Escalonamento de Tarefas em Grades Computacionais guiado pelo Consumo de Energia com os Algoritmos Easy Backfilling e HEFT

Maurício Pillon[1], **Tathiana do Amarante**[1], **Guilherme Koslovski**[1], **Charles Miers**[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC)
Joinville, SC – Brasil

{mauricio.pillon,guilherme.koslovski,charles.miers}@udesc.br

tathiana.duarte@edu.udesc.br

Abstract. *Grid computing, dynamic and flexible computational infrastructures managed using Resource Management Systems (RMS), have been employed for more than two decades in hosting High Performance Computing (HPC) applications. The scheduling algorithms used in computational grids usually prioritize the provision of the infrastructure in favor of the performance of the applications, mainly ignoring approaches which takes into account energy saving. However, with the ecological and financial appeal, some initiatives for the conscious and efficient use of energy have been created. Among them, the proposal of green heuristics which consider energy consumption and application performance (execution time) in grid computing environments. We adapted the traditional algorithms EASY-Backfilling and HEFT for scheduling tasks in computational grids using a set of predefined heuristics. The principles of filling time gaps and overlapping tasks on a single resource are the foundation of our proposal. Our results using the Batsim simulation environment revealed a reduction of energy consumption between* 2% *and* 49% *and, at task execution time, between* 1% *and* 32%*.*

Resumo. *As grades computacionais, ambientes com infraestruturas dinâmicas e flexíveis gerenciadas por Resource Management Systems (RMS), são empregadas há mais de duas décadas na hospedagem de aplicações High Performance Computing (HPC). Os algoritmos de escalonamento empregados em grades computacionais priorizam o fornecimento da infraestrutura em favor do desempenho das aplicações sem, necessariamente, analisar alternativas de economia de energia. Entretanto, com o apelo ecológico e financeiro, algumas iniciativas para uso consciente e eficiente de energia têm sido criadas, dentre elas a proposta de heurísticas verde, que ponderam o consumo de energia e o desempenho das aplicações (tempo de execução) em ambientes de grades computacionais. No presente trabalho, os algoritmos tradicionalmente utilizados para escalonamento de tarefas em grades computacionais EASY-Backfilling e HEFT foram adaptados com as heurísticas pré-definidas. Os princípios de preenchimento de lacunas temporais e a sobreposição de tarefas em único recurso são os alicerces da proposta. Os resultados obtidos com o ambiente de simulação Batsim apresentaram redução do consumo de energia entre* 2% *e* 49% *e, no tempo de execução das tarefas, entre* 1% *e* 32%*.*

1. Introdução

Uma grade computacional é uma infraestrutura de *High Performance Computing* (HPC), caracterizada por viabilizar o compartilhamento de recursos de hardware e de software geograficamente distribuídos [Foster and Kesselman 2003, Buyya and Murshed 2002]. O ambiente de grade computacional se mantém atrativo há mais de duas décadas devido às características de gerenciamento dinâmico de recursos, heterogeneidade, escalabilidade e segurança. Nestes ambientes, o gerenciamento de recursos é de responsabilidade do *Resource Management System* (RMS), o qual está integrado a escalonadores de tarefas, *e.g.*, *Portable Batch System* (PBS)[1], OAR[2] e *Load Sharing Facility* (LSF)[3]. Os usuários de grades submetem as suas tarefas ao RMS especificando as necessidades de recursos para executar as suas aplicações, *e.g.*, número de servidores, o tipo dos servidores (*Symmetric Multiprocessing* (SMP), multicores, *hyperthreding*, *etc.*), a quantidade de memória, tecnologia de rede, o tempo máximo de execução da tarefa, o fabricante do processador, entre outras opções. Assim que a tarefa é escalonada, o RMS retorna um terminal de controle ao usuário, se a requisição for interativa, ou arquivos de saída e erro, se for uma execução em *batch*.

O crescimento das aplicações que necessitam de alto desempenho aliado com a recorrente necessidade de acurácia de informações elevou a carga computacional executada em grades. Consequentemente, o contingenciamento de recursos e a eventual ocorrência de gargalos de execução motivam pesquisadores a propor soluções alternativas de escalonamento de recursos [Foster and Kesselman 2003, De FranÇa 2014, Dutot et al. 2017, Torquato et al. 2018, Yadav et al. 2019]. Conciliar o menor tempo de espera, eficiência energética, desempenho das aplicações, prioridade das tarefas entre outras características, são desafios dos escalonadores de tarefas em grades computacionais.

O apelo ecológico e a busca por redução de custos de manutenção [Mämmelä et al. 2012] motivaram a elaboração de orientações e métricas relacionadas com a eficiência energética, cunhando o termo *Computação Verde* [Avelar et al. 2012]. Nas últimas décadas, o consumo de energia do setor de *Tecnologia da Informação* (TI) se destacou quando comparado com totalidade do consumo dos demais setores, atingindo a terceira posição em 2012, atrás somente do consumo total de países como a China e os Estados Unidos [Corcoran and Andrae 2013].

Como um ambiente focado em HPC, a prioridade dos escalonadores de tarefas em grades computacionais sempre foi o desempenho. Entretanto, estratégias para otimização do uso de recursos têm sido aplicadas na literatura especializada, *i.e.*, hibernação, redução da frequência das unidades de processamento, desativação de servidores, técnicas de escalonamento de tarefas [Mämmelä et al. 2012, Chawla and Saluja 2016, Kim et al. 2007, Tang et al. 2016, Teodoro et al. 2013, Watanabe et al. 2014, Rios et al. 2009, Carastan-Santos et al. 2019]. Porém, o comportamento de demanda por recursos em grades, representado pela taxa de chegada de tarefas, geralmente, não segue um único modelo de distribuição de carga, dificultando a previsão de demanda [Iosup et al. 2008, Iamnitchi et al. 2009,

[1]Disponível em `https://www.pbspro.org/`

[2]Disponível em `https://oar.imag.fr/`

[3]Disponível em `https://www.ibm.com/support/knowledgecenter/en/SSWRJV/product_welcome_spectrum_lsf.html`

Guazzone 2014]. Em determinados momentos, o número de tarefas supera a capacidade de recursos disponíveis e, consequentemente, a fila de tarefas aumenta, em outros momentos pode ocorrer de vários servidores estarem ociosos e assim desperdiçando energia. Equilibrar o conflito entre alto desempenho e desperdiço de energia é uma lacuna científica em evidência atualmente.

O presente trabalho tem como principal objetivo reduzir o consumo de energia de ambientes de grades computacionais e minimizar o impacto no desempenho das aplicações. O princípio da heurística aplicada aos escalonadores é o uso da técnica de sobreposição de tarefas. As principais contribuições deste trabalho são a aplicação da técnica de sobreposição de tarefas em prol da redução energética e a análise do impacto desta heurística com cargas reais. A proposta atua em cenários em que exista fila de tarefas aguardando recursos. A análise do escalonador com sobreposição de tarefas baseou-se na instrumentalização de dois algoritmos de escalonamento reconhecidos pela comunidade científica e utilizados por RMS de grades computacionais: *Heterogeneous-Earliest-Finish-Time* (HEFT) [Topcuoglu et al. 2002] e *Backfilling* (*Extensible Argonne Scheduling System* (EASY) [Lifka 1995] ou *Conservative backfilling* (CONS) [Mu'alem and Feitelson 2001]).

A escolha por simulação para analisar a eficiência da proposta mostrou-se adequada, uma vez que a comparação coerente dos resultados entre algoritmos exige o uso de uma mesma carga de trabalho, em dois ou mais cenários distintos, tornando o uso do ambiente real impróprio para este caso. Ademais, a simulação efetuada com o *Batsim* serve para analisar ambientes reais de sistemas distribuídos de larga escala possibilitando o controle sobre os experimentos e cargas [Dutot et al. 2015, Buyya and Murshed 2002]. Os resultados obtidos são promissores, mostrando que a sobreposição de tarefas atinge redução no consumo de energia de até 16% quando comparado com o mesmo algoritmo não modificado. A comparação das reduções entre os valores máximos e mínimos varia entre 2% a 49% para o consumo de energia e entre 1% e 32% para os tempos de execução.

O conteúdo deste trabalho está organizado como segue. A Seção 2 descreve os principais trabalhos correlatos e define métricas de comparações entre eles. A Seção 3 apresenta as características essenciais dos algoritmos de escalonamento de tarefas e a técnica de sobreposição, base da proposta. Na sequência, a Seção 4 descreve a proposta de heurística verde deste trabalho, seguida da descrição do protótipo implementado. Finalmente, as duas seções finais, têm a Análise Experimental (Seção 5) e as Considerações Finais acompanhada de trabalhos futuros (Seção 6).

2. Trabalhos Relacionados

No âmbito de grades computacionais e HPC, o presente trabalho trata os temas: algoritmos de escalonamento de tarefas e eficiência energética (Grades Verdes). O método de pesquisa utilizado no trabalho foi o de pesquisa bibliográfica sistemática baseado em três mecanismos de buscas: IEEE Explore, Scopus e ACM DL, com as chaves de busca: *("grid computing" ∧ "scheduling algorithms"∧ "energy efficiency"∧ "overlap")*. O contexto aplicado foi de plataformas para HPC (grades ou nuvens computacionais) delimitado ao período de publicação entre 2015 a 2019.

