UNIVERSIDADE FEDERAL FLUMINENSE

Henrique Bueno Rodrigues
Hildebrando Trannin

Avaliação de Heurísticas de Escalonamento de Tarefas Usando Grades Computacionais

Niterói

Agosto de 2006

Resumo

A computação distribuída e paralela oferece a oportunidade de obter melhoras significativas no desempenho quando aplicações tiram proveito de mais de um recurso computacional. Enquanto um *cluster*, tipicamente, é um conjunto de computadores idênticos conectados em uma rede local, uma grade computacional, uma nova estrutura sendo oferecida, é uma aglomeração de computadores diferentes geograficamente distribuídos e interconectados pela Internet. Como uma grade pode ser composta por vários *clusters*, ela possui uma quantidade consideravelmente maior de computadores do que os disponíveis a um usuário na sua instituição.

As maiores restrições para uma ampla utilização de grades por aplicações que demandam alto poder computacional são a dificuldade em coordenar o compartilhamento dos recursos heterogêneos oferecidos pelas diversas máquinas que compõem a grade e o problema de como disponibilizar este ambiente de execução de forma transparente ao usuário. Dessa forma, a atividade de distribuir as tarefas que compõem uma aplicação ao longo dos processadores disponíveis, de modo a acelerar a execução como um todo e ser transparente para o usuário, se torna uma importante necessidade para o sucesso desta forma de computação.

O Portal EasyGrid tem como um dos objetivos oferecer um ambiente amigável e simples para a utilização de computação em grades para a resolução de problemas. De maneira particular, o Portal auxilia o desenvolvimento e avaliação de algoritmos para o problema de escalonamento de tarefas. Neste estudo de caso uma avaliação tipicamente envolve a confecção, execução e coleção e análise dos resultados de milhares de experimentos. Esta projeto final apresenta o Portal EasyGrid com sua nova estratégia de execução em grades computacionais e uma nova estratégia estática de escalonamento de tarefas desenvolvida com base nos resultados de experimentos que utilizaram essa estratégia. Mostrando que este processo de avaliação pode ser realizado de maneira mais rápida e fácil.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nunca foi tão rápida a forma com a qual o homem desenvolveu novas tecnologias e realizou descobertas em diferentes ramos. Este avanço acelerado se deve à dedicação de pesquisadores e à evolução das ferramentas que oferecem suporte a novos estudos. Uma ferramenta que se tornou indispensável ao mundo moderno e cuja evolução abriu novos horizontes para diversas pesquisas foi o computador. O computador ofereceu uma capacidade de processamento de dados nunca imaginada antes de sua concepção. Por outro lado, para aplicações que continuam a evoluir e exigem elevado poder computacional, por exemplo simulações climáticas, pesquisas astronômicas, estudo do genoma, entre outras, o uso de computadores convencionais não está mais oferecendo um tempo de resposta viável. Estas aplicações geralmente realizam um grande volume de operações que sobrecarregam um único computador. Dessa forma, os resultados são obtidos após os prazos definidos, deixando de ser funcionais.

Com o advento das redes de interconexão tornou-se possível a aglomeração de vários recursos com o objetivo de aumentar o poder computacional. Dessa forma, surgiram os sistemas paralelos e mais tarde os *clusters* que são diversos processadores e memória, geralmente com configurações semelhantes, interconectados entre si. Hoje, a maioria dos sistemas de alto desempenho são *clusters*, também conhecideos como supercomputadores, compostos de centenas de processadores. O número médio de processadores dos dez supercomputadores mais rápidos é 25488 [500 2004]. O mais rápido, BlueGene da IBM, pode atingir 280 TFlops, e o último da lista pode atingir 2TFlops. Entretanto, um supercomputador requer um custo muito elevado para desenvolvimento e manutenção e por isso poucas instituições têm condições de obter um.

Devido a existência de redes de longa distância com alta velocidade e a crescente demanda por poder computacional foi proposta a utilização dos diversos recursos já interconecados, pela Internet por exemplo, como um único recurso computacional de alto desempenho mas de baixo custo e acessível. Assim sendo, recursos geograficamente distribuídos poderiam ser utilizados

para executar as tarefas de uma mesma aplicação e usuários não precisariam estar localizados próximo às fontes de processamento. A este sistema foi dado o nome de *grid* (grade) computacional. Por utilizar recursos já existentes, uma grade computacional é criada com um custo muito baixo se comparado com supercomputadores. Porém, apesar de oferecer um elevado poder computacional e baixo custo, a utilização de grades ainda enfrenta grandes problemas, como a administração de uma grande quantidade de recursos heterogêneos e altamente dinâmicos. Essas características restringem o acesso ao *grid* aos profissionais com conhecimento nesta área, dificultando que pesquisadores de outras áreas desenvolvam suas aplicações e tirem proveito dos recursos distribuídos.

Assim, é preciso que exista uma camada de software (similar a um sistema operacional) entre os usuários e a infraestrutura distribuída que simplifique e facilite o acesso aos recursos. Esta interface seria responsável, por exemplo, por alocar as tarefas da aplicação nos processadores disponíveis, administrar a comunicação entre as tarefas, redistribuir as tarefas entre os recursos ao longo da execução, reiniciar a execução de uma tarefa caso ocorra falha no processador alocado, transferência de arquivos, segurança, entre outras atividades. Com o intuito de suprir tal necessidade numerosos grupos de pesquisa estão desenvolvendo ferramentas e *middlewares* para sistemas, como o Globus Toolkit [Foster e Kesselman 1997], e para aplicações, como o EasyGrid AMS [Nascimento et al. 2005].

Um dos maiores problemas para uma execução rápida e eficiente de aplicações em uma grade é o escalonamento das tarefas da aplicação, que consiste na distribuição das tarefas pelos processadores a fim de obter o menor tempo de execução (*makespan*) para a aplicação. Este problema é considerado NP-Completo, assim heurísticas são propostas com o objetivo de obter resultados próximos do ótimo. Apesar da existência de bons algoritmos para o problema de escalonamento de tarefas para *clusters*, poucos existem para grades e a grande maioria não mantém a qualidade dos resultados quando ocorre a variação das configurações da arquitetura da grade e das classes da aplicação, e o compartilhamento dos recursos. Entretanto, a utilização de heurísticas não garante a qualidade dos resultados, não sendo possível definir uma heurística que sempre gere bons resultados para todos os tipos de aplicação.

1.1 Contribuição

Do ponto de vista do desenvolvedor, a confecção de novos algoritmos de escalonamento envolve tipicamente a avaliação de heurísticas existentes com o objetivo de encontrar suas principais características. Esta avaliação envolve o projeto e a execução dos experimentos e a análise

dos resultados. Sendo assim, o Portal EasyGrid [Boeres et al. 2006] foi originalmente projetado para facilitar o desenvolvimento e a avaliação de heurísticas de escalonamento através da simplicação da execução dos experimentos.

Este trabalho final de curso incluiu no Portal uma nova metodologia de execução distribuída desses experimentos de modo a torná-la menos árdua e trabalhosa para o usuário e executá-la mais rapidamente utilizando o poder computacional da própria grade. Além disso, desenvolveu uma nova estratégia de escalonamento a partir da análise dos resultados dos experimentos executados utilizando essa nova metodologia. Embora neste trabalho o foco está na avaliação de heurísticas para grades computacionais, o Portal pode ser utilizado, sem nenhuma modificação, para avaliar heurísticas para outras arquiteturas.

1.2 Organização

Esta dissertação está organizada em 7 capítulos. O Capítulo 2 apresenta o problema de escalonamento de tarefas, algumas definições referentes ao tema em questão e as descrições dos modelos da aplicação, de comunicação e da arquitetura. No Capítulo 3 são apresentadas as heurísticas estáticas estudadas e as diversas prioridades utilizadas nestas e a definição de heurísticas dinâmicas e híbridas. O Capítulo 4 apresenta o Portal EasyGrid como uma ferramenta de avaliação das heurísticas existentes, os problemas enfrentados para a execução destes experimentos e a solução proposta para acelerar e simplificar esta avaliação. O Capítulo 5 descreve os experimentos realizados, as medidas de performance e uma análise detalhada dos resultados. O Capítulo 6 apresenta uma nova estratégia de escalonamento de tarefas com base nos resultados do Capítulo 5. No Capítulo 7, por fim, são apresentados conclusões e trabalhos futuros.

Capítulo 2

O Problema de Escalonamento de Tarefas

Neste capítulo é apresentado o problema de escalonamento de tarefas que consiste na alocação das tarefas de uma aplicação paralela em um ambiente distribuído. Também serão apresentados modelos tipicamente utilizados para definição da aplicação e do ambiente em questão.

2.1 Introdução

O problema de escalonamento de tarefas de uma aplicação paralela, de forma a minimizar o seu tempo de execução, em um conjunto de recursos distribuídos é considerada NP-completa. Mesmo após diversas simplificações (processadores homogêneos, ausência de comunicação entre as tarefas) o problema ainda se mostrou NP-completo [Coffman Jr. 1976]. Devido a esta NP-completude do problema de escalonamento, é necessário empregar heurísticas para obter escalonamentos quase ótimos em tempos razoáveis [Khan et al. 1994, Kwok e Ahmad 1999b]. No entanto, é extremamente difícil criar heurísticas eficientes, dado a variedade de características, tanto arquiteturais (por ex. processadors heterogêneos e contenção da rede) quanto relacionadas com a própria aplicação paralela (por ex. dependência entre as tarefas e granularidade), que influenciam bastante no projeto de algoritmos de escalonamento. As características da arquitetura e da aplicação que mais influenciam no tempo total de execução (*makespan*) são representadas, respectivamente, pelos modelos da arquitetura e da aplicação. Estes modelos são melhor detalhados e exemplificados nas subseções abaixo.

2.2 Modelo da Aplicação

Uma aplicação paralela é comumente representada 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 (nós) que representam as tarefas e $E = \{(v_i, v_j) | v_i, v_j \in V\}$ é o conjunto de arcos (arestas) que

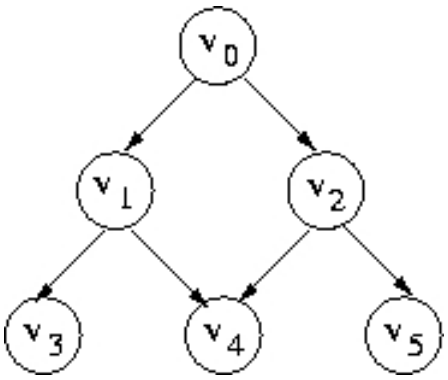


Figura 2.1: Grafo OutTree

representam as relações de precedência entre as tarefas. Denota-se por $\varepsilon(v_i)$, o peso de execução de uma tarefa $v_i \in V$, que representa a quantidade de computação (instruções) necessárias à execução, e por $\omega(v_i, v_j)$, o peso de dados associado ao arco $(v_i, v_j) \in E$, que representa 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 $succ(v_i) = \{v_j | (v_i, v_j) \in E\}$.

2.3 Benchmark GADs

Com o intuito de comparar heurísticas, os pesquisadores normalmente utilizam conjuntos de *benchmark* GADs encontrados na literatura. Exemplos podem ser encontrados em [Kwok e Ahmad 1999a] e [Boeres e Rebello 2002]. Esses conjuntos contém grafos que representam uma grande gama de aplicações. As classes de aplicações utilizadas neste trabalho são descritas abaixo.

2.3.1 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. Um exemplo é apresentado na Figura 2.1.

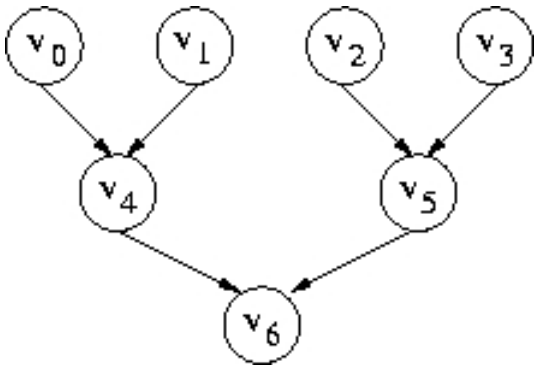


Figura 2.2: Grafo InTree

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

Nestes tipos de árvores, cada nó tem dois predecessores imediatos, com exceção dos nós origens, e apenas um sucessor, com exceção do nó 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. É possível observar um exemplo na Figura 2.2.

2.3.3 Irregulares (*Ir* ou *In*)

Estes grafos representam aplicações irregulares, ou seja, que não possuem um padrão em sua formação. Como exemplo tem-se o Ir41 (figura 2.3) que representa uma aplicação de dinâmica molecular.

2.3.4 Diamantes (*Di*)

Um grafo diamante $G = (V, E)$ é um grafo $n \times n$ isomorfo a uma grade planar como pode ser visto na Figura 2.4. Cada vértice v_i coincide com as coordenadas cartesianas (x, y), onde $x = 1, ..., n$, e $y = 1, ..., n$. O grafo diamante é um bom representante de aplicações como decomposição LU, multiplicação de matrizes e systolic arrays.

2.3.5 Randômicos (*Ran*)

Estes grafos são gerados aleatoriamente e têm como objetivo representar uma aplicação paralela irregular (sem um padrão de formação). São geralmente utilizados para avaliar o desempenho de uma heurística de escalonamento sobre aplicações com estruturas irregulares. Podemos observar um exemplo na figura 2.5.

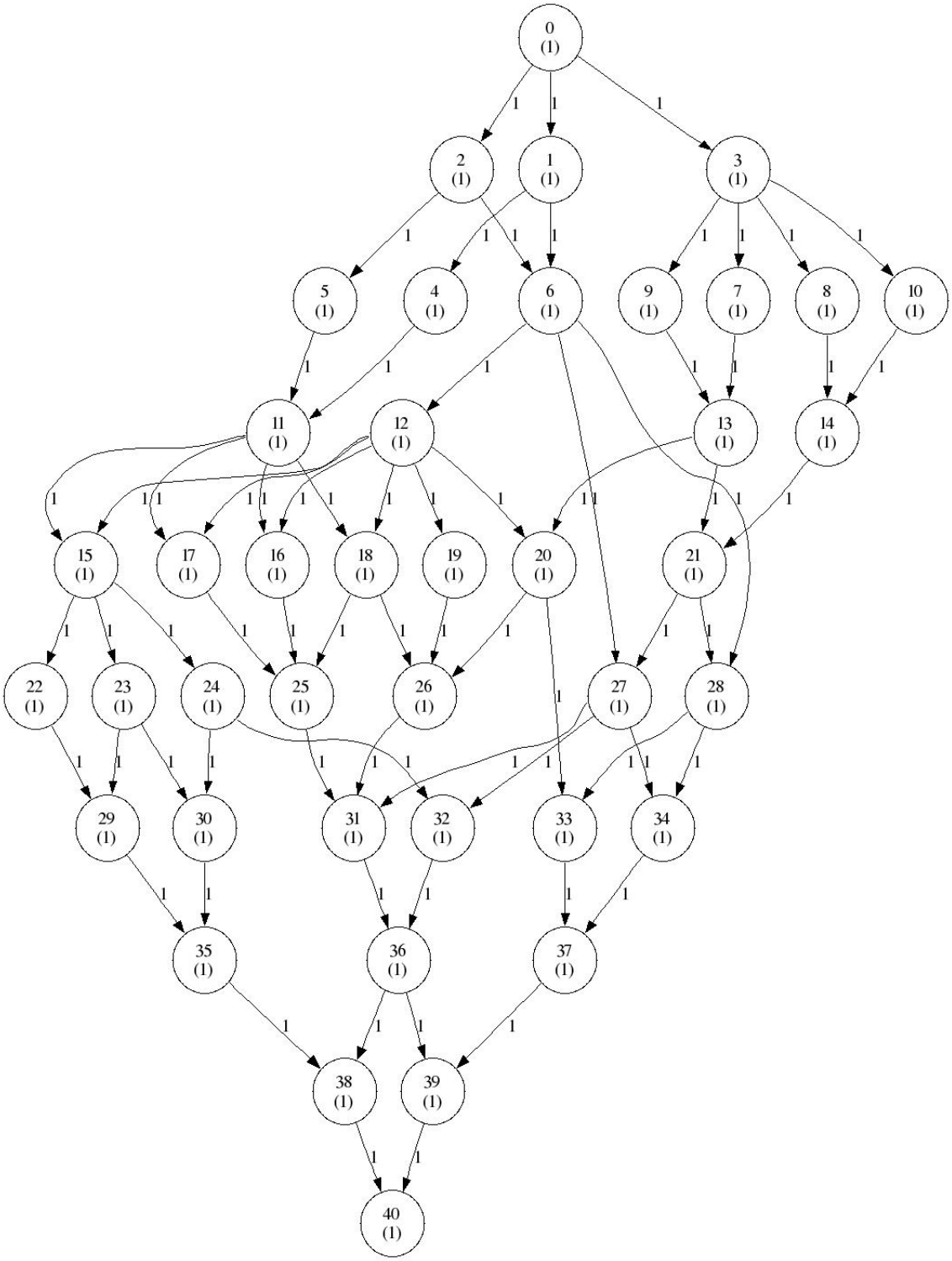


Figura 2.3: Grafo Ir

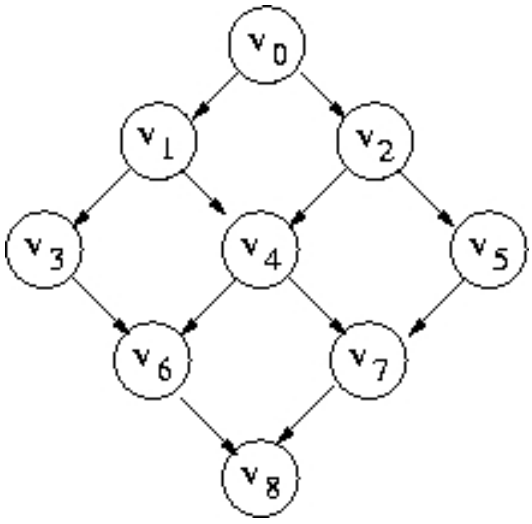


Figura 2.4: Grafo Diamante

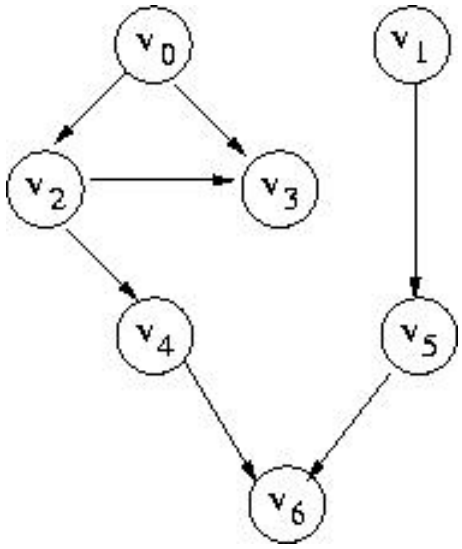


Figura 2.5: Grafo Randômico

2.3.6 *Kwok's Peer Set Graphs* (KPSGs)

Os KPSGs [Kwok e Ahmad 1999b] são 11 exemplos de grafos documentados na literatura, utilizados por diversos 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).

