Um Ambiente para o Desenvolvimento e Avaliação de Algoritmos de Escalonamento para Grades Computacionais*

A.A. Fonseca[1], B.A. Vianna[2], N.T. Moura[3], L.T. Menezes[1], H.A. Mendes
C. Boeres e V.E.F. Rebello
Instituto de Computação, Universidade Federal Fluminense (UFF)
{afonseca, bvianna, nmoura, ltoscano, hmendes, boeres, vinod}@ic.uff.br

Resumo

O objetivo de uma grade computacional é o de agregar uma coleção de recursos geograficamente distribuídos para oferecer poder computacional para aplicações. Contudo, permanece como desafio a exploração eficiente do desempenho deste ambiente, devido principalmente à natureza distribuída, compartilhada e heterogênea dos recursos. Escalonadores eficientes são fundamentais para que aplicações explorarem a potencialidade das grades. Este trabalho apresenta uma ferramenta para facilitar o desenvolvimento de, e a análise de desempenho oferecida por diferentes políticas de escalonamento de aplicações em ambientes grades.

1. Introdução

Grades Computacionais (*Computational Grids*) têm o potencial de se tornarem plataformas poderosas a serem utilizadas pela comunidade de computação distribuída, tanto científica quanto comercial, para a execução de aplicações de grande importância e alto teor computacional. Computação em grades adotou o nome e o conceito das redes elétricas de potência para capturar a noção de fornecimento eficiente de poder computacional, a um custo razoável, de acordo com a demanda, para qualquer um que precisar [8].

O objetivo da grade é o de agregar uma coleção de recursos distribuídos geograficamente, heterogêneos e compartilhados, sendo visto como um único recurso computacional. A disponibilidade deste tipo de ambiente representa uma abertura para novos ramos de pesquisa que previamente encontravam-se limitados e sem exploração por razões econômicas e práticas. As grades computacionais também podem ser vistas como uma alternativa aos supercomputadores para tornar a computação de alto desempenho mais acessível. A existência de redes de longa distância de alta velocidade e baixo custo faz com que computação de alto desempenho também seja acessível a usuários que não necessariamente possuem recursos suficientes localmente para executar suas aplicações.

Entretanto, ainda existem relativamente poucas aplicações que exploram tal poder computacional deste novo ambiente de forma realmente vantajosa. Atualmente, a maioria das aplicações habilitadas às grades tem sido escrita por especialistas de grades e não cientistas, engenheiros ou programadores comuns. Devido aos diversos tipos de recursos e ao comportamento dinâmico e instável das grades, desenvolver aplicações preparadas para executar de forma eficiente nesse ambiente ainda é um desafio. Isso reflete o grau de dificuldade encontrado, especialmente por leigos em computação, em propor versões das aplicações preparadas para executar de forma eficiente em grades computacionais. Evitar reverter essa situação seria motivo suficiente para inibir a aceitação das grades computacionais.

Usuários de sistemas paralelos encontram, com uma certa freqüência, uma grande dificuldade em atingir uma boa fração do pico de desempenho teórico dos sistemas paralelos e distribuídos. De fato, como as arquiteturas paralelas estão se tornando mais complexas e os softwares em constante evolução, um número crescente de problemas relacionados ao desempenho fazem a afinação de aplicações por parte do próprio programador (usuário) uma tarefa complexa e algumas vezes contra-intuitiva.

Por exemplo, a Grade Computacional GridRio [21, 22], em operação desde o final de 2002, está sendo utilizada por especialistas para desenvolver *middleware* voltado para ambientes grade e também por físicos, para executar aplicações cientificas. O poder computacional agregado (e compartilhado entre usuários de três instituições) atualmente é de 94 processadores com um limite de desempenho de mais de 300 GFLOPs, sendo que os processadores variam

* Este trabalho é financiado pelo CNPq (No. do processo 552205/2002-8)
[1] Bolsista de Iniciação Tecnológica Industrial do CNPq
[2] Bolsista de Iniciação Científica do Unibanco
[3] Bolsista de Iniciação Científica do CNPq/PIBIC (UFF)

desde Pentium II 233Mhz (com 128MB de memória) até Pentium IV 3.2 Ghz (1GB de memória) e bi-processadores [10]. Ainda neste ano, esperamos que a infra-estrutura do GridRio venha a atingir mais de 200 processadores e capacidade máxima de processamento de 0.8 TFLOPs. Em um ambiente tão variado, é extremamente difícil que usuários (tipicamente cientistas e engenheiros com pouco conhecimento do sistema) sejam capazes de decidir quais são os recursos mais apropriados para executar cada aplicação eficientemente.