Algoritmos de escalonamento de tarefas têm por objetivo definir o melhor mapeamento entre tarefas e recursos computacionais hospedeiros, buscando minimizar um

ou mais critérios (*i.e.*, o tempo de espera na fila, o tempo total de execução e *makespan*). Como o problema de escalonamento de tarefas pertence a classe de complexidade *NP-difícil*, ambientes de produção em grades computacionais apoiam-se em algoritmos simples, com poucas variáveis, por necessitarem de respostas instantâneas para suas requisições de alocação. Assim, os algoritmos baseados no princípio de *backfilling* (*i.e.*, EASY [Lifka 1995] e CONS [Mu'alem and Feitelson 2001]) vêm se destacando em infraestruturas de grades computacionais.

A Tabela 1 apresenta alguns dos algoritmos de escalonamento de tarefas, seus objetivos e estratégias. Os algoritmos listados apresentam variações do princípio de *backfilling* [GÓmez-MartÍn et al. 2016, Carastan-Santos et al. 2019, Hasan and Goraya 2016, Gaussier et al. 2015, Lelong et al. 2017, Wang et al. 2018], de HEFT [Topcuoglu et al. 2002, Tang et al. 2016, Watanabe et al. 2014, de Carvalho et al. 2011], outras heurísticas de escalonamento [Wang et al. 2018, Ni 2015, Dakkak et al. 2016], aplicação de técnicas de aprendizado de máquina [Gaussier et al. 2015, Nepovinnykh and Radchenko 2016, Selvi and Manimegalai 2017] ou estratégicas de redução do consumo energético [Borro 2014, De FranÇa 2014, Wang et al. 2018, Kraemer et al. 2018]. Além dos trabalhos resumidos na tabela, destacam-se relevantes para esta pesquisa trabalhos que exploram a comparação, análise e discussões de métricas entre dois ou mais algoritmos de escalonamento de tarefas [Singh et al. 2016, Rath et al. 2018] e, que apresentam estratégias para redução do consumo de energia em ambientes computacionais [Zakarya 2018, Tang et al. 2016, Endo and et. al 2016, Nabi et al. 2016, You et al. 2017].

Paralela a discussão de escalonamento de tarefas em ambientes HPC aplicados a infraestruturas verdes, o reconhecido princípio de sobreposição tem sido estudado. Heurísticas baseadas em sobreposição apresentam resultados positivos quanto a otimização de fases do MapReduce [Zheng and Wu 2018], de restrições de janelas de tempo em linhas de produção [Wang et al. 2018] e escalonamento de operações [Demir and İsleyen 2014, Ismail 2012, Rios et al. 2009]. Os trabalhos estudados permitem identificar oportunidades de pesquisa com o relacionamento de energia, escalonamento e algoritmos de sobreposição.

Por fim, o levantamento bibliográfico não exaustivo demonstra uma lacuna científica, passível de exploração, com a aplicação da técnica de sobreposição no contexto de escalonamento de tarefas em grades computacionais verdes. Ambientes de grades computacionais verdes poderão se beneficiar com a redução do consumo de energia sem impactar no desempenho das aplicação de HPC que esta abriga.

3. Escalonamento de Tarefas

Algoritmos de escalonamento de tarefas em grades computacionais buscam associar requisições de alocações de recursos, advindo de usuários com perfis heterogêneos, a um conjunto de recursos computacionais igualmente heterogêneos. As requisições, mapeadas em tarefas, variam no que se refere a quantidade de recursos computacionais, no tempo necessário de uso, no tipo do recurso, entre outros. A infraestrutura, por sua vez, varia (tipo, modelo ou versão) do processador, da arquitetura, de interconexão de rede, de dispositivo de armazenamento, sistemas operacionais, bibliotecas, aplicações, entre ou-

Tabela 1. Algoritmos de escalonamentos de tarefas em ambientes de HPC.

Trabalho/Algoritmo	Objetivo	Estratégia
Carastan-Santos et. al. [Carastan-Santos et al. 2019]	Estudo de caso de potenciais ganhos de desempenho e redução energética de algoritmos simples (*i.e.*, SAF, SPF, SQF) quando comparados com EASY.	*Backfilling*
Kraemer et al. [Kraemer et al. 2018]	Algoritmo de escalonamento que reduz o número de violações no tempo de resposta sem interferir no tempo de execução das tarefas HPC.	*Backfilling* e migração
Fattened backfilling [GÓmez-MartÍn et al. 2016]	Melhorar o reaproveitamento do agendamento de lacunas.	*Backfilling*
Cooperative Computing System [Hasan and Goraya 2016]	Aplicar o princípio de *task backfilling* em computação em nuvem.	*Backfilling*
Gausser et Al. [Gaussier et al. 2015]	Elaboração de uma função de custo aplicada ao algoritmo EASY através de aprendizado de máquina.	*Backfilling* e aprendizado de máquina
Wang et al. [Wang et al. 2018]	Algoritmo de escalonamento orientado a energia.	*Backfilling*
Tuning EASY-Backfilling [Lelong et al. 2017]	Dedica-se a aplicar técnicas de otimização nas filas do algoritmo de escalonamento EASY-*Backfilling*.	EASY
HEFT e CPOP [Topcuoglu et al. 2002]	Atuam em ambientes com processadores heterogêneos e têm objetivo simultâneo de alto desempenho e escalonamento rápido.	Lista de prioridades
DEWTS [Tang et al. 2016]	Baseia-se no algoritmo de HEFT para aplicar a técnica de *Dynamic voltage and frequency scaling* (DVFS)	HEFT e DVFS
PowerHEFTLookhead, *HEFT-DAVM* e *Task Clustering*[Watanabe et al. 2014]	Proposta de três novos algoritmos baseados no HEFT para escalonamento de tarefas em nuvem computacional verde.	HEFT
HGreen [de Carvalho et al. 2011]	Priorizar a eficiência energética dos recursos explorando os perfis do fluxo de execução das aplicações.	HEFT e *Max-Min*
Nepovinnykh et. al. [Nepovinnykh and Radchenko 2016]	Apoia-se no algoritmo HEFT e aprendizado de máquina para escalonamento de tarefas em nuvem computacional.	HEFT e aprendizado de máquina
Hybrid Two PHase VNS [Selvi and Manimegalai 2017]	Aplica busca por vizinhança no escalonamento de tarefas em grades.	*Variable Neighborhood Search*
Maximum Regret e *Greedy* [Borro 2014]	Algoritmos de escalonamento que busca minimizar o consumo de energia para grades móveis.	Adaptações dos algortimos *Maximum Regret* e *Greedy*
GRASP Verde [De FranÇa 2014]	Aplicação da metaheurística GRASP no escalonamento de tareafas em grades computacionais com foco em redução do consumo de energia.	GRASP
Tianwei [Ni 2015]	Considerar a fragmentação de recursos em grades computacionais	Distribuição e redistribuição de tarefas.
Swift Gap [Dakkak et al. 2016]	Considerar a perspectiva do usuário.	*Best Gap* e *Tabu Search*

tros. Além disso, a demanda (taxa de requisições de tarefas), normalmente, não segue um comportamento padrão facilmente modelável [Iosup et al. 2008, Guazzone 2014]. Neste ambiente, o desafio de conciliar as requisições, com suas necessidades e restrições, aos recursos é do algoritmo de escalonamento de tarefas. Considerar todos os fatores citados requer uma ampla análise do espaço de busca (servidores candidatos) perante aos objetivos e configurações informados pelos usuários. Diante do exposto, desde o surgimento de grades computacionais, os algoritmos de escalonamento de tarefas usados em ambientes reais de grande porte têm apoiado-se em heurísticas baseadas em poucas informações, porém com tempo de resposta curtos. Dois dos principais algoritmos com estas características são o *Extensible Argonne Scheduling System* (EASY) e o *Heterogeneous-Earliest-Finish-Time* (HEFT). A compreensão destes algoritmos é essencial ao entendimento da proposta deste trabalho, descrita na seção 4.

3.1. *Extensible Argonne Scheduling System* (EASY)

O algoritmo de escalonamento EASY-*Backfilling* é amplamente empregado em grades computacionais devido a sua simplicidade. Ele trabalha com o princípio de fila de chegada, inspirado na política do *First Come First Serve* (FCFS), escalonando tarefas em tempos de respostas curtos. De posse de um conjunto de recursos, a política de escalonamento do EASY é enfileirar as requisições dos usuários de acordo com a disponibilidade dos servidores e ordem de chegada [Lifka 1995, Mu'alem and Feitelson 2001].

O escalonamento com EASY pode ser modelado em duas dimensões, atribuindo ao eixo x a evolução do tempo e ao eixo y a configuração e disponibilidade dos recursos. Na Figura 1, pode-se observar o conjunto de recursos *versus* o tempo decorrido. As tarefas em execução são representadas por áreas brancas. Neste exemplo, tem-se no tempo t_0, 3 processadores ocupados por 3 tarefas distintas, que finalizam suas execuções em t_1, t_2 e t_3, respectivamente.

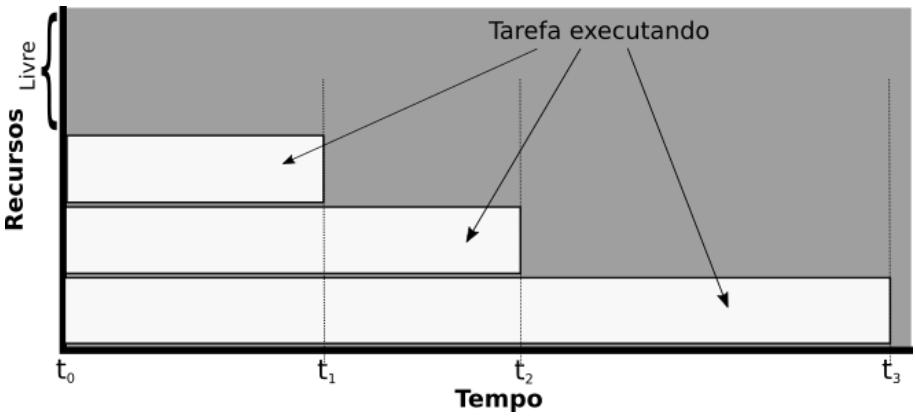

Figura 1. Representação do mapeamento de tarefas *versus* tempo. Adaptado de [Mu'alem and Feitelson 2001].

