Escalonamento de Tarefas Visando Balanceamento de Carga em Ambientes de Computação em Grade

Gustavo Jardim Portella[1], Alba Cristina Magalhães Alves de Melo[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Caixa Postal 4466 – 70.910-900 – Brasília - DF – Brasil
{gustavo,albamm}@cic.unb.br

Resumo. *O escalonamento de recursos é um dos componentes mais importantes de um sistema de computação em grade. Seu objetivo é escalonar um conjunto de aplicações que pertencem a diferentes usuários em um conjunto de recursos heterogêneos e não dedicados, visando maximizar a utilização dos recursos, oferecendo alto throughput. O presente artigo descreve o projeto de uma estratégia de balanceamento de carga para escalonamento de tarefas em grades computacionais. Os resultados preliminares obtidos em um protótipo utilizando o SunGridEngine (SGE) mostram que, para o escalonamento integrado de 13 tarefas em 4 máquinas, a estratégia proposta é capaz de reduzir o comprimento de escalonamento (makespan) em até 22%, quando comparada ao algoritmo default do SGE.*

Abstract. *Resource scheduling is one of the most important components of a grid management system and its goal is to schedule a set of applications which belong to different users to a set of heterogeneous and non-dedicated resources aiming to maximize the resource utilisation and offering high throughput. This article proposes and evaluates a load balancing strategy for scheduling tasks in grids that is based on the analysis of the workstation utilisation by both local and remote tasks. Results obtained with a grid prototype that uses Sun Grid Engine (SGE) show that our strategy is able to reduce the makespan in around 22%, for the worst case, when compared to the default SGE's algorithm.*

1. Introdução

A popularidade da Internet e a disponibilidade de computadores com alto poder de computação e tecnologias de rede de interconexão de alta velocidade a baixo custo, aliados às necessidades reais de aplicações científicas, deram origem à idéia de se criar um "supercomputador" a partir destes componentes, aproveitando os ciclos ociosos das máquinas conectadas à Internet [Foster and Kesselman 99]. Esta abordagem ficou conhecida como metacomputação [Cattlet and Smarr 92]. Na metacomputação, as aplicações eram tipicamente da área científica, como modelagem de reservas de petróleo e simulações de mudanças no clima.

A computação em grade é considerada uma evolução da metacomputação onde não somente o poder computacional das máquinas é compartilhado mas também diversos outros tipos de recursos, como dados, softwares e equipamentos específicos [Foster and Kesselman 99]. Os softwares para grid se preocupam inicialmente em oferecer um *middleware* que oferecesse uma infraestrutura em escala global (*wide area*) para suportar o processamento on-line de aplicações distribuídas [Nabrzyski, Schopf, J. and Weglarz 2003]. O Globus Toolkit 3.0 [Foster and Kesselman 2002], o SunGridEngine (SGE) [Sun 2002], o Legion [Grimshaw and Wulf 1997] e o Mygrid [MyGrid 2004] são exemplos de *middlewares* deste tipo. Os quatro oferecem soluções básicas para problemas como autenticação, descoberta de recursos, acesso a recursos, execução de tarefas remotas e movimento de dados e possibilitaram a construção de diversas aplicações reais para grades computacionais. Eles oferecem, portanto, a infraestrutura básica porém não os mecanismos mais adequados para a gerência de recursos em grade.

O escalonamento de recursos é um dos principais componentes de um mecanismo de gerência de recursos em grade. Consiste, geralmente, em se escalonar um conjunto de aplicações de diferentes usuários em um conjunto de recursos com o objetivo de maximizar a utilização do sistema (*high throughput)*. A partir de uma lista de recursos potenciais gerada por um mecanismo de descoberta de recursos, o escalonador de recursos em grade irá escolher o melhor conjunto de recursos que atende aos requisitos da aplicação. A escolha dos melhores pares de aplicações e recursos é um problema NP-Completo [Papadimitriou 1998] e, por esta razão, heurísticas são geralmente empregadas.

