Robust Hybrid Mechanisms for Scheduling of Grid Tasks

R. L. Curi, D. M. Batista and N. L. S. da Fonseca

Abstract— This paper proposes three new hybrid mechanisms for the scheduling of grid tasks, which integrate reactive and proactive approaches. They differ by the scheduler used to define the initial schedule of an application and by the scheduler used to reschedule the application. The mechanisms are compared to reactive and proactive mechanisms. Results show that hybrid approach produces performance close to that of the reactive mechanisms, but demanding less migrations.

***Keywords*— Computational grids, Scheduling of grid tasks, Reactive approach, Proactive approach.**

I. INTRODUÇÃO

A FIM de prover os requisitos de Qualidade de Serviço (*Quality of Service* – QoS), deve-se adotar, em grades computacionais, mecanismos eficientes para o escalonamento das tarefas que compõem uma aplicação [1], visando assegurar que o mapeamento entre tarefas e recursos minimize o tempo total de execução da aplicação (*makespan*). Mecanismos de escalonamento devem, também, lidar com incertezas sobre os reais valores das demandas das aplicações (capacidade de processamento requerida para a execução de uma tarefa e largura de banda requerida para o envio de dados de uma tarefa a outra), bem como incertezas sobre a disponibilidade dos recursos (capacidade de processamento disponível em cada computador e largura de banda disponível nos enlaces)[2,3].

As incertezas sobre os reais valores de demanda das aplicações e disponibilidade dos recursos surgem porque [2]: os recursos podem estar sendo compartilhados por outros usuários que pertençam ou não à grade; as informações sobre a demanda de uma aplicação são imprecisas, principalmente com relação à quantidade de dados a serem transferidos de uma tarefa a outra; e as informações sobre disponibilidade dos recursos, retornadas pelas ferramentas de monitoramento, são também imprecisas.

No geral, duas abordagens ortogonais podem ser adotadas para o escalonamento em grades computacionais: a reativa e a pró-ativa. Mecanismos reativos [4-7] monitoram os recursos durante a execução de uma aplicação, visando a atualização das informações sobre o estado da grade e a verificação da permanência da efetividade do último escalonamento proposto. Migrações de tarefas podem ser propostas para melhorar o desempenho, se necessário.

Na abordagem pró-ativa, o grau de incerteza sobre os valores de demanda da aplicação e disponibilidade dos recursos são fornecidos como entrada para o escalonador. Uma abordagem para lidar com tais incertezas é formular o problema de escalonamento como um problema de otimização *fuzzy* [8-11]. A abordagem pró-ativa evita o *overhead* (tempo gasto a mais) causado pela computação dos reescalonamentos e pela migração das tarefas. Além disso, ela não necessita de monitoramento frequente e sistemas de *checkpoint*, como na abordagem reativa. Uma desvantagem é que, após a aplicação ter sido submetida para execução, não haverá mais avaliação, de acordo com o estado atual da grade, de oportunidades de propor um escalonamento melhor, isto é, que conduza a um *makespan* menor.

Esse artigo introduz três novos mecanismos, denominados de mecanismos híbridos, que capitalizam sobre os benefícios das abordagens reativa e pró-ativa. Esses mecanismos são orientados para aplicações compostas por um conjunto de tarefas dependentes entre si. Eles são avaliados e comparados com mecanismos baseados na abordagem reativa [4] e na abordagem pró-ativa [10,11]. Adicionalmente, dois novos escalonadores são propostos para esses mecanismos.

Em [6], foram usados os mecanismos pró-ativo e reativo para a construção de grades tolerantes a falhas. Um escalonador baseado em um algoritmo evolucionário, bem como no método *Anycast*, foi empregado. A diferença dessa proposta para as apresentadas no presente artigo é que ela visa somente lidar com falhas, e ignora a variação na disponibilidade dos recursos. Em [7], uma abordagem de auto cura é proposta para realizar migração pró-ativa das tarefas em ambientes de Computação de Alto Desempenho, a qual é também orientada para prover tolerância a falhas. Além disso, os trabalhos em [6] e [7] ignoram a entrada de novos recursos quando avaliam a necessidade de migração de tarefas. Em contraste, os mecanismos híbridos propostos no presente artigo consideram alterações na disponibilidade dos recursos de ambos os tipos (falhas e variações na disponibilidade) e, além disso, evitam migrações desnecessárias.

O restante desse artigo está organizado como segue: A Seção II introduz os três novos mecanismos híbridos e os dois novos escalonadores. A Seção III apresenta os experimentos conduzidos para analisar os mecanismos e a Seção IV conclui o artigo.