1. *Ahmad and Kwok*, grafo com 13 nós (KPSG1);
2. *Al-Maasarani*, grafo com 16 nós (KPSG2);
3. *Al-Mouhamed*, grafo com 17 nós (KPSG3);
4. *Shirazi et al.*, grafo com 11 nós (KPSG4);
5. *Collin and Chretienne*, grafo com 9 nós (KPSG5);
6. *Gerasoulis and Yang*, grafo com 7 nós (KPSG6);
7. *Kruatrachue and Lewis*, grafo com 15 nós (KPSG7);
8. *McCreary and Gill*, grafo com 9 nós (KPSG8);
9. *Chung and Ranka*, grafo com 11 nós (KPSG9);
10. *Wu and Gajski*, grafo com 18 nós (KPSG10);
11. *Yang and Gerasoulis*, grafo com 7 nós (KPSG11).

2.4 Modelo da Arquitetura

O modelo arquitetural define as características do sistema consideradas relevantes ao problema de escalonamento. Tipicamente, os modelos utilizados supõem que o ambiente computacional consiste de unidades de processamento distribuídas que se comunicam via troca de mensagens através de uma rede de interconexão. Dessa forma, podemos representar a arquitetura como um grafo não direcionado $H = (P, C)$, onde P é o conjunto de vértices que representam os processadores, e C o conjunto de arestas que representam os canais de comunicação que interligam os processadores adjacentes.

Dois aspectos da arquitetura que merecem destaque são o modelo de computação e o modelo de comunicação. Estes são descritos nas seções seguintes.

2.4.1 Computação

Um modelo de computação define o conjunto, geralmente limitado, de elementos de processamento, onde o poder computacional de cada elemento pode ou não ser diferente (processadores heterogêneos ou homogêneos). Vários modelos de computação paralela, desde os mais teóricos e abstratos [Aggarwal et al. 1989] até os mais realísticos e práticos [Tam e Wang 1999], têm sido propostos. Os modelos mais teóricos têm relevância a medida que a abstração permite uma análise da complexidade dos algoritmos paralelos. Já nos modelos mais realísticos, grande parte da abstração cede lugar a detalhes específicos da máquina na qual a aplicação paralela será executada.

Em uma grade computacional existem diferentes elementos de processamento e cada um destes é diferenciado pelo seu fator de heterogeneidade fh que representa o poder computacional da demanda de processamento. Como o tempo de execução de uma tarefa é calculado em função deste fator de heterogeneidade, ele varia de um computador para outro. Formalmente, este tempo é definido como $\varepsilon(v_i) \times fh(p_j)$, sendo v_i uma tarefa executada em um processador p_j e $\varepsilon(v_i)$ o peso da tarefa v_i.

2.4.2 Comunicação

Um modelo de comunicação tenta capturar os custos de comunicação entre os processadores e a topologia das interconexões. A maioria dos trabalhos de escalonamento considera que os processadores são totalmente conectados [Kwok e Ahmad 1999b]. Atualmente o modelo padrão de comunicação utilizado em escalonamento de tarefas é o modelo de latência, onde a latência, ou atraso no tempo de transmissão, é o único parâmetro na comunicação considerado significativo. Outra característica assumida neste modelo é que um processador não gasta tempo preparando o envio ou recebimento de mensagens, e portanto permite a sobreposição de comunicação e computação.

Além desse, existem outros modelos de comunicação como BSP, Postal e LogP. Este último é também bastante utilizado por pesquisadores pois é mais realista que o modelo de latência e possui como parâmetros a latência (como descrito anteriormente); as sobrecargas de envio/recebimento (tempo que um processador gasta para preparar uma mensagem); o gap (intervalo entre envio/recebimento consecutivos) e a quantidade de processadores disponíveis. Em alguns modelos, cada um desses parâmetros são considerados funções do tamanho da mensagem.

Este trabalho utiliza o modelo de latência e por isso os demais modelos de comunicação

existentes não serão detalhados. O modelo latência foi o escolhido por ser, por enquanto, o que parece mais apropriado à grades computacionais devido à grande distância entre seus recursos e também por existir, na literatura, uma grande quantidade de algoritmos de escalonamento baseados neste padrão, o que viabiliza estudos comparativos. No caso de grades a latência é considerada entre pares de processadores comunicantes.

Capítulo 3

Heurísticas de Escalonamento

Heurísticas de escalonamento são propostas com o objetivo principal de gerar bons escalonamentos. Podemos classificar as heurísticas de escalonamento em estáticas e dinâmicas dependendo do momento em que as decisões são tomadas. Nas heurísticas estáticas, todas as decisões são tomadas antes de começar a execução das tarefas. Nas heurísticas dinâmicas, as decisões são tomadas ao longo da execução. Um terceiro tipo são as heurísticas híbridas que mesclam características estáticas e dinâmicas. O foco deste trabalho está na avaliação de heurísticas estáticas e o trabalho [Figueira e Kraus 2006] aprofunda os estudos das heurísticas híbridas.

3.1 Heurísticas Estáticas

Neste tipo de heurística, as tarefas de uma aplicação são alocadas nos recursos disponíveis (processadores) em tempo de compilação. Dessa forma, a lista de tarefas relacionadas a cada processador já está definida antes da execução da aplicação paralela. Cada tarefa irá iniciar e concluir sua execução no mesmo processador em que foi escalonada inicialmente.

Pelas heurísticas estáticas gerarem o escalonamento antes da execução da aplicação elas são incapazes de se adaptarem às alterações do ambiente. Entretanto, esse modelo de escalonamento não gera *overheads* em tempo de execução, já que o escalonamento é previamente realizado. Entre as principais heurísticas estáticas existentes na literatura, destacam-se: *List Scheduling*, HEFT, DCP e ETF.

3.1.1 List Scheduling

List Scheduling [Kwok e Ahmad 1999b] é um método heurístico para escalonamento de uma aplicação paralela representada por um grafo acíclico direcionado (GAD) G em um con-

junto de processadores através da associação de prioridades às tarefas da aplicação.

Basicamente esta metodologia segue a seguinte regra: uma tarefa livre deve ser alocada em um processador ocioso tal que seu tempo de início seja o mais cedo possível. Neste contexto, define-se:

- uma tarefa é dita livre quando todos os seus predecessores imediatos já foram escalonados, ou seja, v_i é livre se para todo $v_j \in$ pred(v_i), proc(v_j) $\neq \emptyset$;
- um processador está ocioso em um determinado instante t_k se não existe nenhuma tarefa sendo executada nele naquele instante.

A principal diferença entre vários algoritmos do tipo *list scheduling* propostos na literatura, é a prioridade usada na seleção da próxima tarefa livre a ser escalonada. A estrutura básica do *list scheduling* é definida a seguir:

Algoritmo 1 :*List Scheduling*

```
1  Associe prioridades às tarefas da aplicação;
2  Enquanto existirem tarefas não escalonadas faça
3      selecione a tarefa livre com maior prioridade. Seja v_i esta tarefa;
4      selecione um processador ocioso tal que v_i possa iniciar o mais cedo possível. Seja p_j
       este processador;
5      escalone v_i no processador p_j;
6      determine as novas tarefas livres;
   Fim Enquanto
```

As prioridades associadas às tarefas podem ser características do próprio grafo de entrada G = (V, E). As prioridades utilizadas pelo algoritmo *List Scheduling* neste trabalho são descritas a seguir:

- Nível

  O nível de um nó n é o tamanho do maior caminho desde o nó n, incluindo o custo de execução de n, até um nó de saída. O cálculo deste caminho consiste na soma de todos os custos de computação das tarefas v_i pertencentes a esse caminho, multiplicado pelo fator de heterogeneidade médio das máquinas, e todas as comunicações, multiplicado pela latência, entre duas tarefas v_i e v_{i+1} adjacentes pertencentes a esse mesmo caminho.

- Conível

O conível de um nó n é o custo do caminho mais longo desde um nó de entrada até n, não considerando o custo de execução do próprio nó n. Sendo que, o custo do caminho consiste na soma de todos os custos de computação das tarefas v_i multiplicado pelo fator de heterogeneidade médio das máquinas, e somando todos os custos de comunicação das arestas que ligam duas tarefas adjacentes ao longo do caminho. Para esta prioridade, a tarefa considerada com maior prioridade é aquela que possuir menor valor para seu conível.

- Nível mais Conível

  Seja v_i uma tarefa de uma dada aplicação $G(V, E)$. O cálculo desta prioridade para v_i é dado pela soma do valor do Nível de v_i e do valor do Conível de v_i. Esta prioridade também é conhecida como Caminho Crítico, pois ao somarmos o Nível e o Conível para v_i obtemos o caminho mais longo da tarefa inicial à final passando por v_i.

- LBNível

  Esta prioridade também é conhecida como Nível Estático. Seu valor é calculado de forma semelhante ao cálculo da prioridade Nível, porém neste os valores dos custos de comunicação entre as tarefas são desconsiderados.

- ALAP (*As Late As Possible*)

  O ALAP de um nó é a medida de quanto tempo se pode atrasar o início da execução de uma tarefa, sem que se aumente o tempo total do escalonamento. Podemos obter esta medida mediante a realização da subtração do valor do caminho crítico do GAD pelo atributo nível da tarefa, ou seja, $ALAP(v_i) = caminho\_critico(G) - nivel(v_i)$, onde G é um GAD e $caminho\_critico(G)$ é a máxima soma dos custos dos nós e arestas que formam um caminho entre um nó de entrada e um de saída de G. Note que as tarefas do caminho crítico possuem o valor da prioridade ALAP igual ao valor da prioridade conível.

- ALAP menos maior aresta incidente

  Seja v_i uma tarefa de uma dada aplicação $G(V, E)$. O cálculo desta prioridade para v_i é dado pela subtração do valor da aresta de maior quantidade de dados transmitidos incidente à v_i, do valor da prioridade ALAP de v_i.

- Largura

  Seja v_i uma tarefa de uma dada aplicação $G(V, E)$. O cálculo desta prioridade para v_i é dado pela diferença do valor do Nível de v_i pelo valor do seu Conível.

- Número de sucessores

  Seja v_i uma tarefa de uma dada aplicação $G(V, E)$. O cálculo desta prioridade para v_i é dado pela quantidade de tarefas v_j tal que (v_i,v_j) $\in$ E.

3.1.2 Heterogeneous Earliest-Finish-Time (*HEFT*)

O algoritmo *Heterogeneous Earliest Finish Time* (HEFT) [Topcuoglu et al. 2002] também é uma heurística do tipo *List Scheduling* e considera um número limitado de recursos heterogêneos. O algoritmo HEFT primeiramente ordena todas as tarefas de acordo com a prioridade nível em uma lista. Posteriormente, seguindo a ordem determinada nesta lista, o escalonador vai associando a cada tarefa um processador, de forma que seja minimizado o tempo de fim da tarefa escolhida. O algoritmo HEFT, no entanto, realiza o procedimento de inserção da tarefa em espaços de tempo ociosos nos processadores.

3.1.3 Dynamic Critical Path (*DCP*)

O algoritmo DCP (*Dynamic Critical Path*) [Kwok e Ahmad 1996] foi projetado baseando-se em um atributo que define a mobilidade da tarefa. O algoritmo DCP usa a estratégia *look-ahead* para encontrar o melhor *cluster* de tarefas para um dado vértice. Por isso, para computar o valor de $T_S(n_i)$ (tempo de início de uma tarefa n_i) sobre um processador, o algoritmo DCP também computa o valor de $T_S(n_e)$ (tempo de início de n_e sobre o mesmo processador), onde n_e é o filho de n_i que tem o maior custo de computação e é também chamado de filho crítico de n_i. O algoritmo DCP escalona uma tarefa n_i no processador que fornecer o valor mínimo da soma desses dois atributos. Esta estratégia de *look-ahead* pode evitar que a tarefa seja escalonada em um *cluster* que não tem espaço para acomodar o sucessor crítico, impondo uma comunicação pesada entre eles. O algoritmo DCP examina todos os *clusters* existentes para decidir em qual deles irá escalonar um nó, enquanto outros algoritmos somente procuram um *cluster* até encontrar um que tenha espaço ocioso para armazenar a tarefa da vez.

3.1.4 Earliest Time First (*ETF*)

O algoritmo ETF (*Earliest Time First*) [Hwang et al. 1989] é um algoritmo do tipo *List Scheduling* projetado para um número finito de processadores homogêneos. O ETF determina a cada iteração o tempo de início mais cedo de todas as tarefas livres nos processadores ociosos no momento corrente. A tarefa selecionada é a que tem o menor tempo de início e é alocada no respectivo processador. Se a tarefa escolhida tem possibilidade de iniciar mais cedo ainda

em um processador que não está ocioso no momento, seu escalonamento só é efetivamente realizado em iterações posteriores.

Quando o tempo de início mais cedo de duas ou mais tarefas forem iguais, o algoritmo escolhe a tarefa com o maior nível estático. Portanto, uma tarefa com um maior nível estático não necessariamente consegue ser escalonada primeiro, porque o algoritmo considera mais prioritárias as tarefas com o menor tempo de início.

3.2 Heurísticas Dinâmicas

O escalonamento dinâmico consiste na alocação das tarefas de uma aplicação paralela nos processadores disponíveis em tempo de execução. Ao executar uma aplicação e escaloná-la dinamicamente, o escalonador dinâmico é executado de tempos em tempos para associar um subconjunto de tarefas da aplicação aos processadores disponíveis. Um dos aspectos mais importantes de um escalonador dinâmico é a freqüência com que este deve ser executado.

Chamadas freqüentes do escalonador podem acarretar em um escalonamento ineficiente quando a aplicação é composta por um grande número de tarefas, já que irá gerar um custo adicional para a execução. Entretanto, se a freqüência de chamadas do escalonador for baixa, o escalonador dinâmico pode não ser capaz de ajustar o escalonamento de forma a reduzir o tempo de execução da aplicação, já que mudanças no ambiente podem ocorrer no intervalo entre duas chamadas ao escalonador. O intervalo entre as chamados do escalonador deve estar relacionado com a instabilidade do ambiente computacional.

Dessa forma, apesar de adicionar um custo à execução da aplicação, heurísticas dinâmicas podem trazer resultados bem melhores que àqueles gerados por escalonamentos estáticos quando executadas em ambientes com recursos altamente instáveis. Estas heurísticas também são, tipicamente, baseadas no *List Scheduling* e alguns exemplos são os algoritmos Min-Min e a sua variação Segmented Min-Min [Wu et al. 2000].

3.3 Heurísticas Híbridas

Heurísticas híbridas tem por objetivo unir as melhores características das heurísticas estáticas e dinâmicas. De posse de um escalonamento estático, gerado inicialmente com base nas informações estáticas do ambiente computacional (velocidade dos processadores, custo de comunicação entre os processadores), um escalonamento em tempo de execução será realizado por um escalonador dinâmico. O escalonador dinâmico faz uso do escalonamento estático para

ter uma visão completa da aplicação e poder empregar heurísticas mais sofisticadas.

A combinação das heurísticas estática e dinâmica para a construção de um escalonador híbrido tem como objetivo reduzir o overhead gerado em tempo de execução pelo escalonador dinâmico, oferecendo a ele um escalonamento estático inicial. Assim, quanto mais características do ambiente o escalonador estático considerar, menor poderá ser o trabalho realizado pelo escalonador dinâmico para adaptar as tarefas de uma aplicação às variações do ambiente computacional.

Capítulo 4

Ambiente de Avaliação

Comprovar o grau de complexidade de uma heurística é freqüentemente difícil e geralmente são utilizados os limites do pior caso possível. Para definir a qualidade de uma heurística é tipicamente empregado uma avaliação comparativa. Para uma avaliação completa de heurísticas de escalonamento de tarefas, tanto novas quanto existentes, é imprescindível a realização de uma grande quantidade de experimentos. Os objetivos destes experimentos são avaliar a versatilidade e robustez das heurísticas de escalonamento, em relação às diferentes caraterísticas das aplicações e arquiteturas, e definir o quanto uma heurística é boa em comparação com outras para determinadas classes de aplicações.

Do ponto de vista do projetista, a confecção e execução dos experimentos e a coleção e análise dos resultados é uma tarefa trabalhosa, árdua e demorada. Porém, devido a ausência de relações de dependência entre os experimentos, estes podem ser agrupados como tarefas de uma aplicação paralela e esta avaliação pode ser feita através de uma aplicação *bag-of-tasks* do tipo *parameter sweep* [Abramson et al. 2000].

O EasyGrid Scheduling Portal [Boeres et al. 2006, Fonseca e Vianna 2004], ferramenta *desktop* previamente desenvolvida para o projeto e avaliação de novas heurísticas de escalonamento, foi modificado para criar e executar, em uma grade computacional, automaticamente uma única aplicação que realiza todos os experimentos necessários na avaliação das caraterísticas de interesse. Esta aplicação, implementada em C e MPI [Message Passing Forum 1995], incorpora o *middleware* EasyGrid [Nascimento et al. 2005] para uma execução rápida e robusta, e retorna os resultados para análise no Portal.

