Escalonamento Bi-objetivo de Aplicações Paralelas em Recursos Heterogêneos

Idalmis Milián Sardiña[1], **Cristina Boeres**[1], **Lúcia Drummond**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (IC-UFF)
Rua Passo da Pátria, 156 - São Domingos - 24.210-240 - Niterói - RJ-Brasil

{isardina, boeres, lucia}@ic.uff.br

Abstract. *This paper proposes a static scheduling strategy for heterogeneous distributed systems that not only aims to minimize the makespan, but also maximizes the reliability of the application. In order to solve this problem, a weighted function that includes both objectives has been considered. This work describes an evaluation methodology as well as proposals to adjust the relative weights of the two objectives with the aim of achieving an efficient schedule for the application in accordance with the user's needs. In comparison with related work, the results of the experiments carried out highlight not only the importance of using this flexible strategy for systems of heterogeneous resources but also the benefits of adopting the weighted function to guide the bi-objective scheduling decisions in conjunction with the proposed methodology to adjust the weights.*

Resumo. *Neste trabalho, é proposta uma estratégia de escalonamento estático para sistemas distribuídos heterogêneos que além de minimizar o makespan, maximiza a confiabilidade da aplicação. Para resolver este problema, uma função de custo ponderada incorpora ambos objetivos simultaneamente. Este trabalho também descreve uma metodologia de análise e especifica propostas relativas ao ajuste de pesos dos objetivos, para que um bom escalonamento seja definido de acordo com a necessidade do usuário. Os experimentos realizados destacam a importância de usar uma estratégia flexível em ambientes com recursos heterogêneos. Os resultados mostram que tanto a função ponderada que guia as decisões do escalonamento bi-objetivo quanto a metodologia de ajuste são favoráveis, quando comparados com outros trabalhos correlatos.*

1. Introdução

O uso cada vez maior de sistemas heterogêneos de larga escala para executar aplicações que necessitam de um alto poder computacional tem levado a um estudo aprofundado das características destes sistemas e das próprias aplicações a serem executadas. Estudos realizados como em [Nascimento et al. 2007] mostram que é possível alcançar alto desempenho ao executar aplicações paralelas em sistemas como grades computacionais.

Um aspecto marcante em sistemas distribuídos é a propensão à ocorrência de falhas em recursos. [Reed et al. 2006] destaca a importância deste problema e realiza uma análise estatística sobre a confiabilidade dos sistemas computacionais atuais. Neste mesmo trabalho é discutida a necessidade de novas abordagens adaptativas que aumentem a confiabilidade e ainda, tratem de falhas de tal forma que se permita uma execução satisfatória das aplicações.

Vários trabalhos em escalonamento de aplicações em ambientes distribuídos apresentam diferentes formas de melhorar o tempo de execução das aplicações, incluindo aspectos de confiabilidade [Qin et al. 2002, Dogan and Ozguner 2005, Hakem and Butelle 2007, Dongarra et al. 2007]. Mesmo assim, a maioria dos algoritmos são baseados em modelos que não consideram certas características que são marcantes tanto em relação à aplicação quanto ao sistema distribuído heterogêneo.

Este trabalho propõe uma estratégia de escalonamento bi-objetivo ponderada de uma aplicação paralela, cujas tarefas obedecem restrições de precedência e executam sobre uma arquitetura com processadores heterogêneos propensos a falhas. A estratégia considera a confiabilidade além do tempo de execução como objetivos, através de uma heurística *list scheduling* baseada em [Qin et al. 2002]. No entanto, ao invés de considerar estes dois objetivos de forma hierárquica, como em [Qin et al. 2002], o trabalho propõe uma função de custo ponderada que incorpora os objetivos simultaneamente. Ainda, uma metodologia de análise flexível é proposta para ajustar os pesos e achar um compromisso entre a maximização da confiabilidade e a minimização do tempo de execução.

2. Trabalhos Relacionados

Quando lida-se com ambientes heterogêneos, além do tempo de execução a ser minimizado alguns trabalhos da literatura como [Qin et al. 2002, Dogan and Ozguner 2005, Dongarra et al. 2007, Hakem and Butelle 2007] também atacam o problema de confiabilidade no sistema, especificando em suas abordagens uma função com ambos objetivos.

Trabalhos de escalonamento bi-objetivo como [Dogan and Ozguner 2005] e [Hakem and Butelle 2007] usam funções de custo ponderadas, mas não avaliam a importância da ponderação em ambientes distribuídos heterogêneos. Além do mais, os algoritmos não respondem a questão de como achar um determinado compromisso entre minimização do tempo de execução e a maximização da confiabilidade, de forma a indicar uma solução de escalonamento conveniente ao usuário. Algumas questões importantes como estas, ainda não exploradas na literatura, são apontadas em [Dongarra et al. 2007].

Em geral, os algoritmos em [Qin et al. 2002, Dogan and Ozguner 2005, Dongarra et al. 2007, Hakem and Butelle 2007] não foram projetados especificamente para ambiente heterogêneos de larga escala. Alguns como [Qin et al. 2002] foram desenvolvidos para sistemas de tempo real. Neste caso, o algoritmo de escalonamento ordena as tarefas por *deadline* ao considerar aplicações em sistemas com restrições de tempo. Os objetivos são manipulados de forma hierárquica, ou seja, são escolhidos primeiramente os processadores que maximizam a confiabilidade, e posteriormente, os que produzem o menor tempo de execução. Isto favorece mais a confiabilidade em relação ao tempo de execução, resultado que pode não ser conveniente para ambientes distribuídos.

[Dogan and Ozguner 2005] especifica um algoritmo de escalonamento bi-objetivo com uma função ponderada que considera o compromisso entre a minimização do tempo de execução e a maximização da confiabilidade. No entanto, para tal, primeiro calcula-se cada termo referente a cada objetivo separadamente para cada par tarefa-recurso, gerando-se duas listas ordenadas. Uma função ponderada com dados obtidos destas listas é utilizada para calcular o custo final associado. Este trabalho mostra as vantagens desta abordagem em relação ao algoritmo bi-objetivo proposto anteriormente pelo mesmo autor [Dogan and Ozguner 2002], que introduz uma função de custo sem pesos não normali-

zada, mas que combina confiabilidade e tempo de execução.