R. L. Curi, Universidade Estadual de Campinas (Unicamp), Campinas, São Paulo, Brasil, ra063848@students.ic.unicamp.br

D. M. Batista, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, batista@ime.usp.br

N. L. S. da Fonseca, Universidade Estadual de Campinas (Unicamp), Campinas, São Paulo, Brasil, nfonseca@ic.unicamp.br

II. MECANISMOS HÍBRIDOS

Se por um lado, mecanismos pró-ativos não implicam o *overhead* causado pelo monitoramento, computação de reescalonamentos e migrações de tarefas, por outro, eles não reagem a alterações na disponibilidade dos recursos que ocorram durante a execução de uma aplicação. Contrariamente, mecanismos reativos são capazes de reagir, porém implicam o *overhead* mencionado. Além disso, os escalonamentos produzidos não consideram incertezas - na informação fornecida aos escalonadores - sobre os reais valores de demanda da aplicação e de disponibilidade dos recursos.

Os mecanismos híbridos propostos seguem o ciclo dos mecanismos reativos, mas o escalonamento inicial e/ou os reescalonamentos são dados por escalonadores pró-ativos. Desse modo, reações a alterações na disponibilidade dos recursos podem ser tão imediatas quanto nos mecanismos reativos, e a necessidade de migrações de tarefas pode ser reduzida pela consideração das incertezas sobre os valores de demanda da aplicação e de disponibilidade dos recursos. Variações de mecanismos híbridos são analisadas, considerando-se combinações do uso do escalonador pró-ativo na etapa inicial e nas etapas de reescalonamento.

Em relação à comparação entre as abordagens pró-ativa e reativa, esse artigo tenta responder diversas questões, tais como: qual das duas abordagens diminui o número de migrações de forma mais significativa; qual é o ganho obtido pelo uso da abordagem pró-ativa quando adotada nas diferentes etapas do ciclo reativo; e qual dessas abordagens conduz a valores menores de *makespan*.

Como os desempenhos dos mecanismos dependem fortemente dos escalonadores usados, dois novos escalonadores, que são versões revisadas de escalonadores apresentados em [4,10], são propostos na Subseção II.A, e usados pelos três mecanismos apresentados na Subseção II.B.

A. ESCALONADORES

Os escalonadores propostos são baseados em dois escalonadores existentes: o IPDT [4,12] e o IP-FULL-FUZZY [10,11]. O primeiro é apropriado para mecanismos reativos, enquanto o segundo é usado em mecanismos pró-ativos. Os requisitos da aplicação dada como entrada são representados por um Grafo Acíclico Orientado (*Directed Acyclic Graph* - DAG), onde os nós correspondem às tarefas e os arcos às dependências entre as tarefas, isto é, se a tarefa i tem um arco direcionado para a tarefa j, então a tarefa j depende de dados produzidos pela tarefa i. Os pesos dos nós representam o número de instruções de cada tarefa e os pesos dos arcos representam a quantidade de *bits* que devem ser transferidos de uma tarefa a outra. A grade dada como entrada é representada por um grafo, onde os nós correspondem aos computadores e as arestas correspondem aos enlaces. Os pesos dos nós representam a capacidade de processamento disponível, em minutos/instrução, e os pesos das arestas representam o inverso da largura de banda disponível, dada em minutos/*bit*.

O IPDT resolve um problema de programação linear inteira. O tempo é discretizado e definido como $\mathcal{T} = \{1,\ldots,T_{max}\}$, onde T_{max} corresponde ao teto do *makespan* obtido quando todas as tarefas são executadas sequencialmente no computador mais rápido. A função objetivo do problema é a minimização do *makespan*, o qual deve ser no máximo igual a T_{max}. O conjunto de tarefas é representado por $\mathcal{J} = (V_J,E_J)$, onde $E_J = \mathcal{D}$ é o conjunto de arcos $\{ij : i < j$, e existe um arco do vértice i para o vértice j no DAG$\}$, isto é, as dependências da aplicação. O conjunto de computadores é representado por $\mathcal{H} = (V_H,E_H)$. $\delta(l)$ é o conjunto de computadores conectados com o computador l, incluindo o próprio l. I_i corresponde ao peso da tarefa i, $B_{i,j}$ corresponde ao peso do arco entre as tarefas i e j, TI_l corresponde ao peso do computador l e $TB_{k,l}$ ao peso do enlace entre os computadores k e l.

$x_{i,t,k}$ é uma variável binária que tem valor 1 se a tarefa i executa no computador k finalizando no tempo t, e 0 caso contrário.

A formulação adotada nos experimentos, denominada de IPDT-2, é mostrada a seguir:

$$\text{Minimize} \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{H}} t\, x_{j,t,k}$$

sujeito a

$$\sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{H}} x_{j,t,k} = 1 \quad \text{para } j\in\mathcal{J}; \tag{D1}$$

$$x_{j,t,k} = 0 \quad \text{para } j\in\mathcal{J},\quad k\in\mathcal{H},\quad t\in\{1,\ldots,\lceil MC + I_j TI_k\rceil\}; \tag{D2}$$

$$\sum_{k\in\delta(l)}\sum_{s=1}^{t-\lceil I_j TI_l + B_{i,j}TB_{k,l}\rceil} x_{i,s,k} \geq \sum_{s=1}^{t} x_{j,s,l} \quad \text{para } i,j\in\mathcal{J},\quad ij\in\mathcal{D},\quad l\in\mathcal{H},\quad t\in\mathcal{T}; \tag{D3}$$

$$\sum_{j\in\mathcal{J}}\sum_{s=t}^{t+\lceil I_j TI_k\rceil-1} x_{j,s,k} \leq 1 \quad \text{para } k\in\mathcal{H},\quad t\in\mathcal{T},\quad t\leq T_{max} - \lceil I_j TI_k\rceil; \tag{D4}$$

$$x_{j,t,l}\in\{0,1\} \quad \text{para } j\in\mathcal{J},\quad l\in\mathcal{H},\quad t\in\mathcal{T}. \tag{D5}$$