Na computação em grade, o escalonamento de recursos é bastante complexo pois envolve recursos dispostos em diversos domínios administrativos, geograficamente distribuídos. Neste contexto, os recursos são heterogêneos e somente são aproveitados quando ociosos. Além disso, recurso não se refere somente à CPU, mas também podem dizer respeito a espaço em disco e largura de banda em rede, entre outros. Diversas políticas de escalonamento foram propostas para grades, tais como [Maheswaran 1999] e [He, Sun and Laszewski 2002]. No entanto, a grande maioria destas somente foi testada em ambiente de simulação.

No contexto brasileiro, a computação em grade se apresenta como um fator integrador, permitindo que instituições localizadas geograficamente dispersas colaborem de maneira efetiva na resolução de problemas complexos nos domínios de biologia computacional, física de alta energia e previsão meteorológica, entre outros.

O presente artigo descreve a proposta, implementação e avaliação de um mecanismo de escalonamento de aplicações distribuídas em máquinas que compõem uma grade computacional, visando o balanceamento de carga e levando em conta a heterogeneidade e ocupação local de cada máquina. Os resultados obtidos em um protótipo de teste composto por 5 máquinas de execução rodando o SunGridEngine indicam que o uso da estratégia proposta permite redução significativa no comprimento do escalonamento (*makespan).*

O restante deste artigo está organizado como se segue. A seção 2 apresenta conceitos básicos de computação em grade, com enfâse na arquitetura geral e no escalonamento de recursos. Na seção 3 é descrito o SunGridEngine. O projeto do nosso escalonador de recursos em grade é detalhado na seção 4. A seção 5 apresenta alguns resultados preliminares e os avalia. Finalmente, a seção 6 conclui o trabalho e apresenta trabalhos futuros.

2. Computação em Grade

A proposta da computação em grade é que as organizações existentes no mundo disponibilizem seus recursos computacionais, interligando-os na Internet. Esses recursos de empresas distintas podem, então, ser agrupados e formar organizações virtuais cujos limites reais não existam ou, pelo menos, sejam uma abstração [Foster and Kesselman 1999]. A quantidade de recursos existentes na grade é dinâmica, pois a qualquer hora uma nova organização pode decidir participar ou até mesmo sair da grade. Nesta estrutura, uma mesma organização convencional pode participar em mais de uma organização virtual e ter seus recursos divididos entre elas.

Para que a computação em grade seja viável, deve ser definida uma arquitetura que permita a implementação mecanismos complexos de controle de acesso a recursos distribuídos em escala global, em um ambiente heterogêneo e não dedicado. A arquitetura em grade mais amplamente utilizada é a proposta por [Foster and Kesselman 1999], baseada no modelo da ampulheta e composta por cinco camadas (figura 1).

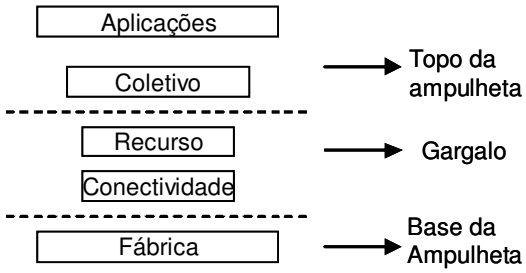

Figura 1. Modelo da Ampulheta

A camada *Fábrica* controla os recursos a serem disponibilizados para os usuários. Neste nível, devem existir mecanismos que possibilitem o gerenciamento de recursos computacionais, recursos de armazenamento, recursos de rede e catálogos.

A camada de *Conectividade* é constituída por protocolos de comunicação e autenticação. Os primeiros possibilitam a troca de dados entre os recursos da camada fábrica e os últimos permitem a identificação e verificação seguras de usuários e recursos.

A camada *Recurso* possibilita a negociação segura de recursos individuais. Oferece informações sobre recursos e permite a especificação de políticas de uso de cada recurso.

A camada *Coletividade* permite o estabelecimento de políticas que considerem um conjunto de recursos, como co-alocação, por exemplo.

Por fim, a camada de *Aplicações* é composta pelos programas desenvolvidos pelo usuário.

O escalonamento de recursos em grade ocorre na camada *Recurso* e consiste em definir o melhor mapeamento entre tarefas e recursos, segundo algum critério de otimização e com base em informações disponibilizadas pelo mecanismo de descoberta de recursos. Além de

ser um problema NP-Completo em sua formulação genérica [Papadimitriou 1998], o problema de escalonamento de recursos em grade é particularmente difícil pois deve levar em conta, para atingir o critério de otimização, a heterogeneidade e a escalabilidade do ambiente de grade.

