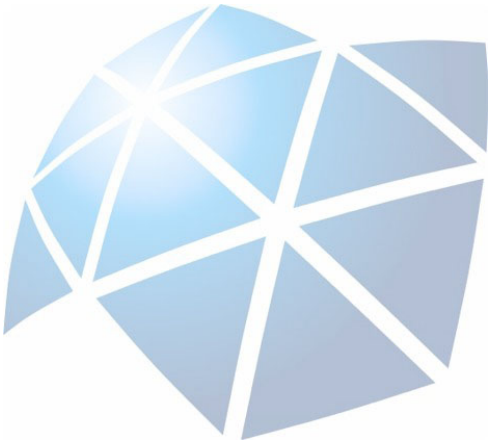

Universidade Estadual Paulista

FEIS - Faculdade de Engenharia de Ilha Solteira

DEE - Departamento de Engenharia Elétrica

Pós-Graduação em Engenharia Elétrica

Estudo Especial II

Avaliação de algoritmos de escalonamento para tarefas em Grids

José Nelson Falavinha Júnior

São José do Rio Preto-SP, setembro/2006

Sumário

Lista de Figuras

Capítulo 1

Introdução Geral

O escalonamento de tarefas em Grid se trata da atividade de escalonar um conjunto de aplicações originadas de usuários diferentes para um conjunto de recursos computacionais espalhados por locais distintos, com o intuito de maximizar a utilização do sistema. Este escalonamento envolve combinar a necessidade da aplicação com a disponibilidade do recurso e garantir qualidade de serviço entre eles [9]. Preocupações como quantidade de informação a ser processada, localização da informação, quantidade de recursos disponíveis e a taxa de comunicação entre eles, são os principais pontos de decisão dos escalonadores.

Este estudo avalia o comportamento dos principais algoritmos de escalonamento para tarefas em Grids, cada algoritmo tem suas próprias características envolvendo diferentes aplicações alvo, políticas de escalonamento, informações úteis, e portanto cada um fornece seus resultados particulares que apesar disso, podem ser comparados e avaliados considerando um ambiente comum para execução de um mesmo tipo de aplicação.

Os algoritmos de escalonamento foram implementados utilizando o simulador para Grids, SimGrid, que além de permitir a simulação do comportamento de um ambiente computacional como Grid, torna possível a implementação e validação dos algoritmos, através da simulação de troca de mensagens, envio e recebimento de tarefas, execução de tarefas, atrasos de comunicação, e outras funcionalidades.

O restante deste segundo estudo se encontra organizado da seguinte forma. No próximo capítulo serão relembradas as características fundamentais dos algoritmos avaliados neste estudo. No Capítulo 3 é apresentada uma descrição do simulador SimGrid, desde suas características até suas funcionalidades que permitiram avaliar os algoritmos e obter os resultados aqui apresentados. O capítulo 4 traz uma breve definição dos ambientes simulados para teste dos algoritmos, mostrando os resultados particulares de cada algoritmo e as comparações entre eles. E por fim, o último capítulo apresenta as conclusões deste estudo, os fatos importantes descobertos com esta avaliação e os próximos passos a serem seguidos para a tese de doutorado.

Capítulo 2

Algoritmos de escalonamento

Escalonamento em Grids é definido como o processo de tomar decisões de escalonamento envolvendo recursos sobre múltiplos domínios administrativos. Este processo envolve três fases principais [6]: **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. Essas fases, e os passos pertencentes a cada uma, são mostradas na figura 2.1.

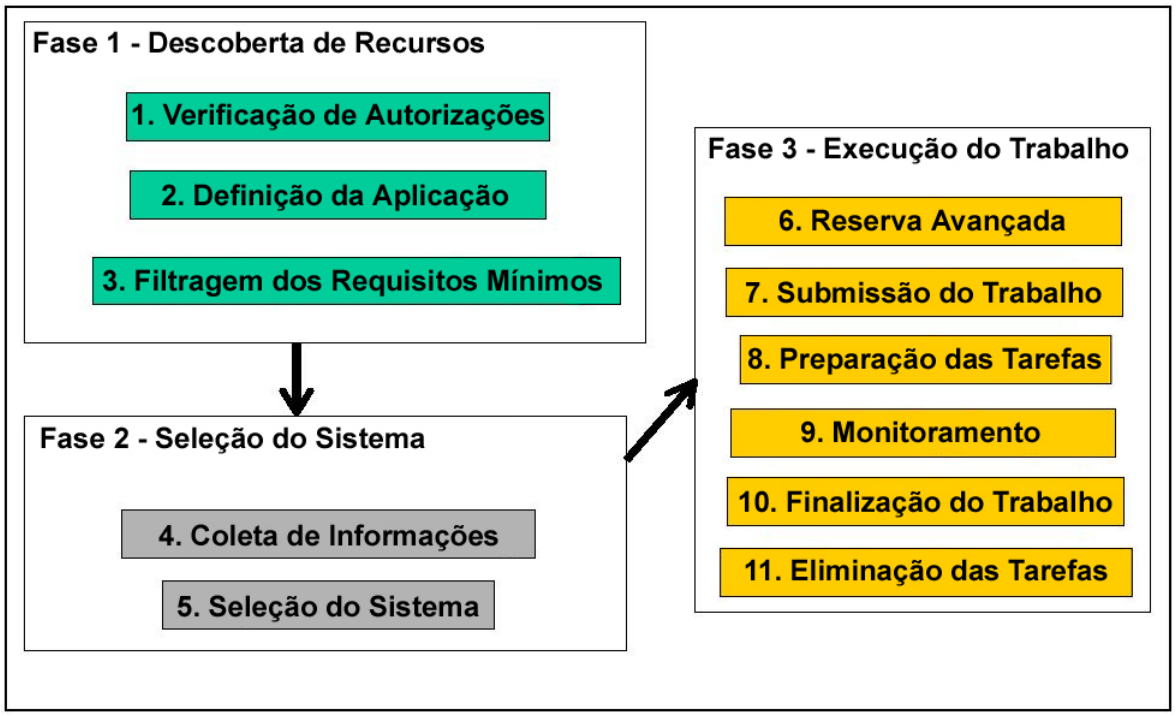

Figura 2.1: Fases do escalonamento

Os algortimos de escalonamento tem suas características próprias, e portanto, cada um atua de maneira diferenciada em determinadas fases do escalonamento. A seguir serão relembrados os algoritmos de escalonamento avaliados neste estudo, descrevendo funcionalidades e metodologias para que fiquem claras as diferenças entre cada um.

