Este trabalho tem como objetivo aplicar a Heurística *Overlap nos escalonadores HEFT e Easy Backfilling* no ambiente de Grade Computacional buscando reduzir o consumo de energia elétrica. Essa heurística significa o compartilhamento de tempo no processador para a execução de *Jobs*. Observando os resultados obtidos, é possível concluir que foi possível reduzir o consumo de energia em todos os cenários aplicados com o percentual de 10% sobreposição.

Orientador: Mauricio Aronne Pillon

JOINVILLE, 2019

ANO
2019

TATHIANA DUARTE DO AMARANTE | HEURÍSTICA OVERLAP APLICADA A ALGORITMOS DE ESCALONAMENTO PARA GRADES COMPUTACIONAIS

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
PROGRAMA DE PÓS GRADUAÇÃO EM COMPUTAÇÃO APLICADA

DISSERTAÇÃO DE MESTRADO

HEURÍSTICA OVERLAP APLICADA A ALGORITMOS DE ESCALONAMENTO PARA GRADES COMPUTACIONAIS

TATHIANA DUARTE DO AMARANTE

JOINVILLE, 2019

TATHIANA DUARTE DO AMARANTE

HEURÍSTICA OVERLAP APLICADA A ALGORITMOS DE ESCALONAMENTO PARA GRADES COMPUTACIONAIS

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, para a obtenção do grau de Mestre em Computação Aplicada.

Orientador: Prof. Dr. Mauricio Aronne Pillon

JOINVILLE

2019

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Setorial do CCT/UDESC,

com os dados fornecidos pelo(a) autor(a)

Amarante, Tathiana Duarte do
Heuristica Overlap Aplicada a Algoritmos de Escalonamento para Grades Computacionais / Tathiana Duarte do Amarante. -- 2019.
97 p.

Orientador: Mauricio Aronne Pillon
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2019.

1. Grades Computacionais. 2. Recursos ociosos. 3. Overlap. 4. Redução de energia. 5. Batsim. I. Pillon, Mauricio Aronne . II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

Heurística Overlap Aplicada a Algoritmos de Escalonamento em Grade Computacional

por

Tathiana Duarte do Amarante

Esta dissertação foi julgada adequada para obtenção do título de

Mestra em Computação Aplicada

Área de concentração em "Ciência da Computação",
e aprovada em sua forma final pelo

CURSO DE MESTRADO ACADÊMICO EM COMPUTAÇÃO APLICADA
DO CENTRO DE CIÊNCIAS TECNOLÓGICAS DA
UNIVERSIDADE DO ESTADO DE SANTA CATARINA.

Banca Examinadora:

Prof. Dr. Maurício Aronne Pillon
CCT/UDESC (Orientador/Presidente)

Prof. Dr. Guilherme Piêgas Koslovski
CCT/UDESC

Prof. Dr. Mehran Misaghi
UNISOCIESC

Joinville,SC, 26 de fevereiro de 2019.

Dedico este trabalho ao meus pais Antonio e Vânia, por me mostrarem o caminho e me ajudarem sempre. Ao meu marido Maurício por estar ao meu lado nos momentos mais difíceis, que não foram poucos durante essa caminhada. E ao meu filho Lorenzo, nosso amor maior e um presente que ganhamos durante essa trajetória.

AGRADECIMENTOS

Gostaria de agradecer a todos que de alguma forma direta ou indireta contribuíram na realização deste trabalho.

Ao professor Maurício Aronne Pillon, por não desistir de mim, com todos esses percalços que acabei passando durante esse período.

Aos amigos Emanoeli Madalosso, Luís Felipe Bilecki, Paulo Roberto Vieira Junior e Anderson Henrique da Silva Marcondes, e todos que estiveram ao meu lado dando apoio, ajudando e proporcionando momentos de alegria.

À minha família por compreender os momentos que estive ausente e em especial ao meu marido Mauricio Rodrigues da Costa, que sempre esteve ao meu lado apoiando e fazendo com que eu não desistisse deste trabalho, mesmo nos momentos mais difíceis sempre me ajudou a buscar uma solução e forças para continuar.

RESUMO

Computação em Grade é uma abordagem que utiliza recursos computacionais geograficamente distribuídos e heterogêneos afim de resolver problemas de custo computacional. Desta forma, um dos maiores desafios encontrado para o desenvolvimento de projetos neste ambiente é o escalonamento de recursos, o qual consiste em mapeá-los, verificando sua disponibilidade e buscando reduzir a ociosidade que acontece no decorrer de todo o processo. Pesquisas científicas estão cada vez mais em ascensão, buscando reduzir o consumo de energia elétrica sem afetar o desempenho das Grades Computacionais. Segue essa abordagem, algorítmos de escalonamento com utilização de heurísticas estão sendo cada vez mais utilizados. Para isto, foi implementado a heurística de *Overlap* aplicado nos escalonadores *HEFT* e *Easy Backfilling*, cujo principal objetivo desta utilização é reduzir o consumo de energia. De maneira geral, essa heurística significa o compartilhamento de tempo no processador para a execução de Jobs, ou seja, esta política permite que haja um percentual de sobreposição de Jobs, visando a utilização de um *slot* que seria insuficiente para execução de um Job, mas que, permaneceria ocioso durante aquele período. Para estes experimentos foi utilizada a ferramenta de simulação *Batsim*, onde foram implementados alguns cenários distintos com *workloads* específicos, buscando simular ambientes reais. Os resultados apresentados avaliam o consumo de energia com utilização de CPU em 100%, 75% e 50%, com percentuais de aplicação em 0%, 10%, 20%, 30%, 40% e 50% de *overlap*. Com a utilização da heurística proposta, foi comprovado que, em todos os cenários estudados houve uma redução significativa de energia, principalmente quando aplicado com percentual de 10% de *overlap* e utilização de CPU de 50%.

Palavras-chaves: Grades Computacionais, recursos ociosos, *overlap*, *HEFT*, *Easy Backfilling*, redução de energia e *Batsim*

ABSTRACT

Grid Computing is an approach that uses computational resources geographically distributed and heterogeneous in order to solve problems computational cost. In this way, one of the greatest challenges found for the development of projects in this environment is the scheduling of resources, which consists in mapping them, verifying their availability and seeking to reduce the idleness that happens throughout the entire process. Scientific research is increasingly on the rise, seeking to reduce the consumption of electric energy without affecting the performance of the Grid Computing. Following this approach, scalability algorithms using heuristics are being increasingly used. For this, the *overlap* heuristic applied to the *HEFT* and *Easy Backfilling* schedulers was implemented, the main purpose of which is to reduce energy consumption. In general, this heuristic means sharing the time in the processor for the execution of Jobs, that is, this policy allows that there is a percentage of Jobs *overlap*, aiming at the use of a *slot* that would be insufficient for execution of a Job, but that, would remain idle during that period. For these experiments, the *Batsim* simulation tool was used, where some distinct scenarios were implemented with specific *workloads*, in order to simulate real environments. The results presented evaluate power consumption with CPU utilization at 100%, 75% and 50%, with application percentages at 0%, 10%, 20%, 30%, 40% and 50% *overlap*. Using the proposed heuristic, it was verified that in all scenarios studied there was a significant reduction of energy, especially when applied with a percentage of 10% *overlap* and CPU utilization of 50%.

Key-words: Computational Grids, Idle Resources, *Overla*, *HEFT*, *Easy Backfilling*, Energy Reduction, and Batsim,

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SÍMBOLOS

%	Porcentagem
J	Joules
s	Segundos
N_i	Determinada tarefa
w_i	Média do tempo de execução do *job*
t_i	Tempo inicial sobre todos os recursos
$suc(t_i)$	Tempo sobre todos os *job* sucessores imediatos
$c_{i,j}$	Média dos custos de comunicação
Rank (u)	Comprimento do caminho crítico de um *job*
P_j	Tempo inicial de conclusão de um dado processador
t_i^E	Tempo inicial de execução de um *job*
t_f^E	Tempo final de execução de um *job*
$t_{inicial}$	Tempo do início da execução do *job*
t_{final}	Tempo final da execução do *job*
$t_{processamento}$	Tempo de processamento do *job*
$t_{liberação}$	Tempo de liberação do *job*
t_{exec}	Tempo de execução do *job*
T_{total}	Tempo total em que o *job* permaneceu no sistema

SUMÁRIO

1 INTRODUÇÃO

Grade Computacional é uma infraestrutura de computação responsável por conectar diversos recursos permitindo a execução de várias aplicações que possam disponibilizar uma demanda maior de processamento. Assim, uma Grade Computacional é um ambiente de alto desempenho, caracterizado por viabilizar compartilhamento de diversos recursos geograficamente distribuídos (FOSTER; KESSELMAN, 2003),(BUYYA; MURSHED, 2002).

As principais características do ambiente de Grade Computacional, como o comportamento dinâmico, compartilhamento de recursos, heterogeneidade de recursos, distribuição em larga escala e segurança, são algumas das principais características que atrai atualmente pesquisadores a desenvolver formas para melhora-las. Assim, se faz necessário o uso de um escalonador, que tem como principal finalidade gerenciar e alocar essas aplicações a recursos disponíveis na Grade, buscando melhorar seu desempenho (FRANÇA, 2014), (DUTOT et al., 2017), (FOSTER; KESSELMAN, 2003). Desta forma, os escalonadores tornaram-se um dos focos principais de estudos, voltados para High Performance Computing (HPC). (KUMAR; KUMAR; KUMAR, 2013).

Com a crescente demanda das tecnologias no ambiente de Grade Computacional, o consumo de energia acabou impactando no elevado custo financeiro para mantê-las em operação (MÄMMELÄ et al., 2012). Desde então, estudos buscando métodos para reduzir esse consumo de energia está presente no desenvolvimento de pesquisas científicas, chamada de *Green Computing*, ou *Computação Verde* (AVELAR et al., 2012).

Melhorar a eficiência energética sem prejudicar o desempenho é um desafio. Desta forma a otimização no uso dos recursos e redução do consumo de energia têm impactos positivos, pois minimizam os custos e consequentemente os danos causados ao meio ambiente(SCARAMELLA; HEALEY, 2007). Desta maneira, alguns trabalhos estão sendo desenvolvidos buscando propor soluções que amenizem o consumo excessivo de energia em HPC, sendo que a maior parte das abordagens de redução de consumo de energia baseiam-se em desligar recursos ociosos, como apresentaram (MÄMMELÄ et al., 2012), (CHAWLA; SALUJA, 2016), ou alteração da frequência de operação da unidade de processamento, utilizado por (KIM; BUYYA; KIM, 2007), (TANG et al., 2016), ou ainda técnicas específicas com alterações nos escalonadores de tarefas, impactando no desempenho, como apresentado por (TEODORO; CARMO; FERNANDES, 2013), (WATANABE et al., 2014) e (RIOS et al., 2014).