[Dongarra et al. 2007] mostra que otimizar tanto o tempo de execução quanto a confiabilidade, não é um problema aproximável a um fator constante. Neste artigo, um algoritmo de aproximação para tarefas independentes e unitárias é proposto. Para obter um melhor compromisso entre os dois objetivos, o produto entre a taxa de falha do processador e o peso de computação da tarefa é especificado como critério para alocar tarefas não unitárias sobre processadores uniformes. Assim, a função de custo é baseada neste produto e pode ser usada, segundo [Dongarra et al. 2007], para incluir o conhecimento de confiabilidade em algoritmos *list scheduling* que só consideram a minimização do tempo de execução.

O trabalho [Hakem and Butelle 2007] é outra abordagem bi-objetivo, que diferente de [Dogan and Ozguner 2005], adiciona parâmetros de ponderação diretamente a seus objetivos em uma função de custo integrada. No entanto, não há um estudo nem uma sugestão sobre a escolha de determinado peso, para ser utilizado na função que leve a um bom escalonamento. Seu algoritmo é comparado, com um algoritmo que não apresenta função ponderada ([Dogan and Ozguner 2002]) e apenas em relação a complexidade. [Hakem and Butelle 2007] é único trabalho relacionado que atribui pesos diretamente aos objetivos, mas utiliza um operador e uma normalização diferentes ao agregar os termos.

No trabalho proposto neste artigo, o algoritmo de escalonamento emprega uma heurística do tipo *list scheduling* com uma função de custo diferente de [Qin et al. 2002, Dogan and Ozguner 2005, Dongarra et al. 2007, Hakem and Butelle 2007]. A função proposta é ponderada, o que permite ao usuário especificar distintas preferências em relação aos objetivos do escalonamento antes de executar sua aplicação. Além do mais, neste trabalho é apresentado um estudo que mostra a vantagem da função proposta. Mais especificamente, uma metodologia de análise é descrita que auxilia no ajuste dos pesos. A partir de uma métrica aqui proposta e conceitos de dominância, a metodologia contribui para a escolha de um determinado escalonamento. Note que nos trabalhos correlatos, aspectos como estes não são considerados. A abordagem proposta responde a questão de como achar determinada solução de escalonamento que represente um compromisso conveniente entre tempo de execução e confiabilidade. A mesma pode ser aplicada aos trabalhos já existentes, desde que sejam introduzidas informações adicionais nos algoritmos de escalonamento, em relação aos objetivos.

3. Modelo de Escalonamento para Ambientes Distribuídos com Falhas

Com o intuito de representar as características relevantes das aplicações e da arquitetura propensa a falha, os modelos da aplicação, arquitetural e de confiabilidade são especificados a seguir.

3.1. Modelagem da Aplicação

Este trabalho considera aplicações paralelas modeladas por *Grafos Acíclicos Direcionados* (GAD), onde os nós do grafo representam as tarefas da aplicação e os arcos, a precedência entre estas. Um GAD $G = (V, E, e, c)$ modela uma aplicação paralela sendo que V é o conjunto de vértices que representam as tarefas, E a relação de precedência, $e(v_i)$ o peso de execução associado com cada tarefa v_i de V e $c(v_i, v_j)$ o peso de comunicação

associado com o arco (v_i, v_j) de E. As relações de precedência das tarefas definem o GAD, considerando um sistema de precedência com $(v_i, v_j) \in V$, ou seja, a execução de v_j não pode ser iniciada enquanto não seja completada a execução de v_i e os dados de v_i para v_j sejam recebidos por este. Os conjuntos de predecessores e sucessores imediatos da tarefa v_i são denotados $Pred(v_i)$ e $Succ(v_i)$, respectivamente.

3.2. Modelo da Arquitetura

No modelo deste trabalho, $P = \{p_0, p_1, ..., p_{m-1}\}$ é o conjunto de m processadores heterogêneos, sendo que a cada p_j é associado o índice de retardo (*computational slowdow index*), denotado por $csi(p_j)$, conforme identificado em [Nascimento et al. 2007], sendo esta métrica inversamente proporcional ao poder computacional de p_j. Desta forma, o tempo de execução da tarefa v no processador p_j é dado por $eh(v, p_j) = e(v) \times csi(p_j)$. Para duas tarefas adjacentes v_i e v_j alocadas em processadores distintos p_l e p_k, respectivamente, o custo associado à comunicação de $c(v_i, v_j)$ dados é definido como $ch(v_i, v_j) = c(v_i, v_j) \times L(p_l, p_k)$, onde a latência $L(p_l, p_k)$ é o tempo de transmissão por *byte* sobre o *link* (p_l, p_k).

3.3. Modelo de Confiabilidade

A confiabilidade de um sistema pode ser definida como a probabilidade que o sistema tem de realizar com sucesso sua função durante um certo período de tempo e sob condições definidas. Uma alta confiabilidade pode ser entendido como uma alta probabilidade do sistema não falhar. Neste modelo são consideradas somente falhas de processador. Assume-se que estas falhas são independentes entre si, e acontecem de acordo com uma distribuição de *Poisson* com taxa de falha $FP(p_j) \ \ \forall p_j \in P$, com valor constante, conforme definido em [Qin et al. 2002]. A taxa de falha de um processador p_j representa a quantidade de falhas por unidade de tempo que podem ocorrer em p_j.

O custo de confiabilidade $RC(v, p_j)$ de execução da tarefa v em um processador p_j é definido como $RC(v, p_j) = FP(p_j) \times eh(v, p_j)$. A confiabilidade do escalonamento da tarefa em um processador é medida pelo custo de confiabilidade. Portanto, $RC(v, p_j)$ deve ser minimizado para que a confiabilidade se aproxime de 1. Desta maneira, o custo de confiabilidade de um processador baseado em um dado escalonamento de tarefas pode ser calculado como a soma dos custos de confiabilidade de suas tarefas. Assim, sendo $ltask(p_j)$ o conjunto de tarefas alocadas a p_j, o custo de confiabilidade associado à p_j é $RC_p(p_j) = \sum_{\forall v \in ltask(p_j)} RC(v, p_j)$. Para um sistema heterogêneo com um total de m processadores, o custo de confiabilidade de escalonar uma aplicação nos processadores de P, pode ser definido como, $RC_s(G, P) = \sum_{\forall p_j \in P} RC_p(p_j)$.

Assim, a confiabilidade total da aplicação G escalonada em P é dada por $RT = e^{-RC_s(G,P)}$. Com base neste modelo, escalonar as tarefas mais críticas, ou seja com maiores pesos de computação e comunicação, em processadores mais confiáveis é uma forma de maximizar a confiabilidade total RT da aplicação no sistema alvo.