2.0.1 *Dynamic FPLTF*

O dinamismo e a heterogeneidade dos recursos presentes em um Grid, levaram ao desenvolvimento de algoritmos dinâmicos como o *Dynamic* FPLTF [3] que foi desenvolvido à partir do estátifco *Fastest Processor to Largest Task First* (FPLTF) [2]

O *Dynamic* FPLTF necessita de três tipos de informação para escalonar as tarefas: *Task Size*, *Host Load* e *Host Speed*. *Host Speed* é a velocidade de um *host* e seu valor é relativo, por exemplo um maquina com *Host Speed*=2 executa uma tarefa duas vezes mais rápido do que uma máquina com *Host Speed*=1. *Host Load* representa a parte da máquina que não está disponível para a aplicação. E finalmente, *Task Size* é o tempo necessário para uma máquina com *Host Speed*=1 completar uma tarefa quando *Host Load*=0.

No começo do algoritmo, um variável TBA(*Time to Become Available*) para cada *host* é inicializada com 0 e as tarefas são organizadas em ordem decrescente de tamanho. Portanto, um tarefa grande é alocada primeiro. Uma tarefa é alocada ao *host* que oferecer o melhor tempo de execução (CT - *completion time*) para ela, onde:

- $CT = TBA + TaskCost$;
- $TaskCost = (TaskSize/HostSpeed)/(1 - HostLoad)$;

Quando uma tarefa é alocada em um *host*, o valor TBA correspondente deste *host* é incrementado pelo *Task Cost*. 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 rodando são escalonadas novamente até que todas as máquinas fiquem em uso. Este esquema continua até todas as tarefas serem completadas. Esta estratégia tenta minimizar os efeitos do dinamismo do Grid.

2.0.2 *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 [3]. 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 uma ordem aleatória. 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 se torna disponível e não há mais nenhuma tarefa pendente para executar. Neste momento, *Workqueue* já terminou seu trabalho e apenas aguarda a finalização de todas as tarefas. Porém, o WQR inicia sua fase de replicação para tarefas que ainda estão executando, e assim que a tarefa original ou uma de suas réplicas finalizarem, as outras são interrompidas.

2.0.3 *Sufferage* e *XSufferage*

O *XSufferage* [5, 4] é um algoritmo de escalonamento baseado nas informações sobre o desempenho dos recursos para associar tarefas aos processadores. Este algoritmo aborda o impacto das grandes transferências de dados em aplicações que usam grandes quantidades de dados executando em Grids através da exploração da reutilização de dados. O *XSufferage* é uma extensão do algoritmo *Sufferage* [7] no qual a idéia básica é determinar quanto cada tarefa seria prejudicada se não fosse escalonada no processador que a executaria de forma mais eficiente. Portanto, o *Sufferage* prioriza as tarefas de acordo com o valor que mede o prejúízo de cada tarefa. 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 *XSufferage* é o método usado para calcular o valor do *sufferage*. O algoritmo *XSufferage* leva em consideração a tranferência dos dados de entrada da tarefa durante o cálculo dos tempos de execução. Isso implica no uso das informações usadas também pelo *Sufferage* mais a largura de banda disponível na rede que conecta os recursos, diferente do *Sufferage* que necessita somente das informações relacionadas a CPU e o tempo estimado de execução da tarefa.

Um ponto 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 melhor conexão de rede receberia todas as tarefas. Esse fato não ocorre no algoritmo *Dynamic FPLTF* que considera todos os recursos, estando eles em uso ou não.

2.0.4 *Storage Affinity*

O *Storage Affinity* [4] foi feito com o intuito de explorar a reutilização de dados para melhorar o desempenho de aplicação 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 em um dado site, ou seja, quanto mais bytes de entrada da tarefa já estiver armazenado no site, mais próximo a tarefa estará do site, pois possui mais *storage affinity*. Portanto, o valor do *storage affinity* de uma tarefa com um site é o número de bytes pertencentes à entrada da tarefa que já estão armazenados no site.

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

O algoritmo *Storage Affinity* é dividido em duas fases. Na primeira fase, as tarefas são associadas para os processadores do Grid e a ordem de escalonamento é determinada através do cálculo do valor de *storage affinity* das tarefas. Após determinar as afinidades, a tarefa que apresentar o maior valor para *storage affinity* é escalonada em um processador

do site com o qual a tarefa apresentou maior afinidade. A segunda fase consiste da replicação de tarefas e é iniciada quando não há mais tarefas aguardando para executar e há, pelo menos, um processador disponível. Um réplica pode ser criada para qualquer tarefa em execução, porém nesta fase há um critério e uma ordem de prioridade para criação de réplicas. Considerando que o grau de replicação de uma tarefa é o número de réplicas criadas para esta tarefa, então ao iniciar a fase de replicação, os seguintes critérios são verificados na escolha de qual tarefa deve ser replicada: i) a tarefa deve estar executando e ter um valor de afinidade positivo, ou seja, alguma porção de sua entrada já deve estar presente no site de algum dos processadores disponíveis no momento; ii) o grau de replicação corrente da tarefa deve ser o menor entre as tarefas que atendem o critério anterior; e iii) a tarefa deve apresentar o maior valor de afinidade entre as tarefas que atendem o critério (ii).

