Avaliação de algoritmos de escalonamento em Grids para diferentes configurações de ambiente

José Nelson Falavinha Junior[1 *], Aleardo Manacero Júnior[2], Luigi Jacometti de Oliveira[2], Davidson Rodrigo Boccardo[1 *]

[1]Depto. de Engenharia Elétrica – Universidade Estadual Paulista (UNESP)
Ilha Solteira – SP – Brasil (* Bolsista Capes)

[2]Depto. de Ciências de Computação e Estatística – Universidade Estadual Paulista (UNESP) São José do Rio Preto – SP – Brasil

junior.falavinha@gmail.com, aleardo@ibilce.unesp.br,
luigijo@gmail.com, flitzdavidson@gmail.com

Abstract. *The efficient use of resources in a grid structure depends on optimized allocation of applications throughout its components. This allocation is performed by task schedulers, which have their operation disturbed by the distributed status, load, and geographical characteristics found in grid environments. Algorithms such as WQR, Dyn-FPLTF, Xsufferage, Storage-Affinity, and MQ-H, explore distinct characteristics of the environment and workload, but have problems to efficiently react against status changes. In this paper, these algorithms are evaluated in order to identify their behavior under network, resources and load variations. Tests conducted through Simgrid point the urge for better dynamic algorithms in order to achieve an efficient grid allocation.*

Resumo. *O uso eficiente dos recursos de um ambiente de grade (grid) depende da alocação otimizada das aplicações entre seus componentes. Essa alocação é feita por escalonadores de tarefas, que têm seu funcionamento afetado pelas características de distribuição geográfica, carga e estados de grids. Algoritmos como WQR, Dyn-FPLTF, Xsufferage, Storage-Affinity e MQ-H exploram diferentes características do ambiente e conjunto de tarefas, mas têm dificuldades em reagir eficientemente às modificações de estado do ambiente. Neste trabalho avalia-se a adequação desses algoritmos, procurando identificar seu comportamento em relação a variações de carga, recursos e conexões de rede. Testes realizados com o Simgrid apontam para a necessidade do uso de melhores algoritmos dinâmicos para obter uma alocação eficiente em grids.*

1. Introdução

O desempenho oferecido por infra-estruturas computacionais de larga escala, como grids, na execução de aplicações de grande porte tem motivado pesquisadores e cientistas de várias áreas como física, biologia, química e engenharias, a adaptar suas aplicações e tarefas para execução nessas plataformas. Como tais adaptações são feitas a partir de diferentes perspectivas, acabam por gerar características de demanda por execução também muito diferentes. Tais aplicações se tornam mais ou menos eficientes a partir de políticas de escalonamento de tarefas específicas para cada caso.

Nesse sentido, como o escalonamento de tarefas tem o objetivo de maximizar a utilização do sistema, combinando a necessidade da aplicação com a disponibilidade do recurso e garantindo qualidade de serviço entre eles [He X. et al. 2002], surgem dificuldades na escolha da política mais adequada. Preocupações como quantidade de informação a ser processada, localização da informação, quantidade de recursos disponíveis e a taxa de comunicação entre eles, passam a ser vitais no momento em que os recursos a serem alocados através do escalonamento são componentes de um grid.

Neste trabalho procura-se avaliar o comportamento dos principais algoritmos de escalonamento para tarefas em grids, considerando diferentes configurações de ambientes. Cada algoritmo tem características próprias, envolvendo diferentes aplicações alvo, informações necessárias e restrições de recursos, fornecendo, portanto, resultados bastante particulares e diferenciados. A identificação de abrangência da adequação de cada algoritmo, do ponto de vista de seu desempenho e do desempenho das aplicações por ele gerenciadas, se torna então um objetivo importante de avaliação.

Nas próximas duas seções são descritos os algoritmos de escalonamento aqui avaliados e um breve cenário sobre trabalhos semelhantes ao aqui apresentado. Segue-se então com a descrição do ambiente de simulação utilizado nos testes de avaliação (com o simulador Simgrid) e a apresentação dos resultados obtidos. Na última seção apresentam-se as conclusões obtidas, bem como possíveis direções a serem seguidas.

2. Algoritmos de escalonamento em grids

Os algoritmos de escalonamento para tarefas em grids são responsáveis pelo processo de tomar decisões de escalonamento envolvendo recursos sobre múltiplos domínios administrativos. Este processo envolve três fases principais: descoberta de recursos, na qual uma lista de recursos disponíveis é gerada; seleção do sistema, coleta de informações dos recursos e seleção do melhor grupo; e execução do trabalho que inclui exibição do arquivo e coleta dos resultados [Schopf J.M. 2002].

Cada algoritmo tem suas próprias características e, portanto, cada um atua de maneira diferente em determinadas fases do escalonamento. A seguir serão descritos os principais algoritmos de escalonamento para tarefas em Grids, descrevendo suas funcionalidades e metodologias para que fiquem claras as diferenças entre cada um.

2.1. Dynamic-FPLTF

O dinamismo e a heterogeneidade dos recursos presentes em um grid tornam necessário o desenvolvimento de algoritmos dinâmicos como o *Dynamic*-FPLTF [Paranhos D. et al 2003] que foi desenvolvido a partir do estático *Fastest Processor to Largest Task First* (FPLTF) [Menascé D. et al. 1995]. O *Dynamic*-FPLTF necessita de três informações para fazer o escalonamento de tarefas: Tamanho da tarefa, Carga local do *host* e Velocidade relativa do *host*. No início do processo de funcionamento desse algoritmo, uma variável TBA (*Time to Become Available*) é iniciada com valor zero para cada *host* e as tarefas são organizadas em ordem decrescente de tamanho conforme chegam. Uma tarefa é alocada ao *host* que oferecer o melhor tempo de execução (CT - *completion time*) para ela, em que:

- CT = TBA + *Custo da tarefa*;
- *Custo da tarefa* = (*Tamanho da tarefa/Velocidade do host*)/(1 – *Carga do host*);

Quando uma tarefa é alocada em um *host*, o valor TBA correspondente é incrementado pelo *Custo da tarefa*. Tarefas são alocadas até que todas as máquinas do grid estejam em uso. Depois disso, a execução da aplicação começa. Quando uma tarefa termina sua execução, todas as tarefas que não estão executando são escalonadas novamente até que todas as máquinas estejam sendo usadas. Este esquema continua até que todas as tarefas sejam completadas. Esta estratégia tenta minimizar os efeitos do dinamismo do grid.

2.2. Workqueue with Replication

O *Workqueue with Replication* (WQR) foi desenvolvido para solucionar o problema da obtenção de informações sobre a aplicação e a carga de utilização dos recursos do grid [Paranhos D. et al. (2003)].

Em sua fase inicial, o WQR é similar a um *Workqueue* tradicional, ou seja, as tarefas são enviadas para execução nos processadores que se encontram disponíveis em ordem de chegada. Quando um processador finaliza a execução de uma tarefa, este recebe uma nova tarefa para processar. Os algoritmos WQR e *Workqueue* passam a diferir no momento em que um processador está disponível e não há mais nenhuma tarefa pendente para ser escalonada. Neste momento, o *Workqueue* apenas aguarda o término de todas as tarefas ou a chegada de uma nova. Já o WQR, inicia a fase de replicação para tarefas que ainda estão executando. Quando a tarefa original ou uma de suas réplicas termina sua execução, as outras são interrompidas.

2.3. XSufferage

O *XSufferag*e [Casanova, H. et al. 2000, Santos-Neto, E. et al 2004] é um algoritmo de escalonamento baseado nas informações sobre o desempenho dos recursos e da rede que os interliga. Este algoritmo aborda o impacto das grandes transferências em aplicações que usam grandes quantidades de dados. O *XSufferage* é uma extensão do algoritmo *Sufferage* [Ibarra, O. and Kim C. 1977], no qual a idéia básica é determinar quanto cada tarefa seria prejudicada se não fosse atribuída ao processador que a executaria de forma mais eficiente. Este valor é denominado *sufferage* e é definido como a diferença entre o melhor e o segundo melhor tempo de execução previsto para a tarefa, considerando todos os processadores do grid.

A principal diferença entre o *Sufferage* e o *XSufferage* é o método usado para calcular o valor de *sufferage*. O algoritmo *XSufferage* leva em consideração também a disponibilidade corrente do *link* de rede e o tempo de transferência dos dados utilizados pela tarefa, diferente do *Sufferage* que necessita somente das informações relacionadas a CPU e o tempo estimado de execução da tarefa.

Um ponto importante a ser observado é que este algoritmo considera somente os recursos livres no momento em que vai escalonar uma tarefa, pois caso contrário sempre o recurso mais rápido e com a melhor conexão de rede receberia todas as tarefas.

2.4. Storage Affinity

O *Storage Affinity* [Santos-Neto, E. et al 2004] foi feito com o intuito de explorar a reutilização de dados a fim de melhorar o desempenho de aplicações que utilizam

grandes quantidades de dados. O método de escalonamento de tarefas é definido sobre o conceito fornecido pelo próprio nome do algoritmo, ou seja, o conceito de afinidade. O valor da "afinidade" entre uma tarefa e um *site* determina quão próximo do *site* esta tarefa está. A semântica do termo próximo está associada à quantidade de *bytes* de entrada da tarefa que já está armazenada remotamente naquele *site*.