A medida que as requisições deixam de ser atendidas por falta de recursos disponíveis, o EASY passa a aplicar o enfileiramento das tarefas e, consequentemente, lacunas podem ser formadas ao longo do tempo. A formação de lacunas de ociosidade temporais são indesejáveis, visto que podem gerar dilatação no tempo de atendimento das requisições e desperdício de utilização dos recursos. Neste ponto que o EASY aplica

o princípio de *backfilling*, buscando preencher as lacunas temporais antecipando tarefas previamente enfileiradas. Dois métodos de preenchimentos de lacunas são ilustrados nas Figuras 2 e 3. Na Figura 2 é utilizado o algoritmo original EASY-*Backfilling*, enquanto na Figura 3, é aplicado o algoritmo nomeado *Conservative backfilling* (CONS).

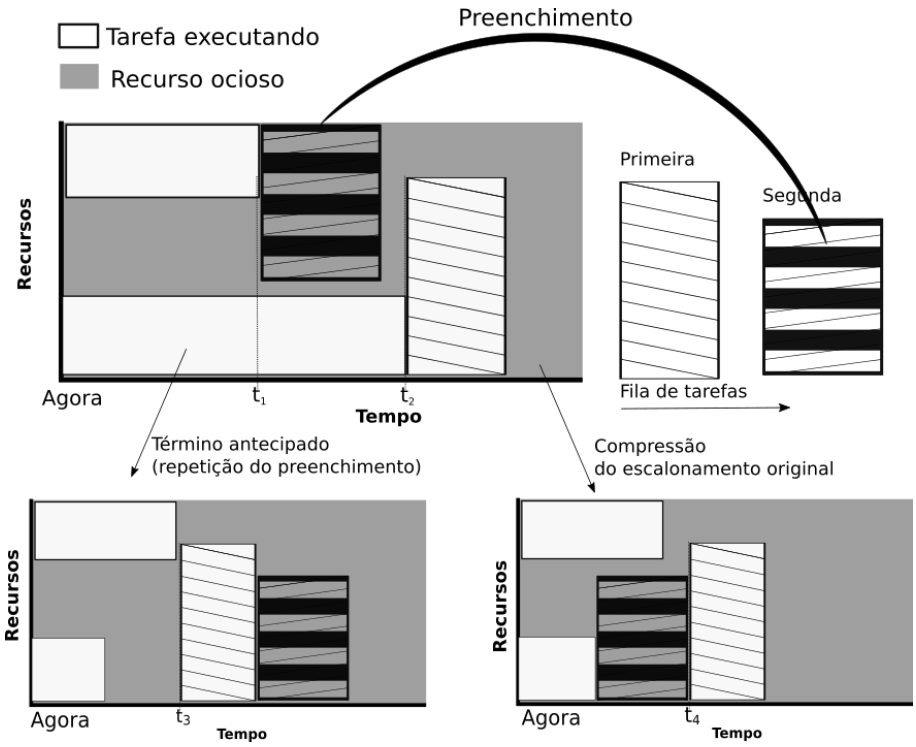

Figura 2. EASY-*Backfilling*. Adaptado de [Mu'alem and Feitelson 2001].

Como pode ser observado na Figura 2, o algoritmo retira as tarefas enfileiradas da fila (na ordem de chegada - FCFS) e agenda suas execuções na primeira faixa de recursos disponíveis no tempo. No exemplo, a primeira tarefa foi agendada para o t_2 e a segunda para o t_1. Observa-se que o algoritmo respeitou a ordem de chegada para o agendamento, mas por conveniência e arranjo de disponibilidade de recursos, a segunda tarefa foi escalonada para executar antes da primeira. Porém, antes do início da execução da segunda tarefa (escalonada para t_1), uma requisição teve seu término antecipado, liberando os recursos computacionais. Neste momento, o método de preenchimento do EASY-*Backfilling* analisa os agendamentos previamente executados, apresentando um novo arranjo de execuções seguindo a ordem de chegada das tarefas. Posteriormente, antes do lançamento das tarefas agendadas, o EASY-*Backfilling* identifica que a inversão na ordem das requisições reduz o *makespan* e, finalmente, chega no estado descrito no último estado do exemplo da Figura 2.

Por sua vez, o algoritmo *Conservative backfilling* (CONS) acrescenta uma única restrição ao preenchimento de lacunas, uma vez encontrado uma lacuna de agendamento para a primeira tarefa da fila, esta não pode mais ser adiada. Esta característica é demonstrada no exemplo da Figura 3, na qual a primeira tarefa possui o indicativo de uma âncora no eixo x (tempo) [Lelong et al. 2017, Mu'alem and Feitelson 2001]. Em virtude da simplicidade da modelagem e abrangência de uso em infraestruturas de grades computacionais, foi escolhida a instrumentalização do algoritmo de *Backfilling* original na

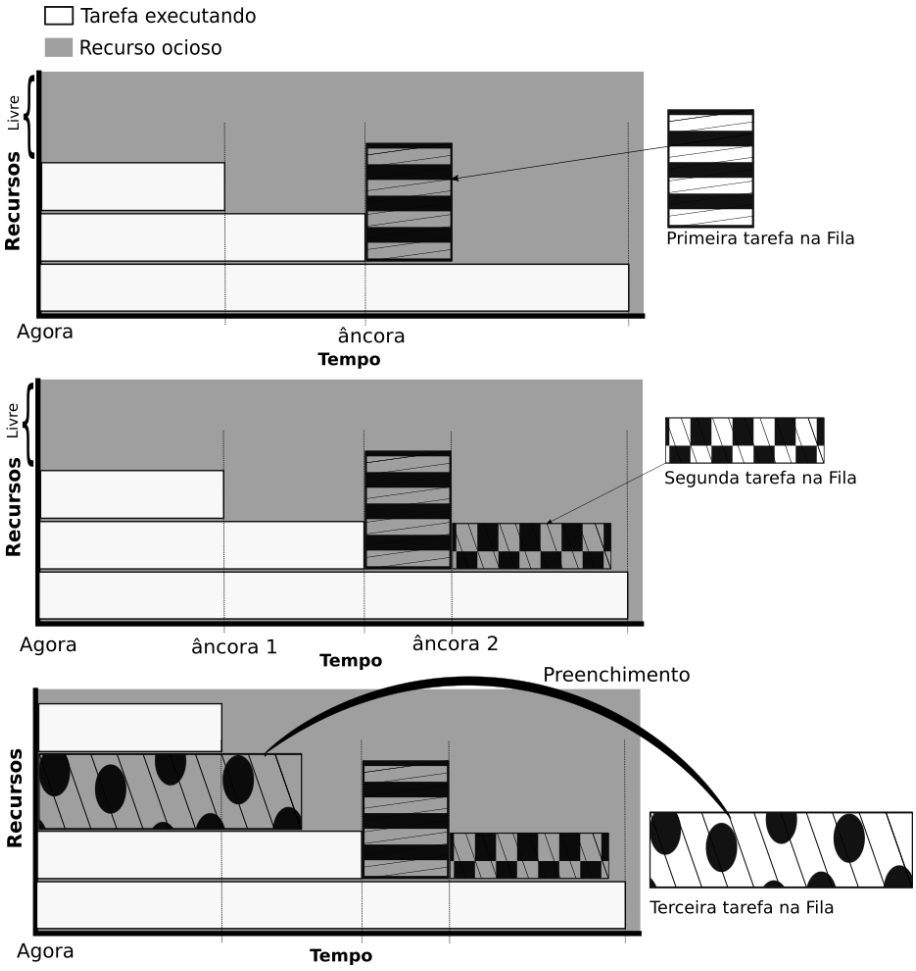

Figura 3. *Conservative backfilling* (CONS). Adaptado de [Mu'alem and Feitelson 2001].

realização deste trabalho, o EASY-*Backfilling*.

3.2. *Heterogeneous-Earliest-Finish-Time* (HEFT)

O *Heterogeneous-Earliest-Finish-Time* (HEFT), algoritmo de escalonamento de tarefas *offline*, é um algoritmo baseado em lista de requisições por prioridades, que modela fluxos de trabalhos através de uma *Directed Acyclic Graph* (DAG) [Topcuoglu et al. 2002]. Composto por duas fases, a execução do HEFT depende da entrada dos seguintes dados: uma DAG das requisições que aguardam recursos, a lista de recursos computacionais disponíveis e o tempo estimado de execução das tarefas (denominado *walltime*). Considerando um conjunto de tarefas N, a priorização das tarefas ($i \in N$) que aguardam recursos, é definido pela Equação 1 [Topcuoglu et al. 2002], sendo que: $succ(i)$ representa o conjunto de sucessores imediatos de i; $\overline{c_{i,j}}$ a média do custo de comunicação entre o par de tarefas (i, j); e $\overline{w_i}$ é custo médio de computação de uma tarefa i. O $rank(i)$ é computado para todos os vértices do DAG e, após finalizado, a tarefa terá a sua prioridade de acordo com o caminho crítico da raiz a sua posição no DAG.

$$rank(i) = \overline{w_i} + max_{j \in succ(i)}(\overline{w_j}) + (\overline{c_{i,j}} + rank(j)) \quad (1)$$

Em resumo, as fases do algoritmo HEFT são a *priorização das tarefas* e a *seleção do recursos* [Topcuoglu et al. 2002]. A fase de priorização das tarefas compreende da atribuição de prioridades através do cálculo do $rank_u$, resultando em uma lista em ordem decrescente, de acordo com o comprimento do caminho crítico, com empates sendo decididos aleatoriamente. Na fase de seleção de recursos, de posse da tarefa com maior prioridade, o HEFT associa a requisição ao recurso que tem o menor tempo de execução esperado para esta tarefa.