O algoritmo finaliza quando todas as tarefas que estão executando completam, e quando um tarefa completa sua execução as outras réplicas são interrompidas. Até isto ocorrer, a replicação de tarefas continua, mas caso seja desejável conter o desperdício de recursos ocasionado pela replicação pode-se limitar o grau de replicação máximo.

No próximo capítulo será apresentado o simulador SimGrid, o qual foi utilizado para implementação e avaliação dos algoritmos descritos neste capítulo.

Capítulo 3

Simulador SimGrid

O Simgrid [1, 8] 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. Neste estudo foi implementado um ambiente de Grid com vários recursos, neste caso clusters, para execução de um grande número de tarefas e avaliação dos algoritmos de escalonamento já mencionados.

Um componente importante do processo de simulação é a modelagem do recurso. Na definição de uma máquina, ou de um cluster por exemplo, são atribuidas variáveis como lantência, ou seja o tempo necessário para acessar o recurso, e taxa de serviço ou poder computacional. O poder computacional é definido neste simulador como sendo o número de unidades de trabalho por unidade de tempo. Nas implementações foram utilizadas as unidades de: segundos (s) para latência, e Mflop/s para o poder computacional.

Outra parte integrante da simulação de um ambiente Grid é a topologia de interconexão dos recursos. A maioria dos algortimos de escalonamento assumem que trabalham sobre uma topologia completamente conectada, e topologias diferentes desta são consideradas como casos especiais da anterior. O Simgrid trabalha com a topologia completa e permite a implementação das conexões de rede entre os recursos. A definição de variáveis como largura de banda ou taxa de transferência, e latência da rede, é o que caracteriza estas conexões e o usuário pode também especificar quais dessas conexões estão habilitadas ou não. Para simular o comportamento de roteadores simples, podem ser criadas múltiplas conexão entre as máquinas do ambiente. Todas estas características possibilitam a implementação de uma ampla variadade de infra-estruturas computacionais.

Além disso, o SimGrid não faz nenhuma distinção entre transferência de dados e computação: ambos são vistos como **tarefas** e é de responsabilidade do usuário garantir que tarefas de computação sejam escalonadas em processadores, e transferência de arquivos para conexões de rede. É importante lembrar 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, mas separada 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. Esta API permite manipular dois tipos de dados estruturados: um para recursos e um para tarefas. Um recurso é descrito pelo nome, um conjunto de métricas relacionadas a desempenho, traços e constantes. Uma tarefa é descrita pelo nome, custo, e estado. Além de funções básicas como criação, inspeção e destruição, são fornecidas funções que descrevem possíveis dependências entre tarefas e funções para designar tarefas para recursos.

Com todas estas funcionalidades e características, a implementação e validação dos algoritmos de escalonamento citados no capítulo são portanto tarefas factíveis tendo em vista a utilização do simulador SimGrid. Mesmo assim, alguns dos algoritmos necessitam de funcionalidades específicas que foram implementadas separadamente, como por exemplo o suporte a aplicações que usam grande quantidade de dados. No capítulo seguinte são apresentados os cenários de testes dos algoritmos, a avaliação de cada um, resultados e comparações.

Capítulo 4

Testes e resultados

Os ambientes construídos no simulador SimGrid para avaliação dos algoritmos são Grids compostos de clusters, ou seja, os recursos computacionais de processamento são clusters, e a definição de cada um é feita como se este cluster fosse uma simples máquina com um determinado nome e poder computacional. Foram também designadas as conexões entre cada cluster e vários caminhos possíveis como se houvesse um roteador simples. O poder computacional dos clusters integrantes varia entre 4000 a 80000 Mflops/s, e as conexões de rede que ligam os clusters variam entre 100 Mbps a 2Gbps.

O Grid simulado é composto por dez grupos de computadores mais o escalonador, ou seja, dez clusters mais uma determinada máquina ou cluster responsável pelo escalonamento, cada um com seu respectivo nome e poder computacional. Também é importante lembrar que o escalonamento em Grids não é um processo centralizado, neste estudo, apenas foram simulados a chegada de tarefas em um determinado *peer* do Grid e o escalonador localizado neste *peer*. Cada integrante do Grid tem sua fila de tarefas e seu escalonador de tarefas, portanto para avaliar o algoritmo de escalonamento basta simular um ponto do Grid, em que ocorre o escalonamento.

Escolhido o integrante do Grid que receberá tarefas e as escalonará nos clusters do Grid, este recebe o nome de escalonador. O escalonador roda dois processos concorrentes, um que determina a chegada das tarefas e outro que faz o escalonamento e envia as tarefas para serem processadas. Os clusters de trabalho, são dez e rodam basicamente dois processos também concorrentes, um que recebe as tarefas para execução, trata todas dependências e a executa, e um outro processo que atende requisições de dados.

Os ambientes de teste dos algoritmos de simulação foram implementados de acordo com três cenários distintos para aplicações em Grids. O primeiro cenário é caracterizado pelo fato de que as tarefas a serem escalonadas são estritamente tarefas de computação, ou seja, tarefas que necessitam somente de processamento, sem utilização de dados de entrada ou qualquer outra coisa. No segundo, as implementações do Grid foram voltadas para aplicações que utilizam grande quantidade de dados, porém a tarefa e os dados se encontram em poder do escalonador e este é responsável pelo envio destes dados de entrada até o cluster escolhido para execução da tarefa. No terceiro e último cenário, também são consideradas aplicações que utilizam grande quantidade de dados de entrada, no entanto agora, os dados estão espalhados pelos clusters do Grid, sendo necessário portanto a

transferência dos dados para o cluster escolhido no escalonamento para depois a tarefa ser executada.

