a execução da aplicação paralela, é possível recuperar o *makespan* associado à execução. Nesse caso, esse valor seria retornado através de uma chamada à função *SG_getClock()*, que recupera o tempo global.

4.4.2 Implementação

Como a implementação do simulador foi realizada sobre a camada do *SimGrid* como núcleo, foi possível focalizar no problema de escalonamento hierárquico. A seguir, será mostrado e detalhado o pseudo-código das funções mais importantes do simulador hierárquico.

O Algoritmo 1 é o principal algoritmo do simulador. Ele representa o funcionamento do escalonador global, carregando e armazenando todas as configurações e informações necessárias para futuras análises, exposição de resultados etc. Além disso, essa função tomará todas as decisões sobre o escalonamento de uma maneira global, de acordo com suas configurações. Em outras palavras, o escalonador global decidirá se o primeiro bloco de tarefas da aplicação será reescalonado, conforme descrito em [26]. Em caso afirmativo, será utilizado alguma política de escalonamento dinâmico para realizar o reescalonamento (através da chamada à função *var_dinamica()*). Caso contrário, será mantido o mesmo escalonamento definido estaticamente (utilizando a função *var_estatica()*). Após o primeiro bloco, todos os demais utilizarão somente informações dinâmicas para realizar seus escalonamentos.

Na sequência, essa função realiza o escalonamento dinâmico no bloco corrente, de acordo com as políticas definidas para o escalonamento dinâmico, através da chamada à função *escalona_dinamico()*. Após essa etapa, cada tarefa do bloco corrente será entregue a seu respectivo *site*, através da chamada da função *difunde_bloco()*. Então, todos os escalonadores do *site* serão ativados com a função *ativa_site()*. Nesse momento, diferentes configurações de políticas assim como a informação se haverá reescalonamento dentro do *site* podem ser carregadas nos escalonadores dos *sites*.

Por fim, a função *executa_bloco()* será chamada para que as tarefas da aplicação sejam executadas de acordo com o evento de escalonamento definido. Nesse ponto, as tarefas serão executadas concorrentemente, pois cada *site* será executado em uma *thread* dife-

rente. Ao terminar esse processo, todas as informações relevantes serão armazenadas em *buffers* pertencentes ao processo principal, possibilitando, assim, que diferentes métricas sejam aplicadas aos resultados da simulação.

Algoritmo 1 : *escalona_global*$(\boldsymbol{G}, \boldsymbol{A}, \boldsymbol{variante})$

```
1   carrega_configuracao(G, A);
2   se variante = DINAMICA então
3       var_dinamica();
    senão
4       var_estatica();
    fim se
5   difunde_bloco(0);
6   ativa_site();
7   executa_bloco(0);
8   para k := 1 até num_blocos − 1 faça
9       escalona_dinamico(bloco[k]);
10      difunde_bloco(k);
11      ativa_site();
12      executa_bloco(k);
    fim para
```

No Algoritmo 1, o pseudo-código do escalonador global é colocado, onde:

- G: Indica o arquivo de entrada contendo as especificações do GAD, como a quantidade de tarefas e seus respectivos custos e a relação de dependência entre elas;
- A: Indica o arquivo com as especificações da arquitetura como a quantidade de processadores, a quantidade de canais de comunicação e a disposição da hierarquia da grade computacional;
- $variante$: Determina se o primeiro bloco será escalonado estática ou dinamicamente.

O Algoritmo 2 simula a execução do escalonador dinâmico, responsável por fazer a associação entre tarefas e processadores durante a execução da aplicação, considerando as características do ambiente. A lista de tarefas recebidas como parâmetro por essa função é inicialmente ordenada através da função *ordena_tarefas()*. Essa ordenação será realizada considerando os parâmetros definidos para as políticas de escalonamento dinâmico. Após essa etapa, um laço de repetição será executado até que não existam mais tarefas na *lista_tarefas*. A cada iteração, um par tarefa processador será escolhido de acordo com alguma política de escalonamento, utilizando as funções *escolhe_tarefa()* e

escolhe_processador() respectivamente. Por fim, será realizada uma associação entre o par escolhido utilizando-se a função *associa()*, que removerá inclusive a tarefa escolhida da *lista_tarefas*. Após a execução dessa função, todas as tarefas da aplicação estarão associadas a algum recurso.

Algoritmo 2 : *escalona_dinamico*(***lista_tarefas***)

```
1  ordena_tarefas();
2  enquanto lista_tarefas <> vazia faça
3      tarefa := escolhe_tarefa();
4      processador := escolhe_processador();
5      associa(tarefa, processador);
   fim para
```

Onde:

- *lista_tarefas*: A lista de tarefas a serem escalonadas dinamicamente em um determinado momento (ou evento de escalonamento);
- *ordena_tarefas*(): Ordena as tarefas de acordo com uma prioridade específica, como, por exemplo, o tempo de fim de cada tarefa;
- *escolhe_tarefas*(): Define a tarefa a ser escalonada dinamicamente considerando a de maior prioridade. Essa prioridade é calculada de acordo com alguma política de escalonamento dinâmica como PS, CS, CD etc;
- *escolhe_processador*(): Define o processador a ser utilizado no escalonamento da tarefa definida na função *escolhe_tarefas*();
- *associa*(): Associa a tarefa escolhida ao processador escolhido, além de remover a tarefa da *lista_tarefas*.

O Algoritmo 3 realiza a distribuição das tarefas, de um determinado bloco, para seus respectivos *sites*. Essa distribuição é realizada com base no conhecimento prévio da associação entre tarefas e processadores (associação essa realizada através da chamada prévia à função *escalona_dinamico()*). Para cada *site*, essa função percorre todas as tarefas do bloco que está sendo difundido, verificando associações entre as tarefas desse bloco e os processadores do *site* corrente. Toda vez que uma associação for encontrada, a tarefa em questão deverá ser adicionada ao *site* corrente. Tarefas são adicionadas a *sites* através da chamada à função *adiciona_tarefa()*.

Algoritmo 3 : *difunde_bloco*(**num_bloco**)

```
1  para i := 0 até num_sites faça
2      para j := 0 até num_procs faça
3          para k := 0 até num_tarefas faça
4              se proc[j] = proc_tarefa então
5                  adiciona_tarefa(site[i]);
               fim se
           fim para
       fim para
   fim para
```

Onde:

- *num_sites*: Número de *sites* da hierarquia;
- *num_procs*: Número de processadores do *site* i;
- *num_tarefas*: Número de tarefas do bloco de índice *num_bloco*;
- *proc_tarefa*: Processador ao qual a tarefa k está associada;
- *adiciona_tarefa*(): Adiciona a tarefa corrente a um determinado *site*.

O Algoritmo 4 é responsável por colocar em execução as tarefas de um determinado bloco da aplicação, de acordo com o evento de escalonamento. Nessa função, o número do bloco a ser escalonado (*num_bloco*) e o evento de escalonamento (*evento*) são fornecidos pelo Escalonador Global. Os eventos de escalonamento presentes nessa função já foram definidos anteriormente. Caso o evento de escalonamento for *TODAS_AS_TAREFAS*, a função *executaTodas()* será chamada para executar todas as tarefas do bloco corrente. Se o evento for *PRIMEIRO_NIVEL*, a função *primeiroNivel()* será chamada para recuperar o primeiro nível do bloco corrente. Então a função *executaNivel()* é chamada para executar tarefas do nível selecionado. Quando o evento de escalonamento for *ULTIMO_NIVEL*, a função *ultimoNivel()* será chamada para retornar as tarefas pertencentes ao último nível do bloco a ser executado. Após isso, a função *executaNivel()* é chamada para executar essas tarefas do nível selecionado.

```
Algoritmo 4 : executa_bloco(num_bloco, evento)
1  se evento = TODAS_AS_TAREFAS então
2      executaTodas();
   senão
3      se evento = PRIMEIRO_NIVEL então
4          nivel := primeiroNivel(numBloco);
5          executaNivel(nivel);
       senão
6          se evento = ULTIMO_NIVEL então
7              nivel := ultimoNivel(numBloco);
8              executaNivel(nivel);
           fim se
       fim se
   fim se
```

Onde:

- *evento*: Representa o tipo de evento de escalonamento escolhido: *TODAS_AS_TAREFAS*, *ULTIMO_NIVEL* ou *PRIMEIRO_NIVEL*;
- *executaTodas*(): Essa função faz uma chamada à *SG_Simulate* com parâmetro *SG_ALL_TASK*, executando todas as tarefas escalonadas;
- *executaNivel*(): Essa função faz chamadas à *SG_Simulate* com parâmetro *SG_ANY_TASK* até que alguma tarefa do *nivel* recebido como parâmetro seja executada;
- *primeiroNivel*(): Retorna o menor nível encontrado dentre as tarefas dos blocos;
- *ultimoNivel*(): Retorna o maior nível encontrado dentre as tarefas dos blocos.

Este Capítulo apresentou o modelo da hierarquia utilizado neste trabalho e detalhes referentes à implementação do *Framework*. No próximo Capítulo será apresentado um ambiente de análise para escalonamentos híbridos.

Capítulo 5

Ambiente de Análise

Um dos principais objetivos desse trabalho é prover um ambiente de simulação para análise e avaliação de políticas de escalonamento, tanto dinâmicas quanto híbridas, em uma grade computacional hierárquica. Esta versão se fez necessária para que um ambiente mais próximo ao *Framework EasyGrid* fosse simulado. Para isso, o simulador desenvolvido foi inserido na ferramenta de simulação [9] do Portal *EasyGrid* [2, 4]. A interface dessa ferramenta foi modificada de modo a atender às necessidades do simulador.

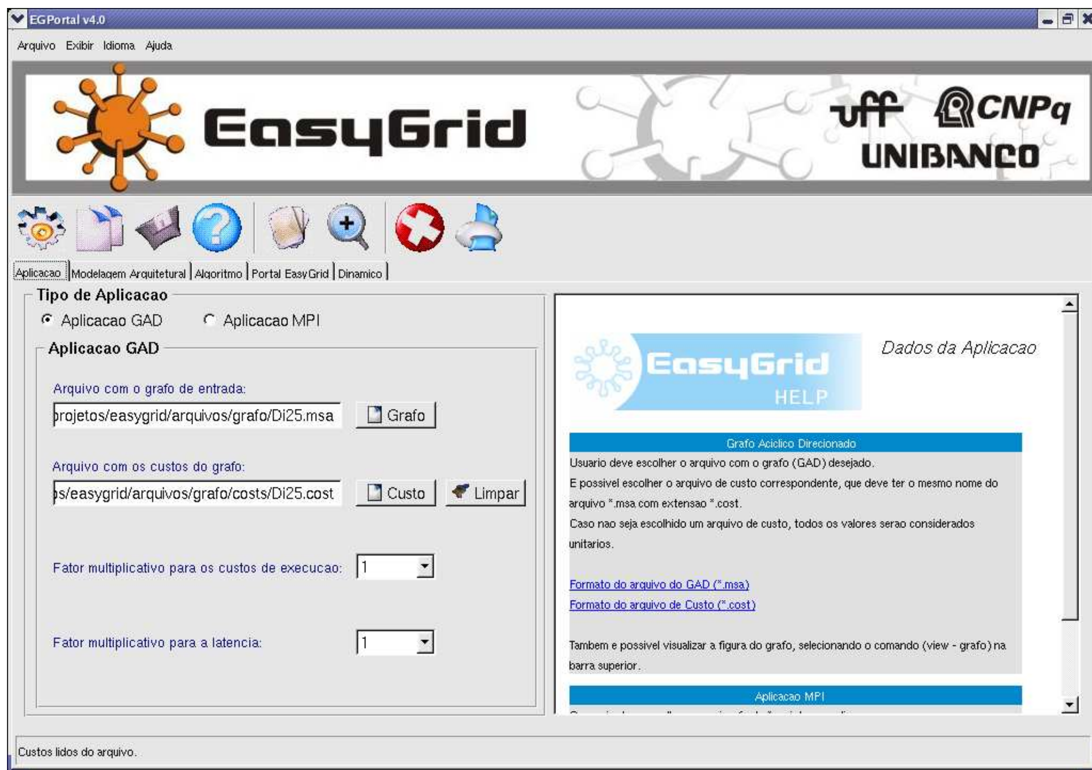

Figura 5.1: Tela da entrada de dados da aplicação

A descrição da ferramenta e das adaptações realizadas podem ser obtidas no Apêndice A.