3.3. Sobreposição de Tarefas

O objetivo da técnica de sobreposição de tarefas difere dos demais algoritmos elencados, pois enaltece a maximização do uso de recursos mesmo que ação impacte na degradação do desempenho. Assim, a proposta de sobreposição de tarefas se encaixa perfeitamente ao contexto de redução do consumo energético. O princípio da técnica estende o algoritmo de FCFS, flexibilizando a regra de atendimento, ou seja, escalonando tarefas em lacunas temporais independente da ordem de chegada [Rios et al. 2009].

O algoritmo apoia-se em uma taxa de sobreposição com o intuito de controlar a degradação da qualidade do escalonamento durante a evolução do tempo. Isto é, a definição da taxa de sobreposição depende da estimativa de tempo de execução informada pelo usuário durante a requisição. O tempo de execução de uma tarefa ($i \in N$) em um único processador pode ser descrito como o somatório do custo da tarefa no processador e o tempo total de execução do seu predecessor mais lento, como descrito na Equação 2 [Rios et al. 2009], sendo que $pred(i)$ indica as tarefas predecessoras de i.

$$t_i = (w_{i,q}) + max_{j\ inpred(i)}(t_j) \tag{2}$$

Para efetuar o encaixe de uma tarefa em uma determinada lacuna temporal de agendamento, o algoritmo precisa calcular o tamanho da lacuna em função das estimativas de tempo de execução das tarefas enfileiradas. A lacuna entre as tarefas k e j comporta uma tarefa i se o $t_k^I - t_j^F$ for maior ou igual ao tempo t_i, como modelado na Equação 3 [Rios et al. 2009], sendo t^I e t^F os instantes inicial e final do escalonamento.

$$t_i \leq t_k^I - t_j^F \tag{3}$$

As Equações 2 e 3 auxiliam na realização do preenchimento de lacunas temporais, mas não atuam na sobreposição de execução das tarefas. O melhor aproveitamento dos recursos com a sobreposição de tarefas é efetuado com a inclusão do cálculo de lacunas com uma taxa de sobreposição. Portanto, acrescenta-se um fator Δ de sobreposição, como descrito pela Equação 4 [Rios et al. 2009].

$$t_i \leq (t_k^I - t_j^F) + ((\Delta * t_i)/100) \tag{4}$$

A aplicação de sobreposição de tarefas é exemplificada na Figura 4. Para análise, considera-se a sequência de requisições (1, 2, 3 e 4) de tarefas, cujo escalonamento inicial está representado pelo cenário A da Figura 4. Observa-se a existência de uma lacuna temporal de ociosidade entre as tarefas 1 e 2, porém o tempo de execução solicitado pela

tarefa 4 é superior a lacuna. Portanto, como resultado da Equação 3, a tarefa 4 é enfileirada após a tarefa 3, compondo o cenário *B* descrito na Figura 4.

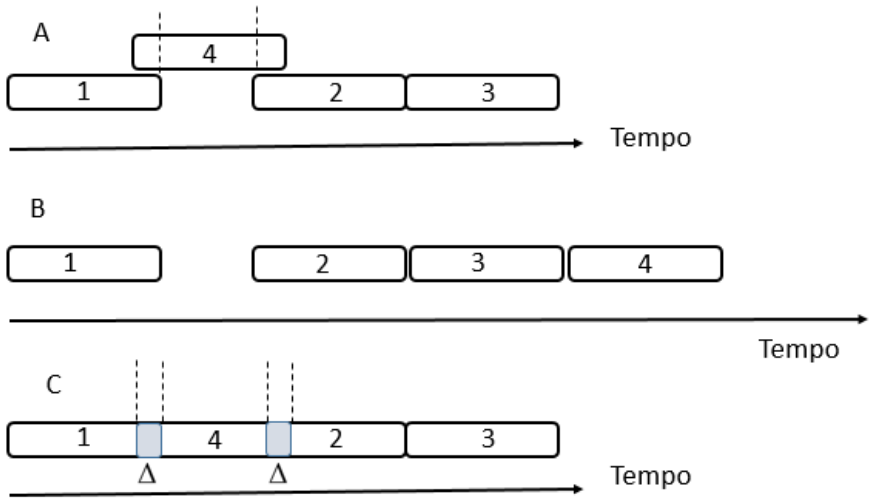

Figura 4. Sobreposição de tarefas. Adaptado de [Rios et al. 2009].

Com a sobreposição de tarefas utilizando $\Delta = 30\%$, o algoritmo é capaz de agendar a tarefa 4 entre as tarefas 1 e 2, como pode ser observado no cenário *C*. A exemplificação induz a conclusão, através de uma análise visual, de que o tempo final do cenário *A* é o mesmo do cenário *C* que, por sua vez, é menor em t_4. Todavia, esta conclusão depende da carga de trabalho das tarefas 1, 4 e 2 nos períodos sobrepostos. Além disso, o tempo de execução (*walltime*) informado pelo usuário durante a requisição não é confiável e deve ser utilizado, simplesmente, como uma previsão [Mu'alem and Feitelson 2001].

4. Heurística Verde de Escalonamento de Tarefas

As grades computacionais são infraestruturas flexíveis com enorme poder computacional e, portanto, origem de diversas aplicações de HPC. Assim como outras grandes infraestruturas de TI, as grades também se destacam no consumo de energia mundial. A preocupação com a utilização eficiente dos recursos naturais tem se intensificado nesta última década, não sendo diferente no contexto de grades computacionais. A proposta de Heurística Verde de escalonamento de tarefas para grades computacionais emergiu da combinação de elementos reconhecidos na área de gerenciamento de recursos computacionais e eficiência energética. Embora a proposta perpasse os conceitos de Grades Verdes, o foco específico é na redução do consumo de energia sem degradação no desempenho. O princípio da heurística é a aplicação da técnica de sobreposição de tarefas associado aos tradicionais algoritmos de escalonamento em grades computacionais, EASY-*Backfilling* e HEFT.

4.1. Política de Escalonamento

O intervalo de tempo de troca entre tarefas em grades computacionais compreende o término de uma tarefa, a comunicação de término desta ao RMS, a consulta ao escalonador de tarefas para identificar o próximo selecionado e o seu lançamento. Minimizar o tempo de troca possibilita redução do consumo de energia pois o agregado ficará menos tempo ocioso, bem como redução do *makespan* pois as tarefas reduzirão o tempo de

troca. Assim, a técnica de sobreposição pode prover melhoramento no tempo quando o tempo de execução é igual ou inferior ao tamanho da lacuna de ociosidade entre duas tarefas [Rios et al. 2009]. O aproveitamento de lacunas temporais eficiente já é feito por algoritmos como EASY-*Backfilling* e HEFT. Porém, estes são capazes apenas de adequar tarefas que se enquadrem nas restrições impostas pela Equação 3. O princípio desta proposta é flexibilizar o encaixe de tarefas, aplicados aos algoritmos EASY-*Backfilling* e HEFT, com uma janela de sobreposição. A taxa de sobreposição (Δ) pode ser individualizada, por lacuna, e dinâmica, possibilitando que a taxa seja adequada a carga de processamento da tarefa que já encontra-se em execução.

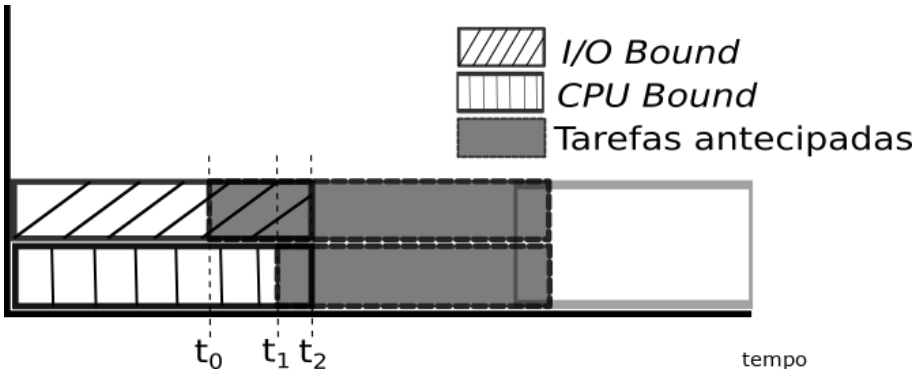

Figura 5. Taxa de sobreposição individualizada.

No exemplo da Figura 5, dois processos com o mesmo *walltime* estão executando e têm seus términos previstos para o tempo t_2. Ao longo de suas execuções, um processo foi caracterizado como *CPU Bound* (processamento intenso) e outro com *I/O Bound* (entrada/saída intensa). No tempo t_0, o algoritmo de escalonamento é chamado para realizar uma análise de reagendamento de tarefas e preenchimento de lacunas. A análise de sobreposição indicou que a taxa de sobreposição para a tarefa *CPU Bound* deve ser a mínima (*i.e.*, $\Delta = 5\%$) e, para a tarefa *I/O Bound*, a máxima (*i.e.*, $\Delta = 30\%$). Desta forma, as tarefas hachuradas foram antecipadas, porém uma iniciou em t_0 sobrepondo sua execução à tarefa *I/O Bound* desde a reanálise, e outra teve que aguardar t_1, buscando minimizar a concorrência pelo processador. Com um adequado controle da sobreposição e preenchimento de lacunas, o algoritmo melhora o uso dos recursos sem degradar o desempenho.