Na implementação dos algoritmos, foram avaliados os quatro algoritmos citados no capítulo 2, são eles: WQR - *Workqueue with Replication* com nível máximo de réplicas igual a 3, *Dynamic FPLTF*, *Xsufferage* e *Storage Affinity* com nível de réplicas igual a 3 também. Estes algoritmos foram avaliados sobre os três cenários já descritos e o grupo de tarefas para escalonamento foi o mesmo para todos os algoritmos, para que as métricas resultantes pudessem ser analisadas e comparadas. Abaixo segue uma pequena legenda correspondente aos cenários para facilitar o entendimento das tabelas e resultados.

- SP - Somente processamento, para o primeiro cenário;
- PDC - Processamento mais dados centralizados, para o segundo cenário;
- PDE - Processamento mais dados espalhados, para o último cenário.

A seguir são apresentados dois estudos de casos, o primeiro estudo de caso avalia a simulação dos três cenários descritos anteriormente para um conjunto de tarefas com características diversas, isto é, sem estabelecer rígida variação da quantidade de processamento de cada tarefa nem a quantidade de dados de entrada. No segundo estudo de caso foi delimitada a variação da quantidade de processamento para cada tarefa, visando encontrar informações importantes sobre cada algoritmo.

4.1 Estudo de caso 1

Neste estudo de caso, as tarefas submetidas para o escalonador possuem atributos que variam dentro de um amplo conjunto. Abaixo são descritas os intervalos de variação para os atributos das tarefas escalonadas neste estudo de caso, estabelecendo limites para os cenários descritos anteriormente quando é levado em consideração a quantidade de dados:

- Quantidade de processamento necessário para execução da tarefa:
  - de 5000 a 8000000 Mflops
- Quantidade de dados:
  - 0 MB (megabytes) para SP; 12,5 a 250 MB para PDC e PDE.

Os resultados obtidos na simulação dos algoritmos são apresentados nas seções seguintes, todos os tempos obtidos nos testes são dados em segundos e carga de processamento em Mflops (*megaflops*).

4.1.1 Cenário SP

As tabelas seguinte mostram os resultados das implementações dos algoritmos com um conjunto de tarefas apenas de processamento, ou seja, no cenário SP. Na tabela 4.1.1

são consideradas primeiramente 50 tarefas, as quais tarefas vão chegando entre o intervalo de tempo 0 a 300 segundos. E depois, 100 tarefas com intervalo de chegada variando de 0 a 600 segundos.

Algoritmos	**50 tarefas**	**100 tarefas**
WQR	350.223	650.223
Dyn - FPLTF	342.730	642.730
Xsufferage	342.659	642.659
Storage Affinity	351.861	651.861

Tabela 4.1: Tempo(s) de simulação do escalonamento de tarefas no cenário SP

Neste primeiro cenário foi possível notar que os algoritmos *Dynamic FPLTF* e *Xsufferage* apresentaram melhor desempenho do que os outros e permaneceram muito próximos entre si. No entanto, a diferença entre os tempos de simulação não é exorbitante, o que indica que as tarefas não exigem enorme quantidade de processamento, ou que os clusters são rápidos e conseguem dar conta das tarefas facilmente. O WQR e *Storage Affinity* também permanecem muito próximos e um pouco além dos outros, o que é justificado pela criação de réplicas. A tabela 4.1.1 mostra a quantidade de réplicas criadas em cada um e o desperdício de computação em Mflops ocasionado pelas réplicas tendo em vista o total de Mflops necessários para execução das tarefas.

Algoritmos	**Réplicas**	**Desperdício**	**Processamento 50 tarefas**
WQR	124	3.01115e+07	3.05303e+07
Storage Affinity	128	3.22023e+07	3.05303e+07
	Réplicas	**Desperdício**	**Processamento 100 tarefas**
WQR	242	6.40423e+07	6.10605e+07
Storage Affinity	249	6.25747e+07	6.10605e+07

Tabela 4.2: Réplicas e desperdício de processamento (Mflops) - SP

Esses dados mostram que o desperdício de computação gasto com réplicas chega até ultrapassar toda carga de processamento necessária para computar todas as tarefas, o que indica que algoritmos que usam replicação não adequados no cenário SP pois além de resultarem em desperdício de processamento o tempo final de simulação foi maior.

A figura 4.1 mostra a evolução dos tempos de simulação pela quantidade de tarefas em cada algoritmo. Nesta figura é possível notar que todos os tempos permanecem muito próximos, o que confirma que a quantidade de computação requerida pelas tarefas não é muito alta.