A análise de comportamento de uma Grade Computacional, pode ser feita através de modelos simulados, ou por experimentos aplicados em ambiente real. A simulação é uma maneira para analisar esses ambientes em sistemas distribuídos de larga escala de recursos heterogêneos e tem sido aplicada modelando aplicações em diversos cenários replicando um ambiente real (DUTOT et al., 2015), (BUYYA; MURSHED, 2002).

O presente trabalho aplica a técnica de *overlap* em algoritmos de escalonamento de *jobs* reconhecidos pela comunidade científica. Os algoritmos escolhidos foram o *HEFT* e o *Easy Backfilling*, onde o princípio do trabalho é sobrepor parte da execução do job com a execução do próximo job escalonado em um determinado recurso. Devido a impossibilidade operacional de modificação de um escalonador em uma Grade computacional sem transtornos a seus usuários, os testes foram aplicados em um ambiente de simulação, o *Batsim*.

O foco do uso dessa técnica é observar o comportamento do consumo de energia com a sobreposição de execução de *jobs* em uma Grade Computacional.

1.1 OBJETIVOS

O uso de heurísticas aplicadas em escalonadores para planejar a execução de aplicações em ambientes de processamento distribuídos, se faz necessária para utilizar de maneira eficiente os recursos oferecidos e obter ganhos no desempenho e reduzir o consumo de energia. Para tal finalidade, foram estudados alguns algoritmos de escalonamentos, verificando suas características e buscando uma análise para a aplicação da heurística *overlap*.

As estratégias propostas neste trabalho foram desenvolvidas levando em conta algumas métricas que forneçam informações sobre os recursos disponíveis e também sobre os *jobs* recebidos, como tempo inicial e final da execução, tempo total de execução, tempo de espera e consumo de energia. Também levou-se em consideração, os espaços vagos que são gerados pelos planos de escalonamento, pois em momentos distintos, uma tarefa entra em execução para em seguida liberar para a próxima tarefa executar.

Sendo assim, este trabalho tem por objetivo implementar algoritmos com a heurística *overlap* nos escalonadores *HEFT*, utilizado em pesquisas para análise de desempenho e no escalonador *Easy Backfilling*, utilizado no ambiente real da Grid'5000, aplicado na ferramenta de simulação *BatSim*, com *workloads* de 10, 20, 100, 1000, 3500 e 7500 *jobs*, onde foi aplicado os percentuais de *overlap* de 10%, 20%, 30%, 40% e 50% e com utilização de processador de 100%, 75% e 50%, e por fim, analisar em quais dos cenários o consumo de energia elétrica será menor, reduzindo assim a

ociosidade dos recursos.

1.2 CARACTERIZAÇÃO METODOLÓGICA DA PESQUISA E ESCOPO

Por se tratar de um tema abrangente e com métodos de aplicação diferenciados, a Computação Verde, assim como a heurística *Overlap* dependem de suas aplicações práticas para validar seus resultados, desta forma, este trabalho pode ser classificado como uma pesquisa exploratória e de natureza aplicada. Os objetivos da pesquisa, a classificam como exploratória e explicativa, uma vez que monitora o consumo energético do ambiente de Grades Computacionais analisando os resultados encontrados. Quanto aos procedimentos técnicos aplicados no desenvolvimento da pesquisa, foram utilizados referências bibliográficas para alcançar o objetivo, buscando métodos de consumo da Computação em Grade através de mapeamento sistemático e aplicando análises quantitativas e qualitativas dos experimentos realizados. Com base em todo o levantamento bibliográfico realizado, a heurística *Overlap* foi implementada, e aplicada no ambiente de Grade, com base nas pesquisas realizadas. Com os resultados dos experimentos realizados, foi possível mensurar o consumo de energia utilizado na simulação e analisar seus resultados.

Desma forma, tem-se como o escopo da pesquisa, a redução de energia elétrica em Grades Computacionais, aplicando a heurística *overlap* nos escalonadores estudados. A implementação da solução proposta é desenvolvida em uma ferramenta de simulação Batsim, onde foi estabelecido alguns cenários com números de *jobs* específicos para analisar o comportamento e o consumo dos escalonadores em situações diferenciadas que contribuem para o desenvolvimento do presente trabalho.

1.3 ORGANIZAÇÃO DO TEXTO

Esta dissertação está organizada em 6 capítulos. No capítulo 2 é apresentada a revisão de literatura, com os principais conceitos de: Computação Distribuída, Computação Verde, Grade Computacional, Gerenciamento de Recursos e Escalonamento de *jobs*.

No capítulo 3 são apresentados os trabalhos relacionados as pesquisas científicas, onde destacam-se os trabalhos que apresentaram formas de escalonamentos utilizados em Grade Computacional, algoritmos de escalonamento e heurística de *overlap*, métodos de redução do consumo de energia elétrica, heurística de escalonamento verde com *overlap*, simuladores para Grades Computacionais. No Capitulo 4, encontra-se a solução encontrada para escalonamento com a heurística *overlap*, definição do problema, algoritmos de escalonamento com a aplicação do *overlap* e simulador Batsim.

Nos capítulos 5 e 6 são apresentados plano de testes, ambientes, métodos, resultados obtidos, conclusões, considerações finais, e por fim, trabalhos futuros.

2 REVISÃO DE LITERATURA

Com o aumento substancial de recursos computacionais dos últimos anos e a busca para melhorar cada vez mais o desempenho, tem-se como principal consequência o maior consumo energético em ambientes de Computação em Grade. Algoritmos de escalonamento de Grade Computacional são de extrema importância para o desempenho deste ambiente, buscando de forma eficiente provisionar os recursos que serão utilizados. Este capítulo apresenta a revisão de literatura sobre Grades Computacionais, Sistemas Distribuídos, Computação Verde, Consumo de Energia, Gerencia de Recursos, Escalonadores, Ferramentas de Simulação, e por fim, Heurística de Escalonamento.

2.1 COMPUTAÇÃO DISTRIBUÍDA E DE ALTO DESEMPENHO

Para trabalhar com o modelo da Computação em ambiente de Grade Computacional é importante compreender sua origem, para isso precisamos voltar aos conceitos da Computação Distribuída, área na qual insere-se a Computação em Grade.

Um sistema distribuído é uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente (FERNANDES, 2015). Essa definição expõe a computação distribuída com o objetivo de disponibilizar poder computacional por meio de diversos computadores interligados, para processar tarefas, de forma coerente e transparente, ou seja, como se apenas um único e centralizado computador estivesse executando-a (FERNÁNDEZ-MONTES et al., 2012).

No início, o modelo de computação distribuída objetivava a distribuição do processamento de uma aplicação por diversas máquinas físicas, resolvendo alguns problemas da computação centralizada, como exemplo, o sistema ficar inacessível devido a uma máquina estar ocupada (WERNER et al., 2011). Não havia também a preocupação em redução de custos, redução de máquinas e equipamentos. Entretanto, atualmente existem outros fatores importantes a margem desse modelo como o custo energético e o desempenho de ambientes que o implementam.

A HPC está relacionada à área da Computação Distribuída por meio da prática de agregar o poder da computação de forma a proporcionar um desempenho muito maior do que se poderia conseguir com computadores típicos, como estações de trabalho, à fim de resolver complexos problemas da ciência, engenharia, ou de negócios (XAVIER et al., 2013).

Esse termo abrange não só computadores, mas também a tecnologia de rede,

os algoritmos e os ambientes necessários para o uso destes sistemas, que podem variar de aglomerados de computadores à supercomputadores. Estas plataformas provedoras de HPC podem possuir sistemas com memória compartilhada, distribuída, combinação de vários processadores e a utilização da computação paralela, visto que por meio da computação paralela o poder de processamento de uma única CPU pode ser multiplicado para resolver uma tarefa. Isso quer dizer que, a computação paralela permite que problemas muito complexos, que exigem muitos cálculos, ou pesquisas em gigantes bases de dados, possam ser resolvidos em um tempo razoável. Isto se aplica, por exemplo, a casos como a previsão global do tempo e o sequenciamento do genoma humano (FISCHBORN et al., 2006).

A utilização de Grades Computacionais e agregados para HPC, são cada vez mais beneficiadas pela evolução das tecnologias de processadores e pelas suas interconexões, ao mesmo tempo em que a utilização de chips multi-cores como nodes desses agregados criam um ambiente paralelo multi-nível. Com tantos núcleos de processamento operando de maneira concorrente e considerando os esforços em direção a um desenvolvimento tecnológico sustentável, faz-se essencial o desenvolvimento de sistemas paralelos que, além de prover alto desempenho, consigam fazê-lo com um eficaz consumo de energia elétrica, contribuindo assim para um melhor aproveitamento dos recursos naturais do planeta (MÓR et al., 2010).

2.2 COMPUTAÇÃO VERDE

Os benefícios que a utilização de Grades Computacionais e agregado, são de grande importância para a tecnologia, porém o impacto desta utilização tem causado danos ao meio ambiente. Estudos demostram que cerca de 2% da energia mundial são utilizados em grandes centros de dados. Sendo assim foi necessário a implantação da computação verde buscando amenizar os danos ao meio ambiente (AVELAR et al., 2012), (FRANÇA, 2014).

O início da computação verde foi marcado pelo lançamento em 1992 do programa voluntário *Energy Star* da agência de proteção ambiental dos EUA, que visava promover produtos energéticos eficientes para reduzir a emissão de gases na atmosfera. Atualmente inúmeras iniciativas governamentais impulsionam a computação verde (ENERGY, 2011).

A computação verde é uma abordagem convencional, que objetiva gerenciar recursos com relação direta a economia de energia, mantendo o comprometimento e o desempenho de acordo com a demanda de serviços (DELVAZ; BOVÉRIO, 2017). Como pode ser observado na Figura 1, onde a redução do consumo de energia elétrica pode chegar a um percentual de 25% do total de economia, a redução de custos

em aproximadamente 24% e a melhoria na performance do sistema em até 18%, percentuais estes de impacto para utilização de práticas verdes diretamente ligados ao desenvolvimento de pesquisas na área.