4.1 EasyGrid Scheduling Portal

Devido à necessidade de um maior poder computacional para resolver problemas mais complexos, as grades computacionais estão sendo cada vez mais utilizadas pelos pesquisadores. Através da transformação da aplicação do usuário em uma aplicação auto-adaptável ao ambiente de execução, o *middleware* EasyGrid poupa o desenvolvedor das preocupações necessárias para executar sua aplicação no ambiente distribuído, permitindo que este foque seus esforços na busca de soluções do problema estudado e não na forma com a qual suas soluções irão executar de forma distribuída.

O Portal EasyGrid foi projetado com o intuito de simplificar o acesso aos recursos de uma grade (por ex. autenticação do usuário, alocação de tarefas às máquinas disponíveis) para a execução de aplicações paralelas por parte de usuários leigos e, em particular, facilitar o desenvolvimento de heurísticas estáticas, dinâmicas e híbridas de escalonamento de tarefas.

O desenvolvimento de heurísticas de escalonamento de tarefas no Portal é tipicamente composta por quatro fases: o projeto e implementação de um ou mais algoritmos de escalonamento, a avaliação das heurísticas através da comparação experimental dos resultados gerados tanto por novos algoritmos como pelos existentes (considerando um conjunto de GADs e modelos arquiteturais escolhidos pelo próprio usuário do Portal), a validação por simulação e/ou a validação através da execução da aplicação em uma grade computacional.

Com o intuito de auxiliar ainda mais o usuário, o Portal EasyGrid oferece uma série de heurísticas seqüenciais de escalonamento (por ex. List-Scheduling, DCP, ETF, HEFT) para os usuários realizarem seus experimentos. Estas heurísticas foram implementadas, em linguagem C++, separadamente do Portal para que possa ser simples uma posterior inclusão de novos algoritmos encontrados na literatura ou desenvolvidos pelos próprios usuários. Devido aos modelos de arquitetura e de GAD serem representados por arquivos texto (Apêndice A), é possível que os usuários adicionam novos modelos ao Portal. Além disso, há ainda uma coleção de modelos arquiteturais (modelos homogêneos e heterogêneos) e de aplicações (GADs) *benchmark* pré-definidos [Boeres e Rebello 2002, Kwok e Ahmad 1999a] para que usuários com aplicações e/ou arquiteturas semelhantes possam agilizar seus experimentos.

4.2 Limitação do Portal

No Portal, cada experimento consiste na execução de uma heurística de escalonamento que tem como parâmetros um grafo (GAD) que representa a aplicação paralela, uma arquitetura que

representa o ambiente de execução e uma granularidade (média das relações entre os custos de comunicação e de processamento das tarefas da aplicação paralela). Por sua vez, é chamada de *avaliação completa* a execução de um conjunto de experimentos e a captura e exibição de seus resultados para análise do usuário. Um exemplo do uso de uma *avaliação completa* seria a realização de vários experimentos para determinar o melhor algoritmo de escalonamento (para um determinado grupo de GADs e arquiteturas) baseado na comparação do *makespan* e na quantidade de processadores utilizados.

Apesar de todos os benefícios obtidos na utilização do Portal EasyGrid, foi observada a necessidade de trabalho e tempo excessivos para realização de uma *avaliação completa*. Para que esta avaliação fosse realizada eficientemente (dando o mínimo de trabalho ao usuário) seria necessário que o Portal permitisse a seleção de diversas combinações de valores dos parâmetros dos experimentos. Entretanto, o Portal somente possibilitava a escolha de uma arquitetura e uma granularidade por vez. Dessa forma, para avaliar m arquiteturas e n granularidades seria necessário executar $m \times n$ experimentos separados. Além disso, pela alta sofisticação dos algoritmos, pela complexidade dos modelos arquiteturais e pela grande quantidade de experimentos que eram executados sequencialmente na máquina local, o usuário esperava um longo tempo (horas) para obter todos os resultados.

Por último, a exibição dos resultados para análise proporcionava ao usuário poucas opções de formatação, pois não era possível se concentrar em um subconjunto dos resultados dos experimentos realizados. As estatísticas da análise fornecidas pelo Portal eram calculadas considerando todos os resultados da avaliação corrente. Então, por exemplo, para avaliar somente um subconjunto desses experimentos seria necessário executar os experimentos novamente escolhendo apenas os parâmetros desejados, provocando assim, a realização de experimentos sem uma necessidade real.

Devido à limitação apresentada, foi projetada uma nova estratégia para realizar e analisar, de maneira rápida e adequada, os experimentos tipicamente necessários na avaliação de heurísticas de escalonamento de tarefas.

4.3 Execução Distribuída

A nova estratégia de execução de experimentos do Portal EasyGrid [Rodrigues et al. 2005] consiste na criação de uma única aplicação paralela do tipo *parameter sweep* (Figura 4.1), onde cada tarefa define um experimento, exceto a tarefa raíz (tarefa zero) que é responsável pela delegação das atividades às demais e, posteriormente, pela coleta de seus resultados.

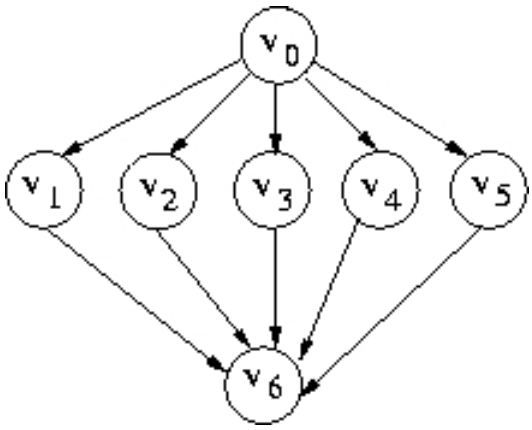


Figura 4.1: Exemplo de uma aplicação do tipo *Parameter Sweep*

Inicialmente, é solicitado ao usuário a definição das heurísticas de escalonamento (Figura 4.2) que ele deseja avaliar a partir de um dado conjunto disponibilizado pelo Portal. Após selecionar uma heurística e configurar os parâmetros que fornecem as políticas adotadas para a escolha da tarefa a ser escalonada e do processador onde esta será alocada, o usuário clica no botão *ADICIONAR* para incluir o algoritmo no conjunto de testes. Depois de definir todos os algoritmos desejados o usuário clica no botão *MÚLTIPLOS EXPERIMENTOS* para escolher os conjuntos de arquiteturas e aplicações (GADs) com os quais deseja realizar os testes. Após esta seleção é preciso definir as granularidades a serem utilizadas e confirmar quais algoritmos irão de fato fazer parte dos experimentos (Figura 4.3).

Confirmado o conjunto de testes, é apresentada uma tela para seleção do ambiente grade onde ocorrerá a execução distribuída (Figura 4.4). O primeiro passo é especificar qual grade será usada. Isso é feito à partir da leitura de um arquivo disponibilizado pelo Portal que contém todos os recursos da grade (Apêndice A). Posteriormente, o usuário apresenta, pela restrição manual do universo total ou pela leitura do arquivo que representa a sua grade (Apêndice A), quais desses recursos ele realmente possui acesso. Depois disso, o usuário deve se autenticar através da criação de um *proxy* para obter acesso aos recursos da grade. É possível criar este *proxy* na aba *PORTAL EASYGRID* desde que o usuário já tenha um certificado na sua conta. O usuário fornece seu *login*, sua senha e quanto tempo seu proxy será válido. Com essas informações o Portal cria o *proxy* e passa a monitorá-lo até a perda de sua validade. Tendo concluído essas etapas, o usuário clica no botão *STATUS DO MEU GRID* para poder coletar as informações de cada recurso. Fazendo uso de algumas funcionalidades providas pelo *globus toolkit* [Foster e Kesselman 1997] é possível obter, por exemplo, a carga, a capacidade de processamento e a memória disponível em cada recurso ativo. Dessa forma, o usuário pode conhecer o *status* da sua grade e selecionar de acordo com as características obtidas quais recursos

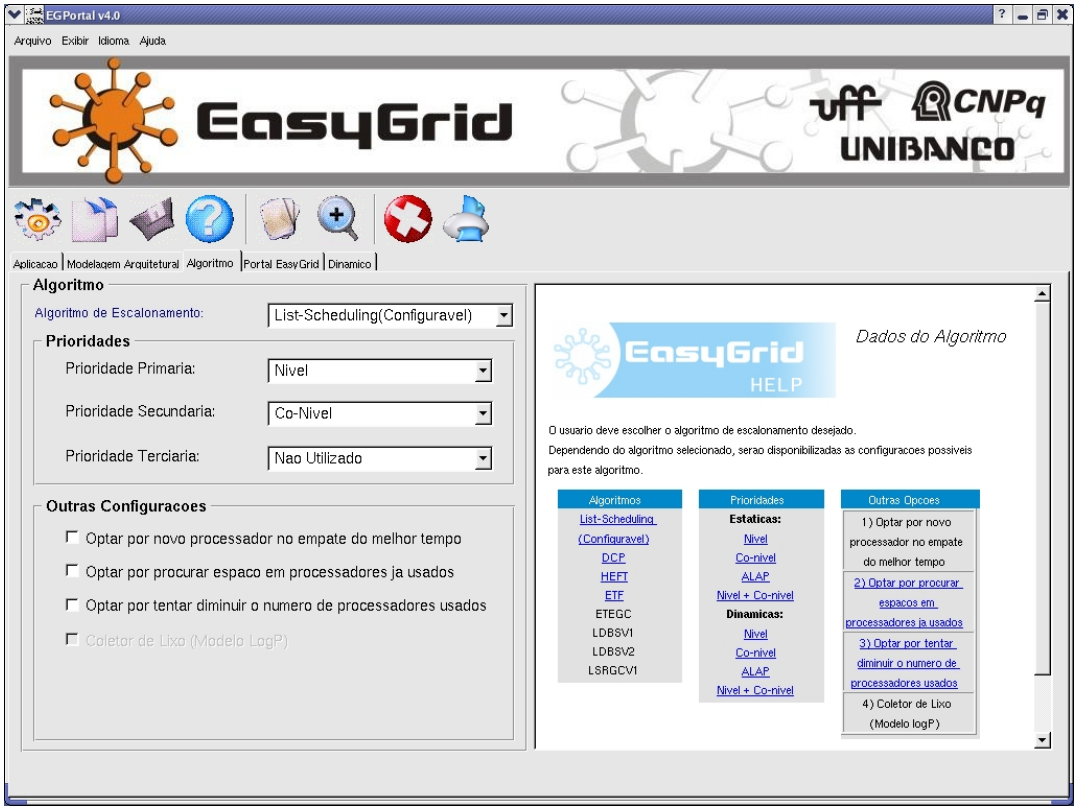


Figura 4.2: Escolha dos algoritmos

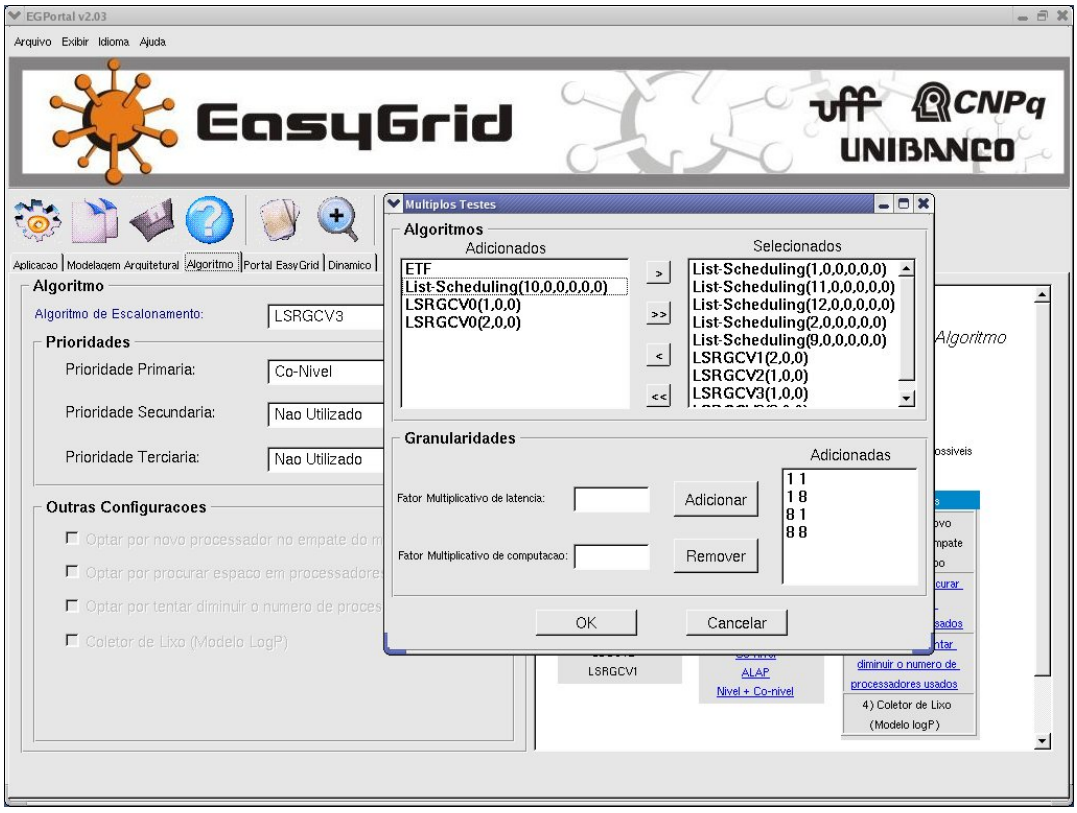


Figura 4.3: Confirmação dos algoritmos e seleção das granularidades

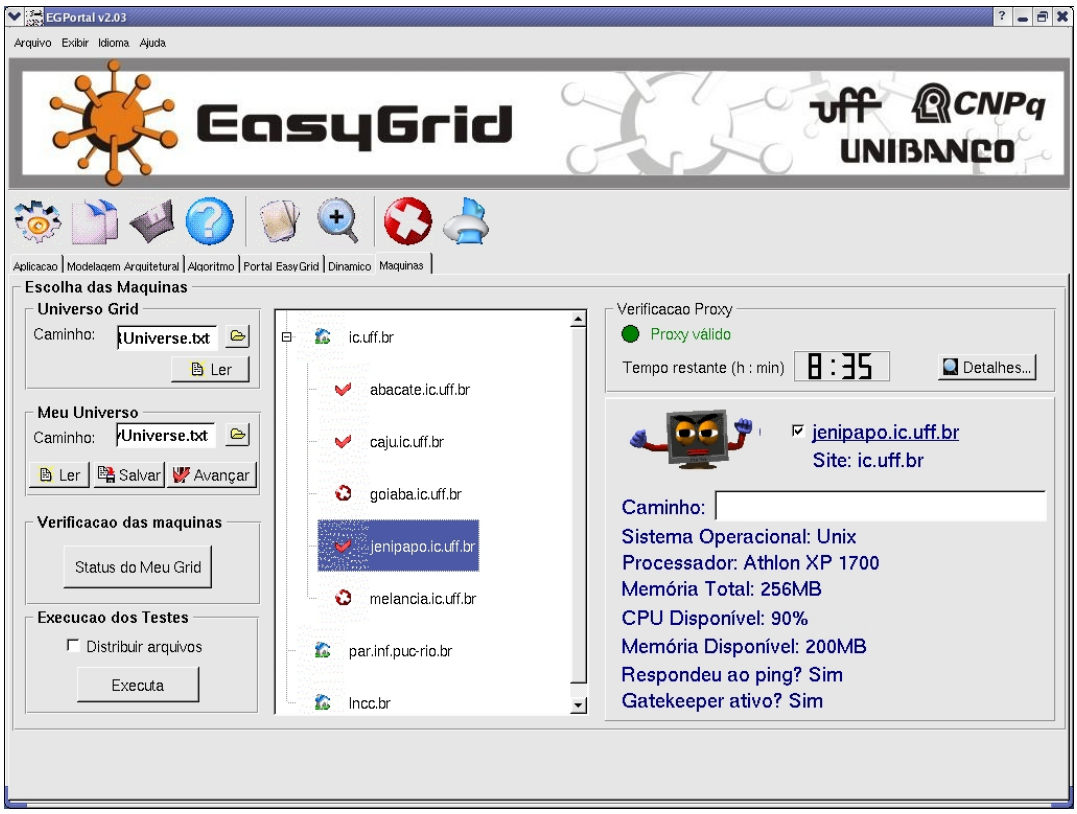


Figura 4.4: Escolha das máquinas para execução dos experimentos

irá realmente utilizar nos experimentos.

Com o ambiente distribuído definido, é preciso selecionar a opção de envio de arquivos caso essa execução distribuída (conjunto de testes e ambiente grade) não tenha sido previamente realizada. Nesse momento, o Portal EasyGrid verifica a utilização do Sistema de Arquivos de Rede (NFS) por parte dos subconjuntos, também chamados de *sites*, do ambiente grade. O NFS possibilita o compartilhamento de arquivos sobre uma rede local. Dessa forma, se um *site* possui NFS, somente o seu servidor é escolhido para receber os arquivos. Caso contrário, todos os recursos do *site* são selecionados para receberem a transferência. Independente da utilização do NFS, para transferir os arquivos é preciso compactá-los, enviá-los e, por último, descompactá-los, através de uma funcionalidade do *globus toolkit*, no processador alvo. Essa metodologia nos forneceu um *overhead* menor do que o esperado no momento de envio dos arquivos para o ambiente grade. Por último, o usuário clica no botão *EXECUTA* para poder acionar a execução distribuída com todas as características definidas anteriormente.