4. Estratégia de Escalonamento Proposta

A função de custo proposta neste trabalho é ponderada, oferecendo ao usuário a possibilidade de especificar suas preferências em relação ao tempo de execução da aplicação e a confiabilidade. Ainda, uma metodologia de análise de pesos é proposta, para ajudar

o usuário a encontrar um compromisso conveniente entre tempo de execução e confiabilidade. Para isto, é definida uma métrica denotada Diferença Média $D(E_k)$, associada a um determinado escalonamento E_k e introduzida no algoritmo proposto para estabelecer uma relação do desequilíbrio entre os objetivos do problema. Com a execução da heurística para diferentes pesos, diversos escalonamentos são gerados com suas respectivas diferenças médias. A partir destas diferenças, uma metodologia é proposta para auxiliar na escolha do melhor escalonamento que atende ao compromisso desejado.

O algoritmo de escalonamento estático proposto é chamado de *Makespan and Realiability Cost Driven* (MRCD) e divide-se em duas etapas principais: ordenação das tarefas segundo um critério de prioridade, e escalonamento da tarefa de maior prioridade em um processador que otimiza a função de custo ponderada.

Observe que para definir o tempo de execução da aplicação (*makespan*), o tempo de fim de cada tarefa v_i no processador p_j, deve ser calculado da forma $EFT(v_i, p_j(v)) = EST(v_i, p_j(v))+eh(v, p_j(v))$. Para computar o $EFT(v_i, p_j)$ de v_i, todas as predecessoras imediatas de v_i devem ter sido escalonadas. Depois de todas as tarefas da aplicação serem escalonadas, o *makespan* do escalonamento E_k pode ser definido como $ms(E_k) = MAX\{EFT(v_i, p_j)\}, \forall v_i \in V$ e $\forall p_j \in P$.

Na primeira etapa de MRCD, as tarefas são ordenadas por seus *b-levels* (*static bottom level*), conforme [Topcuouglu et al. 2002], denotado por $blevel(v), \ \forall v \in V$. Em heurísticas *list scheduling* sobre ambientes heterogêneos, a escolha da tarefa usando *b-level* tem mostrado melhor desempenho para um número maior de GADs [Topcuouglu et al. 2002].

Na segunda etapa, usando um *list scheduling* para escalonar as tarefas, a lista ordenada por $blevel(v)$ é percorrida e, para cada tarefa v da lista é efetuada uma procura pelo melhor processador de acordo com os seguintes critérios. Seja $v \in V$ a próxima tarefa a ser escalonada, o melhor processador $p_j \in P$ a ser escolhido para a execução de v é aquele que minimiza a função de custo $f(v, p_j) = \alpha_1 EFT(v, p_j)+\alpha_2 RC(v, p_j)$. Ou seja, é escolhido o processador que satisfaz $F(v, p) = min_{\forall p_j \in P}\{f(v, p_j)\}$. De acordo com a função $f(v, p_j)$, tarefas mais críticas em relação ao seu tempo de execução e comunicação tendem a ser escalonadas em processadores mais confiáveis e mais rápidos, aumentando o desempenho e a probabilidade da aplicação não falhar.

Note que, para calcular o tempo de início $EST(v, p_j)$ da tarefa v no processador p_j é considerada uma política de inserção de tarefas em espaços ociosos de processador conforme [Topcuouglu et al. 2002]. O peso α_i, associado a cada critério, é uma variável $0 \leq \alpha_i \leq 1$ que representa o nível de importância de cada critério ($i = 1, 2$).

Um problema inerente em uma função de custo é que, normalmente, valores de diferentes critérios não são comparáveis entre si, o que inviabiliza a sua agregação imediata. Para resolver este problema, os valores de $EFT(v, p_j)$ e $RC(v, p_j)$ de cada tarefa v são normalizados para uma mesma escala sobre o conjunto total de processadores P, conforme descrito a seguir.

4.1. Escalonamento baseado em uma Função Ponderada

No algoritmo deste trabalho, ao escalonar as tarefas nos recursos, nenhum dos objetivos têm uma prioridade específica em principio. O processador que apresenta menor custo

para escalonar a tarefa é selecionado de acordo com uma função integrada que incorpora ambos critérios, tanto minimização do tempo de execução quanto maximização da confiabilidade. A função $f(v,p)$ é ponderada, sendo os pesos α_1 e α_2 associados a cada objetivo, onde $0 \leq \alpha_i \leq 1$, para $i = 1, 2$. Se $\alpha_i = 0$, a função zera o valor i-ésimo objetivo; se $\alpha_i = 1$, considera-se o seu valor total.

Para garantir a mesma magnitude dos objetivos $EFT(v, p_j)$ e $RC(v, p_j)$, foi utilizada a normalização da amplitude, que consiste em transformar todas as variáveis de modo que partilhem do mesmo valor mínimo e máximo. Uma forma simples consiste em aplicar um operador linear a todas as variáveis. Se m for o valor mínimo escolhido e M o valor máximo, então o operador linear aplicado ao elemento x_i de um vetor X (correspondente ao valor de um objetivo) é dado por $norm_i(m, M, m_x, M_x, x_i) = (M - m)\frac{x_i - m_x}{M_x - m_x} + m$, onde m_x e M_x são os valores mínimo e máximo desse vetor X, e $norm_i$ corresponde à variável normalizada. Em particular em MRCD, foram atribuídos os valores máximo $M = 100$ e mínimo $m = 0$. Inicialmente, $m_x = \infty$ e $M_x = 0$, para cada vetor objetivo.

A função de custo é reduzida para o caso onde existe uma dependência entre os pesos α_1 e α_2, especificamente $\alpha_2 = 1 - \alpha_1$. Seja w um valor constante, para o caso onde $w = \alpha_2$ com $0 \leq w \leq 1$, a função de custo pode ser especificada da seguinte maneira: $F(v,p) = min_{\forall p_j \in P'} \{(1-w)EFT(v,p_j) + wRC(v,p_j)\}$. Analisando os casos extremos de w, se $w = 0$ o algoritmo MRCD torna-se similar ao algoritmo HEFT [Topcuouglu et al. 2002], ou seja, o termo custo de confiabilidade não é considerado, sendo somente o *makespan* minimizado. Por outro lado, se $w = 1$ o escalonamento é especificado de acordo com a maximização da confiabilidade apenas.

