Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Dissertação de Mestrado

Avaliação de Heurísticas de Escalonamento de Aplicações Bag-of-Tasks em Grids Computacionais Adaptativas à Disponibilidade de Informação

Nelson Alves da Nóbrega Júnior

Campina Grande, Paraíba, Brasil
Agosto - 2006

Avaliação de Heurísticas de Escalonamento de Aplicações Bag-of-Tasks em Grids Computacionais Adaptativas à Disponibilidade de Informação

Nelson Alves da Nóbrega Júnior

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Francisco Vilar Brasileiro

(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA

NÓBREGA JÚNIOR, Nelson Alves da

N337A

Avaliação de Heurísticas de Escalonamento de Aplicações Bag-of-Tasks em Grids Computacionais Adaptativas à Disponibilidade de Informação.

Dissertação (mestrado), Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, Coordenação de Pós-graduação em Informática, Campina Grande - Paraíba, Agosto de 2006.

71 p. Il.

Orientador: Francisco Vilar Brasileiro

Palavras-Chave:

1 - Sistemas Distribuídos
2 – Grid Computacional
3 – Computação Paralela
4 – Heurísticas de Escalonamento
5 – Disponibilidade de informação

CDU - 681.3.066D

"AVALIAÇÃO DE HEURÍSTICAS DE ESCALONAMENTO DE APLICAÇÕES BAG-OF-TASKS EM GRIDS COMPUTACIONAIS ADAPTATIVAS À DISPONIBILIDADE DE INFORMAÇÃO"

NELSON ALVES DA NÓBREGA JÚNIOR

DISSERTAÇÃO APROVADA EM 22.08.2006

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Orientador

PROF. WALFREDO DA COSTA CIRNE FILHO, Ph.D
Examinador

PROF. RENATO ANTÔNIO CELSO FERREIRA, Ph.D
Examinador

CAMPINA GRANDE – PB

Resumo

O escalonamento de aplicações em grids computacionais consiste numa atividade bastante complexa devido à grande heterogeneidade, larga distribuição e dinamismo desses ambientes. No intuito de realizar o mapeamento entre tarefas e recursos de forma eficiente, os escalonadores de grid aplicam heurísticas de escalonamento. As heurísticas de escalonamento existentes podem ser classificadas em duas abordagens: i) heurísticas *bin-packing* e ii) heurísticas de replicação. A primeira abordagem requer informação completa e precisa sobre as aplicações (ex.: tempo estimado de execução de todas as tarefas que compõem a aplicação) e o ambiente de grid (ex.: carga de processamento e velocidade de todos os processadores do grid) para realizar as decisões de escalonamento. Esta abordagem é visivelmente inadequada para ambientes distribuídos como um grid, onde não há um controle administrativo centralizado e conseqüentemente, não é possível garantir que essas informações sempre estarão disponíveis ou, quando disponíveis, serão corretas. Por conta disso, a segunda abordagem não faz uso de informação. Em vez disso, ela aplica o princípio de replicação de tarefas, conseguindo obter desempenho comparável às heurísticas baseadas em informação. Em contrapartida, seu bom desempenho é conseguido ao custo de um consumo extra de recursos devido à replicação. Nesse trabalho investigamos essas duas abordagens de escalonamento e propomos uma nova abordagem que aproveita as vantagens de cada uma delas. A abordagem proposta consiste numa solução adaptativa ao ambiente de grid quanto à disponibilidade de informações. Esta nova abordagem proporciona um processo de escalonamento mais eficiente uma vez que reduz o desperdício de recursos do grid, sem comprometer o desempenho da aplicação.

Abstract

Scheduling application on computational grids is a difficult task. This is due to high heterogeneity, large distribution and the dynamic nature of the grid environment. In order to map tasks onto resources in a efficient way, grid schedulers apply scheduling heuristics. The existing scheduling heuristics can be classified in two approaches: i) *bin-packing schedulers* and ii) *replication schedulers*. The first approach requires complete and accurate information about the applications (e.g. estimated execution time for all tasks that comprise the application) and the grid environment (e.g. current load and speed of all processors in the grid) in order to make scheduling decisions. This approach is clearly unsuitable to distributed environments like a grid in which there is no central administrative control and therefore, it is not possible to guarantee that this information will always be available or, when available, will be correct. Because of this, the second approach does not use any information. Instead, it applies the principle of task replication, achieving performance comparable to those heuristics that are based on information. On the other hand, its good performance is achieved at the expense of extra consumption of resources due to replication. In this work, we investigate these two scheduling approach and propose a new approach that takes advantage of each of them. The proposed approach consists of an adaptive solution to the grid environment according to information availability. This new approach provides a more efficient scheduling process since it reduces the grid resource wasting, without compromising the application performance.

Agradecimentos

A minha família pelo constante apoio e incentivo durante toda minha carreira acadêmica.

A Flávia pelo enorme apoio, paciência, companheirismo e amor dedicado, principalmente nos momentos em que mais precisei. Pela compreensão nos vários fins de semana em que precisei ficar em Campina Grande para me dedicar às atividades deste trabalho.

A Fubica, meu orientador, pela seriedade, confiança, paciência e pelos inúmeros ensinamentos durante todo o mestrado. A Walfredo pelas constantes idéias e discussões que forneceram importantes contribuições para este trabalho.

Ao pessoal do Failure-Spotter: Lívia, Andrey, Ana Cristina, Leo, Flávio, Bruno Catão, que sempre proporcionaram à nossa sala um agradável clima para se trabalhar. Agradeço ainda a Leo pela grande ajuda na preparação dos experimentos necessários para concretização deste trabalho. E pela responsabilidade e dedicação com que conduziu boa parte das atividades da nossa linha de pesquisa.

A Érica Gallindo e Matheus B2 pela ótimo suporte as minhas constantes dúvidas e problemas com o OurGrid. Ao mestre Elizeu pelas fundamentais discussões que resultaram em enormes contribuições para o sucesso desse trabalho.

Por fim, a todo o pessoal do LSD pelo ótimo ambiente de trabalho proporcionado pelas pessoas maravilhosas que fazem (ou fizeram) parte desse grande time.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Grids computacionais (FOSTER; KESSELMAN, 1999)(FOSTER; KESSELMAN, 2001) têm por objetivo prover uma infra-estrutura de hardware e software que permita acesso a uma grande quantidade de recursos computacionais geograficamente distribuídos, de forma confiável, consistente e econômica. No entanto, um grid computacional geralmente é formado por diversos sites, onde cada site representa um domínio administrativo diferente. Cada domínio administrativo possui recursos com características variadas, além de diferentes políticas de segurança e restrições de acesso a tais recursos. Tudo isso contribui para que o ambiente de grid seja bastante heterogêneo e dinâmico, tornando o seu gerenciamento uma atividade complexa e desafiadora. A Figura 1.1 ilustra a idéia de um grid computacional.

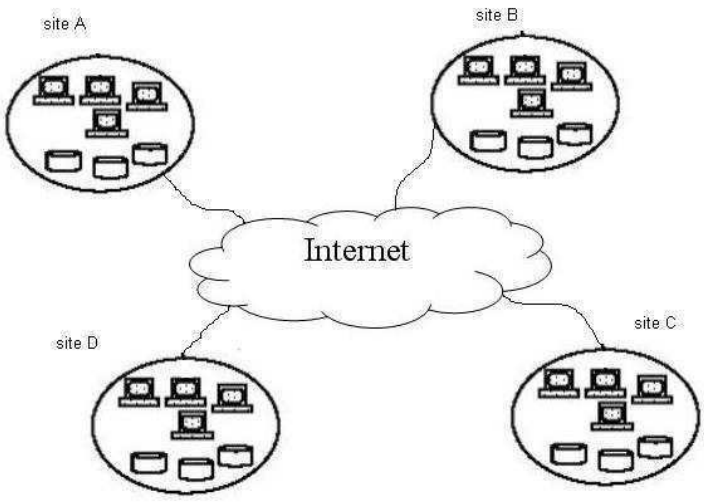


Figura 1.1: Visão Geral de um grid computacional

Um grid computacional pode ser visto como uma plataforma de execução de aplicações paralelas (CIRNE, 2002) que possibilita a alocação de uma grande quantidade de recursos a um custo muito mais baixo do que plataformas tradicionais tais como supercomputadores ou *clusters* dedicados. Aplicações paralelas são compostas por várias tarefas que podem executar em diferentes processadores. As tarefas de uma aplicação paralela podem ser dependentes ou independentes entre si. Neste trabalho, abordaremos aplicações do tipo Bag-of-Tasks (BoT) cujas tarefas são independentes, i.e., não se comunicam entre si. Apesar de sua simplicidade, aplicações BoT são bastante importantes e utilizadas em várias áreas, tais como mineração de dados, pesquisas massivas (como quebra de chave), varredura de parâmetros, simulações de Monte Carlo, manipulação de imagens (como tomografia (SMALLEN; CIRNE; FREY, 2000)), entre outras. Além disso, essas aplicações são bastante apropriadas para execução no grid devido ao fraco acoplamento entre suas tarefas.

Características inerentes ao ambiente de grid tais como heterogeneidade, larga distribuição e complexidade criam novos desafios técnicos a serem tratados. Um dos principais desafios é a atividade de escalonamento de aplicações que consiste em alocar tarefas de uma aplicação paralela a recursos do grid. O escalonamento geralmente é feito por um software denominado *broker* através do qual o usuário submete sua aplicação. Tal software é responsável por descobrir os recursos disponíveis no grid, selecionar os mais apropriados para a aplicação e submeter suas tarefas para execução. Um broker pode possuir diferentes escalonadores. Um escalonador utiliza-se de heurísticas para realizar as decisões de escalonamento. As decisões tomadas pelo escalonador geralmente são baseadas em um modelo de desempenho da aplicação e em dados sobre o estado atual dos vários recursos que formam o grid (CIRNE, 2002). O objetivo principal de um escalonador é minimizar o tempo total de execução de uma aplicação utilizando da melhor forma os recursos necessários.

Grande parte das heurísticas de escalonamento propostas na literatura requerem o uso de informações tanto sobre a aplicação (ex.: tempo estimado de execução, tamanho dos dados de entrada e saída, etc.) quanto sobre os recursos (ex.: velocidade do processador, carga de CPU, memória disponível, conectividade, etc.) para realizar as decisões de escalonamento (CASANOVA et al., 2000) (MAHESWARAN et al., 2000) (JAMES; HAWICK; CODDINGTON, 1999) (MENASCé et al., 1995) (IBARRA; KIM, 1977). Essas informações são necessárias para que o escalonador possa definir quais recursos serão utilizados e quais tarefas cada um desses

recursos executará. Essas heurísticas são conhecidas por *bin-packing schedulers*.

As informações sobre os recursos utilizadas pelos *bin-packing schedulers* podem ser, em geral, mais facilmente obtidas em plataformas homogêneas e acopladas como os clusters. Entretanto, em grids essas informações são difíceis de obter e nem sempre estão disponíveis. E mesmo quando disponível, não se pode garantir que a informação obtida seja correta e confiável devido à natureza dinâmica do grid.

Informações sobre as tarefas que compõem a aplicação também são difíceis de serem obtidas. É sabido que permitir que o próprio usuário forneça o tempo estimado de execução das tarefas, implica claramente que tais informações podem ser imprecisas (LEE et al., 2004). Assim, muitas vezes, a única maneira de obter essa estimativa é submetendo a tarefa para execução em um dado recurso. Uma possível estratégia seria executar uma tarefa experimental em todos os recursos e com base no tempo de execução dessa tarefa seria estimado o tempo de execução das demais tarefas da aplicação. No entanto, para que essas estimativas sejam precisas, é necessário saber qual a relação entre a complexidade computacional da tarefa experimental e a complexidade computacional das demais tarefas.

Nesse sentido, para contornar o problema da difícil obtenção de informações sobre a aplicação e o grid, foram propostas heurísticas de escalonamento que não dependem de nenhuma informação (PARANHOS; CIRNE; BRASILEIRO, 2003)(SANTOS-NETO et al., 2004). Por não utilizarem informações, essas heurísticas aplicam a técnica de replicação de tarefas para diminuir o impacto de eventuais mapeamentos ineficientes entre tarefas e máquinas. Por conta disso, são comumente denominadas de *replication schedulers*. No geral, tais heurísticas apresentam desempenho comparável às heurísticas baseadas em informação. Entretanto, esse bom desempenho é alcançado ao custo de um maior consumo de recursos devido à replicação de tarefas.

Nesse trabalho investigamos as duas abordagens de heurísticas de escalonamento e propomos e avaliamos uma nova abordagem adaptativa que aproveita as vantagens de cada uma delas. A grande vantagem da abordagem baseada em informação é prover um mapeamento mais eficiente entre tarefas e máquinas uma vez que são utilizadas informações do ambiente para tomar decisões. Porém sabemos que nem sempre toda a informação sobre o ambiente está disponível, o que impacta diretamente o desempenho de heurísticas que seguem essa abordagem. A segunda abordagem, por ser independente de informação, tem a vantagem de

apresentar um melhor desempenho em ambientes onde há indisponibilidade de informações, porém consome mais recursos. Dessa forma, a nova abordagem proposta têm como objetivo prover uma solução que seja adaptativa ao ambiente de grid quanto à disponibilidade de informações. Utilizando-se de qualquer informação disponível, a nova abordagem consegue lidar melhor com o dinamismo e heterogeneidade do ambiente de grid, apresentando um bom desempenho para aplicação e, ao mesmo tempo, diminuindo o desperdício de recursos.

O restante do trabalho está organizado da seguinte forma. No Capítulo 2 são explicitados os principais problemas em escalonamento em grids computacionais e discutidas as principais soluções existentes dentro do nosso contexto. No Capítulo 3 formalizamos o problema e definimos as notações utilizadas. No Capítulo 4 introduzimos o novo conjunto de heurísticas de escalonamento adaptativas à disponibilidade de informações. No Capítulo 5 são apresentados os resultados da avaliação de desempenho entre as heurísticas de escalonamento existentes e as novas soluções propostas, observando quais aspectos do grid e da aplicação influenciam no desempenho de cada heurística. No Capítulo 6 são apresentados as conclusões finais e os possíveis trabalhos futuros.

Capítulo 2

Escalonamento em Grids Computacionais

Em toda plataforma de execução de aplicações paralelas, um serviço de escalonamento eficiente é fundamental para que as aplicações possam alcançar de maneira satisfatória os benefícios diretamente proporcionados por tal infra-estrutura. Um escalonador é responsável por selecionar, dentre um conjunto de recursos disponíveis, os mais apropriados que possam prover um bom desempenho para as aplicações. Para isso, os escalonadores fazem uso de heurísticas de escalonamento.

Na prática, definir uma heurística eficiente para escalonar aplicações em grids não é uma tarefa trivial. Diversos problemas inerentes a plataformas largamente distribuídas como grids dificultam esse processo. Os principais problemas que podemos destacar são:

- **Heterogeneidade dos recursos**: Um grid computacional possui uma grande quantidade de recursos que são, geralmente, bastante heterogêneos. Os principais fatores responsáveis por essa heterogeneidade são: diferentes arquiteturas de computadores, velocidade dos processadores, interconexão entre eles, quantidade de memória, entre outros.

- **Múltiplos domínios administrativos**: Os recursos de um grid estão distribuídos por diversos domínios administrativos, cada qual com suas políticas de segurança e acesso a tais recursos. Isso dificulta a obtenção de informações sobre muitos dos recursos que pertencem a esses domínios.

- **Compartilhamento de recursos**: Tanto os processadores quanto os canais de comunicação de um grid são compartilhados por diversos outros usuários e aplicações. Com isso, a variação de carga e de disponibilidade dos recursos é muito grande, afetando diretamente o desempenho das aplicações que fazem uso de tais recursos.

Uma heurística de escalonamento tem por objetivo reduzir o tempo total de execução de uma aplicação. Para isso, ela tenta utilizar os recursos necessários de tal forma que seja proporcionado um maior paralelismo para a aplicação. Para entender melhor, ilustramos na Figura 2.1 a relação entre o momento em que uma ou mais máquinas ficam ociosas e não há mais tarefas a serem escalonadas (T_{1st_free}) e o momento em que aplicação é finalizada (T_{Final}). É possível observar que para se alcançar um maior paralelismo de uma determinada aplicação é necessário ter T_{1st_free} tão tarde quanto possível, isto é, reduzir a diferença entre T_{Final} e T_{1st_free}. Ao minimizar essa diferença, fazemos com que os recursos disponíveis sejam utilizados da melhor maneira possível durante a execução da aplicação de forma que o seu tempo total de execução seja reduzido. Em um cenário ótimo teríamos T_{Final} = T_{1st_free}, ou seja, todas as tarefas finalizando no mesmo instante e a utilização máxima do paralelismo proporcionado pelo grid.

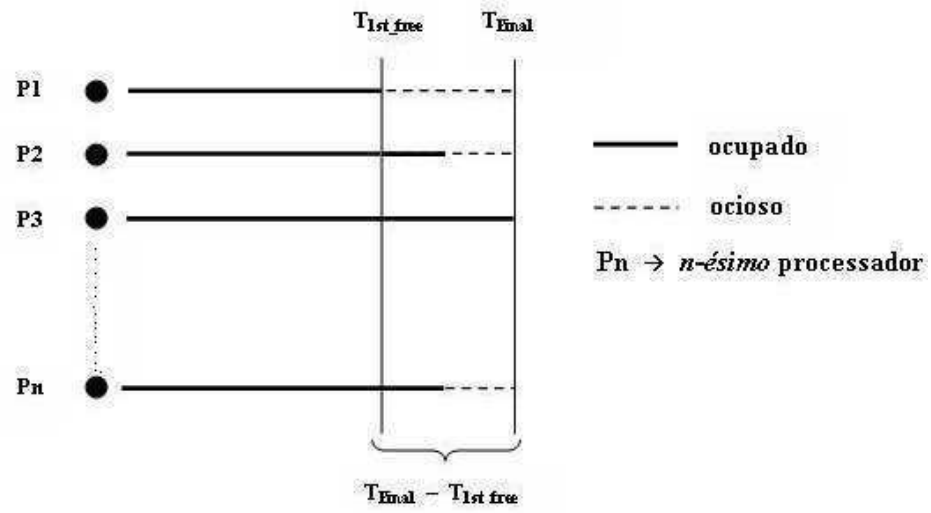


Figura 2.1: Relação entre T_{Final} e T_{1st_free} (ASSIS et al., 2006)

O grau de paralelismo de uma aplicação é influenciado pela forma como suas tarefas são mapeadas sobre os recursos do grid. Dessa forma, se um escalonador possui informação precisa sobre as tarefas e os recursos, ele consegue realizar mapeamentos eficientes de forma a minimizar a diferença entre T_{Final} e T_{1st_free} e, com isso, reduzir o tempo total de execução da aplicação. Essa estratégia é utilizada pelas heurísticas *bin-packing* conforme veremos mais adiante.

Durante o intervalo de tempo entre T_{1st_free} e T_{Final}, alguns recursos podem ficar ociosos. Nesse caso, se o escalonador efetuar mapeamentos ineficientes devido a falta de informação sobre o ambiente, ele pode utilizar esses recursos para executar réplicas de tarefas que ainda estão em execução, com o objetivo de reduzir o tempo total de execução da aplicação. Essa estratégia é utilizada pelas heurísticas de replicação, que não se baseiam em informação alguma, conforme veremos a seguir em mais detalhes.

Nas próximas seções, descrevemos as principais heurísticas que tratam dos desafios de escalonamento de aplicações BoT[1] em grids computacionais. Primeiramente, apresentamos algumas das principais heurísticas baseadas em informação do ambiente (heurísticas *bin-packing*). Em seguida, descrevemos a abordagem que não utiliza informação do ambiente para realizar as decisões de escalonamento (heurísticas de replicação). Por fim, fazemos uma breve análise comparativa entre as duas abordagens.

2.1 Heurísticas *bin-packing*

Na literatura são propostas diversas heurísticas de escalonamento que utilizam informações sobre o ambiente para escalonar tarefas independentes em ambientes heterogêneos como os grids computacionais (MAHESWARAN et al., 2000) (CASANOVA et al., 2000) (BRAUN et al., 2001) (WU; SUN, 2003) (GIERSCH; ROBERT; VIVIEN, 2004) (CASANOVA; HAYES; YANG, 2002) (JAMES; HAWICK; CODDINGTON, 1999). O bom desempenho de grande parte dessas heurísticas depende da disponibilidade e da qualidade da informação obtida. Tais heurísticas podem utilizar informação estática ou dinâmica sobre o ambiente. Informação estática — aquela que permanece inalterada por um período de tempo razoavelmente longo — geralmente pode ser

[1]Em particular, nesse trabalho abordaremos apenas aplicações BoT de computação intensiva (*cpu-intensive*).

obtida através de arquivos de configuração ou através de um sistema de informação de grid (GIS) como o Globus MDS (CZAJKOWSKI et al., 2001) ou o NodeWiz (BASU et al., 2005). Exemplos de informações estáticas são: tipo do processador, velocidade do processador, número de processadores, tamanho da memória, sistema operacional instalado, entre outros. Já informação dinâmica pode ser obtida através de sistemas de monitoração de desempenho como o NWS (WOLSKI; SPRING; HAYES, 1999). Informações dinâmicas são aquelas que variam constantemente ao longo do tempo devido a mudanças nas condições do ambiente. Elas incluem memória disponível, carga de CPU, disponibilidade de largura de banda, latência da rede, entre outros.