O projeto *EasyGrid* [4, 21] tem como proposta resolver o desafio de habilitar aplicações para executar de forma eficiente em ambientes grades, evitando assim que esse trabalho árduo seja realizado pelo próprio usuário. O ambiente *EasyGrid* permite que programadores possam se concentrar na exploração do paralelismo para resolver o problema em si, deixando com que o ambiente gere a *aplicação grade* capaz de utilizar da melhor forma, os recursos disponíveis. O ambiente *EasyGrid* deve fazer com que todos os aspectos relacionados a grade sejam transparentes ao usuário. Para isso, aplicações paralelas são transformadas automaticamente em aplicações *system-aware*, ou seja, aplicações cientes do sistema e que se automodificam, adequando-o às mudanças dinâmicas ocorridas no ambiente de execução.

Inicialmente, o projeto *EasyGrid* focaliza aplicações paralelas escritas em MPI [19], devido ao seu grande uso em aplicações cientificas e na área da programação paralela. Para a aplicação obter um desempenho aceitável é crucial uma boa alocação dos processos nos processadores disponíveis.

Este trabalho se concentra no problema de desenvolver heurísticas de escalonamento apropriadas para o framework *EasyGrid* e grades computacionais, em geral. Para minimizar a complexidade de um escalonador dinâmico, utilizado durante a execução da aplicação e necessário neste ambiente devido ao seu dinamismo e heterogeneidade, é proposto um escalonador estático que fará o pré-escalonamento das tarefas, restando ao escalonador dinâmico apenas os ajustes necessários [3].

Para facilitar o desenvolvimento e análise de algoritmos de escalonamento de tarefas especificamente voltado para atender as características acima, uma ferramenta gráfica é proposta. Seus objetivos principais podem ser resumidos em:

- **[Projeto]** Permitir a investigação de abordagens e (combinações de) técnicas de escalonamento (por exemplo, replicação de tarefas, prioridades das tarefas, regras para escolher processadores).
- **[Avaliação]** A avaliação de novos algoritmos em termos de *makespan* (tempo de conclusão da aplicação paralela) e processadores utilizados. A avaliação experimental requer a comparação com outras heurísticas existentes para que assim, o comportamento dos novos algoritmos considerando várias instâncias seja traçado.
- **[Validação]** A verificação e validação dos escalonamentos gerados pelos algoritmos podem ser feitas por simulação ou via execução num ambiente grade real.

2. O problema de escalonamento de tarefas

O objetivo principal do escalonamento é a execução eficiente de uma aplicação paralela considerando as restrições impostas pelo sistema alvo. Ao se tratar de grades, é muito importante também determinar o uso eficiente dos recursos para as várias aplicações que venham a ser executadas em tal sistema, mas tendo em mente que cada uma dessas aplicações deve ser executada em tempo razoável.

A comunidade de escalonamento geralmente representa uma aplicação paralela através de um grafo acíclico direcionado (GAD), onde os vértices do grafo representam as tarefas e os arcos, as dependências de dados entre pares de tarefas e conseqüentemente comunicação. No entanto, se tarefas dependentes forem alocadas em um mesmo processador, o custo de comunicação associado à transferência de dados é considerado nulo. Deste modo, a alocação das tarefas nos processadores deve ser realizada de forma cuidadosa para que o grau de paralelismo da aplicação seja explorado ao máximo, resultando em um tempo de execução da aplicação mínimo.

Todavia, o problema de escalonar as tarefas de uma aplicação paralela tal que o tempo de execução do escalonamento seja mínimo, é NP-completo [24]. Mesmo com simplificações na aplicação e na máquina paralela, tais como: custo uniforme de execução das tarefas, custo inexistente de comunicação entre as tarefas e disponibilidade ilimitada de processadores, o problema permanece NP-completo exceto para alguns casos restritos [9]. Deste modo, heurísticas de escalonamento são propostas para se tentar obter bons escalonamentos em tempos razoáveis [16]. Em um ambiente de grade, o problema de escalonar tarefas torna-se ainda mais complexo devido à necessidade de um modelo de arquitetura que represente com sensibilidade o custo de comunicação, o dinamismo e a diversidade dos recursos.

Devido à característica dinâmica das grades já citada anteriormente, torna-se necessário um escalonamento de tarefas que possa ser modificado durante a execução da aplicação, ou seja, a utilização

de escalonamento dinâmico, obviamente, de sobrecarga mínima. Para reduzir esta sobrecarga, uma estratégia híbrida é aplicada, onde informações produzidas por um escalonador estático são utilizadas pelo escalonador dinâmico. A fase estática tem como objetivo ajustar a aplicação à configuração momentânea da máquina alvo, enquanto a fase dinâmica, adaptar o escalonamento às variações ocorridas durante o tempo de execução [17, 3].