O algoritmo MRCD proposto é apresentado na Figura 1 para um dado $G = (V, E, e, c)$ e conjunto de processadores P com as características relevantes definidas na Seção 3.2. Seja $V_{Ordprior}$ o conjunto de tarefas ordenado por *blevel* (linha 1). Para cada $v_i \in V_{Ordprior}$ são calculados os valores máximos e mínimos de cada objetivo (EFT_{max}, RC_{max}, EFT_{min} e RC_{min}) sobre o conjunto de processadores P (linhas 7 a 11). Note que $EFT(v_i, p_j)$ é calculado de acordo com um *list-scheduling* similar ao HEFT, que considera inserção de tarefas para calcular $EST(v_i, p_j)$ ($insercaoTarefas()$ na linha 5). Em seguida, os valores máximos e mínimos são utilizados para normalizar os objetivos (linhas 13 e 14). Com os objetivos normalizados EFT_{norm} e RC_{norm} é possível calcular em cada processador, a função de custo parcial $f(v_i, p_j)$ na linha 15. Depois de percorrer a lista de processadores, a tarefa v_i é alocada no processador p_v que minimiza $f(v_i, p_j)$ ($aloca(v_i, p_v)$ na linha 17). De acordo com o processador escolhido, são coletados o *makespan* ms da aplicação, o custo de confiabilidade RC_s e a parcela de diferença $Sdif_v$ (linhas 18, 19 e 20). A partir de RC_s é calculada a confiabilidade total RT na linha 21.

Com o objetivo de ajudar na busca de uma solução de compromisso entre ms e RT, uma informação adicional sobre o escalonamento E_k é especificada, a diferença média D calculada na linha 22. Esta métrica representa uma medida de desequilíbrio que houve entre os objetivos normalizados do escalonamento. $D(E_k)$ é calculada no algoritmo como a média das diferenças $rc - eft$ para cada v_i. O sinal de D indica qual dos objetivos em média contribuiu mais para o escalonamento gerado. Se negativo, a confiabilidade teve maior contribuição; se positivo, o tempo de execução foi dominante. O valor absoluto de D ($|D|$) representa a diferença da contribuição entre os objetivos.

Algorithm MRCD (G, P, w)
1: $\{ V_{Ordprior} = \langle v_1, v_2, \ldots v_n \rangle$ onde $blevel(v_i) > blevel(v_{i+1}), i = 1, \ldots |V|$;
2: **para** $i = 1 \ldots n$ **faça** $\{$
3: $EFT_{max} = 0$; $EFT_{min} = \infty$; $RC_{max} = 0$; $RC_{min} = \infty$; $F = \infty$; $Sdif = 0$;
4: $\forall p_j \in P$ **faça** $\{$
5: Calcular $EST[v_i, p_j]$ de acordo com $insercaoTarefas()$;
6: $EFT[v_i, p_j] = EST[v_i, p_j] + eh[v_i, p_j]$;
7: **se** $(EFT[v_i, p_j] < EFT_{min})$ **então** $EFT_{min} = EFT[v_i, p_j]$;
8: **se** $(EFT[v_i, p_j] > EFT_{max})$ **então** $EFT_{max} = EFT[v_i, p_j]$;
9: $RC[v_i, p_j] = FP[p_j] \times eh[v_i, p_j]$;
10: **se** $(RC[v_i, p_j] < RC_{min})$ **então** $RC_{min} = RC[v_i, p_j]$;
11: **se** $(RC[v_i, p_j] > RC_{max})$ **então** $RC_{max} = RC[v_i, p_j]$; $\}$
12: $\forall p_j \in P$ **faça**$\{$
13: $EFT_{norm} = norm_i(0, 100, EFT_{min}, EFT_{max}, EFT[v_i, p_j])$;
14: $RC_{norm} = norm_i(0, 100, RC_{min}, RC_{max}, RC[v_i, p_j])$;
15: $f(v_i, p_j) = (1 - w) \times EFT_{norm} + w \times RC_{norm}$;
16: **se** $(f(v_i, p_j) < F)$ **então** $F = f(v_i, p_j)$; $p_v = p_j$; $eft = EFT_{norm}$; $rc = RC_{norm}$; $\}$
17: $aloca(v_i, p_v)$;
18: **se** $(EFT[v_i, p_v] > ms)$ **então** $ms = EFT[v_i, p_v]$;
19: $RC_s = RC_s + RC[v_i, p_v]$;
20: $Sdif_v = Sdif_v + (rc - eft)$; $\}$
21: $RT = e^{-RC_s}$;
22: $D = Sdif_v / |V|$;
23: $retorna(ms, RT, D)$; $\}$
end.

Figura 1. Algoritmo MRCD

Se $|D|$ é um valor próximo de 0, comparado com o valor máximo possível ($M = 100$ escolhido), então é considerado que houve certo equilíbrio.

4.2. Ajuste dos Pesos na Função de Custo

Como visto no algoritmo estático MRCD, ao escalonar tarefas, o processador selecionado é aquele que apresenta menor custo da função ponderada $F(v, p_j)$. Executando o algoritmo com diferentes valores de w são gerados diferentes escalonamentos. Uma questão importante e pouco explorada na literatura é como achar determinada solução de escalonamento de forma que obtenha-se um compromisso conveniente entre o *makespan* e a confiabilidade. Com tal propósito, neste trabalho é proposta uma metodologia para escolha dos pesos e consequentemente, do escalonamento.

Em um problema de otimização multi-objetivo em que pode haver objetivos conflitantes, uma solução pode ser a melhor do ponto de vista de um objetivo, mas não em relação aos demais objetivos. Portanto, não se conseguirá uma única solução e sim um conjunto de soluções viáveis eficientes. Para se adotar uma boa solução, será necessário recorrer a informações adicionais que irão contribuir na escolha [Deb 2001].

A partir de um conjunto S de soluções factíveis, o subconjunto S' de soluções não-dominadas de S será aquele cujos elementos não são dominados por qualquer elemento de S. Então, quaisquer duas soluções de S' são não-dominadas entre si e as soluções em $S - S'$ são dominadas por pelo menos um elemento de S'. Esse conjunto S' forma uma aproximação da fronteira de Pareto e é chamado na literatura de *near-optimal pareto-front* no espaço de soluções [Deb 2001]. Para selecionar determinadas soluções dentro deste espaço escolhido S, podem ser considerados somente os elementos desta fronteira S', já que para qualquer solução fora da mesma existe uma solução melhor nela. Sobre a fronteira, não há, em princípio, preferência por nenhuma das soluções. Portanto, é necessário acrescentar informações adicionais, ou seja, expressões que contemplem relações entre

os valores dos objetivos para se escolher determinada solução. Estas informações são denominadas subjetivas, podendo ser, inclusive, informações puramente qualitativas ou baseadas na experiência. Sendo assim, com estas informações de mais alto nível e o conjunto de soluções não-dominadas, será possível selecionar soluções em S com mais qualidade que satisfaçam determinados interesses.