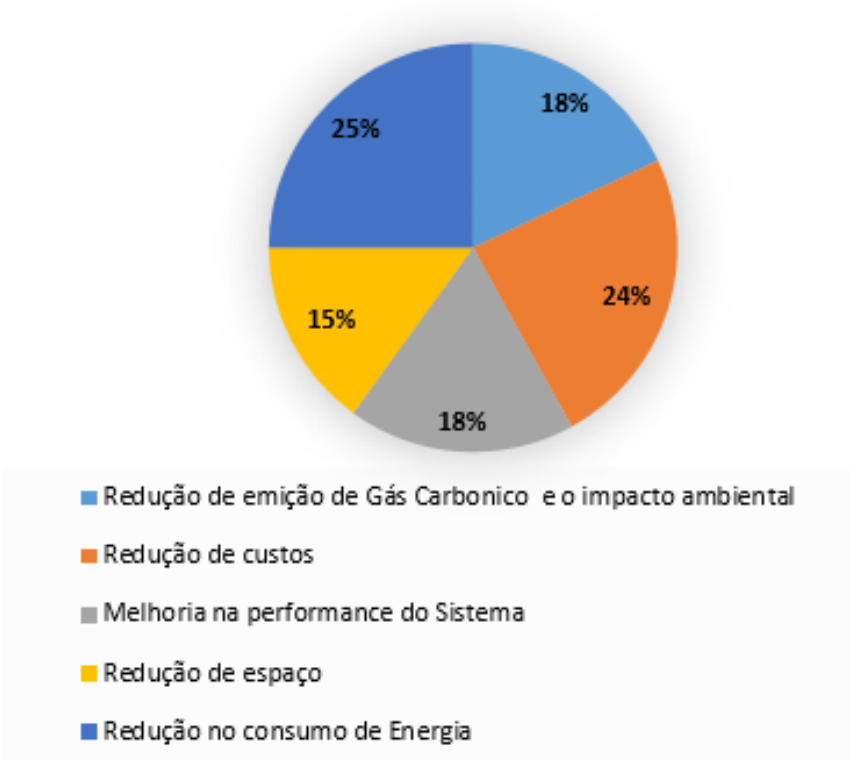


Figura 1 – Principais razões para a utilização de práticas verdes. Fonte adaptado de (SILVA et al., 2010) e (DELVAZ; BOVÉRIO, 2017)

Nos últimos anos, estudos voltados para maior eficiência energética em sistemas computacionais tem crescido motivadas por questões ambientais. A primeira motivação é devido aos problemas causados pelo meio ambiente, com as crescentes mudanças climáticas em decorrência do aumento no consumo mundial de energia. A segunda trata-se do desafio de prolongar o tempo de vida dos equipamentos computacionais e garantir que os requisitos de desempenho sejam atendidos (HAN et al., 2018).

A ascensão da computação verde começou devido ao grande número de plataformas construídas para um poder de processamento maior. Para tanto foi criado uma organização responsável por analisar o potencial desses supercomputadores, a TOP500 [1], criada em 1993 ela é responsável por ranquear os 500 computadores mais potentes do mundo. Consequentemente, sentiu-se a necessidade de aplicar a computação verde também nesta lista, e criou-se a Green500 [2], que desde 2013, classifica dentre os 500 supercomputadores o que possui um maior percentual de redução de

[1] <https://www.top500.org/>
[2] <https://www.top500.org/green500/>

consumo de energia. Como pode ser observado na Figura 2, a eficiência energética do TOP500[3] dos últimos 10 anos, onde foi realizado uma média de consumo das máquinas mais eficientes economicamente que estão na lista dos 500 supercomputadores. Pode-se observar que de 2017 para 2018, o percentual de redução de energia elétrica ficou em média de 8,34% mais eficiente. Porém foi de 2016 para 2017, que a demanda maior ocorreu, com um crescente salto, chegando a 56,43% de eficiência energética (TOP500, 2018).

O sistema mais eficiente em energia da lista Green500 em 2018 é mais uma vez o sistema Shoubu B, um supercomputador ZettaScaler-2.2 instalado no Centro Avançado de Computação e Comunicação, RIKEN, Japão. Ele foi medido novamente e alcançou 17,6 gigaflops / watt durante sua execução de 1,06 petaflops com benckmark Linpack e ocupa a posição 376 na lista TOP500 (TOP500, 2018).

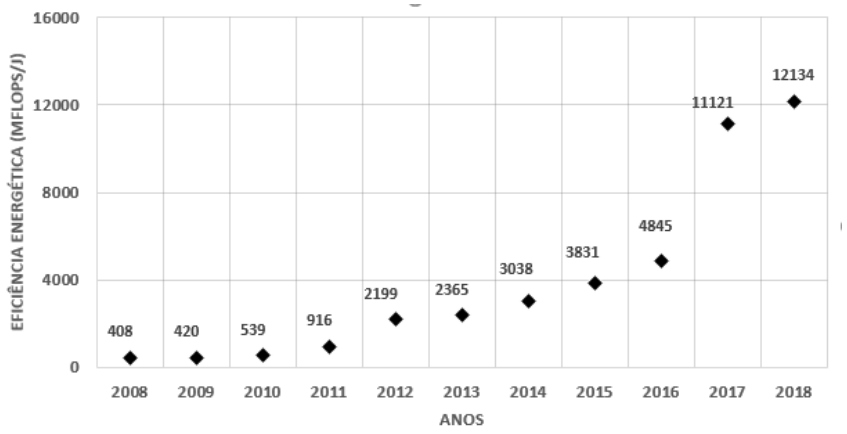


Figura 2 – Evolução da Eficiência Energética das Plataformas retirado do site TOP500 2018. Fonte: próprio autor

Segundo (SILVA et al., 2010), a necessidade de se tomar medidas ecologicamente corretas se justifica não somente por questões de desempenho, mas por se devolver o equilíbrio ao ecossistema do planeta. Cada vez mais se tem uma atenção para um padrão de consumo de energia para todas as tecnologias de informações e comunicações (ARTHI; HAMEAD, 2013).

Algumas formas para reduzir o consumo de energia elétrica foram elaboradas. Dentre elas, as métricas verdes, que aplicam-se em ambientes computacionais buscando analisar e verificar como estaria o consumo energético. Para medir a eficiência no uso de energia em ambientes computacionais uma das métricas com maior impacto é a PUE (Power Usage Effectiveness), que é definida pela relação entre a energia total para manter ambiente computacional e a energia utilizada apenas pelos equipamentos de TI (AVELAR et al., 2012).

[3] <https://www.top500.org/news/china-extends-supercomputer-share-on-top500-list-us-dominates-in-total-performance/>

Algumas estratégias da Computação Verde a partir de componentes de software são (FRANÇA, 2014):

- **Algoritmos Eficientes** têm como objetivo projetar algoritmos que sejam capazes de reduzir o tempo de execução e demanda por recursos, podendo proporcionar uma redução de custo/consumo dos recursos computacionais. Um dos benefícios de algoritmos eficientes na prática é com relação a máquinas ociosas, pois consome menos energia, logo, quanto mais rápido possível terminar a execução e o algoritmo encontrar a solução do problema, mais tempo a máquina irá passar no estado ocioso.
- **Gerenciamento de Hardware** tem como ideia principal informar ao sistema operacional que um determinado hardware não será mais utilizado, logo, desliga-se este dispositivo, evitando assim, o consumo de energia e a ociosidade desnecessária.

Desta forma, pode-se observar que a estratégia da Computação Verde que busca reduzir um consumo de energia elétrica com maior relevância para a aplicação do trabalho são os algoritmos eficientes, pois ele procura uma solução utilizando escalonadores, podendo ser aplicado no ambiente de Grade Computacional.

2.3 GRADE COMPUTACIONAL

Na década de 1990, o termo Grade (Grid) foi criado para descrever tecnologias que permitiriam aos consumidores obter computação sob demanda, ou seja alocar máquinas físicas em ambientes distribuídos para utilizá-las. Posteriormente pesquisadores utilizaram essa ideia para desenvolver sistemas federados em larga escala como *TeraGrid, Open Science Grid, caBIG, EGEE e Earth System Grid* (FOSTER; KESSELMAN, 2003)

Grade Computacional possui uma padronização quanto a sua arquitetura e aos protocolos, assim como o relacionamento entre eles, onde o responsável por essa padronização é o *Open Grid Forum* que foi criado em 2006 com a fusão do *Global Grid Forum* e da *Enterprise Grid Alliance* (RODAMILANS, 2009).

Muitas das definições de *Grade Computacional, Nuvem Computacional e aglomerados* têm características semelhantes, com objetivos parecidos (FOSTER; KESSELMAN; TUECKE, 2001):

- Cada organização possui suas políticas administrativas relacionadas a segurança, permissão de usuários entre outros;

- Utilização de padrões abertos para protocolos e interfaces;
- Qualidade de serviço (QoS), como tempo de resposta, segurança e coordenação de recursos eficaz;
- Heterogeneidade, os recursos podem ser heterogêneos quanto a sua natureza e as tecnologias;
- Escalabilidade, novos recursos podem ser integrados a *Grade Computacional*;
- Adaptabilidade, a *Grade Computacional* deve ser capaz de se adaptar às situações dinâmicas, tais como falha de recursos.

Outra consideração importante é que as tecnologias não são capazes de abranger os diferentes tipos de recursos de maneira a realizar o compartilhamento flexível e controlado para as organizações virtuais. Neste contexto a *Grade Computacional* atua com protocolo de gerenciamento de credenciais, protocolos de informação, segurança e gerenciamento de dados (FOSTER; KESSELMAN; TUECKE, 2001), (RODAMILANS, 2009).

2.3.1 ARQUITETURA DE GRADE COMPUTACIONAL

A arquitetura de *Grade Computacional* fornece protocolos e serviços em cinco camadas diferentes como apresentada na Figura 3 (FOSTER et al., 2008).