O IPDT-2 difere do IPDT pelas restrições D2, D3 e D4. A restrição D2 deve ser aplicada a $t \in \{1,\ldots,|MC+I_j TI_k|\}$, onde MC é relacionado ao fato de que uma tarefa pode iniciar somente após o término da execução de todas as suas dependências, considerando-se o caminho de dependências mais longo. As restrições D3 e D4 colocam t para fora da operação de teto. Quando comparado com a formulação original [3], essas alterações permitem que os inícios das tarefas sejam programados para antes do que seriam no IPDT, e consequentemente o *makespan* decresce.

A restrição D1 estabelece que cada tarefa j deve ser executada em um único computador k, finalizando em um único tempo discreto t. A restrição D2 estabelece que a tarefa j, que executa no computador k, não pode finalizar antes do fim da última tarefa no caminho de dependências mais longo (que inicia na tarefa 0 e finaliza na tarefa j), para quando essas tarefas são executadas sequencialmente no computador mais

rápido (*MC* corresponde a esse tempo de execução). A restrição D3 estabelece que a tarefa *j* pode iniciar somente após ter recebido, de todas as tarefas que ela depende, os dados requeridos. A restrição D4 estabelece que somente uma única tarefa pode executar por vez em cada computador. A restrição D5 estabelece os domínios das variáveis *x*.

O IP-FULL-FUZZY é baseado em um problema de otimização *fuzzy*. Ele usa números *fuzzy* triangulares para modelar os valores das demandas computacionais e de comunicação das aplicações, capacidade de processamento dos computadores e largura de banda disponível nos enlaces da rede. Esses valores são dados como intervalos [*min*,*max*]. O conjunto *fuzzy* é então dado pelo par ([*min*,*max*],μ), e cada ponto $x \in \mathbb{R}$ é associado a um valor da função de pertinência, $\mu(x) \in [0,1]$, que representa o grau de certeza de que um determinado ponto pertence ao conjunto *fuzzy*. Seja $c = (min+max)/2$, então os números *fuzzy* usados no escalonador são do tipo [*min*,*c*,*max*], sendo que:

$$\mu(x) = \begin{cases} 0 & ,\text{para } x < min \\ \frac{x-min}{c-min} & ,\text{para } min \leq x \leq c \\ \frac{x-max}{c-max} & ,\text{para } c < x \leq max \\ 0 & ,\text{para } x > max \end{cases} \quad (1)$$

Os números *fuzzy* triangulares são obtidos para se considerar os graus de incerteza fornecidos como entrada para o escalonador: σ é o grau de incerteza associado às tarefas, ρ é o grau de incerteza associado às dependências, χ é o grau de incerteza associado aos computadores e ω é o grau de incerteza associado aos enlaces. Assim, a quantidade de instruções da tarefa i é dada por $\widetilde{I_i} = [I_i, I_i, \overline{I_i}]$, onde $\underline{I_i} = I_i(1 - \frac{\sigma}{100})$ e $\overline{I_i} = I_i(1 + \frac{\sigma}{100})$. A quantidade de *bits* da dependência *ij* é dada por $\widetilde{B_{i,j}} = [\underline{B_{i,j}}, B_{i,j}, \overline{B_{i,j}}]$, a qual é computada de forma análoga. A capacidade de processamento do computador *k* é dada por $\widetilde{TI_k} = [\underline{TI_k}, TI_k, \overline{TI_k}]$ e a largura de banda disponível entre os computadores *k* e *l* é dada por $\widetilde{TB_{k,l}} = [\underline{TB_{k,l}}, TB_{k,l}, \overline{TB_{k,l}}]$.