A seguir, serão apresentadas as funcionalidades da ferramenta, exemplificadas através de um estudo de caso, onde será mostrado o resultado do escalonamento estático e dinâmico, e um estudo estatístico, contendo avaliações de desempenho das várias políticas de escalonamento implementadas no simulador.

5.1 Estudo de Caso

Esse estudo de caso se divide da seguinte maneira: na primeira parte serão apresentadas todas as configurações relevantes tanto para o escalonador estático quanto para o dinâmico, tais como: o GAD, a arquitetura e a hirarquia; a segunda parte, que se focará no resultado do escalonador estático; e na terceira, detalhando o funcionamento do escalonador híbrido.

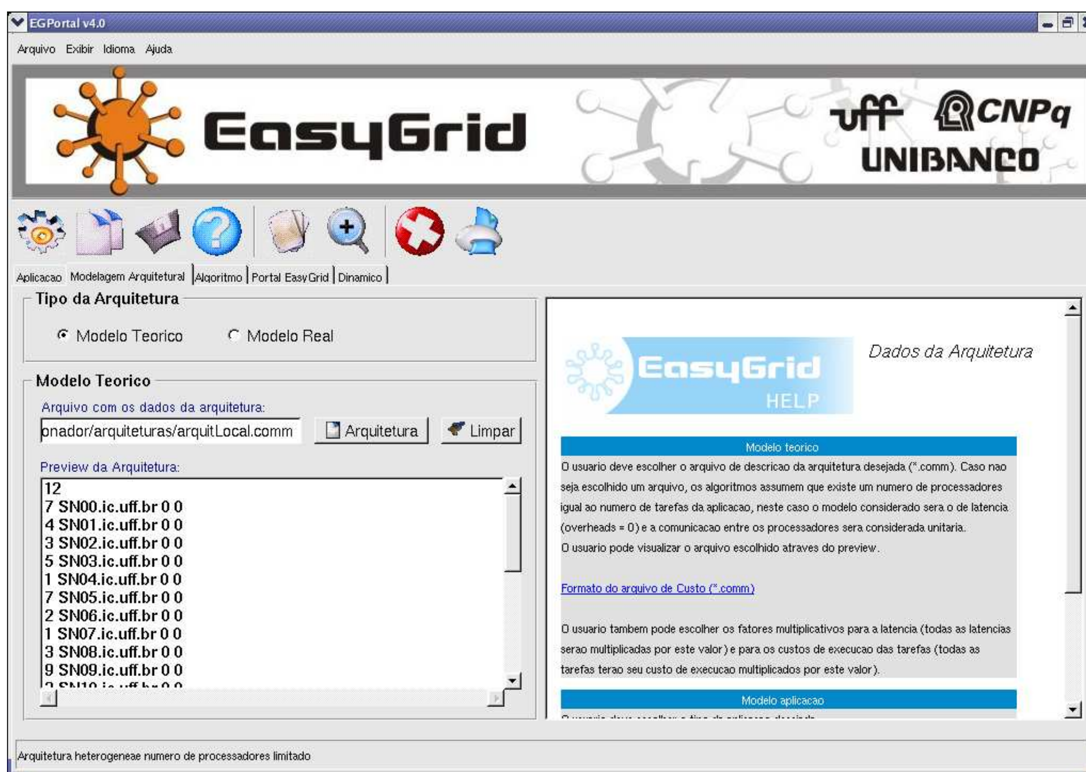

Figura 5.2: Tela da entrada de dados da arquitetura

5.1.1 Configurações Iniciais

Na Figura 5.1, pode-se observar a tela da ferramenta onde são capturados os dados da aplicação.

A Figura 5.2 mostra a tela onde é realizada a entrada dos dados referentes à arquitetura.

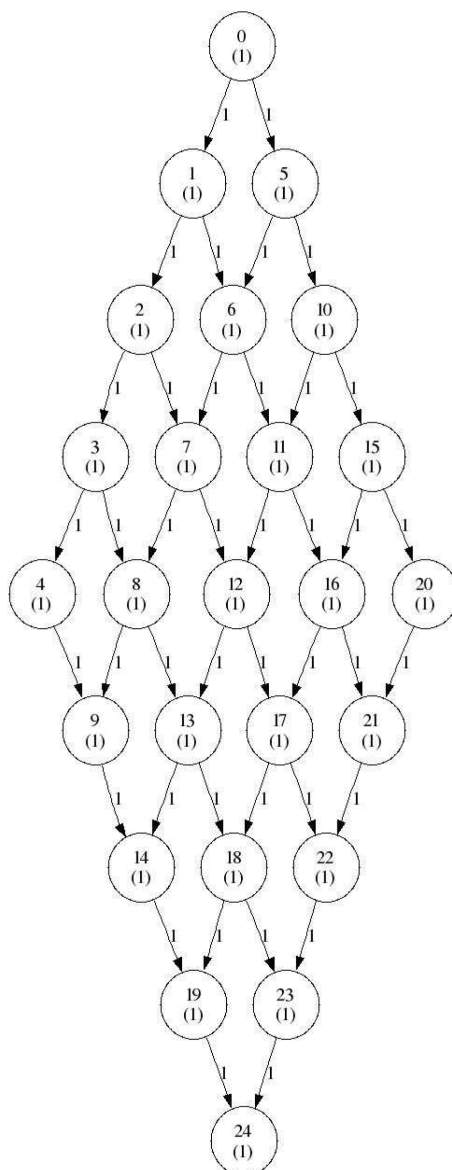

Figura 5.3: Grafo Diamante 25

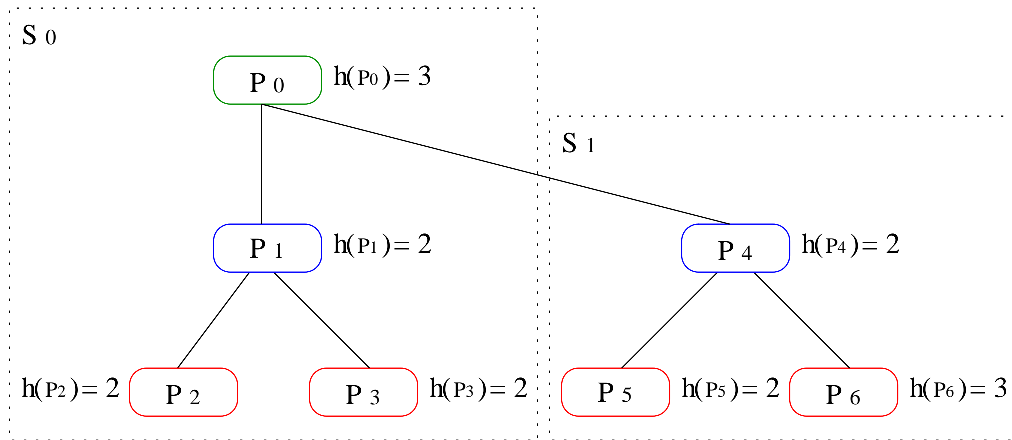

Figura 5.4: Hierarquia representativa da arquitetura utilizada

5.1.1.1 GAD

Para esse estudo de caso foi escolhido um grafo G, do tipo diamante contendo 25 tarefas, como mostrado na Figura 5.3. Todas as tarefas de G possuem custo de execução unitário e o fator multiplicativo de custo foi escolhido como sendo 1, como mostrado na Figura 5.1.

5.1.1.2 Arquitetura e Hierarquia

A arquitetura escolhida para o estudo de caso está mostrada na Figura 5.4. Essa arquitetura possui sete processadores heterogêneos, cada um com seu respectivo fator de heterogeneidade ($h(p_n)$) e utiliza o modelo de latência com fator multiplicativo igual a 1, como mostrado na Figura 5.2. O tempo gasto no preparo para o envio e recebimento de informações é considerado instantâneo.

A hierarquia da arquitetura está dividida em dois sites, o primeiro contendo os processadores p_0, p_1, p_2 e p_3 e o segundo contendo os processadores p_4, p_5 e p_6. O processador p_0 executará o escalonador global (EG) e os processadores p_1 e p_4, os escalonadores do site (ES). Todos os processadores dessa grade estarão executando os escalonadores da máquina (EM), mesmo aqueles que estiverem executando outros escalonadores.

5.1.2 Escalonamento Estático

A Figura 5.5 mostra a tela de configuração do escalonador estático onde é possível determinar o algoritmo estático a ser utilizado assim como diferentes prioridades e configurações para esse algoritmo. Nesse estudo de caso, foi escolhido o *List-Scheduling* Configurável

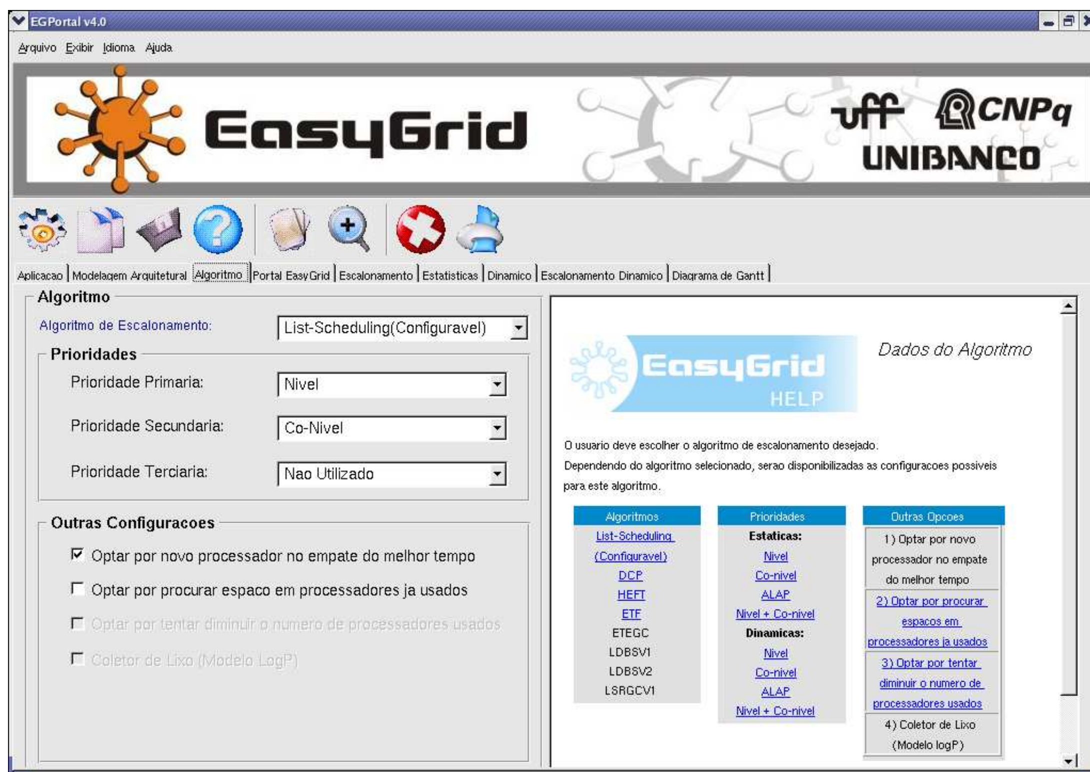

Figura 5.5: Configuração inicial do escalonador estático

para realizar o escalonamento estático.

A Figura 5.6 mostra o *Diagrama de Gantt* referente ao resultado do escalonamento estático. Nesse diagrama, cada linha equivale a um processador, identificado na primeira coluna, e cada tarefa é representada através de um retângulo, identificada pelo número em seu interior. Com isso, é possível identificar a distribuição das tarefas nos processadores e também o *makespan* da aplicação, que foi de 46 unidades de tempo no exemplo estudado.

5.1.3 Escalonamento Dinâmico

Na Figura 5.7, é apresentada a tela inicial de configuração do escalonador dinâmico. A explicação detalhada dessa tela encontra-se no Apêndice A. A seguir será mostrado o passo-a-passo da execução do escalonador híbrido.