Parte dos objetivos do projeto *EasyGrid* se resumem na busca de heurísticas e técnicas de escalonamento que sejam especificamente direcionadas as grades computacionais. Como não existe um consenso de qual modelo arquitetural é mais apropriado para ambientes de grade, atualmente o novo modelo de escalonamento LogP Heterogêneo (HLogP) [18] está sendo proposto e validado. Esse modelo é baseado no modelo LogP [6], já estabelecido por representar melhor a comunicação em sistemas distribuídos [13].

O presente trabalho tem por finalidade facilitar o projeto e avaliação de uma heurística de escalonamento estático rápida e eficiente, visando reduzir significativamente a carga de trabalho do escalonador dinâmico usado no *framework* EasyGrid [4]. Apesar da necessidade de adotar um modelo arquitetural mais real como o HLogP, o modelo de latência [20] também pode ser utilizado para avaliar com maior facilidade as principais heurísticas propostas na literatura. A abordagem *list scheduling* [11, 15] é escolhida devido à sua baixa complexidade aliada a resultados razoáveis de escalonamento, principalmente no que se diz respeito à minimização no uso de processadores.

2.1. *List scheduling* configurável

A classe de heurísticas do tipo *list scheduling* é composta de heurísticas de escalonamento na qual as tarefas de uma aplicação paralela são ordenadas em uma lista, segundo algum tipo de prioridade pré-determinada. As tarefas são posteriormente designadas, uma a uma, aos processadores de acordo com esta ordem na lista. Basicamente, esta metodologia segue a seguinte regra: a primeira tarefa livre da lista ordenada deve ser alocada em um processador ocioso tal que o seu tempo de conclusão seja o mais cedo possível. A estrutura básica do *list scheduling* está representada no framework a seguir:

```
Algoritmo List Scheduling
{
    Definir as prioridades das tarefas do GAD;
    Ordenar as tarefas livres por suas prioridades;
    Enquanto (∃ tarefas não escalonadas) faça {
        v = tarefa livre de maior prioridade;
        p_j = processador onde v termina mais cedo;
        Escalonar v no processador p_j ;
        Determinar as novas tarefas livres;      }
}
```

A principal diferença dentre os vários algoritmos desta classe está principalmente na prioridade utilizada para escolher a próxima tarefa a ser escalonada [16].

Foi desenvolvido um algoritmo configurável baseado na estratégia *list scheduling* para ambientes heterogêneos com um número limitado de processadores, que se baseia no modelo HLogP. A estratégia implementa uma lista *dinâmica* de tarefas e considera diferenças no poder computacional entre os processadores e também, diferentes latências e sobrecargas de comunicação. Obviamente, ambientes homogêneos, um número qualquer de processadores e modelos de comunicação mais simples como o modelo de latência, também podem ser utilizados.

É oferecida ao usuário a possibilidade de escolha de alternativas a serem utilizadas na busca por um melhor escalonamento. A seguir, algumas dessas opções podem ser destacadas:

- *Prioridades para escolha da próxima tarefa* — a maioria das funções utilizadas por algoritmos do tipo *list scheduling*, (*nível, co-nivel, ALAP, CP,* e outros [16]) como prioridades, foram adaptadas para ambientes heterogêneos. Por exemplo, o *nível* estático de uma tarefa, que é o custo do maior caminho da tarefa até um destino, é calculado utilizando o poder computacional médio da tarefa nos processadores disponíveis e o custo médio de comunicação [23];
- *Versões dinâmicas das prioridades* — durante o processo de escalonamento, à medida que tarefas são alocadas, suas prioridades podem mudar devido à eliminação dos custos de comunicação ou a determinação do custo de computação. Assim, a atualização das prioridades após o escalonamento de cada tarefa leva à obtenção de informações mais exatas sobre a real importância das tarefas nas decisões a serem tomadas nas iterações seguintes;
- *Três níveis de prioridade* — em caso de empate, existe a opção de utilizar uma segunda prioridade ao escolher a próxima tarefa a ser escalonada e ainda, persistindo empates, uma terceira prioridade.

Geralmente, empates são geralmente solucionados de forma aleatória [16];

- *Economia na utilização de processadores* — pode ocorrer um empate entre o tempo oferecido por um processador já utilizado e um novo processador (ainda não utilizado no processo de escalonamento). A escolha de um novo processador muitas vezes permite que tarefas sucessoras sejam escalonadas neste mesmo processador reduzindo em muitos casos o *makespan*. Por outro lado com esta opção selecionada, o número de processadores utilizados tende a aumentar;
- *Inserção em espaços ociosos* — durante o processo de escalonamento é possível que espaços de tempo ociosos nos processadores sejam gerados. Neste caso, escalonar a próxima tarefa livre em espaço ocioso pode vir a produzir melhores resultados, principalmente quando os processadores são heterogêneos [23];
- *Redução número de processadores utilizados* — após a realização do escalonamento é possível que um conjunto de tarefas escalonadas possa ser deslocado para um outro processador que possui um espaço de tempo ocioso, o suficiente para executá-las, sem aumentar o *makespan* do escalonamento gerado.