As heurísticas *bin-packing* comumente utilizam-se de uma métrica denominada *Completion Time* (CT) que indica a estimativa do tempo de execução de uma tarefa num determinado recurso (MAHESWARAN et al., 2000). O valor do CT é calculado com base nas informações estáticas e dinâmicas dos recursos do grid (ver Seção 3.3). Com isso, baseado numa matriz de estimativas que contém o CT de cada tarefa em cada recurso do grid, essas heurísticas analisam qual o melhor mapeamento entre tarefas e recursos de forma que seja proporcionada uma redução do tempo total de execução da aplicação. A seguir descreveremos duas das principais heurísticas *bin-packing*: *Dynamic Fastest Processor to Largest Task First* (*Dynamic FPLTF*) e *Sufferage*. O *FPLTF* é um algoritmo estático que apresenta bom desempenho em ambientes dedicados (MENASCé et al., 1995). O *Dynamic FPLTF* é o resultado de uma modificação feita por (PARANHOS; CIRNE; BRASILEIRO, 2003) para torná-lo dinâmico e conseqüentemente utilizável em ambientes Grid. O *Sufferage* é uma algoritmo dinâmico que tem mostrado atingir bom desempenho em ambientes heterogêneos e dinâmicos quando considerado a existência de informação completa (AHRNAD, 1995).

Dynamic FPLTF

O Dynamic FPLTF utiliza três tipos de informação para escalonar as tarefas eficientemente: *Task Size*, *Host Load* e *Host Speed*. *Host Speed* representa a velocidade relativa da máquina. Uma máquina que tem *Host Speed = 2* executa uma tarefa duas vezes mais rápido que uma máquina com *Host Speed = 1*. *Host Load* representa a fração de CPU da máquina que não está disponível para a aplicação em um determinado instante (fração de CPU que está sendo usada por outros usuários e aplicações). Note que *Host Load* varia ao longo do tempo,

dependendo da carga que é imposta à máquina por outros usuários e aplicações. Por fim, *Task Size* é o tempo necessário para uma máquina com *Host Speed = 1* completar a tarefa quando *Host Load = 0*.

No início do algoritmo, o tempo para se tornar disponível *TBA (Time to Become Available)* de cada host é iniciado com 0 e as tarefas ordenadas por tamanho em ordem decrescente. Desse modo, a maior tarefa é a primeira a ser alocada. Cada tarefa é alocada para o host que provê o menor tempo de execução CT (*Completion Time*) para ela. O CT de uma tarefa é definido por:

$$CT = TBA + Task\,Cost$$

onde

$$Task\,Cost = \frac{\frac{Task\ Size}{Host\ Speed}}{1 - Host\ Load}$$

Quando uma tarefa é alocada para uma máquina, o valor do *TBA* correspondente a esse host é incrementado com *Task Cost*. As tarefas são alocadas até que todas as máquinas do grid estejam em uso. Após isso, a execução da aplicação é iniciada. Quando uma tarefa é completada, todas as tarefas que não estão rodando no momento são desescalonadas e re-escalonadas até que cada máquina do grid tenha pelo menos uma tarefa alocada. Esse esquema continua até que todas as tarefas sejam completadas.

Essa estratégia tenta minimizar os efeitos da dinamicidade do Grid uma vez que a carga das máquinas varia ao longo do tempo. Ou seja, uma máquina que, em um primeiro momento, é muito rápida e pouco carregada recebe mais tarefas que uma máquina também muito rápida, mas com bastante carga. Entretanto, em um segundo momento, a máquina com pouca carga, pode passar a ter muita carga e vice-versa. Assim, o processo de re-escalonamento de tarefas tenta corrigir esse problema alocando as tarefas prioritariamente para as máquinas de fato mais rápidas em um determinado instante.

Sufferage

Como o próprio nome reflete, Sufferage (CASANOVA et al., 2000) (MAHESWARAN et al., 2000) é uma heurística *bin-packing* baseada na métrica denominada *sufferage*, que é definida como sendo a diferença entre o menor CT e o segundo menor CT de uma tarefa (calculado da mesma forma como em FPLTF). A idéia é alocar para uma determinada máquina uma tarefa

que "sofreria" mais caso não fosse alocada para aquela máquina, ou seja, a tarefa que possui o maior valor para o *sufferage*.

No início do algoritmo quando todas as máquinas estão livres (sem tarefas alocadas), o valor de *sufferage* é calculado para cada tarefa do conjunto de tarefas a serem escalonadas. Primeiramente, a tarefa que possuir o maior valor de *sufferage* é alocada para a máquina que lhe proporciona uma menor estimativa do CT e, após isso, é removida do conjunto. Em seguida, o procedimento é repetido até que todas as tarefas sejam alocadas. Note que se uma máquina escolhida por uma dada tarefa não estiver livre, os valores de *sufferage* da tarefa previamente alocada a essa máquina e da nova tarefa a ser alocada são comparados. A tarefa que ficará nessa máquina será a que possuir o maior valor de *sufferage*, enquanto a outra é retornada para o conjunto de tarefas a serem escalonadas.

Naturalmente, os valores do *sufferage* de cada tarefa variam durante a execução da aplicação devido à variação de carga dos recursos do grid. Da mesma forma que é efetuado para *Dynamic FPLTF*, o algoritmo de *Sufferage* é invocado várias vezes durante a execução da aplicação. Toda vez que uma tarefa é completada, todas as tarefas que ainda não começaram a rodar são desescalonadas e o algoritmo é invocado novamente, usando os valores do *sufferage* atuais. Conseqüentemente, o algoritmo executa novamente re-escalonando as tarefas restantes, mas dessa vez com a nova carga das máquinas. Esse esquema é repetido até que todas as tarefas sejam completadas.

2.2 Heurísticas de replicação

Diferentemente das heurísticas *bin-packing*, essas heurísticas não utilizam informação do ambiente para efetuar o mapeamento entre tarefas e máquinas. Assim, para minimizar o impacto negativo no desempenho da aplicação devido à não utilização de tais informações, elas fazem uso da técnica de replicação de tarefas, que consiste em disparar múltiplas réplicas de uma mesma tarefa, durante a execução da aplicação, a partir do momento em que todas as tarefas já foram escalonadas e existirem recursos ociosos. O objetivo dessa estratégia é permitir que uma tarefa que tenha sido alocada a uma máquina muito lenta ou sobrecarregada possa ser alocada a outra máquina em melhores condições e, com isso, reduzir o seu tempo de execução. Conseqüentemente, o tempo total de execução da aplicação também é reduzido.

Na literatura, diversas heurísticas de replicação têm sido propostas e estudadas (BARATLOO et al., 1996) (PARANHOS; CIRNE; BRASILEIRO, 2003) (SANTOS-NETO et al., 2004) (CIRNE et al., 2005) (FUJIMOTO; HAGIHARA, 2003) (GHARE; LEUTENEGGER, 2004). Nesse trabalho, utilizamos *Workqueue with Replication* (*WQR*) (PARANHOS; CIRNE; BRASILEIRO, 2003) como heurística representativa para efeitos de comparação, cujo funcionamento é descrito a seguir.

WQR

A heurística *WQR* (PARANHOS; CIRNE; BRASILEIRO, 2003) basicamente acrescenta a técnica de replicação de tarefas ao clássico algoritmo *WorkQueue* (*WQ*). O *WQ* seleciona tarefas de forma aleatória e essas são alocadas a máquinas que estejam disponíveis no momento. Enquanto houver tarefas a escalonar, as execuções do *WQ* e *WQR* são idênticas. O *WQR* diferencia-se do *WQ* a partir do momento que todas as tarefas existentes já foram alocadas. A partir desse ponto, no *WorkQueue*, as máquinas que forem finalizando suas tarefas, ficam ociosas durante todo o restante da execução da aplicação. Enquanto que no *WQR*, a partir desse ponto é iniciado o processo de replicação de tarefas. À medida que máquinas vão sendo liberadas, réplicas de tarefas que ainda estejam executando são escolhidas de forma aleatória e alocadas para elas. Quando uma tarefa é replicada, a chance de que pelo menos uma réplica seja alocada a uma máquina mais rápida aumenta. Quando uma das réplicas de uma tarefa termina, as demais são abortadas.

No mais, *WQR* ainda impõe um limite para o nível de replicação, ou seja, uma tarefa é replicada apenas se um limite predefinido de réplicas não for alcançado (grau de replicação). Por exemplo, se o limite de replicação for 3, cada tarefa só poderá ter no máximo 3 réplicas ativas. Assim, uma aplicação com 10 tarefas, terá um limite máximo de 30 réplicas ativas. Vale ressaltar ainda que a *n-ésima* réplica de uma determinada tarefa só é criada quando todas as tarefas da aplicação ainda em execução tiverem pelo menos *n-1* réplicas criadas. O objetivo dessa estratégia consiste em ter um melhor controle do nível de desperdício de recursos.

2.3 Análise Comparativa

O trabalho apresentado em (PARANHOS; CIRNE; BRASILEIRO, 2003) faz uma avaliação de desempenho entre heurísticas *bin-packing* e de replicação. Em particular, são analisadas as heurísticas *Dynamic FPLTF*, *Sufferage* e *WQR*. Na Tabela 2.1, temos um sumário dos resultados obtidos nas simulações executadas naquele trabalho.

Heurística	**Tempo Total de Execução**	**Desperdício**
Dynamic FPLTF	12901 s	-
Sufferage	13530 s	-
WQR	12123 s	48%

Tabela 2.1: Média do desempenho das heurísticas *Dynamic FPLTF*, *Sufferage* e *WQR* considerando informação perfeita (PARANHOS; CIRNE; BRASILEIRO, 2003)

A partir desses resultados, podemos observar que o desempenho das duas abordagens quanto ao tempo total de execução da aplicação são comparáveis. No entanto, é possível destacar que a heurística de replicação *WQR* apresenta um grande nível de desperdício de recursos, ou seja, muitos recursos são utilizados para processar tarefas que posteriormente são abortadas. Outro ponto a ser destacado é que nos cenários simulados, as heurísticas *bin-packing* (*Sufferage* e *Dynamic FPLTF*) utilizam informações perfeitas (sempre disponíveis e correta), situação que é praticamente impossível de se obter em um ambiente real de grid.

As heurísticas *bin-packing* conseguem realizar um escalonamento eficiente de aplicações em grids através da utilização de informações do ambiente. Tais informações auxiliam na tomada de decisões por parte dos escalonadores para tentar definir o melhor mapeamento entre tarefas e recursos de forma a minimizar o tempo total de execução da aplicação e utilizar os recursos necessários eficientemente. Porém, sabemos que essas informações são difíceis de obter com precisão e freqüentemente não estão disponíveis devido aos inúmeros problemas inerentes à plataforma de Grid.

Grandes esforços têm sido realizados para tentar monitorar os recursos de um grid e obter as informações necessárias para o escalonamento (BASU et al., 2005) (FRANCIS et al., 1999) (LEE et al., 2005) (WOLSKI, 1998) (WOLSKI, 2003). No entanto, monitorar sistemas largamente distribuídos não é uma tarefa fácil uma vez que nem sempre é possível instalar

sensores em todos os recursos devido a restrições de acesso a determinados domínios. E mesmo assim, nos recursos onde é possível fazer o monitoramento, a informação colhida nem sempre é correta e atualizada devido ao dinamismo do ambiente (FRANCIS et al., 1999). Quando a informação é incompleta ou incorreta, o desempenho das heurísticas acaba sendo prejudicado (CASANOVA et al., 2000).

Nesse sentido, as heurísticas de replicação, que não utilizam nenhuma informação para efetuar o escalonamento, têm surgido como uma alternativa para lidar com a falta de informação. Elas conseguem atingir desempenho comparáveis com as heurísticas *bin-packing* conforme vemos na Tabela 2.1. Para isso, no entanto, elas consomem uma quantidade extra de recursos devido à técnica de replicação de tarefas, ocasionando, assim, grande desperdício de recursos.

Dentro desse contexto, temos de um lado uma abordagem de escalonamento que assume que todas as informações necessárias estão disponíveis e corretas. E do outro lado, temos uma abordagem que não faz uso de informação alguma mesmo nos casos onde há informação disponível. A primeira não funciona bem quando falta informação, enquanto que a segunda apresenta um grande desperdício de recursos. Nesse trabalho propomos uma abordagem mais flexível que seja adaptativa ao ambiente de grid com relação à disponibilidade de informações. Ao invés de utilizar informação apenas quando esta estiver totalmente disponível conforme a primeira abordagem ou abdicar do uso de qualquer informação conforme a segunda, o funcionamento dessa nova abordagem consiste em fazer uso de qualquer informação disponível. Devido à grande heterogeneidade e dinamismo do ambiente de grid, onde informações podem ou não estar disponíveis, uma solução ajustável aparenta ser mais adequada, proporcionando maior eficiência na execução de aplicações BoT com uma sensível redução do nível de desperdício de recursos, sem comprometer o desempenho da aplicação.

2.4 Considerações finais

Este capítulo descreveu em detalhes a motivação do trabalho. Inicialmente foram discutidos os principais desafios no escalonamento de aplicações BoT em grids computacionais. Dentre os diversos problemas inerentes a plataformas de grids foram destacados a heterogeneidade dos recursos, a larga distribuição por múltiplos domínios administrativos e a varia-

ção de carga devido ao compartilhamento desses recursos. Em seguida, foram apresentadas duas das principais soluções existentes: heurísticas *bin-packing* (baseadas em informação) e heurísticas de replicação (independentes de informação). Foram descritos o objetivo e as principais características de cada uma dessas abordagens de heurísticas de escalonamento e, por fim, realizada uma breve análise comparativa entre elas.

Capítulo 3

Modelo do Escalonamento de Aplicações em Grids Computacionais

Neste capítulo apresentamos o modelo do sistema adotado nas simulações e experimentos. O objetivo principal do modelo é refletir o escalonamento de aplicações BoT (jobs) em um grid computacional. O modelo está dividido em três partes: *modelo da plataforma de grid*, *modelo da aplicação* e *modelo da informação*. A seguir descrevemos cada uma delas.

3.1 Modelo da Plataforma de Grid

Um grid G pode ser definido como um conjunto de sites. Os sites representam domínios administrativos diferentes geograficamente distribuídos. Cada site pode possuir um conjunto de processadores de grid (P), um conjunto de servidores de dados (S) e um conjunto de processadores base (B). Os processadores de grid são responsáveis pela execução das tarefas. Os servidores de dados armazenam os dados de entrada necessários para a execução das tarefas, assim como, os dados de saída gerados por elas. Por fim, os processadores base executam o escalonador através do qual os usuários submetem suas aplicações para serem executadas no grid. Formalmente, temos:

$$G = \{site_1, site_2, ..., site_g\}, g > 0;$$
$$site_i = P_i \cup S_i \cup B_i;$$
$$S_i \neq \emptyset \wedge P_i \cup B_i \neq \emptyset \, ,$$

onde P_i é o conjunto de processadores de grid que fazem parte do $site_i$, B_i é o conjunto de processadores base do $site_i$ e S_i é o conjunto de servidores de dados do $site_i$.

Para simplificar o modelo, consideramos que cada *site* possui apenas um único servidor de dados denominado σ_i. Consideramos também que cada site possui no máximo um processador base que, quando existe, é denominado β_i . Dessa forma temos:

$$S_i = \{\sigma_i\} \quad e \quad B_i = \{\beta_i\} \vee B_i = \emptyset \, .$$

Definimos ainda um conjunto que representa todos os processadores de grid (P_G) de um grid G. Formalmente, temos:

$$P_G = \bigcup_{i=1}^{|G|} P_i$$

Cada processador de grid é definido como uma tupla formada por dois elementos: velocidade (cpu_speed) e carga de processamento disponível (cpu_avail). Formalmente, temos:

$$\pi = (cpu\_speed, cpu\_avail), \forall \pi \in P_G \, ,$$

onde $\pi.cpu\_speed$ e $\pi.cpu\_avail$ representam a velocidade e a carga de processamento disponível no processador π, respectivamente. Vale ressaltar que a carga de processamento disponível para cada processador varia ao longo da execução de uma aplicação, ou seja, é uma informação dinâmica (ver Seção 3.3).

Outro aspecto importante na modelagem da plataforma de grid é a rede de comunicação, principalmente na modelagem de aplicações *data-intensive*. Para tal, adotamos um modelo baseado no modelo de rede desenvolvido por (CASANOVA; MARCHAL, 2002) (CASANOVA, 2005) e definido em (NÓBREGA-JÚNIOR et al., 2005). Consideramos uma topologia na qual todos os *sites* estão interconectados entre si através de uma topologia totalmente conectada, ou seja, cada site possui uma conexão direta (*link*) para cada um dos demais sites. Nesse modelo, um link possui três parâmetros: largura de banda por conexão (bw_con), latência

(lat) e limite de largura de banda (bw_lim). A latência representa o intervalo de tempo entre a requisição de um dado e a chegada deste, ou seja, representa o atraso associado ao link. O parâmetro bw_con indica quanto de largura de banda pode ser conseguido por uma nova conexão aberta caso o link estivesse dedicado (não compartilhado com outras conexões), enquanto que o bw_lim indica o limite de largura de banda que pode ser compartilhado entre as várias conexões simultâneas num determinado link. Formalmente, temos que a rede do grid (N_G) pode ser representada como um conjunto de *links* (λ) da seguinte forma:

$$N_G = \{\lambda_{i,j} \mid 1 \leq i, j \leq g\}, g = |G|;$$
$$\lambda = (lat, bw\_con, bw\_lim), \forall \lambda \in N_G \ ,$$

onde $\lambda.lat$, $\lambda.bw\_con$ e $\lambda.bw\_lim$ representam a latência, largura de banda por conexão e limite de largura de banda do link λ, respectivamente. $\lambda_{i,j}$, com $i \neq j$, representa um link entre o $site_i$ e o $site_j$ (*inter-site*), enquanto que $\lambda_{i,i}$ representa o *link* que conecta os recursos do $site_i$ (*intra-site*).

Outra informação importante sobre a rede é a largura de banda disponível (bw_avail) que consiste no valor de largura de banda que uma nova conexão pode, de fato, conseguir num dado link λ dada a quantidade atual de conexões abertas naquele *link*. Dessa forma, bw_avail é resultado de uma função que recebe como entrada bw_con e bw_lim. Formalmente:

$$\lambda.bw\_avail = f(\lambda.bw\_con, \lambda.bw\_lim) \ ,$$

onde $\lambda.bw\_avail$ é a largura de banda disponível do link λ. A função f verifica se a soma das larguras de banda de todas as conexões ativas no link λ é inferior ao limite de largura de banda. Se for inferior, o valor recebido por bw_avail para a nova conexão é o mesmo de bw_con. Caso contrário, bw_avail recebe como valor uma fração do valor de bw_lim de acordo com o número de conexões ativas no momento, ou seja, bw_lim é dividido proporcionalmente entre as conexões (NÓBREGA-JÚNIOR et al., 2005). O parâmetro bw_lim tem por objetivo evitar situações irreais onde infinitas conexões possam conseguir a mesma fração de toda a banda disponível em um *link* de *backbone* (*inter-site*). Assim, esse parâmetro é fundamental para modelar o comportamento dessa categoria de *link*. Enquanto o limite estabelecido por bw_lim não for atingido, conexões simultâneas que estejam alocadas a esse *link* não interferem uma nas outras. A partir do momento que se atinge tal limite, elas passam a causar interferência entre si tal como ocorre em *links* locais (*intra-site*).

Note que nosso modelo de rede é bastante flexível uma vez que define um único tipo de *link* através do qual é possível modelar tanto um *link inter-site* como um *link intra-site*. Para isso basta configurar os seus três parâmetros corretamente. Por exemplo, para modelar o comportamento de um *link intra-site*, devemos configurar os parâmetros bw_con e bw_lim com os mesmo valores. Na Figura 3.1 apresentamos uma ilustração do modelo da plataforma de grid. Ela ilustra um cenário com quatro *sites* totalmente conectados.