Uma das medidas mais utilizadas na avaliação de um escalonador de recursos é o comprimento do escalonamento (*makespan*), que é o tempo contado desde a colocação da primeira tarefa na fila de prontos até o término da última tarefa [Kwok and Ahmad 1999]. O *makespan* é a medida básica de rendimento (*throughput*) de sistemas computacionais heterogêneos [He, Sun and Laszewski 2002].

Como exemplos de escalonadores de recursos em grade, podemos citar o Nimrod/G [Buyya, Abramson, and Giddy 2000], que usa um modelo computacional econômico baseado em custos artificiais para a escolha dos recursos; o AppLes [Berman and Wolsky 2000], que executa o escalonamento centrado na aplicação e o trabalho de [He, Sun and Laszewski 2002], que visa reduzir o *makespan* utilizando o modelo probabilístico de previsão de desemepnho proposto por [Gong, Sun and Watson 2002], obtendo bons resultados em ambiente de simulação. A nossa estratégia de escalonamento é baseada neste último trabalho.

3. Escalonamento de Tarefas no Sun Grid Engine

A arquitetura do SGE é composta por 3 camadas: a camada de acesso, que provê mecanismos de autenticação e acesso à arquitetura, a camada de gerenciamento, que compreende os serviços de gerência de recursos distribuídos, e a camada de computação, que fornece o poder computacional do sistema como um todo [S02a]. Dentro desta arquitetura, temos os *hosts* de submissão na camada de acesso, os *hosts* mestre e gerencial na camada de gerenciamento, e finalmente os *hosts* de execução na camada de computação.

O gerenciamento de tarefas no SGE é composto por 4 etapas (figura 2): submissão, escalonamento, execução e armazenamento de resultados. A seguir, cada uma das etapas é detalhada.

a. Submissão: quando um usuário submete uma tarefa a um *host* de submissão, uma requisição de submissão de tarefa é enviada ao *host* mestre.

b. Escalonamento: o *host* mestre determina o local no qual a tarefa será executada, acessando os dados da mesma e verificando a disponibilidade de *hosts* de execução. São verificados também os requisitos de software para execução bem como o grau de prioridade de cada tarefa.

c. Execução: após obter as informações de escalonamento, o *host* mestre envia a tarefa ao *host* selecionado. O *host* de execução salva a tarefa em uma base de dados de informações de tarefas e dispara um processo de controle que inicia a execução da tarefa em um *slot* livre de uma de suas filas de execução, aguardando até a sua finalização.

d. Armazenamento dos resultados: quando a tarefa é completada, o processo de controle retorna as informações da tarefa, dessa forma o *host* de execução reporta o término da execução ao *host* mestre e remove a tarefa da base de dados de tarefas. Em seguida, o *host* mestre atualiza a base de dados de progresso para refletir o término da tarefa.

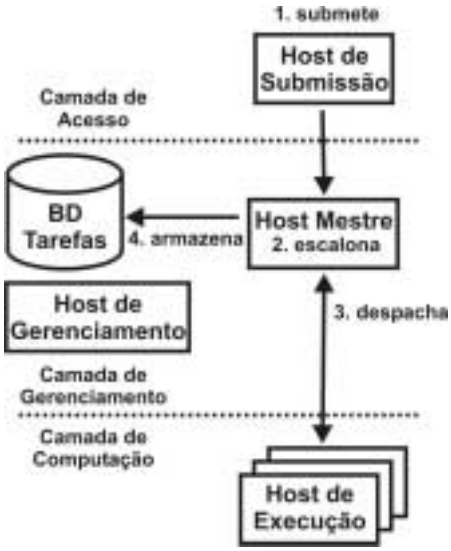

Figura 2 – Fluxo das tarefas no Sun Grid Engine

A atividade de escalonamento de tarefas descrita na etapa (b) é realizada por um processo denominado *daemon* de escalonamento ou escalonador. Este processo é executado em segundo plano no *host* mestre e é ativado periodicamente a cada evento de escalonamento. Durante este evento, cada uma das tarefas é analisada por ordem de chegada FCFS (*First Come, First Served*) [Sun 2002]. A presença do parâmetro de prioridade faz com que as tarefas prioritárias sejam avaliadas primeiramente. No algoritmo de escalonamento *default* do SGE, a atribuição das tarefas a hosts de execução se dá segundo a política round-robin.