Neste momento, o Portal EasyGrid faz uso de um modelador [Mendes 2004] a fim de obter o poder computacional disponível de cada máquina selecionada e a latência entre os recursos da grade. Com este dado é possível efetuar uma alocação inicial (escalonamento estático) dos ex-

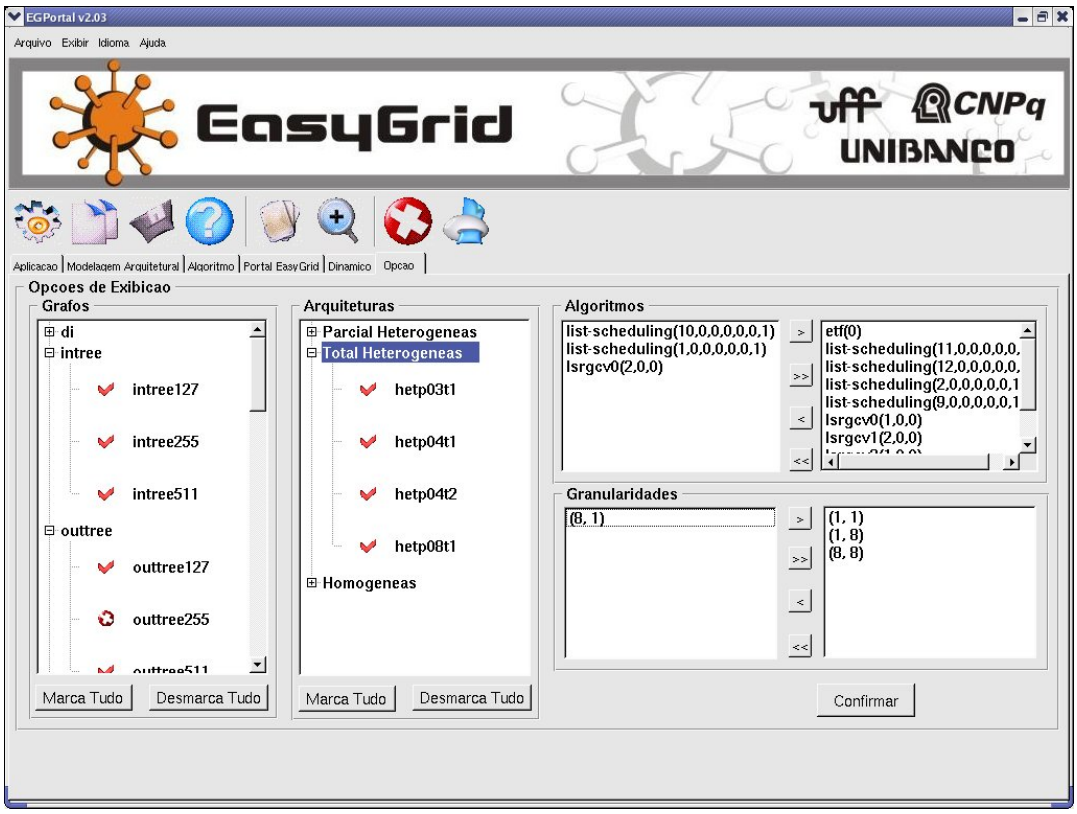
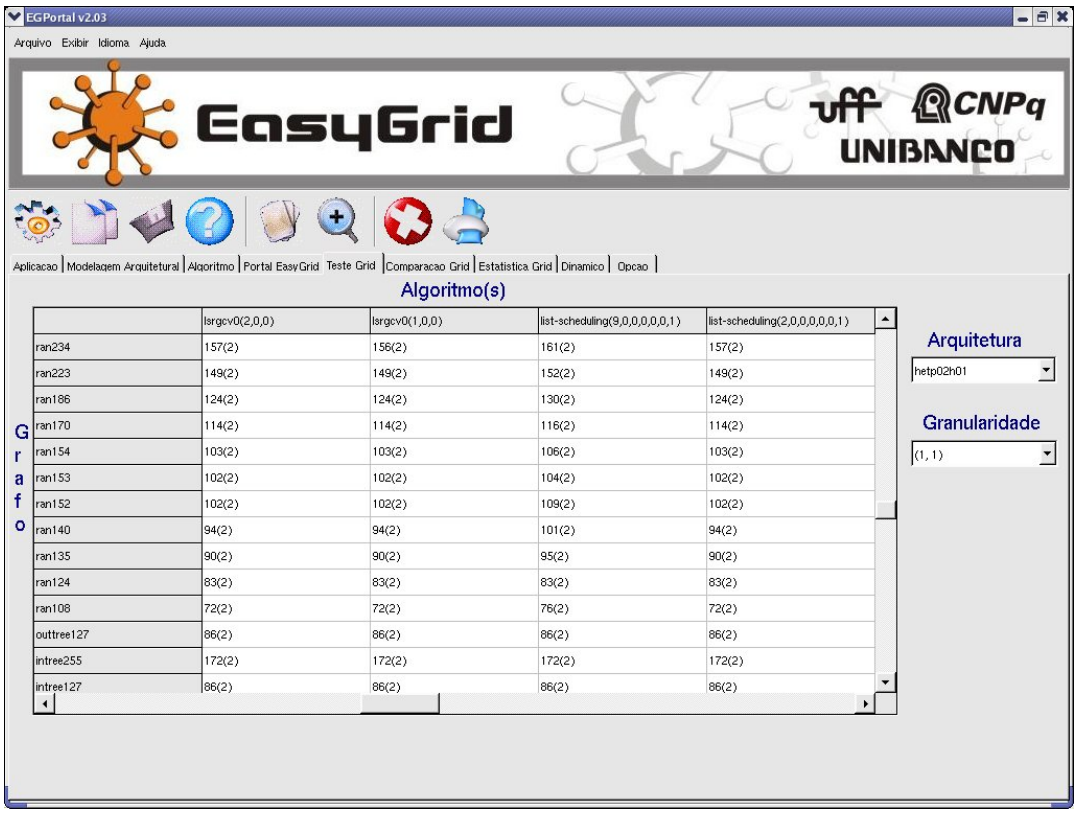


Figura 4.5: Seleção dos resultados a serem exibidos e avaliados

perimentos. Um exemplo do arquivo que representa este escalonamento é encontrado no Apêndice A. O passo seguinte é enviar todos os arquivos para os recursos previamente escolhidos de acordo com a política, descrita acima, de verificação de NFS nos *sites*. Finalmente, é criada uma aplicação MPI *system-aware*, utilizando o *middleware* EasyGrid [Nascimento et al. 2005], para gerenciar e realizar a execução de todos os experimentos nos recursos disponíveis. Sendo adaptativa, auto-escalonável e tolerante a falhas, a aplicação é capaz por si só de executar eficientemente em ambientes distribuídos compartilhados, dinâmicos e instáveis como grades computacionais porque reage e se ajusta aos seus requisitos de sistema de acordo com as suas necessidades atuais e às mudanças que ocorrerem no ambiente de execução.

O sistema de monitoramento da aplicação fornece informação sobre quanto poder computacional está sendo oferecido por cada recurso. Junto com as estimativas de custos de cada experimento (baseado na complexidade da heurística sendo avaliada), o escalonador dinâmico da aplicação consegue minimizar o tempo necessário para executar todos os experimentos balanceando o trabalho eficientemente quando as cargas dos recursos mudam. No caso de um ou mais experimentos terminarem de executar por alguma falha no sistema, a aplicação re-executa os experimentos em um recurso apropriado.



	lsrgcv0(2,0,0)	lsrgcv0(1,0,0)	list-scheduling(9,0,0,0,0,0,1)	list-scheduling(2,0,0,0,0,0,1)
ran234	157(2)	156(2)	161(2)	157(2)
ran223	149(2)	149(2)	152(2)	149(2)
ran186	124(2)	124(2)	130(2)	124(2)
ran170	114(2)	114(2)	116(2)	114(2)
ran154	103(2)	103(2)	106(2)	103(2)
ran153	102(2)	102(2)	104(2)	102(2)
ran152	102(2)	102(2)	109(2)	102(2)
ran140	94(2)	94(2)	101(2)	94(2)
ran135	90(2)	90(2)	95(2)	90(2)
ran124	83(2)	83(2)	83(2)	83(2)
ran108	72(2)	72(2)	76(2)	72(2)
outtree127	86(2)	86(2)	86(2)	86(2)
intree255	172(2)	172(2)	172(2)	172(2)
intree127	86(2)	86(2)	86(2)	86(2)

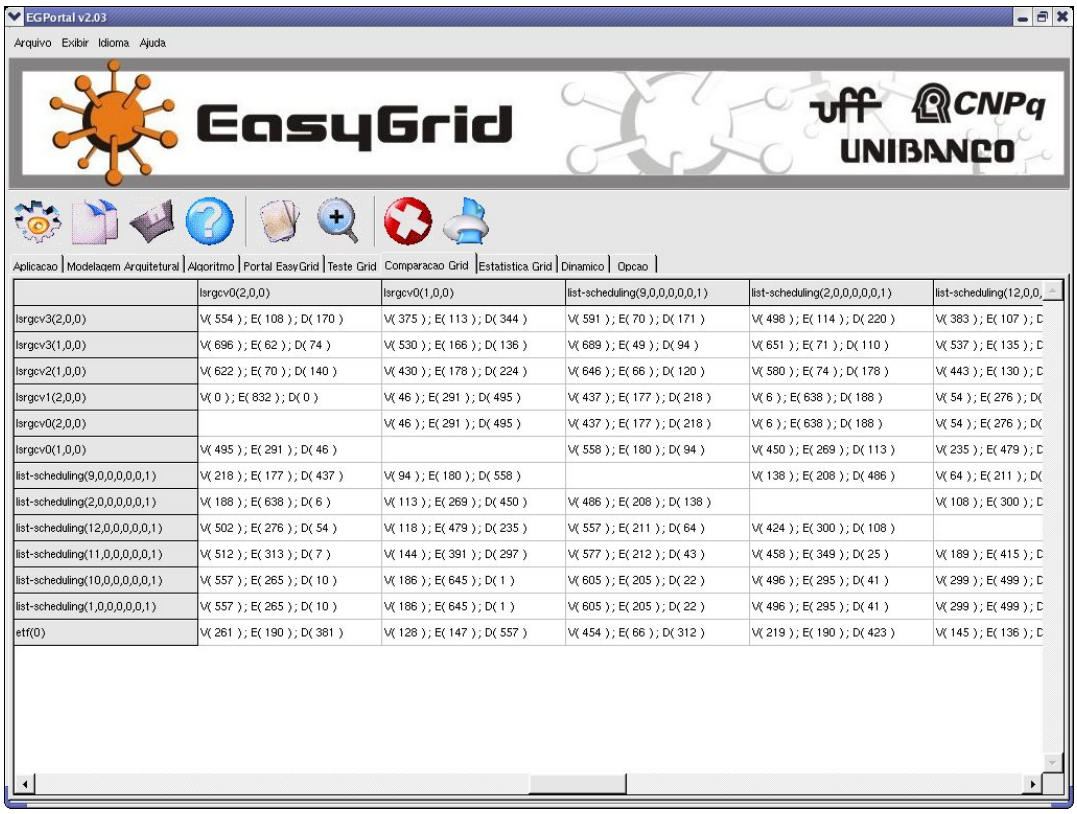
Figura 4.6: Tabela Algoritmo x GAD

Após o término da execução é mostrada ao usuário a tela de escolha do formato de visualização dos resultados (Figura 4.5). O usuário pode selecionar qualquer combinação de parâmetros que deseja. Dessa forma, é possível analisar resultados para, por exemplo, classes de aplicações ou arquiteturas específicas. Uma vez que se pode obter qualquer subconjunto desejado por causa desta nova forma de visualização, não é preciso realizar experimentos sem uma necessidade real como acontecia anteriormente. Tendo escolhido todos os parâmetros, o usuário clica no botão *CONFIRMAR* para poder visualizar os resultados nas tabelas Algoritmo x GAD, Par-a-Par e Estatística.

A tabela Algoritmo x GAD (Figura 4.6) permite a verificação do número de processadores e do tempo total de escalonamento (*makespan*) de cada par (Algoritmo, GAD) para uma arquitetura e uma granularidade. O usuário pode escolher a combinação da arquitetura e da granularidade da maneira como desejar.

A tabela Par-a-Par (Figura 4.7), compara cada par de algoritmos em termos do número de vitórias, empates e derrotas. Sendo assim, permite que o usuário saiba se um determinado algoritmo obteve bons resultados em relação a cada uma das outras heurísticas da avaliação.

A tabela Estatística (Figura 4.8) mostra quantas vezes cada algoritmo conseguiu gerar um



	lsrgcv0(2,0,0)	lsrgcv0(1,0,0)	list-scheduling(9,0,0,0,0,0,1)	list-scheduling(2,0,0,0,0,0,1)	list-scheduling(12,0,0,
lsrgcv3(2,0,0)	V(554); E(108); D(170)	V(375); E(113); D(344)	V(591); E(70); D(171)	V(498); E(114); D(220)	V(383); E(107); D
lsrgcv3(1,0,0)	V(696); E(62); D(74)	V(530); E(166); D(136)	V(689); E(49); D(94)	V(651); E(71); D(110)	V(537); E(135); D
lsrgcv2(1,0,0)	V(622); E(70); D(140)	V(430); E(178); D(224)	V(646); E(66); D(120)	V(580); E(74); D(178)	V(443); E(130); D
lsrgcv1(2,0,0)	V(0); E(832); D(0)	V(46); E(291); D(495)	V(437); E(177); D(218)	V(6); E(638); D(188)	V(54); E(276); D(
lsrgcv0(2,0,0)		V(46); E(291); D(495)	V(437); E(177); D(218)	V(6); E(638); D(188)	V(54); E(276); D(
lsrgcv0(1,0,0)	V(495); E(291); D(46)		V(558); E(180); D(94)	V(450); E(269); D(113)	V(235); E(479); D
list-scheduling(9,0,0,0,0,0,1)	V(218); E(177); D(437)	V(94); E(180); D(558)		V(138); E(208); D(486)	V(64); E(211); D(
list-scheduling(2,0,0,0,0,0,1)	V(188); E(638); D(6)	V(113); E(269); D(450)	V(486); E(208); D(138)		V(108); E(300); D
list-scheduling(12,0,0,0,0,0,1)	V(502); E(276); D(54)	V(118); E(479); D(235)	V(557); E(211); D(64)	V(424); E(300); D(108)	
list-scheduling(11,0,0,0,0,0,1)	V(512); E(313); D(7)	V(144); E(391); D(297)	V(577); E(212); D(43)	V(458); E(349); D(25)	V(189); E(415); D
list-scheduling(10,0,0,0,0,0,1)	V(557); E(265); D(10)	V(186); E(645); D(1)	V(605); E(205); D(22)	V(496); E(295); D(41)	V(299); E(499); D
list-scheduling(1,0,0,0,0,0,1)	V(557); E(265); D(10)	V(186); E(645); D(1)	V(605); E(205); D(22)	V(496); E(295); D(41)	V(299); E(499); D
etf(0)	V(261); E(190); D(381)	V(128); E(147); D(557)	V(454); E(66); D(312)	V(219); E(190); D(423)	V(145); E(136); D

Figura 4.7: Tabela Par-a-Par

escalonamento com o melhor *makespan* em relação aos outros e o grau de degradação médio do melhor *makespan* conhecido nos experimentos sendo analisados. É importante destacar que os melhores escalonamentos obtidos são aqueles com o grau de degradação médio mais próximo de um e, à medida que este valor aumenta, a qualidade do escalonamento piora.

4.4 Avaliação de Desempenho

Com a inclusão da interface para múltiplos experimentos, o Portal EasyGrid tem conseguido realizar os experimentos com facilidade e rapidez. Diferentemente do exemplo da Seção 4.3, com uma única execução conseguimos o mesmo resultado antes obtido através de $m \times n$ repetições, reduzindo o tempo total de execução dos experimentos.

Como exemplo, considere um conjunto de experimentos típico de 58500 experimentos obtidos através da combinação de 15 configurações de arquiteturas, 4 diferentes granularidades, 65 aplicações distribuídos em 6 classes e 15 heurísticas. Este conjunto demorava aproximadamente 3 horas utilizando a ferramenta original e foi reduzido para 2 horas (executado em uma única máquina usando o *middleware* EasyGrid) ou para um resultado ainda mais significativo

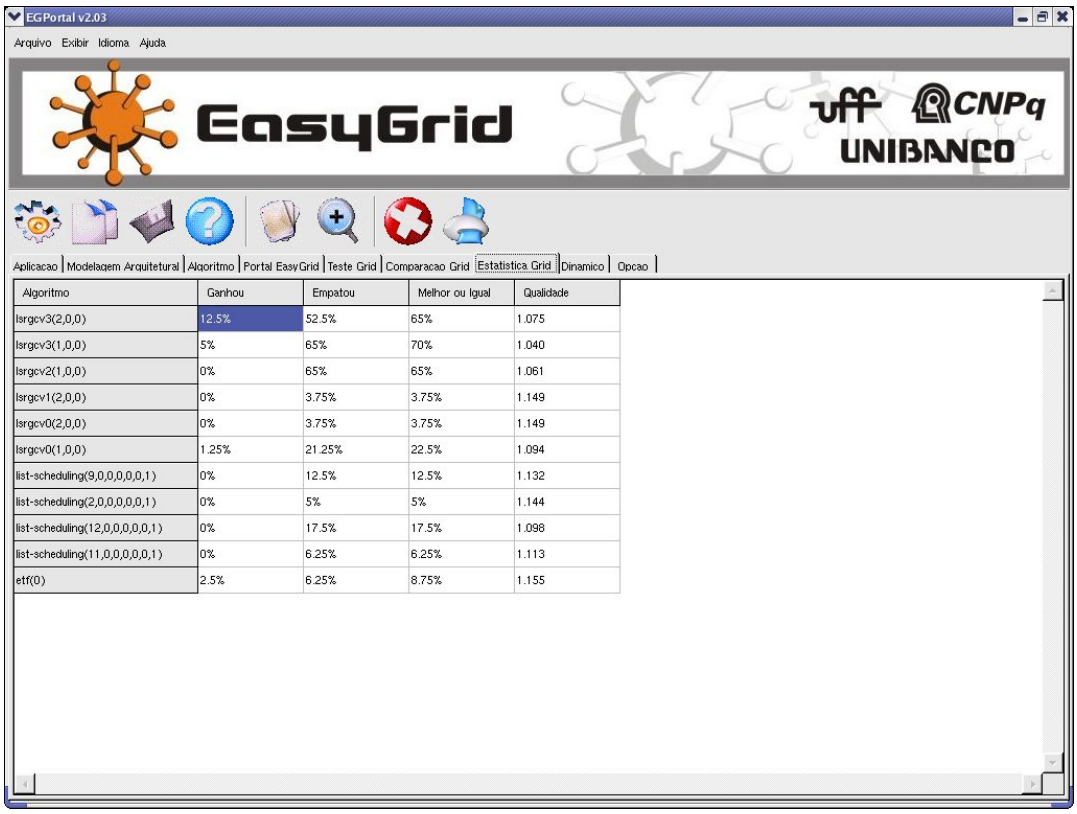


Figura 4.8: Tabela Estatística

de apenas 8 minutos quando executado em um ambiente grade com trinta e duas máquinas (compartilhadas com outros usuários).