Na Figura 5.8 pode-se observar o primeiro bloco escalonado. Nesse primeiro evento de escalonamento, foram escalonadas as tarefas 0, 1, 5, 2 e 6 no processador 1 e a tarefa 10 no processador 2.

Na Figura 5.9, observa-se que no segundo evento de escalonamento foram escalonadas as tarefas 3, 7, 4, 8 e 9 no processador 1 e as tarefas 11, 15, 12, 16, 20, 13, 17 e 21 no

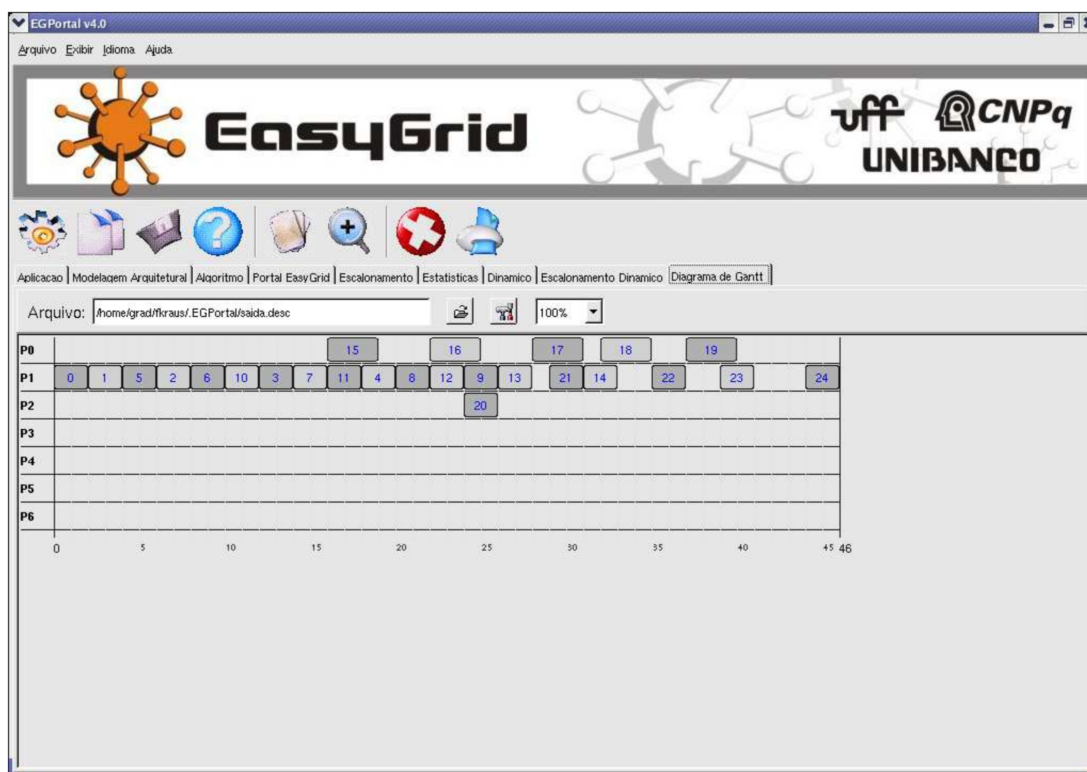

Figura 5.6: Resultado final do escalonamento estático

processador 2. Nesse momento, a tabela do lado esquerdo já exibe as tarefas do primeiro bloco que foram executadas.

Na tabela da direita na parte superior da tela da Figura 5.10, é mostrado as tarefas do terceiro bloco: 14, 18, 22, 19, 23 e 24. Todas elas foram escalonadas no processador 2. Nesse momento, pode-se observar na tabela da esquerda que todas as tarefas já terminaram de executar.

A Figura 5.11 mostra o resultado final do escalonamento híbrido (e execução da aplicação) através do diagrama de Gantt.

5.2 Outras Funcionalidades

Em um ambiente de análise faz-se necessário a existência de funcionalidades que facilitem a geração de resultados e a extração de informações dos mesmos. A parte de geração de resultados do Portal EasyGrid possui uma interface para realização de testes cujos resultados são exibidos nas abas *Algoritmo* $\times$ *Grafo*, *Algoritmo Par-a-Par* e *Estatísticas*, onde é possível coletar informações sobre os testes realizados.

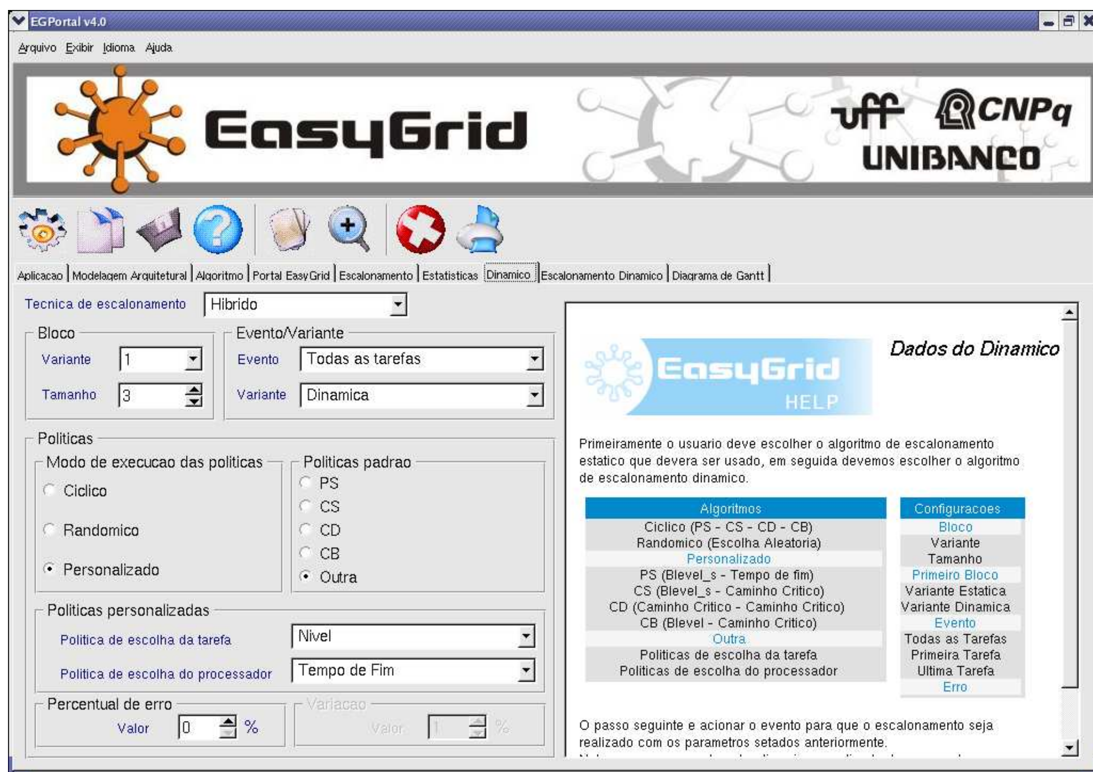

Figura 5.7: Configuração inicial do escalonador híbrido

Visando agilizar a geração de resultados, a ferramenta permite ao usuário selecionar vários algoritmos e várias aplicações para serem executadas. Além disso, pode-se acrescentar a mesma heurística com diferentes parâmetros de execução.

5.2.1 Algoritmo × Grafo

A Figura 5.12 mostra a tela de resultados *Algoritmo* × *Grafo*, onde cada linha equivale a uma aplicação e cada coluna a um algoritmo. Cada célula possui duas informações: o *makespan* e o número de processadores utilizados naquela execução. O número de processadores fica entre parênteses.

Como, muitas vezes, é difícil identificar uma heurística que seja sempre melhor do que todas as outras, pode-se avaliar aquela que apresenta o melhor desempenho em classes de grafos específicos. Essa tela permite esse tipo de avaliação e além disso, também leva em consideração a arquitetura utilizada.

A partir da Figura 5.12 é possível concluir qual o melhor algoritmo de escalonamento para cada classe de aplicação neste experimento, considerando apenas o *makespan* e o número de processadores utilizados por esse algoritmo. Para as aplicações *Diamante*

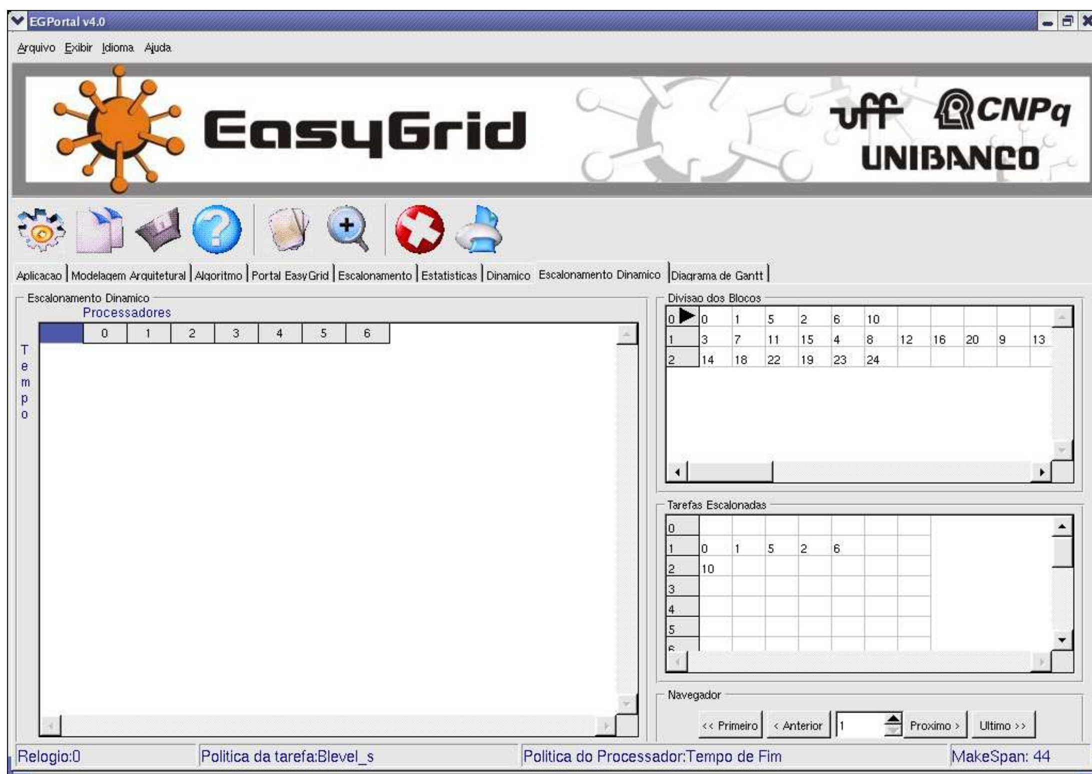

Figura 5.8: Primeiro evento de escalonamento

(Di), observa-se que os algoritmos *CS* e *CD* são os mais indicados, pois apresentam os melhores *makespans*. Para as aplicações *KPSG*, os algoritmos *Cíclico* e *CD* obtiveram os melhores makespans, porém o algoritmo *Cíclico* utilizou o menor número de processadores. Para as aplicações *In-Tree* (In) e demais aplicações, todos os algoritmos obtiveram o mesmo *makespan*, porém o algoritmo *PS* se destacou já que utilizou um número menor de processadores.

5.2.2 Algoritmo Par-a-Par

A Figura 5.13 mostra a tela de resultados *Algoritmo Par-a-Par*, onde tanto as linhas quanto as colunas equivalem as heurísticas. Cada célula representa a comparação de um par de heurísticas, indicando a quantidade de Vitórias (V), Empates (E) e Derrotas (D).

Como o problema de escalonamento é NP-Completo e diferentes resultados são obtidos de acordo a modelagem do problema, a solução ótima para um escalonamento dificilmente estará disponível. Por isso, a comparação entre heurísticas é necessária. Essa tela tem por objetivo apresentar a comparação de diversas heurísticas levando em consideração o *makespan*.