O problema de otimização *fuzzy* é também modelado como um problema de programação linear inteira, o qual é obtido ao se transformar as restrições *fuzzy* nas restrições puras correspondentes [13]. Nessa transformação, T''_{max} é o valor de tempo máximo, dado por $T''_{max} = T_{max}(1 + \frac{\sigma}{100})(1 + \frac{\chi}{100})$. Na formulação adotada nos experimentos, denominada de IP-FULL-FUZZY-2, uma nova heurística é usada para encontrar o valor de T''_{max}, a fim de diminuir a complexidade do problema de programação linear inteira. Isso faz com que a resolução do problema seja mais rápida e permita valores de discretização menores. Assim, o novo T''_{max} é dado por $T''_{max} = T_{max} + T_f$, onde T_f é o maior tempo de término dentre as tarefas já executadas, e é computado no processo de reescalonamento. Se o valor de T''_{max} conduz a um problema inviável, então o reescalonamento é produzido novamente com o novo valor de T''_{max} dado por $T''_{max} + T_f$. Esse processo é repetido até uma solução viável ser obtida. Isso não resulta em um acréscimo significativo no tempo gasto para produzir os escalonamentos nas etapas de reescalonamento, visto que a resposta retornada pela biblioteca de otimização é muito mais rápida quando o problema é inviável do que quando ele não é. T''_{min} corresponde ao menor tempo de execução da aplicação e é igual ao tempo gasto executando-se, no computador mais rápido, as instruções das tarefas pertencentes ao caminho mais curto (*SP*) entre a tarefa 0 e a tarefa n-1. Assim, T''_{min} é dado por $T''_{min} = min(\{TI_{k|k \in V_H}\}) \times \sum_{i \in SP} I_i$. $\lambda \in [0,1]$ corresponde ao grau de satisfação do escalonamento proposto e é inversamente proporcional ao *makespan* obtido, representado por f_n. Devido ao limite de páginas, são apresentadas somente as restrições que diferem das utilizadas pelo IPDT-2.

A formulação adotada considera que a capacidade de computadores e enlaces pode somente diminuir, o que é coerente, pois objetiva-se a produção de escalonamentos robustos no pior caso. Além disso, a consideração de alta disponibilidade de recursos pode ser perigosa, dado que conduz a um desempenho ruim quando tal pressuposto é falso. A formulação difere da formulação do IPDT-2 no que se refere: à função objetivo (maximização de λ); à adição da restrição F1, que estabelece uma relação entre λ e f_n (o tempo de término da última tarefa); e à restrição F4, a qual considera o tempo máximo de processamento e transmissão dos dados, prevenindo a tarefa de ser executada antes de seus predecessores.

A nova restrição e a restrição modificada são mostradas a seguir:

$$f_n \leq (1 - \lambda)T''_{max} + \lambda T''_{min}; \quad \text{(F1)}$$

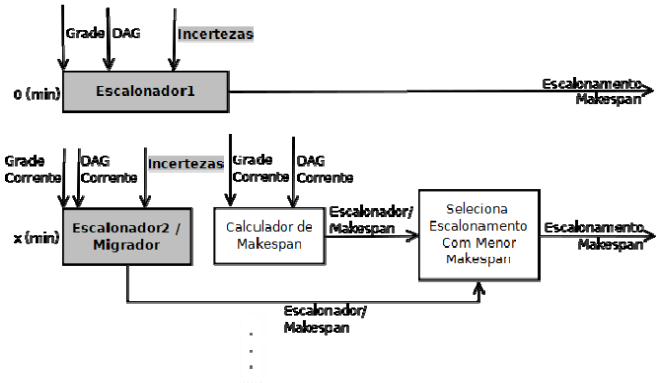
$$\sum_{k \in \delta(l)} \sum_{s=1}^{t - \lceil \frac{I_j TI_l + \overline{B_{i,j}} TB_{k,l}}{} \rceil} x_{i,s,k} \geq \sum_{s=1}^{t} x_{j,s,l}$$
$$\text{para } i, j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad l \in \mathcal{H}, \quad t \in \mathcal{T}; \quad \text{(F4)}$$

B. Mecanismos

Três novos mecanismos são propostos. Eles diferem pelos escalonadores usados em cada fase de sua operação. O mecanismo híbrido1 usa o escalonador IP-FULL-FUZZY-2 para produzir o escalonamento inicial e os escalonamentos nas etapas de reescalonamento. O mecanismo híbrido2 usa o IP-FULL-FUZZY-2 para produzir o escalonamento inicial e o IPDT-2 nas etapas de reescalonamento. O mecanismo híbrido3 usa o IPDT-2 para produzir o escalonamento inicial e o IP-FULL-FUZZY-2 nas etapas de reescalonamento.

O modo de operação dos mecanismos híbridos é ilustrado na Fig. 1. A diferença entre os mecanismos está no escalonador usado no escalonamento inicial e no escalonador usado nas etapas de reescalonamento. O escalonamento inicial é produzido e, periodicamente, o processo de reescalonamento é executado. Nas etapas de reescalonamento, cada tarefa que já iniciou sua execução, e ainda não finalizou, é dividida em duas: uma contendo as instruções já executadas, e a outra

contendo as instruções restantes. O resultado da divisão de uma tarefa é ilustrado na Fig. 2, onde a tarefa *T*5, que é executada no computador h1, foi dividida nas tarefas *T*5′ e *T*5″, com *T*5′ contendo as instruções já executadas e sendo fixada para rodar em h1, e *T*5″ contendo as instruções a serem executadas em um novo computador, caso o restante de *T*5 seja reescalonado para um computador diferente de h1. O arco antes direcionado de *T*5 para *T*8 está agora direcionado de *T*5″ para *T*8. O arco entre *T*5′ e *T*5″ tem peso correspondente à quantidade de dados a serem transferidos de h1 para o novo computador caso ocorra a migração. O escalonamento é produzido para o novo DAG originado desse processo de divisão das tarefas já iniciadas e não concluídas. As novas tarefas geradas pelo processo de divisão são migradas somente se a migração conduz a um *makespan* menor do que o estimado para o escalonamento atual.