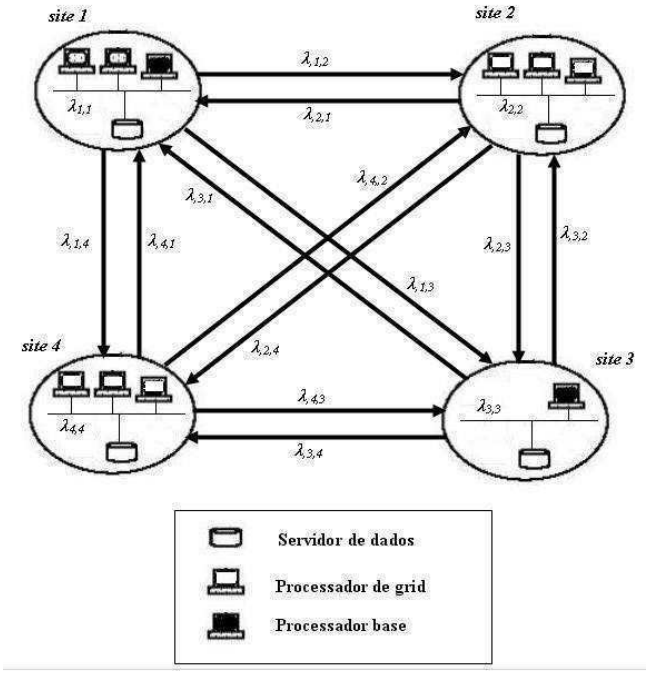


Figura 3.1: Modelo da Plataforma de Grid

3.2 Modelo da Aplicação

Nesse modelo, definimos uma aplicação (job) J como sendo um conjunto de n tarefas independentes. Cada tarefa (τ) é definida por um conjunto de dados de entrada ($input\_set$), um conjunto de dados de saída ($output\_set$) e um custo computacional ($computational\_cost$) correspondente ao seu tempo estimado de execução em um *processador-referência* em regime dedicado, ou seja, um processador com $\pi.cpu\_speed = 1$ e $\pi.cpu\_avail = 100\%$ durante toda a execução da tarefa. Formalmente, temos:

$$J = \{\tau_1, \tau_2, ..., \tau_n\}, onde\ |J| = n\ ;$$

$$\tau_k = (input\_set, output\_set, computational\_cost), 1 \leq k \leq n \ ,$$

onde os conjuntos de entrada e saída da tarefa τ_k podem ser representados por $\tau_k.input\_set$ e $\tau_k.output\_set$, respectivamente. De forma similar, $\tau_k.computational\_cost$ representa o custo computacional da tarefa τ_k.

Para o problema de escalonamento, definimos ainda quatro medidas de desempenho para uma tarefa: tempo estimado de processamento (PT), tempo estimado de transferência dos dados de entrada (TTI) e de saída (TTO) e tempo total de execução estimado (CT). Para aplicações que processam grandes quantidades de dados, temos que:

- $PT_{k,\pi}$ indica o tempo estimado para que a tarefa k seja executada pelo processador π. Formalmente:

$$PT_{k,\pi} = \frac{\tau_k.computational\_cost}{\pi.cpu\_speed \times \pi.cpu\_avail}$$

- $TTI_{k,i,j}$ indica o tempo estimado para que os dados de entrada da tarefa k sejam transferidos do processador base (β_i) do $site_i$ para o processador π_j. De forma similar, temos que $TTO_{k,i,j}$ indica o tempo estimado para que os dados de saída da tarefa k sejam transferidos do processador π_j no $site_j$ para o processador base (β_i) do $site_i$. Formalmente:

$$TTI_{k,i,j} = \lambda_{i,i}.lat + \lambda_{i,j}.lat + \lambda_{j,j}.lat + \frac{|\tau_k.input\_set|}{\min(\lambda_{i,i}.bw\_avail, \lambda_{i,j}.bw\_avail, \lambda_{j,j}.bw\_avail)}$$

e

$$TTO_{k,i,j} = \lambda_{i,i}.lat + \lambda_{i,j}.lat + \lambda_{j,j}.lat + \frac{|\tau_k.output\_set|}{\min(\lambda_{i,i}.bw\_avail, \lambda_{i,j}.bw\_avail, \lambda_{j,j}.bw\_avail)}$$

- $CT_{k,\pi}$ indica o tempo estimado para completar a execução de uma tarefa k em um processador π no $site_j$, considerando que a tarefa foi submetida a partir do processador base (β_i) do $site_i$. Formalmente:

$$CT_{k,\pi} = TTI_{k,i,j} + PT_{k,\pi} + TTO_{k,i,j}$$

As tarefas de aplicações de computação intensiva (*cpu-intensive*) não possuem dados ou estes são muito pequenos ao ponto de não influenciar no tempo total de execução de uma tarefa. Nesse caso, os valores de TTI e TTO são considerados desprezíveis e, assim, podemos considerar para esse tipo de aplicação que:

$$CT_{k,\pi} = PT_{k,\pi} = \frac{\tau_k.computational\_cost}{\pi.cpu\_speed \times \pi.cpu\_avail}$$

3.3 Modelo da Informação

As informações sobre o grid podem ser de natureza dinâmica ou estática. Informação dinâmica é aquela que muda ao longo da execução de uma aplicação (ex.: carga de processamento disponível em um processador). Elas são expressas através de uma função no tempo. Por exemplo, $\pi.cpu\_avail(t)$ representa a carga de processamento disponível no processador π no instante t. Informação estática é aquela que permanece inalterada durante a execução de uma aplicação (ex.: velocidade do processador). Tais informações podem ser obtidas de diversas formas. No nosso contexto, consideraremos que tais informações podem ser descritas pelo usuário em um arquivo de configuração (ex: arquivos JDF e SDF do *middleware* OurGrid (ANDRADE et al., 2003) (CIRNE et al., 2006)), obtidas através de ferramentas de monitoração (ex: NWS (WOLSKI; SPRING; HAYES, 1999), Iperf (TIRUMALA et al., 2004)) ou estimadas usando ferramentas de previsão de carga como o NWS.

Podemos classificar as informações em três conjuntos de acordo com a entidade à qual elas estão associadas: *informações sobre a rede*, *informações sobre os recursos computacionais* e *informações sobre a aplicação*.

As informações sobre a rede englobam informações dinâmicas relativas aos *links* que interligam os diversos recursos do grid, ou seja, latência, largura de banda por conexão e limite de largura de banda. Essas informações podem ser obtidas de medições usando ferramentas de monitoramento como NWS ou Iperf.

As informações sobre os recursos são aquelas relativas aos processadores que compõem o grid computacional. No nosso trabalho consideramos importante o uso de duas informações: velocidade do processador (estática) e carga de processamento disponível (dinâmica). A primeira pode ser obtida de arquivos de configuração enquanto que a segunda pode ser obtida através de medições ou estimativas usando a ferramenta NWS.

As informações sobre a aplicação englobam informações sobre suas tarefas. Para o problema de escalonamento, consideramos as seguintes informações estáticas: os arquivos de entrada e saída e as taxas de processamento dos dados de entrada e dos dados de saída.

Formalmente, os três conjuntos de informações podem ser representados da seguinte

forma:

$$\Im_{app} = \{input\_set, output\_set, input\_tpt, output\_tpt, computational\_cost\}$$
$$\Im_{proc} = \{cpu\_speed, cpu\_avail\}$$
$$\Im_{net} = \{lat, bw\_con, bw\_lim\} \ ,$$

onde $\Im_{app}$ representa o conjunto de informações sobre a aplicação, $\Im_{proc}$ representa as informações sobre os recursos computacionais e $\Im_{net}$ representa o conjunto de informações sobre a rede.

Algumas heurísticas de escalonamento utilizam as informações descritas anteriormente para realizar decisões de escalonamento (CASANOVA et al., 2000)(MAHESWARAN et al., 2000). Entretanto, em plataformas distribuídas, essas informações nem sempre estão disponíveis, e quando estão, nem sempre são precisas. Dado que o enfoque deste trabalho é avaliar heurísticas de escalonamento adaptativas às condições do grid quanto à indisponibilidade ou imprecisão destas informações, definimos uma determinada informação (i) como sendo uma tupla formada por dois elementos: disponibilidade (δ) e erro (ϵ). O primeiro indica se a informação está disponível ou não, enquanto que o segundo indica o nível de erro (percentual) associado a tal informação. Formalmente, temos:

$$i = (\delta, \epsilon), \forall i \in \Im$$
$$\delta \in \{true, false\}$$
$$\Im = \Im_{app} \cup \Im_{proc} \cup \Im_{net} \ ,$$

onde $i.\delta$ e $i.\epsilon$ indicam a disponibilidade e o percentual de erro associados à informação i, respectivamente. A informação i está disponível quando $i.\delta = true$, e indisponível quando contrário, $i.\delta = false$.

3.4 Escalonamento

O escalonamento de uma aplicação (*job*) é resultado do escalonamento de todas as suas tarefas. Considerando que uma tarefa pode possuir uma ou mais réplicas, o escalonamento de uma tarefa consiste em associar cada réplica a um processador que irá executá-la. Dessa

forma, num dado instante, réplicas de uma mesma tarefa podem ser associadas a processadores distintos. Consideramos ainda que pelo menos uma réplica de cada tarefa termina sua execução, o que acarreta no término da execução de tal tarefa. Logo, o escalonamento (Σ) de um *job* J é definido como um conjunto de réplicas executadas. Cada réplica é definida por uma tupla que associa uma tarefa a um processador e indica o percentual de computação cp realizado ($0 \leq cp \leq 1$). Formalmente, temos:

$$\Sigma_J = \{\mu_1, \mu_2, ..., \mu_m\}, m \geq |J|$$
$$\mu_i = \{\pi, \tau, cp\}, \forall \mu_i \in \Sigma_J$$
$$\forall \tau_k \in J, \exists\ \mu_i.\tau = \tau_k \wedge \mu_i.cp = 1\ ,$$

onde a réplica μ_i representa a associação entre uma tarefa τ e um processador π e $\mu_i.cp$ o percentual da tarefa que foi processado pelo processador $\mu_i.\pi$. Quando $\mu_i.cp = 1$, a execução da tarefa $\mu_i.\tau$ foi totalmente computada pelo processador $\mu_i.\pi$. Vale ressaltar que quando uma réplica de uma determinada tarefa τ_k é concluída, as demais réplicas são abortadas, o que ocasiona um desperdício dos recursos utilizados para execução parcial de tais réplicas.

Métricas de Desempenho

As heurísticas analisadas nesse trabalho têm como objetivo a redução do *tempo total de execução da aplicação* e do *nível de desperdício de recursos*. Com isso, a avaliação da eficiência de cada heurística será baseada na verificação dessas duas métricas.

O tempo total de execução da aplicação, normalmente referenciado como *makespan* (PINEDO, 2001) da aplicação, é definido como sendo o intervalo de tempo entre o momento em que a primeira tarefa do job é iniciada e o primeiro instante em que todas as tarefas finalizaram suas execuções. Formalmente, o *makespan* de uma aplicação J qualquer é definido por:

$$makespan_J = T_f - T_i\ ,$$

onde T_i indica o menor instante de tempo no qual há uma réplica de uma tarefa da aplicação J executando e T_f indica o maior instante de tempo no qual há uma réplica de uma tarefa da aplicação J em execução.

Considerando $badput$ como sendo a soma dos tempos gastos por todas as réplicas abortadas e $goodput$ a soma dos tempos gastos por todas as réplicas finalizadas, definimos o *nível*

de desperdício de recursos como sendo a relação entre $badput$ e $goodput$. Formalmente, temos que o nível de desperdício de um escalonamento Σ_J qualquer é definido por:

$$ResourceWasting_{\Sigma_J} = \frac{badput_{\Sigma_J}}{goodput_{\Sigma_J}}$$

Considerando $CR_{\Sigma_J}^{\tau_k}$ como sendo o conjunto não vazio composto por todas as réplicas μ de Σ_J referentes à tarefa τ_k do job J que completaram suas execuções, temos que:

$$CR_{\Sigma_J}^{\tau_k} = \{\mu \mid \mu \in \Sigma_J \wedge \mu.\tau = \tau_k \wedge \mu.cp = 1\}$$

Embora possam haver casos em que uma ou mais réplicas de uma mesma tarefa finalizem no mesmo instante de tempo, a cardinalidade do conjunto $CR_{\Sigma_J}^{\tau_k}$ normalmente é 1. Nesse sentido, consideramos ainda $\mu_{\Sigma_J}^{\tau_k}$ como sendo a única réplica da tarefa τ_k em um escalonamento Σ_J que finalizou sua execução. Sendo assim, temos que:

$$\mu_{\Sigma_J}^{\tau_k} = (\mu \in CR_{\Sigma_J}^{\tau_k}) \wedge (CT_{\mu.\tau,\mu.\pi} < CT_{\mu'.\tau,\mu'.\pi}, \forall\, \mu' \in CR_{\Sigma_J}^{\tau_k} \wedge \mu' \neq \mu)$$

Dessa forma, temos que $badput$ e $goodput$ são definidos formalmente por:

$$goodput_{\Sigma_J} = \sum_{k=1}^{|J|} CT_{\mu_{\Sigma_J}^{\tau_k}.\tau\ ,\ \mu_{\Sigma_J}^{\tau_k}.\pi}\ ,\ \forall\, \mu \in \Sigma_J$$

$$badput_{\Sigma_J} = \sum_{k=1}^{|J|} CT_{\mu_{\Sigma_J}^{\tau_k}.\tau\ ,\ \mu_{\Sigma_J}^{\tau_k}.\pi} - goodput_{\Sigma_J}\ ,\ \forall\, \mu \in \Sigma_J$$

3.5 Considerações finais

Neste capítulo foi descrito o modelo do sistema adotado nas simulações e experimentos realizados nesse trabalho. O modelo reflete o escalonamento de aplicações BoT (jobs) em um grid computacional. Ele foi dividido em três partes: *modelo da plataforma de grid*, *modelo da aplicação* e *modelo da informação*. Vale ressaltar que embora o foco desse trabalho sejam as aplicações BoT de computação intensiva (*cpu-intensive*), o modelo apresentado nesse capítulo também permite que sejam modeladas aplicações BoT que processam grandes quantidades de dados (*data-intensive*).

Capítulo 4

Heurísticas Adaptativas à Disponibilidade de Informação

Neste capítulo descrevemos uma nova abordagem de escalonamento de aplicações em grids computacionais que provê uma solução adaptável ao ambiente de grid quanto à disponibilidade de informações. Com isso, propomos e definimos duas heurísticas capazes de lidar com diferentes situações que vão desde ambientes sem informação alguma até ambientes com informação total. A primeira segue uma estratégia inspirada nas heurísticas *bin-packing*, ou seja, consiste em uma tentativa de adaptar as heurísticas *bin-packing* para lidar com diferentes situações de disponibilidade de informações. A segunda baseia-se nas heurísticas de replicação na tentativa de minimizar o desperdício de recursos através do uso da informação que esteja disponível. Nas seções a seguir, descrevemos cada um delas.

4.1 Adaptando Heurísticas *bin-packing*

Para realizar um mapeamento eficiente entre os recursos do grid e as tarefas de uma aplicação, as heurísticas *bin-packing* necessitam de toda a informação sobre o ambiente de grid. Elas utilizam as informações sobre os recursos para medir o desempenho dos recursos disponíveis e selecionar aqueles que possam executar a aplicação eficientemente. Desse modo, sabemos que essas heurísticas não funcionam corretamente em ambientes onde há indisponibilidade total ou parcial dessas informações, pois torna-se inviável efetuar uma avaliação de desempenho eficiente e completa para todos os recursos. Os recursos que não possuem

informação não são considerados durante o processo de escalonamento dessas heurísticas.

Nesta seção apresentamos uma nova heurística de escalonamento que consegue lidar com a falta de informações sobre os recursos do grid. Sendo assim, os recursos sem informação disponível também são considerados nas decisões de escalonamento. Essa nova heurística, denominada *Information Adaptive*, baseia-se no algoritmo FPLTF (MENASCé et al., 1995), ou seja, ela tenta escalonar primeiro as maiores tarefas sobre os recursos que apresentam melhor desempenho. Entretanto, o fato de não termos informações sobre todo o ambiente impossibilita uma classificação de todos os recursos por desempenho de maneira uniforme e, assim, conseguir identificar os melhores recursos. Com isso, para lidar com a indisponibilidade de informações, essa nova heurística define duas técnicas de classificação de recursos:

- ***Otimista***: A técnica *Otimista* considera que os recursos *com informação disponível* são, em média, melhores que os recursos *sem informação*. Ou seja, essa técnica apresenta uma visão *otimista* em relação aos recursos que possuem informação disponível, classificando estes como sendo melhores que os que não possuem.

- ***Pessimista***: A técnica *Pessimista* classifica os recursos exatamente de maneira inversa, ou seja, os recursos que *possuem informação disponível* são considerados piores que os recursos *sem informação*. Assim, essa técnica apresenta uma visão *pessimista* com relação à informação disponível, apostando na hipótese de que os recursos *sem informação* possam apresentar um melhor desempenho.

Como podemos perceber, devido ao fato de não ser possível obter informação completa do grid, o principal objetivo dessas técnicas consiste em "inferir" o desempenho dos recursos dos quais não se tem informação. Assim, os recursos são classificados em dois grupos denominados *best* e *worst*, onde o primeiro grupo refere-se aos recursos considerados como tendo melhor desempenho e o segundo refere-se aos recursos considerados como tendo desempenho inferior. Note que a distribuição dos recursos *com informação* e *sem informação* em cada um dos grupos depende da técnica adotada. Assim, utilizando a técnica *Pessimista*, os recursos *com informação* são inseridos no grupo *worst* enquanto os *sem informação* no grupo *best*. De modo análogo, utilizando a técnica *Otimista*, os recursos *sem informação* são classificados no grupo *worst* e os recursos com informação no grupo *best*.

Além disso, a técnica *Otimista* aplica ainda um filtro nos recursos *com informação* de

forma que nem todos estes recursos fazem parte do grupo *best*. Para isso, essa técnica exclui do grupo *best* os recursos que apresentam desempenho inferior a um determinado limite e, então, os insere no grupo *worst* juntamente com os recursos *sem informação*. Esse limite é calculado utilizando-se uma tarefa qualquer como referência para estimar o desempenho de cada recurso *com informação* na execução de tal tarefa. A partir disso é calculada a média e o desvio padrão do desempenho de todos os recursos *com informação*, sendo o limite definido pelo valor da média mais o desvio padrão.

O Algoritmo 1 descreve o funcionamento da heurística de escalonamento *Information Adaptive* quando aplicada a técnica *Otimista*. A heurística recebe como entrada um job J e um grid G. Primeiramente, as tarefas do job são ordenadas de acordo com seu custo da maior para a menor. Em seguida, os recursos são divididos nos grupos *best* e *worst*, onde o primeiro contém apenas recursos *com informação* e o segundo possui recursos *sem informação*, além de recursos *com informação* que apresentam desempenho inferior a um limite estabelecido pelo cálculo descrito anteriormente (CT_{limite}). Em um primeiro momento, as tarefas de maior custo são escalonadas nos recursos do grupo *best* utilizando critérios semelhantes aos da heurística *Dynamic FPLTF* até que todos os recursos estejam alocados. Depois, as menores tarefas são escalonadas nos recursos do grupo *worst* utilizando a heurística *WQR* devido à indisponibilidade de informações de grande parte dos recursos desse grupo.

O Algoritmo 2 descreve o funcionamento da técnica *Pessimista* da heurística *Information Adaptive*. Alguns trechos do código são semelhantes ao Algoritmo 1 como o fato de as tarefas serem ordenadas pelo custo e os recursos do grid divididos em dois grupos. No entanto, esta divisão é efetuada de maneira inversa, de forma que o grupo *best* engloba todos os recursos *sem informação* enquanto o grupo *worst* contém todos aqueles *com informação*. Outro ponto que diferencia essa técnica da anterior são os critérios de escalonamento aplicados para cada grupo. Inicialmente, as tarefas de maior custo computacional são escalonadas no grupo *best* usando critério semelhante à WQR. Em seguida, as menores tarefas são escalonadas no grupo *worst* utilizando um método inverso ao da heurística *Dynamic FPLTF*, ou seja, as tarefas de menor custo são escalonadas primeiro para os seus respectivos melhores processadores[1]. Essa forma inversa da heurística *Dynamic FPLTF* é aplicada no intuito

[1]Podemos dizer que neste caso aplica-se um algoritmo denominado "FPSTF" (***F**astest **P**rocessor to **S**mallest **T**ask **F**irst*)

de evitar que tarefas muito custosas sejam escalonadas em recursos do grupo *worst* e que, assim, possam prejudicar o desempenho da aplicação.

Note que ambas as técnicas procuram inferir o desempenho de recursos dos quais não se tem informação. Dessa forma, essas técnicas nem sempre conseguem inferir com precisão o desempenho desses recursos devido à grande heterogeneidade do ambiente de grid, o que pode impactar diretamente o desempenho do escalonador conforme veremos no Capítulo 5.

Nesse sentido, na tentativa de minimizar este problema, investigamos e definimos ainda uma terceira técnica denominada *Híbrida*, que faz uso das duas anteriores de acordo com as condições do ambiente. O objetivo dessa técnica consiste em descobrir em tempo de execução qual técnica (*Otimista* ou *Pessimista*) é mais apropriada para determinado ambiente de grid. Para isso, essa técnica define como parâmetro de entrada um *fator de ajuste* que indica qual a porcentagem de tarefas da aplicação deve ser processada durante a fase de descoberta.