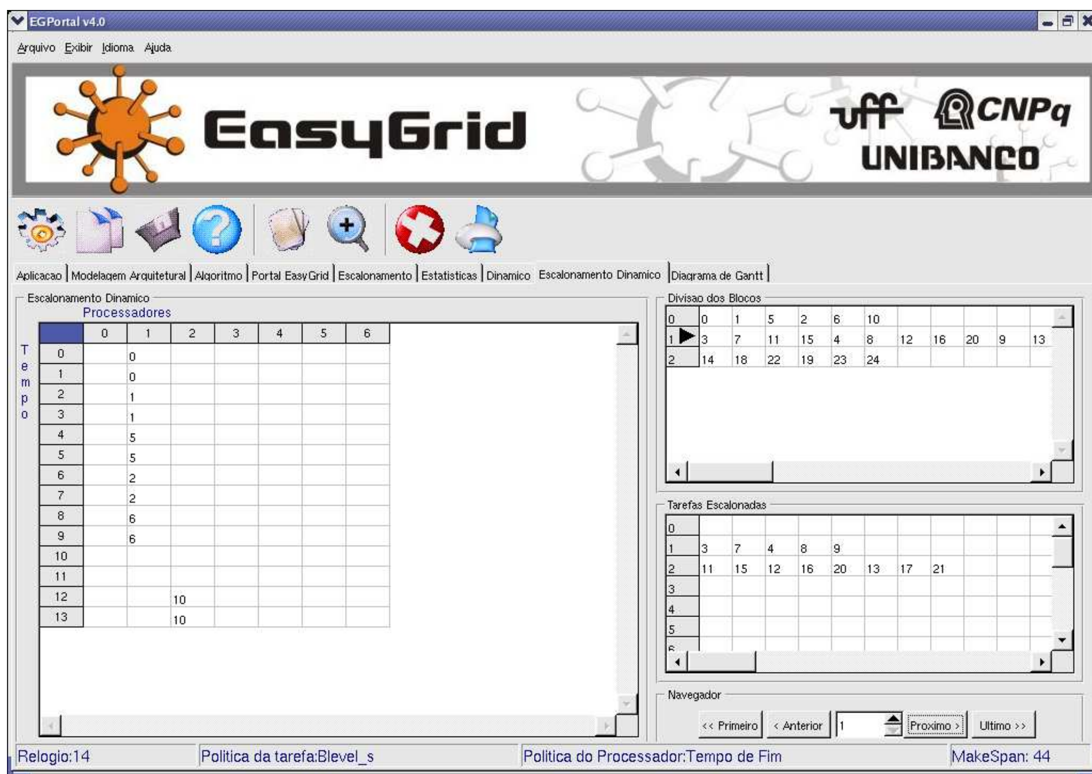

Figura 5.9: Segundo evento de escalonamento

Através da Figura 5.13 pode-se definir o melhor algoritmo em relação a todos os outros. Essa tabela mostra que o algoritmo *PS* possui a melhor contagem de vitórias em relação a todos os outros, para esse dado experimento. Em seguida, o algoritmo *CD* é classificado como sendo o segundo, seguido dos algoritmos *CB* e *CS*.

5.2.3 Estatísticas

A tela de resultados estatísticos compila todos os resultados de *Algoritmo* × *Grafo* e *Algoritmo Par-a-Par*, gerados a partir de dados fornecidos previamente tais como grafos de diferentes classes, arquiteturas e heurísticas de escalonamento. Após isso, gera estatísticas relativas ao percentual de ganho, empate e número de vezes em que cada heurística produziu resultados melhores ou iguais a todas as outras. Além disso, realiza uma medição de qualidade, através do cálculo do grau de degradação médio do melhor *makespan*, considerado o escalonamento de maior qualidade aquele que for mais próximo de um.

A Figura 5.14 mostra a tela de resultados estatísticos, onde os algoritmos estão representados na primeira coluna, o percentual de vitórias na segunda, o percentual de empate na terceira, o percentual de vezes em que foi melhor ou igual na quarta e a qualidade na

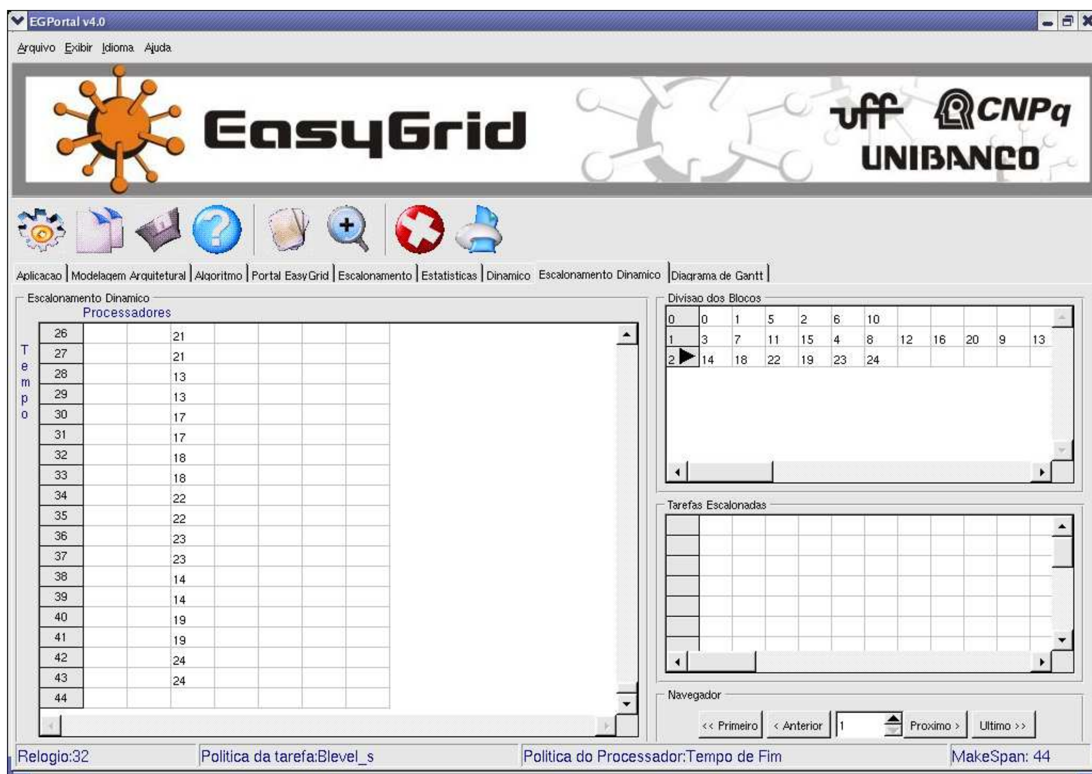

Figura 5.10: Terceiro evento de escalonamento

quinta coluna.

Na Figura 5.14 observa-se que, para este dado experimento, os algoritmos obtiveram resultados iguais na maioria das vezes, resultando em um elevado índice de empate. Os algoritmos *PS* e *CD* nunca obtiveram resultados melhores do que nenhum outro nesse cenário e, com isso, suas medidas de qualidade foram as que mais se afastaram da qualidade ótima. Já os algoritmos *CS* e *CB* possuem o mesmo percentual de vitórias. Apesar disso, o algoritmo *CB* ainda é considerado o melhor, pois possui a qualidade mais próxima do valor ótimo.

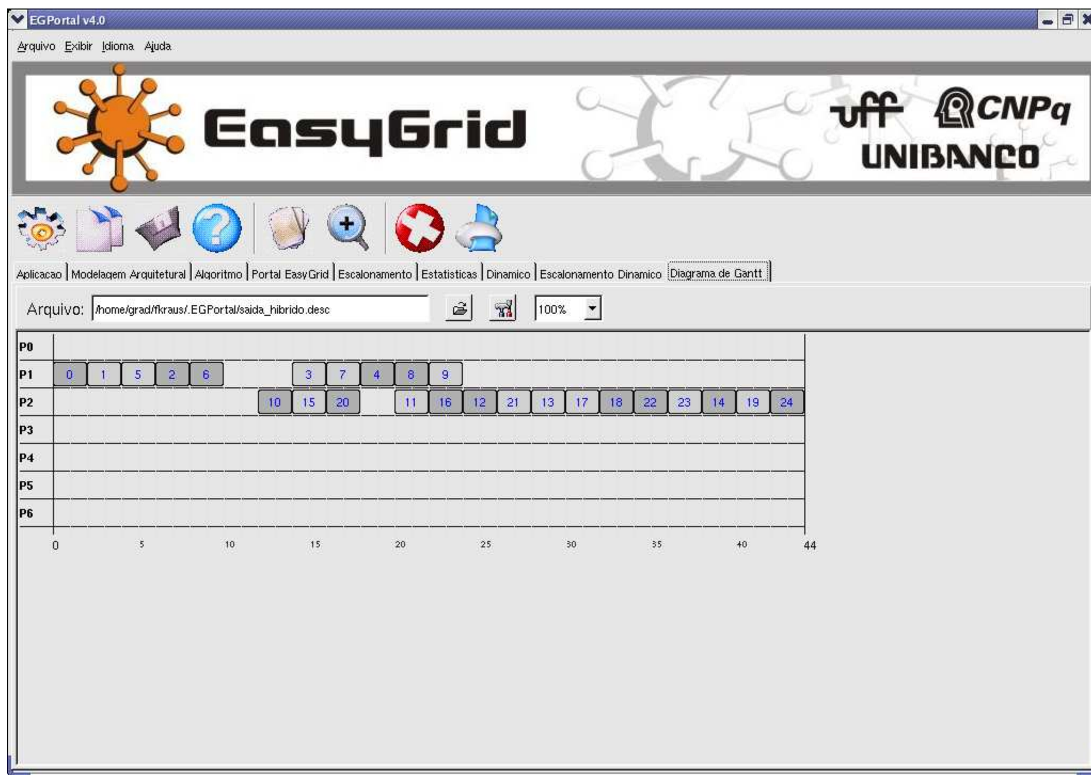

Figura 5.11: Resultado de escalonamento híbrido

Arquivo View Idioma Ajuda

EasyGrid

Aplicação | Modelagem Arquitetural | Algoritmos | Portal EasyGrid | Teste | Comparacao Grid | Estatistica Grid | Dinâmico | Escalonamento Dinâmico

Algoritmo(s)

Grafo	Cíclico(2,1)	Randômico(2,1)	PS(2,1)	CS(2,1)	CD(2,1)
Di100	31(10)	32(10)	36(9)	28(10)	28(10)
Di16	11(4)	11(4)	12(3)	10(4)	10(4)
Di25	14(5)	13(5)	16(4)	13(5)	13(5)
Di36	17(6)	17(6)	20(5)	16(6)	16(6)
Di64	24(8)	25(8)	28(7)	22(8)	22(8)
Di9	7(3)	7(3)	8(2)	7(3)	7(3)
g10_54	57(25)	57(20)	57(10)	57(29)	57(27)
g4_9	12(4)	12(4)	12(4)	12(4)	12(4)
g8_35	39(14)	38(15)	38(8)	38(15)	38(16)
grafo2	16(4)	16(4)	16(4)	16(4)	16(4)
InTree15	7(11)	7(10)	7(8)	7(11)	7(11)
InTree3	3(2)	3(2)	3(2)	3(2)	3(2)
InTree63	11(45)	11(44)	11(32)	11(47)	11(47)
KPSG10	450(4)	490(4)	490(4)	450(5)	450(5)
KPSG11	18(4)	18(4)	18(3)	18(4)	18(4)