A definição de uma série de prioridades é um caminho promissor na busca por uma solução eficiente. A ordem em que as tarefas são escalonadas interfere profundamente na construção da solução. Além disso, não somente as características arquiteturais como também a topologia do grafo de entrada (que caracteriza uma classe de aplicações) são fatores determinantes na especificação de uma heurística.

2.2. Algoritmos do ambiente de análise

Devido à complexidade do problema de escalonamento, e ainda mais, considerando heterogeneidade quanto aos recursos, a análise analítica da otimalidade das soluções alcançadas por uma estratégia é um trabalho árduo e muitas vezes só realizado para uma classe específica de instâncias[2]. Assim, para analisar por completo o comportamento de uma estratégia proposta, é primordial a comparação experimental com outros algoritmos existentes. Então, para compor o ambiente de análise proposto, os seguintes algoritmos de escalonamento da literatura foram implementados e também, algumas adaptações foram propostas, conforme descritos a seguir.

O algoritmo ETF (*Earliest Time First*) [11] é um algoritmo do tipo *list scheduling* projetado para um número finito de processadores homogêneos. 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.

O algoritmo DCP (*Dynamic Critical Path*) [14] usa a estratégia *look-ahead* para encontrar o melhor *cluster* de tarefas para um dado vértice. Por isso, para computar o tempo de inicio de uma tarefa em um processador, DCP também computa o tempo de início do seu sucessor imediato crítico no mesmo processador. DCP escalona a tarefa 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. Neste trabalho, tanto o DCP quanto o ETF foram adaptados para um conjunto limitado de recursos heterogêneos.

O algoritmo *Heterogeneous Earliest Finish Time* (HEFT) [23], também classificada como uma heurística *list scheduling,* considera um número limitado de recursos heterogêneos. HEFT inicialmente ordena todas as tarefas de acordo com a prioridade nível em uma lista. Posteriormente, seguindo a ordem determinada nesta lista, associa cada tarefa a um processador, de forma que seja minimizado o tempo de conclusão da tarefa escolhida. HEFT, no entanto, realiza o procedimento de inserção da tarefa em espaços de tempo ociosos nos processadores.

Na literatura existem muito poucos algoritmos que tratam os parâmetros de comunicação definidos nos modelos tipo LogP [6]. A maioria adota a estratégia de aglomeração com replicação de tarefas [2]. O único algoritmo tipo *list scheduling* é uma versão do algoritmo ETF modificado para o modelo LogP proposto por Kalinowski *et al.* [12] que também foi implementado na nossa ferramenta. Utilizando as mesmas técnicas, uma versão HLogP do HEFT também foi implementado. Técnicas alternativas para tratar as sobrecargas de comunicação em algoritmos *list scheduling* estão sendo investigadas.

3. A ferramenta *Task Scheduling Testbed*

A ferramenta *Task Scheduling Testbed*, foi implementada com o objetivo de facilitar o processo de

desenvolvimento de algoritmos de escalonamento de aplicações para sistema heterogêneos e distribuídos. Esse processo tipicamente consiste de 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 sendo construídos como pelos existentes, considerando um conjunto de *GADs* e modelos arquiteturais escolhidos pelo próprio usuário da ferramenta; a validação por simulação utilizando *SimGrid 2.0* [5] e/ou; a validação por execução na Grade Computacional GridRio. Para o desenvolvimento desta ferramenta foi utilizado o Borland Kylix 3.0 [BorlWeb].

3.1. Projeto de um algoritmo

Na versão atual da ferramenta, o usuário seleciona um algoritmo da lista de heurísticas disponíveis (*list scheduling* configurável, DCP, ETF, HEFT, por exemplo), escolhe a aplicação de entrada (representada por um *GAD*), e define o conjunto das características principais da arquitetura alvo (detalhes da aplicação e da arquitetura são lidos de arquivos texto e pode ser escrito pelo usuário). Junto com a ferramenta, se encontra disponível todos os *GADs* dos *benchmarks* de escalonamento de [15] e [1] e outros grafos da literatura. A interface amigável da ferramenta facilita o entendimento do usuário na seleção de todas essas características e prioridades necessárias. No caso do algoritmo *list scheduling,* o usuário pode configurá-lo tanto no que se refere à escolha de tarefas (por exemplo, listas de tarefas estática ou dinâmica e as prioridades de escolha) quanto à escolha do processador (economia ou não de processadores em caso de empate e inserção de tarefas em espaços ociosos). A ferramenta assim, a partir da especificação da política de escalonamento a ser utilizada e seus dados de entrada, apresenta os resultados necessários à avaliação através dos seguintes módulos de interface.