Além de aproveitar a reutilização de dados, o *Storage Affinity*, também realiza replicação de tarefas, tratando da dificuldade de obtenção de informações dinâmicas sobre o grid, e sobre o tempo de execução das tarefas. A idéia é que as replicas tenham a chance de serem submetidas a processadores mais rápidos do que aqueles associados às tarefas originais, reduzindo o tempo de execução da aplicação.

2.5. Multiple Queue - Hierarchy

A idéia básica do algoritmo MQ-H é organizar os recursos disponíveis do grid em níveis de hierarquia de acordo com suas características de desempenho. Cada nível de hierarquia composto de recursos fornece melhor desempenho para tarefas de intervalos de complexidade específicos. Em outras palavras, uma tarefa de determinada complexidade tem um nível de hierarquia que melhor atende suas necessidades em termos de *overhead* e recursos disponíveis [Silberstein, M., et al. 2006].

Neste modelo de níveis, são atribuídas uma ou mais filas de tarefas para cada nível, sendo que cada fila tem dois valores limites de sinalização: *Tq* (tempo máximo de espera na fila), e *Te* (tempo limite para execução de uma tarefa nesta fila); sendo que $Te \leq Tq$. Neste esquema, todas as tarefas novas são submetidas para filas no primeiro nível hierárquico do grid, englobando sistemas de baixo poder computacional, e esperam até serem executadas, ou até estourar o tempo limite da fila, sendo neste caso transferida para filas do próximo nível. A cada nível o poder computacional do nó do grid aumenta, assim como aumenta também os valores de tempos limite para espera e execução.

3. Trabalhos relacionados

A avaliação do desempenho de algoritmos de escalonamento para grids tem seguido uma linha orientada para a justificativa de cada nova abordagem criada. Assim, aparecem os trabalhos realizados por Menascé [Menascé D. et al. 1995], Paranhos [Paranhos D., et al. 2003], Casanova [Casanova, H., et al. 2000], Santos-Neto [Santos-Neto, E. et al 2004], e Fujimoto [Fujimoto, N. and Hagihara, K. (2004)] em que são introduzidos respectivamente os algoritmos FPLTF, Dynamic-FPLTF, WQR, XSufferage, Storage Affinity, e RR, e que apresentam comparações entre alguns desses algoritmos para situações em que suas propostas teriam maior aplicabilidade.

Entre os resultados apresentados nesses trabalhos podem ser destacadas as comparações feitas entre o WQR, que se caracteriza pela não utilização de informações sobre o grid, com o *Dynamic*-FPLTF e com o *Xsufferage*, que utilizam informações sobre o estado dos recursos. Porém, o WQR consome mais ciclos que os outros, e os testes mostrados em [Cirne, W., Santos-Neto E. 2005] relatam que quanto maior a heterogeneidade dos sistemas do grid, maior é a sobrecarga de processamento do WQR.

Em [Santos-Neto, E. et al 2004], o algoritmo *Storage Affinity* é comparado com o WQR e com *XSufferage* mas no contexto de aplicações que utilizam grandes quantidades de dados. Nesse aspecto, devido ao fato de não obter informações da

largura de banda, custo de transferência, localização dos dados e outros diversos fatores, o WQR se mostrou inferior aos outros. Já a comparação entre o *XSufferage* e *Storage Affinity* foi feita sobre alguns pontos principais, como impacto da granulação da aplicação, relação entre o tamanho da entrada e o custo computacional também sobre a heterogeneidade do grid. Ambos tiveram desempenhos parecidos, com pontos favoráveis e desfavoráveis diferentes para os dois lados.

Enquanto nos trabalhos anteriores o foco principal era avaliar alguns algoritmos trabalhando numa situação específica, visando justificar o uso de um novo algoritmo, neste trabalho o foco principal é uma completa avaliação dos algoritmos sobre diferentes configurações de ambiente, sem favorecer nenhum deles. Assim, procura-se determinar as aplicações alvo de cada algoritmo e verificar seus comportamentos diante de mudanças no *status* de ambiente ou características de um grid.

4. Ambiente de simulação

Para a realização dos testes sobre os algoritmos de escalonamento foi definido que o melhor procedimento seria o de simulação, em que se pudesse ter absoluto controle sobre as diversas variáveis de contorno, como diferentes cargas de processamento, volume de dados transferidos e assim por diante. O simulador escolhido foi o Simgrid, por sua fácil adaptação para as diferentes condições de teste exigidas. Nos próximos parágrafos estão descritas tanto as características do Simgrid, quanto as configurações de teste utilizadas.

4.1. O simulador utilizado

O Simgrid [Legrand, A., et al. 2003, Simgrid Project Web Page 2007] é uma ferramenta que fornece um conjunto de abstrações e funcionalidades que podem ser usadas para implementar simuladores para aplicações específicas e/ou topologias de infra-estrutura computacional. Um componente importante do processo de simulação é a modelagem do recurso. Na definição de uma máquina, ou de um *cluster*, por exemplo, são atribuídas variáveis como latência, e taxa de serviço ou poder computacional. O poder computacional é definido como sendo o número de unidades de trabalho por unidade de tempo. Nas implementações desse trabalho foram utilizadas as unidades de: segundos (s) para latência, *Bytes*/s para as conexões de rede, e *flops* para o poder computacional.

Outra parte integrante da simulação de um ambiente grid é a topologia de interconexão dos recursos. O Simgrid admite a definição de variáveis como largura de banda ou taxa de transferência, e latência da rede para as conexões entre recursos. Além disso, o usuário pode especificar quais dessas conexões estão habilitadas ou não, e sua disponibilidade, o que também é possível para recursos computacionais, tornando o ambiente dinâmico com variações de disponibilidade e estado dos recursos ou da rede.

4.2. Plataforma do ambiente para as simulações

A plataforma grid construída, no simulador para realização dos testes, é composta de trinta unidades de processamento divididas em seis *clusters*, com cinco máquinas em cada um, sendo que o poder de processamento dos *clusters* varia entre 800 *Mflops* e 4500 *Mflops*. Todas as máquinas de um *cluster* são conectadas aos outros *clusters* através do *front-end*, e todas as requisições de transferência de dados ou recebimento de

tarefas passam por ele. As conexões internas de um *cluster* são ou de 100 Mbit/s ou 1Gbit/s, e as conexões entre os clusters variam dentro desse intervalo.

Para submissão de tarefas existe uma máquina denominada "Escalonador", responsável por receber tarefas, realizar o escalonamento e submeter ao *front-end* do nó escolhido para execução, o *front-end* recebe a requisição e repassa a tarefa. Essa máquina "Escalonador" é conectadas a todos *front-ends* com a mesma variação de valores para *links* externos, e seu poder computacional é irrelevante pois ela não executa nenhuma tarefa de processamento. Na realidade, a velocidade do escalonamento é determinada pela máquina onde executa a simulação, que é quem realiza todas as operações de seleção de recursos. A plataforma grid construída no simulador é apresentada na figura 1.

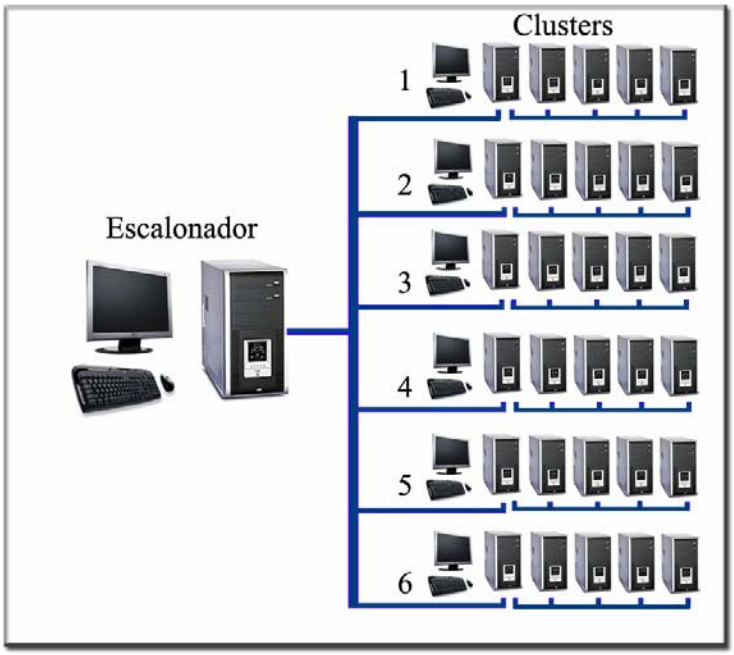

Figura 1. Plataforma grid.

Deve-se observar ainda que como o Simgrid não faz nenhuma distinção entre transferência de dados e computação, sendo ambos vistos como tarefas, torna-se responsabilidade do usuário garantir que tarefas de computação sejam escalonadas em processadores, e transferência de dados para conexões de rede. É importante lembrar ainda que a versão atual do Simgrid assume que todas as tarefas computacionais são *CPU-bound* e que transferência de dados são *bandwidth-bound*, portanto diferentes tarefas computacionais e transferência de dados devem ser implementadas no próprio algoritmo ou de alguma outra forma, porém separadas do simulador.