O algoritmo MRCD(G,P, w) além de informar o *makespan* $ms(E_k)$ e a confiabilidade $RT(E_k)$ para w, gera também a informação adicional $D(E_k)$ sobre o escalonamento E_k obtido. Para cada solução, se $D(E_k) > 0$, indica que o *makespan* contribuiu mais que a confiabilidade. Se $D(E_k) < 0$, a confiabilidade contribuiu mais que o tempo de execução. Para $D(E_k) = 0$ ou próximo, os objetivos contribuíram de forma idêntica ou equilibrada para a solução do problema. Uma dessas soluções pode ser escolhida de acordo com o interesse do usuário. Com o conhecimento do valor de $D(E_k)$ é possível ajustar o w e chegar a uma solução de escalonamento conveniente.

Para gerar soluções com maior qualidade, a escolha de w pode ser feita conforme a metodologia descrita a seguir. Executando-se o algoritmo para diversos valores de w, pode ser construído o subconjunto de soluções não-dominadas S' do espaço de busca inicial S. Com o objetivo de achar um compromisso em S', uma solução de equilíbrio, denotada por S_t, pode ser obtida ao procurar o menor $|D(E_k)|$ em S'. Já aquela com menor $D(E_k)$ no conjunto S', oferece a melhor solução para a confiabilidade (S_{RT}) e a de maior $D(E_k)$, a solução que prioriza o *makespan* (S_{ms}).

5. Análise de Desempenho

Com o objetivo de avaliar o desempenho da estratégia de escalonamento proposta, MRCD é comparado com outros algoritmos de escalonamento da literatura. Para realizar os experimentos foram empregados duas classes de aplicações, Gauss e RAND. No primeiro caso, é avaliada uma aplicação real que representa o método de eliminação de Gauss, utilizado para solucionar sistemas de equações lineares. Já a segunda classe RAND, representa aplicações com GADs gerados aleatoriamente. As notações G_N e R_N são usadas para indicar os GADs de Gauss e RAND, respectivamente, sendo N o número de tarefas.

Para gerar o escalonamento estático de cada GAD sobre a configuração considerada, foram associadas estimativas do ambiente real às características do modelo. Para facilitar esta especificação, foram consideradas aplicações sintéticas dos GADs com o uso de tempos de comunicação e computação pré-fixados. O peso unitário de comunicação associado aos arcos de qualquer instância foi escolhido. Os grafos que representam a aplicação de eliminação de Gauss se caracterizam por possuir pesos distintos de computação. Já os grafos da aplicação aleatória RAND foram especificados com pesos de computação homogêneos, pré-fixados para $e(v) = 50$, para todo $v \in V$.

Durante a análise, as heurísticas comparadas foram divididas em dois grupos, heurísticas que geram uma única solução [Topcuouglu et al. 2002, Dongarra et al. 2007, Qin et al. 2002] do Grupo 1, e heurísticas que geram múltiplas soluções [Hakem and Butelle 2007] do Grupo2.

No Grupo1, o algoritmo HEFT gera um escalonamento mono-objetivo que não considera confiabilidade, a função de custo só minimiza o *makespan*. A partir da solução especificada pelo HEFT, calcula-se a confiabilidade associada aquele escalonamento, para

permitir a comparação deste com MRCD. RCDMod gera um escalonamento bi-objetivo usando uma função de custo hierárquica, conforme [Qin et al. 2002], mas sem restrições de tempo (*deadlines*) e a função de custo considera o tempo de fim das tarefas e não o tempo de início. Finalmente, o escalonamento bi-objetivo RHEFT usa uma função de custo integrada que minimiza a função de custo formada pelo produto $EFT(v,p) \times FP(p)$ conforme sugerido em [Dongarra et al. 2007].

O Grupo2 corresponde às heurísticas que geram múltiplas soluções de escalonamento. Assim como MRCD, o algoritmo BSA de [Hakem and Butelle 2007] especifica também um escalonamento bi-objetivo ponderado, mas utiliza a seguinte função de custo $\sqrt{\theta\left(\frac{EFT(v,p)}{EFT_{max}(v,p)}\right)^2 + (1-\theta)\left(\frac{RC(v,p)}{RC_{max}(v,p)}\right)^2}$. Para comparar as abordagens, a versão implementada deste algoritmo (BSAMod), ordena as tarefas inicialmente por $blevel()$.

Cada heurística de escalonamento dentro dos grupos anteriormente relacionados é comparada com a abordagem proposta. Para um conjunto de instâncias, a quantidade de escalonamentos *Dominantes* e *Dominados* é a principal medida ao comparar as heurísticas. Seja $(E_l, ms(E_l), RT(E_l))$ uma tripla tal que E_l é o escalonamento especificado por uma dada heurística, com *makespan* $ms(E_l)$ e confiabilidade $RT(E_l)$. Um escalonamento E_1 de uma heurística é considerado dominado por outro escalonamento E_2 de outra heurística, se e somente se, E_1 apresenta valores $ms(E_1)$ e $RT(E_1)$ piores que $ms(E_2)$ e $RT(E_2)$, respectivamente, podendo somente um dos valores ms ou RT ser iguais. Neste caso E_2 domina E_1. Os escalonamentos são iguais quando ambas soluções objetivas são idênticas. Em qualquer outro caso diferente dos anteriores, os escalonamentos são considerados incomparáveis ou não dominados entre si. Nas heurísticas de múltiplas soluções, só foram consideradas as soluções não-dominadas contidas em S'. Note que o conjunto S é gerado com w: $\{0.1, 0.2, 0.3, ..., 0.9\}$, conforme Seção 4.2, formando um total de nove soluções em todos os experimentos realizados.

O sistema distribuído simulado para o ambiente é composto por m processadores agrupados em três grupos P_0, P_1 e P_2, cada com $m/3$ processadores. Os números de tarefas para os GADs analisados foram para Gauss $|V| = \{152, 252, 377, 527, 702, 1034\}$ tarefas, e para RAND $|V| = \{80, 98, 152, 256, 364, 546\}$ tarefas.

Nos diferentes grupos de processadores, as taxas de falha FP foram geradas uniformemente nos intervalos $[10^{-5}, 3.3 \times 10^{-5}]$ $\forall p_i \in P_0$, $[3.4 \times 10^{-5}, 6.6 \times 10^{-5}]$ $\forall p_i \in P_1$ e $[6.7 \times 10^{-5}, 10^{-4}]$ $\forall p_i \in P_2$, respectivamente. Em relação ao índice de retardo, $csi(p_i) = 73$ $\forall p_i \in P_0$, $csi(p_i) = 53$, $\forall p_i \in P_1$ e $csi(p_i) = 33$ $\forall p_i \in P_2$ (valores esses coletados de [Nascimento et al. 2007]). FP e csi por grupo apresentam comportamentos diferentes, ou seja, um grupo apresenta os processadores mais lentos e mais confiáveis e um outro grupo, os processadores mais rápidos e menos confiáveis. Esta configuração é escolhida para estudar cenários de risco onde é difícil chegar a um acordo entre os objetivos do escalonamento.