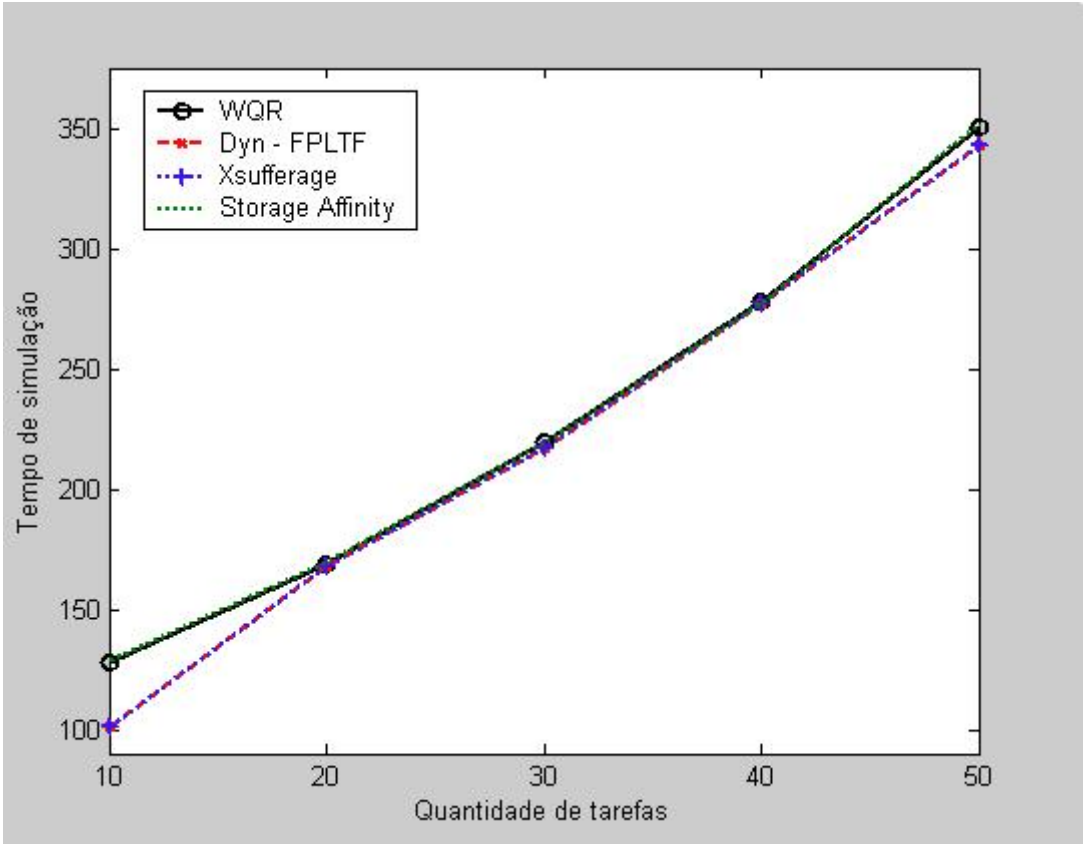

Figura 4.1: Evolução dos tempos de simulação (s) pela quantidade de tarefas - SP

4.1.2 Cenário PDC

Neste cenário, como descrito anteriormente, as tarefas são caracterizadas por carga de processamento com dados de entrada, com o fato de que os dados de entrada são enviados pelo próprio escalonador. Para manter possível comparações entre cenários, as cargas de processamento das tarefas não foram alteradas, portanto foram apenas incluídos os dados de entrada. A tabela 4.1.2 mostra os tempos de simulação obtidos, ainda levando em consideração também os mesmos intervalos de surgimento para as tarefas.

Algoritmos	**50 tarefas**	**100 tarefas**
WQR	403.541	786.323
Dyn - FPLTF	347.862	647.862
Xsufferage	349.198	649.198
Storage Affinity	442.842	747.439

Tabela 4.3: Tempo(s) de simulação do escalonamento de tarefas no cenário PDC

Ainda, neste cenário os algoritmos *Dynamic FPLTF* e *Xsufferage* apresentaram melhor desempenho do que os outros e permaneceram muito próximos entre si. No entanto, agora a diferença entre eles e os outros aumentou o que indica que este tipo de aplicação

com dados de entrada centralizados num ponto do Grid também não são favorecidas pelos algoritmos WQR e *Storage Affinity* como pode ser visto também no gráfico mostrado na figura 4.2.

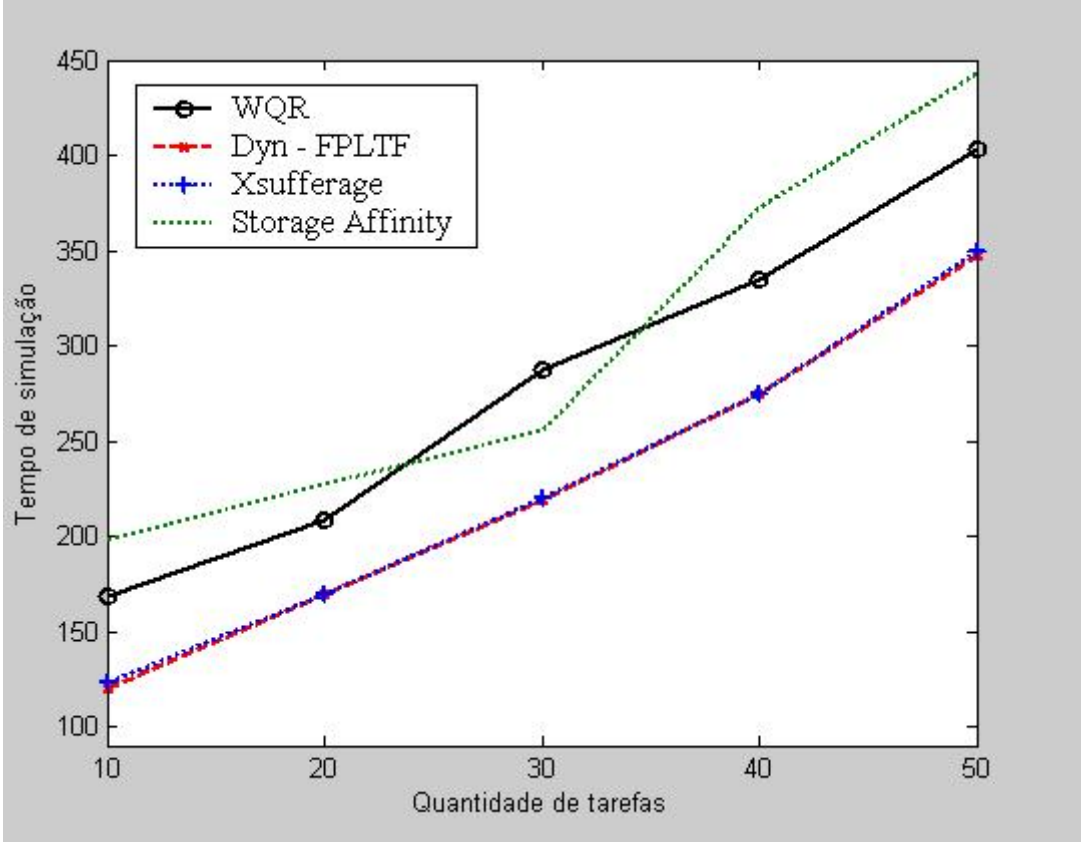

Figura 4.2: Evolução dos tempos de simulação (s) pela quantidade de tarefas - PDC

Um fato importante a ser notado é que, do cenário anterior para este, os tempos de simulação dos melhores algoritmos não tiveram grande acréscimo levando em consideração que este novo cenário gasta tempo com a comunicação dos dados. Essa fato indica que a quantidade de dados utilizada é pouca ou as conexões de rede são muito boas. Como as conexões de rede foram previamente definidas e caracterizam um ambiente comparável a grids computacionais, então a quantidade de dados deve ser o problema.