A forma de implementação dos algoritmos de escalonamento é através da programação das políticas de escalonamento com a utilização da API em C fornecida pelo Simgrid. Além de funções básicas como criação, inspeção e destruição de tarefas, são também fornecidas funções que descrevem possíveis dependências entre tarefas e para atribuição de tarefas a recursos. Mesmo com todas estas funcionalidades, alguns algoritmos necessitaram de funções específicas que foram implementadas separadamente, como por exemplo, o suporte a aplicações que usam grande quantidade de dados.

4.3. Tipos de aplicações

Os tipos de aplicações avaliadas nas simulações são determinados por tarefas independentes, considerando que uma aplicação pode ser dividida em tarefas e estas subdivididas em qualquer número de tarefas assíncronas independentes [Bharadwaj, V., Robertazzi, T. G. and Ghose D. 1996].

As tarefas foram modeladas seguindo parâmetros de quantidade de computação e quantidade de dados de entrada. A quantidade de computação para cada tarefa varia entre 25 e 4250 *Gflops*, considerando o intervalo entre 25 e 150 *Gflops* para tarefas pequenas, 150 e 2000 *Gflops* para tarefas de tamanho médio, e por fim, entre 2000 e 4250 *Gflops* para tarefas grandes. Para o segundo parâmetro, quantidade de dados, a variação está entre 0,125 e 125 *MBytes*(ou 1Mbit e 1Gbit), considerando a quantidade de dados pequena para variação entre 0,125 e 1,25 *MBytes*, média entre 1,25 e 12,5 *MBytes* e, grande entre 12,5 e 125 *MBytes*. Outro ponto importante a ser lembrado é que o intervalo de chegada das tarefas está relacionado a sua carga computacional, sendo que tarefas pequenas chegam com mais freqüência do que tarefas maiores.

As combinações adequadas entre quantidade de computação e quantidade de dados determinam os tipos de aplicações e/ou tarefas modeladas para simulação e avaliação dos algoritmos. Algumas combinações foram descartadas por caracterizarem situações irreais. Os casos testados estão descritos na próxima seção.

5. Testes e resultados

Os testes foram feitos sobre estudos de casos, em que cada caso determina um tipo de aplicação alvo. As simulações dos cinco algoritmos avaliados neste trabalho, *Dynamic*-FPLTF, WQR, *XSufferage*, *Storage Affinity* e MQ-H, foram executadas no cluster do GSPD (Grupo de Sistemas Paralelos e Distribuídos) da Unesp de São José do Rio Preto. Para cada configuração foram executadas cinqüenta simulações, sendo que em cada simulação variam as características das tarefas, dos recursos do grid e das conexões de rede, como tempo de chegada para tarefas e disponibilidade para recursos e conexões. Os resultados apresentados aqui são a média dos resultados de cada simulação.

As simulações foram feitas para uma quantidade de tarefas variando entre 10, 25, 50, 100 e 200 tarefas, a fim de observar o comportamento dos algoritmos diante o aumento do número de tarefas. Para diferenciação entre os tipos de tarefas, foram utilizadas abreviações para quantidade carga computacional e volume de dados: P - para pequena, M - média, G - grande e V – para variada, em que o conjunto variado implica numa mistura aleatória de tarefas pequenas, médias e grandes. A seguir são descritos os estudos de casos e apresentados os resultados.

5.1. Estudo de caso 1: Tarefas *CPU-bound* – Quantidade mínima de dados

O primeiro estudo de caso mostra resultados obtidos nas simulações de tarefas *CPU-bound* com quantidade mínima de dados (0,125 a 1,25 *MBytes*), e variação de pequena, média, grande e variada para carga computacional. Para melhor identificação dos testes realizados, a tabela 1 apresenta uma legenda das combinações avaliadas.

Tabela 1. Combinações avaliadas para o estudo de caso 1

		Volume de dados			
		P	M	G	V
Carga computacional	P	**PP**	-	-	-
	M	**MP**	-	-	-
	G	**GP**	-	-	-
	V	**VP**	-	-	-

A figura 2 mostra o gráfico para a combinação PP em que foram feitas simulações para tarefas de pequena carga computacional com pequena quantidade de dados. O gráfico mostra o tempo médio (s) gasto segundo a simulação da execução de um número determinado de tarefas. O algoritmo *Dynamic*-FPLTF foi melhor que todos os outros, chegando a ser 20% mais rápido que o segundo colocado (*Storage Affinity*) para 200 tarefas.

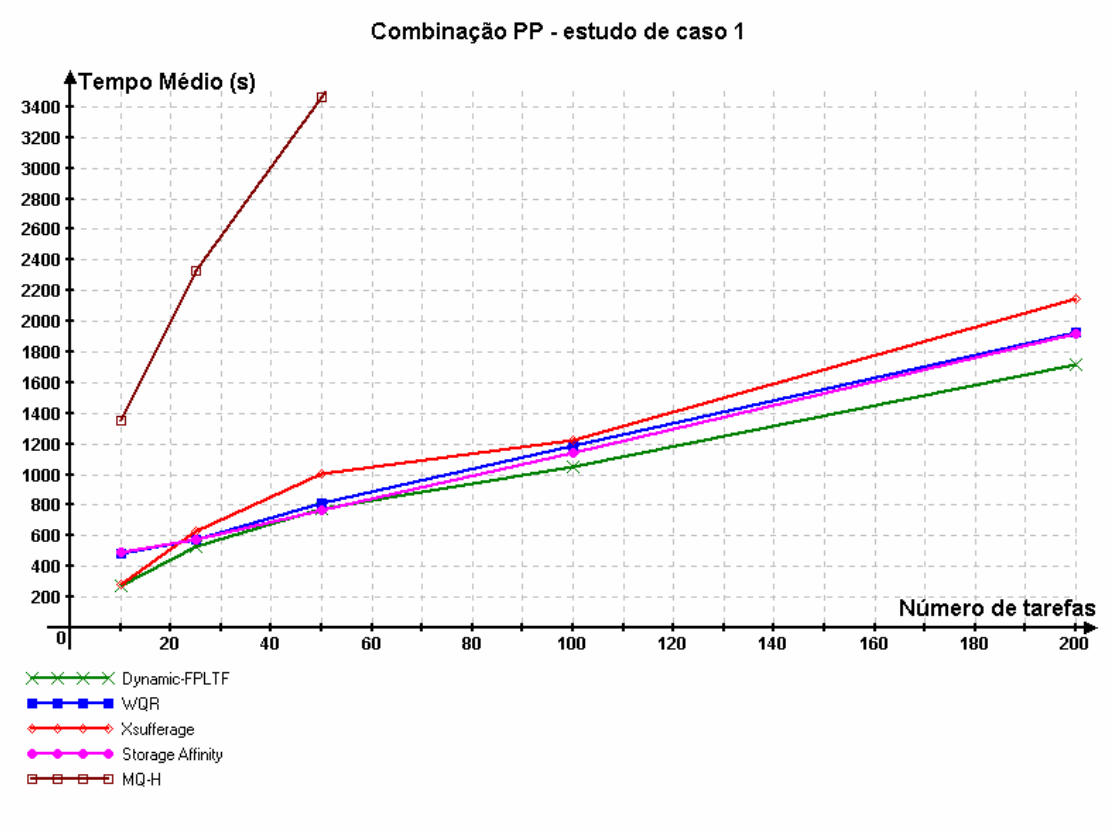

Figura 2. Combinação PP (processamento pequeno e volume de dados pequeno).

Na figura 3 para a combinação MP, fica mais clara a diferença entre o primeiro e segundo colocado no resultado do *makespan* (tempo de simulação), com o aumento do número de tarefas. O algoritmo *Dynamic*-FPLTF mantém sua vantagem de 20% em relação ao *Storage Affinity* para 200 tarefas. Já o algoritmo MQ-H mostrou não ser adequado para esta combinação, assim como para a combinação anterior.

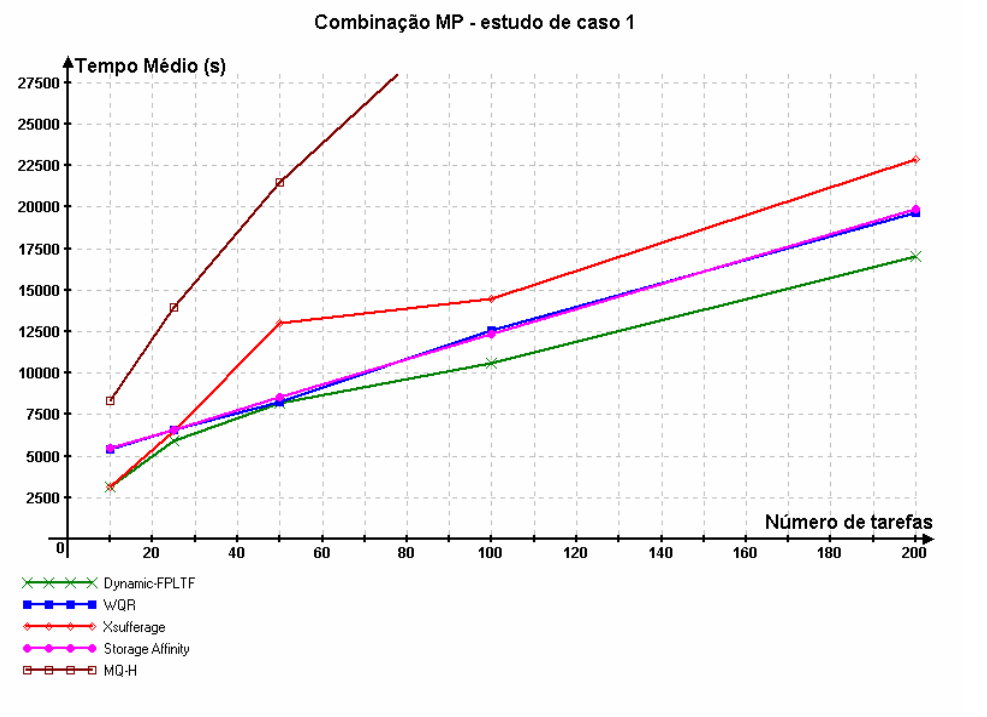

Figura 3. Combinação MP (processamento médio e volume de dados pequeno).