O funcionamento da técnica *Híbrida* é descrito no Algoritmo 3. Além do grid G e do Job J, o algoritmo recebe ainda como entrada o valor do *fator de ajuste* (fa). Primeiramente, as tarefas do Job J são divididas em dois conjuntos, $J1$ e $J2$, de forma que ambos tenham em média o mesmo custo computacional total. Depois, cada conjunto tem suas tarefas ordenadas pelo custo em ordem ascendente[2]. Em seguida, as tarefas do job $J1$ são escalonadas usando a técnica *Otimista* enquanto as tarefas do job $J2$ são escalonadas usando a técnica *Pessimista*. Este processo de escalonamento é efetuado até que a quantidade de tarefas do job J definida pelo *fator de ajuste* tenha sido processada (ex.: 10%). Após isso, verifica-se qual grupo teve o maior número de tarefas finalizadas. Nesse caso, temos duas situações:

1. Caso o maior número de tarefas finalizadas seja do job $J1$, então os recursos *com informação* são melhores em média do que os recursos *sem informação*. Assim, significa que a técnica *Otimista* "acertou" ao classificá-los como sendo do grupo *best*. Portanto, a técnica *Otimista* torna-se mais apropriada para este grid.

2. Caso o maior número de tarefas finalizadas seja do job $J2$, então os recursos *sem informação* são melhores em média do que os recursos *com informação*. Desse modo,

[2]O fato de ordenar as tarefas em ordem crescente tem por objetivo minimizar as chances de uma tarefa muito custosa ser escalonada de forma ineficiente durante a fase de ajuste

Algoritmo 1 Heurística de Escalonamento *Information Adaptive* - Técnica *Otimista*

Entrada: G,J
Saída: Σ_J

$J \leftarrow$ **ordenaTarefasPeloCustoDescendentemente**(J)

$\tau_{ref} \leftarrow$ **getTarefaAleatoria**(J)
$CT_{limite} \leftarrow$ **calculaMediaDoCT**(G, τ_{ref}) − **calculaDesvioPadraoDoCT**(G, τ_{ref})

$G_{best} \leftarrow \{\forall \pi \in G \mid \pi$ **tem** $informacao$ **e** $CT_{ref,\pi} \leq CT_{limite}\}$
$G_{worst} \leftarrow \{\forall \pi \in G \mid \pi$ **nao tem** $informacao$ **ou** $CT_{ref,\pi} > CT_{limite}\}$

enquanto $(J \neq \emptyset)$ **faça**
  /******** Escalona-se as maiores tarefas no grupo *best* ********/
  $k \leftarrow 1$
  enquanto $(k \leq |J|)$ **e** $(\exists \pi \in G_{best} \mid \pi\ esta\ livre)$ **faça**
    $\pi_{choosen} \leftarrow nil$
    $bestCT \leftarrow \infty$
    para $n \leftarrow 1$ **até** $|G_{best}|$ **faça**
      se (**calculaCT**$(\tau_k, \pi_n) \leq bestCT$) **então**
        $bestCT \leftarrow$ **calculaCT**(τ_k, π_n)
        $\pi_{choosen} \leftarrow \pi_n$
      fim se
    fim para
    associe$(\tau_k, \pi_{choosen})$
    $J \leftarrow J - \{\tau_k\}$
    $k \leftarrow k + 1$
  fim enquanto
  /**/
  /******** Escalona-se as menores tarefas no grupo *worst* ********/
  $k \leftarrow |J|$
  enquanto $(k \geq 1)$ **e** $(\exists \pi \in G_{worst} \mid \pi\ esta\ livre)$ **faça**
    $\pi_{choosen} \leftarrow$ **getPrimeiroProcessadorLivreEm**(G_{worst})
    associe$(\tau_k, \pi_{choosen})$
    $J \leftarrow J - \{\tau_k\}$
    $k \leftarrow k - 1$
  fim enquanto
  /**/
  se $(\forall \pi \in G \mid \pi\ esta\ ocupado)$ **então**
    esperePelaFinalizacaoDeAlgumaTarefa()
  fim se
fim enquanto
/***** Replica as tarefas ainda executando no grupo *worst* ******/
$J_{exec} \leftarrow$ **getTarefasExecutandoEm**(G_{worst})
enquanto $(J_{exec} \neq \emptyset)$ **faça**
  $k \leftarrow 1$
  enquanto $(k \leq |J_{exec}|)$ **e** $(\exists \pi \in G \mid \pi\ esta\ livre)$ **faça**
    /**** $maximoNumeroDeReplicas$ estabelece um limite para replicar tarefas****/
    se (**getNumeroDeReplicas**$(\tau_k)\ <\ maximoNumeroDeReplicas$) **então**
      $\pi_{choosen} \leftarrow$ **getPrimeiroProcessadorLivreEm**(G)
      /**** A letra r indica o numero desta replica****/
      $replica_k^r \leftarrow$ **criaReplica**(τ_k)
      associe$(replica_k^r, \pi_{choosen})$
    fim se
    $k \leftarrow k + 1$
  fim enquanto
  se $(\forall \pi \in G \mid \pi\ esta\ ocupado)$ **então**
    esperePelaFinalizacaoDeAlgumaTarefa()
    canceleTodasAsReplicasDeTarefasFinalizadas()
  fim se
  $J_{exec} \leftarrow$ **getTarefasExecutandoEm**(G_{worst})
fim enquanto
/**/

Algoritmo 2 Heurística de Escalonamento *Information Adaptive* - Técnica *Pessimista*

Entrada: G,J
Saída: Σ_J

$J \leftarrow$ **ordenaTarefasPeloCustoDescendentemente**(J)

$G_{best} \leftarrow \{\forall \pi \in G \mid \pi$ **nao tem** $informacao\}$
$G_{worst} \leftarrow \{\forall \pi \in G \mid \pi$ **tem** $informacao\}$

enquanto $(J \neq \emptyset)$ **faça**
  /******** Escalona-se as maiores tarefas no grupo *best* ********/
  $k \leftarrow 1$
  enquanto $(k \leq |J|)$ **e** $(\exists \pi \in G_{best} \mid \pi\ esta\ livre)$ **faça**
    $\pi_{choosen} \leftarrow$ **getPrimeiroProcessadorLivreEm**(G_{best})
    associe$(\tau_k, \pi_{choosen})$
    $J \leftarrow J - \{\tau_k\}$
    $k \leftarrow k + 1$
  fim enquanto
  /**/
  /******** Escalona-se as menores tarefas no grupo *worst* ********/
  $k \leftarrow |J|$
  enquanto $(k \geq 1)$ **e** $(\exists \pi \in G_{worst} \mid \pi\ esta\ livre)$ **faça**
    $\pi_{choosen} \leftarrow nil$
    $bestCT \leftarrow \infty$
    para $n \leftarrow 1$ **até** $|G_{worst}|$ **faça**
      se (**calculaCT**$(\tau_k, \pi_n) \leq bestCT$) **então**
        $bestCT \leftarrow$ **calculaCT**(τ_k, π_n)
        $\pi_{choosen} \leftarrow \pi_n$
      fim se
    fim para
    associe$(\tau_k, \pi_{choosen})$
    $J \leftarrow J - \{\tau_k\}$
    $k \leftarrow k - 1$
  fim enquanto
  /**/
  se $(\forall \pi \in G \mid \pi\ esta\ ocupado)$ **então**
    esperePelaFinalizacaoDeAlgumaTarefa()
  fim se
fim enquanto
/***** Replica as tarefas ainda executando no grupo *worst* ******/
$J_{exec} \leftarrow$ **getTarefasExecutandoEm**(G_{worst})
enquanto $(J_{exec} \neq \emptyset)$ **faça**
  $k \leftarrow 1$
  enquanto $(k \leq |J_{exec}|)$ **e** $(\exists \pi \in G \mid \pi\ esta\ livre)$ **faça**
    /**** $maximoNumeroDeReplicas$ estabelece um limite para replicar tarefas****/
    se (**getNumeroDeReplicas**$(\tau_k)\ <\ maximoNumeroDeReplicas$) **então**
      $\pi_{choosen} \leftarrow$ **getPrimeiroProcessadorLivreEm**(G)
      /**** A letra r indica o numero desta replica****/
      $replica_k^r \leftarrow$ **criaReplica**(τ_k)
      associe$(replica_k^r, \pi_{choosen})$
    fim se
    $k \leftarrow k + 1$
  fim enquanto
  se $(\forall \pi \in G \mid \pi\ esta\ ocupado)$ **então**
    esperePelaFinalizacaoDeAlgumaTarefa()
    canceleTodasAsReplicasDeTarefasFinalizadas()
  fim se
  $J_{exec} \leftarrow$ **getTarefasExecutandoEm**(G_{worst})
fim enquanto
/**/

significa que a técnica *Pessimista* "acertou" ao classificá-los como sendo do grupo *best*, tornando-se, portanto, mais apropriada para este grid.

Por fim, a partir deste ponto, a técnica escolhida é então utilizada durante o restante do processo de escalonamento.

Algoritmo 3 Heurística de Escalonamento *Information Adaptive* - Técnica *Híbrida*

Entrada: G, J, fa
Saída: Σ_J

$divideJobProporcionalmenteEmDois(J1, J2)$

$J1 \leftarrow$ **ordenaTarefasPeloCustoAscendentemente**$(J1)$
$J2 \leftarrow$ **ordenaTarefasPeloCustoAscendentemente**$(J2)$

/*****Fase de Ajuste: Escalona-se $J1$ usando Otimista e $J2$ usando Pessimista*****/
$qtdeTarefas\_J1 = 0$
$qtdeTarefas\_J2 = 0$
$totalTarefasEscalonadas = 0$
enquanto $(J1 \neq \emptyset)$ **e** $(J2 \neq \emptyset)$ **e** $(totalTarefasEscalonadas < (fa \times |J|))$ **faça**
  /*** $escalonaJobUsandoCriteriosDaTecnica(Job, Tecnica)$ retorna a quantidade de tarefas escalonadas ***/
  $qtdeTarefas\_J1 = qtdeTarefas\_J1 +$ **escalonaJobUsandoCriteriosDaTecnica**$(J1, Otimista)$
  $qtdeTarefas\_J2 = qtdeTarefas\_J2 +$ **escalonaJobUsandoCriteriosDaTecnica**$(J2, Pessimista)$
  $totalTarefasEscalonadas = qtdeTarefas\_J1 + qtdeTarefas\_J2$
fim enquanto
/**********Fim da fase de ajuste**********/
/******** Escalona-se o restante das tarefas de J usando a tecnica escolhida**********/
se $(qtdeTarefas\_J1 > qtdeTarefas\_J2)$ **então**
  escalonaJobUsandoCriteriosDaTecnica$(J, Otimista)$
senão se $(qtdeTarefas\_J1 \leq qtdeTarefas\_J2)$ **então**
  escalonaJobUsandoCriteriosDaTecnica$(J, Pessimista)$
fim se
/**********/

De acordo com as três técnicas descritas anteriormente, podemos concluir que a heurística *Information Adaptive* consegue lidar com a falta de informação dos recursos do grid. Entretanto, como ela foi projetada com base na heurística *Dynamic FPLTF*, seu correto funcionamento depende ainda que a informação sobre o custo computacional das tarefas esteja disponível conforme mostram os Algoritmos 1 e 2. A estratégia de escalonamento descrita na próxima seção apresenta uma solução alternativa para este problema.

4.2 Adaptando Heurísticas de replicação

Sabemos que as heurísticas de replicação não utilizam informação alguma durante o escalonamento de aplicações no grid, o que resulta, algumas vezes, em mapeamentos ineficientes

entre tarefas e recursos. Dessa forma, no intuito de manter um bom desempenho na execução das aplicações, essas heurísticas utilizam a técnica de replicação de tarefas no final do escalonamento. Entretanto, essa técnica tem como principal desvantagem o desperdício de recursos ocasionados pelo consumo extra de recursos necessários para execução das réplicas.

Nesta seção apresentamos uma nova estratégia de escalonamento, inspirada nas heurísticas de replicação, que gerencia melhor o processo de replicação de tarefas através do uso da informação que esteja disponível sobre a aplicação e os recursos. Em particular, definimos uma nova heurística de escalonamento baseada na heurística WQR. Tal heurística utiliza um novo critério de replicação de tarefas com o objetivo de diminuir o nível de desperdício de recursos, mantendo o bom desempenho da aplicação. Denominamos essa heurística de *Adaptive WQR*.

Quando não há nenhuma informação disponível, o funcionamento da heurística *Adaptive WQR* é semelhante ao WQR. A grande diferença entre as duas está justamente quanto ao uso de informação, onde a primeira faz uso de informações disponíveis sobre o ambiente, enquanto que a segunda jamais utiliza informação alguma. No Algoritmo 4, vemos que *Adaptive WQR* recebe como entrada um job J e um grid G. Primeiramente, os recursos do grid G são divididos em dois grupos: recursos *com informação* (G_{info}) e recursos *sem informação* (G_{no_info}). Em seguida, no início da fase de escalonamento, verifica-se se o job J possui informação disponível sobre o custo de suas tarefas. Caso positivo, as tarefas são ordenadas pelo custo da maior para a menor e o escalonamento é realizado da seguinte forma: primeiro as tarefas são alocadas aos recursos *com informação* (G_{info}) usando FPLTF e depois as tarefas restantes são alocadas aos recursos *sem informação* (G_{no_info}) utilizando *WorkQueue*. Caso contrário, não é feita essa classificação de tarefas e todas as tarefas de J são escalonadas nos recursos disponíveis em G de maneira aleatória (utilizando *WorkQueue*).

Em seguida, quando não há mais nenhuma tarefa a escalonar e, pelo menos, um recurso está disponível, a fase de replicação é iniciada para as tarefas que ainda estão executando. No início dessa fase, verifica-se se o job J possui informação disponível sobre o custo de suas tarefas. Caso positivo, as tarefas são ordenadas pelo custo da maior para a menor. Caso contrário, não é feita essa classificação. Então, para cada tarefa em execução, é verificado se o recurso no qual ela está executando possui ou não informação disponível. Dessa forma, são utilizados os quatro critérios de replicação de tarefas descritos a seguir:

- **Informação disponível para aplicação e recursos**: Uma tarefa é replicada apenas quando o tempo estimado restante para finalizar a tarefa for maior que o tempo estimado para executar a nova réplica. Para isso, é calculado o tempo restante necessário para finalização da tarefa atual e o tempo que a nova réplica gastaria para ser executada no melhor recurso disponível no momento. Então, para cada tarefa, calcula-se o valor da diferença entre esses dois tempos. A tarefa que apresentar o maior valor é replicada primeiro.

- **Informação disponível para aplicação e não disponível para recursos**: Nessa situação, o processo de replicação de tarefas é o mesmo utilizado por WQR com uma pequena diferença: pelo fato de haver informação disponível sobre a aplicação, as maiores tarefas são selecionadas para serem replicadas primeiro.

- **Informação não disponível para aplicação e disponível para recursos**: Nesse caso, uma tarefa é replicada apenas quando algum recurso disponível apresenta um desempenho[3] melhor que o recurso no qual a tarefa está executando atualmente. Note que diferente da primeira situação, não é possível saber se a réplica irá finalizar antes de sua tarefa correspondente mesmo que ela seja replicada em um recurso com melhor desempenho.

- **Informação não disponível para aplicação e recursos**: Nessa situação, a fase de replicação utiliza exatamente os mesmo critérios de WQR.

É importante ressaltar que como os recursos são divididos em dois grupos, as tarefas que estão executando em um grupo, são replicadas apenas em recursos disponíveis desse mesmo grupo. Ou seja, tarefas alocadas em recursos *com informação* são replicadas apenas em recursos *com informação* e vice-versa. Esse critério permite que se tenha um maior controle do nível de desperdício de recursos.

Por fim, essa heurística pode ser considerada mais completa do que a apresentada na seção anterior, pois a mesma tem a capacidade de lidar com a falta de informações tanto do grid quanto da aplicação, diferentemente da heurística *Information Adaptive* que depende ainda de informação sobre o custo das tarefas da aplicação para seu correto funcionamento.

[3] O valor do desempenho é calculado com base na velocidade do processador (cpu_speed) e na carga de processamento disponível (cpu_avail)

Algoritmo 4 Heurística de Escalonamento *Adaptive WQR*

Entrada: G,J
Saída: Σ_J
$G_{info} \leftarrow \{\forall \pi \in G \mid \pi$ **tem** $informacao\}$
$G_{no\_info} \leftarrow \{\forall \pi \in G \mid \pi$ **nao tem** $informacao\}$
se $(\forall \tau \in J \mid \tau$ **tem** $informacao)$ **então**
  $J \leftarrow$ **ordenaTarefasPeloCustoDescendentemente**(J)
fim se
/*** Fase de Escalonamento ***/
enquanto $(J \neq \emptyset)$ **faça**
  se $(\forall \tau \in J \mid \tau$ **tem** $informacao)$ **então**
    escalonaTarefasUsandoFPLTFEm(G_{info})
    escalonaTarefasUsandoWorkQueueEm(G_{no_info})
  senão
    escalonaTarefasUsandoWorkQueueEm(G)
  fim se
fim enquanto
/***** Fase de Replicação (replica as tarefas ainda executando) ******/
$J_{exec} \leftarrow$ **getTarefasExecutandoEm**(G)
se $(\forall \tau \in J_{exec} \mid \tau$ **tem** $informacao)$ **então**
  $J_{exec} \leftarrow$ **ordenaTarefasPeloCustoDescendentemente**(J_{exec})
fim se
enquanto $(J_{exec} \neq \emptyset)$ **faça**
  $maiorCTRestante \leftarrow -1; k \leftarrow 1; \pi_{choosen} \leftarrow t_{choosen} \leftarrow nil; replicar \leftarrow false$
  enquanto $(k \leq |J_{exec}|)$ **e** $(\exists \pi \in G \mid \pi\ esta\ livre)$ **e** $(replicar = false)$ **faça**
    /**** $maximoNumeroDeReplicas$ estabelece um limite para replicar tarefas****/
    se (**getNumeroDeReplicas**$(\tau_k) < maximoNumeroDeReplicas$) **então**
      $\pi_k \leftarrow$ **getProcessadorAlocadoPara**(τ_k)
      se $(\pi_k \in G_{info})$ **então**
        $\pi_{choosen} \leftarrow$ **getProcessadorLivreComMelhorDesempenhoEm**(G_{info})
        se $(\tau_k$**tem** $informacao)$ **então**
          se (**calculaCT**$(\tau_k, \pi_{choosen}) <$ **calculaCTRestante**(τ_k, π_k)) **então**
            se (**calculaCTRestante**$(\tau_k, \pi_k) > maiorCTRestante$) **então**
              $\tau_{choosen} \leftarrow \tau_k$
              $maiorCTRestante \leftarrow$ **calculaCTRestante**(τ_k, π_k)
            fim se
          fim se
        senão se $(\tau_k$**nao tem** $informacao)$ **então**
          se (**calculaDesempenho**$(\pi_{choosen}) >$ **calculaDesempenho**(π_k)) **então**
            $\tau_{choosen} \leftarrow \tau_k$
          fim se
        fim se
      senão se $(\pi_k \in G_{no\_info})$ **então**
        $\pi_{choosen} \leftarrow$ **getPrimeiroProcessadorLivreEm**(G_{no_info})
        $\tau_{choosen} \leftarrow \tau_k$
        $replicar \leftarrow$ **true**
      fim se
    fim se
    $k \leftarrow k + 1$
  fim enquanto
  se $(\tau_{choosen} \neq nil)$ **então**
    $replica^r_{choosen} \leftarrow$ **criaReplica**$(\tau_{choosen})$
    associe$(replica^r_{choosen}, \pi_{choosen})$
  senão
    esperePelaFinalizacaoDeAlgumaTarefa()
  fim se
  se $(\forall \pi \in G \mid \pi\ esta\ ocupado)$ **então**
    esperePelaFinalizacaoDeAlgumaTarefa()
    canceleTodasAsReplicasDeTarefasFinalizadas()
  fim se
  $J_{exec} \leftarrow$ **getTarefasExecutandoEm**(G)
fim enquanto

Por ter sido derivada de uma heurística de replicação, *Adaptive WQR* foi projetada de maneira mais simples e menos dependente de informação.

4.3 Considerações finais

Neste capítulo foi descrita uma nova abordagem de escalonamento de aplicações em grids computacionais adaptável ao ambiente de grid quanto à disponibilidade de informações. Em particular, definimos e apresentamos duas novas heurísticas de escalonamento: *Information Adaptive* e *Adaptive WQR*. A heurística *Information Adaptive* consiste numa tentativa de adaptar a heurística *bin-packing Dynamic FPLTF* para lidar com a falta de informação sem degradar o seu desempenho. Por outro lado, a heurística *Adaptive WQR* consiste na tentativa de adaptar a heurística de replicação WQR ao uso de informação disponível de forma a controlar melhor o nível de desperdício de recursos. Os resultados da avaliação de desempenho das heurísticas aqui apresentadas são discutidos no Capítulo 5.