O gráfico da Figura 4.9 apresenta, para esse conjunto, os tempos de execução e resposta para diferentes quantidades de máquinas. Na primeira coluna (legenda: execução), é mostrado o tempo para executar os experimentos e, na coluna ao lado (legenda: resposta), é apresentado o tempo total que inclui, além do tempo de execução, o retardo para a criação do ambiente MPI-LAM, a distribuiçao dos arquivos e, após a execução, a coleta dos resultados e o encerramento do ambiente. Como é possível perceber, este gráfico confirma que a utilização de um número crescente de recursos computacionais proporciona uma redução significativa no tempo de execução dos experimentos.

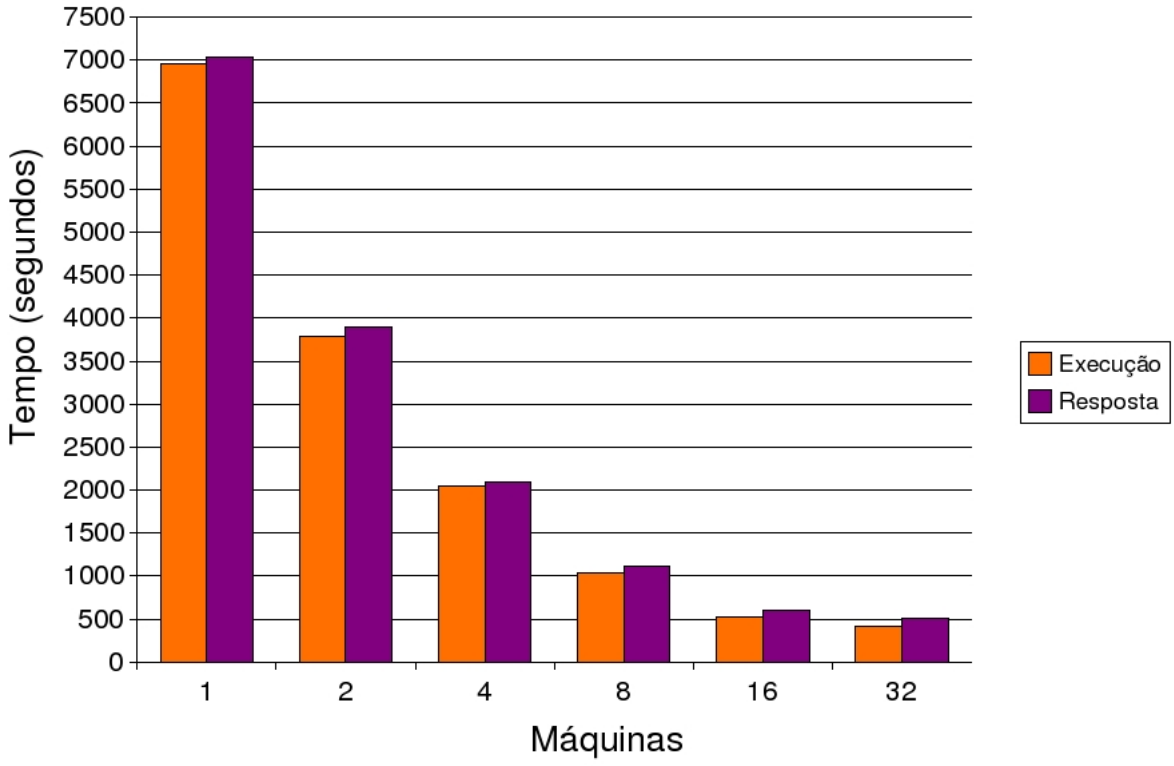


Figura 4.9: Desempenho da interface

Capítulo 5

Experimentos Computacionais

Este capítulo apresenta um estudo comparativo das heurísticas de escalonamento de tarefas do tipo List-Scheduling. Com o intuito de avaliar as principais características de cada heurística, milhares de experimentos foram realizados. Para isto, o novo ambiente de avaliação do Portal EasyGrid foi utilizado.

Neste também são apresentados as medidas de performance utilizadas para a avaliação das heurísticas, os resultados das avaliações através de tabelas comparando os resultados obtidos pelos algoritmos de escalonamento com os diferentes tipos de aplicações e por fim, os algoritmos que obtiveram os melhores resultados para cada classe de aplicação.

Os testes foram divididos em 3 grandes grupos. O primeiro teste teve o objetivo de eliminar a enorme quantidade de algoritmos existentes através da escolha dos algoritmos que obtiveram os melhores resultados para cada tipo de aplicação. O resultado deste primeiro teste funcionou como entrada para o segundo teste que selecionou de maneira mais criteriosa os melhores pares de primeira e segunda prioridade. O último teste utilizou a mesma entrada do segundo teste mas selecionou os algoritmos que geraram os melhores resultados na média, ou seja, foram bons para a maioria dos tipos de aplicações.

5.1 Condições gerais dos experimentos

Um conceito extremamente importante para se compreender os testes comparativos e seus resultados é a definição de experimento. Um experimento é composto por um algoritmo de escalonamento de tarefas, uma aplicação, uma arquitetura alvo e uma granularidade.

Os algoritmos avaliados são do tipo *List-Scheduling* e, como citado no Capítulo 3, esta heurística de escalonamento associa prioridades às tarefas de uma aplicação e com base nestes valores escolhe as tarefas que devem ser escalonadas primeiro. A diferença entre cada algoritmo

List-Scheduling avaliado está na forma com a qual as tarefas de uma aplicação paralela são escolhidas para serem escalonadas. Os testes realizados utilizaram as prioridades definidas no Capítulo 3, que são: nível, conível, nível mais conível, alap, número de sucessores, lbnível, largura e alap menos maior aresta incidente. É válido destacar que no algoritmo List-Scheduling utilizado também foi possível definir uma segunda prioridade que é utilizada para o desempate entre duas tarefas, no caso das tarefas terem o mesmo valor para a primeira prioridade.

As aplicações definem os problemas dos usuários e podem ser classificadas com base em suas características, por exemplo, alta comunicação entre as tarefas, aplicação no formato de uma árvore, etc. Estes testes utilizaram aplicações das classes definidas no Capítulo 2, que são: Diamante (Di), Árvores Binárias Reversas Completas (*Intree*), Irregular (Ir), Kwok's Peer Set Graphs (PSG), Árvores Binárias Completas (*Outtree*), Randômico (Ran).

Como o objetivo da computação em grades é utilizar recursos geograficamente distribuídos, é improvável que ao executar uma aplicação paralela um usuário tenha um conjunto de recursos semelhantes ou homogêneos. Dessa forma, para dar veracidade à execução dos experimentos, também foram utilizados dois grupos de arquiteturas heterogêneas: arquiteturas totalmente heterogêneas e arquiteturas parcialmente heterogêneas.

O último parâmetro de configuração de um experimento é a granularidade que representa a relação entre a taxa de comunicação e processamento das tarefas de uma dada aplicação. A granularidade de uma aplicação é definida por um par ordenado (x,y), tal que x é o fator multiplicativo de latência e y o fator multiplicativo de computação. Os custos de comunicação entre as tarefas e o custo de processamento de cada tarefa de uma aplicação são multiplicados pelo fator multiplicativo de latência e pelo fator multiplicativo de computação respectivamente.

Os testes foram divididos em três fases. Para cada fase os resultados eram avaliados e filtrados de tal forma que os melhores algoritmos eram selecionados para fazerem parte da fase seguinte. Na primeira fase todas as combinações de algoritmos, aplicações, arquiteturas e granularidades foram geradas, resultando assim em uma enorme quantidade de experimentos. Os experimentos das demais fases foram gerados com base nos resultados das fases anteriores, de tal forma que a quantidade de experimentos foi diminuindo.

Cada algoritmo do tipo List-Scheduling utilizado possui uma primeira e uma segunda prioridade. Na primeira fase foram utilizadas todas as 8 prioridades definidas anteriormente para o valor da primeira prioridade. Ainda nesta primeira fase, foram utilizados todos os mesmos valores da primeira prioridade para a segunda prioridade mais o valor nulo, resultando em um total de 9 segundas prioridades. Dessa forma, é fácil compreender que para a primeira fase de testes foram utilizados $8 \times 9 = 72$ algoritmos. As quantidade dos demais parâmetros de confi-

guração dos experimentos desta primeira fase de testes foram: 11 arquiteturas, 4 granularidades e 65 aplicações de diferentes classes. Portanto, a quantidade de experimentos dessa primeira fase pode ser calculada multiplicando a quantidade dos parâmetros de configuração dos experimentos, resultando em um total de $72 \times 11 \times 4 \times 65 = 205920$ experimentos. É válido destacar que esses experimentos foram agrupados através da primeira prioridade dos algoritmos. Assim, os 205920 experimentos foram resultado de 8 (número de primeiras prioridades) execuções de 25740 (9 segundas prioridades, 11 arquiteturas e 4 granularidades) experimentos.

Os experimentos foram realizados em grupos em virtude das limitações do middleware EasyGrid já que a maior aplicação executada pelo middleware tinha 150000 tarefas e o conjunto total de experimentos é formado por um número bem maior que este. Uma outra restrição foi a capacidade da interface de coletar e manter em memória o resultado de tantos experimentos, uma vez que testes da ordem de 200000 experimentos ainda não tinham sido realizados. Atualmente, trabalhos paralelos a este estão realizando manutenção sobre a interface do Portal EasyGrid e o middleware EasyGrid para que estes possam dar suporte as necessidades apresentadas acima.

Para a segunda fase foram selecionados 24 algoritmos, para cada classe de aplicação, dos 72 utilizados para todas as classes na primeira, 11 formatos de arquitetura, 4 granularidades e 50 aplicações. As 50 aplicações se distribuem pelas classes Di (11), Ir (7), KPSG (11) e Ran (21). Dessa forma, é possível dizer que foram executados 11616 ($24 \times 11 \times 4 \times 11$) experimentos para a classe Di, 7392 ($24 \times 11 \times 4 \times 7$) para Ir, 11616 ($24 \times 11 \times 4 \times 11$) para KPSG e 22176 ($24 \times 11 \times 4 \times 21$) para Ran, resultando em um total de 52800 experimentos. A forma com a qual os algoritmos da segunda fase de testes foram selecionados será descrita na próxima seção.

Na última fase foram executados 68640 experimentos, resultado da combinação dos 24 algoritmos escolhidos ao término da primeira fase, de 11 arquiteturas, 4 granularidades e 65 aplicações das diferentes classes de aplicação já definidas.

5.2 Medidas de performance

Embora os parâmetros que caracterizam um experimento já tenham sido analisados na seção anterior, nada foi mencionado sobre o resultado de um experimento, ou seja, o que é gerado a partir de sua execução. Como já foi dito, um experimento consiste na simulação da execução de um algoritmo de escalonamento para uma dada aplicação de entrada em uma determinada máquina alvo, valendo-se de uma granularidade para alterar de forma constante os custos de execução e comunicação das tarefas da aplicação. Dessa forma, após esta execução são gerados

Experimento	Configuração
algoritmos	alg0, alg1
aplicações	app0, app1
arquiteturas	arqt0, arqt1
granularidades	gran0

Tabela 5.1: Exemplo: configuração de entrada

um escalonamento, que define em quais máquinas da arquitetura alvo cada tarefa da aplicação irá ser executada e em qual instante, e um *makespan* que representa o tempo que aquela aplicação irá demorar para ser executada mediante as condições apresentadas. Assim, quanto melhor o algoritmo de escalonamento menor será o valor do *makespan* para uma aplicação. Como em computação paralela é admitido que os recursos utilizados para executar uma aplicação são distribuídos, e conseqüentemente estão presentes em grande quantidade, e o foco deste trabalho está na avaliação dos algoritmos de escalonamento, o número de processadores utilizados por um algoritmo de escalonamento para distribuir as tarefas de uma aplicação não está sendo considerado como parâmetro de maior importância para a análise dos algoritmos neste trabalho. Em contrapartida, o valor do *makespan* de uma dada execução é tido como matéria prima para a avaliação dos experimentos realizados. Os valores dos *makespans* são utilizados para o cálculo da qualidade dos algoritmos. Uma opção não abordada neste trabalho é analisar o número de processadores utilizados por um escalonamento para o desempate de algoritmos, caso os valores dos *makespans* gerados fossem iguais. Atualmente o desempate é feito de forma aleatória pelo Portal EasyGrid.

Seja T um teste com w algoritmos, x aplicações, y arquiteturas e z granularidades, com um total de $w \times x \times y \times z$ experimentos. Dessa forma, é fácil concluir que cada conjunto de parâmetros (aplicação, arquitetura e granularidade) será executado com w algoritmos distintos, ou seja, gerarão w resultados (*makespans*). Assim, para identificar o algoritmo cujo experimento obteve o menor *makespan* para um conjunto de parâmetros (aplicação, arquitetura e granularidade), é preciso avaliar o *makespan* de todos os experimentos que tenham este mesmo conjunto de parâmetros. Considere o exemplo das tabelas 5.1 e 5.2, que consiste na configuração de entrada de um teste e seus experimentos.

Para identificar o algoritmo que gerou o melhor *makespan* para o conjunto de parâmetros (app1, arqt0, gran0) será preciso avaliar o *makespan* gerado pelos experimentos 2 e 6. Esse valor é dado por min{makespan(experimento2),makespan(experimento6)}.

A qualidade de um algoritmo para um determinado experimento com um conjunto de parâmetros (aplicação, arquitetura e granularidade) é dado pela divisão do *makespan*, gerado pelo

Experimento	Configuração
experimento 0	(alg0, app0, arqt0, gran0)
experimento 1	(alg0, app0, arqt1, gran0)
experimento 2	(alg0, app1, arqt0, gran0)
experimento 3	(alg0, app1, arqt1, gran0)
experimento 4	(alg1, app0, arqt0, gran0)
experimento 5	(alg1, app0, arqt1, gran0)
experimento 6	(alg1, app1, arqt0, gran0)
experimento 7	(alg1, app1, arqt1, gran0)

Tabela 5.2: Exemplo: experimentos

algoritmo para este experimento, pelo valor do melhor *makespan* gerado pelos experimentos com o mesmo conjunto de parâmetros (aplicação, arquitetura e granularidade). Estendendo este conceito, é possível definir a qualidade de um algoritmo para um conjunto de experimentos com diferentes conjuntos de parâmetros (aplicação, arquitetura e granularidade), como a média das qualidades do algoritmo obtidas para cada conjunto de parâmetros (aplicação, arquitetura e granularidade). Assim, de acordo com o exemplo da Tabela 5.2 é possível dizer que a qualidade do algoritmo alg1 (qualidade(alg1)) é dado por:

$$A = \frac{makespan(exp4)}{\min\{makespan(exp0), makespan(exp4)\}}$$

$$B = \frac{makespan(exp5)}{\min\{makespan(exp1), makespan(exp5)\}}$$

$$C = \frac{makespan(exp6)}{\min\{makespan(exp2), makespan(exp6)\}}$$

$$D = \frac{makespan(exp7)}{\min\{makespan(exp3), makespan(exp7)\}}$$

$$qualidade(alg1) = \frac{A + B + C + D}{4}$$

5.3 Resultados

Para a execução dos testes deste trabalho, as prioridades dos algoritmos foram indexadas segundo a Tabela 5.3 e a nomenclatura dos algoritmos seguiu o seguinte padrão. Admita que o algoritmo List-Scheduling com primeira e segunda prioridades a e b respectivamente está associado a um experimento. Assim, neste trabalho este algoritmo é apresentado como $List - Scheduling(num(a), num(b))$, onde p é uma prioridade e $num(p)$ é o número da prioridade p

Prioridade	Número da Prioridade
Nulo	0
Nível	1
Conível	2
Nível mais conível	3
Alap	4
Número de sucessores	9
LBNível	10
Largura	11
Alap menos maior aresta incidente	12

Tabela 5.3: Legenda das Prioridades

segundo a Tabela 5.3.

É válido destacar que a prioridade Nulo é apenas utilizada como segunda prioridade de um algoritmo e indica que nenhuma segunda prioridade deve ser utilizada. Assim, caso duas tarefas a serem escalonadas tenham o mesmo valor para sua primeira prioridade, a tarefa escolhida será a primeira tarefa avaliada segundo a ordem que o algoritmo de escalonamento analisa as tarefas de uma aplicação.

As tabelas 5.4 até 5.11 apresentam os resultados obtidos através do primeiro teste. Assim, para cada primeira prioridade, 8 no total, é apresentada uma tabela com os valores das médias das qualidades obtidas para cada segunda prioridade e tipo de aplicação. Na primeira linha de cada tabela há o número da primeira prioridade seguida das segundas prioridades, de acordo com a Tabela 5.3, os elementos da primeira coluna, exceto o primeiro, representam os tipos de aplicação e as demais células exibem as qualidades médias obtidas nos testes.

Os resultados só podem ser comparados dentro de cada tabela, já que o cálculo das qualidade de cada tabela se baseou em uma primeira prioridade específica, ou seja, as comparações foram feitas entre algoritmos com a mesma primeira prioridade. Assim, para cada primeira prioridade foram escolhidas as três segundas prioridades que obtiveram a melhor qualidade para cada tipo de aplicação e, no caso de empate, as prioridades foram escolhidas aleatoriamente. Os melhores valores estão marcados de vermelho.