Mecanismo	Escalonador1	Escalonador2
Híbrido1	IP-FULL-FUZZY-2	IP-FULL-FUZZY-2
Híbrido2	IP-FULL-FUZZY-2	IPDT-2
Híbrido3	IPDT-2	IP-FULL-FUZZY-2

Figura 1. Mecanismos híbridos

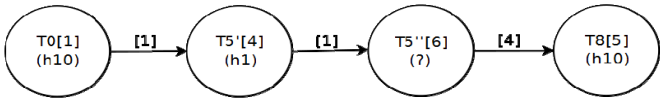


Figura 2. Divisão de uma tarefa para o processo de reescalonamento.

O *makespan* do escalonamento atual é computado pelo componente "Calculador de *Makespan*", considerando-se as instruções restantes, de todas as tarefas, a serem executadas no novo estado da grade. É importante observar que o cálculo do *makespan*, quando ocorrem migrações, considera o tempo gasto na transferência dos dados e na migração das tarefas.

Além dos três mecanismos propostos, três outros foram avaliados: o reativo, que usa o IPDT-2 para produzir o escalonamento inicial e os escalonamentos nas etapas de reescalonamento; o pró-ativo, que usa o IP-FULL-FUZZY-2 para produzir um único escalonamento; e o não-pró-ativo, que usa um único escalonamento dado pelo IPDT-2. A comparação dos seis mecanismos considera diferentes níveis de flutuação. Um nível de flutuação corresponde à porcentagem máxima de variação do valor de um determinado recurso. As seguintes métricas foram avaliadas: *makespan*, número médio de migrações e demanda computacional dos escalonadores, que inclui o tempo total de processamento gasto nos processos de escalonamento e reescalonamento. O *makespan* é útil para analisar a qualidade dos escalonamentos e migrações produzidos. O número médio de migrações é útil para analisar a interferência na rede causada pelos mecanismos. A demanda computacional dos escalonadores é útil para analisar se os escalonamentos e migrações produzidos podem ser obtidos em um tempo viável em relação ao tempo requerido para a execução da aplicação.

III. ANÁLISE DE DESEMPENHO

Simulações foram conduzidas para comparar o desempenho dos mecanismos híbridos com os mecanismos pró-ativo, reativo e não-pró-ativo. Um programa foi escrito em C para simular o funcionamento dos seis mecanismos. Os escalonadores (IPDT-2 e IP-FULL-FUZZY-2) foram implementados usando bibliotecas da *Fico Xpress Optimization Suite 7* [14], com suporte a *multi-core*. Todos os experimentos foram executados em um computador equipado com um *Intel Xeon* 2.27GHz 64-*bit Quad-core*, 6GB de RAM e rodando o sistema operacional *Debian squeeze*.

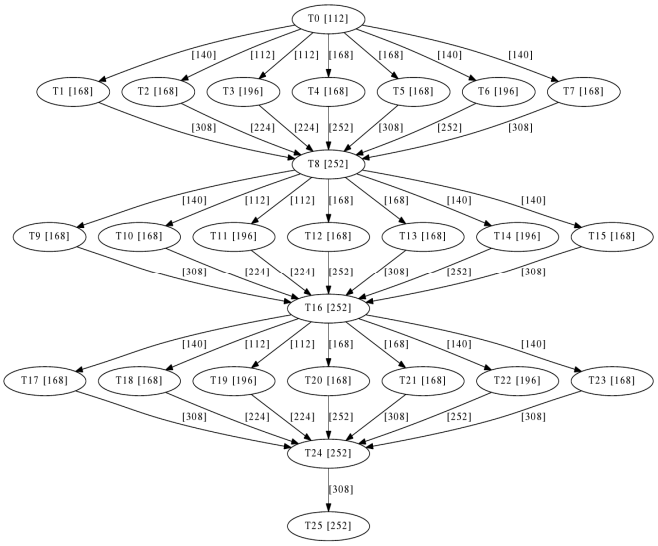


Figura 3. Aplicação simulada.

As mesmas entradas foram fornecidas aos seis mecanismos. O DAG da aplicação simulada é mostrado na Fig. 3. Ele é baseado em uma aplicação de química quântica denominada de WIEN2K [15], desenvolvida na Universidade Tecnológica de Viena [16] (*Vienna University of Technology*). Os pesos das tarefas na Fig. 3 estão na faixa [112,252], em uma escala de 10^5 milhões de instruções, e os pesos associados aos dados das dependências estão na faixa [112,308], em uma escala de 10^6 *bits*. As topologias das grades simuladas foram geradas pelo software BRITE [17], utilizando-se o modelo *Barabasi-Albert* [18], um modelo usado para gerar topologias de rede similares àquelas encontradas na *Internet*. Os valores de largura de banda disponíveis foram obtidos aleatoriamente de

uma distribuição uniforme no intervalo [10,1024]Mbps, e a capacidade de processamento dos computadores foram obtidos aleatoriamente de uma distribuição uniforme no intervalo [840,3300]MIPS. Uma grade com 75 computadores foi simulada. As incertezas projetadas dadas como entrada para o escalonador IP-FULL-FUZZY-2 variam entre 0 e 200% e a mesma faixa foi usada para simular as flutuações na disponibilidade dos recursos durante a execução da aplicação. Foram simuladas flutuações na capacidade de processamento dos computadores e na disponibilidade da largura de banda dos enlaces. Os valores dos pesos de nós e arcos no grafo da grade foram aumentados, correspondendo a um decréscimo na disponibilidade dos recursos.