Capítulo 5

Avaliação de Desempenho

Neste capítulo apresentamos os resultados da avaliação de desempenho das heurísticas discutidas no Capítulo 4. As heurísticas foram comparadas com *WQR* e *Dynamic FPTLF*. A decisão de utilizar essas duas heurísticas como referência deve-se ao fato delas representarem soluções eficientes para escalonamento de aplicações BoT em grids computacionais, e além disso, de terem servido como base para definição e implementação das heurísticas propostas neste trabalho. *WQR* consiste numa heurística eficiente para escalonamento de aplicações sem utilizar informação alguma, enquanto que *Dynamic FPTLF* apresenta bons resultados usando informação completa sobre os recursos e a aplicação.

A metodologia da avaliação de desempenho foi baseada na execução de simulações e experimentos, onde são comparados os valores do *makespan* e do nível de desperdício de recursos apresentados por cada heurística de escalonamento. Os cenários avaliados e os resultados obtidos são apresentados a seguir.

5.1 Simulações

Para conduzir as simulações foi utilizada a ferramenta de simulação SimGrid (LEGRAND; MARCHAL; CASANOVA, 2003), que provê as funcionalidades básicas para simulação de aplicações distribuídas em grids computacionais. A partir dessa ferramenta, foi desenvolvido um simulador que permitiu o estudo e avaliação de heurísticas de escalonamento de aplicações BoT em grids computacionais. Sabendo que o desempenho do escalonamento é bastante influenciado pelas características dos recursos do grid e da aplicação (FEITELSON; RUDOLPH,

1998) (FEITELSON, 2003), o simulador foi implementado de forma que pudesse contemplar uma grande variedade de cenários com diferentes configurações.

Para cada cenário simulado, foram executadas as heurísticas *Dynamic FPLTF*, *WQR*, *Information Adaptive* e *Adaptive WQR*. A heurística *Information Adaptive* foi executada de três formas, sendo cada uma com uma técnica diferente (*Otimista*, *Pessimista* e *Híbrida*). Todas as heurísticas baseadas em replicação foram executadas com três níveis de replicação diferentes: $2x$ (limite de replicação 2), $4x$ (limite de replicação 4) e xx (limite de replicação infinito). A heurística *Adaptive WQR* foi executada de duas formas para cada cenário, sendo uma considerando a informação sobre a aplicação disponível e outra sem utilizar esta informação. Por fim, a técnica *Híbrida* da heurística *Information Adaptive* foi executada com três valores diferentes para o *fator de ajuste* (10%, 25% e 50%) no intuito de avaliar qual o valor mais apropriado para esse parâmetro.

Dessa forma, cada simulação consistia na execução, sob um mesmo cenário, de um total de 21 escalonadores distintos: *Dynamic FPLTF*, *WQR*-($2x$, $4x$, xx), *Otimista*-($2x$,$4x$,xx), *Pessimista*-($2x$,$4x$,xx), *Híbrida*-10-($2x$,$4x$,xx), *Híbrida*-25-(xx), *Híbrida*-50-(xx), *AdaptiveWQR*-($2x$,$4x$,xx) e *AdaptiveWQR-JobInfo*-($2x$,$4x$,xx). *AdaptiveWQR-JobInfo* indica que o escalonador está executando a heurística *Adaptive WQR* com a utilização de informação sobre a aplicação. *Híbrida*-10 indica a execução da heurística *Information Adaptive* utilizando a técnica *Híbrida*, com o *fator de ajuste* configurado para o valor 10%. Da mesma forma, *Híbrida*-25 e *Híbrida*-50 correspondem à execução dessa técnica com os valores 25% e 50%, respectivamente.

Com o objetivo de obter resultados estatisticamente válidos, foram simulados um total de 9000 cenários. Após uma análise estatística sobre a média e o desvio padrão desses resultados, pudemos constatar que essa quantidade de simulações já era suficiente para que os resultados obtidos apresentassem 95% de intervalo de confiança com uma margem de erro de apenas 2% para mais ou para menos (DEVORE, 2000). Como cada simulação levava em média 2 horas para executar, seriam necessários aproximadamente 750 dias para executar todo o conjunto de simulações em uma única máquina. Felizmente, o próprio conjunto de simulações representava uma aplicação BoT e, assim, as simulações puderam ser executadas em um ambiente de grid computacional. Como ambiente de execução das simulações, utilizamos o OurGrid (CIRNE et al., 2006), que consiste em um middleware para grid com-

putacional que dá suporte à execução de aplicações paralelas do tipo BoT. O OurGrid provê uma infraestrutura *peer-to-peer* que permite o compartilhamento de recursos computacionais em escala mundial a qualquer usuário interessado em se juntar ao grid e executar suas aplicações paralelas. A Comunidade OurGrid (http://www.ourgrid.org) é um grid aberto e em produção desde 2004, que usa o middleware OurGrid como plataforma. Durante a execução de nossas simulações, conseguimos utilizar, em média, 40 máquinas obtidas através da Comunidade OurGrid (ver *http://status.ourgrid.org* para ter uma visão do estado atual da comunidade).

5.1.1 Cenários

No intuito de investigar o impacto do dinamismo e da heterogeneidade do ambiente de grid no processo de escalonamento, as simulações foram executadas sobre uma grande quantidade de cenários distintos. Os cenários variavam nos seguintes aspectos: i) *heterogeneidade dos recursos do grid*; ii) *heterogeneidade e granularidade da aplicação*; iii) *relação entre o número de tarefas e recursos* e iv) *disponibilidade e erro das informações sobre o ambiente*.

Heterogeneidade do Grid

De acordo com a Seção 3.2, cada tarefa possui um custo computacional que expressa quanto tempo tal tarefa executaria em um *processador-referência* (π_{ref}) em regime dedicado. Como o grid pode ser formado por processadores que foram adquiridos em diferentes épocas, é uma tendência que os grids sejam bastante heterogêneos (i.e. a velocidade dos processadores apresenta uma grande variação). Considerando-se o valor da velocidade relativa do *processador-referência* como sendo igual a 1 ($\pi_{ref}.cpu\_speed = 1$), um processador dedicado e com velocidade relativa igual a 2 ($\pi_{ref}.cpu\_speed = 2$) executaria uma tarefa de 100 segundos em 50 segundos.

Para analisar o impacto da heterogeneidade do grid, definimos um modelo da heterogeneidade dos processadores do grid inspirado na Lei de Moore (DAVIDSON; GARCIA-MOLINA; SKEEN, 1985), que afirma que a velocidade dos processadores praticamente dobra a cada 18 meses. A Lei de Moore sugere, então, que num período de 54 meses (4 anos e meio) a velocidade de novos processadores aumenta 8 vezes. Dessa forma, para que pudéssemos cobrir

até aproximadamente 5 anos de vida útil dos processadores, consideramos quatro níveis de heterogeneidade que vão desde um grid homogêneo até um grid bastante heterogêneo. Em cada nível, a velocidade dos processadores varia de acordo com uma distribuição uniforme de tal forma que a média da velocidade de todos processadores do grid seja aproximadamente igual a 10. A faixa de valores desses níveis são listados na Tabela 5.1, onde $U(x, y)$ representa uma distribuição uniforme no intervalo $[x, y]$. Assim, para o nível de heterogeneidade $1x$, todos os processadores do grid possuirão velocidade igual a 10 (grid homogêneo). Em contrapartida, para o nível de heterogeneidade $8x$ (grid bastante heterogêneo), o processador mais rápido terá sua velocidade 8 vezes maior que a do mais lento.

Heterogeneidade do Grid	**Distribuição da velocidade dos processadores**
$1x$	$U(10, 10)$
$2x$	$U(6.7, 13.4)$
$4x$	$U(4, 16)$
$8x$	$U(2.2, 17.6)$

Tabela 5.1: Níveis de Heterogeneidade dos Processadores do Grid

A velocidade total do grid é determinada pela soma da velocidade de todos os seus processadores. Para todas as simulações a velocidade do grid foi fixada em 500. Um grid é "construído" pela adição de um processador por vez até que a sua velocidade atinja o valor 500. Logo, como a média da velocidade dos processadores é aproximadamente 10 em qualquer nível de heterogeneidade (ver Tabela 5.1), a quantidade de processadores do grid mantém-se em torno 50 em todos os cenários. Como o grid é composto por vários sites, os processadores são distribuídos proporcionalmente entre eles. No nosso modelo, o número de sites varia de acordo com a distribuição uniforme $U(10, 12)$.

Vale salientar ainda que os recursos do grid são compartilhados por várias aplicações e, portanto, nem sempre os processadores estão em regime dedicado durante todo o tempo ($\pi.cpu\_avail = 100\%$). Dessa forma, simulamos também a variação de carga de processamento disponível nos processadores do grid ao longo do tempo. Para isso, utilizamos arquivos de *traces* obtidos através de monitorações em recursos reais com o NWS (WOLSKI, 2003). A ferramenta NWS (*Network Weather Service*) consiste em um serviço distribuído

que periodicamente monitora e prevê de forma dinâmica o desempenho que recursos computacionais e de rede podem prover durante um determinado intervalo de tempo. Os arquivos de *traces* produzidos pelo NWS foram utilizados pelo simulador para que os processadores simulados apresentassem comportamento similar a processadores reais. Além disso, como o simulador foi desenvolvido de forma a permitir a modelagem de aplicações *data-intensive*, também foi necessário realizar monitorações de carga na rede no intuito de produzir *traces* que refletissem o comportamento dos *links* que conectam os processadores do grid. Nesse caso, utilizamos ainda *traces* para latência (lat) e largura de banda disponível (bw_avail) durante as simulações. Embora em nossas simulações tenhamos executados apenas aplicações *cpu-intensive*, a configuração desses *traces* era necessária para que o simulador funcionasse corretamente.

Heterogeneidade e Granularidade da Aplicação

Em todos os cenários analisados, o custo computacional total da aplicação foi fixado em 3125000 segundos. Nesse sentido, em um mundo perfeito (todos os processadores estando em regime dedicado e todas as suas tarefas finalizando ao mesmo tempo), essa aplicação seria finalizada em aproximadamente 104 minutos em um grid de velocidade total igual a 500. Considerando essa aplicação, variamos suas características quanto à granularidade e heterogeneidade de suas tarefas. A granularidade de uma aplicação refere-se ao valor médio do custo computacional de suas tarefas. Por outro lado, a heterogeneidade das tarefas reflete a variação do custo de cada tarefa em relação ao custo médio.

Nas simulações, foram considerados três grupos de aplicação definidos pelos seguintes valores de granularidade: 5000, 25000 e 125000 segundos. Para simular a heterogeneidade das tarefas dentro de cada grupo, consideramos cinco fatores de heterogeneidade: 0%, 25%, 50%, 75% e 100%. Sendo assim, cada grupo de granularidade é subdividido em cinco grupos conforme o fator de heterogeneidade de suas tarefas, resultando em 15 tipos de aplicações diferentes. Por exemplo, considerando-se uma aplicação com granularidade 5000 segundos, temos que quando o fator de heterogeneidade utilizado é 0%, todas as tarefas possuem o mesmo custo computacional (média de 5000 segundos, com variação de 0%), ou seja, as tarefas são homogêneas. Quando o fator de heterogeneidade é 25%, o custo de cada tarefa é variável de acordo com a seguinte distribuição uniforme $U(4.375, 5.625)$ (média de 5000,

com variação de $12,5\%$ para mais e $12,5\%$ para menos). Para o valor de heterogeneidade igual a 50%, o custo das tarefas é definido pela distribuição uniforme $U((3750, 6250)$ (média de 5000, com variação de 25% para mais e 25% para menos). Os fatores de heterogeneidade 75% e 100% seguem o mesmo princípio.

Semelhante ao modo como é "construído" o grid, as tarefas são adicionadas na aplicação de acordo com cada distribuição até que a soma do custo de cada tarefa atinja o valor máximo de 3125000 segundos.

Relação entre o Número de Tarefas e Recursos

Outro aspecto analisado foi o impacto da relação entre o número de tarefas da aplicação e o número de processadores do grid no desempenho do escalonamento. Note que como os valores para a velocidade total do grid e para o custo total da aplicação são fixos, esta relação é inversamente proporcional à granularidade da aplicação. Ou seja, ao variar o valor da granularidade em 5000, 25000 e 125000 segundos, o número médio de tarefas também varia em valores aproximados de 625, 125 e 25 tarefas, respectivamente. Com isso, a relação de tarefas por processadores também varia conforme é mostrado na Tabela 5.2.

Granularidade da Aplicação	**No. de Tarefas**	**No. de Tarefas por Processadores**
5000	625	$12,5$
25000	125	$2,5$
125000	25	$0,5$

Tabela 5.2: Relação entre o número médio de tarefas e o número de processadores

Essa relação indica quantas tarefas, em média, existem para serem processadas por cada processador. A variação dos valores desta relação proporciona uma avaliação do desempenho dos algoritmos tanto em situações em que existem muito mais tarefas do que processadores disponíveis no grid, bem como em situações onde há mais processadores do que tarefas.

Disponibilidade da Informação

Quanto à disponibilidade de informações, fizemos algumas simplificações no modelo aplicado nas simulações. Consideramos que cada um dos conjuntos de informações $\Im_{app}$, $\Im_{proc}$

ou $\Im_{net}$ (ver Seção 3.3) pode estar totalmente disponível ou não. Em outras palavras, ou um determinado conjunto possui todas as suas informações disponíveis ou não possui nenhuma delas disponível.

Além disso, consideramos também que os conjuntos de informações sobre grid ($\Im_{proc}$ e $\Im_{net}$) estão disponíveis em nível de site. Ou seja, um site possui toda a informação disponível para todos os seus recursos ou toda a informação está indisponível para todos os recursos. Com isso, definimos dois tipos de *sites*: *full-info* para indicar os do primeiro caso e *empty-info* para os do segundo caso.

Dessa forma, é possível evidenciar três possíveis configurações de um grid quanto à disponibilidade de informações. Um grid que possua apenas *sites full-info* é um grid *completo de informações*. Quando possui apenas *sites empty-info*, o grid não possui nenhuma informação disponível (*vazio de informações*). Por fim, se o grid possui *sites* de ambos os tipos, ele é considerado um grid com *informação parcialmente disponível*.

Simulamos cenários com diferentes níveis de disponibilidade de informação sobre o grid, abrangendo desde aqueles com toda a informação disponível até os sem informação alguma. Os cenários foram classificados em três categorias de acordo com o nível de informação que possuíam, conforme é mostrado na Tabela 5.3.

Categoria	**Descrição**
$0\% - 10\%$	Até 10% dos sites possuem informação disponível
$30\% - 70\%$	Apenas de 30% a 70% dos sites possuem informação
$90\% - 100\%$	Quase todos o sites do grid possuem informação (de 90% a 100%)

Tabela 5.3: Classificação dos cenários por nível de informação disponível

Erro na Informação

Para obtermos valores que refletissem o comportamento do erro presente nas informações colhidas ao longo do tempo sobre os recursos do grid, conduzimos experimentos com a ferramenta de monitoração de carga NWS. Com isso, modelamos o comportamento do erro para as seguintes informações: carga de processamento (cpu_avail), largura de banda (bw_avail) e latência (lat).

A ferramenta NWS (WOLSKI, 1998) possui dois módulos: *monitoramento* e *previsão*. O módulo de *monitoramento* realiza medições periódicas sobre o estado atual dos recursos ao longo do tempo. O módulo de *previsão* recebe como parâmetro as medições realizadas pelo módulo de monitoramento e, aplicando métodos estatísticos, tenta estimar o desempenho futuro dos recursos em um determinado intervalo de tempo.

Dessa forma, o valor medido pelo módulo de *monitoramento* sobre um dado recurso num determinado instante t, representa a informação sobre o valor do desempenho daquele recurso no instante t. Enquanto que o valor gerado pelo módulo de *previsão* em um dado instante t para um determinado recurso, representa a estimativa do valor do desempenho de tal recurso no instante $t + 1$. A partir disso, podemos definir o erro (ϵ) de uma determinada informação i como sendo a diferença entre o valor real medido no instante t e o valor estimado no instante $t - 1$. Formalmente:

$$i.\epsilon(t) = value_i(t) - prediction_i(t - 1)$$

onde $value_i(t)$ representa o valor real da informação i medido no instante t e $prediction_i(t-1)$ indica o valor estimado da informação i no instante $t-1$ que vale para a medição realizada no instante t. Por fim, $i.\epsilon(t)$ representa o erro associado à informação i medido no instante t.

Note que a validade da informação depende diretamente da freqüência com que são coletadas as informações. Por exemplo, se as informações são coletadas a cada 20 segundos, então o que o NWS medir ou estimar no tempo 25, valerá até o tempo 45 que é quando ocorre outra medição ou previsão. Dessa forma, para previsões de longo prazo, o erro da informação será maior uma vez que a ferramenta considera que o valor medido permanece constante até a próxima previsão, não refletindo a variação ocorrida no intervalo de tempo entre duas previsões. Logo, quanto maior a freqüência de coleta de informações, maior será a precisão das informações colhidas.

Como os valores dos arquivos de *traces* utilizados neste trabalho foram colhidos a cada 10 segundos, o erro apresentado nas informações era mínimo. Vale salientar que os experimentos para modelagem do erro conduzidos neste trabalho foram baseados nos experimentos realizados em (WOLSKI, 1998)(WOLSKI, 2003), cujos resultados também mostraram que o erro presente nas informações utilizadas apresentam em média valores desprezíveis. Com isso, concluiu-se que o impacto no uso informações incorretas sobre o desempenho das heu-

rísticas *bin-packing* torna-se menos danoso do que o impacto de não haver de informações.

5.1.2 Resultados

Nesta seção são apresentados os resultados obtidos nas simulações das heurísticas de escalonamento. Nosso objetivo consiste em mostrar o desempenho das heurísticas *Information Adaptive* e *Adaptive WQR* através de uma análise comparativa com os algoritmos *Dynamic FPLTF* e *WQR*. A heurística *Adaptive WQR* foi analisada considerando duas formas de execução, sendo uma com o uso de informação sobre a aplicação (*AdaptiveWQR-JobInfo*) e outra sem o uso desta informação (*AdaptiveWQR*). Com relação a *Information Adaptive*, foi analisado o desempenho das técnicas *Otimista*, *Pessimista* e *Híbrida*.

Vale lembrar que (conforme descrito na Seção 4.1) a técnica *Híbrida* utiliza, como parâmetro de entrada, um *fator de ajuste* (%) para descobrir em tempo de execução qual das outras duas técnicas (*Otimista* ou *Pessimista*) é mais adequada para o grid em determinado momento e, assim, aplicá-la no restante do processo de escalonamento. Nesse sentido, com o objetivo de investigar um valor apropriado para o *fator de ajuste*, analisamos primeiramente a técnica *Híbrida* com os seguintes valores: 10%, 25% e 50%. Os resultados obtidos para cada valor foram bem próximos conforme podemos ver na Tabela 5.4. Com isso, pudemos concluir que o valor de 10% já era suficiente para permitir o correto funcionamento desta técnica. Assim, nos demais resultados apresentados mais adiante, a técnica *Híbrida* foi analisada considerando-se um *fator de ajuste* de 10%.

Heurística	**Makespan**	**Desperdício**
Híbrida-10	36171 s	25%
Híbrida-25	36922 s	22%
Híbrida-50	37195 s	21%

Tabela 5.4: Média geral do desempenho da técnica *Híbrida* para os fatores de ajuste 10%, 25% e 50%

A seguir são analisados o impacto do *nível de disponibilidade de informação do grid*, do *nível de replicação de tarefas* e da *granularidade da aplicação* sobre o desempenho de cada uma das heurísticas.

Impacto do Nível de Informação Disponível

A partir desse ponto investigamos o impacto do nível de informação disponível no desempenho das heurísticas *Information Adaptive* e *Adaptive WQR*. Os gráficos a seguir apresentam os resultados agrupados por nível de informação ($0\% - 10\%$, $30\% - 70\%$ e $90\% - 100\%$). Os valores exibidos correspondem à média do *makespan* e do desperdício de recursos nos cinco níveis de heterogeneidade do grid, nos cinco níveis de heterogeneidade da aplicação e nos três grupos de granularidade da aplicação. Para as heurísticas que utilizam replicação, foi considerado o nível de replicação infinito (xx).