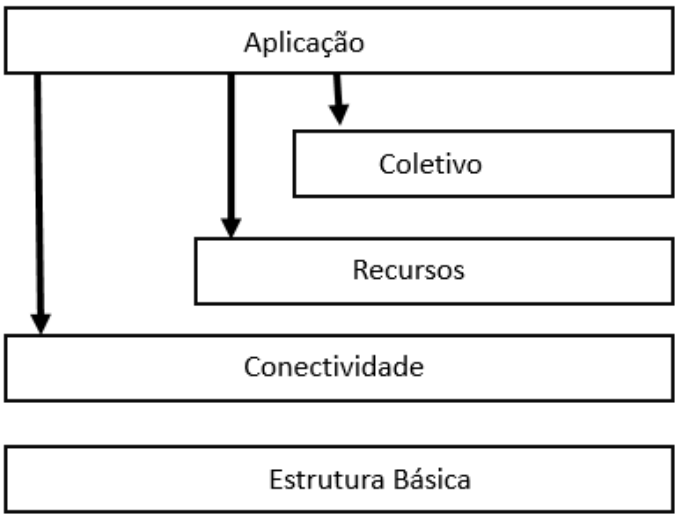


Figura 3 – Arquitetura de Protocolo de Grade Computacional. Adaptado de (FOSTER et al., 2008)

A camada *Estrutura Básica*, tem o papel de conectar todos os recursos de rede da *Grade Computacional*. Acima da camada de *Rede*, encontra-se a de *Conectividade*, que define a comunicação central, protocolos de autenticação para rede e

segura, como o GSI *(Grid Security Infrastructure)*. Na camada de *Recursos* são definidos protocolos para a publicação, descoberta, negociação, monitoramento, contabilidade, pagamento e compartilhamento das operações em recursos individuais . A camada que pode ser utilizada como estudo e desenvolvimento do trabalho é a *Coletivo*, que é responsável por capturar as interações entre recursos, serviços de diretório, monitoramento e descoberta de novos recursos, por fim a quinta camada *Aplicação* compreende quaisquer aplicativos de usuários construído em cima dos protocolos e APIs.

Sendo assim, o conhecimento da arquitetura das camadas que uma Grade Computacional possui pode facilitar a aplicação de formas de reduzir o consumo de energia elétrica neste ambiente.

2.3.2 CONSUMO DE ENERGIA NA GRADE COMPUTACIONAL

A computação verde tem influenciado diversas áreas da computação, como por exemplo, o projeto de hardware, gerência de *Data Centers* e *Grades Computacionais* (PONCIANO et al., 2010). Quando um computador está disponível para a grade, mas a grade não o utiliza, ele fica em estado ocioso (idle). Ao longo do tempo, a permanência dos computadores nesse estado caracteriza ciclos de ociosidade na grade e desperdício de energia (KONDO et al., 2007).

A Grid'5000 é um ambiente real que foi planejado para testes experimentais em computação distribuída onde oferece mais de 5 mil processadores, todos interligados em locais geograficamente distribuídos. Para isso, foi desenvolvido um ambiente de monitoramento para estudar o perfil do consumo de energia das aplicações nestes ambientes (ASSUNCAO et al., 2012).

Um exemplo de aplicação foi a pesquisa desenvolvida por (ORGERIE; LEFÈVRE; GELAS, 2010) monitorou a real utilização da Grid'5000 durante aproximadamente doze meses, e constatou que, o tempo em que ela permanece ociosa é de aproximadamente 50,57%, elevando assim o consumo de energia elétrica sem uma real necessidade. Desta forma, (ASSUNCAO et al., 2012), estudou o comportamento do ciclo de vida típico de um aplicativo com *benckmarks* específicos em um nó da Grid'5000, como pode-se observar na Figura 4.

Nesta representação a inicialização do nó dura aproximadamente 120 segundos e é caracterizada por picos de consumo de energia, chegando até 252 watts (boot). Períodos de ociosidade chegam a consumir 240 watts (idle). Acessos intensivos ao disco consome em média 243 watts (hdparm). Uso intensivo de rede utiliza cerca de 263 watts (iperf) e a utilização total de CPU chega a 277 watts (cpuburn/ stress). Este experimento, foi monitorado durante 474 segundos, com um consumo de

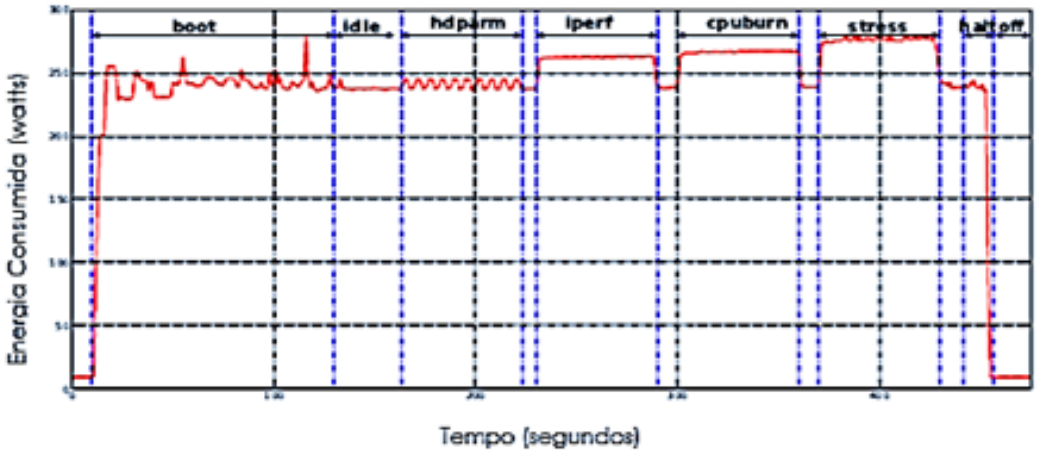


Figura 4 – Comportamento de uma aplicação demostrando o consumo de energia de um nó do ambiente da Grid'5000. Fonte:(ASSUNCAO et al., 2012)

111.296,34 Joules.

Na Figura 5, apresenta-se o caminho realizado por um agregado real que encontras-se na Grid'5000. Sua finalidade é verificar a parte energética do aglomerado, cujo os consumos aproximados em uma média de 10 repetições foram de: processamento de dados de aproximadamente 190 watts, máquina em estado ocioso consome 95 watts, ligar uma máquina consome em média 125 watts com um tempo de duração de 2,5 minutos, portanto o consumo total seria de 18.750 Joules.

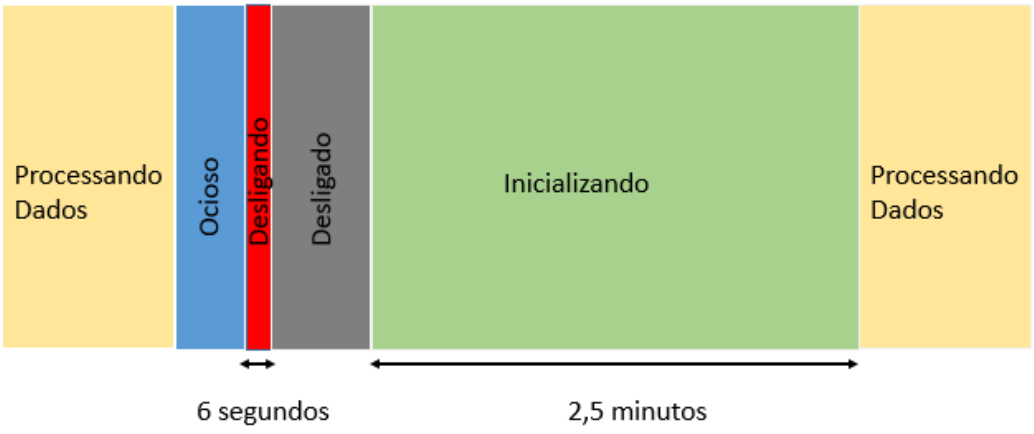


Figura 5 – Comportamento do aglomerado Taurus da Grid'5000 demostrando seu consumo de energia. Fonte adaptada (DUTOT et al., 2017)

Outras pesquisas foram desenvolvidas com base no consumo de energia elétrica em ambientes de Grades Computacionais, algumas em ambiente real como a do (ASSUNCAO et al., 2012), (ORGERIE; LEFÈVRE; GELAS, 2010) e outras em ambien-

tes simulados como (POQUET, 2017b), (TEODORO; CARMO; FERNANDES, 2013) e todas constataram que o consumo de energia é considerável se tratando de recursos disponíveis em estado de ociosidade. Desta maneira a melhor estratégia para se reduzir o consumo é melhorar o tempo de ociosidade que os recursos dispõe em determinados momentos.

2.4 GERENCIAMENTO DE RECURSOS

O gerenciamento de recursos influencia diretamente no desempenho dos sistemas, podendo ser utilizado em diversos ambientes computacionais. Ele é responsável por prover mecanismos com os quais os consumidores e provedores de recursos possam interagir atendendo a uma determinada política (CASAVANT; KUHL, 1988). Os consumidores precisam de recursos que satisfaçam os seus objetivos, os provedores oferecem esses recursos de acordo com seus requerimentos e o gerenciador procura ligá-los buscando performance e eficiência.

Em se tratando de *Computação em Grade*, os recursos gerenciados podem ser processadores, dispositivos de armazenamento de dados, largura de banda e serviços. Esses recursos podem ser heterogêneos, distribuídos em diferentes organizações virtuais, e requerem diferentes políticas de utilização (FOSTER; KESSELMAN; TUECKE, 2001). Essas características, atribuídas em larga escala, implicam em dificuldades, tais como, na obtenção de informações dos recursos, preocupação com segurança e respeito às diferentes políticas administrativas (RODAMILANS, 2009).

Sendo assim, várias aplicações são executadas simultaneamente na Grade Computacional e os recursos são compartilhados. As aplicações procuraram utilizar as informações atuais para otimizar sua execução. Para o alcance de uma performance adequada da Grade, o escalonamento será fundamental (FOSTER; KESSELMAN, 2003).

2.5 ESCALONAMENTO DE JOBS

O termo *escalonamento* (scheduling) refere-se à alocação do ponto de vista do consumidor e o termo *alocação de recursos* (resource allocation) atribui-se à alocação do ponto de vista do provedor (FOSTER; KESSELMAN, 2003). O escalonador é responsável por decidir qual *job* irá executar primeiro, caso hajam vários *jobs* prontos para executar, competindo pelo uso da CPU (TANEMBAUM, 2012). O escalonador também é responsável por decidir qual é o processador, ou nó de processamento mais adequado para executar cada *job*, decidir qual é o intervalo de tempo que cada *job* executará em cada processo e alocar os recursos necessários para cada *job*. Sendo assim, um escalonador deverá garantir que os *jobs* executem ao máximo os recursos

disponíveis e terminem no menor tempo possível, sempre respeitando as restrições de tempo ou outras políticas aplicadas aos *jobs* (DONG; AKL, 2006).

Outra definição encontrada por (SCHOPF, 2002) define o escalonamento em Grade Computacional como o processo de tomar decisões envolvendo recursos de diversos domínios administrativos. Sendo assim, o escalonamento de *jobs* pode ser definido com três fases específicas, apresentadas na Figura 18 baseado em (SCHOPF, 2002).