A seguir, na tabela 4.1.2 são apresentados os resultados quanto a réplicas e desperdício dos algoritmos que utilizam replicação.

Algoritmos	**Réplicas**	**Desperdício**	**Processamento 50 tarefas**
WQR	33	1.65274e+07	3.05303e+07
Storage Affinity	27	1.55561e+07	3.05303e+07
	Réplicas	**Desperdício**	**Processamento 100 tarefas**
WQR	57	3.12905e+07	6.10605e+07
Storage Affinity	34	2.57447e+07	6.10605e+07

Tabela 4.4: Réplicas e desperdício de processamento (Mflops) - PDC

Desta vez, o número de réplicas caiu consideravelmente se comparado ao cenário anterior, isto devido ao maior tempo gasto para replicação, sendo que replicar neste cenário, trata-se de criar uma cópia de uma tarefa e enviar junto com ela também os dados de entrada ao cluster escolhido para processamento. O desperdício da quantidade de processamento também caiu devido a mesma justificativa, chegando a ficar em torno da metade do total necessário para todas as tarefas. As diferenças entre um e outro são explicadas pela quantidade de troca de informações, mais necessárias no algoritmo *Storage Affinity* do que no WQR.

4.1.3 Cenário PDE

As tarefas neste cenário são caracterizadas, assim como no anterior, por carga computacional com dados de entrada, porém agora os os dados estão espalhados pelo Grid. Isto significa que quando um determinado cluster recebe uma tarefa para processar, ele deve requisitar os dados de entrada da tarefa para todos integrantes do Grid que possuem algum dado desta tarefa, até que ele tenha todos os dados desta tarefa para assim então começar a execução da tarefa. Como indicado no definição do algoritmo, é necessário manter um lista com a localização dos dados para que as requisições possam ser feitas, contudo, esse problema foi amenizado nos testes feitos neste estudo pois tendo em vista que o número de integrantes do Grid é pequeno, basta fazer requisições a todos eles, e os que tiverem dados respondem às requisições. Em um Grid real, com um maior número de integrantes talvez esta estratégia não seja adequada.

A tabela 4.1.3 mostra os tempos de simulação obtidos, também levando em consideração os intervalos para surgimento das tarefas, denifidos para o primeiro cenário.

Algoritmos	**50 tarefas**	**100 tarefas**
WQR	485.734	712.357
Dyn - FPLTF	420.708	720.708
Xsufferage	701.959	1432.5
Storage Affinity	404.492	712.99

Tabela 4.5: Tempo(s) de simulação do escalonamento de tarefas no cenário SP

Esses resultados mostraram a eficiência do algoritmo *Storage Affinity* que seleciona o cluster com maior quantidade de dados de uma tarefa para execução da mesma. No entanto os algoritmos *Dynamic FPLTF* e WQR não ficaram muito distantes, fato que reforça as conclusões tiradas nos outros cenários, de que nem a carga computacional e nem a quantidade de dados para as tarefas são muito grandes. O algoritmo *Xsufferage* apresentou um resultado que parece inadequado, mas na o que explica a diferença em relação aos outros algoritmos é fato que o *Xsufferage* leva em consideração apenas os clusters não ocupados durante o escalonamento das tarefas, e como os clusters ficam mais tempo ocupados com troca de dados, alguma tarefa com alta carga de processamento foi atribuída a um cluster de pouco poder computacional, causando uma demora na execução das tarefas. O algoritmo *Dynamic FPLTF* não sofre deste problema, pois considera até os cluster ocupados e não deixa que uma tarefa de grande carga computacional seja escalonada para um cluster de baixo poder computacional, mesmo estando um melhor ocupado, a não ser que este escalonamento compense. O gráfico da figura 4.3 mostra a evolução dos tempos de simulação pela quantidade de tarefas.