No final do evento é construída uma lista de ordens de despacho, que contém as informações sobre o local no qual cada tarefa será executada. Para construir esta lista, o escalonador avalia os *hosts* de execução que possuem *slots* livres em uma de suas filas de execução. Normalmente, há mais de uma fila apropriada para a execução da tarefa e, sendo assim, dependendo da política adotada, a escolha da fila obedecerá ao número identificador seqüencial da fila, ou o critério de menor sobrecarga dentre os *hosts* candidatos.

4. Estratégia Proposta de Escalonamento com Balanceamento de Carga

A estratégia proposta de balanceamento de carga se baseia no modelo de previsão de tempo de execução de tarefas em ambientes heterogêneos proposto em [Gong, Sun and Watson 2002]. Com base na idéia de que a chegada de processos do usuário local em uma estação de trabalho que pertencente à grade computacional influencia no tempo de término da tarefa proveniente da grade, uma melhoria no escalonamento padrão do SGE foi proposta a fim de aprimorar a geração das listas de despachos de tarefas pelo escalonador. Essa melhoria considera o progresso da liberação de *slots* no *host*, sendo diretamente influenciada pela execução de tarefas locais, ou seja, quanto mais utilização local houver na estação, menos progresso terá o conjunto de tarefas provenientes da grade e, conseqüentemente, menos *slots* serão liberados.

A análise das tarefas a serem escalonadas durante o evento de escalonamento é feita da mesma forma que a abordagem de escalonamento padrão FCFS, porém a escolha da fila de execução obedece à proporção de carga real das mesmas no momento do escalonamento. Para o cálculo desta proporção para cada fila, foi utilizada a fórmula (1):

$$p = n^o \textit{ slots} \text{ livres} / n^o \text{ total de } \textit{slots} \qquad (1)$$

Desta forma, para cada fila de execução candidata ao recebimento de uma tarefa proveniente da grade computacional, a proporção *p* correspondente é calculada e com base nela as ordens de despacho são construídas, como é mostrado no algoritmo apresentado na figura 3.

```
1 para cada tarefa t_i , faça:
2   para cada fila de execução q_j , faça:
3     p_ij = slots_livres(q_j) / slots(q_j)
4   fim
5   descubra q_j tal que min(p_ij)
6  adicione ordem(t_i,q_j)
7  slots_livres(q_j) = slots_livres(q_j) – 1
8 fim

        onde:
        t_i = tarefa a ser escalonada
        q_j = fila de execução não cheia
```

Figura 3 - Algoritmo de balanceamento de carga para a escolha das filas candidatas

Os *hosts* de execução que possuam uma configuração de hardware (processador e memória) superior em relação aos outros *hosts* tem condições de suportar proporcionalmente uma fila de execução com mais *slots* em relação às outras filas de execução participantes da grade. Sendo assim, o algoritmo proposto pode tirar mais proveito da proporcionalidade de *slots*, obtendo-se um ganho em tempo de execução no cálculo do *makespan*.

5. Resultados Experimentais

A estratégia descrita na seção 4 foi implementada em C e integrada ao SunGridEngine. A realização dos primeiros testes experimentais envolveu um protótipo contendo os seguintes *hosts* com as seguintes configurações:

a) 1 *host* mestre com processador Intel Pentium 4, 256 MB RAM, Linux Red Hat 9, SGE 5.3p3,

b) 3 *hosts* (h1, h2 e h3) de submissão com processador AMD Athlon XP 1.1 GHz, 256 MB RAM, Linux Red Hat 9, SGE 5.3p3,

c) 1 *host* (h4) de submissão com processador AMD Athlon 900MHz, 128 Mb RAM, Linux Red Hat 9, SGE 5.3p3.

Nesse ambiente, somente os hosts h1, h2, h3 e h4 são hosts de execução.

Os testes iniciais foram realizados com o cálculo do valor de pi utilizando-se o algoritmo de Bailey-Borwein-Plouffe [Bailey Borwein and Plouffe 1995]. Primeiramente, foi utilizada uma implementação do algoritmo escrito em linguagem C, que realiza o cálculo com precisão de 50 casas decimais.