4.2. Implementação do Protótipo

A simulação é uma técnica amplamente adotada para analisar ambientes reais de sistemas distribuídos de larga escala possibilitando o controle sobre os experimentos e cargas [Dutot et al. 2015, Buyya and Murshed 2002]. A comparação justa dos resultados entre escalonadores exige o uso de uma mesma carga, em dois ou mais cenários distintos, tornando o uso do ambiente real impróprio para este caso. Dentre as ferramentas de simulação existentes [Kondo 2007, Caniou and Gay 2009, Diaz et al. 2007, Krzysztof Kurowski et al. 2007], o *Batsim* é um simulador de grade computacional que permite descrever infraestruturas físicas de agregados heterogêneos e estender seus componentes, *i.e.*, algoritmos de escalonamento [Dutot et al. 2015, Poquet 2017]. Os demais simuladores focam na avaliação de algoritmos de escalonamento, porém não são flexíveis

e tem suas versões desatualizadas, sem manutenção ou em fase de reformulação. Baseado no *SimGrid*[4], o *Batsim* foi validado com a grade computacional privada *Grid'5000*[5].

Os principais componentes da arquitetura do *Batsim* são descritos na Figura 6. O núcleo do ambiente de simulação está baseado na plataforma *SimGrid* e são importados pelo *Batsim*. O controle do *Batsim* é aplicado a *Execução do Escalonador*, *Métricas de Escalonamento* e *Métricas de Trabalho*. A execução de uma simulação depende da definição de uma infraestrutura e carga, entradas do sistema. Como saída, o sistema fornece uma variedade de relatórios, *i.e.*, consumo total de energia, *makespan*.

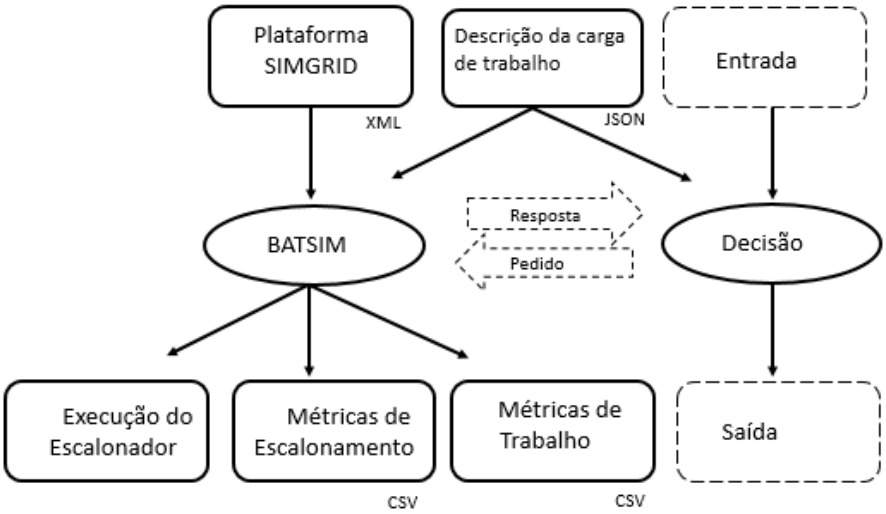

Figura 6. Processos de simulação com a ferramenta BatSim. Adaptado de [Dutot et al. 2015].

Especificamente sobre o cálculo do consumo energético, o simulador delega ao *SimGrid* e considera quatro cenários:

(i) *Idle (*100 *Watts)*: servidor está ligado, porém sem processamento;
(ii) *OneCore (*120 *Watts)*: servidor ligado com apenas um núcleo ativo (carga de 100%);
(iii) *AllCores (*200 *Watts)*: servidor ligado com todos os núcleos ativos (todos em 100%); e
(iv) *Off (*10 *Watts)*: servidor está desligado (*standby*).

O protótipo, associado ao ambiente de simulação *Batsim*, é descrito no diagrama da Figura 7. Previamente, o usuário deve preparar dois arquivos, um no formato *XML*, com a descrição da infraestrutura física a ser simulada, e outro em *JSON*, com a descrição da carga de trabalho. O passo seguinte exige a escolha do algoritmo de escalonamento instrumentalizado com a Heurística Verde, atualmente, com duas implementações disponíveis, EASY-*Backfilling* e HEFT. A execução simulada das tarefas é efetuada e monitorada pelo ambiente *Batsim*, com a coleta de dados de execução. Os últimos passos referem-se as instruções internas do *Batsim* para elaboração de relatórios e finalização das tarefas

[4]Disponível em: `https://simgrid.org/`
[5]Disponível em: `https://www.grid5000.fr/`

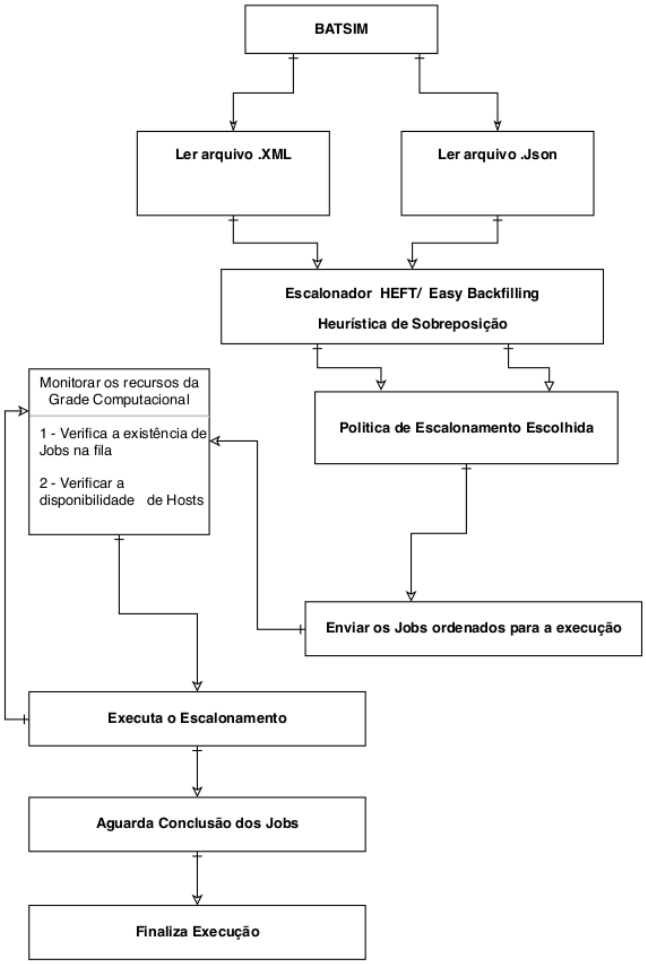

Figura 7. Diagrama do Protótipo da Heurística Verde de Escalonamento de Tarefas.

O protótipo da Heurística Verde incorporou ao ambiente a descrição de cargas de trabalho com diferentes taxas de processamentos, possibilitando a escolha de cargas com 0, 25, 50, 75 e 100% de processamento. Para isso, considera-se o período total de execução da tarefa alternando a carga com 0% ou 100% pelo tempo necessário para representar a carga desejada. O código fonte das implementações e os arquivos de descrição da infraestrutura física estão públicos no *GitHub* (`https://github.com/tathianaduarte/projeto_overlap_batsim`).

5. Análise Experimental

5.1. Cenários para Análise

As simulações foram baseadas no ambiente *Batsim*, aplicando cargas sintéticas disponíveis no ambiente e cargas modeladas a partir de observações do ambiente real *Grid'5000*. Para analisar o comportamento da Heurística Verde de escalonamento de tarefas, os resultados dos algoritmos originais, EASY-*Backfilling* e HEFT, foram comparados com as versões modificadas pela Heurística Verde, esta, por sua vez, com diferentes

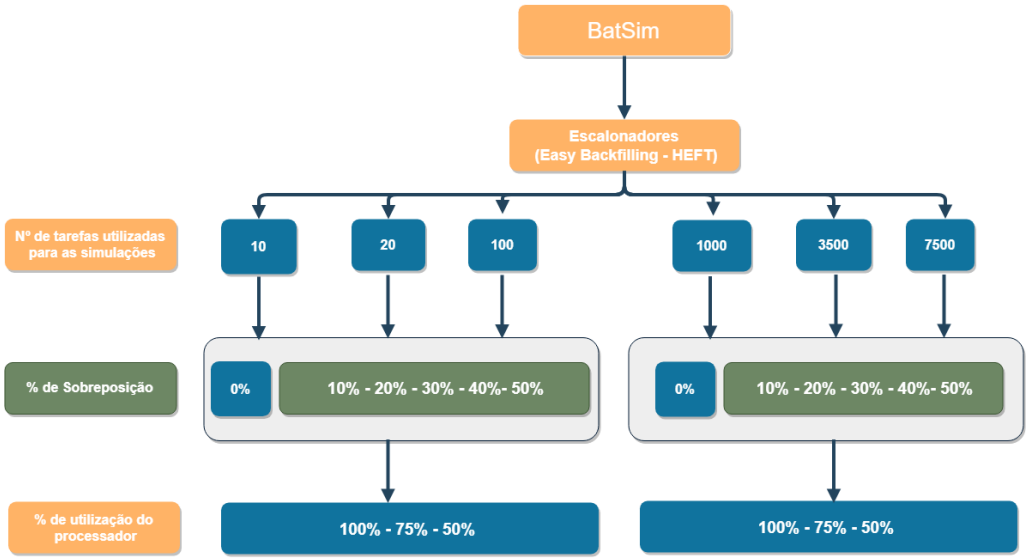

Figura 8. Plano de testes realizados.

taxas de sobreposição (Δ), como resumido na Figura 8. Os resultados com os algoritmos originais não tem sobreposição, representado com 0%.