Na combinação GP apresentada no gráfico da figura 4, os algoritmos tiveram resultados muito próximos. A maior diferença entre as medidas de *makespan* foi de 30%, porém para 10 tarefas, e conforme aumenta o número de tarefas, essa diferença não ultrapassa 6% aproximadamente, desconsiderando obviamente os resultados do algoritmo MQ-H, que se mantém como o pior algoritmo.

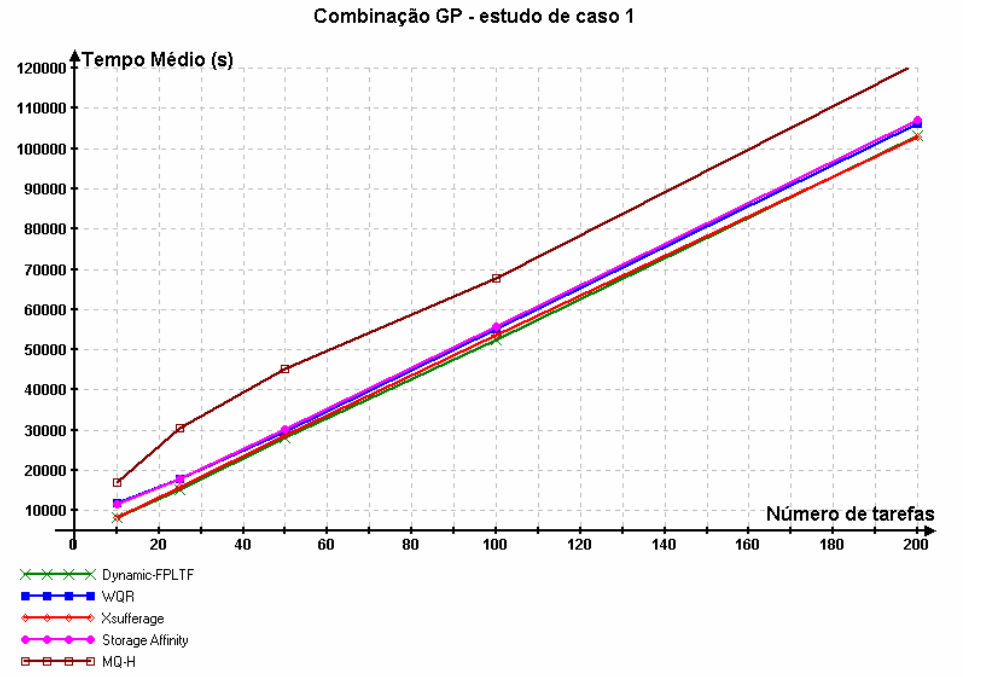

Figura 4. Combinação GP (processamento grande e volume de dados pequeno).

A próxima combinação foi feita com o intuito de avaliar o comportamento dos algoritmos diante uma carga computacional variável. A figura 5 mostra resultados semelhantes aos obtidos para as demais configurações. A diferença entre os algoritmos não ultrapassou 2% para nenhuma quantidade de tarefas, exceto para o MQ-H.

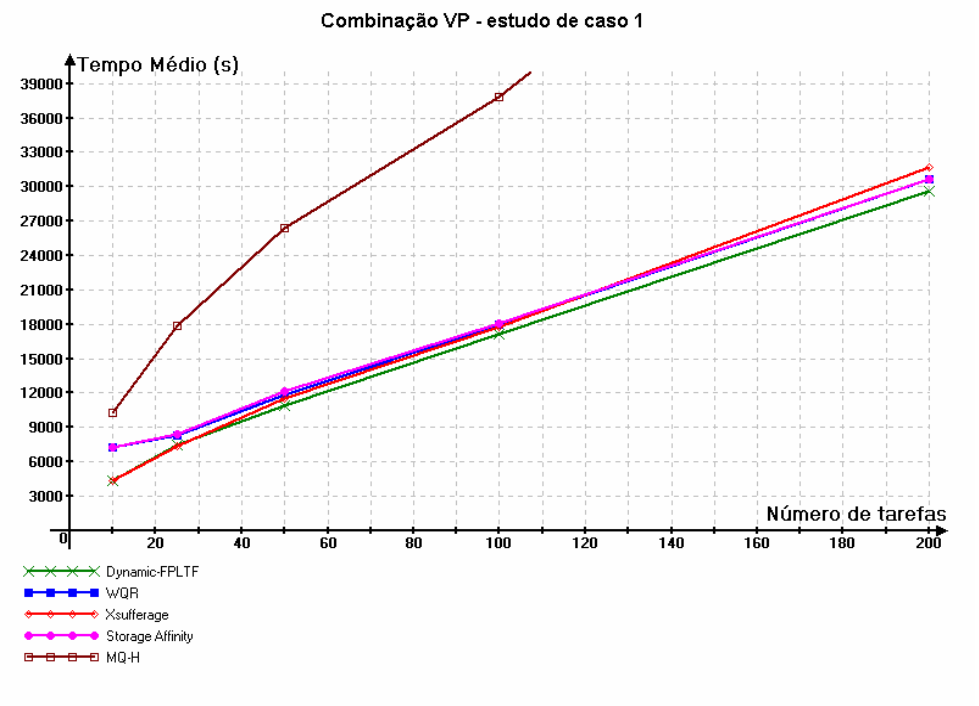

Figura 5. Combinação VP (processamento variado e volume de dados variado).

5.2. Estudo de caso 2: Tarefas *CPU-bound* – Dados centralizados

Neste segundo estudo de caso são consideradas tarefas *CPU-bound* com quantidade grande de dados, porém os dados estão centralizados, ou seja, os dados chegam junto com a tarefa e são enviados junto com ela ao o nó escolhido para sua execução. A tabela 2 mostra a legenda para as combinações deste estudo de caso. Também são avaliadas as combinações nos casos médio e variado para ambos os parâmetros, carga computacional e quantidade de dados, e também para o caso variado em ambos.

Tabela 2. Combinações avaliadas para o estudo de caso 2

		Volume de dados			
		P	M	G	V
Carga computacional	P	-	-	**PG**	-
	M	-	**MM**	**MG**	-
	G	-	-	**GG**	-
	V	-	-	**VG**	**VV**

O primeiro gráfico apresentado é o da combinação PG na figura 6, em que tarefas de pequena carga computacional possuem grande quantidade de dados a serem processados. Os resultados relevantes desta combinação são o melhor desempenho do

Dynamic-FPLTF, e uma inesperada piora do algoritmo *XSufferage*, que deveria se destacar neste segundo estudo de caso pois em teoria é mais adequado a tarefas com grande quantidade de dados centralizados. Uma possível explicação para esse desempenho é que o custo de avaliação das conexões da rede não é compensado pela baixa carga computacional das tarefas simuladas.

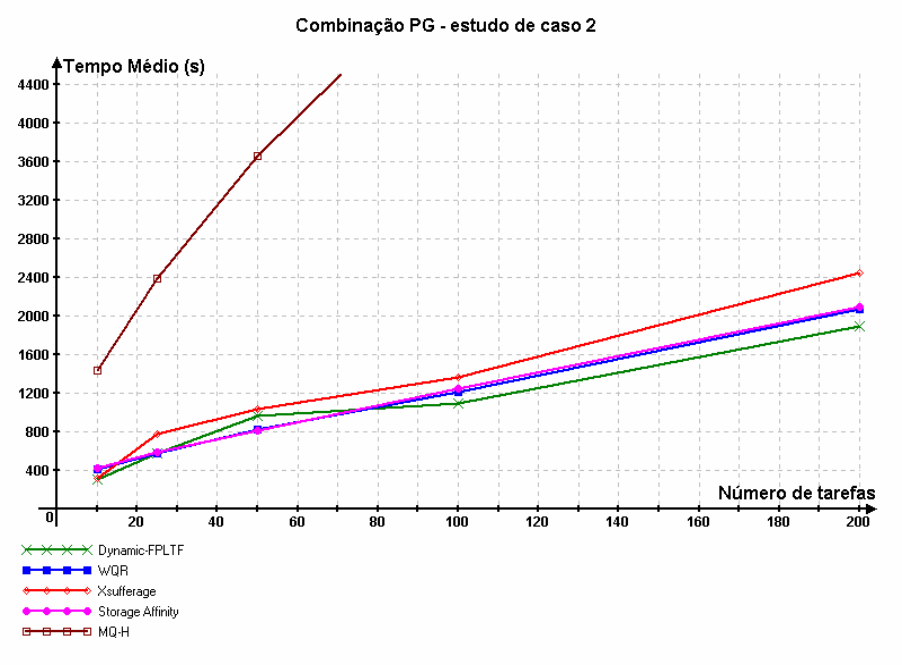

Figura 6. Combinação PG (processamento pequeno e volume de dados grande).