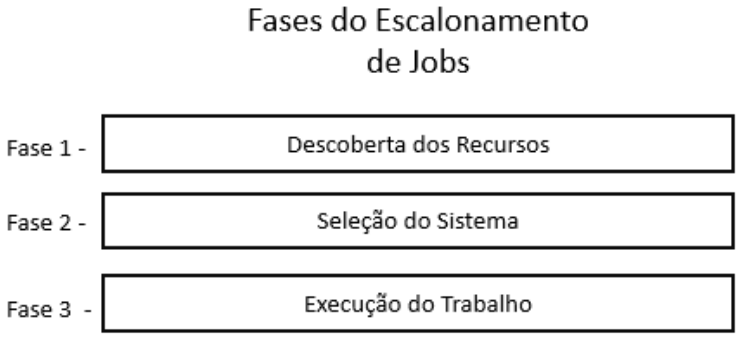


Figura 6 – Fases para o escalonamento de *jobs*. Elaborado pelo autor

A primeira fase, *Descoberta dos Recursos*, consiste em descobrir quais os recursos estão disponíveis para um determinado usuário, verifica quais recursos o usuário tem acesso para a submissão de uma aplicação (conjunto de jobs); averigua quais são os requerimentos para a submissão da aplicação - quantidade de memória, sistema operacional, dentre outros; retorna quais recursos o usuário tem acesso e quais deles atendem aos requisitos da aplicação.

A segunda fase, *Seleção do Sistema*, seleciona qual recurso (ou conjunto de recursos) será destinado para a aplicação. Esta fase reuni as informações que, a partir de dados disponíveis, procura detalhes sobre as informações dinâmicas dos recursos, e os modos como os usuários podem acessá-las e escolhe o recurso para a aplicação a partir das informações reunidas.

A terceira fase é a *Execução da Aplicação*, responsável pela reserva avançada dos recursos, a partir da seleção realizada anteriormente. Depois contém as ações necessárias para executar a aplicação e o monitoramento, afim de que o usuário possa saber o andamento da execução ou fazer o reescalonamento e finalmente a finalização do escalonamento da aplicação.

2.6 CONSIDERAÇÕES PARCIAIS

A Computação Distribuída tem sido o foco de pesquisas nos últimos anos, pois dentro dela existe uma infinidade de estudos focados em ambientes específicos, como o de Grade Computacional e agregados. Com o constante aumento na utilização de recursos, a Computação Verde tem se destacado com a intensão de reduzir o consumo de energia nos ambientes aplicados, pois é de suma importância para a comunidade científica melhorar as tecnologias, reduzindo o impacto no consumo energético. Para a aplicação da Computação Verde, algumas estratégias foram estabelecidas, como os algoritmos eficientes, que é utilizado no desenvolvimento desta pesquisa.

Estudos envolvendo o consumo de Grades Computacionais e agregados estão em ascensão no ambiente científico, dentre os pontos, destaca-se o gerenciamento dos recursos e os escalonadores de *jobs* que visam melhorar o desempenho do ambiente e reduzir o consumo de energia.

3 TRABALHOS RELACIONADOS

Os trabalhos apresentados a seguir tem como principal contribuição os métodos com que buscam solucionar seus problemas propostos para o desenvolvimento de suas pesquisas, mostrando alguns escalonadores para o gerenciamento da Grade Computacional, escalonadores *HEFT* e *Easy Backfilling* e suas politicas de escalonamento, redução do consumo e heurística *overlap*

Sendo assim, serão apresentado os trabalhos estudados que acrescentaram para o desenvolvimento do trabalho e para a elaboração da pesquisa.

3.1 ESCALONAMENTO EM GRADES COMPUTACIONAIS

A heterogeneidade dos recursos é um dos desafios encontrados para escalonamento em Grades Computacionais, além disso, a interconexão de computadores em rede com a utilização de diversos protocolos, as diferentes larguras de banda entre eles, são alguns exemplos de características de ambientes de Grades Computacionais (ANDRIEUX et al., 2003). Estas diversidades aumentam a complexidade dos escalonadores (DONG; AKL, 2006).

A complexidade no escalonamento de *jobs* em Grade Computacional deve-se principalmente à dificuldade em tratar diversos modelos de aplicações e diversos modelos arquiteturais. Dentre os algoritmos de escalonamento existentes, podem-se destacar os baseados em *List Scheduling*, que de maneira geral, estes algoritmos definem a prioridade de cada *job* que encontra-se na fila para ser escalonado e em seguida selecionam o processador que reduz o tempo de finalização de cada *job*. O tempo de finalização de um *job* é o custo de execução deste *job* em um processador, somado ao instante em que o *job* pode iniciar a sua execução. *HEFT* (TOPCUOGLU; HARIRI; WU, 2002), e DCP (Dynamic Critical Path) (KWOK; AHMAD, 1996) são exemplos de algoritmos de *List Scheduling*. As principais desvantagens destes algoritmos são o escalonamento independente de cada aplicação e o conhecimento limitado das cargas atribuídas aos recursos, sendo considerado apenas o próximo instante livre de cada processador.

Para realizar o escalonamento de múltiplas aplicações em ambientes heterogêneos, foi apresentado por (ZHAO; SAKELLARIOU, 2006) heurísticas que, basicamente realizam uma composição de todos os modelos das aplicações a escalonar em um único DAG (Directed Acyclic Graphs). Em uma das heurísticas é criado um DAG composto e agrupando os *jobs* de acordo com os níveis de cada um dos DAGs existentes, ou seja, os *jobs* de um mesmo nível de cada DAG são agrupadas em um

mesmo nível da DAG resultante. Nestas abordagens, os múltiplos DAGs devem estar prontos para serem executados no mesmo instante. Além disso, não é considerado o impacto de um novo escalonamento nas aplicações previamente escalonadas.

Outro desafio está relacionado à autonomia, pois, as instituições possuem suas próprias políticas de escalonamento dificultando a predição de *jobs*. A prioridade que a instituição determina para *job* também influencia no tempo de espera e execução. Os escalonadores de Grades Computacionais podem não ter todas as informações dos recursos, o que dificulta o escalonamento (RODAMILANS, 2009). Para o desenvolvimento do trabalho, os algoritmos de escalonamento estudados para a simulação da heurística *overlap* foram o *HEFT* e o *Easy Backfilling*.

3.1.1 ALGORITMOS DE ESCALONAMENTO HEFT

Uma característica importante no escalonamento é saber para qual recurso o *job* será encaminhado. Isto envolve características dos processos, como de uso intensivo de CPU ou de E/S, largura de banda, quantidade de memória e capacidade de processamento. Em geral, o usuário solicita recursos ao escalonador, e este indica quais recursos o usuário poderá utilizar (TEODORO; CARMO; FERNANDES, 2013).

HEFT é um algoritmo de escalonamento de fluxos de trabalho modelados utilizando um dígrafo acíclico para um número limitado de computadores heterogêneos. Ele é um exemplo de escalonador baseado em lista, ou seja, funciona ordenando os *jobs* a serem executados ao definir prioridades de escalonamento. Em seguida, processa um Job por vez até que um escalonamento válido seja produzido. Para cada *job*, é definido o processador que irá acomodar este *job*. Todo o processamento do *HEFT* é feito antes do escalonamento ser executado, ou seja, é um escalonador off-line (CAMPOS, 2013).

Como o problema de escalonamento de *job*s é NP-difícil, o *HEFT* é frequentemente utilizado como referência na literatura para comparar o desempenho de novas propostas, como por exemplo na pesquisa de (BATISTA; CHAVES; FONSECA, 2011).

Este escalonador recebe como entrada um conjunto de *jobs* modelados como um DAG, um conjunto de nós computacionais, os tempos para executar um *job* em um dado nó e os tempos necessários para comunicar os resultados de um Job para cada um de seus *job*s filhos no DAG. Como saída o algoritmo gera um escalonamento, mapeando cada *job* a uma máquina. O *HEFT* consiste de duas fases principais: priorização e seleção.

Fase de Priorização: Na fase de Priorização, algoritmo *HEFT*, caracteriza os *jobs* que devem ser priorizados considerando o comprimento do caminho crítico, ou

seja, maior caminho, de um dado *job* até o final do fluxo de trabalho. A lista de *jobs* a serem executados é então ordenada em ordem decrescente do comprimento do caminho crítico, com empates sendo decididos aleatoriamente. Com essa ordem, é produzida uma ordenação dos *jobs*, preservando as restrições de precedência do DAG. A prioridade de um *job* N_i é definido recursivamente como mostrada na equação 3.1 (WATANABE et al., 2014):

$$rank(n_i) = w_i + \underset{t\epsilon suc(t_i)}{max} + (\overline{c_{i,j}} + rank(n_j)) \tag{3.1}$$

Onde, w_i é a média do tempo de execução do *job*, t_i tempo de todos os recursos, $suc(t_i)$ são todos os *jobs* sucessores imediatos de t_i, $c_{i,j}$ é a média dos custos de comunicação entre n_i e n_j. Basicamente, *rank* é o comprimento do caminho crítico de um *job* n_i até o *job* final do fluxo, incluindo o custo de computação dos *jobs* n_i (WATANABE et al., 2014). No caso de mais um caminho possível até a saída, é escolhido o de peso máximo. Após essa etapa de construção da lista (fase de priorização), o *job* pronto com peso mais alto é selecionado e associado ao recurso que tem o menor tempo de execução esperado para concluir esse *job* (fase de seleção). O processo se repete até que todos os *jobs* estejam escalonados. Nos trabalhos apresentados por (WATANABE et al., 2014) e (CAMPOS, 2013) o algoritmo *HEFT*, foi adaptado para a economia de energia conseguindo atingir o objetivo esperado pelos autores no ambiente de nuvem computacional, porém, pode ser implementado para outros ambientes como, *agregados* e Grades Computacionais.

Fase de seleção: Na maioria dos algoritmos de escalonamento, o tempo inicial de conclusão de um dado processador P_j para a execução de um *job* é o momento quando P_j completa a execução do último *job* que foi designado a ele. No entanto, o algoritmo *HEFT* possui uma política de inserção que considera a possibilidade de inserir um *job* em um espaço vago entre dois *jobs* já escalonados em um processador (TOPCUOGLU; HARIRI; WU, 2002). Deve-se observar que, o escalonamento neste intervalo deve obedecer às restrições de precedência.

No algoritmo *HEFT*, a busca por um *slot* de tempo vago para um *job* n_i em um processador P_j começa no momento igual ao tempo que o *job* n_i estará pronto para executar em P_j , ou seja, o momento quando todos os dados de entrada de n_i foram enviados pelos predecessores imediatos de n_i ao processador P_j. A busca continua até que seja encontrado um intervalo de tempo capaz de suportar o custo computacional de n_i (TOPCUOGLU; HARIRI; WU, 2002).