Na realização de um *benchmark* inicial em cada um dos *hosts*, foram obtidos os seguintes tempos de execução (min:seg:centésimoseg) do algoritmo com a execução de 1 instância única, e com 2, 3, 4 e 5 instâncias concorrentes.

Tabela 1- Benchmark para o cálculo de pi com 50 casas decimais

	1 x pi	2 x pi	3 x pi	4 x pi	5 x pi
h1	00:11:17	00:22:34	00:33:51	00:44:69	00:55:57
h2	00:11:16	00:22:31	00:33:48	00:44:49	00:55:77
h3	00:10:95	00:20:84	00:33:01	00:43:76	00:54:69
h4	00:12:38	00:24:75	00:37:13	00:49:53	01:01:91

Com base no observado na tabela 1, os *hosts* de execução h1, h2 e h3 foram configurados cada um com uma única fila de execução com 5 *slots*. O *host* h4, apontado como o mais "lento" no cálculo do valor de pi, segundo a tabela 1, foi configurado com uma única fila de execução contendo 4 *slots*.

Nos cenários 1, 2 e 3, foram submetidas 5, 9 e 13 tarefas de cálculo de pi ao sistema, respectivamente. A tabela 2 mostra o ganho obtido no *makespan* em cada cenário para o caso médio, melhor e pior caso. Os algoritmos de escalonamento comparados foram o default (default do SGE) e a nossa estratégia de escalonamento (*load-balanced*).

Tabela 2 – Cenários 1, 2 e 3 (pi com 50 casas decimais) com 5, 9 e 13 tarefas (hh:mm:ss)

	Teste 1			Teste2			Teste3		
	Pior	**Médio**	**Melhor**	**Pior**	**Médio**	**Melhor**	**Pior**	**Médio**	**Melhor**
default	00:00:25	00:00:24	00:00:23	00:00:38	00:00:34	00:00:33	00:00:49	00:00:46	00:00:43
load-balanced	00:00:23	00:00:23	00:00:23	00:00:33	00:00:33	00:00:33	00:00:43	00:00:43	00:00:43
ganho(%)	**8,70%**	**4,35%**	**0,00%**	**15,16%**	**3,03%**	**0,00%**	**13,95%**	**6,98%**	**0,00%**

No caso do cenário 3, utilizando-se o escalonador default, o pior caso ocorre quando são atribuídas 4 tarefas ao *host* com o menor número de *slots*, no caso o *host* h4, a princípio visto como o mais "lento" da arquitetura. De forma oposta, o melhor caso, utilizando-se o mesmo escalonador, ocorre quando o *host* h4 recebe apenas 3 tarefas.

Nos mesmos cenários 1, 2 e 3, o escalonador *load-balanced* não atribui mais tarefas para um *host* que tenha menor número de *slots* total em relação aos demais, mantendo o mesmo número de *slots* ocupados para todos. Isto acontece porque que a proporção de carga descrita na equação (1) para a fila do *host* em questão o torna o candidato menos provável na classificação das filas feita na linha 5 do algoritmo mostrado na figura 3. Desta forma, em todas as situações onde haja a necessidade de se escolher uma fila candidata, o parâmetro de proporção de carga faz com que o as tarefas sejam despachadas para filas configuradas para receberem mais tarefas.

Testes semelhantes realizados para o cálculo de pi com número maior de casas decimais levaram a resultados que comprovam a redução no *makespan* proporcionada pela estratégia de escalonamento proposta, como é mostrado na tabela 3.

Tabela 3 – Cenários 4, 5 e 6 (pi com 80 casas decimais) com 5, 9 e 13 tarefas (hh:mm:ss)

	Cenário 4			Cenário 5			Cenário 6		
	Pior	**Médio**	**Melhor**	**Pior**	**Médio**	**Melhor**	**Pior**	**Médio**	**Melhor**
default	00:02:03	00:01:54	00:01:41	00:03:06	00:02:51	00:02:32	00:04:07	00:03:48	00:03:23
load-balanced	00:01:41	00:01:41	00:01:41	00:02:32	00:02:32	00:02:32	00:03:23	00:03:23	00:03:23
ganho(%)	**21,80%**	**12,87%**	**0,00%**	**22,37%**	**12,50%**	**0,00%**	**21,67%**	**12,32%**	**0,00%**

A tabela 3 mostra os resultados da comparação feita entre o escalonador *default* e o proposto para cenários semelhantes aos mostrados na tabela 2, com tarefas que calculam o valor de pi com a precisão de 80 casas decimais. A figura 4 mostra a diferença entre dois casos, melhor e pior.