Na figura 7, com a combinação MG, os resultados do *makespan* apontam o mesmo problema de desempenho relacionado ao algoritmo *XSufferage*, e confirmam os bons resultados do *Dynamic*-FPLTF.

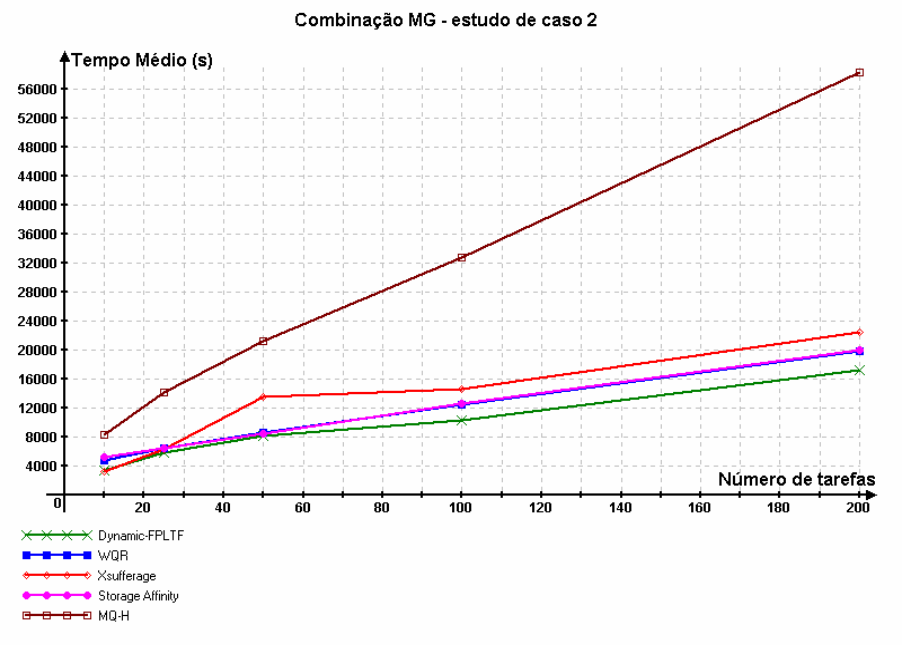

Figura 7. Combinação MG (processamento médio e volume de dados grande).

Na combinação GG mostrada na figura 8, assim como na MG, o desempenho do algoritmo MQ-H não se mostra tão inferior aos demais, o que pode ser creditado ao aumento da carga computacional das tarefas e, conseqüentemente, melhor ocupação das filas de escalonamento. Aqui, entretanto, o algoritmo *XSufferage* mostra melhores resultados, atendendo às suas características para este segundo caso.

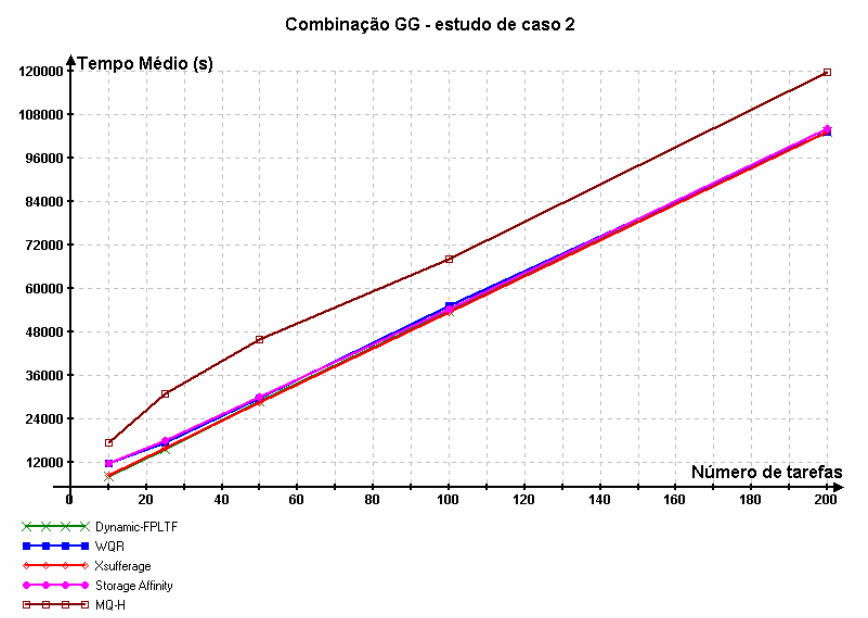

Figura 8. Combinação GG (processamento grande e volume de dados grande).

No gráfico da figura 9, apresentam-se os resultados obtidos com a simulação de tarefas com carga computacional variada com grande quantidade de dados. Os resultados obtidos mostram uma maior proximidade dos tempos dos algoritmos diante uma ampla variação de carga computacional.

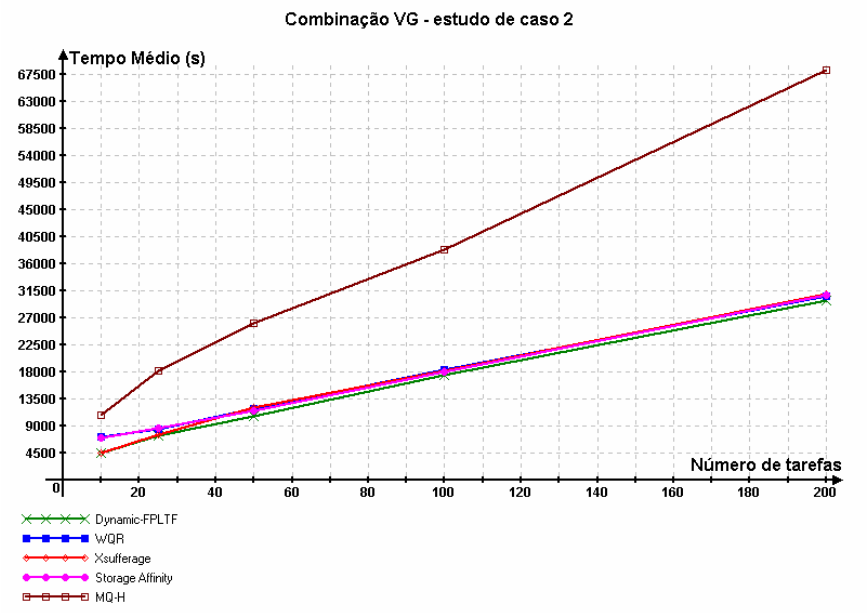

Figura 9. Combinação VG (processamento variado e volume de dados grande).

Nas figuras 10 e 11 são apresentados resultados de simulações feitas considerando cenários mais gerais, com tarefas com carga computacional e quantidade de dados de tamanho médio ou variado. Os resultados da combinação MM, figura 9, confirmam o melhor desempenho do *Dynamic*-FPLTF, com uma diferença em torno de 15% em relação ao segundo colocado, para 200 tarefas, e relembra o problema de desempenho relatado sobre o *XSufferage*. Já os resultados da combinação VV, figura 10, reforçam aqueles obtidos para a combinação VG, porém com valores de *makespan* dos algoritmos mais distribuídos, devido à variação da quantidade de dados.

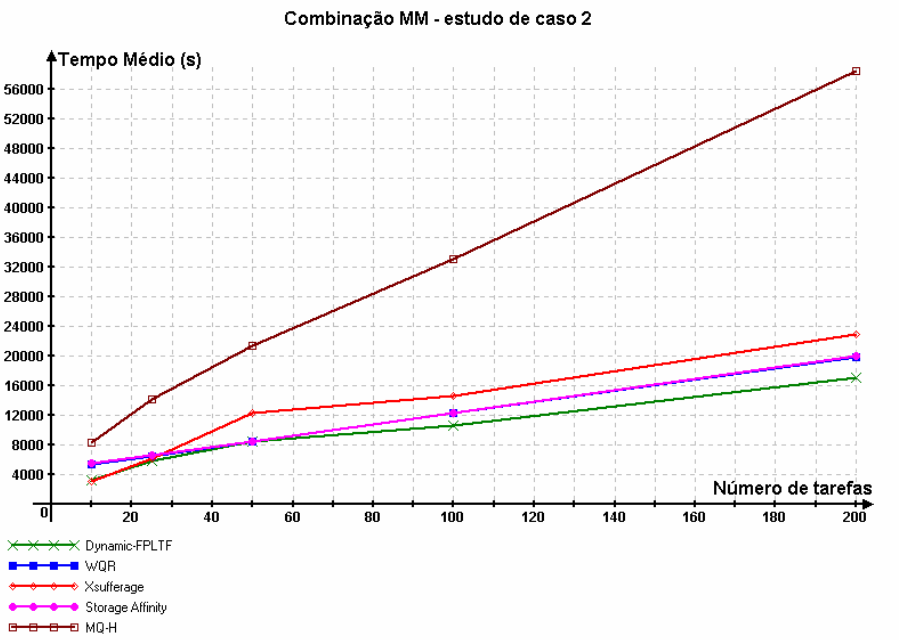

Figura 10. Combinação MM (processamento médio e volume de dados médio).