Especificação do algoritmo e seus dados de entrada: O módulo da interface que possibilita a especificação da estratégia a ser utilizada e seus dados de entrada pode ser visualizado na Figura 1.

Visualização do grafo da aplicação (GAD): O usuário pode visualizar o grafo escolhido selecionando o arquivo que contém uma descrição textual do grafo. Utilizando um software livre de AT&T Research Labs, uma figura do mesmo pode ser gerada para visualização na tela ou impressão. Um exemplo de GAD pode ser visto na Figura 2: tarefas da aplicação são representadas pelos vértices rotulados pela sua identificação e seu peso de execução entre parênteses; e as relações de precedência, representadas por arcos, são rotuladas pelos pesos de comunicação.

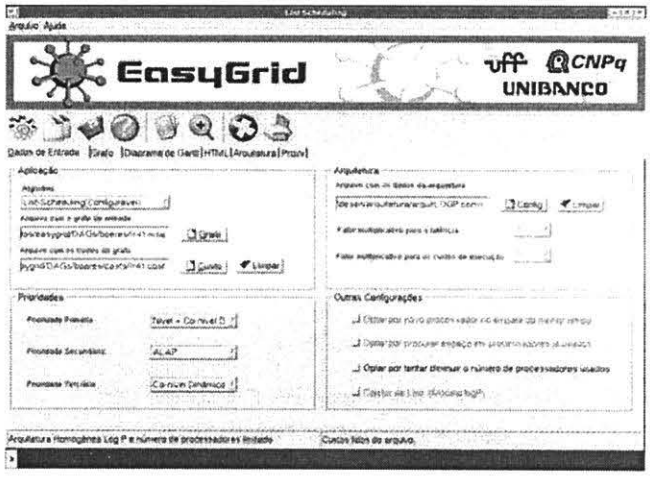

Figure 1. Escolha do algoritmo, do grafo da aplicação e do modelo de arquitetura.

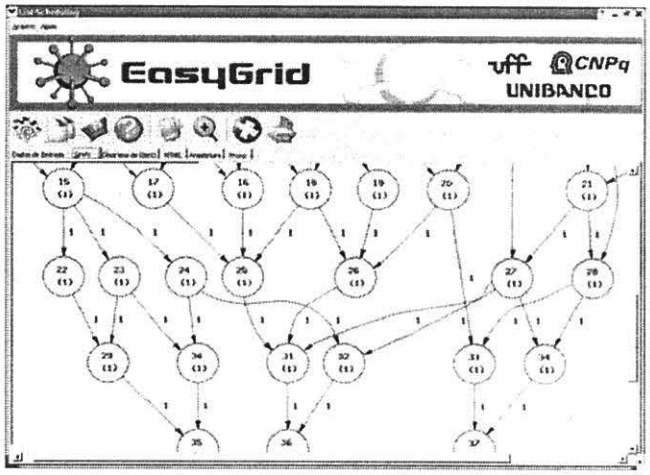

Figure 2. GAD representando a aplicação.

Visualização do escalonamento: O resultado do escalonamento é apresentado em uma tabela *processador* × *tempo*, onde o tempo é incrementado em unidades ou por um diagrama de Gantt (que facilita a análise de escalonamentos com *makespan* de alto valor). No exemplo da Figura 3, cada linha corresponde a um processador utilizado, e cada bloco, representando uma tarefa, tem comprimento proporcional ao seu custo de execução.

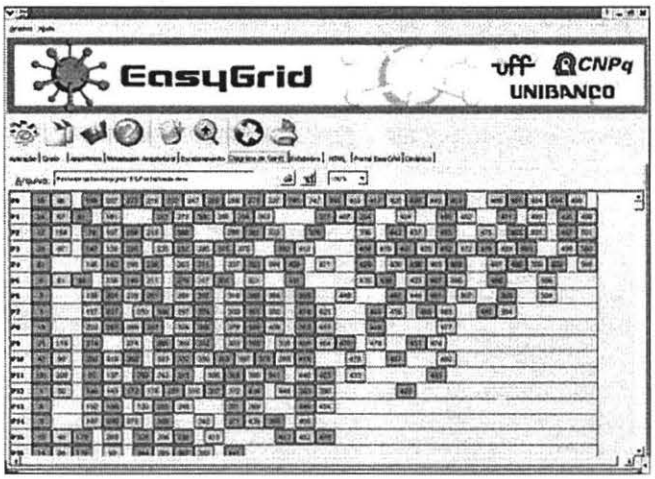

Figure 3. Diagrama de Gantt de um escalonamento resultante.