Três métricas foram avaliadas: o *makespan* da aplicação, o número médio de migrações e a demanda computacional dos mecanismos. Cada ponto, nos gráficos apresentados nas próximas subseções, é uma média de 20 execuções derivada da produção de intervalo de confiança com nível de confiança de 95%, usando-se o método da replicação. Cada valor no eixo horizontal x representa o acréscimo dos valores dos pesos do grafo da grade a partir de uma distribuição uniforme no intervalo [0,x%].

Subseção III.A apresenta os resultados para o *makespan*, Subseção III.B apresenta os resultados para o número médio de migrações e Subseção III.C apresenta os resultados para as demandas computacionais dos mecanismos.

A. *Makespan*

A Fig. 4 plota o *makespan* obtido pelos mecanismos. É possível ver que há uma tendência de acréscimo do *makespan* com o aumento do nível de flutuação. No geral, o mecanismo pró-ativo não possui melhor desempenho que os mecanismos híbridos e reativos. A exceção ocorreu para os níveis de flutuação de 75%, 125% e 200% (mas somente o nível de flutuação 200% é mostrado na Fig. 4), para os quais o mecanismo pró-ativo obteve melhor desempenho que os outros mecanismos. Além disso, a comparação entre os mecanismos pró-ativo e reativo demonstra que a efetividade do mecanismo pró-ativo depende das incertezas projetadas fornecidas como entrada. Isso torna possível que o mecanismo não-pró-ativo forneça menor *makespan* que o mecanismo pró-ativo (ambos não produzem migrações). Por exemplo, isso pode ser observado para os níveis de flutuação de 50% e 100% na Fig. 4, para os quais as incertezas projetadas para a capacidade de processamento e largura de banda foram de 50% e 100%, respectivamente.

Os mecanismos híbridos e o reativo mostraram, no geral, *makespans* bastante similares. Pela comparação dos escalonamentos dados pelos mecanismos híbrido1 e híbrido2 (ambos usam o IP-FULL-FUZZY-2 como escalonador inicial) com aqueles dados pelo mecanismo pró-ativo, é possível ver a importância das migrações e do reescalonamento. Por exemplo, pode ser observado que, para níveis de flutuação de 50% e 100%, as migrações realizadas por esses mecanismos diminuíram o *makespan*. Os resultados obtidos para os mecanismos híbrido1 e híbrido2 foram similares. As exceções aconteceram quando não houve flutuação (quando o híbrido2 produziu melhor resultado que o híbrido1). Da mesma forma, os resultados obtidos pelos mecanismos híbrido3 e reativo foram similares, exceto quando não houve flutuação.

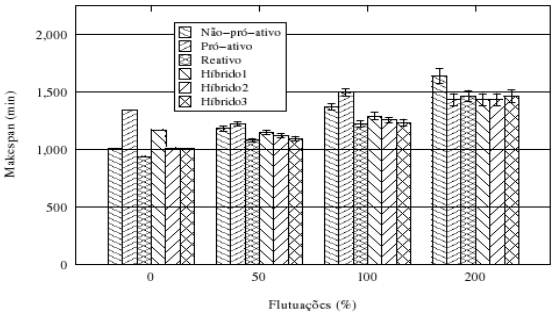


Figura 4. *Makespans* obtidos pelos mecanismos quando houve flutuações na capacidade de processamento dos computadores e na largura de banda disponível dos enlaces, para grades com 75 computadores.

B. Número Médio de Migrações

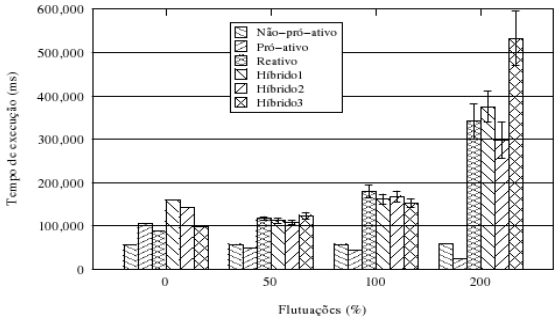
A TABELA I mostra o número médio de migrações obtidos pelos mecanismos. Os números em negrito representam o menor número médio de migrações para cada nível de flutuação. Os mecanismos não-pró-ativo e pró-ativo não são apresentados na tabela porque eles não realizam migração de tarefas. No geral, nenhuma tendência pôde ser observada em relação ao número de migrações produzidas pelos mecanismos que utilizam o mesmo escalonador inicial, ou em relação ao número de migrações produzidas pelos mecanismos que utilizam o mesmo escalonador nas etapas de reescalonamento. Entretanto, pode ser observado que o IP-FULL-FUZZY-2 produziu um bom escalonamento inicial quando as incertezas projetadas foram de 200% para a capacidade de processamento e para a largura de banda disponível, o que levou a um número baixo de migrações. O número médio de migrações foi o mais baixo quando se utilizou os mecanismos híbrido1 e híbrido2 para níveis de flutuação de 200%, como pode ser visto na Tabela I. Isso sugere que a melhor forma de lidar com o comportamento instável do IP-FULL-FUZZY-2 ocorre quando a entrada refere-se às projeções de incerteza mais adequadas.