Figura 5.12: Resultados *Algoritmo* × *Grafo*

Figura 5.13: Resultados *Algoritmo Par-a-Par*

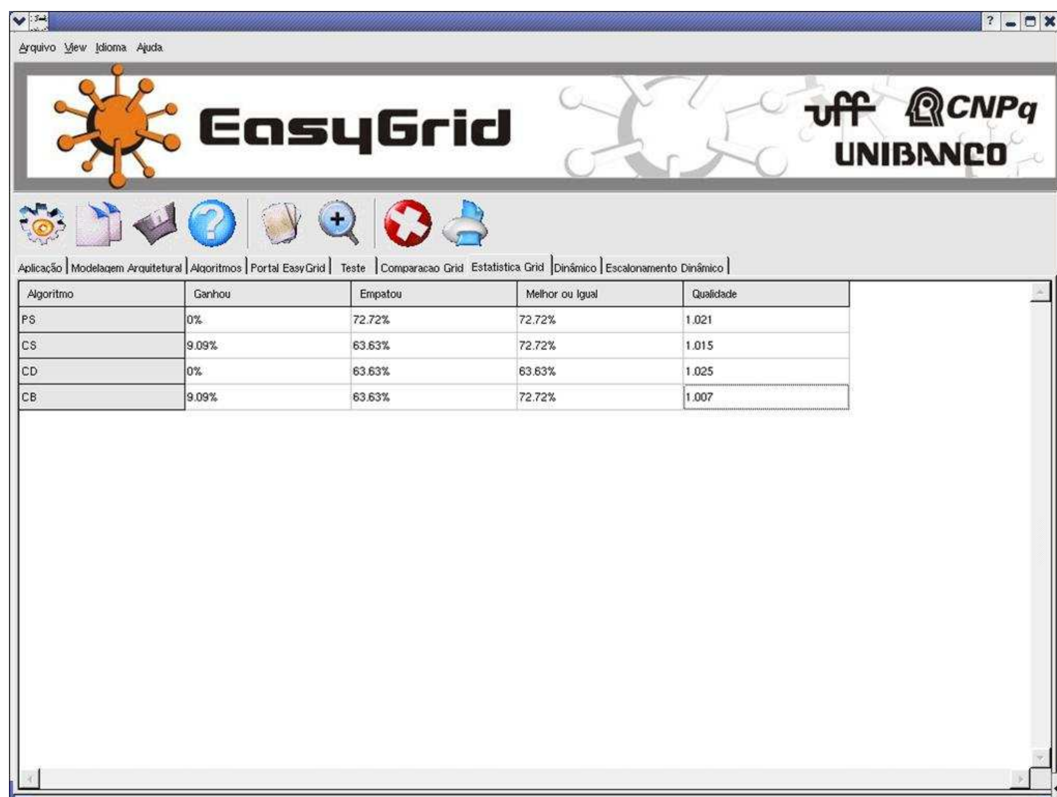

Figura 5.14: Resultados estatísticos

Capítulo 6

Trabalhos Relacionados

A simulação tem sido exaustivamente utilizada para a modelagem e avaliação de sistemas reais em diversas áreas do conhecimento. Embora existam muitas ferramentas de simulação de escalonamento de aplicações paralelas, apenas um número reduzido delas é dedicado à grades computacionais. Nesse contexto, esse Capítulo apresentará uma breve descrição das principais ferramentas pesquisadas neste trabalho, com o objetivo de identificar conceitos necessários à implementação do simulador de ambiente hierárquico de escalonamento dinâmico e execução de uma aplicação.

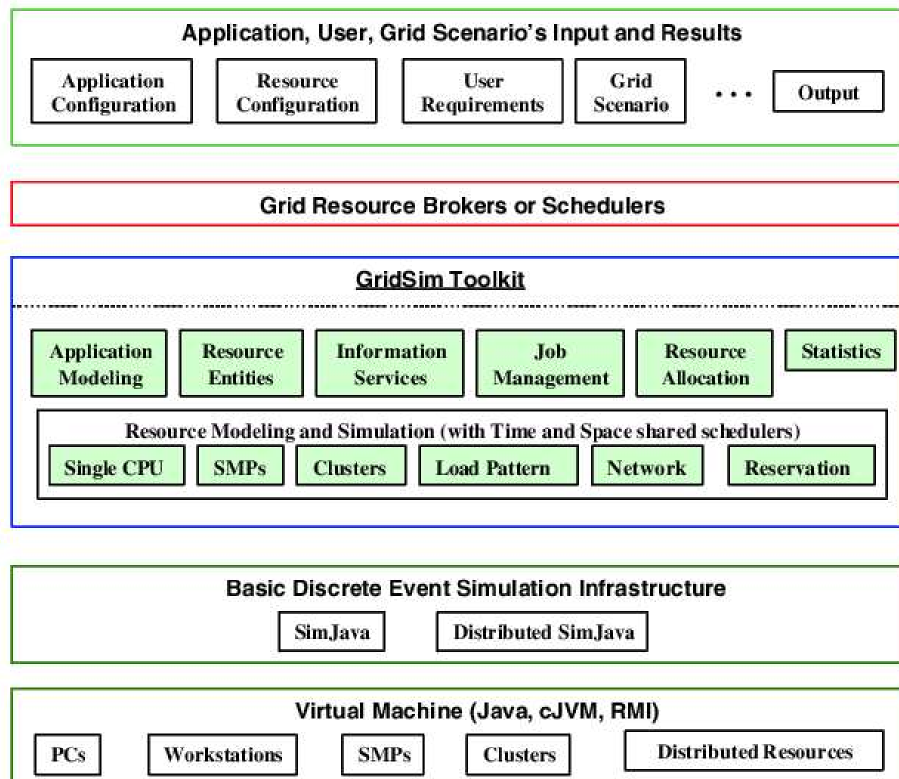

Figura 6.1: Arquitetura da plataforma e dos componentes do *GridSim*.

O *GridSim* [6], é uma ferramenta desenvolvida em Java que permite a modelagem e a simulação de usuários de sistema, aplicações paralelas e recursos computacionais. Um recurso computacional pode ser um único processador ou um multi-processador com memória compartilhada ou distribuída. O gerenciamento dessa memória e do tempo são compartilhados entre os escalonadores. Considerando essas características, essa ferramenta permite a avaliação de algoritmos de escalonamento em ambientes paralelos e distribuídos.

Na Figura 6.1, observa-se a arquitetura do *GridSim*. Nos níveis mais externos dessa arquitetura encontram-se a ***Virtual Machine*** e o ***Basic Discrete Event Simulation Infrastructure*** que denotam os componentes Java da ferramenta *GridSim*.

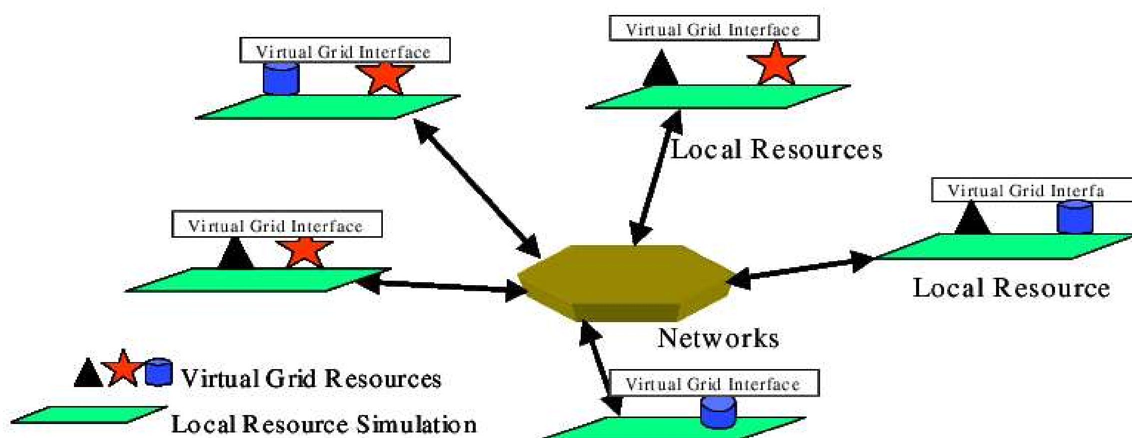

Figura 6.2: Arquitetura do simulador *MicroGrid*.

O *MicroGrid* [24], é uma ferramenta que permite a simulação de aplicações paralelas em um ambiente *Grid* virtual. É uma ferramenta flexível e permite a composição de um *Grid* virtual mascarando recursos físicos heterogêneos reais nos quais as aplicações serão realmente executadas. O *MicroGrid* suporta aplicações que utilizam a infraestrutura do *middleware Globus Grid* [12, 10], que é um pacote de serviços básicos para Grid que reúne um conjunto de ferramentas para o controle dos recursos computacionais.

Na Figura 6.2, observa-se que a simulação de recursos locais fornecem um *Grid* virtual com os recursos do ambiente computacional e também a simulação de uma infraestrutura de rede através da interação entre os recursos virtuais locais.

Finalmente, após o estudo dessas ferramentas, observou-se que qualquer uma delas poderia ter sido utilizada como base para o simulador. Apesar disso, o *GridSim* não se adequaria a esse trabalho em particular, pois tal trabalho é parte integrante do Projeto EasyGrid [2, 4], desenvolvido utilizando a linguagem C++. Já o *MicroGrid* não foi

utilizado pois incorpora características do ambiente real que não estavam no escopo desse trabalho.

Assim, dando continuidade ao trabalho realizado [26], o *SimGrid* foi escolhido como base do simulador implementado neste trabalho.

Capítulo 7

Conclusões e Trabalhos Futuros

Neste trabalho foi implementado um simulador de *framework* de escalonamento hierárquico para ambientes heterogêneos que, conforme proposto em [3], divide o trabalho de escalonamento entre os três níveis da hierarquia: escalonador global, escalonador do *site* e escalonador da máquina.

O escalonador global gerencia o andamento do escalonamento e envia todas as tarefas designadas a cada um dos escalonadores do *site*. Os escalonadores do *site* são responsáveis por gerenciar localmente os recursos computacionais e repassar aos escalonadores das máquinas as tarefas que deverão ser executadas. Os escalonadores da máquina simulam a execução das tarefas da aplicação, reportando os resultados ao respectivo escalonador do *site* que por sua vez reportam a concatenação desses resultados ao escalonador global. Desta forma, pode-se otimizar o escalonamento dinâmico em ambientes de grades já que o gerenciamento do escalonamento é feito em níveis.

Anteriormente, foi implementado uma versão na qual o escalonador global era uma entidade que gerenciava todos os recursos da grade sem custos associados à sua execução e comunicação com os processadores [26]. Essa versão não podia ser utilizada com precisão para o desenvolvimento e avaliação de heurísticas de escalonamento.

Através do modelo hierárquico implementado neste trabalho, o escalonamento de aplicações em ambientes distribuídos tornou-se mais realista ao agregar características do modelo de escalonamento real do projeto *EasyGrid*, tais como:

- A distribuição geográfica dos recursos, ou seja, a organização dos recursos em *sites*;
- O custo de comunicação entre os escalonadores da máquina e os locais e os escalonadores locais e o global;
- O custo de execução dos escalonadores.

No futuro, espera-se realizar os seguintes trabalhos:

- Adicionar ao simulador e à ferramenta de simulação a opção de realizar ou não o reescalonamento de tarefas em cada *site*. Atualmente, é possível optar pelo reescalonamento, porém essa opção se aplica a todos os *sites* não sendo possível selecionar individualmente a opção para cada *site*;
- Adicionar a possibilidade de associar na interface da ferramenta de simulação os custos ao escalonador global e aos escalonadores locais;
- Adicionar um editor de hierarquia na ferramenta de simulação, com o propósito de simplificar a criação e manipulação da hierarquia de escalonadores na própria interface. Assim, dada uma arquitetura alvo, será possível definir a organização estrutural dos recursos na grade computacional;
- Adicionar à interface da ferramenta de simulação a possibilidade de inclusão de novas heurísticas de escalonamento sem a necessidade de alteração no código-fonte da mesma. Desse modo, a ferramenta se tornará mais versátil e dinâmica.

APÊNDICE A

Usando o *Framework*

O *framework* de simulação está inserido no projeto *Easygrid* [4, 2] e pode ser utilizado através de uma das abas do projeto entitulada *Dinâmico*, Figura A.1. Nesta aba, é possível ao usuário definir todas as configurações para o simulador.