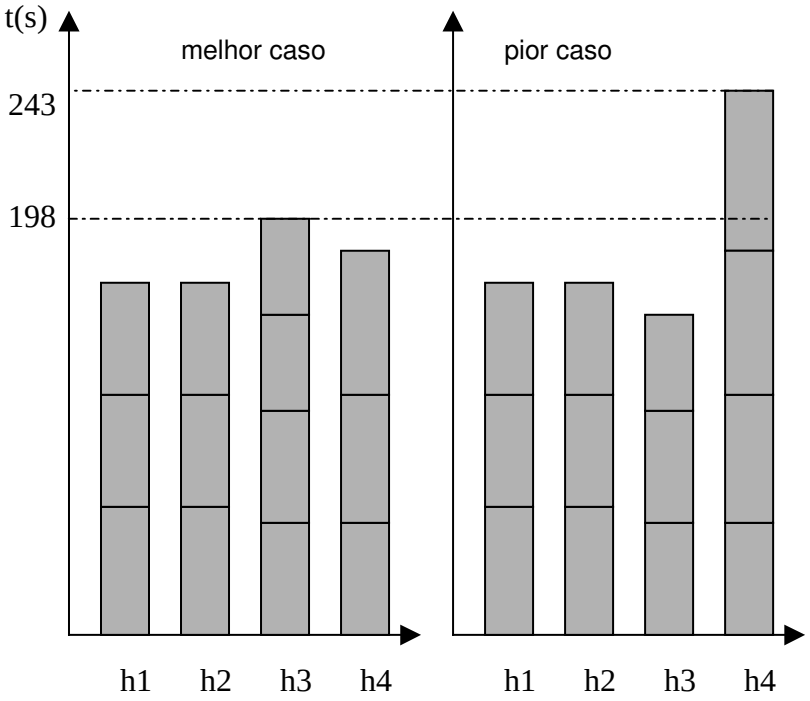

Figura 4 – Melhor caso e pior caso utilizando-se o escalonador default no cenário 6

Como pode ser observado nas tabelas 2 e 3, para o cálculo do pi com 50 e 80 casas decimais, a nossa estratégia de escalonamento (*load-balanced)* conseguiu uma redução de até 22,37% no *makespan* (pior caso) e 12,87% (caso médio), quando comparado ao algoritmo de escalonamento *default* do SGE.

Alguns experimentos também foram feitos com uma aplicação de bioinformática cujo objetivo é fazer a comparação local de duas seqüências de DNA. Essa aplicação, chamada SW, implementa o algoritmo proposto por [Smith and Waterman 1981], que é baseado em programação dinâmica e possui complexidade de tempo $O(n^2)$.

A tabela 4 apresenta 4 cenários, onde 5, 9, 13 e 17 tarefas SW fazem a comparação entre 1000 seqüências de nucleotídeos.

Tabela 4. Makespans para os cenários 7, 8, 9 e 10 (SW para 1000 comparações) com 5, 9, 13 and 17 tarefas de grid (hh:mm:ss)

	Cenário 7 (5 tarefas)			Cenário 8 (9 tarefas)		
	Pior	**Médio**	**Melhor**	**Pior**	**Médio**	**Melhor**
default	00:01:37	00:01:27	00:01:19	00:02:26	00:02:11	00:01:59
Load-balanced	00:01:19	00:01:19	00:01:19	00:01:59	00:01:59	00:01:59
Ganho(%)	**22.78%**	**10.13%**	**0%**	**22.69%**	**10.08%**	**0%**