3.2. Avaliação dos algoritmos

O usuário tem possibilidade de selecionar um conjunto de algoritmos e GADs para ser usado como base de comparação dos algoritmos. É gerada uma tabela *algoritmo* × *grafo* com o *makespan* obtido e o número de processadores necessários para o escalonamento gerado por cada algoritmo, dando oportunidade assim de comparar com mais detalhes os resultados. Na Figura 4, os resultados dos algoritmos DCP, ETF, HEFT e duas novas versões de *list scheduling* construídos a partir da ferramenta são apresentados para uma série de grafos de entrada.

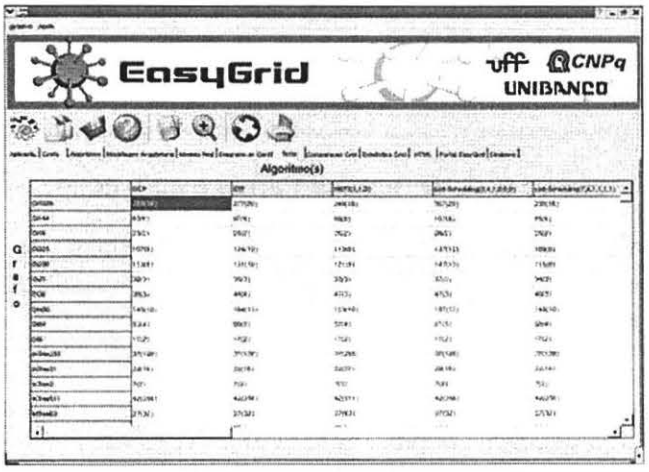

Figure 4. Resultados de DCP, ETF, HEFT e dois algoritmos *list scheduling*.

Um resumo dos resultados pode ser apresentado como uma coleção de comparações par-a-par (comparando cada par de algoritmos sob investigação) em termos de número de vitórias, empates e derrotas, ou simplesmente, como na Figura 5, o percentual de vezes que cada algoritmo gera o melhor escalonamento (é apontado quando o algoritmo é melhor que todos os outros e quando é tão bom quanto algum outro).

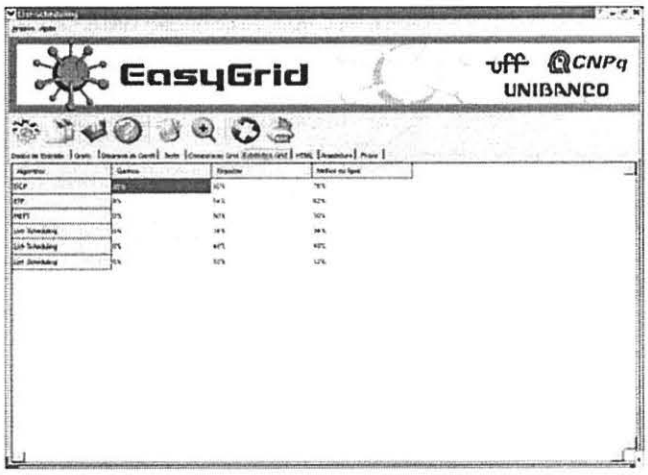

Figure 5. Resumo dos resultados.

O usuário pode salvar o escalonamento ou a comparação em um arquivo texto ou *html*, e as figuras em um arquivo *jpeg*, além de imprimí-los.

3.3. Validação dos escalonamentos

O *makespan* calculado pelo algoritmo de escalonamento é, na verdade, somente baseado nas características consideradas pelo modelo arquitetural escolhido. O tempo de execução real da aplicação depende, no entanto, de vários outros aspectos, normalmente omitidos em tais modelos arquiteturais. Assim, para validar o escalonamento produzido é necessário executar a aplicação escalonada em um ambiente real ou simulado.

Enquanto a simulação oferece a possibilidade de considerar e controlar vários aspetos de ambientes de grade, as conclusões práticas dependem da precisão da modelagem adotada pelo simulador. Resultados exatos podem ser obtidos se uma grade real for utilizada, mas as conclusões só se aplicam para aquela grade.

A ferramenta utiliza o simulador *SimGrid* [5], que fornece funcionalidades para simular um ambiente paralelo e distribuído utilizado no estudo de escalonamento de aplicações. No momento, a interface gráfica ao *SimGrid* está sendo incorporada para que a

execução das tarefas da aplicação seja visualizada e comparada com a previsão.