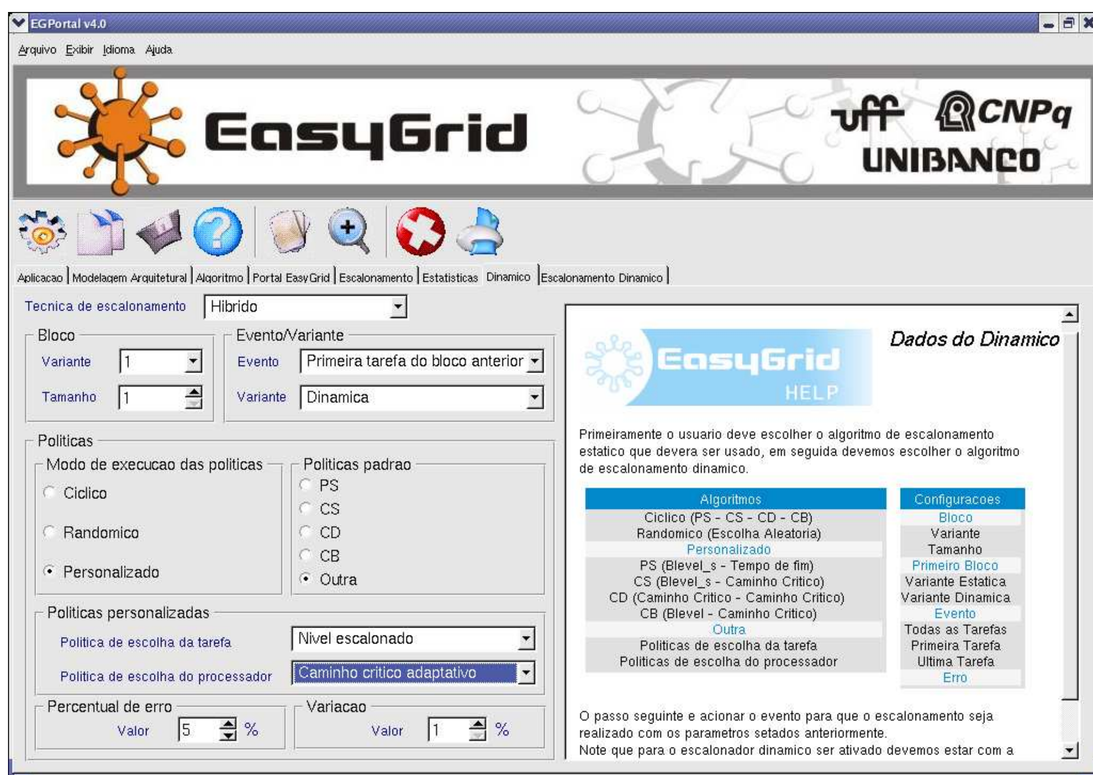

Figura A.1: Tela de configuração do escalonador dinâmico na ferramenta *Easygrid*

Na parte superior, é possível definir a *Técnica de Escalonamento* desejada. Atualmente, existem duas técnicas dinâmicas: a técnica *Híbrido*, como descrito no Capítulo 3, que utiliza uma mistura de escalonamento estático e dinâmico e a técnica *Min-Min*, que realiza um escalonamento dinâmico puro.

Na caixa *Bloco* é possível selecionar a *variante do bloco* e o *tamanho do bloco*. A *variante do bloco* é utilizada para determinar a maneira como os blocos serão divididos. A combinação dos valores das variantes acima, determinam diferentes valores para o número de blocos e a distribuição de tarefas nos mesmos.

Ao lado de *Bloco*, está a caixa *Evento/Variante*. No ítem *evento*, é possível definir quando o próximo bloco de tarefas será escalonado. Existem três tipos de eventos de escalonamento:

- *Todas as Tarefas*: o próximo bloco será escalonado quando todas as tarefas do bloco corrente forem executadas;
- *Primeiro Nível*: o próximo bloco será escalonado quando qualquer tarefa do primeiro nível do bloco corrente for executada;
- *Último Nível*: o próximo bloco será escalonado quando qualquer tarefa do último nível do bloco corrente for executada;

A *variante* corresponde ao modo sobre o qual o primeiro bloco de tarefas será escalonado, ou seja, se utilizará informações dinâmicas (opção *Dinâmica* da caixa de seleção) ou estáticas (opção *Estática* da caixa de seleção) para escalonar o primeiro bloco da execução.

Logo abaixo, encontra-se a caixa *Política*. Ela possui cinco subcaixas: *Modo de Execução das Políticas*, *Políticas Padrão*, *Políticas Personalizadas*, *Percentual de Erro* e *Variação*.

Na subcaixa *Políticas Padrão*, lista-se todas as políticas padrões de escalonamento dos blocos. Essas políticas de escalonamento, definidas em [8, 19], são compostas de uma política de escolha das tarefas e uma política para a escolha dos processadores. São elas:

- PS (*Minimal Partial Completion Time Static Priority*): *Nível Escalonado* para as tarefas e *Tempo de Fim* para os processadores;
- CS (*Minimal Completion Time Static Priority*): *Nível Escalonado* para as tarefas e *Caminho Crítico* para os processadores;
- CD (*Minimal Completion Time Dynamic Priority*): *Caminho Crítico* para as tarefas e *Caminho Crítico* para os processadores;
- CB (*Minimal Completion Time Bottom Level Priority*): *Nível* para as tarefas e *Caminho Crítico* para os processadores;

- Outra: Libera a subcaixa *Políticas Personalizadas*, descrita na sequência.

Esta subcaixa somente será ativada quando a opção *Personalizado* da subcaixa *Modo de Execução das Políticas* estiver selecionada.

Na subcaixa *Modo de Execução das Políticas*, é possível optar pelo modo de execução das políticas de escalonamento para os blocos, podendo ser uma única para todos os blocos ou um combinado de políticas, segundo alguma ordem. São eles:

- Cíclico: essa opção define que a cada bloco, uma política de escalonamento será executada, de forma cíclica e seguindo a ordem em que foram descritas acima;
- Randômico: a opção *Randômico* seleciona aleatoriamente uma das políticas padrão para cada bloco;
- Personalizado: como descrito acima, essa opção ativa a subcaixa *Políticas Padrão*, permitindo ao usuário selecionar uma política de escalonamento única para os blocos.

A subcaixa *Políticas Personalizadas* permite ao usuário selecionar uma política para a escolha das tarefas e uma outra para a escolha do processador. Essa opção é muito útil quando o usuário deseja especificar políticas, para a tarefa e para o processador, que não estão combinadas na subcaixa *Políticas Padrão*.

A subcaixa *Percentual de Erro* define o nível de instabilidade da aplicação, ou seja, essa subcaixa altera o custo de execução das tarefas de acordo com um valor aleatório que varia no intervalo de [-e, e], onde e é o *Percentual de Erro*.

A subcaixa *Variação* é ativada quando o usuário define a política *Caminho Crítico Adaptativo* para a escolha do processador na subcaixa *Políticas Personalizadas*. Essa subcaixa define a maior variação possível entre o tempo corrente estimado e tempo corrente real para um processador, onde tempo corrente é o momento no qual um processador vai estar ocioso.

Os resultados da execução podem ser visualizados em uma outra aba chamada *Escalonamento Dinâmico*, Figura A.2.

Na Figura A.2, observa-se que a tabela do lado esquerdo exibe o tempo de início e fim de cada tarefa em seus respectivos processadores. Nessa tabela, colunas representam processadores e linhas unidades de tempo.

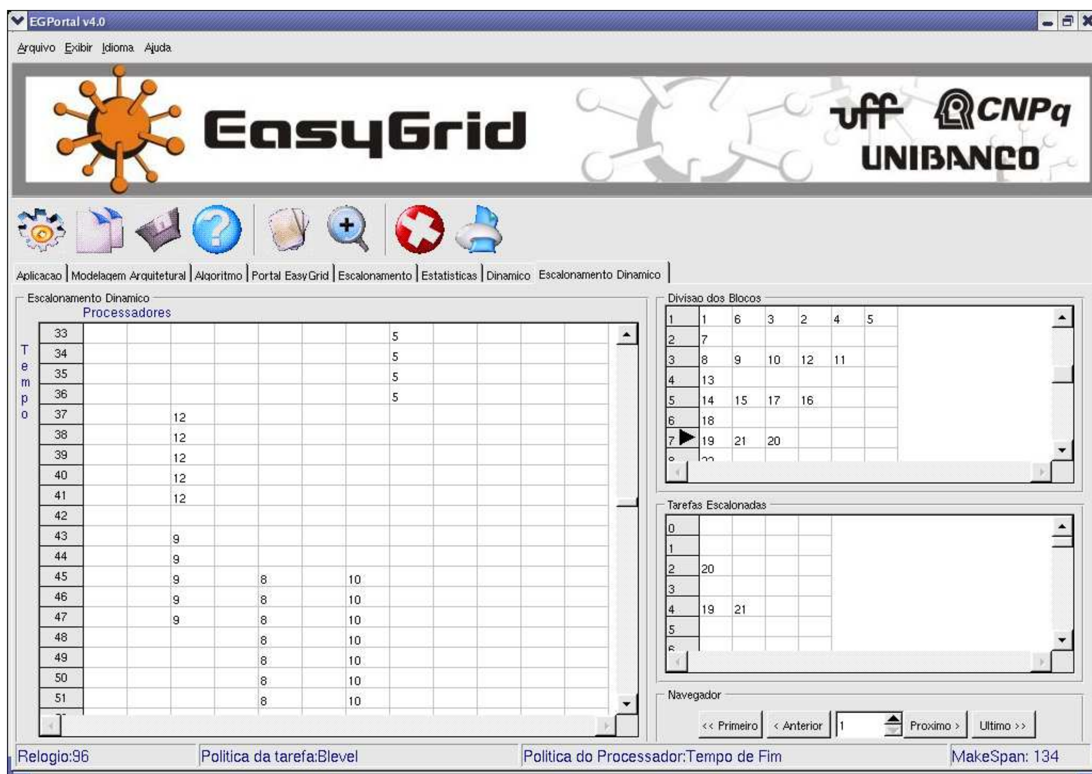

Figura A.2: Tela de visualização dos resultados do escalonador dinâmico na ferramenta *Easygrid*

A tabela do lado direito na parte superior exibe a divisão dos blocos, onde o bloco corrente é determinado ao longo da execução por uma **seta**, que na Figura A.2 esta na linha 7. Nessa tabela, cada linha equivale a um bloco contendo um conjunto de tarefas.

A tabela do lado direito na parte inferior representa o escalonamento de tarefas do bloco corrente. Durante esse evento de escalonamento, as tarefas do bloco corrente são associadas à processadores de acordo com as decisões do escalonador global. Nessa tabela, cada linha representa um processador seguido pelas tarefas associadas a ele durante esse evento de escalonamento.

Podemos observar também que existe um navegador, que é responsável por controlar as tabelas, avançando ou retornando quantos passos forem especificados no campo entre os botões. Além disso, a barra de status mostra o *makespan* total e também informa o tempo corrente, a política de escolha da tarefa e a política de escolha do processador para cada evento de escalonamento.

APÊNDICE B

Exemplos de arquivos de entrada e saída

Nesse apêndice serão apresentados exemplos dos formatos dos arquivos de entrada e saída utilizados pelo simulador. Por simplicidade, eles foram mostrados através de exemplos. São eles: arquivo contendo os parâmetros de entrada do *framework*, arquivo de entrada oriundo do escalonador estático, arquivo de saída do escalonamento híbrido, arquivo de saída do escalonador dinâmico e arquivo de saída da divisão dos blocos.

Formato do arquivo contendo os parâmetros de entrada

Este arquivo armazena todos os parâmetros de entrada do *framework*, possuindo a extensão ".txt".

/home/Grafos/Di16.msa	{Caminho do arquivo do grafo}
/home/Grafos/Custos/Di16.cost	{Caminho do arquivo contendo os custos de comunicação}
/home/Arquiteturas/arquitetura.comm	{Caminho do arquivo da arquitetura}
/home/.EGPortal/saida_hibrido.desc	{Caminho do arquivo de saída contendo o escalonamento híbrido}
/home/.EGPortal/saida.din	{Caminho do arquivo de saída contendo o resultado do escalonador dinâmico}
/home/.EGPortal/entrada_estatico.desc	{Caminho do arquivo de entrada contendo o escalonamento estático}
/home/.EGPortal/saida_divisao_blocos.blc	{Caminho do arquivo de saída contendo a divisão dos blocos}
1	{Fator multiplicativo de custo}
1	{Fator multiplicativo de latência}
1	{Variante para divisão dos blocos}
1	{Tamanho de cada bloco}
1	{Evento de escalonamento}
0	{Variante para o primeiro bloco}
0	{Modo de execução das políticas em cada bloco}
1	{Política de escolha das tarefas}
0	{Política de escolha dos processadores}
0	{Técnica de escalonamento a ser utilizada}
0	{Reescalona ou não as tarefas no site}
1	{Variação do nível de anormalidade(necessariamente $>$ 0)}
0	{Percentual de erro na variação do custo das tarefas}

Formato do arquivo de entrada oriundo do escalonador estático

Este arquivo possui informações sobre todo o escalonamento estático da aplicação que o usuário deseja executar e esse escalonamento é realizado previamente na ferramenta de simulação. Possui a extensão ".desc".