5.1. Um Estudo de Caso: Gauss com $N = 702$

São apresentados inicialmente os resultados experimentais sobre um GAD que representa a aplicação de Gauss com 702 tarefas (G_{702}) no cenário descrito anteriormente com 24 processadores, sendo que, neste caso, o índice de retardo $csi(p_i) = 53$ para todo $p_i \in P$.

A Tabela 1 mostra os resultados obtidos para este exemplo, considerando o con-

Tabela 1. Ajuste do GAD G_{702}

w	ms	RT	D	S'
$w = 0.1$	1450.0	0.41396	41.0	-
$w = 0.2$	1450.0	0.48640	33.1	-
$w = 0.3$	1450.0	0.49544	30.4	-
w=0.4	**1450.0**	**0.49939**	**28.9**	**x** (S_{ms})
$w = 0.5$	1450.0	0.49884	29.2	-
w=0.6	**2010.8**	**0.58722**	**-0.84**	**x** (S_t)
$w = 0.7$	2431.6	0.62786	-10.9	x
$w = 0.8$	2988.1	0.64296	-11.8	x
w=0.9	**6434.2**	**0.69350**	**-46.6**	**x** (S_{RT})

junto S para os valores de w: $\{0.1, 0.2, 0.3, ..., 0.9\}$. Inicialmente, são selecionadas as soluções não-dominadas S' dentre as calculadas em S. Conforme a Tabela 1, as soluções em S' são obtidas para os valores $w = \{0.4, 0.6, 0.7, 0.8, 0.9\}$. Dentre as soluções em S', podem ser selecionadas três soluções de interesse: uma solução de menor *makespan* S_{ms}, ou seja, com o maior D que ocorre para $w = 0.4$, a solução de maior confiabilidade S_{RT}, ou seja, com o menor D para $w = 0.9$ e uma solução de compromisso S_t que oferece um maior equilíbrio entre os objetivos, ou seja, o menor $|D|$ para $w = 0.6$. Apesar de outras soluções obtidas terem o mesmo *makespan* que o da solução S_{ms} com $w = 0.4$, conforme a Tabela 1, a confiabilidade das outras soluções são menores, sendo soluções dominadas. Desta forma, com a abordagem proposta é possível achar uma solução de compromisso entre o *makespan* e a confiabilidade, a qual pode ser de interesse para o usuário.

5.2. Comparação com Outras Heurísticas

Para avaliar o algoritmo proposto MRCD com cada heurística de uma única solução do Grupo 1 (HEFT, RCDMod, RHEFT), três conjuntos denotados por C1, C2 e C3 são especificados nas Tabelas 2, 3, 4 e 5. A cada conjunto está associado duas colunas que contém as soluções do escalonamento obtido para cada GAD na forma (ms,RT). Em C1 são mostrados os resultados de HEFT e MRCD, em C2, RCDMOD e MRCD e finalmente em C3 de RHEFT e MRCD. A solução MRCD apresentada é escolhida dentro do subconjunto S' (considerando S com nove soluções para w: $\{0.1, 0.2, 0.3, ..., 0.9\}$) de forma que domine a solução do outro algoritmo quando possível ou esteja próximo dela. As duas últimas linhas das tabelas mostram o número de soluções dominadas e dominantes, conforme definido no início desta seção.

Em um primeiro experimento foi utilizado um ambiente de 24 processadores dividido nos grupos P_0, P_1 e P_2. Os GADs de Gauss e RAND analisados contém um número variado de tarefas N, conforme visto nas Tabelas 2 e 3, respectivamente. Com o aumento de N, nota-se que o *makespan* ms aumenta e diminui a confiabilidade RT. Quanto mais tarefas, maior a quantidade de trabalho por processador e maior a probabilidade da aplicação falhar.

Em um segundo experimento, o número total de processadores m foi variado, sendo que $m/3$ processadores pertenceram a cada grupo P_0, P_1 e P_2. As Tabelas 4 e 5 mostram os resultados para G_{1034} e R_{546}, respectivamente. Pode ser observado nesses resultados que, o *makespan* tende a diminuir e a confiabilidade tende a aumentar. Com o crescimento do ambiente, a melhora de ambos objetivos se deve ao fato de que o grau de paralelismo é explorado pelas estratégias de escalonamento. As tarefas podem ser alocadas em um número maior de processadores com melhores características, tanto em

Tabela 2. Gauss com $m = 24$ em Grupo1

G_N	C1		C2		C3	
	HEFT	MRCD	RCDMod	MRCD	RHEFT	MRCD
G_{152}	(190.0, 0.92071)	**(190.0, 0.93294)**	(2406.0, 0.95761)	(1283.3, 0.95546)	(694.9, 0.94573)	(449.4, 0.94287)
G_{252}	(318.7, 0.83511)	**(318.7, 0.85879)**	(5201.9, 0.91061)	(2715.6, 0.90609)	(1309.6, 0.88130)	(856.3, 0.87844)
G_{377}	(480.4, 0.71847)	**(480.4, 0.75140)**	(9603.8, 0.84124)	(4981.5, 0.83350)	(2204.6, 0.78693)	(1494.8, 0.78560)
G_{527}	(675.1, 0.58537)	**(675.1, 0.61228)**	(15976.7, 0.75007)	(8211.0, 0.73851)	(3560.9, 0.66798)	(2273.0, 0.66578)
G_{702}	(902.8, 0.44064)	**(902.8, 0.45539)**	(24685.6, 0.64124)	(12687.4, 0.62604)	(5258.9, 0.52865)	**(3463.3, 0.53184)**
Dominadas	5	0	0	0	1	0
Dominantes	0	5	0	0	0	1

Tabela 3. RAND com $m = 24$ em Grupo 1

R_N	C1		C2		C3	
	HEFT	MRCD	RCDMod	MRCD	RHEFT	MRCD
R_{80}	(330.0, 0.82648)	**(330.0, 0.84159)**	(5840.0, 0.90021)	(3066.0, 0.89523)	(1022.0, 0.85271)	**(949.0, 0.86478)**
R_{98}	(330.0, 0.79053)	**(330.0, 0.80765)**	(7154.0, 0.87917)	(3650.0, 0.87303)	(1460.0, 0.83060)	**(1314.0, 0.84035)**
R_{152}	(396.0, 0.69599)	**(396.0, 0.69907)**	(11096.0, 0.81895)	(5694.0, 0.81015)	(1679.0, 0.73349)	**(1533.0, 0.75338)**
R_{256}	(528.01, 0.54215)	**(528.00, 0.54215)**	(18688.0, 0.71434)	(10512.0, 0.70276)	(1606.01, 0.56659)	**(1314.0, 0.58304)**
R_{377}	(759.0, 0.41596)	**(759.0, 0.41804)**	(26572.0, 0.61983)	(13505.0, 0.60385)	(2993.0, 0.45685)	**(1825.0, 0.46744)**
Dominadas	5	0	0	0	6	0
Dominantes	0	5	0	0	0	6

relação a taxas de falha ($FP()$) quanto ao poder computacional ($csi()$).