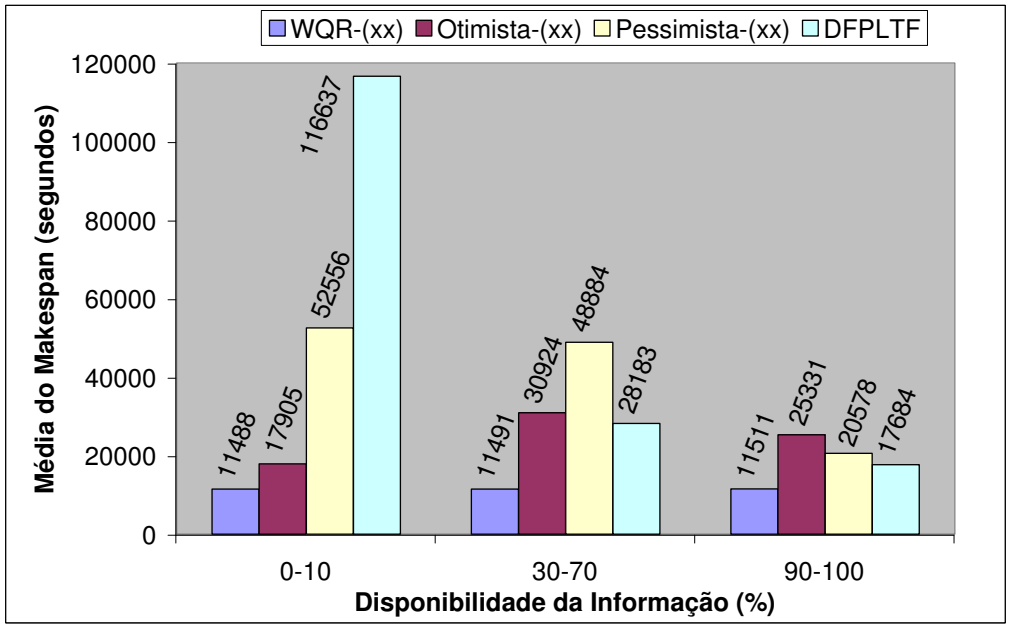
A Figura 5.1 mostra o desempenho da heurística *Information Adaptive* usando as técnicas *Otimista* e *Pessimista* nos três níveis de informação disponível. Os valores do *makespan* e do desperdício de recursos são comparados com os valores apresentados por *Dynamic FPLTF* e *WQR*. De acordo com esses resultados, observamos que a heurística *Dynamic FPLTF* degrada o seu desempenho à medida que diminui o nível de informação disponível enquanto que a heurística *WQR* apresenta um desempenho constante tanto para o *makespan* como para o nível de desperdício. Isso é naturalmente explicado pelo fato da primeira depender de informação completa sobre o ambiente enquanto que a segunda não faz uso de nenhuma informação.

Podemos observar ainda na Figura 5.1 que as técnicas *Otimista* e *Pessimista* apresentam comportamento semelhante, porém de modo inverso. À medida que há menos informação disponível, a técnica *Otimista* melhora o seu desempenho quanto ao *makespan*, porém ela aumenta o nível de desperdício. Diferentemente, a técnica *Pessimista* melhora a média do *makespan* quando a disponibilidade de informações é maior, ao mesmo tempo que também aumenta o nível de desperdício de recursos. Esse comportamento inverso ocorre devido à forma diferente com que os recursos são classificados nos grupos *best* e *worst* por cada uma das técnicas (ver Seção 4.1). Como apenas as tarefas executando sobre os recursos do grupo *worst* são replicadas, a técnica *Otimista* replica mais quando há uma maior quantidade de recursos *sem informação* e a técnica *Pessimista* quando há muitos recursos *com informação*.

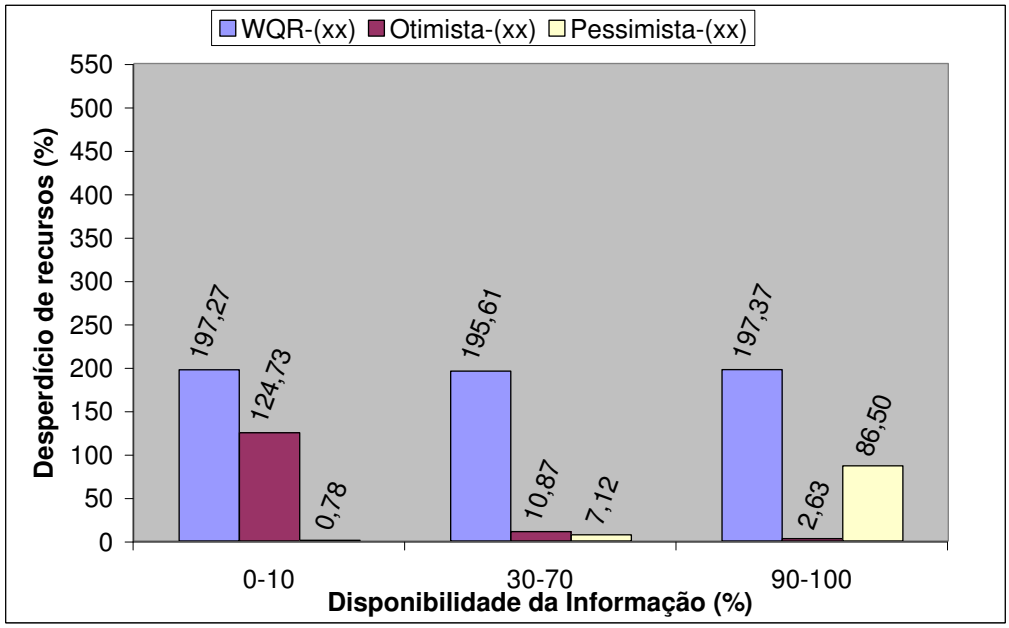
De modo geral, analisando o desempenho das técnicas *Otimista* e *Pessimista*, vemos que elas não alcançaram resultados significativos. Note que embora o nível de desperdício tenha diminuído bastante, os valores apresentados para o *makespan* são ainda muito altos quando comparados com a heurística *WQR*. Além disso, quando comparadas com *Dynamic*

FPLTF, ambas apresentam um melhor desempenho apenas nos cenários com até 10% da informação disponível. O mau desempenho apresentado por ambas as técnicas é explicado pelo fato delas tentarem literalmente "adivinhar" o comportamento de recursos dos quais não se tem informação. De um lado, temos que a técnica *Otimista* acredita que os recursos *com informação* são melhores. Do outro lado, a técnica *Pessimista* acredita na hipótese de que os recursos *sem informação* possam ser melhores. No intuito de refletir a heterogeneidade e dinamismo de um ambiente de grid, sabemos que foi simulada uma grande variedade de cenários. Desse modo, haviam cenários onde as técnicas "acertavam" bem como também haviam outros onde elas "erravam" na classificação dos recursos nos grupos *best* e *worst*. Por conta disso, foi definida e avaliada a técnica *Híbrida* na tentativa de obter melhores resultados.

Na Figura 5.2, apresentamos uma análise comparativa do desempenho da heurística *Information Adaptive* aplicando a técnica *Híbrida* com as heurísticas *Dynamic FPLTF* e *WQR*. O *fator de ajuste* utilizado foi de 10%. Os resultados estão agrupadas por nível de informação disponível. Desses resultados podemos perceber que a técnica *Híbrida* melhora o seu desempenho com relação ao *makespan* na medida que a quantidade de informação disponível aumenta, além de apresentar baixos valores para o nível de desperdício de recursos. Com relação a *Dynamic FPLTF*, note que a técnica Híbrida proporcionalmente degrada menos o seu desempenho, tolerando, assim, melhor a falta de informações sobre os recursos. Entretanto, seu desempenho geral ainda é bastante inferior quando comparada, principalmente, ao desempenho da heurística *WQR*. O péssimo desempenho apresentado pela técnica *Híbrida* pode ser influenciado por eventuais mapeamentos ineficientes que possam ocorrer durante a fase de ajuste da técnica *Híbrida* uma vez que durante essa fase a heurística ainda não sabe qual a melhor técnica a ser aplicada. Outro fator que pode influenciar no desempenho da heurística é a variação de carga dos recursos de forma que não é possível garantir que um determinado cenário de grid seja favorável para a execução da técnica escolhida durante todo o processo de escalonamento. Ou seja, um determinado cenário pode ter sido detectado como sendo favorável para execução da técnica *Otimista* durante a fase de ajuste. No entanto, em algum momento no restante do processo de escalonamento, os recursos *com informação* podem ficar mais sobrecarregados do que os *sem informação*, tornando assim tal cenário favorável para a técnica *Pessimista* devido a essa variação de carga dos recursos.

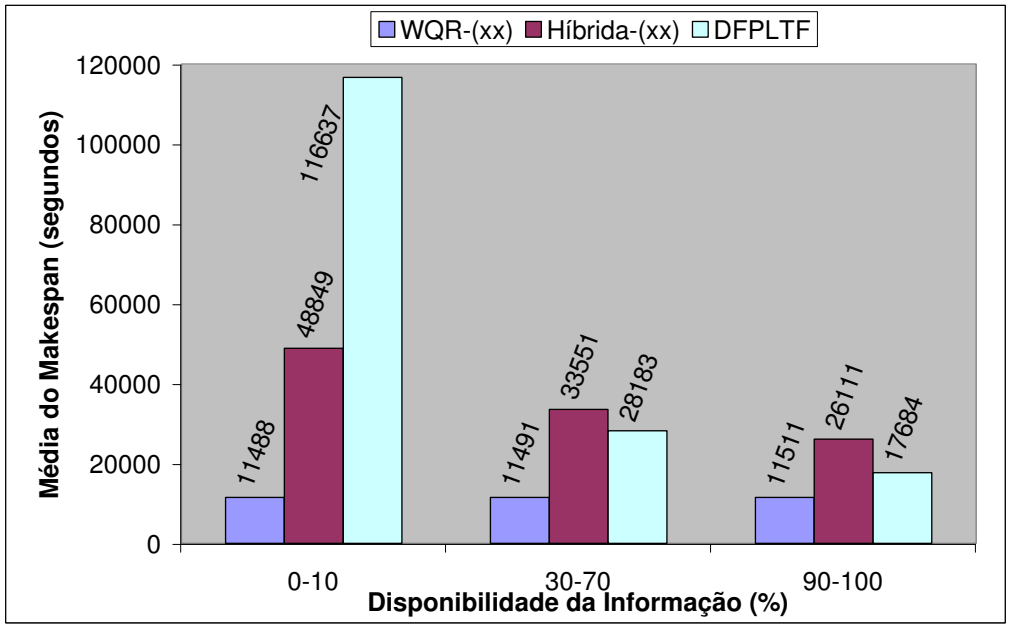


(a) *Makespan* da aplicação

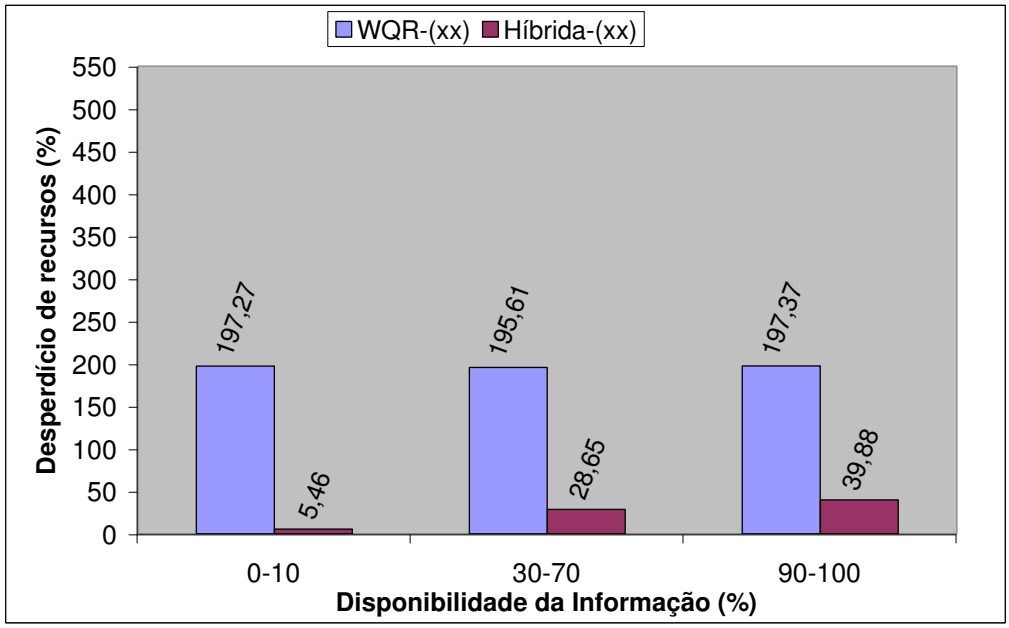


(b) Desperdício de recursos

Figura 5.1: Impacto do nível de informação disponível sobre o grid no desempenho das técnicas *Otimista* e *Pessimista*



(a) *Makespan* da aplicação



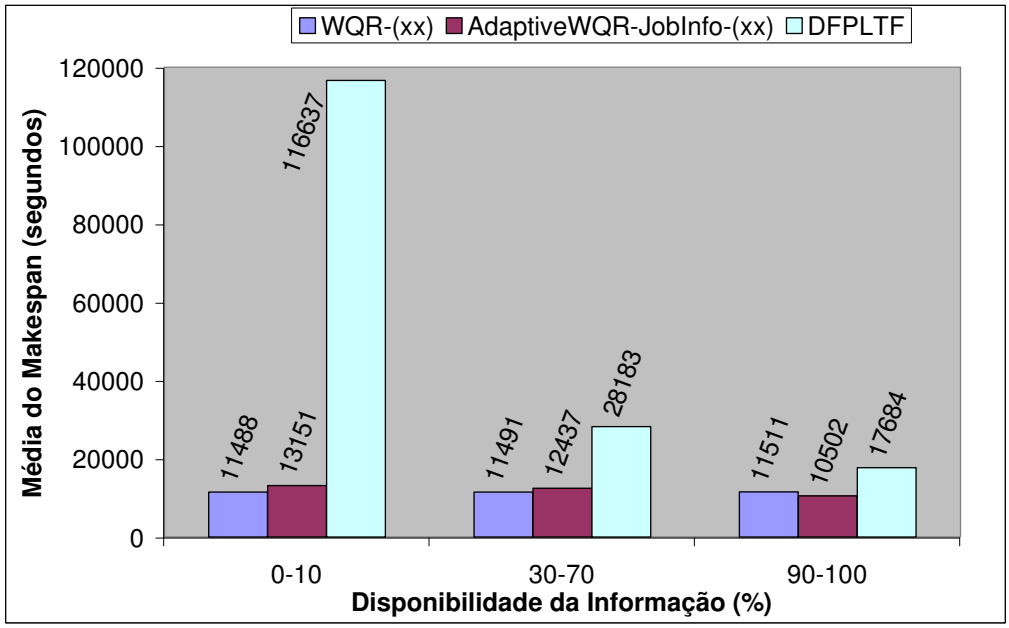
(b) Desperdício de recursos

Figura 5.2: Impacto do nível de informação disponível sobre o grid no desempenho da técnica *Híbrida*

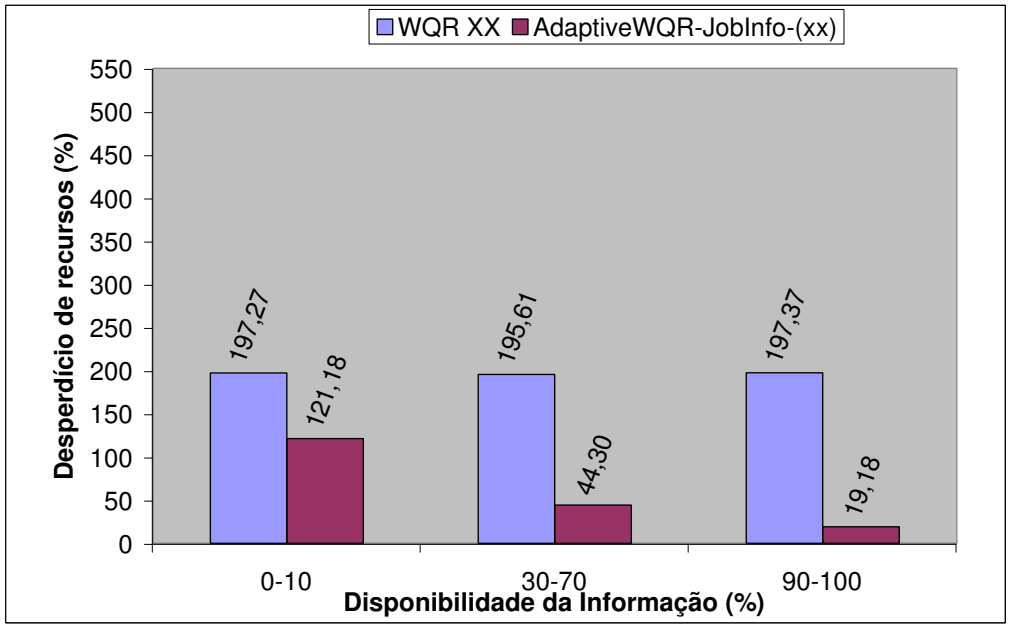
Dessa forma, de acordo com os resultados apresentados e discutidos até aqui, podemos observar que com relação a *Dynamic FPLTF*, a heurística *Information Adaptive* consegue prover um melhor desempenho em ambientes com pouca informação (0%-10%). Sendo assim, os resultados mostraram que *Information Adaptive* tolera a falta de informações de maneira mais eficiente que heurísticas *bin-packing* tradicionais. No entanto, é possível destacar que essa heurística não obteve sucesso na tentativa de diminuir o desperdício de recursos e, ao mesmo tempo, manter o bom desempenho alcançado por heurísticas de replicação como *WQR*. Com isso, pode-se concluir que essa primeira estratégia de adaptar heurísticas *bin-packing* para lidar com a falta de informações demonstrou não ser uma solução eficiente. Outra desvantagem dessa estratégia é que ela ainda depende da disponibilidade de informação sobre a aplicação para funcionar corretamente.

Em seguida investigamos a heurística *Adaptive WQR*, cuja estratégia consiste na adaptação da heurística *WQR* ao uso de informação disponível no intuito de reduzir o desperdício de recursos durante a fase de replicação. Uma importante vantagem a ser destacada dessa estratégia é o fato de ser menos dependente de informação. Diferentemente da heurística anterior, o algoritmo de *Adaptive WQR* consegue funcionar mesmo com a falta de informação sobre as tarefas da aplicação (ver Seção 4.2). Na Figura 5.3, apresentamos uma análise comparativa de *Adaptive WQR* com o desempenho das heurísticas *Dynamic FPLTF* e *WQR*. Os resultados correspondem à média do *makespan* e do nível de desperdício de recursos agrupados por nível de informação. Primeiramente, os cenários foram avaliados considerando que a informação sobre a aplicação estaria disponível e, portanto, era utilizada também pela heurística *Adaptive WQR* (*AdaptiveWQR-JobInfo*).

De acordo com a Figura 5.3(a), podemos observar que, em relação ao *makespan*, a heurística *Adaptive WQR* apresenta um desempenho bastante superior à heurística *Dynamic FPLTF*. Para o nível de informação disponível entre 90% e 100%, *Adaptive WQR* apresenta um ganho de aproximadamente 40%. À medida que o nível de informação diminui esse ganho é ainda maior, chegando a uma melhoria de pouco mais de 80% com relação ao *makespan* apresentado por *Dynamic FPLTF*. Podemos notar ainda que a heurística *Adaptive WQR* apresenta valores comparáveis com relação à heurística *WQR*, sendo esta última ligeiramente melhor em ambientes com menos informação.