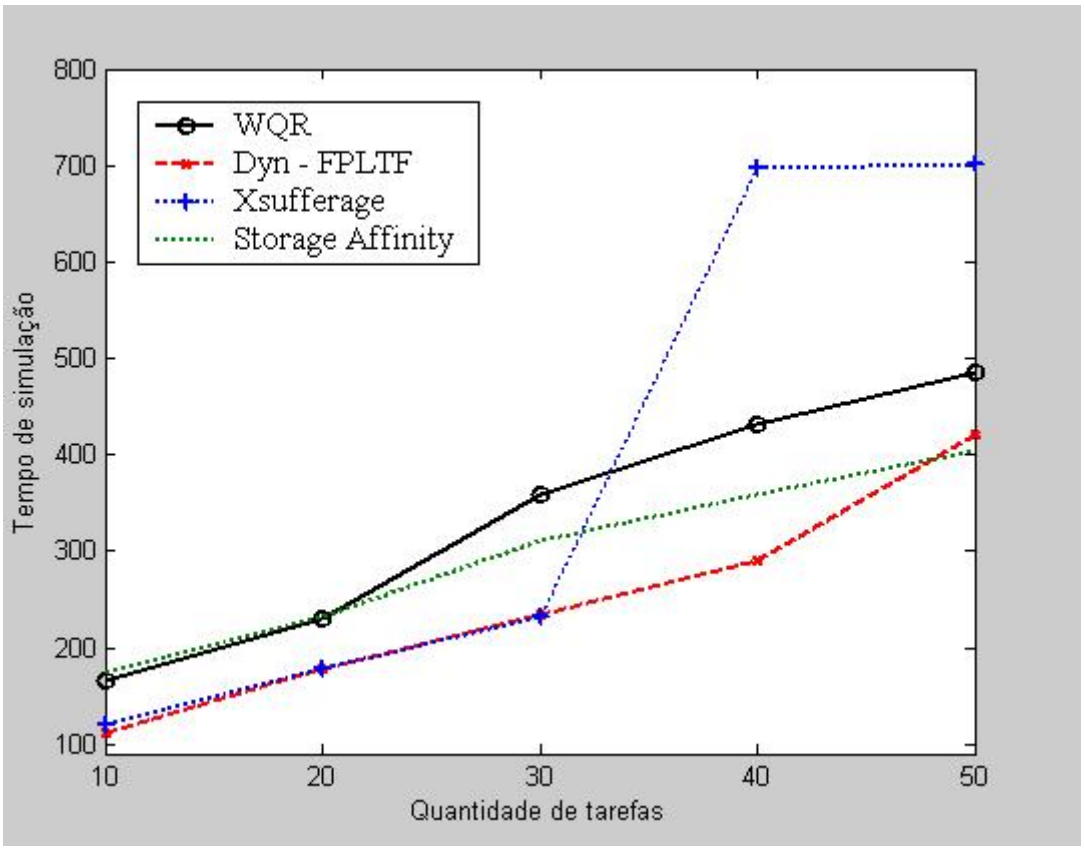

Figura 4.3: Evolução dos tempos de simulação (s) pela quantidade de tarefas - PDE

De acordo com o gráfico pode-se notar que os algoritmos *Storage Affinity* e WQR permanecem com tempos de simulação próximos, e que o algoritmo *Dynamic FPLTF* tem melhor resultado até a quantidade de tarefas chegar em 50, o que parece incoerente. Contudo, o fato que explica essa diferença, é uma dependência dos algoritmos, que fazem replicação de tarefas, ao método para cancelamento de tarefas do simulador SimGrid. O simulador permite o cancelamento das tarefas se estas estiverem em execução ou sendo

transferidas, o que atende os requisitos dos dois primeiros cenários. Neste cenário, uma tarefa é transmitida para um cluster para ser processada, porém antes de iniciar a execução, este cluster deve requisitar todos os dados utilzados por esta tarefa, então se esta tarefa é uma réplica e uma outra réplica ou a tarefa original finaliza sua execução em outro cluster durante esta requisição de dados, eu devo cancelar esta réplica, mas ela não está executando e nem sendo transferida, então ela só poderá ser cancelada assim que entrar em execução, ou seja, assim que todos os dados de entrada da tarefa forem transmitidos. Este fato explica o acréscimo de tempo para *Storage Affinity* e WQR em alguns trechos do gráfico, e portanto mesmo no ponto (50 tarefas) em que o *Storage Affinity* é superior aos outros, ele teria melhor resultado do que o apresentado no gráfico.

A seguir, na tabela 4.1.3 são apresentados os resultados quanto a réplicas e desperdício dos algoritmos que utilizam replicação, sem resultados muito relevantes para novas conclusões.

Algoritmos	**Réplicas**	**Desperdício**	**Processamento 50 tarefas**
WQR	34	1.36993e+07	3.05303e+07
Storage Affinity	31	1.47736e+07	3.05303e+07
	Réplicas	**Desperdício**	**Processamento 100 tarefas**
WQR	43	2.64338e+07	6.10605e+07
Storage Affinity	51	2.98757e+07	6.10605e+07

Tabela 4.6: Réplicas e desperdício de processamento (Mflops) - PDE

4.2 Estudo de caso 2

Neste segundo estudo de caso foram simulados os escalonadores para tarefas levando em consideração dois intervalos de variação para a carga de processamento para as tarefas. Estes testes foram feitos com o intuito de limitar a análise e avaliação dos algoritmos para situações específicas a fim de esclarecer diferenças, vantagens e desvantagens de cada um.

O primeiro intervalo para variação da carga de processamento das tarefas foi limitado entre 5000 e 100000 Mflops, ou seja, as tarefas terão seu atributo carga de processamento variando neste intervalo. Todas as outras características quanto a dados para as tarefas, poder computacional para os clusters e largura de banda das conexões permanecem as mesmas. A tabela 4.2 mostra os resultados obtidos nas simulações para cargas de tarefas dentro deste primeiro intervalo e em todos os cenários, SP, PDC e PDE.

Algoritmos	50 tarefas			100 tarefas		
	SP	PDC	PDE	SP	PDC	PDE
WQR	300.499	311.667	436.498	600.499	611.667	685.537
Dyn - FPLTF	300.185	300.685	315.655	600.185	600.685	615.655
Xsufferage	300.185	300.400	309.291	600.185	600.400	609.291
Storage Affinity	301.142	318.706	394.963	601.142	618.706	673.018

Tabela 4.7: Tempo(s) de simulação - Tarefas com carga entre 5000 e 100000 Mflops

Estes resultados mostram que o comportamento dos algoritmos permanecem muito parecidos, com tempos de simulação muito próximos. Apenas no cenário PDE, os algoritmos WQR e *Storage Affinity* diferiram de forma considerável em relação aos outros, porém isto é explicado pelo problema de cancelamento de tarefas do simulador, já descrito anteriormente. Portanto os tempos desses algoritmos seriam menores na realidade, chegando a ficar próximos dos outros.