TABELA I
NÚMERO MÉDIO DE MIGRAÇÕES OBTIDO QUANDO HOUVE FLUTUAÇÕES NA CAPACIDADE DE PROCESSAMENTO E NA LARGURA DE BANDA DISPONÍVEIS, PARA GRADES COM 75 COMPUTADORES.

Mecanismos	**0%**	**50%**	**100%**	**125%**	**150%**	**175%**	**200 %**
Híbrido1	7.00	4.00	4.75	1.75	4.80	4.60	0.10
Híbrido2	1.00	4.30	3.45	4.40	4.85	5.45	0.00
Híbrido3	0.00	2.90	4.90	2.90	3.15	2.95	2.80
Reativo	1.00	1.85	4.35	4.70	4.85	4.05	5.00

C. Demanda Computacional

A Fig. 5 plota a demanda computacional (o tempo gasto para produzir o escalonamento inicial e os escalonamentos nas etapas de reescalonamento) dos mecanismos. É possível observar que os mecanismos que não produzem migrações demandam menos tempo, como seria de se esperar, visto que executam o escalonador apenas uma vez. Entre os mecanismos que produzem migrações, o mecanismo reativo demonstrou o comportamento mais estável, com uma tendência para o aumento da demanda computacional com o aumento do nível de flutuação. Os outros mecanismos, que usam o IP-FULL-FUZZY-2 em uma de suas etapas, também demonstraram tendência para aumento da demanda computacional com o aumento do nível de flutuações. Entretanto, os mecanismos híbrido1 e híbrido2 demonstraram uma tendência oposta para níveis de flutuação acima de 50%. Isso pode ser explicado ao se observar o desempenho do



mecanismo pró-ativo, visto que quanto maior foi a incerteza projetada fornecida como entrada para o IP-FULL-FUZZY-2, menor foi a demanda computacional do escalonamento inicial.

Figura 5. Demanda computacional dos mecanismos quando houve incertezas projetadas para a capacidade de processamento e para a largura de banda disponível, para grades com 75 computadores.

Os resultados obtidos pelos mecanismos híbridos são bastante similares, exceto para níveis de flutuação de 200%. Nesse caso, o híbrido3 foi o que demandou mais, seguido pelo híbrido1 e pelo híbrido2. Isso pode ser explicado pelo fato de, no geral, o IP-FULL-FUZZY-2 necessitar de valores maiores do T_{max} para produzir soluções viáveis do que o IPDT-2. Assim, a complexidade dos problemas resolvidos pelo híbrido3 e pelo híbrido1 foi maior que a requerida pelo híbrido2. Além disso, a diferença entre o híbrido1 e o híbrido3 pode ser explicada pelo fato do escalonamento inicial produzido pelo IP-FULL-FUZZY-2, com incertezas projetadas de 200%, ter um *makespan* menor que aquele produzido pelo IPDT-2, o que resultou em um número menor de etapas de reescalonamento para o híbrido1 em relação ao híbrido3.

IV. CONCLUSÃO

Mecanismos que adotam abordagens híbridas para o escalonamento das tarefas da aplicação em grades computacionais podem se beneficiar das qualidades de ambas as abordagens, a reativa e a pró-ativa. Nesse artigo, são propostos três novos mecanismos que diferem pelo escalonador usado para produzir o escalonamento inicial e para produzir os escalonamentos nas etapas de reescalonamento. Adicionalmente, dois novos escalonadores foram propostos, um pró-ativo e um não-pró-ativo, que são usados pelos mecanismos.

Os resultados apresentados na seção prévia mostraram que a eficiência do IP-FULL-FUZZY-2 pode estar fortemente associada às incertezas projetadas fornecidas como entrada. Parece mais adequado considerar incertezas projetadas de 200%, visto que foram obtidos melhores desempenhos quando esse valor foi usado. Os resultados são consistentes com aqueles encontrados em [10]. Assim, é uma boa política usar mecanismos híbridos que tenham o IP-FULL-FUZZY-2 como escalonador inicial. O uso do híbrido2 pode ser uma boa escolha se o tempo para produção dos escalonamentos nas etapas de reescalonamento é importante, visto que o IPDT-2 demanda um tempo menor que o demandado pelo IP-FULL-FUZZY-2.