16			{Número de tarefas: 16}
4			{Número de processadores: 4}
15			{Makespan: 15}
0			{Tempo em segundos: 0}
544			{Tempo em milisegundos: 3794}
0	SN00.ic.uff.br	15	{Máquina 0; Hostname: SN00.ic.uff.br; Número de tarefas executadas: 15}
0	0	1	{Tarefa: 0; Tempo de início: 0; Tempo de fim: 1}
1	1	2	{Tarefa: 1; Tempo de início: 1; Tempo de fim: 2}
4	2	3	{Tarefa: 4; Tempo de início: 2; Tempo de fim: 3}
2	3	4	{Tarefa: 2; Tempo de início: 3; Tempo de fim: 4}
5	4	5	{Tarefa: 5; Tempo de início: 4; Tempo de fim: 5}
8	5	6	{Tarefa: 8; Tempo de início: 5; Tempo de fim: 6}
3	6	7	{Tarefa: 3; Tempo de início: 6; Tempo de fim: 7}
6	7	8	{Tarefa: 6; Tempo de início: 7; Tempo de fim: 8}
9	8	9	{Tarefa: 9; Tempo de início: 8; Tempo de fim: 9}
7	9	10	{Tarefa: 7; Tempo de início: 9; Tempo de fim: 10}
10	10	11	{Tarefa: 10; Tempo de início: 10; Tempo de fim: 11}
13	11	12	{Tarefa: 13; Tempo de início: 11; Tempo de fim: 12}
11	12	13	{Tarefa: 11; Tempo de início: 12; Tempo de fim: 13}
14	13	14	{Tarefa: 14; Tempo de início: 13; Tempo de fim: 14}
15	14	15	{Tarefa: 15; Tempo de início: 14; Tempo de fim: 15}
1	SN01.ic.uff.br	1	{Máquina 1; Hostname: SN01.ic.uff.br; Número de tarefas executadas: 1}
12	7	9	{Tarefa: 12; Tempo de início: 7; Tempo de fim: 9}
2	SN02.ic.uff.br	0	{Máquina 2; Hostname: SN02.ic.uff.br; Número de tarefas executadas: 0}
3	SN03.ic.uff.br	0	{Máquina 3; Hostname: SN03.ic.uff.br; Número de tarefas executadas: 0}

Formato do arquivo de saída do escalonamento híbrido

Possui o mesmo formato apresentado acima. Os resultados deste arquivo, porém, podem diferir do anterior na alocação das tarefas pelos processadores, assim como nos valores de *Makespan*, tempo em segundos, tempos em milisegundos. Por dificuldades na implementação, o *framework* não gera os dados das tarefas alocadas a um mesmo processador em ordem cronológica como ocorre no escalonamento estático, porém garante a corretude dos dados.

```
16                              {Número de tarefas: 16}
4                               {Número de processadores: 4}
15                              {Makespan: 15}
0                               {Tempo em segundos: 0}
544                             {Tempo em milisegundos: 35048}
0     SN00.ic.uff.br   15       {Máquina 0; Hostname: SN00.ic.uff.br; Número de tarefas executadas: 14}
0     0                1        {Tarefa: 0; Tempo de início: 0; Tempo de fim: 1}
1     1                2        {Tarefa: 1; Tempo de início: 1; Tempo de fim: 2}
2     3                4        {Tarefa: 2; Tempo de início: 3; Tempo de fim: 4}
3     5                6        {Tarefa: 3; Tempo de início: 5; Tempo de fim: 6}
4     2                3        {Tarefa: 4; Tempo de início: 2; Tempo de fim: 3}
5     4                5        {Tarefa: 5; Tempo de início: 4; Tempo de fim: 5}
6     6                7        {Tarefa: 6; Tempo de início: 6; Tempo de fim: 7}
7     7                8        {Tarefa: 7; Tempo de início: 7; Tempo de fim: 8}
9     9                10       {Tarefa: 9; Tempo de início: 9; Tempo de fim: 10}
10    10               11       {Tarefa: 10; Tempo de início: 10; Tempo de fim: 11}
11    12               13       {Tarefa: 11; Tempo de início: 12; Tempo de fim: 13}
13    11               12       {Tarefa: 13; Tempo de início: 11; Tempo de fim: 12}
14    13               14       {Tarefa: 14; Tempo de início: 13; Tempo de fim: 14}
15    14               15       {Tarefa: 15; Tempo de início: 14; Tempo de fim: 15}
1     SN01.ic.uff.br   2        {Máquina 1; Hostname: SN01.ic.uff.br; Número de tarefas executadas: 2}
8     5                7        {Tarefa: 8; Tempo de início: 5; Tempo de fim: 7}
12    7                9        {Tarefa: 12; Tempo de início: 7; Tempo de fim: 9}
2     SN02.ic.uff.br   0        {Máquina 2; Hostname: SN02.ic.uff.br; Número de tarefas executadas: 0}
3     SN03.ic.uff.br   0        {Máquina 3; Hostname: SN03.ic.uff.br; Número de tarefas executadas: 0}
```

Formato do arquivo de saída do escalonador dinâmico

Esse formato de arquivo possui a extensão ".din"e possui todas informações sobre a execução do escalonador dinâmico, tais como a alocação das tarefas nos processadores, a divisão destas tarefas nos blocos e os tempos de execução de cada bloco. Em cada bloco, é informado também a política da tarefa e do processador utilizadas neste bloco, respectivamente nomeadas de *pt* e *pp* no formato descrito abaixo. O simulador permite ao usuário, se desejar, selecionar políticas diferentes para cada bloco.

Após cada *evento de escalonamento*, o simulador tira um *snapshot* da execução das tarefas de um bloco. Uma tarefa poderá aparecer em um ou mais *snapshots* caso ela não consiga ser totalmente executada durante o *evento de escalonamento* associado a seu bloco.

```
7                        {Número de Blocos: 7}
0.000000    1   1   1    {Bloco 0; Tempo de início: 0.000000; Número de tarefas escalonadas: 1; pt: 1; pp: 1}
0           2            {Tarefa: 0; Processador: 2}
1.000000    2   2   1    {Bloco 1; Tempo de início: 1.000000; Número de tarefas escalonadas: 2; pt: 2; pp: 1}
5           2            {Tarefa: 5; Processador: 2}
1           2            {Tarefa: 1; Processador: 2}
3.000000    3   0   1    {Bloco 2; Tempo de início: 3.000000; Número de tarefas escalonadas: 3; pt: 0; pp: 1}
10          2            {Tarefa: 10; Processador: 2}
2           2            {Tarefa: 2; Processador: 2}
6           2            {Tarefa: 6; Processador: 2}
6.000000    4   1   0    {Bloco 3; Tempo de início: 6.000000; Número de tarefas escalonadas: 4; pt: 1; pp: 0}
15          2            {Tarefa: 15; Processador: 2}
3           2            {Tarefa: 3; Processador: 2}
7           2            {Tarefa: 7; Processador: 2}
11          2            {Tarefa: 11; Processador: 2}
9.000000    6   1   1    {Bloco 4; Tempo de início: 9.000000; Número de tarefas escalonadas: 6; pt: 1; pp: 1}
15          2            {Tarefa: 15; Processador: 2}
20          2            {Tarefa: 20; Processador: 2}
4           2            {Tarefa: 4; Processador: 2}
8           2            {Tarefa: 8; Processador: 7}
12          2            {Tarefa: 12; Processador: 2}
16          2            {Tarefa: 16; Processador: 2}
13.000000   6   2   1    {Bloco 5; Tempo de início: 13.000000; Número de tarefas escalonadas: 6; pt: 2; pp: 1}
20          2            {Tarefa: 20; Processador: 2}
16          2            {Tarefa: 16; Processador: 2}
9           2            {Tarefa: 9; Processador: 2}
13          2            {Tarefa: 13; Processador: 2}
17          2            {Tarefa: 17; Processador: 2}
21          2            {Tarefa: 21; Processador: 2}
18.000000   4   0   1    {Bloco 6; Tempo de início: 18.000000; Número de tarefas escalonadas: 4; pt: 0; pp: 1}
21          2            {Tarefa: 21; Processador: 2}
14          2            {Tarefa: 14; Processador: 2}
18          2            {Tarefa: 18; Processador: 2}
19          2            {Tarefa: 19; Processador: 2}
```

Formato do arquivo de saída da divisão dos blocos

Possui a extensão ".blc"

```
6    {Número de blocos: 6}
7    {Número de tarefas do Bloco 0: 7}
0    {Tarefa 0}
1    {Tarefa 1}
2    {Tarefa 2}
3    {Tarefa 3}
4    {Tarefa 4}
5    {Tarefa 5}
6    {Tarefa 6}
6    {Número de tarefas do Bloco 1: 6}
12   {Tarefa 12}
8    {Tarefa 8}
7    {Tarefa 7}
10   {Tarefa 10}
9    {Tarefa 9}
11   {Tarefa 11}
5    {Número de tarefas do Bloco 2: 5}
13   {Tarefa 13}
15   {Tarefa 15}
14   {Tarefa 14}
17   {Tarefa 17}
16   {Tarefa 16}
4    {Número de tarefas do Bloco 3: 4}
18   {Tarefa 18}
19   {Tarefa 19}
20   {Tarefa 20}
21   {Tarefa 21}
3    {Número de tarefas do Bloco 4: 3}
22   {Tarefa 22}
23   {Tarefa 23}
24   {Tarefa 24}
2    {Número de tarefas do Bloco 5: 2}
25   {Tarefa 25}
26   {Tarefa 26}
```

APÊNDICE C

Exemplos de GAD's e arquiteturas

No apêndice C, são mostrados exemplos de GAD's (Grafos Acíclicos Direcionados) e de arquiteturas. Os dados de um GAD são divididos em dois arquivos: o arquivo do grafo contendo as tarefas e suas precedências (com extensão ".msa") e o arquivo de custos associados ao GAD, que guardam os pesos de execução das tarefas. Possui extensão ".cost". Os arquivos de arquitetura possuem as informações sobre o modelo arquitetural da aplicação e possuem a extensão ".comm". Existe também um outro arquivo (com extensão ".comm~dl"), destinado a fornecer a hierarquia dos processadores. Esta informação foi criada em um arquivo separado visando não alterar o formato do arquivo de arquitetura já que este é padrão nas demais áreas do projeto. Necessariamente, esse arquivo auxiliar deverá possuir o mesmo nome do arquivo de arquitetura.