Conforme observado no esquema organizacional da Figura 8, as cargas de trabalhos variaram para averiguar cenários representativos. O primeiro conjunto de testes foi caracterizado por baixas taxas de chegada de tarefas (10, 20 e 100), representando períodos de pouca utilização dos recursos. O segundo, com altas taxas de chegadas (1000, 3500 e 7500), quando existe alta taxa de utilização dos recursos, consequentemente, maior probabilidade de formação de filas. As faixas de sobreposição, individualmente aplicadas, foram de 0% a 50% com passos de 10%. Finalmente, a execução completa do plano de testes resultou em 136 simulações, fornecendo resultados do consumo de energia em *Joules* e o tempo de execução em *segundos*.

5.2. Ambiente de Testes

O ambiente de simulação escolhido foi o *Batsim* (versão 2.0.0[6]) desenvolvido em *python*3 (versão 3.5.3) e baseado no *SimGrid* (versão 3.13.91[7]). O *Batsim* foi instalado em um computador equipado com processador Intel $I5-2450M$ (4 núcleos com frequência de 2.5 GHz), 6GB RAM, disco rígido 500 GB e sistema operacional GNU/Linux Ubuntu 18.04. A infraestrutura virtual da grade computacional simulada foi inspirada no hardware do sítio Nantes do ambiente real *Grid'5000*[8]. A escolha do agregado de Nantes foi impírica, porém o estudo pode ser estendido para os demais agregados do *Grid'5000*. O agregado modelado foi o *Nantes Econome* constituído de 22 nós computacionais, totalizando 44 CPUs, 352 cores (2.20 GHz), 64 GB RAM, 2.0 TB de disco rígido e conexão de rede a 10 $Gbps$.

[6]Disponível em: `https://github.com/oar-team/batsim`
[7]Disponível em: `https://gforge.inria.fr/projects/simgrid/`
[8]Disponível em: `https://www.grid5000.fr/w/Nantes:Hardware`

5.3. Discussão dos Resultados

A análise dos resultados segue: três linhas, (i) a apresentação dos consumos energéticos do agregado *Nantes Econome*, (ii) a análise do impacto no tempo total das tarefas e (iii) a linha de cruzamento dos melhores resultados de cada conjunto de tarefas. O total de consumo em *Joules* de todos os experimentos do plano de testes estão na Tabela 2. As colunas especificam os algoritmos utilizados, EASY e HEFT, e a taxa de sobreposição aplicada à Heurística Verde de Escalonamento, sendo 0%, o algoritmo original, e 10% a 50% a própria heurística. As três linhas em destaque na Tabela 2 marcam o início do conjunto de experimentos (abaixo) com uma determinada carga de trabalho. As cargas aplicadas foram 100% de utilização de CPU, primeiro conjunto, seguidos de 75% e 50% para os demais. Todas as tarefas do conjunto possuem a mesma carga. As demais linhas da tabela representam a quantidade de tarefas a serem escalonadas e seus respectivos consumos energéticos.

Tabela 2. Sumarização dos consumos energéticos de *Nantes Econome* (*Joules*).

Tarefas	*0%*		*10%*		*20%*		*30%*		*40%*		*50%*			
	EASY	**HEFT**	**EASY**	**HEFT**	**EASY**	**HEFT**	**EASY**	**HEFT**	**EASY**	**HEFT**	**EASY**	**HEFT**	**MÍNIMO**	**MÁXIMO**
Carga de trabalho das tarefas 100%														
10	$16.2x10^3$	$15.2x10^3$	$15.9x10^3$	$13.9x10^3$	$15.0x10^3$	$13.5x10^3$	$14.9x10^3$	$13.2x10^3$	$15.1x10^3$	$13.1x10^3$	$15.3x10^3$	$13.1x10^3$	$13.1x10^3$	$16.2x10^3$
20	$32.6x10^3$	$31.5x10^3$	$30.5x10^3$	$29.7x10^3$	$30.2x10^3$	$29.6x10^3$	$30.2x10^3$	$29.1x10^3$	$31.1x10^3$	$29.3x10^3$	$31.9x10^3$	$29.5x10^3$	$29.1x10^3$	$32.6x10^3$
100	$163.9x10^3$	$163.5x10^3$	$158.2x10^3$	$159.9x10^3$	$158.5x10^3$	$159.9x10^3$	$159.8x10^3$	$160.3x10^3$	$161.2x10^3$	$161.5x10^3$	$162.9x10^3$	$161.7x10^3$	$158.2x10^3$	$163.9x10^3$
1000	$1.7x10^6$	$1.9x10^6$	$1.6x10^6$	$1.8x10^6$	$1.6x10^6$	$1.8x10^6$	$1.6x10^6$	$1.8x10^6$	$1.6x10^6$	$1.8x10^6$	$1.6x10^6$	$1.9x10^6$	$1.6x10^6$	$1.9x10^6$
3500	$21.7x10^6$	$23.9x10^6$	$20.2x10^6$	$22.3x10^6$	$20.3x10^6$	$22.5x10^6$	$20.6x10^6$	$22.8x10^6$	$21.3x10^6$	$23.5x10^6$	$21.8x10^6$	$24.5x10^6$	$20.2x10^6$	$24.5x10^6$
7500	$39.5x10^9$	$42.0x10^9$	$38.9x10^9$	$39.7x10^9$	$39.5x10^9$	$41.9x10^9$	$40.0x10^9$	$43.7x10^9$	$40.1x10^9$	$43.9x10^9$	$41.8x10^9$	$45.0x10^9$	$38.9x10^9$	$45.0x10^9$
Carga de trabalho das tarefas 75%														
10	$16.1x10^3$	$15.0x10^3$	$15.2x10^3$	$13.2x10^3$	$14.9x10^3$	$12.9x10^3$	$14.2x10^3$	$12.9x10^3$	$16.0x10^3$	$13.0x10^3$	$15.8x10^3$	$14.0x10^3$	$12.9x10^3$	$16.1x10^3$
20	$31.5x10^3$	$31.0x10^3$	$29.9x10^3$	$28.3x10^3$	$30.0x10^3$	$29.2x10^3$	$30.1x10^3$	$28.9x10^3$	$30.9x10^3$	$29.2x10^3$	$30.7x10^3$	$29.4x10^3$	$28.3x10^3$	$31.5x10^3$
100	$159.2x10^3$	$161.2x10^3$	$157.2x10^3$	$159.3x10^3$	$157.3x10^3$	$159.6x10^3$	$158.1x10^3$	$160.0x10^3$	$159.9x10^3$	$160.4x10^3$	$161.0x10^3$	$161.0x10^3$	$157.2x10^3$	$160.4x10^3$
1000	$1.5x10^6$	$1.8x10^6$	$1.4x10^6$	$1.7x10^6$	$1.4x10^6$	$1.7x10^6$	$1.4x10^6$	$1.7x10^6$	$1.5x10^6$	$1.7x10^6$	$1.5x10^6$	$1.8x10^6$	$1.4x10^6$	$1.8x10^6$
3500	$19.8x10^6$	$22.3x10^6$	$17.9x10^6$	$21.5x10^6$	$17.9x10^6$	$21.8x10^6$	$18.2x10^6$	$22.1x10^6$	$18.3x10^6$	$22.1x10^6$	$19.1x10^6$	$22.9x10^6$	$17.9x10^6$	$22.3x10^6$
7500	$38.9x10^9$	$41.3x10^9$	$37.9x10^9$	$38.2x10^9$	$37.9x10^9$	$38.2x10^9$	$38.0x10^9$	$39.9x10^9$	$38.8x10^9$	$40.9x10^9$	$39.2x10^9$	$41.8x10^9$	$37.9x10^9$	$41.8x10^9$
Carga de trabalho das tarefas 50%														
10	$15.9x10^3$	$14.9x10^3$	$15.1x10^3$	$13.1x10^3$	$14.0x10^3$	$11.9x10^3$	$14.1x10^3$	$12.2x10^3$	$15.0x10^3$	$13.0x10^3$	$15.2x10^3$	$14.2x10^3$	$11.9x10^3$	$15.9x10^3$
20	$30.8x10^3$	$29.9x10^3$	$29.5x10^3$	$28.0x10^3$	$29.1x10^3$	$28.8x10^3$	$29.9x10^3$	$28.7x10^3$	$30.2x10^3$	$29.0x10^3$	$30.2x10^3$	$29.1x10^3$	$29.0x10^3$	$30.8x10^3$
100	$156.3x10^3$	$158.2x10^3$	$153.4x10^3$	$156.1x10^3$	$153.5x10^3$	$156.8x10^3$	$154.9x10^3$	$157.2x10^3$	$155.0x10^3$	$157.9x10^3$	$155.9x10^3$	$158.5x10^3$	$153.4x10^3$	$158.5x10^3$
1000	$1.3x10^6$	$1.7x10^6$	$1.1x10^6$	$1.6x10^6$	$1.2x10^6$	$1.6x10^6$	$1.2x10^6$	$1.6x10^6$	$1.2x10^6$	$1.7x10^6$	$1.3x10^6$	$1.8x10^6$	$1.1x10^6$	$1.8x10^6$
3500	$17.6x10^6$	$20.1x10^6$	$16.2x10^6$	$19.5x10^6$	$16.3x10^6$	$19.9x10^6$	$16.8x10^6$	$19.9x10^6$	$17.2x10^6$	$20.4x10^6$	$17.5x10^6$	$20.5x10^6$	$16.2x10^6$	$20.5x10^6$
7500	$36.8x10^9$	$41.0x10^9$	$35.8x10^9$	$39.3x10^9$	$35.9x10^9$	$39.9x10^9$	$36.2x10^9$	$39.8x10^9$	$36.9x10^9$	$40.8x10^9$	$37.0x10^9$	$41.8x10^9$	$35.8x10^9$	$41.8x10^9$

A primeira análise dos resultados destaca os consumos máximos (em vermelho) e mínimos (em verde) de cada conjunto de tarefas. Para facilitar a análise e a visualização dos valores da Tabela 2, os consumos estão descritos na base 10. Vale observar que os valores variam entre 10^3 e 10^9. Caso os consumos obtidos tenham valores equivalentes, o destaque é efetuado no valor com menor taxa de sobreposição. A primeira constatação é a tendência de polarização dos valores extremos, que pode ser observado pelo posicionamento da maioria dos mínimos e máximos nas laterais da Tabela 2, somente três valores foram obtidos com a sobreposição central (20% e 30%).