Como trabalho futuro, recomenda-se a identificação do critério de viabilidade para os problemas de programação linear inteira do IP-FULL-FUZZY-2, visando evitar a necessidade de computação de novos escalonamentos quando o IP-FULL-FUZZY-2 fornece soluções inviáveis.

REFERÊNCIAS

[1] J. M. Schopf, "Ten Actions when Grid Scheduling," in Grid Resource Management: State of the Art and Future Trends, Jarek Nabrzyski and Jennifer M. Schopf and Jan Weglarz, Ed. Springer, 2003, pp. 15–23.

[2] D. M. Batista and N. L. S. da Fonseca, "A Survey of Self-adaptive Grids," IEEE Communications Magazine, vol. 48, no. 7, pp. 94–100, Jul 2010.

[3] D. M. Batista, N. L. S. da Fonseca, F. Granelli, and D. Kliazovich, "Self-adjusting grid networks," in In Proc of IEEE International Conference on Communications 2007 (ICC 2007).

[4] D. M. Batista, N. L. S. da Fonseca, F. K. Miyazawa, and F. Granelli, "Selfadjustment of Resource Allocation for Grid Applications," Computer Networks, vol. 52, no. 9, pp. 1762–1781, Jun 2008.

[5] S. S. Vadhiyar and J. J. Dongarra, "A Performance Oriented Migration Framework for the Grid," in Proceedings of CCGRID'03, May 2003, pp. 130–137.

[6] M. Imran, I. A. Niaz, S. Haider, N. Hussain, and M. A. Ansari, "Towards Optimal Fault Tolerant Scheduling in Computational Grid," in Proceedings of the ICET 2007, Nov 2007, pp. 154–159.

[7] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive Process-Level Live Migration in HPC Environments," in Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008, pp. 43:1–43:12. 14

[8] C. Fayad, J. Garibaldi, and D. Ouelhadj, "Fuzzy grid scheduling using tabu search," in Fuzzy Systems Conference, july 2007, pp. 1–6.

[9] I. Gonzalez-Rodriguez, C. Vela, and J. Puente, "A memetic approach to fuzzy job shop based on expectation model," in Fuzzy Systems Conference, 2007. FUZZIEEE 2007. IEEE International, july 2007, pp. 1 –6.

[10] D. M. Batista and N. L. S. da Fonseca, "Robust scheduler for grid networks under uncertainties of both application demands and resource availability," Comput. Netw, volume 55, pages 3-19, 2011.
[11] D. M. Batista, A. C. Drummond, and N. L. S. da Fonseca, "Robust scheduler for grid networks," in Proceedings of the 2009 ACM symposium on Applied Computing, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 35–39. [Online]. Available: http://doi.acm.org/10.1145/1529282.1529289
[12] D. M. Batista, N. L. S. da Fonseca, and F. K. Miyazawa, "A set of schedulers for grid networks," in Proceedings of the 2007 ACM symposium on Applied computing, ser. SAC '07, 2007, pp. 209–213.
[13] H.-J. Zimmermann, Fuzzy set theory — and its applications, 4th ed. Norwell, MA, USA: Kluwer Academic Publishers, 1996, pp. 11–13;336–347.
[14] FICO, "FICO Xpress Optimization Suite 7," 2010.
[15] P. Blaha and K. Schwarz and G. Madsen and D. Kvasnicka and J. Luitz,"WIEN2K," 2011.
[16] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of scientific workflows in the askalon grid environment," SIGMOD Rec., vol. 34, pp. 56– 62, September 2005.
[17] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems. Washington, DC, USA: IEEE Computer Society, 2001, p. 346.
[18] A. L. Barabasi and R. Albert. "Emergence of scaling in random networks", Science, volume 286, pages 509-512, 1999.

Rafael Lima Curi possui o título de Bacharel em Ciência da Computação pela Universidade Estadual de Campinas (2009). Atualmente é aluno de Mestrado pela Universidade Estadual de Campinas. Suas principais áreas de interesse são grades computacionais, computação em nuvens e teoria dos jogos aplicada a problemas de roteamento.

Daniel Macêdo Batista possui graduação em Ciência da Computação pela Universidade Federal da Bahia e mestrado e doutorado em Ciência da Computação pela Unicamp. Atualmente ele é professor doutor da Universidade de São Paulo. Tem experiência na área de Ciência da Computação, com ênfase em redes de computadores, atuando principalmente nos seguintes temas: análise de desempenho, engenharia de tráfego, grades, nuvens e virtualização.

Nelson Luis Saldanha da Fonseca possui graduação em Engenharia Elétrica com ênfase em sistemas pela Pontifícia Universidade Católica do Rio de Janeiro (1984), mestrado em Informática pela PUC-Rio (1987), mestrado em Master In Computer Engineering - University of Southern California (1993) e PhD in Computer Engineering - University of Southern California (1994). Obteve o título de Livre Docente em Redes de Computadores pela Universidade Estadual de Campinas (Unicamp) em 1999. É professor titular da Universidade Estadual de Campinas, onde, é, atualmente, chefe do departamento de Sistemas de Computação.