Com base na análise das tabelas 5.4 até 5.11, foram escolhidas as três melhores segundas prioridades, para cada tipo de aplicação, de cada tipo de primeira prioridade (cada tabela), resultando em um total de 24 algoritmos escolhidos para cada classe de aplicação. Assim a quantidade de algoritmos que inicialmente eram 72 (para cada classe), passou a ser 24 para a segunda fase. Em virtude dos resultados observados na primeira fase, as classes de aplicações *intree* e *outree* não foram selecionadas para este segundo teste. A opção de restringir as classes

1	0	1	2	3	4	9	10	11	12
Di	1,0090	1,0090	1,0090	1,0090	1,0090	1,0012	1,0090	1,0090	1,0090
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0038	1,0038	1,0056	1,0041	1,0038	1,0021	1,0038	1,0056	1,0038
KPSG	1,0143	1,0143	1,0052	1,0130	1,0143	1,0139	1,0143	1,0052	1,0137
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0155	1,0155	1,0128	1,0484	1,0155	1,0217	1,0155	1,0128	1,0311
Todos	1,0093	1,0093	1,0071	1,0198	1,0093	1,0098	1,0093	1,0071	1,0143

Tabela 5.4: List-Scheduling com primeira prioridade nível

2	0	1	2	3	4	9	10	11	12
Di	1,0090	1,0090	1,0090	1,0090	1,0090	1,0012	1,0090	1,0090	1,0090
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0349	1,0344	1,0349	1,0344	1,0344	1,0232	1,0250	1,0344	1,0344
KPSG	1,0186	1,0145	1,0186	1,0145	1,0145	1,0206	1,0075	1,0145	1,0145
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0247	1,0209	1,0247	1,0209	1,0209	1,0208	1,0209	1,0209	1,0209
Todos	1,0164	1,0144	1,0164	1,0144	1,0144	1,0129	1,0122	1,0144	1,0144

Tabela 5.5: List-Scheduling com primeira prioridade conível

3	0	1	2	3	4	9	10	11	12
Di	1,0354	1,0955	1,0955	1,0354	1,0955	1,0606	1,0955	1,0955	1,0955
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0028	1,0028	1,0028	1,0028	1,0028	1,0108	1,0028	1,0028	1,0028
KPSG	1,0035	1,0005	1,0005	1,0035	1,0005	1,0035	1,0005	1,0005	1,0005
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0288	1,0226	1,0226	1,0288	1,0226	1,0279	1,0226	1,0226	1,0226
Todos	1,0162	1,0238	1,0238	1,0162	1,0238	1,0210	1,0238	1,0238	1,0238

Tabela 5.6: List-Scheduling com primeira prioridade nível mais conível

4	0	1	2	3	4	9	10	11	12
Di	1,0090	1,0090	1,0090	1,0090	1,0090	1,0012	1,0090	1,0090	1,0090
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0038	1,0038	1,0056	1,0041	1,0038	1,0021	1,0038	1,0056	1,0038
KPSG	1,0143	1,0143	1,0052	1,0130	1,0143	1,0139	1,0143	1,0052	1,0137
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0155	1,0155	1,0128	1,0484	1,0155	1,0217	1,0155	1,0128	1,0311
Todos	1,0093	1,0093	1,0071	1,0198	1,0093	1,0098	1,0093	1,0071	1,0143

Tabela 5.7: List-Scheduling com primeira prioridade alap

9	0	1	2	3	4	9	10	11	12
Di	1,0304	1,0631	1,0631	1,0304	1,0631	1,0304	1,0631	1,0631	1,0631
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0378	1,0173	1,0364	1,0340	1,0173	1,0378	1,0346	1,0244	1,0174
KPSG	1,0533	1,0336	1,0326	1,0675	1,0336	1,0533	1,0463	1,0269	1,0347
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0426	1,0366	1,0393	1,0608	1,0366	1,0426	1,0366	1,0308	1,0419
Todos	1,0320	1,0301	1,0328	1,0399	1,0301	1,0320	1,0341	1,0278	1,0320

Tabela 5.8: List-Scheduling com primeira prioridade número de sucessores

10	0	1	2	3	4	9	10	11	12
Di	1,0090	1,0090	1,0090	1,0090	1,0090	1,0012	1,0090	1,0090	1,0090
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0127	1,0082	1,0055	1,0134	1,0082	1,0145	1,0127	1,0055	1,0082
KPSG	1,0239	1,0149	1,0072	1,0176	1,0149	1,0223	1,0239	1,0063	1,0148
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0155	1,0155	1,0128	1,0484	1,0155	1,0217	1,0155	1,0128	1,0311
Todos	1,0119	1,0099	1,0075	1,0216	1,0099	1,0125	1,0119	1,0073	1,0150

Tabela 5.9: List-Scheduling com primeira prioridade lbnivel

11	0	1	2	3	4	9	10	11	12
Di	1,0090	1,0090	1,0090	1,0090	1,0090	1,0012	1,0090	1,0090	1,0090
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0033	1,0033	1,0033	1,0033	1,0033	1,0016	1,0033	1,0033	1,0033
KPSG	1,0016	1,0007	1,0016	1,0007	1,0007	1,0008	1,0005	1,0016	1,0007
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0190	1,0120	1,0252	1,0120	1,0120	1,0124	1,0120	1,0190	1,0120
Todos	1,0083	1,0059	1,0103	1,0059	1,0059	1,0045	1,0058	1,0083	1,0059

Tabela 5.10: List-Scheduling com primeira prioridade largura

12	0	1	2	3	4	9	10	11	12
Di	1,0090	1,0090	1,0090	1,0090	1,0090	1,0012	1,0090	1,0090	1,0090
Intree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ir	1,0039	1,0037	1,0037	1,0041	1,0037	1,0022	1,0037	1,0037	1,0039
KPSG	1,0016	1,0015	1,0037	1,0014	1,0015	1,0016	1,0016	1,0037	1,0016
Outtree	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Ran	1,0153	1,0153	1,0168	1,0318	1,0153	1,0180	1,0153	1,0168	1,0153
Todos	1,0071	1,0071	1,0080	1,0125	1,0071	1,0065	1,0071	1,0080	1,0071

Tabela 5.11: List-Scheduling com primeira prioridade alap menos maior aresta incidente

1	0	1	2	3	4	9	10	11	12
Di	1,0970	1,0970	---	---	---	1,0882	---	---	---
Ir	---	1,0865	---	---	1,0865	1,0848	---	---	---
KPSG	---	---	1,0555	1,0633	---	---	---	1,0555	---
Ran	---	1,0202	1,0174	---	---	---	---	1,0174	---

Tabela 5.12: List-Scheduling com primeira prioridade nível

2	0	1	2	3	4	9	10	11	12
Di	1,0970	1,0970	---	---	---	1,0882	---	---	---
Ir	---	---	---	---	1,1030	1,0918	1,0937	---	---
KPSG	---	1,0843	---	---	1,0843	---	1,0774	---	---
Ran	---	1,0963	---	---	1,0963	1,0956	---	---	---

Tabela 5.13: List-Scheduling com primeira prioridade conível

de aplicações é explicada na seção seguinte.

Os resultados gerados por este segundo teste são apresentados nas tabelas 5.12 até 5.19 de forma análoga aos resultados do primeiro teste, exceto pelo fato de que, como as segundas prioridade escolhidas não foram semelhantes para os diferentes tipos de primeira prioridade e classe de grafo, algumas células das tabelas estão com valores nulos.

É possível observar que nenhuma heurística se destacou de forma acentuada nos testes. Como a diferença entre os resultados foi muito pequena e ocorreram muitos empates não foi possível eleger, para cada classe de aplicação, a melhor heurística. Dessa forma, foi feita uma seleção com o intuito de escolher as heurísticas que geraram os melhores *makespans* ou valores próximos a estes.

A seleção dos melhores algoritmos para esta segunda fase, diferentemente da seleção do

3	0	1	2	3	4	9	10	11	12
Di	1,0369	---	---	1,0369	---	1,0621	---	---	---
Ir	1,0965	1,0965	1,0965	---	---	---	---	---	---
KPSG	---	1,0775	1,0775	---	1,0775	---	---	---	---
Ran	---	1,1228	1,1228	---	1,1228	---	---	---	---

Tabela 5.14: List-Scheduling com primeira prioridade nível mais conível

4	0	1	2	3	4	9	10	11	12
Di	1,0970	1,0970	---	---	---	1,0882	---	---	---
Ir	---	1,0865	---	---	1,0865	1,0848	---	---	---
KPSG	---	---	1,0555	1,0633	---	---	---	1,0555	---
Ran	---	1,0202	1,0174	---	---	---	---	1,0174	---

Tabela 5.15: List-Scheduling com primeira prioridade alap

9	0	1	2	3	4	9	10	11	12
Di	1,0621	---	---	1,0621	---	1,0621	---	---	---
Ir	---	1,0634	---	---	1,0634	---	---	---	1,0635
KPSG	---	---	1,0610	---	1,0610	---	---	1,0542	---
Ran	---	1,0956	---	---	---	---	1,0956	1,0893	---

Tabela 5.16: List-Scheduling com primeira prioridade número de sucessores

10	0	1	2	3	4	9	10	11	12
Di	1,0970	1,0970	---	---	---	1,0882	---	---	---
Ir	---	---	1,0838	---	1,0865	---	---	1,0838	---
KPSG	---	---	1,0434	---	---	---	---	1,0424	1,0509
Ran	---	1,0202	1,0174	---	---	---	---	1,0174	---

Tabela 5.17: List-Scheduling com primeira prioridade lbnível

11	0	1	2	3	4	9	10	11	12
Di	1,0970	1,0970	---	---	---	1,0882	---	---	---
Ir	---	1,0847	1,0847	---	---	1,0830	---	---	---
KPSG	---	1,0544	---	---	1,0544	---	1,0541	---	---
Ran	---	1,0406	---	1,0406	1,0406	---	---	---	---

Tabela 5.18: List-Scheduling com primeira prioridade largura

12	0	1	2	3	4	9	10	11	12
Di	1,0970	1,0970	---	---	---	1,0882	---	---	---
Ir	---	1,0797	1,0797	---	---	1,0781	---	---	---
KPSG	---	1,0622	---	1,0622	1,0622	---	---	---	---
Ran	1,0361	1,0361	---	---	1,0361	---	---	---	---

Tabela 5.19: List-Scheduling com primeira prioridade alap menos maior aresta incidente

primeiro teste, escolheu os melhores algoritmos para cada classe de aplicação, independentemente de suas primeiras prioridades. Dessa forma, foram escolhidos 24 algoritmos para as 8 classes de aplicações, sendo 6 algortimos para a classe Di, 6 para a classe Ir, 3 para a classe KPSG e 9 para a classe Ran. Os algoritmos selecionados foram aqueles que alcançaram os dois melhores valores de qualidade gerados para cada classe de aplicação. Neste caso, a comparação foi feita entre pares de primeira e segunda prioridade, para cada tipo de aplicação, ao longo das diferentes tabelas.

O terceiro e último teste teve a mesma configuração de entrada do segundo teste. Neste entretanto, ao contrário do segundo, os experimentos foram executados de uma única vez e não divididos por classe de aplicação. Isto permitiu observar o comportamento dos algoritmos para todos os tipos de aplicação e não para alguns tipos em específico. Através da observação deste comportamento foi possível identificar os algoritmos que, independentemente da classe da aplicação, geram resultados que estão sempre entre os melhores. Diferentemente da apresentação do resultados anteriores, a apresentação do resultado do terceiro teste é feita através de uma única tabela, que é a tabela 5.20, onde a primeira coluna e a primeira linha indicam as primeiras e segundas prioridades utilizadas respectivamente.

De forma análoga ao segundo teste nenhuma heurística se destacou. Assim, as heurísticas que obtiveram os melhores resultados foram selecionadas. Isso ressalta a observação de que não existe uma única heurística extremamente eficiente para todas as classes de aplicação.

5.4 Análise dos Resultados

Como o conjunto de testes foi realizado em cascata, ou seja, os melhores resultados de um teste serviram como entrada para o próximo teste, a análise dos resultados teve que ser feita em meio a execução dos testes e não quando todos os testes fossem gerados. Isso permitiu que alguns parâmetros de entrada fossem eliminados, uma vez que seus resultados não se alteravam.

	0	1	2	3	4	9	10	11	12
1	1,0444	---	1,0422	---	---	---	---	1,0422	---
2	---	1,0742	---	---	---	1,0724	1,0721	---	---
3	1,0735	---	---	1,0735	---	1,0781	---	---	---
4	1,0444	---	1,0422	---	---	---	---	1,0422	---
9	---	1,0657	---	---	1,0657	---	---	1,0633	---
10	---	---	1,0397	---	1,0422	---	---	1,0395	---
11	---	---	---	---	1,0491	1,0476	1,0491	---	---
12	---	1,0485	---	---	1,0485	1,0479	---	---	---

Tabela 5.20: Resultados do terceiro teste

Esta situação ocorreu no primeiro teste para o caso das aplicações da classe *intree* e *outtree* que obtiveram o valor 1 para todas as qualidades.

Para estas duas classes de aplicações os *makespans* gerados foram iguais para todas a configurações de experimentos. Isso ocorreu pelo fato de que todas as prioridades utilizadas, exceto conível e número de sucessores, se baseiam na prioridade nível e esses tipos de aplicações são altamente regulares. Dessa forma, quando o algoritmo List-Scheduling escolhia a tarefa livre com maior prioridade, a mesma tarefa era escolhida e conseqüentemente o escalonamento era o mesmo, resultando em *makespans* iguais. Para a prioridade conível o resultado obtido foi o mesmo pelo fato de que para esta prioridade o algoritmo List-Scheduling escolhe a tarefa livre com menor valor de conível para ser escalonada, diferentemente das demais prioridades onde a tarefa com maior prioridade é escolhida. Assim, a tarefa livre escolhida em determinado momento com menor valor de conível é a mesma tarefa com maior valor de nível, resultando em escalonamentos iguais para as duas prioridades no caso de aplicações regulares. Já a prioridade número de sucessores obteve o mesmo valor para o *makespan* em virtude do formato das aplicações que, representadas por grafos, são árvores binárias completas. Isso faz com que as tarefas tenham prioridade 2 e conseqüentemente sejam escalonadas na mesma ordem em que os nós de uma ávore são visitados em uma busca em largura, gerando um resultado igual ao escalonamento gerado pela prioridade nível. A eliminação destas duas classes acelerou o processo de refinamento da busca pelos melhores algoritmos.

O objetivo do primeiro teste foi fazer um filtro inicial no grande volume de algoritmos existentes, que somavam 72 algoritmos. Através da seleção das três melhores segundas prioridades para cada tipo de aplicação, obteve-se apenas 24 algoritmos, possivelmente distintos, para cada classe de aplicação.

O segundo teste foi executado em quatro fases, uma para cada tipo de aplicação (Di, Ir, KPSG, Ran). O objetivo deste teste foi definir, para cada classe de aplicação, os melhores algoritmos de escalonamento do tipo List-Scheduling. A escolha teve como base os dois melhores

valores médios da qualidade obtidos para cada classe de grafo. Assim, os algoritmos que atigiram esses valores foram selecionados sem restrição de quantidade de algoritmos por classe de aplicação. Desse modo, 24 pares de primeira e segunda prioridades foram escolhidos no total.

À partir do resultado do último teste foram escolhidos 7 diferentes algoritmos. Eles foram aqueles que obtiveram os melhores resultados para a maioria dos tipos de aplicação. O objetivo da escolha destes é oferecer um algoritmo de escalonamento de tarefas eficiente a usuários que possuam aplicações que não se encaixam nas classes definidas na ferramenta.

5.5 Conclusões

Através dos três grandes grupos de testes que foram realizados, foi possível definir as prioridades que geraram os melhores resultados na média para cada classe de aplicação e àquelas prioridades mais genéricas que mantiveram a qualidade do resultado independentemente do formato da aplicação. A redução da quantidade de heurísticas de 72 para, na média, 6 por classe de aplicação, simplifica o trabalho de determinação de um bom escalonamento. Em virtude dos resultados obtidos nos dois primeiros testes e valendo-se da legenda definida na Tabela 5.3, é possível listar os melhores algoritmos do tipo *List-Scheduling*, obtidos após um conjunto de 254740 experimentos, na Tabela 5.21.

Através da Tabela 5.5 é possível dizer que as melhores prioridades foram lbnível (10), nível (1) e alap (4), e que as melhores segundas prioridades foram conível (2), número de sucessores (11) e alap (4). A presença das prioridades lbnível, nível e conível entre as melhores, independentemente de primeira ou segunda prioridade, é compreensível, uma vez que estas prioridades são muito semelhantes. Por outro lado, a prioridade número de sucessores obteve bons resultados na medida que escolhe as tarefas que são as maiores geradoras de dependência para serem escalonadas, aumentando o número de tarefas livres. É válido destacar que os resultados foram obtidos através de médias e, dessa forma, não é possível garantir que esses melhores algoritmos sempre gerarão os melhores *makespans* para qualquer aplicação.

Os algoritmos do terceiro teste que atenderam o nível de flexibilidade e as requisições apresentadas foram os seguintes: List-Scheduling(10, 11), List-Scheduling(10, 2), List-Scheduling (10, 4), List-Scheduling(1, 2), List-Scheduling(1, 11), List-Scheduling(4, 2) e List-Scheduling (4, 11). As prioridades aqui selecionadas serão utilizadas no próximo capítulo para a proposta de uma nova técnica de escalonamento de tarefas implementada no Portal EasyGrid.

Tipo de aplicação	Algoritmo
Di	List-Scheduling(3,0) List-Scheduling(3,3) List-Scheduling(3,9) List-Scheduling(9,0) List-Scheduling(9,3) List-Scheduling(9,9)
Ir	List-Scheduling(9,1) List-Scheduling(9,4) List-Scheduling(9,12) List-Scheduling(12,1) List-Scheduling(12,2) List-Scheduling(12,9)
KPSG	List-Scheduling(10,2) List-Scheduling(10,11) List-Scheduling(10,12)
Ran	List-Scheduling(1,1) List-Scheduling(1,2) List-Scheduling(1,11) List-Scheduling(4,1) List-Scheduling(4,2) List-Scheduling(4,11) List-Scheduling(10,1) List-Scheduling(10,2) List-Scheduling(10,11)

Tabela 5.21: Melhores Prioridades

Capítulo 6

Estratégia Proposta

O Portal EasyGrid também poupa o usuário das preocupações necessárias para executar uma aplicação paralela no ambiente distribuído através da transformação desta em uma aplicação auto-adaptável ao ambiente de execução. Para que esta transformação seja feita de maneira a tirar o melhor proveito do ambiente paralelo é preciso obter um bom escalonamento estático da aplicação do usuário. Neste capítulo será apresentada uma nova estratégia de escalonamento estático de tarefas desenvolvida com base nos resultados dos experimentos computacionais e a aplicabilidade deste algoritmo no Portal EasyGrid.