Para uma execução simulada do algoritmo *HEFT*, considera-se o fluxo de trabalho apresentado no artigo original do algoritmo (TOPCUOGLU; HARIRI; WU, 2002)

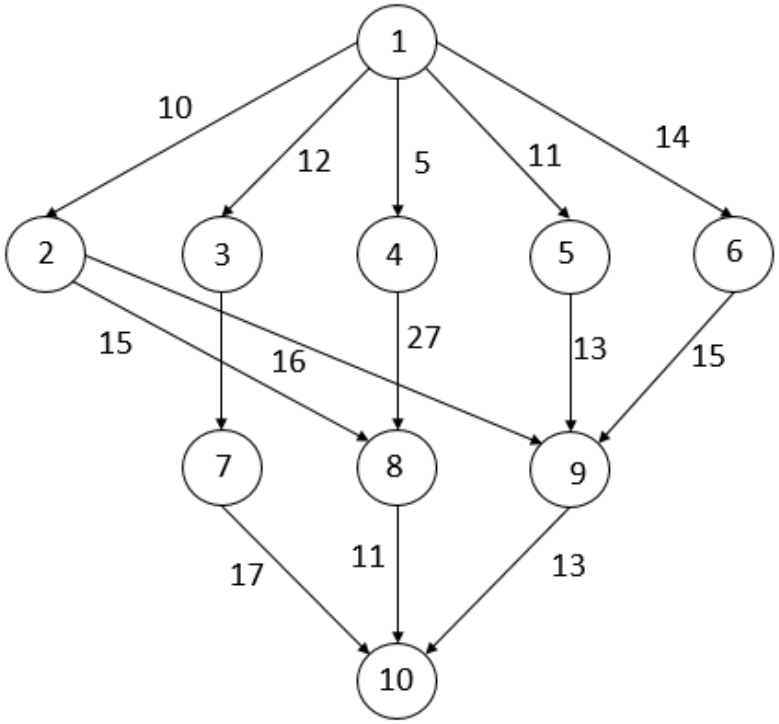


Figura 7 – Fluxo de trabalho Simulado contendo os custos médios de transmissão de dados entre um nó de processamento e outro. Fonte (CAMPOS, 2013)

descrito na Figura 7. Cada vértice do DAG é um *job* a ser executado em um dos três processadores (heterogêneos) disponíveis: P1, P2 e P3. Os rótulos nas arestas indicam o tempo necessário para transferir os resultados de um *job* entre dois processadores distintos.

Durante a pesquisa bibliográfica foram estudados alguns algoritmos de escalonamento que poderiam ser aplicados na proposta, como apresentou (RODAMILANS, 2009), porém seu foco não era redução de energia elétrica e sim uma comparação de desempenho entre algoritmos para avaliar a ferramenta de escalonamento implementada. Outra pesquisa que utilizou o algoritmo *HEFT* com as técnicas *overlap*, cálculo de pertubação e política de inserção foi o (RIOS et al., 2014), onde foi realizado uma análise com o escalonador *HEFT* e seus resultados foram positivos para o objetivo proposto analisando seus desempenhos em diferentes cenários.

3.1.2 Política de escalonamento do Algoritmo *HEFT*

O algoritmo de escalonamento *HEFT*, associa *jobs* a prioridades atribuindo um peso a cada vértice e arco no DAG, ordenando-as em uma lista, de modo que os *jobs* de maior prioridade estão no início da lista e são escalonados primeiro, como pode ser observado no exemplo da Figura 8, com a representação de um grafo dirigido acíclico, DAG (CAMPOS, 2013).

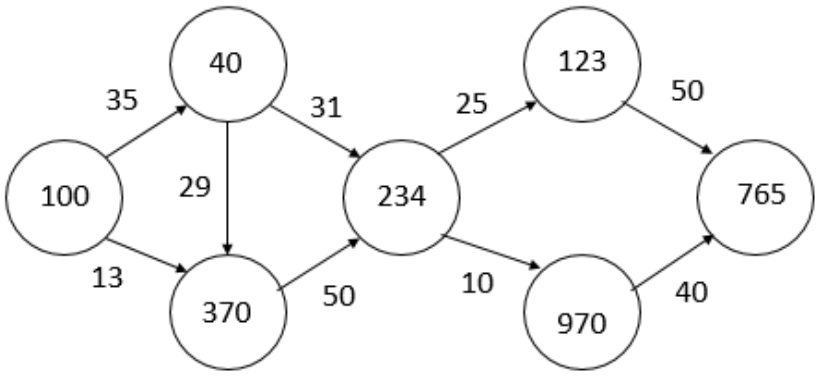


Figura 8 – Grafo Dirigido Acíclico. Fonte: (WATANABE et al., 2014)

O peso para cada *job* consiste no custo médio de execução do *job* em cada recurso computacional, enquanto o peso de cada arco compõe a média dos custos de comunicação entre os *jobs*, ou seja, custos de transferência de dados, como foi apresentado em suas definições (TOPCUOGLU; HARIRI; WU, 2002).

Para desenvolver o escalonador, o algoritmo *HEFT* tem a seguinte política de escalonamento :

1º - Definir os custos computacionais dos e os custos de comunicação entre os com valores médios;

2º - Calcular $Rank(t_i)$ para todos os varrendo o grafo de baixo para cima, iniciando pelo *job* final;

3º - Ordenar os em uma lista de escalonamento utilizando uma ordem não crescente de valores de $Rank(t_i)$;

4º - Enquanto há não escalonados na lista, ele, selecione o primeiro *job*, n_i da lista de escalonamento. Para cada processador, calcula o tempo menor de conclusão do *job* n_i, considerando que ele execute;

5º- Definir o *job* n_i para executar no processador que minimiza o tempo de conclusão do *job* n_i.

Os escalonadores estudados para o desenvolvimento deste trabalho tem suas própias políticas de escalonamento, isto é descrito na forma apresentada para do *HEFT* e que será apresentado a seguir no *Easy Backfilling*, considerado um dos escalonadores mais utilizados em pesquisas científicas com foco em escalonamento (KUMAR; KUMAR; KUMAR, 2013).

3.1.3 ALGORITMOS DE ESCALONAMENTO EASY BACKFILLING

O algoritmo de escalonamento *Easy Backfilling* é um dos algoritmos de gerenciamento de recursos mais utilizados, sua popularidade pode ser explicada pela facilidade de implementação e alta utilização em centros de HPC. Sua principal característica é armazenar os trabalhos pendentes em uma fila, que são ordenadas pela ordem de chegada. Sendo assim, primeiramente o escalonador *Easy Backfilling* varre a fila de trabalhos em ordem, desde que, os trabalhos possam ser executados imediatamente. Esta parte do algoritmo é a mesma política First Come First Serve (FCFS), como pode ser observado na Figura 9. (MU'ALEM; FEITELSON, 2001), (DUTOT et al., 2017).

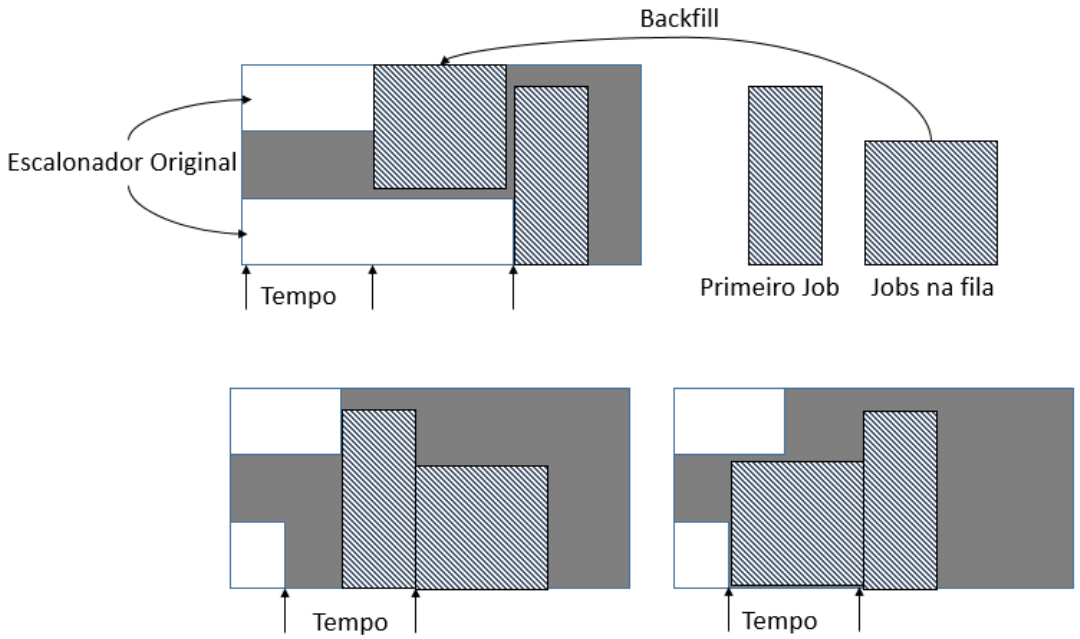


Figura 9 – Método de preenchimento do escalonador Backfilling. Fonte adaptada (MU'ALEM; FEITELSON, 2001)

A Figura 9 demostra como o algoritmo de escalonamento *Easy Backfilling* percorre a fila no momento em que um *job* não pode ser executado, a fim de encaixar um outro *job* que possa aproveitar os recursos disponíveis e ociosos naquele instante de tempo. A diferença é que este método é considerado agressivo, pois permite adiantar qualquer *job* da fila, independente de sua posição de espera desde que obedeça o princípio de não causar atraso ao primeiro da fila (LELONG; REIS; TRYSTRAM, 2017), que tem como sempre a prioridade para a execução.

O escalonador *Easy Backfilling* garante portanto que, uma vez que o *job* vem para o início da fila e é atribuído um tempo de partida, ele não será atrasado. Porém, isso não significa que nenhum outro *job* será atrasado devido ao *backfilling*, pois os

podem ser adiados muitas vezes antes de chegar no início da fila, sendo assim, o algoritmo favorece utilização de tempo de espera (KELEHER; ZOTKIN; PERKOVIC, 2000), (MU'ALEM; FEITELSON, 2001) e (KLUSÁČEK; CIRNE; DESAI, 2018).

Este algoritmo tem duas propriedades que, juntas, criam uma combinação interessante (MU'ALEM; FEITELSON, 2001):

- 1: Os trabalhos na fila podem sofrer um atraso ilimitado. A razão para isto é que, se um trabalho não é o primeiro na fila, os novos trabalhos que chegarem mais tarde podem ignorá-lo. Enquanto tais trabalhos são garantidos para não atrasar o primeiro trabalho na fila, eles podem de fato atrasar todos os outros. Essa é a razão pela qual o sistema não pode prever quando um trabalho será eventualmente executado.