Para executar aplicações MPI em ambientes grades deve-se seguir uma seqüência de passos que pode ser complexa e monótona para usuários inexperientes. Desta forma, para facilitar este trabalho, a funcionalidade de um *portal* grade foi integrado à ferramenta de escalonamento. Neste *portal*, as etapas necessárias à execução da aplicação são apresentadas na ordem requisitada, e a seqüência de passos necessária é feita automaticamente através de um processo de seleção de opções.

O primeiro passo para a execução de uma aplicação é a criação de um *proxy* (exige a identificação do usuário por *login* e *senha*), que permite acesso aos recursos da grade. O *proxy* tem um prazo de validade, o que é administrado pelo portal (Figura 6).

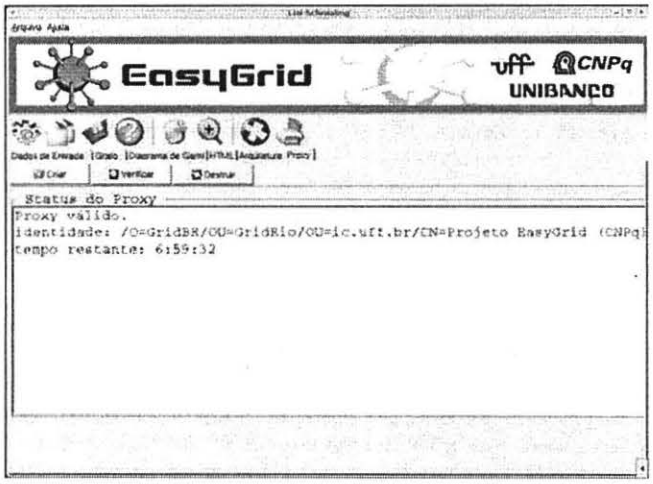

Figure 6. Gerenciamento de *Proxy*.

Em seguida, é realizada a leitura do arquivo que contém todas as máquinas às quais o usuário tem acesso. O passo seguinte é a verificação de quais das máquinas disponíveis estão de fato ativas (estão disponíveis na rede e aceitando *jobs*), conforme apresentado na Figura 7. Neste caso, as características relevantes destas máquinas para o escalonamento e posterior execução da aplicação paralela são obtidas, através dos dados fornecidos pelo MDS do Globus [7]. De posse destas informações, o usuário pode escolher dentre as máquinas ativas, quais são aquelas que devem ser levadas em consideração pelo algoritmo de escalonamento (isto é, calibrar o modelo HLogP pelos custos atuais da grade), e a partir daí, gerar o escalonamento da aplicação. Alternativamente, o usuário pode optar por não fazer uso dos escalonadores disponíveis na ferramenta, e executar a aplicação usando o escalonador do MPI.

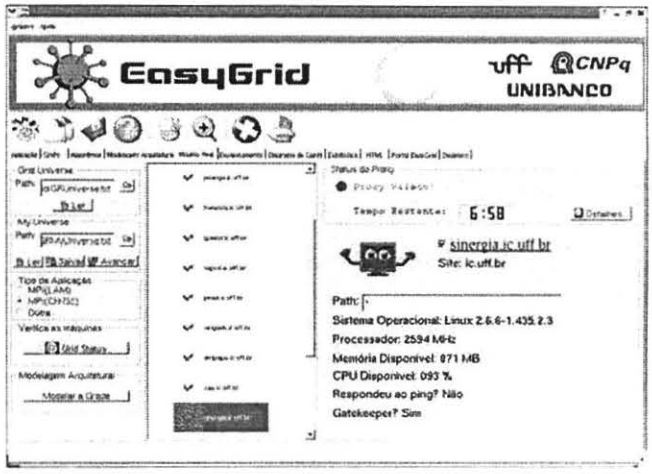

Figure 7. Escolha dos recursos da grade.

10. Conclusões

A natureza heterogênea e dinâmica de grades faz com que o desenvolvimento de softwares de sistema, ferramentas e aplicações paralelas para tais ambientes se torne um desafio.

O objetivo principal do escalonamento é a execução eficiente de uma aplicação paralela considerando as restrições impostas pelo sistema alvo. Ao se tratar de grades, é muito importante determinar o uso eficiente dos recursos para diversas aplicações a serem executadas, mas também que o desempenho de cada uma das aplicações seja maximizada. Nestes ambientes, a variação de custo de comunicação, a heterogeneidade dos recursos e a natureza compartilhada tornam ainda mais complexa a atribuição eficiente de tarefas aos recursos disponíveis.