6.1 MHLSS

Após o desenvolvimento da nova metodologia de execução distribuída do Portal (Capítulo 4) e a realização, captura e análise de milhares de experimentos das heurísticas conhecidas (Capítulo 5), o passo seguinte passou a ser a criação de uma estratégia de escalonamento a partir da conclusão de todo esse trabalho. Sendo assim, o *Multiple Heterogeneous List Scheduling Strategy*, ou simplesmente MHLSS, foi desenvolvido, em linguagem C++, incorporando as melhores heurísticas encontradas nos experimentos com o objetivo de sempre gerar bons escalonamentos para qualquer tipo de aplicação paralela.

Inicialmente, observando os resultados finais dos experimentos computacionais, verificou-se que eles são separados por classes de GAD e que, para cada uma dessas, existem alguns algoritmos que fornecem os melhores escalonamentos. Com base nessa observação, a estrutura básica do MHLSS foi encontrada e é apresentada a seguir:

- Descobrir qual a classe da aplicação, através do conhecimento prévio de sua formação;
- Executar as melhores heurísticas para a classe descoberta.

A partir dessa estrutura básica foram levantados os seguintes pontos: a execução dos algoritmos deveria ser concorrente e o melhor escalonamento seria encontrado pela comparação de todos os escalonamentos gerados. O primeiro ponto foi solucionado através da criação de diversas *threads* (biblioteca pthread do Sistema Operacional Fedora Core 2 [Red Hat 2004]) no MHLSS, sendo que cada uma delas ficou responsável pela execução de uma heurística. Logo, se existirem n heurísticas, existirão n *threads*. É válido destacar que, atualmente, a quantidade de melhores algoritmos por classe de GAD é pequena (média de seis para cada classe) e a implementação com *threads* é satisfatória, entretanto à medida que novos algoritmos forem sendo desenvolvidos e incluídos no Portal EasyGrid esta quantidade pode aumentar consideravelmente. Caso isso venha a ocorrer, é possível desenvolver uma aplicação MPI *system-aware* que utilize o middleware EasyGrid [Nascimento et al. 2005], de forma análoga à aplicação apresentada no Capítulo 4, para gerenciar e realizar a execução distribuída dos algoritmos. O segundo ponto utilizou o conhecimento prévio do formato do arquivo de saída dos algoritmos de escalonamento (Apêndice A) para capturar todos os makespans e, posteriormente, selecionar o melhor.

Além dos pontos descritos, a seguinte pergunta surgiu: quais algoritmos deveriam ser executados caso não fosse possível especificar a classe da aplicação do usuário? Foi preciso retornar aos resultados dos experimentos para observar que algumas classes (Randômica, PSG e Irregular) não tinham um padrão de formação e existia um resultado que englobava todas as classes ao mesmo tempo. Dessa forma, o MHLSS passou a executar todas as melhores heurísticas das classes Randômica, PSG e Irregular e do resultado que abrange todas as classes para aplicações que não seria possível descobrir o tipo.

Após responder à pergunta e solucionar os pontos levantados, foi possível concluir o desenvolvimento do MHLSS. O algoritmo básico é o apresentado abaixo:

Algoritmo 2 : *MHLSS*

1 Descubra qual a classe do GAD. Seja C esta classe;
2 Selecione de acordo com C os algoritmos a serem executados;
3 Execute os algoritmos concorrentemente;
4 Encontre o que forneceu o melhor makespan;

6.2 Aplicação no Portal EasyGrid

Uma importante funcionalidade oferecida pelo Portal EasyGrid é a execução de uma aplicação MPI escolhida pelo usuário. Inicialmente, o usuário deve fornecer os arquivos de GAD e de custo (Apêndice A) que representam a aplicação (Figura 6.1). Terminada esta etapa, o

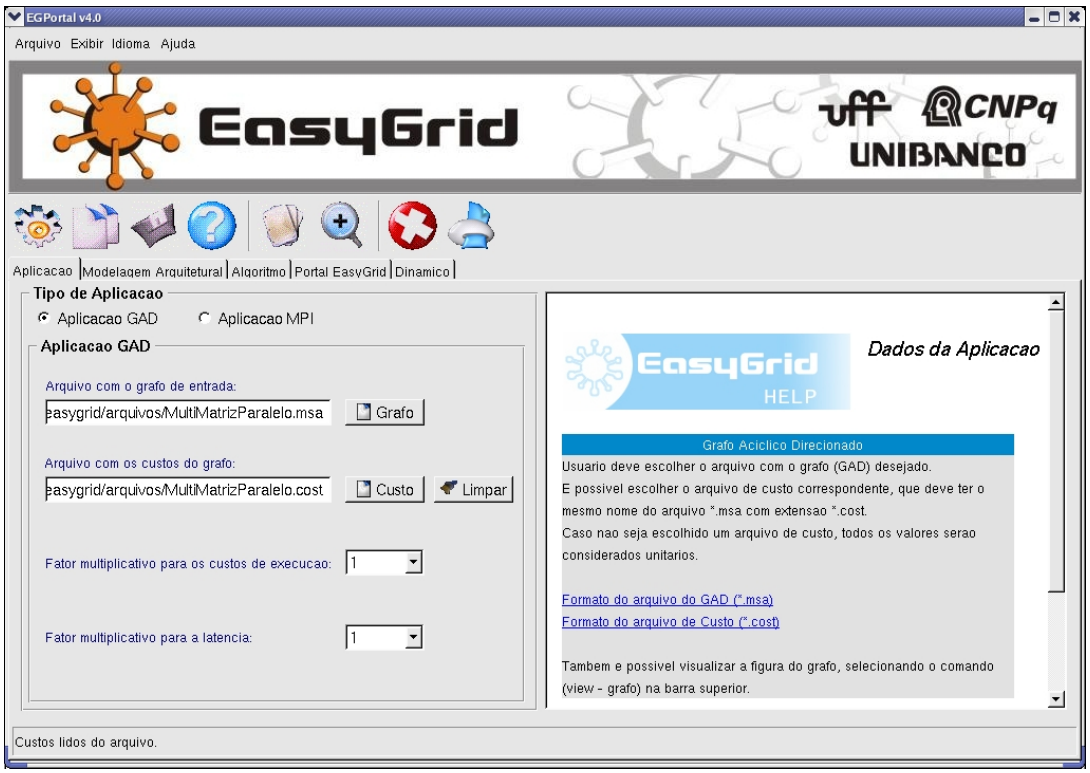


Figura 6.1: Escolha do GAD que representa a aplicação

usuário escolhe o código fonte, em C/C++, de sua aplicação MPI (Figura 6.2). O usuário pode visualizar o código para ter certeza que selecionou o arquivo correto.

Neste momento, o usuário deve clicar na aba *MODELO REAL* (Figura 6.3) para que, assim como no Capítulo 4, selecione o ambiente grade onde a aplicação será executada. Novamente, o usuário deve especificar o universo total da grade, os recursos ao qual possui acesso e autenticar-se no *globus toolkit* [Foster e Kesselman 1997] caso isso ainda não tenha sido feito. Depois disso, o usuário precisa especificar para qual versão do MPI a sua aplicação foi desenvolvida, as possíveis opções são MPI-LAM, MPI-CHG2 [Message Passing Forum 1995] ou alguma outra. Utilizando o *globus toolkit* [Foster e Kesselman 1997] após o clique no botão *STATUS DO GRID*, verificamos o status (por ex. carga do processador e memória disponível) de cada recurso do ambiente para que, com essas informações, o usuário possa selecionar os recursos adequados à execução. Feito isso o usuário deve clicar no botão *MODELAR A GRADE* para utilizar um modelador [Mendes 2004] para obter o poder computacional de cada recurso. Este modelador cria um arquivo representando a arquitetura do ambiente grade (Apêndice A) escolhido pelo usuário.

Posteriormente, o usuário deve selecionar a aba *PORTAL EASYGRID* (Figura 6.4) para configurar as últimas opções necessárias à execução distribuída. Em primeiro lugar, o usuário deve

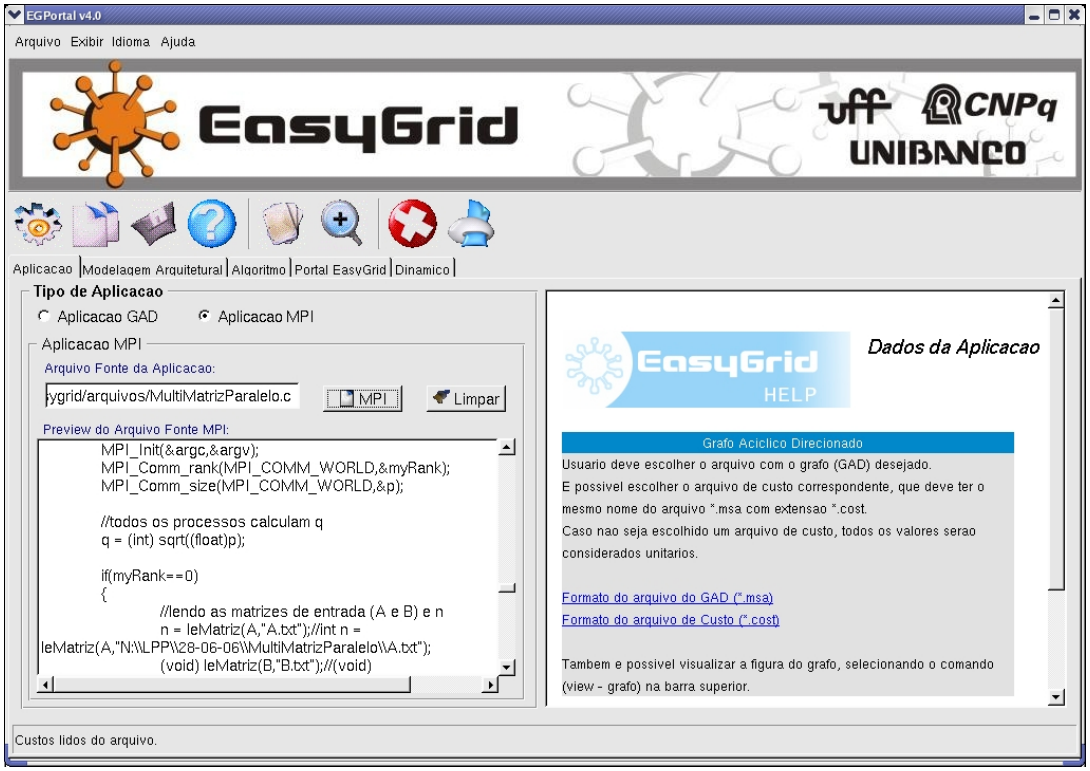


Figura 6.2: Escolha da aplicação

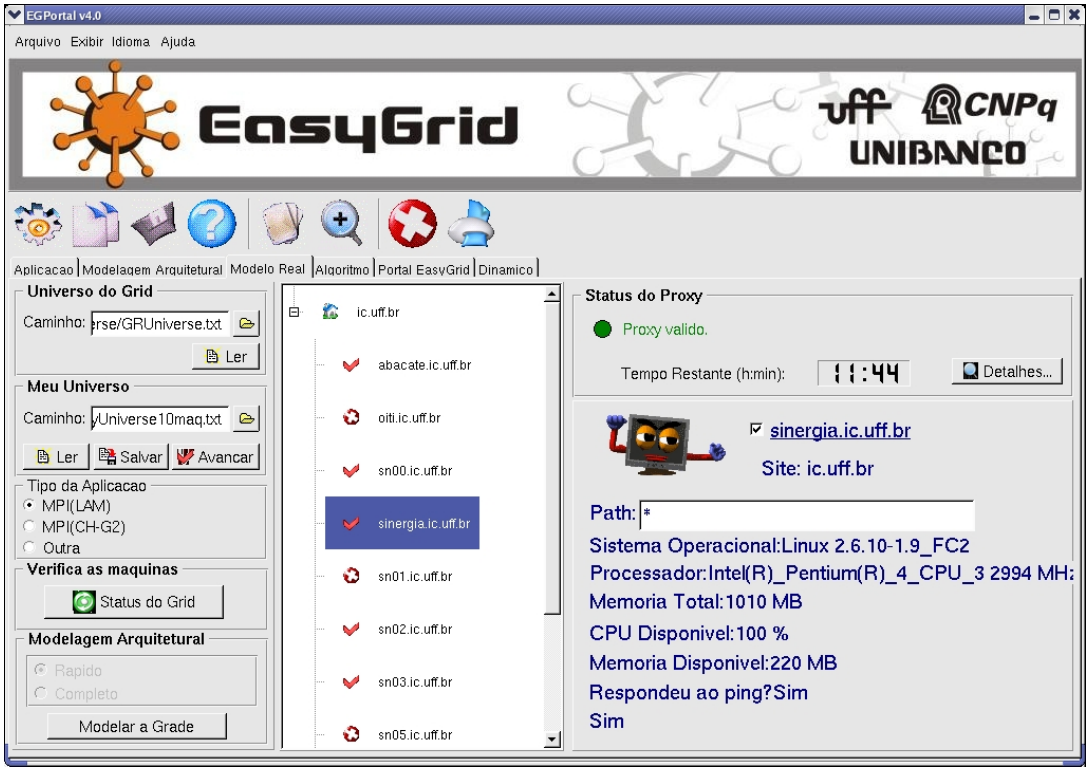


Figura 6.3: Modelando a grade que será utilizada

escolher se o GAD e seu custo especificados serão utilizados como representação de sua aplicação. Em uma versão futura do Portal será possível gerar um GAD a partir de uma aplicação paralela e por isso esta opção aqui detalhada se faz necessária.

Agora o usuário já pode obter o escalonamento estático de sua aplicação. Ao clicar no botão *ESCALONAR* o Portal faz uso do algoritmo de escalonamento estático MHLSS para gerar um bom escalonamento. Após a geração do escalonamento, o usuário pode verificar se realmente deseja prosseguir utilizando esse escalonamento através do Diagrama de Gantt (Figura 6.5). Nesse diagrama é possível verificar, para cada tarefa da aplicação, a ordem e em quais processadores foi alocada. Caso o usuário não fique satisfeito com o escalonamento visualizado, é possível retornar ao passo de modelagem do seu ambiente grade e refazê-lo para tentar encontrar um escalonamento satisfatório.

Terminada essa validação, é preciso compilar e executar a aplicação. Na compilação é possível que o usuário escolha se fará uso do middleware EasyGrid [Nascimento et al. 2005] (opção *com Middleware EsyGrid*) e do escalonamento obtido pelo algoritmo MHLSS (opção *EasyGrid*), além da possibilidade de informar algumas diretivas de compilação. Depois o usuário clica no botão *CRIAR EXECUTÁVEL* para poder compilar. Na execução, deve ser selecionado se o Portal deve enviar os arquivos necessários para as máquinas da grade caso seja a primeira execução. Após esta seleção é preciso clicar no botão *EXECUTAR* para finalmente executar a aplicação no ambiente distribuído.

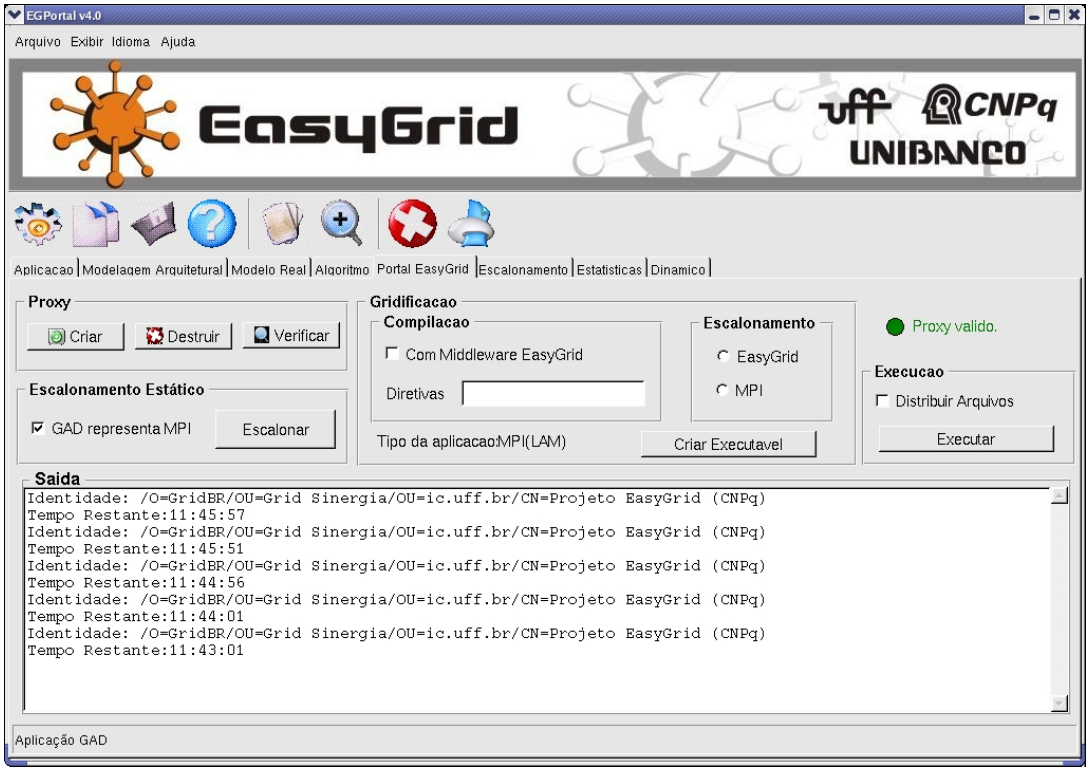


Figura 6.4: Geração do escalonamento estático

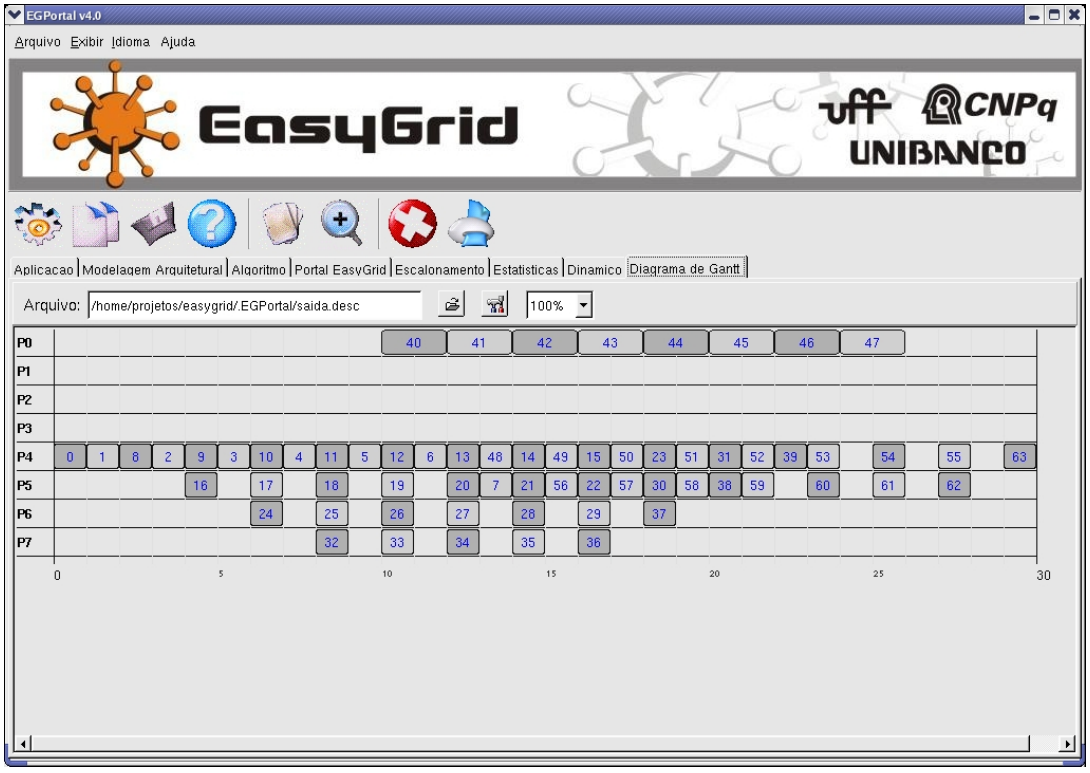


Figura 6.5: Diagrama de Gantt

Capítulo 7

Conclusões e Trabalhos Futuros

Os objetivos deste trabalho foram desenvolver uma nova técnica para execução de testes no Portal EasyGrid, de tal forma que os resultados fossem obtidos de uma forma mais rápida, realizar um estudo sobre as prioridades existentes no Portal EasyGrid para o algoritmo List-Scheduling e com os resultados obtidos propor uma nova estratégia de escalonamento de tarefas.