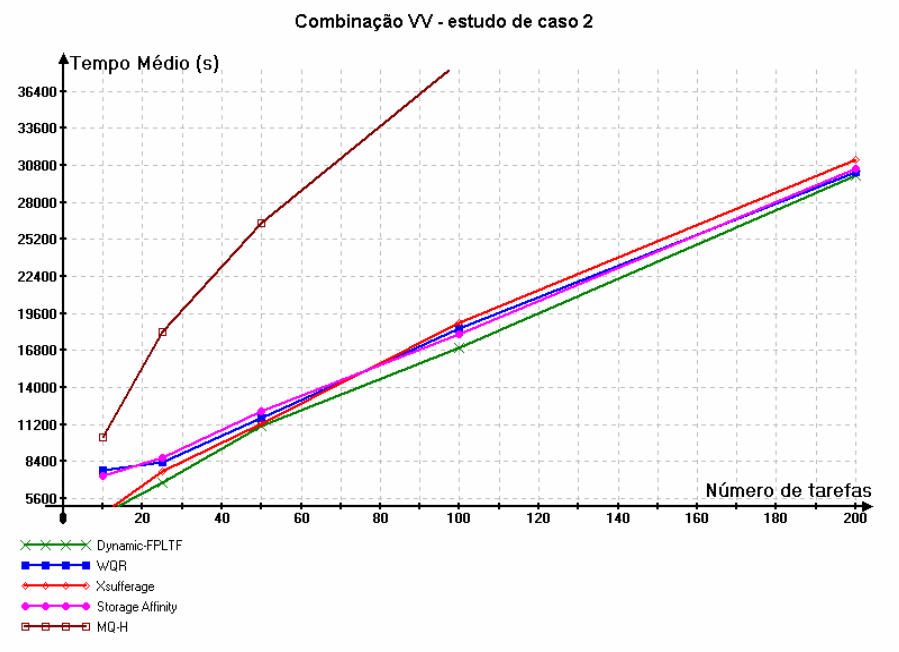

Figura 11. Combinação VV (processamento variado e volume de dados variado).

5.3. Estudo de caso 3: Tarefas *CPU-bound* – Dados distribuídos

Neste terceiro estudo de caso são também consideradas tarefas *CPU-bound* com quantidade grande de dados, mas agora os dados estão distribuídos pelo grid. As tarefas chegam com uma quantidade mínima de dados (só o binário), e a maior parte dos dados utilizados por esta tarefa está distribuída pelo grid. A tabela 3 mostra a legenda para as combinações deste estudo de caso. Também são avaliadas as combinações nos casos médio e variado para ambos parâmetros.

Tabela 3. Combinações avaliadas para o estudo de caso 3

		Volume de dados			
		P	M	G	V
Carga computacional	P	-	-	**PG**	-
	M	-	**MM**	**MG**	-
	G	-	-	**GG**	-
	V	-	-	**VG**	**VV**

O intuito deste estudo de caso é avaliar o comportamento de todos algoritmos, mas principalmente do algoritmo *Storage Affinity*, que em teoria é o mais adequado para a situação de dados distribuídos. Nos gráficos das figuras 12 e 13 nota-se, como esperado, uma vantagem do algoritmo *Storage Affinity* em relação aos outros no intervalo de 10 a 100 tarefas. Depois, para 200 tarefas, o *Dynamic*-FPLTF apresenta um melhor desempenho, em especial na combinação MG.

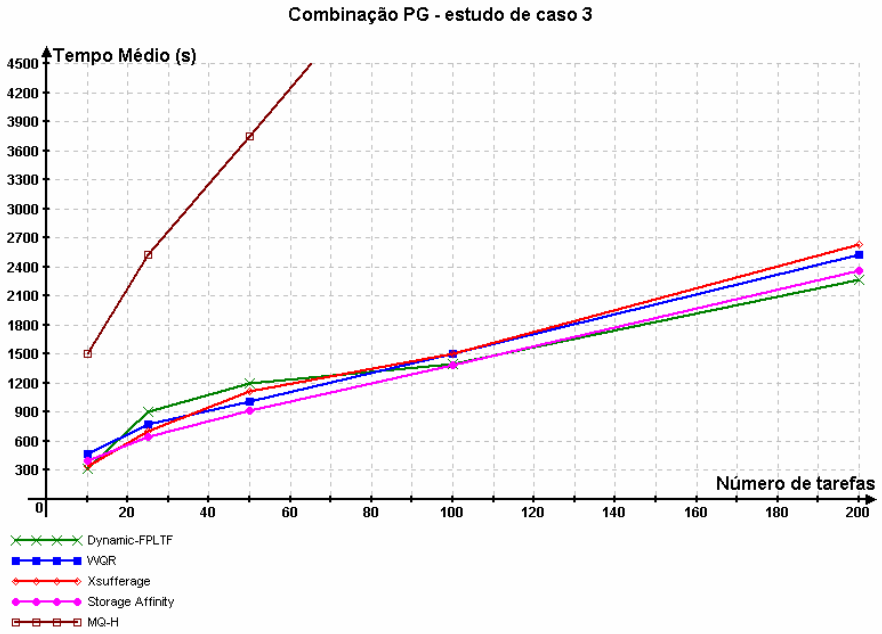

Figura 12. Combinação PG (processamento pequeno e volume de dados grande–caso 3).

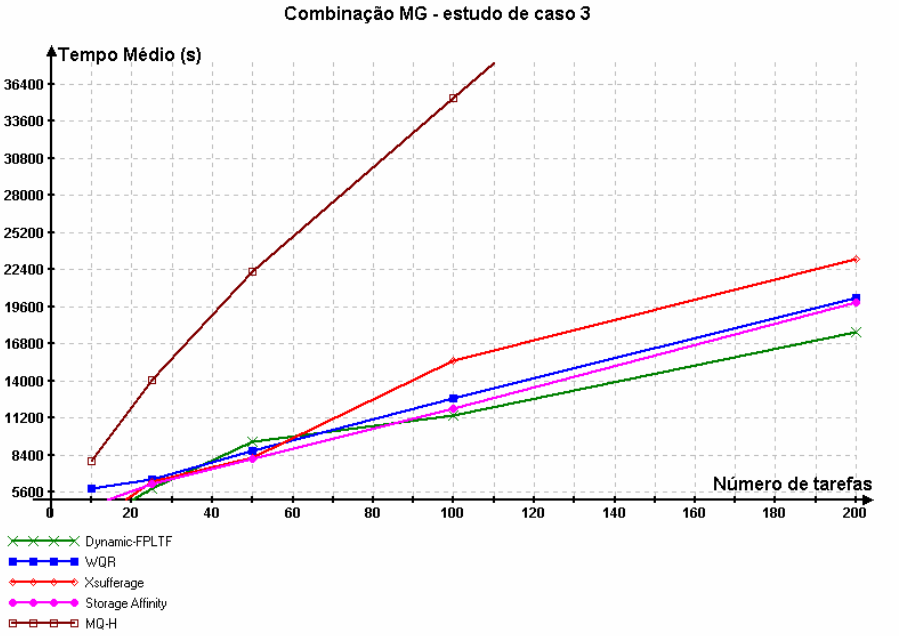

Figura 13. Combinação MG (processamento médio e volume de dados grande-caso 3).

Nas combinações GG e VG, os algoritmos tiveram resultados bem próximos, com uma diferença menor de 5% entre os tempos de simulação.

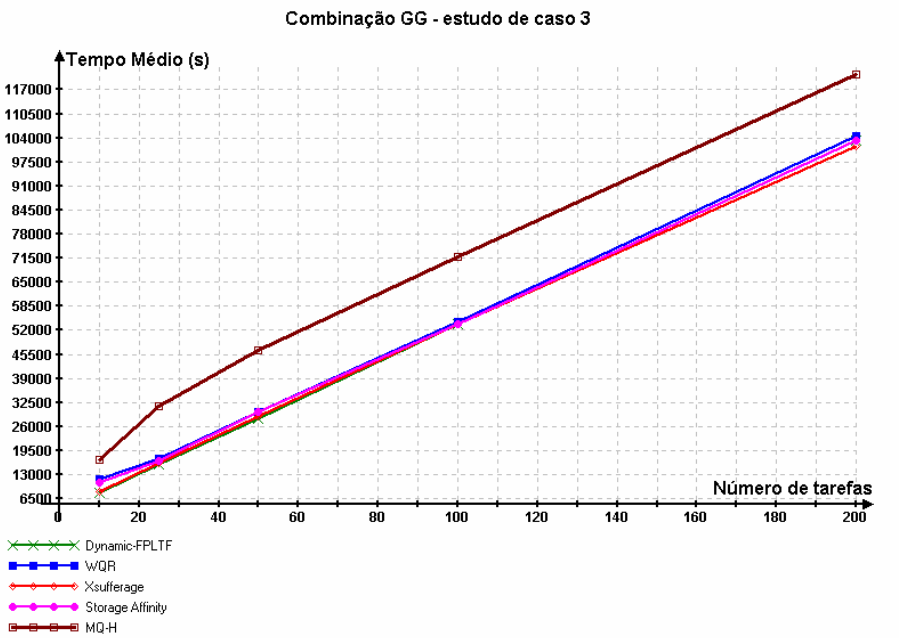

Figura 14. Combinação GG (processamento grande e volume de dados grande-caso 3).