- 2: O atraso de fila para o primeiro trabalho depende apenas de trabalhos que já estão em execução, isso porque, os trabalhos preenchidos não irão atrasá-lo.

3.1.4 Política de escalonamento do Algoritmo *Easy Backfilling*

O objetivo principal do *Easy Backfilling* é melhorar a execução dos quando for possível executa-los antes do *job* inicial. Em alguns casos como observou (MU'ALEM; FEITELSON, 2001), mesmo com uma política de inserção que busca ser determinista, podem ocorrer alguns atrasos nos que foram passados a frente

Para facilitar o entendimento, a política de escalonamento funciona da seguinte maneira (KELEHER; ZOTKIN; PERKOVIC, 2000):

1º - Cada trabalho é descrito pelo número de processadores necessários, com uma estimativa do tempo de execução e o tempo de envio;

2º - O algoritmo tem acesso apenas às informações sobre trabalhos que estão atualmente na fila;

3º - Quando um trabalho é alocado, ele é iniciado em uma partição do tamanho solicitado e pode ser executado com duração igual à estimativa do tempo de execução;

4º - Se esse limite é atingido, o trabalho é morto para evitar atrasar outros;

5º - Quando um trabalho termina, o algoritmo verifica se o primeiro trabalho pode ser inicializado; se o número de processadores e suficiente.

6º - Então todos os trabalhos que estão atualmente em execução no sistema são classificados em ordem de sua conclusão;

7º - O escalonador calcula o tempo em que haverá um número suficiente de processadores disponível para o primeiro trabalho na fila.

8º - Processadores livres são denominados ”extra nós ”

Sendo assim, o algoritmo repete todas verificações, até que todos os estejam devidamente enfileirados para o seu escalonamento. Feito isto, o momento seguinte foi encontrar formas para reduzir o consumo de energia elétrica no ambiente de Grade Computacional.

3.2 MÉTODOS DE REDUÇÃO DE CONSUMO DE ENERGIA

Como alternativa para reduzir o consumo de energia em Grade Computacional, a pesquisa desenvolvida por (FRANÇA, 2014), traz uma solução para o problema de escalonamento de baseado em algoritmos genéticos para distribuir de maneira mais eficiente em um ambiente de Grade. Seu objetivo principal foi reduzir o tempo total de execução dos , permitindo assim, diminuir o tempo de processamento e aumentar a eficiência da Grade. Também foi desenvolvido por (CARVALHO; SANTANA et al., 2011) uma heurística chamada de HGree, que consiste em atribuir de fluxo de trabalho para recursos da rede e definir a sua ordem de execução. Também com o intuito de reduzir o consumo de energia (WANG; HAN; WANG, 2018) desenvolveu um agendador de , baseado no *Easy Backfilling*, com reconhecimento de energia aplicando um método de controle baseado em regras e levando em consideração perfis de potência e aceleração de dados reais para melhorar a eficiência energética. Os resultados intensivos das simulações mostraram que, o método proposto é capaz de alcançar a máxima utilização de recursos de computação em comparação com os algoritmos de programação de linha de base, mantendo o custo de energia abaixo do limite.

O trabalho de (BORRO, 2014) tem com objetivo o contexto do gerenciamento energético em grades móveis, onde o mesmo propõe dois algoritmos de escalonamentos Maximum Regret e Greedy que buscam minimizar o consumo de energia, considerando cenários estáticos e dinâmicos. Tais algoritmos foram projetados a partir de soluções heurísticas para o problema de escalonamento ciente do consumo de energia em grades móveis. Entretanto, este trabalho não se preocupa com dispositivos conectados a uma fonte de energia.

Também pode-se citar o trabalho de (LELONG; REIS; TRYSTRAM, 2017) que não tinha como principal objetivo a redução de energia e sim a redução do tempo de espera das aplicações, onde considera uma abordagem para o ajuste automático da heurística *Easy Backfilling* aplicada em plataformas HPC. Mais precisamente, o problema consiste em escolher uma política de reordenação para a fila de trabalhos. Sua principal contribuição e buscar reduzir o tempo médio de espera utilizando duas formas especificas chamadas de *full feedback* e *bandit feedback*, que são respecti-

vamente, método que utiliza simulação com dados coletados do sistema passando a carga de trabalho atual e o segundo método é um simulador menos abrangente que tem como principal finalidade medir o desempenho do sistema. Ambos os métodos são avaliados especificamente, desenvolvido e simulado. Com essas abordagens os autores conseguiram uma redução do tempo médio de espera em até 40%.

Outro trabalho interessante foi o apresentado por (KRAEMER et al., 2018) onde demostrou que, as plataformas HPC podem hospedar e executar alguns trabalhos em nuvem com baixa interferência em ambientes HPC e um baixo número de violações de tempo de resposta, e foi proposto uma estratégia de programação de , visando reduzir o número de violações de tempo de resposta de na nuvem sem interferir na execução dos de HPC.

A heurística *overlap* é utilizada em alguns trabalhos correlatos para demostrar e simular desempenho de processadores, como foi apresentado por (NETO, 2004), onde definiram um conjunto de roteiros para o ensino de arquitetura dos computadores com enfoque em arquiteturas superescalares. O procedimento é baseado em verificação da influência dos parâmetros arquiteturais dos processadores utilizando *overlap* , em termos funcionais e de desempenho.

Foi aplicada também a heurística *overlap* com a sobreposição de slots de processamento com o intuito de reduzir o consumo de energia e melhorar o tempo de execução das aplicações, como apresentaram (RIOS et al., 2014) e (ISMAIL, 2012), que verificaram, respectivamente, o comportamento de suas aplicações com a sobreposição de em slots de aproximadamente 10% e também para fazer uma análise comparativa de desempenho e custo de aplicações em diferentes cenários e plataformas.

A tabela 1, apresenta um breve resumo de trabalhos estudados durante a execução da pesquisa científica, onde pode-se destacar algumas aplicações de escalonadores, as métricas que foram consideradas para a execução e se a heurística *overlap* foi aplicada.

Tabela 1 – Trabalhos correlatos aplicados em ambientes de HPC e seus principais critérios. Fonte: Próprio Autor.

Autor	**Escalonador**	**Métricas Aplicadas**	**Overlap**
CAMPOS, 2013	HEFT	Tempo de Execução / Desligar máquinas	Não
WATANABE et al, 2014	HEFT	Tempo de Execução / Desligar máquinas	Não
CARVALHO; SANTANA et al, 2011	HEFT	Reduefinir a ordem de execução	Não
TOPCUOGLU;HARIRI;Wu, 2002	HEFT	Tempo de execução	Não
BORRO, 2014	Maximum Regret e Greedy	Tempo de execução / Desligar máquinas	Não
FRANÇA, 2014	Algoritmo Genético	Reduzir o tempo de processamento	Não
LELONG; REIS; TRISTRAM, 2017	Easy Backfiling	Reduzir tempo de espera	Não
KRAEMER et al, 2018	Easy Backfiling	Reduzir o tempo de resposta	Não
WANG, 2018	Easy Backfiling	Tempo Execução	Não
NETO, 2004	-	Melhorar desempenho	Sim
RIOS et al, 20014	Slot / CPOP/HEFT	Tempo de execução	Sim

Com a tabela, pode-se verificar que, em pesquisas científicas recentes utilizando os escalonadores *HEFT* e *Easy Backfilling*, não foram encontrados aplicações da heurística *overlap* utilizando os escalonadores aqui apresentados. Nos poucos trabalhos encontrados, como o do (RIOS et al., 2014) essa heurística foi aplicada com o objetivo principal de comparação de desempenho dos recursos, como pode ser observado na coluna de métricas aplicadas, e não para a redução de consumo de energia, ou seja buscando uma heurística para o escalonamento verde.

3.3 HEURÍSTICAS DE ESCALONAMENTO VERDE COM *OVERLAP*

O presente trabalho consiste em aplicar a heurística *overlap* (sobreposição) com o foco em escalonamento verde *Scheduler Resource Computational Green* aplicadas em Grades Computacionais. Nestes ambientes de Grades Computacionais, existem uma grande quantidade de em execução, portanto é importante que a heurística implementada garanta que todos os processadores utilizados estejam realmente sendo ocupados, para que a eficiência energética seja atingida.

Para a overlap (sobreposição), um slot é considerado satisfatório quando o tempo de execução do *job* é igual ou inferior ao tamanho do slot. Contudo, a rigidez na seleção dos intervalos impede o uso de grande capacidade de processamento dos recursos, principalmente ao somar-se os slots ociosos (RIOS et al., 2014).

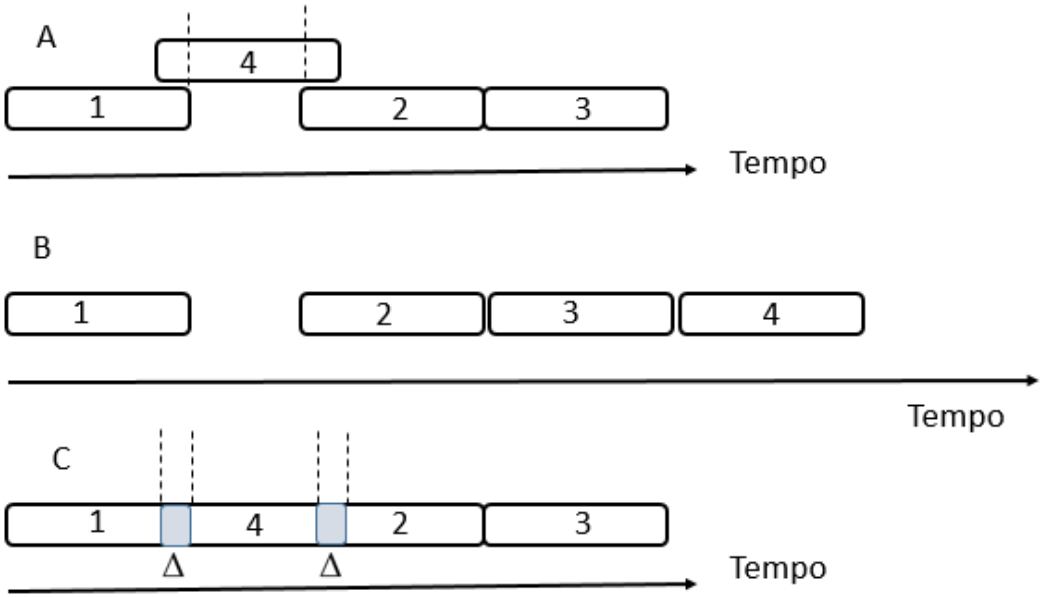


Figura 10 – Exemplo de sobreposição de . Fonte adaptada: (RIOS et al., 2014)

A Figura 10 exemplifica uma situação de sobreposição de , apresentado por (RIOS et al., 2014). Os 1, 2, 3 e 4 não necessariamente fazem parte da mesma aplicação e os números representam a ordem de chegada no escalonador, podendo não

existir relação entre eles. A situação inicial da Figura 10 representa um processador com os 1, 2 e 3 alocados para execução e a chegada do *job* 4 no instante inicial, enquanto o *job* 1 está sendo executado. O escalonador buscaria alocar o *job* recém chegado em um intervalo de ociosidade do processador mas, como não ha espaço no slot suficientemente grandes, o *job* 4 é alocado no final, depois da execução do *job* 3. Isso é demostrado na situação seguinte. Porém, com o uso da política de *overlap*, o escalonador tenta alocar o *job* 4 sobrepondo um percentual determinado pelo usuário.