A ferramenta *Task Scheduling TestBed* apresenta uma interface gráfica bastante atrativa para o usuário, provendo um ambiente de fácil utilização para a construção e análise de algoritmos de escalonamento para grades através da combinação de diferentes mecanismos. A ferramenta oferece a oportunidade de avaliar os escalonamentos gerados por vários algoritmos.

Uma versão da ferramenta está sendo desenvolvida, que integra as funcionalidades do Portal com o escalonamento inicial dos processos e gerenciamento das aplicações *system-aware*. Além disso, uma série de políticas de escalonamento dinâmico, que utilizam informações de um dado escalonador estático [3], estão sendo propostas e adicionadas à ferramenta. Suporte para usuários móveis também será oferecido. Neste caso, a interface gráfica seria re-implementada em Java, se comunicando com a parte da ferramenta (instalado em um servidor persistente) responsável

pela execução dos algoritmos de escalonamento, o simulador e o Portal Grade, via *Web Services*.

Trabalhos futuros irão focalizar a aplicabilidade dos algoritmos (existentes e novos) para escalonar outras classes de aplicações distribuídas (por exemplo, aplicações SPMD).

10. Referências bibliográficas

[1] C. Boeres e V. E. F. Rebello. On solving the static task scheduling problem for real machines. In *Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications*, chapter 3, pages 53-84. Kluwer Academic Publishers, 2002.

[2] C. Boeres e V.E.F. Rebello. Towards optimal task scheduling for realistic machine models: Theory and Practice. *The International Journal of High Performance Computing Applications*, Vol. 17, No. 2, 2003. Sage Public.

[3] C. Boeres, A. Lima e V.E.F. Rebello. Hybrid Task Scheduling: Integrating Static and Dynamic Heuristics. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2003)*, São Paulo, Brazil, November 2003. IEEE Computer Society Press.

[4] C. Boeres e V.E.F. Rebello. EasyGrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Practice and Experience*, 16 (5): 425-432, 2004. John Wiley and Sons.

[BorlWeb] Borland. http://www.borland.com.br/

[5] H. Casanova, SimGrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, e T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, USA, May 1993.

[7] I. Foster e C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115--128, 1997.

[8] I. Foster e C. Kesselman (editors). *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[9] M. R. Garey e D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., 1979

[10] The GridRio Computational Grid. http://easygrid.ic.uff.br/grid/GridRio.html

[11] J. Hwang, Y. Chow, F. Anger e B. Lee, Scheduling precendence graphs in systems with interprocessor communications times. *SIAM Journal of Computing*, 18(2):1-8, 1989.

[12] T. Kalinowski,, I. Kort, e D. Trystram, List scheduling of general task graphs under LogP. *Parallel Computing*, 26(9):1109-1128, 2000.

[13] T. Kielmann, H. Bal, S. Gorlatch, K. Verstoep, e R. Hofman. Network performance-aware collective communication for clustered wide area systems. *Parallel Computing*, 27(11):1431-1456, 2001.

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

[15] Y. K. Kwok e I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381-422, Dec. 1999.

[16] Y-K Kwok e I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4), Dec. 1999.

[17] M. Maheswaran and H. J. Siegel. A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems". In *The Proceedings of the 7th Heterogeneous Computing Workshop (HCW98)*, pages 57-69, Orlando, Florida, March 1998. IEEE Comp. Soc. Press

[18] H.A. Mendes, *HLogP: Um modelo de escalonamento para a execução de aplicações MPI em grades computacionais*, Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, 2004.

[19] Message Passing Forum. *MPI: a Message Passing Interface*. Technical report, University of Tennessee, 1995.

[20] C.H. Papadimitriou, and M. Yannakakis, Towards and architecture-independent analysis of parallel algorithms. SIAMJ Computer, v. 19, p. 322-328, 1990.

[21] V.E.F. Rebello e C. Boeres, Projeto EasyGrid: Um framework para a habilitação automática de aplicações MPI em Grids Computacionais (e a Iniciativa GridRio). Nos *Anais do I Workshop em Grade Computacional e Aplicações*, Programa de Verão do LNCC, Petrópolis, RJ, Brazil, Janeiro 2003.

[22] J.A. Silva, A.A. Fonseca, B.A. Vianna, C. Boeres e V.E.F. Rebello, A Grade Computacional GridRio e o Projeto EasyGrid. Nos *Anais do II Workshop em Grade Computacional e Aplicações*, Programa de Verão 2004 do LNCC, Petrópolis, RJ, Brazil, Fevereiro, 2004.

[23] H. Topcuoglu, S. Hariri, and M. Wu, Performance-effective and low-complexity task cheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13,(3): 260-274, March 2002.

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