(a) *Makespan* da aplicação



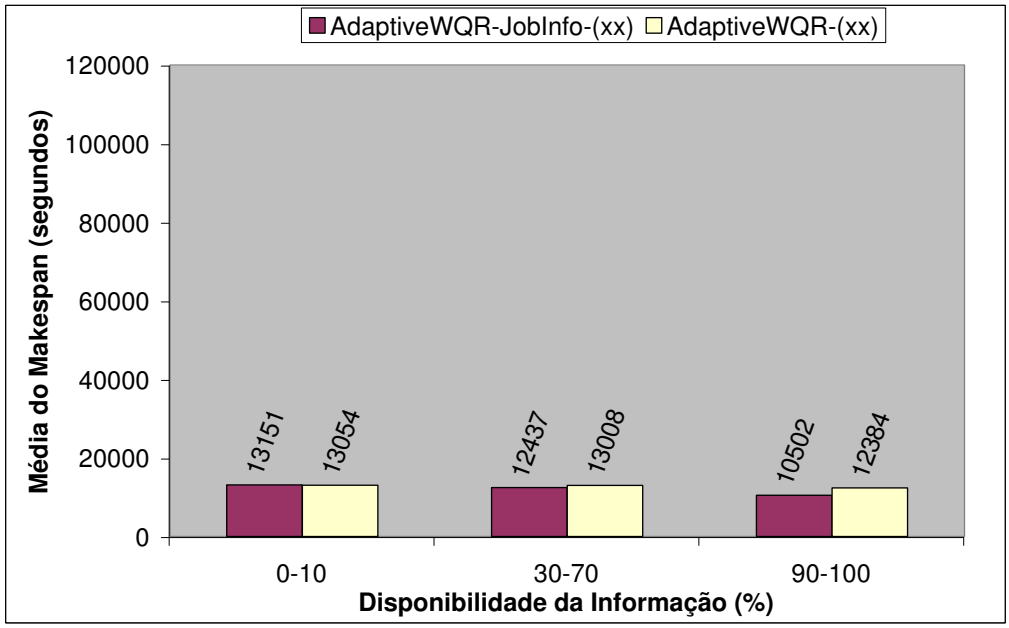
(b) Desperdício de recursos

Figura 5.3: Impacto do nível de informação disponível sobre o grid no desempenho da heurística *Adaptive WQR*

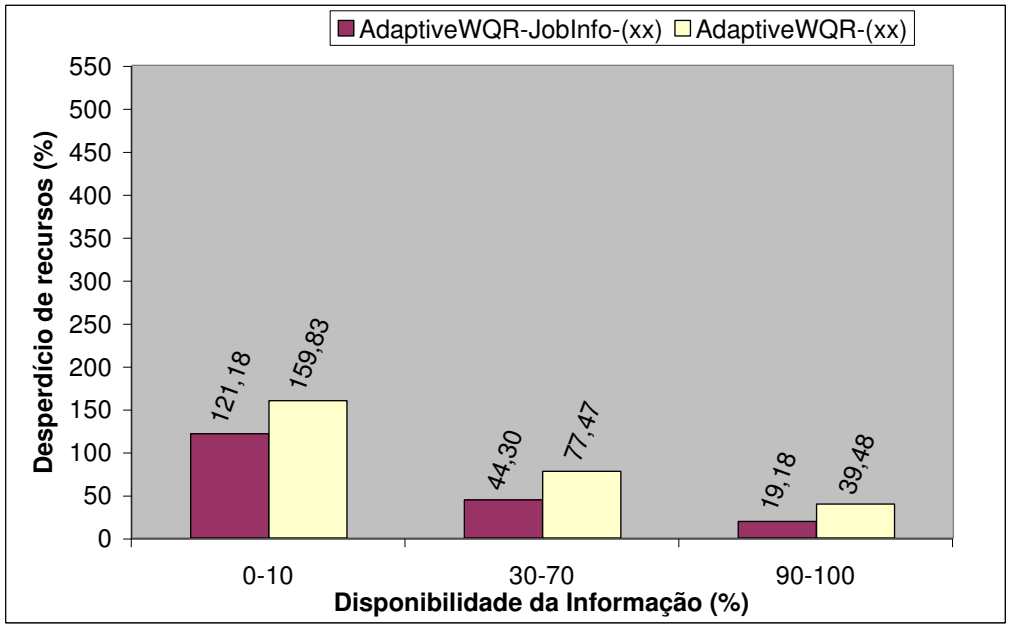
A Figura 5.3(b) apresenta o nível de desperdício de recurso das heurísticas *Adaptive WQR* e *WQR*. Desses resultados podemos observar que enquanto os valores para *WQR* permanecem constante, a heurística *Adaptive WQR* diminui o desperdício de recursos na medida que existe mais informações disponíveis. No menor nível de informação disponível (0% a 10%), *Adaptive WQR* consome aproximadamente 40% menos recursos que WQR. Nos cenários onde nível de informação disponível está entre 30% e 70%, *Adaptive WQR* apresenta uma economia de quase 80%. No maior nível de informação disponível (90% a 100%), o consumo dela chega a ser 10 vezes menor do que *WQR*. Com isso, vemos que a heurística *Adaptive WQR* consegue fazer um bom uso da informação quando disponível e, com isso, economizar mais recursos do que heurísticas de replicação tradicionais, sem comprometer o desempenho da aplicação quanto ao *makespan*.

Para analisar o impacto da falta de informações sobre a aplicação no desempenho da heurística *Adaptive WQR*, executamos também simulações sob os mesmos cenários considerando que não havia informação disponível sobre a aplicação. Na Figura 5.4, temos a comparação do desempenho de *Adaptive WQR* com o uso de informação sobre as tarefas da aplicação (*AdaptiveWQR-JobInfo*) e sem um uso desta (*AdaptiveWQR*).

Analisando os resultados da Figura 5.4, podemos perceber que os valores para o *makespan* são bem próximos nas duas situações dentro da margem de erro das simulações. Desse modo, podemos destacar que mesmo sem informação sobre a aplicação, o desempenho da heurística *Adaptive WQR* não é prejudicado, apresentando ainda um *makespan* comparável com *WQR*. Note ainda que embora o desperdício de recursos quando não há informação sobre a aplicação tenha aumentado um pouco, os valores apresentados por *Adaptive WQR* são bem menores que do que os valores de WQR. Esse aumento discreto é naturalmente explicado pelo fato de não ser possível estimar o tempo de execução das tarefas em cada recurso. Assim, pode ocorrer a seguinte situação: uma tarefa executando em um processador lento pode ser replicada para um processador mais rápido. Porém, o tempo restante para concluir a tarefa rodando no processador lento pode ser menor que o tempo necessário para concluir a sua réplica no outro processador. Com o uso de informação sobre as tarefas da aplicação, é possível realizar estas estimativas e, assim, evitar desperdício de recursos ocasionados por situações como essa.



(a) *Makespan* da aplicação



(b) Desperdício de recursos

Figura 5.4: Impacto da falta de informações sobre a aplicação no desempenho da heurística *Adaptive WQR* em cada nível de informação disponível sobre o grid

Impacto do Nível de Replicação

Nesse ponto avaliamos o impacto do nível de replicação utilizado durante a simulação das heurísticas. Como *Dynamic FPLTF* não utiliza replicação alguma, o desempenho das heurísticas foi comparado apenas com o desempenho de *WQR*. As Tabelas 5.5 e 5.6 exibem a média, em todos cenários, do *makespan* e do desperdício de recursos agrupados por nível de replicação. Os resultados mostram que todas as heurísticas apresentam comportamento semelhante. Na medida que se aumenta o nível de replicação, os valores para o *makespan* melhoram. Isso é explicado pelo fato de um número maior de réplicas proporcionar maiores chances de uma tarefa ser escalonada em um processador mais rápido. Em contrapartida, com o aumento do nível de replicação, as heurísticas passam a consumir mais recursos e, assim, aumentam o nível de desperdício.

Observando ainda esses resultados, podemos destacar que o melhor desempenho, com relação ao *makespan*, dentre todas as heurísticas apresentadas é alcançado pela heurística *Adaptive WQR*. Note que os valores para o *makespan* são comparáveis com os resultados de *WQR* para os níveis de replicação $4x$ e xx, com *Adaptive WQR* chegando a ser melhor no nível de replicação é $2x$. Com relação ao nível de desperdício de recursos, podemos observar que não importa o nível de replicação, *Adaptive WQR* sempre apresenta menores índices de desperdício de recursos, conseguindo economizar, em média, a metade da quantidade consumida por *WQR*. Note ainda que para o maior o nível de replicação (xx), *WQR* consome quase 5 vezes mais do que no menor nível ($2x$) enquanto que *Adaptive WQR* apresenta um crescimento mais discreto, chegando a consumir no máximo 3 vezes mais. Isso demonstra que o critério de replicação adotado pela heurística *Adaptive WQR* permite ter um melhor controle do nível de desperdício de recursos, ao mesmo tempo que mantém o bom desempenho da aplicação quanto ao *makespan*.

Por outro lado, comparando as técnicas *Otimista*, *Pessimista* e *Híbrida*, podemos perceber que apesar delas apresentarem baixos níveis para o desperdício de recursos, elas não atingiram um bom desempenho geral por apresentarem valores extremamente altos com relação ao *makespan*. Com isso, vemos que esses algoritmos não se mostraram eficientes para um ambiente dinâmico e carente de informações como um grid computacional. O principal fator que contribui para isso deve-se ao fato dessas heurísticas dependerem mais da disponibilidade de informações do que *Adaptive WQR*. Desse modo, o aumento do nível de repli-

cação utilizada por essas técnicas demonstra não ser suficiente para se alcançar desempenho comparável com heurísticas de replicação.

	Makespan		
Heurística	**rep. 2x**	**rep. 4x**	**rep. xx**
WQR	20946 s	13679 s	11496 s
Otimista	26941 s	25365 s	24719 s
Pessimista	42180 s	41106 s	40672 s
Híbrida-10	37999 s	36715 s	36170 s
AdaptiveWQR	18327 s	13860 s	12815 s
AdaptiveWQR-JobInfo	17098 s	13654 s	12763 s

Tabela 5.5: Média do *makespan* por nível de replicação para todas as heurísticas

	Desperdício		
Heurística	**rep. 2x**	**rep. 4x**	**rep. xx**
WQR	43%	117%	196%
Otimista	15%	28%	46%
Pessimista	11%	19%	31%
Híbrida-10	9%	13%	24%
AdaptiveWQR	28%	62%	92%
AdaptiveWQR-JobInfo	20%	40%	61%

Tabela 5.6: Média do desperdício de recursos por nível de replicação para todas as heurísticas

Impacto da Granularidade da Aplicação

A seguir investigamos o impacto da granularidade da aplicação no desempenho das heurísticas. Vale lembrar que a variação da granularidade implica na variação da quantidade de tarefas que compõem a aplicação. Dessa forma, também estaremos avaliando o impacto da *relação entre o número de tarefas e recursos* sobre o desempenho do escalonador.

Nas Tabelas 5.7 e 5.8, temos os resultados agrupados por granularidade da aplicação (5000, 25000 e 125000 segundos). Os valores exibidos correspondem à média do *makespan*

e do desperdício de recursos nos cinco níveis de heterogeneidade do grid e da aplicação e nos três níveis de disponibilidade de informações. Como as todas heurísticas apresentaram comportamento semelhante nos três níveis de replicação, nesta tabela consideramos apenas o nível de replicação xx.

	Makespan		
Heurística	**gran. 5000**	**gran. 25000**	**gran. 125000**
Dynamic FPLTF	40902 s	43115 s	44858 s
WQR	6908 s	9037 s	18545 s
Otimista	9837 s	17458 s	46865 s
Pessimista	10792 s	25892 s	85333 s
Híbrida-10	11541 s	24668 s	72303 s
AdaptiveWQR	7014 s	9638 s	21795 s
AdaptiveWQR-JobInfo	6865 s	9512 s	21913 s

Tabela 5.7: Média do *makespan* por granularidade da aplicação para todas as heurísticas

	Desperdício		
Heurística	**gran. 5000**	**gran. 25000**	**gran. 125000**
Dynamic FPLTF	-	-	-
WQR	25%	39%	526%
Otimista	14%	16%	109%
Pessimista	6%	8%	64%
Híbrida-10	9%	13%	24%
AdaptiveWQR	22%	24%	230%
AdaptiveWQR-JobInfo	20%	18%	146%

Tabela 5.8: Média do desperdício de recursos por granularidade da aplicação para todas as heurísticas

De acordo com esses resultados, pode-se notar que independente da heurística usada, uma menor granularidade implica em um melhor desempenho do escalonador. Isto ocorre pois tarefas com um custo computacional menor resultam num maior número de tarefas que

compõem a aplicação (lembrando que a aplicação possui um custo total fixo), proporcionando assim um maior poder de paralelismo para aplicação. Além disso, tarefas maiores podem causar um maior impacto no *makespan* da aplicação caso sejam escalonadas em processadores lentos ou sobrecarregados.

Podemos observar que, de maneira geral, as técnicas *Otimista*, *Pessimista* e *Híbrida* apresentam comportamento semelhante. Quando comparadas com *Dynamic FPLTF*, elas apresentam em média valores melhores para o *makespan*, sendo inferiores apenas quando a relação tarefas por processadores é igual a $0,5$ (granularidade 125000). Para a granularidade 5000, elas alcançam uma melhoria de aproximadamente 70%. No entanto, para a granularidade 125000, o desempenho delas é bastante inferior a *Dynamic FPLTF*. Sendo assim, os resultados indicam que a heurística *Information Adaptive* consegue tolerar melhor a falta de informações do que uma heurística *bin-packing* em granularidades menores. A queda de desempenho devido ao aumento da granularidade e conseqüente redução do número de tarefas pode ser explicada por dois motivos. Primeiro, a técnica *Híbrida* pode errar mais durante a fase de ajuste uma vez que há menos tarefas para realizar a análise do ambiente e descobrir qual técnica é mais adequada. Segundo, tarefas maiores podem causar um maior impacto sobre as técnicas *Otimista* e *Pessimista* quando estas realizam mapeamentos ineficientes devido a eventuais classificações incorretas dos recursos nos grupos *best* e *worst*.

Além disso, note que as três técnicas apresentam desempenho bastante inferior à heurística *WQR*. Embora, elas apresentem uma grande redução do nível de desperdício de recursos, observe que os valores do *makespan* apresentados por elas são piores que os valores apresentados pela heurística *WQR*. Quanto maior a granularidade, maior se torna essa diferença de desempenho. Note que para granularidade 5000, as técnicas *Otimista* e *Pessimista* são piores que *WQR* em aproximadamente 40%. Para a granularidade 125000, os resultados são ainda piores com *Otimista* atingindo um desempenho 150% inferior a *WQR*. E a técnica *Pessimista* chega a apresentar um *makespan* 4 vezes pior.

Por outro lado, mais uma vez é possível destacar o bom desempenho da heurística *Adaptive WQR*, que apresenta valores para o *makespan* comparáveis com *WQR* e menores níveis de desperdício de recursos em todas as granularidades. Sem o uso da informação sobre a aplicação, *Adaptive WQR* consegue reduzir o gasto dos recursos para pouco mais da metade consumida por *WQR* quando a granularidade da aplicação é 25000. A redução no nível

de desperdício de recursos é ainda maior com aplicações de granularidade 125000, onde *Adaptive WQR* consome cerca de 60% a menos que *WQR* mesmo sem informação sobre a aplicação. Com o uso da informação sobre a aplicação, os níveis de desperdício de recursos diminuem ainda mais, chegando a menos da metade para a granularidade 25000. Para aplicações de 125000, o consumo é 3 vezes menor que WQR. Vale salientar ainda que para essa granularidade, o *makespan* de *Adaptive WQR* degrada apenas 16% em relação à *WQR*.

Resultados gerais

A Tabela 5.9 apresenta um sumário dos resultados obtidos em todas as simulações. São exibidos a média e o desvio padrão do *makespan* da aplicação e do nível de desperdício de recursos em todos os cenários. Os resultados demonstram que o melhor desempenho é alcançado pelas heurísticas *WQR* e *Adaptive WQR*. Com relação ao *makespan*, é possível notar que, na média, as heurísticas *WQR* e *Adaptive WQR* apresentam desempenho comparáveis. No entanto, a grande diferença entre elas está no nível de desperdício de recursos. A heurística *Adaptive WQR* consome muito menos recursos que *WQR*, provendo uma significativa redução do desperdício de recursos. Enquanto que *WQR* apresenta índice médio de 118% de desperdício de recursos, a heurística *Adaptive WQR* apresenta um nível de desperdício de apenas 50% em média. Essa grande economia de recursos provida pela heurística *Adaptive WQR* deve-se justamente ao seu critério de replicação diferenciado que leva em consideração o nível de informação disponível no ambiente para decidir quando replicar uma determinada tarefa.

Por outro lado, temos que na média geral, as três técnicas da heurística *Information Adaptive* não apresentaram nenhum comportamento de muito destaque. Embora elas tenham apresentado baixo nível de desperdício de recursos, pode-se perceber que os valores para o *makespan* da aplicação são bastante inferiores a *WQR* e *Adaptive WQR*. No entanto, é importante ressaltar que com relação a *Dynamic FPLTF*, a heurística *Information Adaptive* tolera melhor a falta de informação sobre o grid, apresentando menores valores para a média e o desvio padrão do *makespan* da aplicação.

	Makespan		Desperdício	
Heurística	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
Dynamic FPLTF	42961 s	55112 s	-	-
WQR	15373 s	8526 s	118%	144%
Otimista	25675 s	29758 s	30%	126%
Pessimista	41319 s	54436 s	20%	84%
Híbrida-10	36961 s	45908 s	15%	64%
AdaptiveWQR	15001 s	8924 s	61%	132%
AdaptiveWQR-JobInfo	14505 s	8715 s	40%	106%

Tabela 5.9: Sumário dos resultados obtidos em todas as simulações

Por fim, os resultados mostraram que a estratégia de adaptar heurísticas de replicação consiste numa alternativa de escalonamento de aplicações BoT em grids bastante eficiente, onde o nível de desperdício de recursos pode ser melhor controlado e reduzido, sem prejudicar o desempenho da aplicação.

5.2 Experimentos

No intuito de validar os resultados obtidos nas simulações, conduzimos alguns experimentos em um ambiente real de grid, usando um protótipo da heurística *Adaptive WQR*. O protótipo foi implementado como uma nova heurística de escalonamento de aplicações no software *OurGrid* (CIRNE et al., 2006). O seu desempenho foi comparado com desempenho da heurística *WQR* a qual já possuía uma versão estável implementada no OurGrid.

É importante ressaltar que a heurística *Information Adaptive* não foi implementada por dois motivos. Primeiro, seus resultados não apresentaram nenhum comportamento relevante. Segundo, o *OurGrid* não possuía até o momento nenhuma integração com alguma infraestrutura de monitoração que pudesse fornecer informações dinâmicas sobre a carga dos recursos, necessárias para o funcionamento dessa heurística. Pelo mesmo motivo, também não foi possível implementar a heurística *Dynamic FPLTF*.

Também devido à falta de uma infraestrutura de monitoração, na implementação do pro-

tótipo da heurística *Adaptive WQR* consideramos como conjunto de informações disponíveis apenas a informação estática sobre os recursos do grid. Em particular, consideramos apenas a informação sobre a velocidade dos processadores. Nesse sentido, foi utilizado o valor do *BogoMIPS*[1] como métrica de desempenho para efeitos de comparação entre os processadores.

5.2.1 Cenários

O grid utilizado nos experimentos era composto por 15 máquinas todas localizados no Laboratório de Sistemas Distribuídos da Universidade Federal de Campina Grande. A heterogeneidade do grid era caracterizada pelos diferentes valores do *BogoMIPS* apresentado por cada processador. A Tabela 5.10 lista as máquinas utilizadas nos experimentos e seus respectivos valores de *BogoMIPS*, ordenados do menor para o maior.

Com relação à disponibilidade de informações do grid, consideramos três configurações diferentes: 0%, 50% e 100%. Na primeira, não há informação sobre os processadores. Na segunda, há informação disponível para metade dos processadores. E na terceira, há informação disponível sobre todos os processadores do grid.

A aplicação utilizada para executar os experimentos era composta por tarefas com um custo computacional médio de aproximadamente 5 minutos (300 segundos), onde cada tarefa consistia simplesmente em um *loop* escrito em C (*cpu-intensive*). Para estimar o tempo médio de execução das tarefas, tomamos como base a máquina *pirapema.lsd.ufcg.edu.br* que possuía um BogoMIPS com valor de `3831`. É importante destacar que é difícil estimar o tempo de execução de uma tarefa em um ambiente real com a mesma precisão que um modelo de simulação permite.

Do mesmo modo das simulações, também variamos nos experimentos o número de tarefas que compõem a aplicação no intuito de avaliar o impacto da relação entre número de tarefas e o número de processadores no desempenho do escalonador. Foram executadas três tipos de aplicações, cada uma composta por 40, 15 e 8 tarefas, respectivamente.

[1] Medição da velocidade de um processador realizada pelo Kernel do Linux durante o processo de inicialização. MIPS vem de Million Instructions per Second, ou milhões de instruções por segundo que, no caso, é a unidade de medida de execução de um programa.

Nome da máquina	Valor do BogoMIPS
lula.lsd.ufcg.edu.br	3547
cherne.lsd.ufcg.edu.br	3555
namorado.lsd.ufcg.edu.br	3555
raposa.lsd.ufcg.edu.br	3578
tuvira.lsd.ufcg.edu.br	3578
beta.lsd.ufcg.edu.br	3578
enchova.lsd.ufcg.edu.br	3792
palmito.lsd.ufcg.edu.br	3792
nemo.lsd.ufcg.edu.br	3814
ciobinha.lsd.ufcg.edu.br	5521
atum.lsd.ufcg.edu.br	5537
palhaco.lsd.ufcg.edu.br	5537
bicuda.lsd.ufcg.edu.br	5931
mocinha.lsd.ufcg.edu.br	5931
viola.lsd.ufcg.edu.br	5931

Tabela 5.10: Máquinas utilizadas nos experimentos de validação

Dessa forma, considerando-se os 3 níveis de disponibilidade de informação e os 3 tipos de aplicação, foram avaliados um total de 9 cenários diferentes. Para cada cenário foram executados 10 experimentos, contabilizando assim um total de 90 experimentos. Cada experimento consistia na execução *back-to-back* das duas heurísticas com o objetivo de minimizar os efeitos do dinamismo do grid. Vale salientar ainda que o nível de replicação utilizado nos experimentos foi $3x$.

De acordo com os valores da média e do desvio padrão obtidos nos experimentos, foi possível constatar que os resultados apresentavam 95% de intervalo de confiança com uma margem de erro de 10% para mais ou para menos.