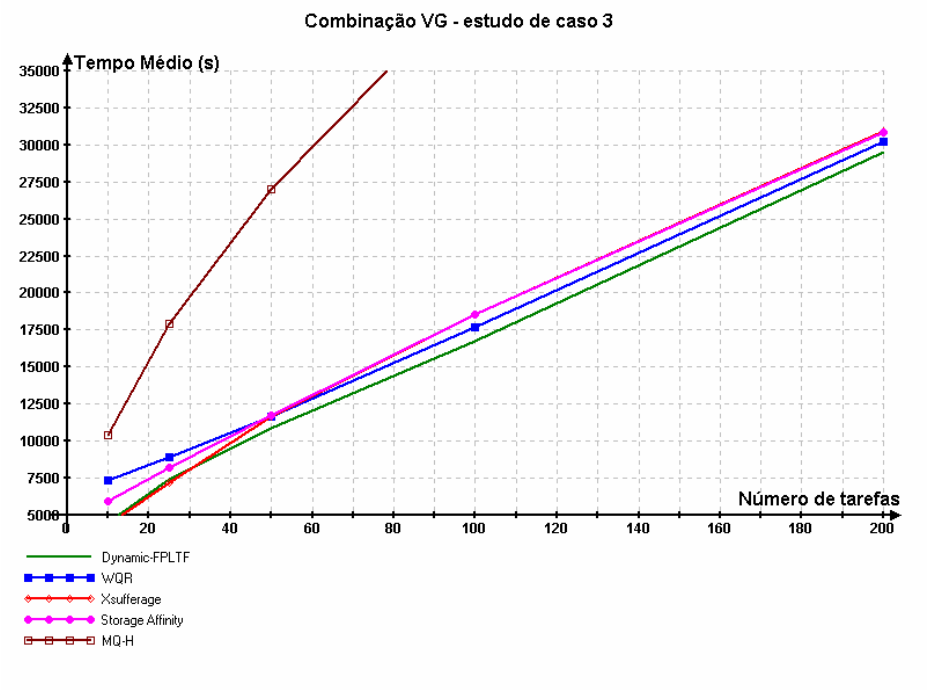

Figura 15. Combinação VG (processamento variado e volume de dados grande-caso3).

A combinação MM, figura 16, que caracteriza um caso mais geral para o tipo de tarefa, forneceu resultados esperados e, confirmou a eficiência do *Storage Affinity* para dados distribuídos. O *Storage Affinity* foi melhor para um pequeno número de tarefas e depois se manteve em segundo lugar, atrás apenas do *Dynamic*-FPLTF para um grande número de tarefas.

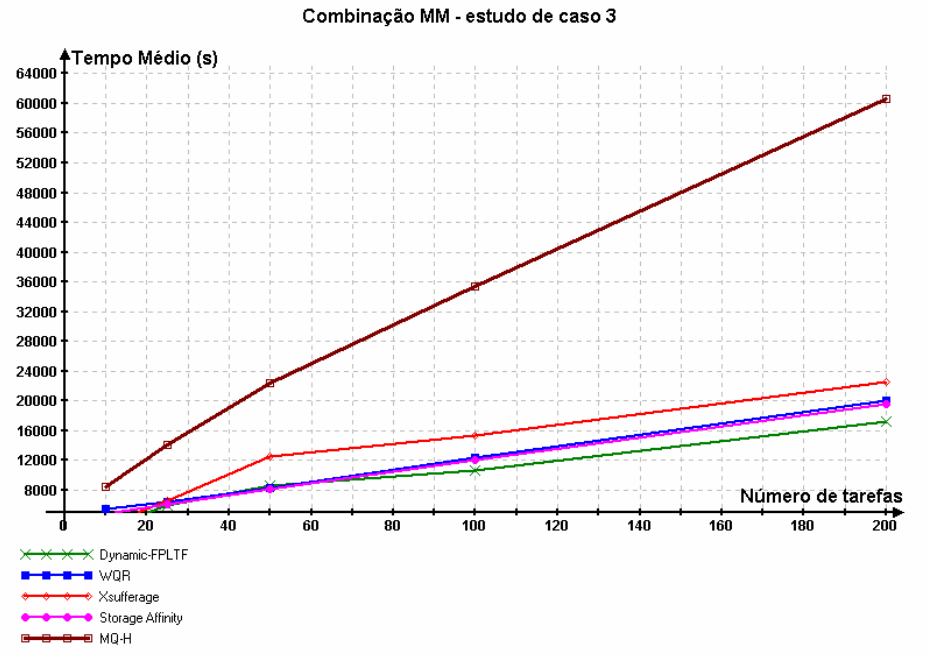

Figura 16. Combinação MM (processamento médio e volume de dados médio-caso 3).

No último gráfico deste terceiro estudo de caso, figura 17, são mostrados os resultados para um caso bem geral. Porém, a ampla variação dos parâmetros carga computacional e quantidade de dados não forneceu nenhuma informação muito relevante, a não ser pela maior proximidade de desempenho dos algoritmos nesta combinação.

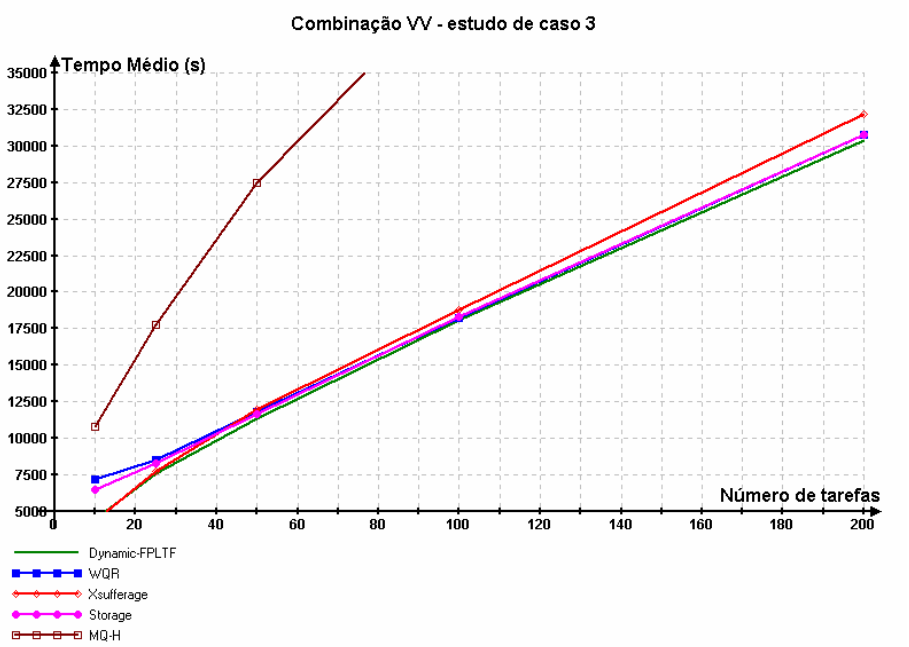

Figura 17. Combinação VV (processamento variado e volume de dados variado-caso 3).

5.4. Resultados sobre sobrecarga de processamento

A sobrecarga de processamento está ligada aos algoritmos WQR e *Storage Affinity*, que são os únicos que fazem replicação de tarefas. Assim, a quantidade de réplicas criadas durante uma simulação é o que determina o *overhead* de processamento. Em todas as simulações foi definido um máximo de três réplicas por tarefa, procurando limitar a sobrecarga de processamento. No gráfico da quantidade de réplicas, figura 18, são mostrados os números de réplicas para algumas das combinações discutidas anteriormente. Como pode ser notado, quanto maior o número de tarefas menor a proporção do número de réplicas. Para 10 tarefas o número de réplicas chega ao limite de três réplicas por tarefa, já para 200 tarefas a quantidade de réplicas gira em torno de 170, que é menos que um terço do número máximo de réplicas (600).

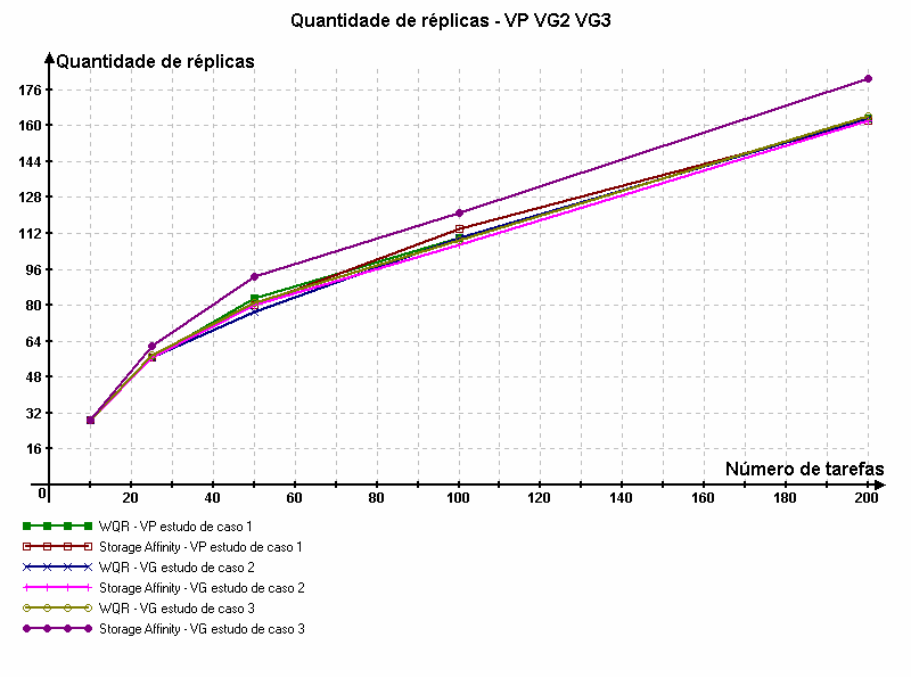

Figura 18. Quantidade de réplicas.