O segundo intervalo para variação da carga de processamento das tarefas foi limitado entre 1000000 e 8000000 Mflops, ou seja, as tarefas terão seu atributo carga de processamento variando neste intervalo. Estes testes foram feitos com intuito de analisar o comportamento dos algoritmos em um ambiente com tarefas com grande carga computacional, lembrando-se que todas as outras características ainda permanecem as mesmas. A tabela 4.2 mostra os resultados obtidos nas simulações em todos os cenários, SP, PDC e PDE.

Algoritmos	50 tarefas			100 tarefas		
	SP	PDC	PDE	SP	PDC	PDE
WQR	681.702	876.309	786.309	1326.28	1544.24	1448.76
Dyn - FPLTF	767.011	824.71	771.848	1277.04	1716.54	1660.11
Xsufferage	1746.91	1807.39	1782.91	1748.44	1887.36	1820.47
Storage Affinity	684.587	922.356	785.384	1288.12	1481.43	1444.12

Tabela 4.8: Tempo(s) de simulação - Tarefas com carga entre 1000000 e 8000000 Mflops

Analisando estes resultados, percebe-se que todos os algoritmos mantêm o mesmo comportamento, ou seja, no cenário PDC o tempo de simulação é superior aos dos outros cenários. Isso ocorre porque os dados e a tarefa são transferidos juntos do escalonador para o cluster escolhido no escalonamento. Enquanto ocorre a transferência, o escalonador fica parado esperando que tudo seja transmitido, e portanto o escalonador e demais clusters livres ficam ociosos neste tempo.

Os algoritmos WQR, *Storage Affinity* e *Dynamic FPLTF* mostraram tempos de simulação muito próximos, levando em consideração a grande quantidade de carga computacional. Já os resultados de tempo para o algoritmo *Xsufferage* diferiram muito dos outros, e ficaram muito próximos entre si, mesmo com as mudanças de cenários. O que aconteceu nestas simulações, foi o escalonamento de uma tarefa muito grande (6615000

Mflops) para um cluster de baixo poder computacional (4000 Mflops/s). Em todos os cenários esta mesma tarefa foi escalonada ao mesmo cluster, levando cerca de 1655 segundos para finalizar sua execução, o que explica os tempos elevados para o algoritmo *Xsufferage*. Esse escalonamento foi tão inapropriado que mesmo quando o número de tarefas cresce para 100, todos os outros clusters conseguem executar as novas tarefas e o cluster de 4000 Mflops/s se mantem ocupado com a tarefa, sendo portanto o fator determinante para o tempo de simulação final.

No próximo capítulo serão fornecidas as conclusões desse segundo estudo, considerando todos os testes e resultados obtidos de cada algoritmo, além de determinar testes futuros e trabalhos futuros para a tese de doutorado.

Capítulo 5

Conclusão

Neste estudo foram avaliados os principais algoritmos para escalonamento de tarefas em Grids computacionais, o desenvolvimento e a implementação de cada algoritmo foi um fator determinante para o melhor entendimento do funcionamento de todos eles e ampliou o conhecimento sobre as características de um ambiente computacional como Grid através do uso do simulador SimGrid.

Os resultados das avaliações e estudos de casos dos algoritmos permitiram identificar a eficácia de algoritmos dinâmicos para escalonamento de tarefas em Grids. Os algoritmos WQR, *Storage Affinity* têm seu dinamismo caracterizado pela criação e distribuição de réplicas pelos cluster ociosos. Já o algoritmo *Dynamic FPLTF* analisa a carga de processamento de todos os clusters do Grid no momento de escalonar uma tarefa, não permitindo que uma tarefa muito grande seja atribuída a um cluster muito ruim, a não ser que compense. Do outro lado, o *Xsufferage* não é dinâmico, mesmo levando em consideração as conexões de rede do Grid, seu problema está relacionado ao fato de considerar apenas clusters livres no momento do escalonamento, causando atribuições inapropriadas como mostrado nos resultados das avaliações.

Os trabalhos que seguirão este estudo estão relacionados a novos testes levando em consideração a quantidade de dados para as tarefas, estabelecendo limites de variação para uma melhor avaliação dos algoritmos nessas condições. Com o conhecimento adquirido, também é possível pensar em variações ou combinações dos algoritmos com o intuito de conseguir melhores resultados para um caso geral de aplicações em Grids. Devido a heterogeneidade dos Grids, estes ambientes computacionais podem ter tanto tarefas de alta carga computacional quanto tarefas que utilizam grande quantidade de dados, e portanto um algoritmo que atenda a todas as tarefas não importando suas características especificas seria mais útil e adequado. Enfim, a seqüência deste estudo para a tese de doutorado segue este linha de pesquisa, na procura de qualidades dos algoritmos a fim de mesclar as mais relevantes e adequadas para implementação de um método ou algoritmo mais eficiente e com melhor desempenho.

Referências Bibliográficas

[1] A. Legrand, L. Marchal, and H. Casanova. *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, May 12–15 2003, pp. 138-145.

[2] D. Menasc, D. Saha and S. Porto et al. *Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures*. Journal of Parallel and Distributed Computing, pp. 1-18. 1995.

[3] D. Paranhos, W. Cirne, F. V. Brasileiro. *Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids*, 2003.

[4] E. Santos-Neto. *Escalonamento de Aplicações que processam Grande Quantidade de Dados em Grids Computacionais*. Dissertação de mestrado, Universidade Federal de Campina Grande - UFCG, Março 2004.

[5] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, *Heuristics for Scheduling Parameter Sweep Applications in Grid environments*, in 9th Heterogeneous Computing Systems Workshop (HCW 2000).

[6] J.M. Schopf, *A General Architecture for Scheduling on the Grid*, Special Issue on Grid Computing, J. Parallel and Distributed Computing, April 2002.

[7] O. Ibarra and C. Kim. *Heuristic algorithms for scheduling independent tasks on nonindentical processors*. Journal of the ACM, 24(2):280–289, Apr 1977.

[8] SimGrid Project Web Page, *http://simgrid.gforge.inria.fr/*, (Maio, 2006).

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