A análise dos máximos, representando o pior caso, os resultados demonstram uma concentração ainda maior. Pode-se concluir que, independente da carga das tarefas (100%, 75% ou 50%), os máximos se alternam entre os dois extremos, algoritmos originais (0%) ou Heurística Verde com sobreposição de 50%. Os algoritmos originais obtiveram a maioria dos máximos nos conjuntos com poucas tarefas (10, 20 e 100) e a Heurística Verde com sobreposição de 50% nos conjuntos maiores (1000, 3500 e 7500). Quanto ao algoritmo, 61% dos máximos foram obtidos com o HEFT, sendo que 38% foi com HEFT modificado pela Heurística Verde com sobreposição de 50%, indicando claramente a ineficiência da aplicação da técnica com taxas elevadas de sobreposição.

Os melhores casos, representados pelos valores mínimos, apresentam comportamento bem definido: 67% dos melhores resultados foram obtidos pela Heurística Verde com 10% de sobreposição associada ao algoritmo EASY. O restante dos mínimos foram obtidos pelo HEFT com taxas de sobreposição 20%, 30% e 40%, porém também em situações bem definidas, poucas tarefas (10 ou 20). Pode-se concluir que, cenários com mais de 100 tarefas, o melhor resultado energético é obtido pela Heurística Verde EASY com 10%. Todavia, o potencial de ganho energético está na diferença de consumo entre os máximos e os mínimos. Nesta análise, o ganho variou entre 2% e 49% de redução no consumo de energia. A maior taxa de redução foi com o conjunto de 1000 tarefas e carga de 50%, e a pior taxa, foi com 100 tarefas e carga de 75%.

Seguindo a mesma metodologia de análise, baseado nos valores máximos e mínimos, obtêm-se os resultados dos tempos de execução (Tabela 3). A Tabela 3 é cons-

Tabela 3. Sumarização dos tempos de execução.

	Tarefas	Consumo Máximo		Consumo Mínimo		%
		Algoritmo	Tempo	Algoritmo	Tempo	
Carga 100%	10	EASY 0%	37.0*s*	HEFT 40%	34.4*s*	7
	20	EASY 0%	52.9*s*	HEFT 30%	50.3*s*	5
	100	EASY 0%	260.2*s*	EASY 10%	258.1*s*	1
	1000	HEFT 0%	2145.8*s*	EASY 10%	1462.8*s*	32
	3500	HEFT 50%	536*d*	EASY 10%	465*d*	14
	7500	HEFT 50%	1.157*d*	EASY 10%	1.090*d*	6

tituída dos tempos de execução dos casos destacados na Tabela 2, com os melhores e os piores casos de consumo de energia, para cada conjunto de tarefas (10, 20, 100, 1000, 3500 e 7500) com a carga de trabalho de 100%. O comportamento com as demais cargas é semelhante. Os conjuntos de tarefas entre 10 e 1000 têm seus valores de tempo em unidade de segundos (*s*) e, os dois outros conjuntos, em dias (*d*). A última coluna da Tabela 3 apresenta os percentuais de redução do tempo entre os valores máximos e mínimos descritos a cada linha.

Os resultados permitem concluir que, a carga de trabalho da tarefa tem pouca influência no tempo final de execução para conjuntos que representam pouca utilização dos recursos (10, 20 e 100 tarefas). A faixa de redução é igual ou inferior a 7%. As diferenças acentuam-se quando o conjunto de tarefas representa maior utilização dos recursos (1000, 3500 e 7500), variando entre 6% e 32% de redução. Portanto, a escolha do algoritmo e taxa de sobreposição são fatores essenciais no escalonamento das tarefas, tanto na redução do consumo de energia quanto no tempo total de execução de um conjunto de requisições.

Para finalizar a análise, o gráfico da Figura 9 relaciona o percentual de redução do consumo de energia com a redução no tempo total de execução. No eixo *x*, tem-se os percentuais de redução no consumo entre máximos e mínimos (destacados na Tabela 2) e, no eixo *y*, os percentuais de redução de tempo total de execução. Os pontos representam número de tarefas (*i.e.*, 10, 20, 100, 1000, 3500 e 7500) do conjunto e carga de trabalho (*i.e.*, 100% - máxima, 75% - média e 50% - mínima), respectivamente.

Os conjuntos que se destacam nos percentuais de redução são os com 1000 e 3500 tarefas. Nestes casos, os consumos máximos vieram do algoritmo HEFT com

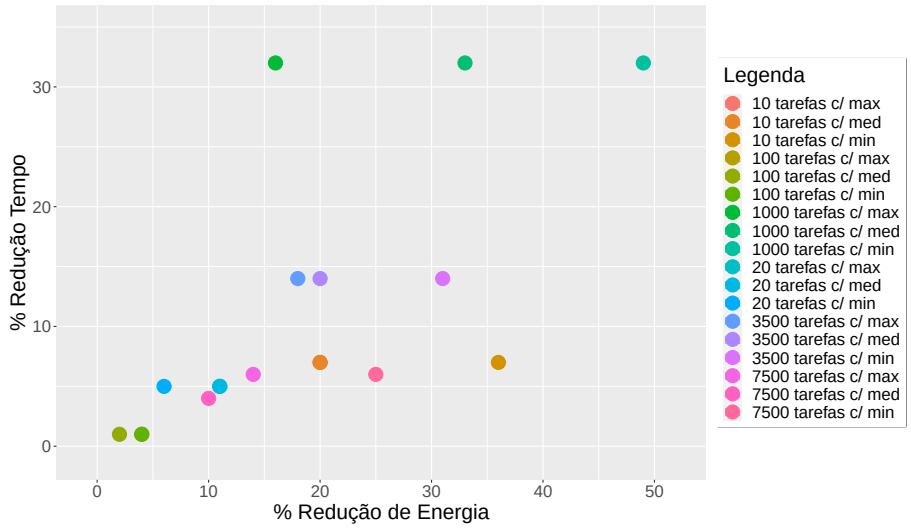

Figura 9. Relação percentual de reduções de consumo de energia *versus* tempo de execução.

sobreposição de 50%, que revelaram um consumo de energia excessivo e atrasos no tempos de execução. A concentração dos resultados se deu nas reduções de até 10% dos tempos e de até 20% no consumo de energia. Restringindo as comparações de percentuais de redução da Heurística Verde com 10% de sobreposição, melhor caso da heurística, com o algoritmo original EASY-*Backfilling*, o percentual de redução atingiu 16% com carga de trabalho 50% e número de tarefas 1000. As faixas de redução encontradas reforçam a importância da adequação dos algoritmos, apresentados nesta proposta, ao comportamento das tarefas e as taxas de chegadas. Finalmente, o aspecto de escalabilidade da solução não foi analisado.

6. Considerações e Trabalhos Futuros

Uma grade computacional objetiva a entrega de potência computacional máxima para aplicações oriundas de diferentes áreas. Porém, a tendência atual de preservação de recursos (Grades Verdes), entre eles o consumo de energia, exige a adequação destes ambientes a nova realidade mundial: o fornecimento de potência de cálculo com minimização dos desperdícios de energia. A Heurística Verde proposta, baseada na sobreposição de tarefas, traz uma alternativa a escalonadores de tarefas usados em cenários de produção, integrados a RMS reconhecidos pela comunidade acadêmica e indústria, que possibilita a redução no consumo de energia sem degradação nos tempos de execução das tarefas. O principal objetivo da Heurística Verde é atacar os desperdícios oriundos de atividades administrativas entre trocas de tarefas, *i.e.*, comunicação de término da execução ao RMS, consulta ao escalonador.

Os resultados indicaram que a Heurística Verde proposta é eficiente quando aplicada nos períodos de trocas, representada pela taxa de sobreposição de 10%, ocasionando na obtenção de 67% dos melhores resultados. Em contrapartida, a Heurística Verde, com 50% de sobreposição, destacou-se por seus resultados ruins, em alguns casos, ocasionando atrasos maiores que 10% no tempo de execução, como já se esperava. A redução no consumo de energia, comparando-se os máximos e os mínimos, variou entre 2% e 49% e nos tempos entre 1% e 32%. A análise dos resultados permite concluir que o uso

de técnicas de sobreposição pode ser eficaz na redução do consumo de energia em grades computacionais. O acréscimo no tempo de execução total das aplicações ocorre em alguns casos, tornando essencial a configuração correta da taxa de sobreposição. O trabalho ainda deixa como legado a sociedade científica, a revisão bibliográfica atualizada sobre o tema e soluções para redução do consumo de energia em plataformas Grades Computacionais. Como trabalhos futuros, tem-se a implementação da Heurística Verde com outros algoritmos (*i.e.*, *Conservative backfilling* (CONS)), e a aplicação de cargas heterogêneas e a modelagem de outros sítios da *Grid5000*.