Em todos os testes, as duas primeiras colunas C1 (HEFT x MRCD) onde o tempo é prioridade, os valores de *makespans* obtidos são muito menores do que os valores das colunas C2 e C3, correspondentes. Os algoritmos de escalonamentos (HEFT e MRCD) tendem a alocar mais tarefas no grupo de processadores de menor velocidade, $csi = 33$. Já a confiabilidade calculada de acordo com o escalonamento gerado por HEFT, conforme visto em C1, para estes casos é pior devido à taxa de falha mais alta que foi associada a esse processadores.

Pelos resultados obtidos na comparação com o Grupo 1 (Tabelas 2, 3, 4 e 5), MRCD sempre gera soluções que dominam as soluções obtidas por HEFT, sendo que HEFT não considera a confiabilidade em sua função objetivo. Mesmo que HEFT obtenha boas soluções em relação ao *makespan* por priorizar somente este objetivo, com MRCD os resultados alcançam o mesmo *makespan*, e ainda apresentam valores superiores de confiabilidade. Repare que, para C1 nessas tabelas, as soluções de MRCD escolhidas são do subconjunto S', dando prioridade à minimização do *makespan* (com o maior valor de D), mas ainda considerando a maximização da confiabilidade.

Já nas colunas C2 do Grupo 1 (nas Tabelas 2, 3, 4 e 5), RCDMod se caracteriza por priorizar sempre a confiabilidade, embora o tempo de execução seja também considerado como um objetivo secundário (função hierárquica). Como esperado, os valores gerados de confiabilidade nos diferentes testes realizados são mais altos do que MRCD. No entanto, em compensação os *makespans* das soluções geradas por RCDMod praticamente dobram

Tabela 4. GAD G_{1034} em Grupo 1 variando m

m	C1		C2		C3	
	HEFT	MRCD	RCDMod	MRCD	RHEFT	MRCD
24	(1504.1, 0.86253)	**(1499.8, 0.86628)**	(44389.8, 0.92320	(22758.4, 0.91922)	(8743.9, 0.88970)	**(5309.0, 0.89017)**
45	(1335.8, 0.84661)	**(1335.8, 0.88030)**	(44389.8, 0.93558)	(23805.3, 0.92982)	(9291.4, 0.91737)	**(4623.8, 0.91816)**
66	(1335.8, 0.85161)	**(1335.8, 0.89391)**	(22229.9, 0.95234)	(9640.3, 0.94698)	(5673.5, 0.93053)	**(4940.6, 0.93929)**
87	(1335.8, 0.84610)	**(1335.8, 0.90048)**	(22229.9, 0.95234)	(7797.8, 0.94846)	(4885.1, 0.93406)	**(4137.6, 0.93877)**
108	(1335.8, 0.84508)	**(1335.8, 0.90251)**	(44389.8, 0.95234)	(11617.2, 0.94738)	(6556.8, 0.93440)	(4108.4, 0.93349)
213	(1335.8, 0.84468)	**(1335.8, 0.91101)**	(22229.9, 0.95234)	(6723.2, 0.94844)	(4363.9, 0.93854)	(2955.0, 0.93662)
Dominadas	6	0	0	0	4	0
Dominantes	0	6	0	0	0	4

Tabela 5. GAD R_{546} em Grupo 1 variando m

m	C1		C2		C3	
	HEFT	MRCD	RCDMod	MRCD	RHEFT	MRCD
24	(1122.0, 0.87705)	**(1122.0, 0.87737)**	(39858.0, 0.93076)	(20513.0, 0.92717)	(4453.0, 0.88868)	**(2628.0, 0.89193)**
45	(629.0, 0.87248)	**(627.0, 0.87358)**	(39858.0, 0.94196)	(20805.0, 0.93659)	(2920.0, 0.89700)	**(2263.0, 0.90978)**
66	(627.0, 0.87567)	**(627.0, 0.88199)**	(19929.0, 0.95710)	(8541.0, 0.95214)	(2847.0, 0.90954)	**(1533.0, 0.91425)**
87	(627.0, 0.87032)	**(627.0, 0.88323)**	(19929.0, 0.95710)	(6935.0, 0.95350)	(2409.0, 0.91396)	**(1347.0, 0.91838)**
108	(627.0, 0.86827)	**(627.0, 0.89088)**	(39858.0, 0.95710)	(10804.0, 0.95271)	(2628.0, 0.91306)	**(1274.0, 0.91814)**
213	(627.0, 0.86071)	**(627.0, 0.90427)**	(19929.0, 0.95710)	(5986.0, 0.95347)	(1971.0, 0.92740)	**(1387.0, 0.93447)**
Dominadas	6	0	0	0	6	0
Dominantes	0	6	0	0	0	6

os de MRCD. Nesta comparação os melhores w para MRCD estão próximos de $w = 0.9$, pois soluções com maior prioridade à confiabilidade foram escolhidas (com o menor D).

As soluções obtidas pela última heurística bi-objetivo RHEFT do Grupo 1 comparadas em C3, apresentam maior equilíbrio entre os objetivos que as soluções geradas pelas outras duas heurísticas HEFT e RCDMod. Isto se deve ao fato de que RHEFT difere das outras pois usa um função integrada com o propósito de considerar simultaneamente os dois objetivos do problema ao escalonar as tarefas. Soluções geradas por MRCD (mais próximas de RHEFT), dominam na maioria das vezes as soluções de RHEFT como mostram as quatro tabelas mencionadas. Observe que nas poucas soluções onde o algoritmo proposto não domina RHEFT, o *makespan* de MRCD é quase a metade do RHEFT em média, e a confiabilidade embora menor, fica muito próxima da solução de RHEFT. No caso da estratégia RHEFT, embora a intenção seja priorizar igualmente ambos objetivos do problema, a função acaba priorizando um pouco mais a confiabilidade do que a minimização do *makespan*. De acordo com as Tabelas 2, 3, 4 e 5, as soluções não dominantes de MRCD sobre RHEFT, também são soluções não dominadas por RHEFT.