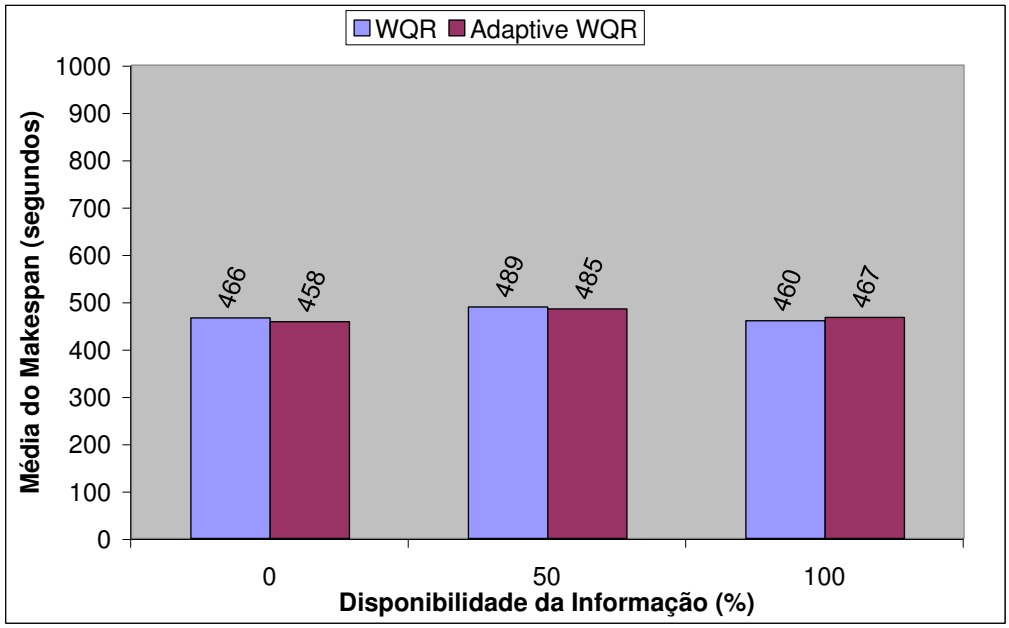
5.2.2 Resultados

A Figura 5.5 apresenta a média do desempenho das heurísticas *Adaptive WQR* e *WQR* para cada nível de informação disponível. Os resultados mostram que as duas heurísticas apresentam o mesmo comportamento geral apontado nas simulações. Podemos observar que ambas apresentam valores comparáveis com relação ao *makespan*. Note ainda que *Adaptive WQR* diminui o nível de desperdício de recursos na medida que há mais informações disponíveis no ambiente. Diferentemente, *WQR* apresenta valores constantes com relação ao desperdício de recursos independente da quantidade de informações disponíveis. Nos cenários onde toda a informação sobre os processadores está disponível (100%), a heurística *Adaptive WQR* chega a desperdiçar em média 40% menos recursos que *WQR*.

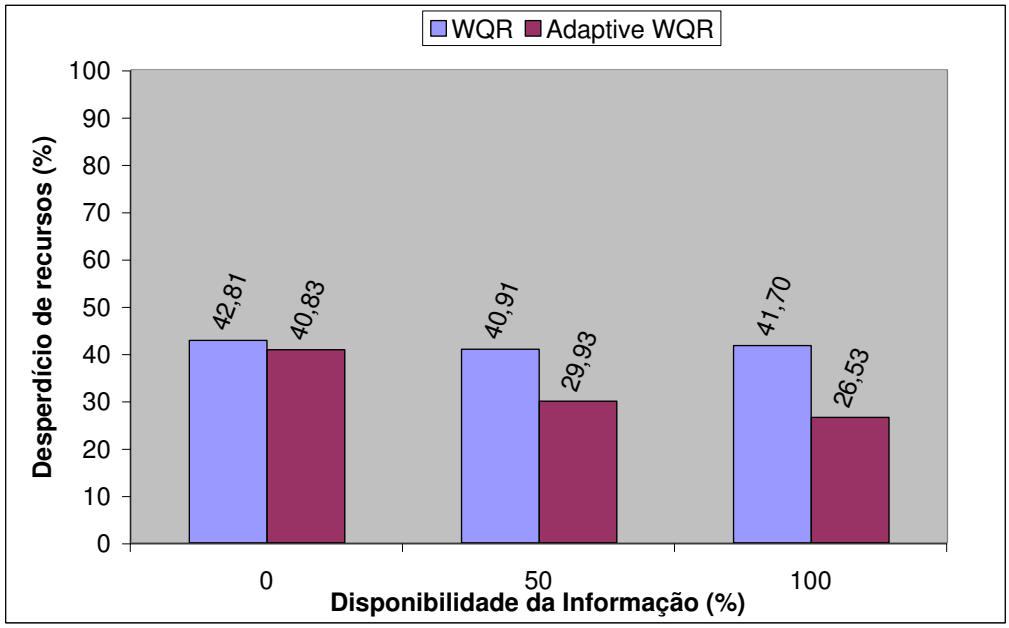
Na Tabela 5.11, apresentamos um sumário dos resultados obtidos em todos os experimentos. Mais uma vez podemos observar que as heurísticas apresentam o mesmo comportamento obtido nas simulações. Porém, note que, na média geral, o ganho em economia de recursos da heurística *Adaptive WQR* é menor nos experimentos do que nas simulações (ver Tabela 5.9). Isso ocorre pelo fato de nos experimentos ter sido utilizada apenas uma informação estática — *BogoMIPS* — sobre os processadores durante as decisões de escalonamento. Sabendo claramente que o desempenho de um processador também depende da informação sobre sua carga de processamento disponível em determinado momento, acreditamos que o nível de desperdício de recursos poderia ser ainda mais reduzido caso fosse utilizada também essa informação.

	Makespan		**Desperdício**	
Heurística	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
WQR	472 s	259 s	41%	39%
Adaptive WQR	470 s	258 s	32%	30%

Tabela 5.11: Sumário do resultados obtidos nos experimentos



(a) *Makespan* da aplicação



(b) Desperdício de recursos

Figura 5.5: Média do desempenho das heurísticas *Adaptive WQR* e *WQR* por nível de informação disponível em ambiente real

5.3 Considerações finais

Nesse capítulo foram apresentados e discutidos os resultados da avaliação de desempenho das duas heurísticas adaptativas propostas neste trabalho: *Adaptive WQR* e *Information Adaptive*. Elas foram comparadas com as heurísticas WQR (de replicação) e *Dynamic FPLTF* (*bin-packing*). A avaliação de desempenho tomou como base a execução de simulações e experimentos, onde foram comparados os valores do *makespan* da aplicação e do nível de desperdício de recursos.

Primeiramente, foram simulados um total de 9000 cenários que variavam nos seguintes aspectos: i) *heterogeneidade dos recursos do grid*; ii) *heterogeneidade e granularidade da aplicação*; iii) *relação entre o número de tarefas e recursos* e iv) *disponibilidade e erro das informações sobre o ambiente*. Analisando os resultados obtidos nas simulações foi possível constatar que a heurística *Adaptive WQR* apresentou um melhor desempenho uma vez que conseguiu reduzir o nível de desperdício de recursos, sem degradar o desempenho da aplicação quanto ao *makespan*.

Por fim, foram executados alguns experimentos com o objetivo de validar os resultados das simulações. Um protótipo da heurística *Adaptive WQR* foi implementado no software *OurGrid* e o seu desempenho foi comparado com a heurística WQR. Os resultados obtidos mostraram que as heurísticas apresentaram o mesmo comportamento geral apontado nas simulações.

Capítulo 6

Conclusões

As principais heurísticas existentes para o escalonamento de aplicações BoT em grids computacionais podem ser classificadas em duas abordagens. De um lado, temos uma abordagem que se baseia em informações sobre o grid e a aplicação para realizar as decisões de escalonamento (heurísticas *bin-packing*). Do outro lado, temos uma abordagem baseada na replicação de tarefas, que não utiliza nenhuma informação (heurísticas de replicação). A primeira não funciona bem quando falta informação, enquanto que a segunda apresenta um grande desperdício de recursos em alguns casos.

Neste trabalho, foi proposta uma nova abordagem de escalonamento de aplicações BoT em grids computacionais, cuja principal característica consiste em ser adaptativa quanto à disponibilidade de informações sobre o ambiente. Em particular, foram propostas e avaliadas duas novas heurísticas de escalonamento denominadas *Information Adaptive* e *Adaptive WQR*. A primeira segue uma estratégia inspirada nas heurísticas *bin-packing*, ou seja, consiste em uma tentativa de adaptar as heurísticas *bin-packing* para lidar com a falta de informações. A segunda baseia-se nas heurísticas de replicação na tentativa de minimizar o desperdício de recursos através do uso da informação que esteja disponível. O desempenho dessas novas heurísticas foi comparado com as heurísticas *Dynamic FPLTF* e *WQR*.

A partir dos resultados apresentados foi possível constatar que a heurística *Dynamic FPLTF* de fato degrada o seu desempenho sensivelmente à medida que há menos informação disponível no ambiente. Por outro lado, a heurística *WQR* apresentou desempenho constante independente do nível de informação disponível. Ela apresentou bons resultados com relação ao *makespan*, porém esse bom desempenho é alcançado ao custo de um consumo extra

de recursos, apresentando assim um nível de desperdício de recursos muito elevado.

Com relação à heurística *Information Adaptive* percebemos que, no geral, ela conseguiu lidar melhor com a falta de informação do que a heurística *bin-packing Dynamic FPLTF*. No entanto, seu desempenho foi bastante inferior às demais heurísticas. O seu péssimo desempenho pode ser creditado ao fato dela tentar "adivinhar" de maneira empírica o desempenho de recursos dos quais não se tem informação. Os resultados gerais mostraram que a estratégia de adaptar heurísticas *bin-packing* para lidar com a falta de informações foi bastante ineficaz.

Por outro lado, a heurística *Adaptive WQR* mostrou-se bastante eficiente em grande parte dos cenários analisados, alcançando um desempenho comparável à heurística *WQR* e reduzindo o nível de desperdício de recursos consideravelmente. Os resultados mostraram ainda que *Adaptive WQR* apresenta um bom desempenho mesmo com a falta de informação sobre a aplicação, diferentemente de *Information Adaptive* que depende dessa informação para funcionar corretamente. Nesse sentido, concluímos que a estratégia de utilizar informação disponível para controlar melhor o processo de replicação de tarefas proporcionou uma maior eficiência na execução de aplicações BoT ao reduzir o nível de desperdício de recursos, sem prejudicar o desempenho da aplicação.

Por fim, os resultados mostraram que a nova abordagem adaptativa demonstrou ser mais adequada para lidar com o dinamismo e heterogeneidade de um ambiente de grid — onde informações podem ou não estar disponíveis — do que as abordagens *bin-packing* e de replicação.

6.1 Trabalhos Futuros

Nesta seção são apresentadas algumas propostas de trabalhos futuros que podem dar continuidade aos estudos realizados neste trabalho. Como possíveis trabalhos futuros, propomos os seguintes tópicos:

- **Definir e avaliar uma heurística adaptativa que possa tratar também de aplicações BoT que processam grandes quantidades de dados (*data-intensive*)**: Através do nosso modelo já é possível modelar esse tipo de aplicação, o que já facilitaria a investigação e análise de uma nova heurística que levasse em consideração informação

sobre o tamanho dos dados. Essa nova heurística poderia estender as características e funcionalidades da heurística *Storage Affinity* (SANTOS-NETO et al., 2004), fazendo com que o uso de informação disponível possa oferecer um maior controle do nível de desperdício de recursos, além de proporcionar uma melhoria na técnica de reaproveitamento de dados.

- **Estudo e avaliação de uma meta-escalonador que implemente as várias abordagens de heurísticas**: Esse escalonador possuiria uma coleção de heurísticas e seria capaz de identificar em tempo de execução qual tipo de heurística é mais apropriada para determinado ambiente de grid. Essa linha de investigação teria o objetivo de estudar e avaliar técnicas eficientes para detectar e identificar se determinada configuração do grid é mais favorável para execução de uma heurística *bin-packing*, de replicação ou adaptativa. Isso seria feito de forma dinâmica de acordo com as mudanças nas condições do ambiente de grid, ou seja, o meta-escalonador poderia mudar de heurística dinamicamente.

- **Descoberta de informações durante a execução da aplicação**: Esta linha de investigação teria objetivo de estudar e avaliar técnicas eficientes de como descobrir informações sobre recursos *sem informação* durante a execução da própria aplicação. Uma possível forma de descoberta dessas informações seria aplicar métodos estatísticos sobre dados obtidos de execuções de aplicações passadas (MITZENMACHER, 2000).

- **Melhoramento da técnica *Híbrida***: Investigar maneiras mais eficientes de se aplicar a técnica *Híbrida* em um ambiente de grid, ou seja, torná-la mais híbrida de fato. Uma possível modificação seria fazer com que a decisão de qual heurística aplicar fosse diferenciada para cada site ao invés de ser uma única escolha para o todo o grid. Isso poderia trazer resultados mais eficientes uma vez que determinados sites podem ser ambientes mais favoráveis para a execução da técnica *Otimista*, enquanto que outros sites podem ser mais adequados para a técnica *Pessimista*.

- **Implementação de um protótipo da heurística *Adaptive WQR* integrado com uma infraestrutura de monitoração**: Esta linha de investigação teria objetivo de estudar e avaliar a possibilidade de integrar alguma infraestrutura de monitoração de recursos

com a heurística *Adaptive WQR*. Acreditamos que o desperdício de recursos poderia ser reduzido ainda mais com a utilização de tal infraestrutura.

Bibliografia

AHRNAD, I. Resource Management in Parallel and Distributed Systems with Dynamic Scheduling. *Concurrency: Practice and Experience*, v. 7, p. 587–590, October 1995.

ANDRADE, N. et al. Ourgrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In: *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*. [S.l.: s.n.], 2003.

ASSIS, L. et al. Uma heurística de particionamento de carga divisível para grids computacionais. In: *Anais do 24º Simpósio Brasileiro de Redes de Computadores (SBRC'2006)*. Curitiba - PR - Brasil: [s.n.], 2006.

BARATLOO, A. et al. Charlotte: Metacomputing on the Web. In: *Proocedings of the 9th Conference on Parallel and distributed Computing Systems*. [S.l.: s.n.], 1996.

BASU, S. et al. NodeWiz: Peer-to-peer Resource Discovery for Grids. In: *Proceedings of IEEE/ACM GP2PC 2005*. Cardiff, UK: [s.n.], 2005.

BRAUN, T. et al. A Comparion of Eleven Static Heuristics for Mapping Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, v. 61, p. 810–837, 2001.

CASANOVA, H. Network Modeling Issues For Grid Application Scheduling. *International Journal of Foundations of Computer Science*, v. 6, n. 2, p. 145–162, 2005.

CASANOVA, H.; HAYES, J.; YANG, Y. Algorithms and Software to Schedule and Deploy Independent Tasks in Grid Environments. In: *Workshop on Distributed Computing, Metacomputing and Resource Globalization*. France: [s.n.], 2002.

CASANOVA, H. et al. Heuristics for Scheduling Parameter Sweep Applications in Grid environments. In: *Proceedings of the 9th Heterogeneous Computing Workshop*. [S.l.: s.n.], 2000.

CASANOVA, H.; MARCHAL, L. *A Network Model for Simulation of Grid Application*. Technical Report 2002-40, Laboratoire d'Informatique du Parallélisme, ENS-Lyon, France, October 2002.

CIRNE, W. Grids Computacionais: Arquiteturas, Tecnologias e Aplicações. In: *Terceiro Workshop em Sistemas Computacionais de Alto Desempenho*. [S.l.: s.n.], 2002.

CIRNE, W. et al. Labs of the World, Unite!!! *Accepted for publication in Journal of Grid Computing*, 2006.

CIRNE, W. et al. *On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems*. Technical Report 09/2005, Universidade Federal de Campina Grande, Departamento de Sistemas e Computação, UFCG/DSC, 2005.

CZAJKOWSKI, K. et al. Grid Information Services for Distributed Resource Sharing. In: *Proceeding of the 10th IEEE Symposium on High-Performance Distributing Computing*. [S.l.: s.n.], 2001.

DAVIDSON, S.; GARCIA-MOLINA, H.; SKEEN, D. Consistency in Partitioned Networks. *ACM Computing Surveys*, v. 7, p. 341–370, September 1985.

DEVORE, J. L. *Probability and Statistics for Engineering and The Sciences*. 2nd. ed. [S.l.]: John Wiley and Sons, 2000.

FEITELSON, D.; RUDOLPH, L. Metrics and Benchmarking for Parallel Job Scheduling. In: *Proceedings of 4th Job Scheduling Strategies for Parallel Processing (JSSPP'1998)*. [S.l.: s.n.], 1998.

FEITELSON, D. G. Metric and Workload Effects on Computer Systems Evaluation. *Computer*, v. 36, n. 9, p. 18–25, September 2003.

FOSTER, I.; KESSELMAN, C. *The Grid: Blueprint for a new Computing Infrastructure*. San Francisco, USA: Morgan Kaufmann Publishers, Inc., 1999.

FOSTER, I.; KESSELMAN, C. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, v. 15, n. 3, 2001.

FRANCIS, P. et al. An Architecture for Global Internet Host Distance Estimation Service. In: *Proceedings of IEEE INFOCOM*. [S.l.: s.n.], 1999.

FUJIMOTO, N.; HAGIHARA, K. Near-optimal Dynamic Scheduling of Independente Coarse-grained Tasks onto a Computational Grid. In: *32nd Anual International Conference on Parallel Processing (ICPP-03)*. Taiwan: [s.n.], 2003. p. 391–398.

GHARE, G.; LEUTENEGGER, S. Improving Speedup and Response Times by Replication Parallel Programs on a Snow. In: *10th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'2004)*. [S.l.: s.n.], 2004.

GIERSCH, A.; ROBERT, Y.; VIVIEN, F. Scheduling tasks sharing files on heterogeneous master-slave platforms. In: *Proceedings of 12th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP'2004)*. [S.l.: s.n.], 2004. p. 364–371.

IBARRA, O. H.; KIM, C. E. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM (JACM)*, v. 24, n. 2, p. 280–289, 1977.

JAMES, H.; HAWICK, K.; CODDINGTON, P. *Scheduling Independente Tasks on Metacomputing Systems*. The University of Adelaide. DHCP-066, 1999.

LEE, C. et al. Are user runtime estimates inherently inaccurate? In: *Proceedings of 10th Job Scheduling Strategies for Parallel Processing (JSSPP'2004)*. [S.l.: s.n.], 2004.

LEE, S. J. et al. Measuring Bandwidth between Planetlab Nodes. In: *Proceedings of PAM 2005*. Boston, MA: [s.n.], 2005. p. 292–305.

LEGRAND, A.; MARCHAL, L.; CASANOVA, H. Scheduling Distributed Applications: the Simgrid Simulation Framework. In: *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID´03)*. Tokyo, Japan: [s.n.], 2003.

MAHESWARAN, M. et al. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In: *Proceedings of the 9th Heterogeneous Computing Workshop*. [S.l.: s.n.], 2000.

MENASCé, D. et al. Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures. *Journal of Parallel and Distributed Computing*, p. 1–18, 1995.

MITZENMACHER, M. How Useful Is Old Information. *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n. 1, p. 6–20, January 2000.

NÓBREGA-JÚNIOR, N. et al. *A Simple and Accurate Dynamic Network Model for Grid Simulation*. Technical Report 2005-001, Failure Spotter Project, HP/LSD-UFCG, Brazil, August 2005.

PARANHOS, D.; CIRNE, W.; BRASILEIRO, F. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In: *Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing*. [S.l.: s.n.], 2003. p. 169–180.

PINEDO, M. *Scheduling: Theory, Algorithms and Systems*. 2nd. ed. New Jersey, USA: Prentice hall, 2001.

SANTOS-NETO, E. et al. Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids. In: *Proceedings of 10th Job Scheduling Strategies for Parallel Processing (JSSPP'2004)*. [S.l.: s.n.], 2004.

SMALLEN, S.; CIRNE, W.; FREY, J. Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience. In: *Heterogeneous Computing Workshop*. [S.l.: s.n.], 2000.

TIRUMALA, A. et al. *Iperf: the TCP/UDP bandwidth measurement tool*. 2004. Http://dast.nlanr.net/Projects/Iperf.

WOLSKI, R. Dynamically Forecasting Network Performance Using the Network Weather Service. *Journal of Cluster Computing*, v. 1, p. 119–132, January 1998.

WOLSKI, R. Experiences with Predicting Resource Performance On-line in Computational Grids. In: *ACM SIGMETRICS Performance Evaluation Review*. [S.l.: s.n.], 2003. v. 30, n. 4, p. 41–49.

WOLSKI, R.; SPRING, N.; HAYES, J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, v. 15, n. 5-6, p. 757–768, October 1999.

WU, M.; SUN, X.-H. A General Self-adaptive Task Scheduling System for Non-dedicated Heterogeneous Computing. In: *Proceedings of IEEE Cluster Computing Conference*. Hong Kong: [s.n.], 2003.