Agradecimentos

Os autores agradecem a Fundação de Amparo a Pesquisa de Santa Catarina (FAPESC) pelo apoio ao Laboratório de Processamento Paralelo Distribuído (LabP2D) do CCT/UDESC, e aos membros do LabP2D pela colaboração na pesquisa.

Referências

Avelar, V., Azevedo, D., French, A., and Power, E. N. (2012). "PUE™: A Comprehensive Examination of the Metric". *White paper*, 49.

Borro, L. C. (2014). *Escalonamento em grades móveis: uma abordagem ciente do consumo de energia*. PhD thesis, Universidade de São Paulo.

Buyya, R. and Murshed, M. (2002). "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing". *Concurrency and computation: practice and experience*, 14(13-15):1175–1220.

Caniou, Y. and Gay, J. S. (2009). Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems. In César, E., Alexander, M., Streit, A., Träff, J. L., Cérin, C., Knüpfer, A., Kranzlmüller, D., and Jha, S., editors, *Euro-Par 2008 Workshops - Parallel Processing*, pages 223–234, Berlin, Heidelberg. Springer Berlin Heidelberg.

Carastan-Santos, D., Camargo, R., Trystram, D., and Zrigui, S. (2019). One can only gain by replacing EASY Backfilling: A simple scheduling policies case study. In *CCGrid 2019. IEEE International Symposium on Cluster Computing and the Grid, 2019.*

Chawla, S. and Saluja, K. (2016). "enhanced job scheduling algorithm with budget constraints in Computational Grids". In *Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on*, pages 515–520. IEEE.

Corcoran, P. and Andrae, A. (2013). Emerging Trends in Electricity Consumption for Consumer ICT.

Dakkak, O., Awang Nor, S., and Arif, S. (2016). Scheduling through Backfilling Technique for HPC Applications in Grid Computing Environment. In *2016 IEEE Conference on Open Systems (ICOS)*, pages 30–35.

de Carvalho, L. A. V., Santana, R., et al. (2011). A workflow scheduling algorithm for optimizing energy-efficient grid resources usage. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 642–649. IEEE.

De FranÇa, J.-C. (2014). Método GRASP baseado em Computação Verde aplicado ao Escalonamento de Recursos e Tarefas em Grades Computacionais.

Demir, Y. and İsleyen, S. K. (2014). An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations. *International Journal of Production Research*, 52(13):3905–3921.

Diaz, A., Batista, R., and Castro, O. (2007). Realtss: a real-time scheduling simulator. In *2007 4th International Conference on Electrical and Electronics Engineering*, pages 165–168.

Dutot, P.-F., Georgiou, Y., Glesser, D., Lefevre, L., Poquet, M., and Rais, I. (2017). Towards Energy Budget Control in HPC. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 381–390. IEEE Press.

Dutot, P.-F., Mercier, M., Poquet, M., and Richard, O. (2015). Batsim: A Realistic Language-independent Resources and Jobs Management Systems Simulator. In *Job Scheduling Strategies for Parallel Processing*, pages 178–197. Springer.

Endo, P. T. and et. al (2016). High availability in clouds: systematic review and research challenges. *JoCCASA*, 5:16.

Foster, I. and Kesselman, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Gaussier, E., Glesser, D., Reis, V., and Trystram, D. (2015). Improving Backfilling by Using Machine Learning to Predict Running Times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 64:1–64:10, New York, NY, USA. ACM.

Guazzone, M. (2014). Mining the Workload of Real Grid Computing Systems. *CoRR*, abs/1412.2673.

GÓmez-MartÍn, C., Vega-RodrÍguez, M. A., and GonzÁlez-SÁnchez, J.-L. (2016). Fattened backfilling: An improved strategy for job scheduling in parallel systems. *Journal of Parallel and Distributed Computing*, 97:69 – 77.

Hasan, M. and Goraya, M. S. (2016). Priority based cooperative computing in cloud using task backfilling. *Lecture Notes on Software Engineering*, 4(3):229–233.

Iamnitchi, A., Doraimani, S., and Garzoglio, G. (2009). Workload characterization in a high-energy data grid and impact on resource management. *Cluster Computing*, 12(2):153–173.

Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., and Epema, D. H. (2008). The Grid Workloads Archive. *Future Generation Computer Systems*, 24(7):672 – 686.

Ismail, L. (2012). Performance versus Cost of a Parallel Conjugate Gradient Method in Cloud and Commodity Clusters. *International Journal of Computer Science and Network Security*, 12(11):25–34.

Kim, K. H., Buyya, R., and Kim, J. (2007). Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters. In *International*

Symposium in Cluster, Cloud, and Grid Computing (CCGrid), volume 7, pages 541–548. IEEE/ACM.

Kondo, D. (2007). Simboinc: A simulator for desktop grids and volunteer computing systems.

Kraemer, A., Maziero, C., Richard, O., and Trystram, D. (2018). Reducing the number of response time service level objective violations by a Cloud-HPC convergence scheduler. *Concurrency and Computation: Practice and Experience*, 30(12):e4352.

Krzysztof Kurowski, Nabrzyski, J., Oleksiak, A., and Weglarz, J. (2007). Grid scheduling simulations with GSSIM. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8.

Lelong, J., Reis, V., and Trystram, D. (2017). Tuning EASY-Backfilling queues. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 43–61. Springer.

Lifka, D. A. (1995). The ANL/IBM SP scheduling system. In Feitelson, D. G. and Rudolph, L., editors, *Job Scheduling Strategies for Parallel Processing*, pages 295–303, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mämmelä, O., Majanen, M., Basmadjian, R., De Meer, H., Giesler, A., and Homberg, W. (2012). Energy-aware job scheduler for High-Performance Computing. *Computer Science-Research and Development*, 27(4):265–275.

Mu'alem, A. W. and Feitelson, D. G. (2001). "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling". *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543.

Nabi, M., Toeroe, M., and Khendek, F. (2016). Availability in the cloud: State of the art. *Journal of Network and Computer Applications*, 60:54–67.

Nepovinnykh, E. A. and Radchenko, G. I. (2016). Problem-oriented scheduling of cloud applications: PO-HEFT algorithm case study. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 180–185.

Ni, T. (2015). An Improved Job Scheduling Algorithm Based on the First Distribution and Redistribution Strategy for Grid Computing. In *The 5th International Conference on Computer Engineering and Networks*, volume 259, page 060. SISSA Medialab.

Poquet, M. (2017). *Approche par la simulation pour la gestion de ressources*. PhD thesis, Université Grenoble Alpes.

Rath, C. K., Suar, S. S., and Biswal, P. (2018). A Comparative study on Dynamic Task Scheduling algorithms. *i-Manager's Journal on Information Technology*, 7(1):1–6.

Rios, R. A., Jacinto, D. S., Schulze, B., and Guardia, H. C. (2009). Análise de Heurísticas para Escalonamento Online de Aplicações em Grade Computacional. 1:13–24.

Selvi, S. and Manimegalai, D. (2017). DAG Scheduling in Heterogeneous Computing and Grid Environments Using Variable Neighborhood Search Algorithm. *Applied Artificial Intelligence*, 31(2):134–173.

Singh, P., Quadri, Z., and Kumar, A. (2016). Comparative Study of Parallel Scheduling Algorithm for Parallel Job. *International Journal of Computer Applications*, 134(10):10–14.

Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S. U., and Li, K. (2016). An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. *Journal of Grid Computing*, 14(1):55–74.

Teodoro, S., do Carmo, A. B., and Fernandes, L. G. (2013). Energy efficiency management in computational grids through energy-aware scheduling. In *Proceedings Annual ACM Symposium on Applied Computing*, pages 1163–1168. ACM.

Topcuoglu, H., Hariri, S., and Wu, M.-y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274.

Torquato, M. D., Torquato, L., and Maciel, P. (2018). Modelos para avaliação de disponibilidade orientada a capacidade de uma nuvem privada. *Revista de Informática Teórica e Aplicada*, 25(2):73–84.

Wang, J., Han, D., and Wang, R. (2018). A new rule-based power-aware job scheduler for supercomputers. *The Journal of Supercomputing*, 74(6):2508–2527.

Wang, M., Srivathsan, S., Huang, E., and Wu, K. (2018). Job Dispatch Control for Production Lines With Overlapped Time Window Constraints. *IEEE Transactions on Semiconductor Manufacturing*, 31(2):206–214.

Watanabe, E., Campos, P. P., Braghetto, K., and Batista, D. (2014). Algoritmos para economia de energia no escalonamento de workflows em nuvens computacionais. In *32nd Brazilian Symposium on Computer Networks and Distributed Systems, Florianopolis, Brazil*.

Yadav, R. R., Campos, G. A. S., Sousa, E. T. G., and Lins, F. A. (2019). A Strategy for Performance Evaluation and Modeling of Cloud Computing Services. *Revista de Informática Teórica e Aplicada*, 26(1):78–90.

You, X., Li, Y., Zheng, M., Zhu, C., and Yu, L. (2017). A Survey and Taxonomy of Energy Efficiency Relevant Surveys in Cloud-Related Environments. *IEEE Access*, 5:14066–14078.

Zakarya, M. (2018). Energy, performance and cost efficient datacenters: A survey. *Renewable and Sustainable Energy Reviews*, 94:363 – 385.

Zheng, H. and Wu, J. (2018). Joint Scheduling of Overlapping MapReduce Phases: Pair Jobs for Optimization. *IEEE Transactions on Services Computing*, pages 1–1.