Repetindo os dois tipos de experimentos, inicialmente fixando $m = 24$ e posteriormente variando o número de processadores, MRCD foi comparado com o algoritmo BSAMod do Grupo 2 (heurísticas de múltiplas soluções). As Tabelas 6 e 7 mostram

Tabela 6. Variação da aplicação em Grupo2 com $m = 24$

	Dominantes		Dominadas	
	BSAMod	MRCD	BSAMod	MRCD
G_{152}	0	2	2	0
G_{252}	0	2	2	0
G_{377}	0	3	3	0
G_{527}	0	3	3	0
G_{702}	0	4	4	0
Total1	**0**	**14**	**14**	0
R_{80}	0	2	2	0
R_{98}	0	3	2	0
R_{152}	1	2	2	1
R_{256}	1	0	0	1
R_{364}	1	1	1	1
Total2	**3**	**8**	**7**	**3**

Tabela 7. Variação do ambiente em Grupo2 para G_{1034} e R_{546}

G_{1034}	Dominantes		Dominadas	
	BSAMod	MRCD	BSAMod	MRCD
$m = 24$	0	4	3	0
$m = 45$	0	5	4	0
$m = 66$	0	3	3	0
$m = 87$	0	3	3	0
$m = 108$	0	2	2	0
$m = 213$	0	1	1	0
Total1	**0**	**18**	**16**	**0**
R_{546}				
$m = 24$	0	3	3	0
$m = 45$	1	0	0	1
$m = 66$	1	0	0	1
$m = 87$	0	4	4	0
$m = 108$	0	2	2	0
$m = 213$	0	3	4	0
Total2	**2**	**12**	**13**	**2**

o número de soluções dominantes e dominadas, tanto em relação a BSAMod quanto a MRCD. Para cada linha da tabela foram geradas, com cada heurística, nove soluções em total que formam o conjunto inicial S. Dentre estas soluções foram consideradas apenas as do subconjunto S'. No total, a aplicação de Gauss apresenta os melhores resultados quando comparada com RAND, apresentando um número variado de soluções dominantes em MRCD e nenhuma solução dominante em BSAMod. Para a aplicação RAND, MRCD provê a maioria das melhores soluções sobre BSAMod. Da mesma forma, com o aumento do número de processadores MRCD mostra também ser superior que BSAMod. Em geral, MRCD conseguiu superar BSAMod na maioria dos testes realizados. Portanto, para ambientes distribuídos heterogêneos, MRCD se mostra favorável comparado com BSAMod em relação a sua função de custo.

6. Conclusões e Trabalhos Futuros

Foi proposta uma nova estratégia de escalonamento de tarefas em ambientes distribuídos para maximizar a confiabilidade da aplicação além de minimizar seu tempo de execução. Através de uma função de custo ponderada diferente e uma metodologia de análise dos pesos desta função, a estratégia consegue gerar escalonamentos mais promissores. A partir de informações adicionais geradas pelo escalonamento estático e conceitos de dominância, a abordagem sugere soluções convenientes para o problema bi-objetivo. Diferentemente da literatura, este trabalho responde a questão de como achar determinada solução de escalonamento que represente um compromisso entre o *makespan* e confiabilidade. A abordagem pode ser aplicada aos trabalhos já existentes, desde que se introduzam informações adicionais nos algoritmos, em relação aos objetivos.

Testes realizados destacam a importância de usar uma abordagem flexível em ambientes distribuídos heterogêneos e e os resultados se mostram favoráveis, comparados com outras propostas bi-objetivo existentes. Os resultados mostram que na maioria dos casos foi possível achar uma solução MRCD muito próxima das soluções obtidas pelas outras heurísticas, sendo que na maioria das vezes MRCD domina a outra solução. Em particular, para HEFT, escalonamento mono-objetivo, os resultados obtidos mostram a importância de usar uma abordagem de escalonamento bi-objetivo que considere também a confiabilidade em ambientes heterogêneos propensos a falhas. Embora as ou-

tras heurísticas considerem a confiabilidade além do tempo, os resultados com RCDMod e RHEFT mostram a importância de usar um escalonamento ponderado, que permita mudar as prioridades dos objetivos de acordo a distintos interesses. Já o único algoritmo que propõe ponderação BSAMod, gera resultados inferiores a MRCD na maioria dos casos.

A estratégia de escalonamento pode ser utilizada em trabalhos que consideram as máquinas multicores atuais. Cada core é modelado como uma unidade de processamento, que tem associados uma probabilidade de falha e um fator de heterogeneidade de computação. A latência de comunicação é associada à comunicação entre cada par de unidades de processamento.

Existe uma segunda parte deste trabalho, em fase de finalização, que considera tolerância a falhas além da confiabilidade. Nesta abordagem, além das tarefas primárias, as versões *backups* são também escalonadas de acordo com o algoritmo bi-objetivo deste trabalho. Para garantir tolerância a falhas sobre um ambiente real, uma ferramenta em MPI que utiliza estes algoritmos, está sendo desenvolvida para grades computacionais.

Referências

Deb, K. (2001). Multi-objetive optimization using evolutionary algorithms. *Wiley and Sons. England*.

Dogan, A. and Ozguner, F. (2002). Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):308–323.

Dogan, A. and Ozguner, F. (2005). Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. *Comput. J.*, 48(3):300–314.

Dongarra, J., Jeannot, E., Saule, E., and Shi, Z. (2007). Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Proceedings of 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '07)*. ACM.

Hakem, M. and Butelle, F. (2007). Reliability and scheduling on systems subject to failures. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, page 38.

Nascimento, A. P., Sena, A. C., Boeres, C., and Rebello, V. E. F. (2007). Distributed and dynamic self-scheduling of parallel MPI grid applications. *Concurrency and Computation: Practice and Experience*, 19(14):1955–1974.

Qin, X., Jiang, H., and Swanson, D. R. (2002). An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing (ICPP'02)*, page 360, Washington, DC, USA. IEEE Computer Society.

Reed, D. A., da Lu, C., and Mendes, C. L. (2006). Reliability challenges in large systems. *Future Generation Computer Systems*, 22(3):293–302.

Topcuouglu, H., Hariri, S., and you Wu, M. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274.