Como dito anteriormente, o gráfico da sobrecarga de processamento está ligado diretamente à quantidade de réplicas. Como mostra a figura 19, as curvas do gráfico nela apresentadas são muito parecidas com as curvas do gráfico da figura 18. O *overhead* de processamento apresentado segue praticamente a mesma proporção da quantidade de réplicas. Para 10 tarefas, por exemplo, o *overhead* chega a ser três vezes maior do que a carga computacional de todas as tarefas. Já para 200 tarefas, ele não ultrapassa um terço do total de carga computacional.

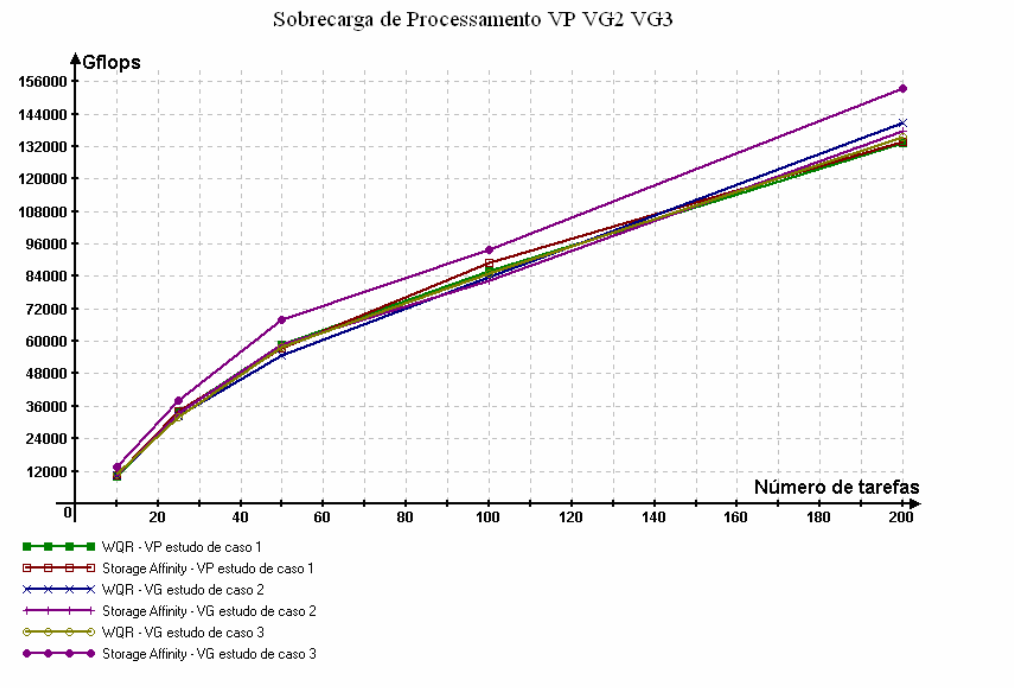

Figura 19. Sobrecarga de processamento.

6. Conclusões e trabalhos futuros

Este trabalho permite algumas conclusões interessantes sobre escalonamento de tarefas em grids. Uma primeira conclusão é que algoritmos dinâmicos como o *Dynamic*-FPLTF, por serem mais adaptativos, reagem bem diante da dinamicidade de um grid computacional. Algoritmos como o WQR e *Storage Affinity* tentar tratar a dinamicidade do grid com a criação de réplicas, mas isto causa uma enorme sobrecarga de processamento.

Uma outra conclusão diz respeito ao algoritmo MQ-H, que apresentou resultados muito ruins, em parte pela forma que foi implementado. Sua implementação seguiu o modelo mais simples do algoritmo, que faz com que todas as tarefas fiquem durante muito tempo no primeiro nível de hierarquia, pois os tempos limites de fila e de execução são grandes. A sobrecarga de processamento também é grande nesse algoritmo, mas devido os resultados obtidos não foi necessário fazer esse levantamento. A melhor maneira de implementar este algoritmo seria examinar o tempo limite de fila das tarefas já enfileiradas no primeiro nível, e caso esse tempo estivesse perto de estourar, não atribuir uma nova tarefa a esse nível, mas sim para um próximo nível. Embora isso melhore consideravelmente o desempenho do algoritmo, não foi tratado nesse trabalho por ser, na realidade, uma adaptação do modelo proposto.

Em relação ao algoritmo *XSufferage*, que é considerado dinâmico em partes, por verificar a situação da rede, pode-se concluir que ele perde muito por desconsiderar máquinas que já estão ocupadas. Em muitos casos, mesmo com uma enorme largura de banda disponível é melhor deixar uma tarefa esperando por um processador bom do que atribuí-la a um ruim.

Os tipos de aplicações alvo para cada algoritmo podem ser definidos de acordo com os resultados. Assim:

- Para o *Storage Affinity*, tarefas com grande quantidade de dados *distribuídos* (acima até do limite máximo estabelecido neste trabalho) e carga computacional de média pra pequena são seu melhor cenário.
- Para o WQR, tarefas de pouca carga computacional causam muitas réplicas, então as mais adequadas seriam tarefas de grande carga computacional pra diminuir a sobrecarga de processamento, e com quantidade de dados variada, pois este atributo não causou perturbação.
- Para o *XSufferage*, as tarefas devem ter grande quantidade de dados, mas centralizados, ou seja, fornecidos juntos com a tarefa, e pequena quantidade de carga computacional para que o tempo de processamento não degrade o tempo economizado na transferência dos dados.
- Para o *Dynamic*-FPLTF se destacou em todos os casos, mas seguindo sua idéia principal, tarefas CPU-*bound* com quantidade mínima de dados são as mais adequadas, pois ele não faz nenhuma verificação em relação ao comportamento da rede do grid.

Os trabalhos futuros envolvem uma avaliação precisa com uma quantidade maior de dados para as tarefas, a fim de limitar o intervalo de ação do algoritmo *Storage Affinity*, a avaliação de possíveis mesclas de algoritmos à procura de melhores

resultados. E o desenvolvimento de uma nova heurística de escalonamento que melhor se adapte as alterações de um grid.

Referências

He X., Sun X.-H., and Laszewski G. (2002) "A QoS Guided Scheduling Algorithm for the Computational Grid", In the Proc. of the International Workshop on Grid and Cooperative Computing (GCC02), Hainan, Chian.

Schopf J.M. (2002) "A General Architecture for Scheduling on the Grid", Special Issue on Grid Computing, J. Parallel and Distributed Computing.

Paranhos D., Cirne W. and Brasileiro F. V. (2003) "Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids", In Proc. of the Euro-Par (Lecture Notes in Computer Science), v. 1. p. 169-180.

Menascé D., Saha D. and Porto S. et al. (1995) "Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures", In Journal of Parallel and Distributed Computing, pp. 1-18.

Casanova, H., Legrand A., Zagorodnov D. and Berman F. (2000) "Heuristics for Scheduling Sweep Applications in Grid environments", In 9th Heterogeneous Computing Systems Workshop.

Santos-Neto, E. et al (2004) "Escalonamento de Aplicações que processam grande quantidade de dados em Grids Computacionais" Dissertação de mestrado, Universidade Federal da Campina Grande – UFCG.

Fujimoto, N. and Hagihara, K. (2004) "A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks." In Proceedings of the 2004 Symposium on Applications and the internet-Workshops (SAINT 2004 Workshops) (January 26 - 30, 2004). SAINT-W. IEEE Computer Society, Washington, DC, 674.

Ibarra, O. and Kim C. (1977) "Heuristic algorithms for scheduling independent tasks on nonindentical processors", In journal of ACM, 24(2): 280-289.

Silberstein, M., Geiger, D., Schuster, A. and Livny, M. (2006) "Scheduling Mixed Workloads in Multi-grids: The Grid Execution Hierarchy", In *Proceedings of the 15th IEEE Symposium on High Performance Distributed Computing (HPDC)*.

Legrand, A., Marchal L. and Casanova H. (2003) "Scheduling distributed applications: The Simgrid simulation framework", In Proc. of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan, pp.138-145.

Simgrid Project Web Page (2007), http://simgrid.gforge.inria.fr.

Bharadwaj, V., Robertazzi, T. G. and Ghose D. (1996) "Scheduling Divisible Loads in Parallel and Distributed Systems", In IEEE Computer Society Press, Los Alamitos, CA, USA.

Cirne, W. and Santos-Neto E. (2005) "Grids Computacionais: da Computação de Alto Desempenho a Serviços sob Demanda", SBRC, Brasil.