Um grande problema enfrentado inicialmente pelos usuários do Portal EasyGrid era a execução de experimentos que demandava muito tempo. A nova abordagem apresentada neste trabalho além de simplificar o processo de configuração dos testes e permitir que os usuários analisassem seus resultados através de filtros sem a necessidade de refazer sub-testes para obterem resultados específicos, reduziu consideravelmente o tempo total de execução dos experimentos.

Através desta nova abordagem de execução de testes, milhares de experimentos foram executados afim de analisar as diferentes prioridades utilizadas pelo algoritmo List-Scheduling. Este estudo avaliou as prioridades que obtiveram os melhores resultados para cada tipo de aplicação e determinou as heurísticas mais genéricas, ou seja, que na média obtiveram bons resultados para todos os tipos de aplicações.

Por fim foi desenvolvida a estratégia MHLSS para o escalonamento de tarefas. Esta estratégia utiliza a análise dos resultados dos testes para executar as aplicações dos usuários de uma forma mais eficiente. Através da identificação da classe da aplicação do usuário, o MHLSS determina os algoritmos de escalonamento que obtiveram os melhores resultados para esta classe e os utiliza para escalonar as tarefas da aplicação.

Os próximos trabalhos a serem desenvolvidos são:

- Considerar a quantidade de processadores utilizados em um escalonamento como um critério a mais de avaliação das heurísticas. Sendo assim, o problema de avaliação das

melhores heurísticas se tornará multiobjetivo por considerar dois parâmetros de avaliação.

- Inclusão de um módulo no Portal EasyGrid para auxiliar o usuário na transformação de sua aplicação paralela em um grafo acíclico direcionado.

- Melhoria do algoritmo de escalonamento de tarefas MHLSS através da inclusão de algumas técnicas mais sofisticadas, por exemplo replicação de tarefas e *look-ahead*, de escolha do processador no qual a próxima tarefa livre será escalonada.

- Desenvolvimento de um servidor, em linguagem Java, que terá somente as funcionalidades de execução da aplicação de um usuário em uma grade computacional, e de uma interface, em linguagem C++, que funcionará como um cliente deste servidor. Estas aplicações se comunicarão através da tecnologia Web Services para que o usuário possa utilizar a grade computacional de onde estiver, necessitando apenas da interface cliente e de uma conexão com a internet.

APÊNDICE A – Modelos dos arquivos

Neste apêndice são apresentados os modelos dos arquivos de entrada e saída utilizados pelo Portal Easygrid. São encontrados os seguintes tipos de arquivos: aplicação (GAD), custo dos GADs, arquitetura da grade, universo total do ambiente distribuído, universo acessado pelo usuário e saída dos algoritmos de escalonamento. A seguir é exibido um exemplo explicativo para cada um desses tipos.

A.1 Grafo (GAD)

Formato do arquivo, com extensão .msa, que representa uma aplicação.

```
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>
```

A.2 Custo do GAD

Formato do arquivo, com extensão .cost, que representa o custo de uma aplicação.

```
16      <número de tarefas: 16>
2       <custo da tarefa 0: 2>
5       <custo da tarefa 1: 5>
3       <custo da tarefa 2: 3>
1       <custo da tarefa 3: 1>
4       <custo da tarefa 4: 4>
1       <custo da tarefa 5: 1>
5       <custo da tarefa 6: 5>
3       <custo da tarefa 7: 3>
6       <custo da tarefa 8: 6>
1       <custo da tarefa 9: 1>
7       <custo da tarefa 10: 7>
7       <custo da tarefa 11: 7>
8       <custo da tarefa 12: 8>
1       <custo da tarefa 13: 1>
9       <custo da tarefa 14: 9>
2       <custo da tarefa 15: 2>
1 5     <peso das arestas: entre 0 e 1: 1 – entre 0 e 4: 5>
3 4     <peso das arestas: entre 1 e 2: 3 – entre 1 e 5: 4>
7 6     <peso das arestas: entre 2 e 3: 7 – entre 2 e 6: 6>
1       <peso das arestas: entre 3 e 7: 1>
1 1     <peso das arestas: entre 4 e 5: 1 – entre 4 e 8: 1>
4 1     <peso das arestas: entre 5 e 6: 4 – entre 5 e 9: 1>
9 8     <peso das arestas: entre 6 e 7: 9 – entre 6 e 10: 8>
2       <peso das arestas: entre 7 e 11: 2>
1 1     <peso das arestas: entre 8 e 9: 1 – entre 8 e 12: 1>
6 3     <peso das arestas: entre 9 e 10: 6 – entre 9 e 13: 3>
2 2     <peso das arestas: entre 10 e 11: 2 – entre 10 e 14: 2>
5       <peso das arestas: entre 11 e 15: 5>
1       <peso das arestas: entre 12 e 13: 1>
9       <peso das arestas: entre 13 e 14: 9>
4       <peso das arestas: entre 14 e 15: 4>
        <peso das arestas:  ou -1>
```

A.3 Arquitetura

Formato do arquivo, com extensão .comm, que representa a arquitetura da grade.

```
4                       <número de máquinas (n): 4>
1 sinergia.ic.uff.br 1 2    <numMaq: 0 – fh: 1 – nomeMaq: sinergia – Os: 1 – Or: 2>
1 sn16.ic.uff.br 2 3        <numMaq: 1 – fh: 1 – nomeMaq: sn16 – Os: 2 – Or: 3>
2 banana.ic.uff.br 4 1      <numMaq: 2 – fh: 2 – nomeMaq: banana – Os: 4 – Or: 1>
2 pitanga.ic.uff.br 1 1     <numMaq: 3 – fh: 2 – nomeMaq: pitanga – Os: 1 – Or: 1>
0 1 1 1
1 0 1 1                 <matriz n x n da latência de comunicação >
1 1 0 1                 <a latência de i para j é dada pelo elemento a[i][j]
1 1 1 0
```

A.4 Universo

Formato do arquivo, com extensão .txt, que representa todo o universo do ambiente paralelo.

```
13                          <total de máquinas: 13>
S ic.uff.br 8 1             <tipo: site (S) – nomeSite: ic.uff.br – numMaqsSite: 29 – NFS: 1>
M abacate.ic.uff.br 1 1 1   <tipo: máquina (M) – nomeMaq: abacate – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 1>
M caju.ic.uff.br 1 1 0      <tipo: máquina (M) – nomeMaq: caju – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 0>
M goiaba.ic.uff.br 1 1 0    <tipo: máquina (M) – nomeMaq: goiaba – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 0>
M melancia.ic.uff.br 1 1 0  <tipo: máquina (M) – nomeMaq: melancia – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 0>
M pitanga.ic.uff.br 1 1 0   <tipo: máquina (M) – nomeMaq: pitanga – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 0>
M manga.ic.uff.br 1 1 0     <tipo: máquina (M) – nomeMaq: manga – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 0>
M sinergia.ic.uff.br 1 1 1  <tipo: máquina (M) – nomeMaq: sinergia – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 1>
M sn16.ic.uff.br 1 1 0      <tipo: máquina (M) – nomeMaq: sn00 – MPICH-G2: 1 – MPILAM: 1 – servidorArq: 0>
S par.inf.puc-rio.br 5 0    <tipo: site (S) – nomeSite: par.inf.puc-rio.br – numMaqsSite: 5 – NFS: 0>
M n00.par.inf.puc-rio.br 1 0 1  <tipo: máquina (M) – nomeMaq: n00 – MPICH-G2: 1 – MPILAM: 0 – servidorArq: 1>
M n01.par.inf.puc-rio.br 1 0 1  <tipo: máquina (M) – nomeMaq: n01 – MPICH-G2: 1 – MPILAM: 0 – servidorArq: 1>
M n02.par.inf.puc-rio.br 1 0 1  <tipo: máquina (M) – nomeMaq: n02 – MPICH-G2: 1 – MPILAM: 0 – servidorArq: 1>
M n03.par.inf.puc-rio.br 1 0 1  <tipo: máquina (M) – nomeMaq: n03 – MPICH-G2: 1 – MPILAM: 0 – servidorArq: 1>
M n04.par.inf.puc-rio.br 1 0 1  <tipo: máquina (M) – nomeMaq: n04 – MPICH-G2: 1 – MPILAM: 0 – servidorArq: 1>
```

A.5 Meu Universo

Formato do arquivo, com extensão .txt, que representa o universo da grade acessado pelo usuário.

```
abacate.ic.uff.br *         <nomeMaq: abacate – pathMPI: *>
caju.ic.uff.br *            <nomeMaq: caju – pathMPI: *>
sinergia.ic.uff.br *        <nomeMaq: sinergia – pathMPI: *>
manga.ic.uff.br *           <nomeMaq: manga – pathMPI: *>
n00.par.inf.puc-rio.br *    <nomeMaq: n00 – pathMPI: *>
n04.par.inf.puc-rio.br *    <nomeMaq: n04 – pathMPI: *>
```

A.6 Saída do escalonamento

Formato do arquivo, com extensão .desc, que representa a saída de um algoritmo de escalonamento de tarefas.

16	<Número de tarefas: 16>
4	<Número de processadores: 4>
15	<Makespan: 15>
0	<Tempo em segundos: 0>
544	<Tempo em milisegundos: 544>
0 sinergia.ic.uff.br 15	<numMaq: 0 – nomeMaq: sinergia.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 abacate.ic.uff.br 1	<numMaq: 1 – nomeMaq: abacate.ic.uff.br – Número de tarefas executadas: 1>
12 7 9	<Tarefa: 12 – Tempo de início: 7 – Tempo de fim: 9>
2 sn16.ic.uff.br 0	<numMaq: 2 – nomeMaq: sn16.ic.uff.br – Número de tarefas executadas: 0>
3 pitanga.ic.uff.br 0	<numMaq: 3 – nomeMaq: pitanga.ic.uff.br – Número de tarefas executadas: 0>

APÊNDICE B – Publicação e Prêmio

A publicação e o prêmio recebidos através deste trabalho estão descritos neste apêndice.

B.1 Publicação

Henrique B. Rodrigues, Hildebrando Trannin, Alexandre C. Sena, Vinod E.F. Rebello, *Usando Grades Computacionais para a Avaliação de Heurísticas de Escalonamento de Tarefas*, VI Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2005), pp218-221, Rio de Janeiro, RJ, Brasil, Outubro de 2005.

B.2 Prêmio

Vencedores do prêmio UFF Vasconcellos Torres de Ciência e Tecnologia 2005 do XV Seminário de Iniciação Científica da Universidade Federal Fluminense (UFF), ocorrido de 07 a 11 de novembro de 2005, na área de Engenharias pelo projeto *Usando Grades Computacionais para Avaliação de Heurísticas de Escalonamento*.

Referências

[500 2004] 500, T. **TOP500 Supercomputing Sites**. site, http://www.top500.org, 2004.

[Abramson et al. 2000] Abramson, D.; Giddy, J. e Kotler, L. **High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?** In: *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, p. 520–528, 2000.

[Aggarwal et al. 1989] Aggarwal, A.; Chandra, A. K. e Snir, M. **On communications latency in PRAM Computations**. *Technical Report Rerc 14973, IBM T. J. Watson Research Center, YorkTown Heights, NY, USA Concurrency and Computation: Pratice and Experience*, September 1989.

[Boeres et al. 2006] Boeres, C.; Fonseca, A. A.; Mendes, H. A.; Menezes, L. T.; Moura, N. T.; Silva, J. A.; Vianna, B. A. e Rebello, V. E. F. **An EasyGrid Portal for Scheduling System-aware Applications on Computational Grids**. *Concurrency and Computation: Pratice and Experience*, John Wiley and Sons Ltd, 2006.

[Boeres e Rebello 2002] Boeres, C. e Rebello, V. E. F. **On solving the static task scheduling problem for real machine**. *Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications*, Kluwer Academic Publishers, p. 53–84, 2002.

[Coffman Jr. 1976] Coffman Jr., E. **Computer and Job Shop Scheduling Theory**. New York-John Wiley, 1976.

[Figueira e Kraus 2006] Figueira, C. e Kraus, F. **Framework de Simulação de Escalonadores Dinâmicos Distribuídos**. *Monografia Final de Curso, Graduação em Ciência da Computação*, Agosto 2006.

[Fonseca e Vianna 2004] Fonseca, A. e Vianna, B. **Um Ambiente para Desenvolvimento e Avaliação de Algoritmos de Escalonamento para Grades tradicionais**. *Monografia Final de Curso, Graduação em Ciência da Computação*, Agosto 2004.

[Foster e Kesselman 1997] Foster, I. e Kesselman, C. **Globus: A Metacomputing Infrastructure Toolkit**. *The International Journal of Supercomputer Applications and High Performance Computing*, v. 11, n. 2, p. 115–128, Summer 1997.

[Red Hat 2004] Red Hat, I. **Fedora Core 2 Release Notes**. site, http://www.gnu.org/licenses/fdl.html, 2004.

[Hwang et al. 1989] Hwang, J. J.; Chow, Y. C.; Anger, F. D. e Lee, C. Y. **Scheduling Precedence Graphs in Systems with Interprocessor Communication Times**. *SIAMJ Comp.*, v. 18, p. 244–257, April 1989.

[Khan et al. 1994] Khan, A.; McCreary, C. e Jones, M. **Comparison of Multiprocessor Scheduling Heuristics**. In: IEEE COMPUTER SOCIETY PRESS. *Proc. of the 8th International Parallel Processing*, Cancun, Mexico, II, p. 243–250, April, 1994.

[Kwok e Ahmad 1996] Kwok, Y.-K. e Ahmad, I. **Dynamic Critical-Path scheduling: an effective technique for allocating tasks graphs to multiprocessors**. *IEEE transactions on Parallel and Distributed Systems*, v. 7, n. 5, May 1996.

[Kwok e Ahmad 1999a] Kwok, Y.-K. e Ahmad, I. **Benchmarking and Comparison of the task graph scheduling algorithms**. *Journal of Parallel and Distributed Computing*, v. 59, n. 3, p. 381–422, Dec. 1999.

[Kwok e Ahmad 1999b] Kwok, Y.-K. e Ahmad, I. **Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors**. *ACM Computing Surveys*, v. 31, n. 4, Dec. 1999.

[Mendes 2004] Mendes, H. A. **HLogP: Um Modelo de Escalonamento para a Execução de Aplicações MPI em Grades Computacionais**. Dissertação (Mestrado), Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2004.

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

[Nascimento et al. 2005] Nascimento, A. P.; Sena, A. C.;da Silva, J. A.; Vianna, D. Q. C.; Boeres, C. e Rebello, V. E. F. **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.

[Rodrigues et al. 2005] Rodrigues, H. B.; Trannin, H.; Sena, A. C. e Rebello, V. E. F. **Usando Grades Computacionais para Avaliação de Heurísticas de Escalonamento de Tarefas**. In: *VI Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD - 2005)*, : IEEE Computer Society Press, October, 2005.

[Tam e Wang 1999] Tam, A. e Wang, C. **Realistic commmunication model for parallel computing on cluster**. *The proceedings of the 1st IEEE INTERNATIONAL WORKSHOP ON CLUSTER COMPUTING, Melbourne, Australia*, December 1999.

[Topcuoglu et al. 2002] Topcuoglu, H.; Hariri, S. e Wu, M. **Performance effective and low-complexity task scheduling for heterogeneous computing**. *IEEE transactions on Parallel and Distributed Systems*, v. 13, n. 3, March 2002.

[Wu et al. 2000] Wu, M.-Y.; Shu, W. e Zhang, H. **Segmented Min-Min: A Static Mappin Algorithm fo Meta-tasks on Heterogeneous Computing Systems**. 2000.