Formato dos arquivos de entrada GAD

Nesse arquivo, é discretizado a quantidade de tarefas que a aplicação possui e, a seguir, a lista de sucessores de cada tarefa.

```
16      {Número de tarefas: 16}
1 4     {Tarefa: 0 - Lista de sucessores: 1, 4}
2 5     {Tarefa: 1 - Lista de sucessores: 2, 5}
3 6     {Tarefa: 2 - Lista de sucessores: 3, 6}
7       {Tarefa: 3 - Lista de sucessores: 7}
5 8     {Tarefa: 4 - Lista de sucessores: 5, 8}
6 9     {Tarefa: 5 - Lista de sucessores: 6, 9}
7 10    {Tarefa: 6 - Lista de sucessores: 7, 10}
11      {Tarefa: 7 - Lista de sucessores: 11}
9 12    {Tarefa: 8 - Lista de sucessores: 9, 12}
10 13   {Tarefa: 9 - Lista de sucessores: 10, 13}
11 14   {Tarefa: 10 - Lista de sucessores: 11, 14}
15      {Tarefa: 11 - Lista de sucessores: 15}
13      {Tarefa: 12 - Lista de sucessores: 13}
14      {Tarefa: 13 - Lista de sucessores: 14}
15      {Tarefa: 14 - Lista de sucessores: 15}
        {Tarefa: 15 - Lista de sucessores: Vazia}
```

Formato do arquivo de entrada de custos do GAD

Nesse arquivo, são descritos o número de tarefas, o custo de execução de cada tarefa e, a seguir, o custo de comunicação entre cada tarefa e seus respectivos sucessores.

```
16    {Número de tarefas: 16}
1     {Custo de execução da tarefa 0: 1}
1     {Custo de execução da tarefa 1: 1}
1     {Custo de execução da tarefa 2: 1}
1     {Custo de execução da tarefa 3: 1}
1     {Custo de execução da tarefa 4: 1}
1     {Custo de execução da tarefa 5: 1}
1     {Custo de execução da tarefa 6: 1}
1     {Custo de execução da tarefa 7: 1}
1     {Custo de execução da tarefa 8: 1}
1     {Custo de execução da tarefa 9: 1}
1     {Custo de execução da tarefa 10: 1}
1     {Custo de execução da tarefa 11: 1}
1     {Custo de execução da tarefa 12: 1}
1     {Custo de execução da tarefa 13: 1}
1     {Custo de execução da tarefa 14: 1}
1     {Custo de execução da tarefa 15: 1}
1 1   {Peso de comunicação entre a tarefa 0 e seu primeiro sucessor(1): 1, segundo sucessor(4): 1}
1 1   {Peso entre 1 e 2: 1, peso entre 1 e 5: 1}
1 1   {Peso entre 2 e 3: 1, peso entre 2 e 6: 1}
1     {Peso entre 3 e 7: 1}
1 1   {Peso entre 4 e 5: 1, peso entre 4 e 8: 1}
1 1   {Peso entre 5 e 6: 1, peso entre 5 e 9: 1}
1 1   {Peso entre 6 e 7: 1, peso entre 6 e 10: 1}
1     {Peso entre 7 e 11: 1}
1 1   {Peso entre 8 e 9: 1, peso entre 8 e 12: 1}
1 1   {Peso entre 9 e 10: 1, peso entre 9 e 13: 1}
1 1   {Peso entre 10 e 11: 1, peso entre 10 e 14: 1}
1     {Peso entre 11 e 15: 1}
1     {Peso entre 12 e 13: 1}
1     {Peso entre 13 e 14: 1}
1     {Peso entre 14 e 15: 1}
```

Note que este arquivo possui uma associação direta com o arquivo de GAD, pois para cada tarefa há uma linha contento os pesos das arestas para seus sucessores, porém a identificação destes sucessores está no arquivo de GAD, na ordem em que foram criados. Por exemplo, a lista de sucessores da tarefa zero são um e quatro e seus pesos de comunicação são um e um, respectivamente.

Formato do arquivo de entrada de arquitetura

Nesse arquivo são descritos diversas informações para modelar a arquitetura da grade. Na primeira linha é definido o número de processadores que compõem a grade. A partir da segunda linha, são descritas as informações de cada máquina, a saber: o fator de heterogeneidade *fh*, o *hostname* da máquina, o overhead de envio Oe e o overhead de recebimento Or. Após, é descrito as latências entre as máquinas através da matriz de comunicação.

12	{Número de processadores: 12}
7 SN00.ic.uff.br 0 0	{Máquina 0, fh: 7, Hostname: SN00.ic.uff.br, Oe: 0, Or: 0}
4 SN01.ic.uff.br 0 0	{Máquina 1, fh: 4, Hostname: SN01.ic.uff.br, Oe: 0, Or: 0}
1 SN02.ic.uff.br 0 0	{Máquina 2, fh: 1, Hostname: SN02.ic.uff.br, Oe: 0, Or: 0}
5 SN03.ic.uff.br 0 0	{Máquina 3, fh: 5, Hostname: SN03.ic.uff.br, Oe: 0, Or: 0}
2 SN04.ic.uff.br 0 0	{Máquina 4, fh: 2, Hostname: SN04.ic.uff.br, Oe: 0, Or: 0}
10 SN05.ic.uff.br 0 0	{Máquina 5, fh: 10, Hostname: SN05.ic.uff.br, Oe: 0, Or: 0}
2 SN06.ic.uff.br 0 0	{Máquina 6, fh: 2, Hostname: SN06.ic.uff.br, Oe: 0, Or: 0}
1 SN07.ic.uff.br 0 0	{Máquina 7, fh: 1, Hostname: SN07.ic.uff.br, Oe: 0, Or: 0}
3 SN08.ic.uff.br 0 0	{Máquina 8, fh: 3, Hostname: SN08.ic.uff.br, Oe: 0, Or: 0}
9 SN09.ic.uff.br 0 0	{Máquina 9, fh: 9, Hostname: SN09.ic.uff.br, Oe: 0, Or: 0}
3 SN10.ic.uff.br 0 0	{Máquina 10, fh: 3, Hostname: SN10.ic.uff.br, Oe: 0, Or: 0}
8 SN11.ic.uff.br 0 0	{Máquina 11, fh: 8, Hostname: SN11.ic.uff.br, Oe: 0, Or: 0}
0 1 1 1 2 3 3 3 2 3 3 3	{Matriz 12x12 descreve a matriz de comunicação.}
1 0 1 1 3 4 4 4 3 4 4 4	{Seja a matriz A_ixj,}
1 1 0 1 3 4 4 4 3 4 4 4	{dado um elemento a_ij pertencente à matriz,}
1 1 1 0 3 4 4 4 3 4 4 4	{este representa a latência de comunicação da máquina i}
2 3 3 3 0 1 1 1 2 3 3 3	{para a máquina j}
3 4 4 4 1 0 1 1 3 4 4 4	
3 4 4 4 1 1 0 1 3 4 4 4	
3 4 4 4 1 1 1 0 3 4 4 4	
2 3 3 3 2 3 3 3 0 1 1 1	
3 4 4 4 3 4 4 4 1 0 1 1	
3 4 4 4 3 4 4 4 1 1 0 1	
3 4 4 4 3 4 4 4 1 1 1 0	

Formato do arquivo de entrada da hierarquia da arquitetura

É descrito neste arquivo a quantidade de máquinas locais e na sequência, os cargos hierárquicos que cada máquina desempenhará.

```
3   {Número de Máquinas Locais: 3}
1   {Máquina: 0, Bit de Local: 1}
0   {Máquina: 1, Bit de Local: 0}
0   {Máquina: 2, Bit de Local: 0}
0   {Máquina: 3, Bit de Local: 0}
1   {Máquina: 4, Bit de Local: 1}
0   {Máquina: 5, Bit de Local: 0}
0   {Máquina: 6, Bit de Local: 0}
0   {Máquina: 7, Bit de Local: 0}
1   {Máquina: 8, Bit de Local: 1}
0   {Máquina: 9, Bit de Local: 0}
0   {Máquina: 10, Bit de Local: 0}
0   {Máquina: 11, Bit de Local: 0}
```

Uma máquina local é definida como a máquina, situada dentro de um site, responsável pela comunicação com a máquina global, possuindo uma hierarquia acima das demais máquinas deste site e centralizando todas as comunicações entre as máquinas pertencentes a este site. Para cada site haverá somente uma máquina local. O Bit de Local indica se uma máquina será local em um site. Caso o Bit de Local seja um, para determinada máquina, significa que esta máquina será a máquina local de um site e todas as máquinas subsequentes a esta que possuirem Bit de Local igual a zero pertencerão a este mesmo site.

APÊNDICE D

Publicação

Durante o período de Iniciação Científica, foi publicado um artigo relacionado a este trabalho de projeto final de curso. Ele foi submetido e aprovado como resumo estendido de iniciação científica no evento WSCAD, evento este ocorrido no Rio de Janeiro de 24 à 27 de Outubro de 2005.

- Figueira, C.; Boeres, C. e Kraus, F. **Ambiente de Simulação de Escalonamento em Sistemas Distribuídos Hierárquicos**. In *6º Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2005)*, Rio de Janeiro, RJ, October 24-27, 2005.

Referências

[1] M. Baker, R. Buyya, and D. Laforenza. Grids and grid techinologies for wide-area distributed computing. *Software: Practice and Experience*, 32(15):1437–1466, 2002.

[2] C. Boeres, A. A. Fonseca, H. A. Mendes, L. T. Menezes, N. T. Moura, J. A. Silva, B. A. Vianna, and V. E. F. Rebello. An EasyGrid portal for scheduling system-aware applications on computational grids. *Concurrency and Computation: Pratice and Experience*, 2005. (to appear).

[3] Cristina Boeres and Vinod E. F. Rebello. Easygrid: Um framework para a habilitação automática de aplicações mpi em grids computacionais (e a iniciativa gridrio). *Anais do I Workshop em Grade Computacional e Aplicações, Programa de Verão do 2003 do LNCC*, 2003.

[4] Cristina Boeres and Vinod E. F. Rebello. EasyGrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Pratice and Experience*, 16(5):425–432, April 2004.

[5] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems.* Prentice Hall, 1999.

[6] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience*, 14, Nov.-Dec. 2002.

[7] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001.

[8] J. V. Filho. Uma estratégia de aglomeração de tarefas par sistemas de processadores heterogêneos. *Tese de Mestrado, Pós Graduação em Computação, Universidade Federal Fluminense*, 2004.

[9] Ariel Fonseca and Bruno Vianna. Um ambiente para desenvolvimento e avaliação de algoritmos de escalonamento para grades tradicionais. *Monografia Final de Curso, Graduação em Ciência da Computação*, Agosto 2004.

[10] I. Foster and C. Kesselman. The globus project: A status report. *IPPS/SPDP'98 Heterogenous Computing Workshop*, pages 4–18, 1998.

[11] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1999.

[12] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[13] J. J. Hwang, Y.C. Chow, F. D. Anger, and C. Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAMJ Comp.*, 18:244–257, April 1989.

[14] Y. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. *12th. International Parallel Processing Symposium*, page 531, 1998.

[15] Y. K. Kwok and I. Ahmad. Dynamic critical-path scheduling: an effective technique for allocating tasks graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, May 1996.

[16] G. Laszewski. The paralellization of a weather prediction model.

[17] G. Laszewski, M. Westbrook, I. Foster, and E. Westbrook. Using computational grid capabilities to enhance the capability of an x-ray source for structural biology. *Springer Netherlands*, 3(3):187–199, September 2000.

[18] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the simgrid simulation framework. *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003.

[19] Alexandre Lima. Escalonamento híbrido de tarefas: Integração de heurísticas estáticas e dinâmicas. *Dissertação de Mestrado, Pós Graduação em Computação, Universidade Federal Fluminense*, Novembro 2004.

[20] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. *In The Proceedings of the Annual International Symposium on Computer Architecture*, pages 85–97, June 1997.

[21] Message Passing Forum. MPI: A Message Passing Interface. Technical report, University of Tennessee, 1995.

[22] A. P. Nascimento, A. C. Sena, J. A. Silva, D. Q. C. Vianna, C. Boeres, and V. E. F. Rebello. Managing the execution of large scale MPI applications on computational grids. In *Proceedings of the 17th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2005)*. IEEE Computer Society Press, October 2005.

[23] Inc Red Hat. Fedora core 2 release notes. site, http://www.gnu.org/licenses/fdl.html, 2004.

[24] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, K. Taura X. Zhang, and A. Chien. The microgrid: a scientific tool for modeling computational grids. *Proceedings of SC2000*, 2000.

[25] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algoritms for heterogeneous processors. *In Proc. Heterogeneous Computing Workshop*, 1999.

[26] Luiz Toscano and Nilmax Moura. Um ambiente de análise para escalonamento híbrido de tarefas em sistemas distribuídos. *Monografia Final de Curso, Graduação em Ciência da Computação*, Janeiro 2005.

[27] J. D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393, 1975.

[28] Min-You Wu, Wei Shu, and Hong Zhang. Segmented min-min: A static mappin algorithm for meta-tasks on heterogeneous computing systems. 2000.