	Cenário 9 (13 tarefas)			Cenário 10 (17 tarefas)		
	Pior	**Médio**	**Melhor**	**Pior**	**Médio**	**Melhor**
default	00:03:14	00:02:55	00:02:38	00:04:02	00:03:39	00:03:18
load-balanced	00:02:38	00:02:38	00:02:38	00:03:18	00:03:18	00:03:18
Ganho (%)	**22.78%**	**10.76%**	**0%**	**22.22%**	**10.61%**	**0%**

É interessante notar que os resultados obtidos com a aplicação SW foram similares aos obtidos com a aplicação *pi* de 80 dígitos. Em ambos os casos, a estratégia proposta foi capaz de obter uma redução em torno de 22% para o pior caso e em torno de 10% para o caso médio, quando comparados com a estratégia default do SGE. Isso pode ser explicado pelo fato de ambas aplicações possuírem aproximadamente os mesmos tempos de execução e fazerem uso intenso da capacidade de processamento.

7. Conclusões e Trabalhos Futuros

O presente artigo apresentou o projeto e avaliação de uma estratégia de balanceamento de carga para escalonamento de tarefas em grades computacionais. A principal característica desta estratégia consiste em considerar a ocupação local das máquinas no momento da atribuição das tarefas da grade. Os resultados preliminares obtidos com um protótipo de grade composto por 5 máquinas mostraram que a estratégia proposta é capaz de obter uma redução significativa do *makespan*, contribuindo assim para aumentar o rendimento *(throughput)* do ambiente.

Como trabalho futuro imediato, iremos integrar uma análise de histórico de ocupação das máquinas à nossa estratégia. Além disso, a estratégia de escalonamento será avaliada com aplicações reais da área de bioinformática, com tempos de execução medidos em horas, dentro do projeto BIOFOCO, que envolve a UnB, a UCB e a EMBRAPA.

Referências

Buyya, R., Abramson, D., and Giddy, J. (2000) "Nimrod/G: An Architecture for Resource Management and Scheduling System in a Global Computational Grid", Technical Report, Monash University, Australia.

Bailey, D. Borwein, P. and Plouffe, S., (1995) "On the Rapid Computation of Various Polylogarithmic Constants.".

Berman, F. and Wolsky, R. (2000) "The AppLeS Project: A Status Report", Technical Report, Dep. of Computer Science and Engineering, Univ. of California at San Diego.

Cattlet, C. and Smarr, L. (1992) "Metacomputing", Communications of the ACM, 35(6), p. 44-62.

Foster, I., Kesselman, C., (eds.), (1999) "The Grid: Blueprint of a Future Computing Infrastructure", Morgan Kaufmann.

Foster, I., Kesselman, C., Nick, J. and Tuecke, S., (2002) "Grid Services for Distributed Systems Integration", IEEE Computer, 35(6), p. 37-46.

Gong, L, Sun, X. and Watson, E. (2002) "Performance Modeling and Prediction of Non-Dedicated Network", IEEE Transactions on Computers , September, 2002.

Grimshaw, A., Wulf, W. (1997) "The Legion Vision of a Worldwide Computer", Communications of the ACM, January, v.40, n.1.

He, X. Sun, X, Laszewski, G. (2002) "A QoS Guided Scheduling Algorithm for Grid Computing", Int. Workshop on Grid and Cooperative Computing, Hainan, China, 2003.

Kwok, Y. and Ahmad, I., "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors", ACM Computing Surveys, v.31, n.4, December, 1999.

Maheswaran, M. (1999) "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", 8th IEEE Heterogeneous Computing Workshop, Porto Rico, p. 30-44.

My Grid Web Page, http://www.ourgrid.org/mygrid, consultada em Maio/2004.

Nabrzyski, J., Schopf, J. and Weglarz (eds.), (2003) "Grid Resource Management: State of the Art and Future Trends", Kluwer Academic Publishers, (draft).

Papadimitriou, G. and Steiglitz, K. (1998) "Combinatorial Optimization: Algorithms and Complexity", Dover Publications Inc., New York, USA.

Sun Microsystems (2002a), "Sun Cluster Grid Architecture - A technical white paper describing the foundation of Sun Grid Computing", White Paper.

Sun Microsystems (2002b), "Sun Grid Engine 5.3 Administration and User's Guide", White Paper.

T. F. Smith, M. S. Waterman, *Identification of common molecular sub-sequences* – Journal of Molecular Biology, 147 (1) 195-197 – 1981.