O tempo para a execução de um *job* t_i^E em um processador é basicamente o somatório entre o custo do *job* no processador e o tempo total de execução do *job* predecessor mais lento, como apresenta na equação 3.2 (RIOS et al., 2014):

$$t_i^E = (w_{i,q}) + max_{jpred(i)}(t_j^E) \tag{3.2}$$

Para verificar como o processador será utilizado, se faz necessário analisar o custo computacional que definirá o tamanho do *job* no processador e também utilizado para encontrar slots livres com o mesmo tamanho ou superior, ou seja, onde o tempo de execução dos *jobs* n_i, t_E^t, é menor ou igual que o valor resultante da subtração entre o instante de início do próximo slot ocupado pelo *job* n_K, t_I^k , e o instante final de execução do slot predecessor ocupado pelo *job* n_J, t_F^j , como na equação 3.3 (RIOS et al., 2014):

$$t_E^t \leq t_I^k - t_F^j \tag{3.3}$$

Para realizar a alteração da política de inserção com a sobreposição de slots, foi realizada a substituição da equação 3.2 pela equação 3.3, onde o *delta* representará a variação da sobreposição máxima permitida (RIOS et al., 2014). Como exemplo utilizando o delta = 0,30 , significa que será permitido uma sobreposição de no máximo 30% de um slot.

$$t_E^t \leq (t_I^k - t_F^j) + ((\Delta * t_E^t)/100) \tag{3.4}$$

De maneira geral, a sobreposição de slots significa o compartilhamento de tempo no processador, alvo para execução de dois ou mais *jobs*, ou seja, esta política permite que haja uma perturbação de um novo *job* naqueles previamente alocados, visando a utilização de um slot que seria insuficiente para execução de um *job* mas que, de outro modo, permaneceria ocioso (RIOS et al., 2011), (RIOS et al., 2008).

Para a aplicação e verificação da heurística *overlap*, foi estudado maneiras para realizar a simulação em alguns cenários, para isso foi necessário a escolha de um simulador para o ambiente de Grades Computacionais.

3.4 SIMULADORES PARA GRADES COMPUTACIONAIS

Alguns algoritmos de escalonamento são avaliados por simulação na literatura, como o *SimGrid*, que é um simulador usado para avaliar heurísticas, protótipos ou até mesmo avaliar aplicativos de MPI (Message Passing Interface). Essa ferramenta fornece modelos e APIs prontas para uso. Com características escaláveis e com vários recursos disponíveis para teste. Possui *plugins* importantes e de fácil migração para outras ferramentas (HIROFUCHI; LEBRE; POUILLOUX, 2018).

O simulador *Alea* é baseado no Kit de ferramentas de simulação *GridSim* e permite comparar diferentes algoritmos de escalonamento. *Alea* e *Batsim* escolheram confiar em estruturas de simulação existentes em vez de implementar tudo do zero. No entanto, o *Alea* não permite comparações diretas com código RJMS (Resource and Jobs Management Systems), nem permite qualquer linguagem de programação. Outro fator importante é que ele não foi validado em um ambiente real (KLUSÁČEK; RUDOVÁ, 2010).

Outra abordagem interessante foi encontrada no artigo (PASCUAL; MIGUEL-ALONSO; LOZANO, 2015), onde ele consiste em usar offline o simulador de rede refinado INSEE (PEREZ; MIGUEL-ALONSO, 2005) para obter o tempo de execução preciso em todas as configurações de trabalho possíveis. Já no artigo de (PASCUAL; MIGUEL-ALONSO; LOZANO, 2015), a ideia é propor políticas de colocação profissional que garantam que nenhuma interferência na rede pode ocorrer entre os trabalhos, permitindo usar tempos de execução offline enquanto simula uma carga de trabalho on-line.

Também foi implementado como base o *SimGrid* a ferramenta de simulação *Simbatch*. O projeto foi mantido de 2007 a 2015, mas foi atualizado recentemente com atualizações do *SimGrid*. Uma característica do *Simbatch* (GAY; CANIOU, 2005).

Alguns RJMSs existentes podem ser utilizados em simulação, como apresentou (AHN et al., 2014), no qual o modelo de simulação é incluído. Também destaca-se o *Slurm*, onde inicialmente não foi projetado para ser um simulador, porém o trabalho foi conduzido para permitir o ajuste de parâmetros aplicados em simulação, pode ser adaptado ao simulador Batsim.

Na Tabela 2, encontram-se os trabalhos correlatos aplicados em Simuladores de Grades Computacionais e suas principais características.

Foram estudados alguns simuladores e suas características como linguagem de programação, plataformas aplicadas, ambientes que os projetos possuem, licenças, e por fim as arquiteturas para a execução da pesquisa, para buscar algum simulador, em que a proposta fosse executada sem causar dúvidas nos seus resultados.

Tabela 2 – Trabalhos correlatos aplicados em Simuladores de Grade Computacional

	Batsim	**GridSim**	**SimGrid**	**Alea**	**Simbatch**
Tipo de Ferramenta	Simulador	Simulador	Simulador	Simulador	Simulador
Implementação	python, C++	Java	Java, C	Java	C
Plataformas	Linux	S.O com JVM	S.O com JVM	S.O com JVM	S.O com JVM
Ambiente de Projeto	Biblioteca	Biblioteca	Biblioteca	Biblioteca	Biblioteca
Licença	Opensource	Opensource	Opensource	Opensource	Opensource
Arquitetura	Def. pelo usuário	Def. parc. pelo usuário	Def. pelo usuário	Def. parc. pelo usuário	Def. pelo usuário

Desta forma destacou-se o Batsim, por ser um simulador desenvolvido para o ambiente da Grid'5000, ser atualizado com frequência e possuir documentação eficiente para a implementação das plataformas.

3.5 CONSIDERAÇÕES PARCIAIS

A Grade Computacional é um ambiente de HPC com recursos disponíveis para a execução de *jobs*, sendo necessário a utilização de escalonadores para o gerenciamento destes recursos, que envolve diversas características importantes ao direcioná-los. Os escalonadores *HEFT* e *Easy Backfilling*, possuem características diferentes ao escalonar os *jobs*, respectivamente, um prioriza a ordenação pelo caminho crítico considerado o maior caminho, e o outro utiliza o método de escalonamento por ordem de chegada dos *jobs*.

Com o objetivo de reduzir o consumo de energia elétrica neste ambiente, foram estudados métodos que utilizassem formas de reduzir o consumo, sem afetar seu desempenho. Desta forma encontrou-se a heurística *overlap*, que sobrepõe *jobs* por alguns instantes, compartilhando recursos, buscando alocá-los em slots ociosos, reduzindo desta forma o impacto no consumo de energia em Grades Computacionais e podendo ser aplicado nos escalonadores estudados.

4 SOLUÇÃO DE ESCALONAMENTO COM A HEURÍSTICA OVERLAP

O escalonamento em Grades Computacionais é importante para o direcionamento de recursos disponíveis para a execução de *jobs* em ambientes de HPC. Para tanto, se faz necessário a utilização de métodos e heurísticas que consigam de alguma maneira reduzir o consumo de energia elétrica, minimizando o impacto no tempo de execução.

4.1 DEFINIÇÃO DO PROBLEMA

A alta demanda de recursos utilizados em Grades Computacionais nas ultimas décadas, implicou em um maior consumo de energia para a execução de suas tarefas. Desde então, estudos buscando métodos para reduzir este consumo estão sendo cada vez mais utilizados em pesquisas científicas.

No cenário de Grades Computacionais o monitoramento dos recursos é realizado com frequência, em busca de alocações de *jobs* nos recursos disponíveis para sua execução, no entanto a preocupação inicial era o desempenho desses recursos com a alta demanda de serviços. No decorrer dos anos, essa preocupação começou a voltar-se para o alto consumo em decorrência de sua utilização, desta forma a comunidade científica começou a estudar maneiras para minimizar este impacto no consumo energético.

Os escalonadores não têm como função principal a redução do consumo de energia, pois visam escalonar os *jobs* para recursos disponíveis sem verificar a ociosidade. Para tal fato, os algoritmos de escalonamento foram os mais estudados, sendo assim, o presente trabalho tem como objetivo utilizar a ferramenta de simulação Batsim com a implementação dos escalonadores *HEFT* e *Easy Backfilling* utilizando a heurística *overlap* buscando sobrepor por alguns instantes pré definidos os *jobs* nos slots de processamento que estão ociosos.

4.2 ALGORITMOS DE ESCALONAMENTOS COM APLICAÇÃO DA HEURÍSTICA *OVERLAP*

Para a implementação da heurística *overlap*, os escalonadores escolhidos foram o *HEFT* e o *Easy Backfilling*, com formas de ordenações de *jobs* e políticas de escalonamento diferenciadas.

- O algoritmo *HEFT* é baseado em listas, processa um *job* por vez e pode ser aplicado a uma rede com computadores heterogêneos, como explicado na se-

ção 3.1.1. Portanto primeiramente foi implementado o algoritmo e aplicado suas fórmulas para a ordenação dos *jobs* e depois foi implementado a heurística *overlap* dentro do escalonador.

- A política de escalonamento do algoritmo Easy Backfilling consiste em, o primeiro *job* da fila é sempre o primeiro a chegar, porém isso não garante que seja o primeiro a executar. Isto porque, caso a fila possua algum *jobs* menor que se encaixe em espaços ociosos, ele poderá passar a sua frente, desde que isso não ocasione atrasos para a sua execução, como descrito na seção 3.1.3.

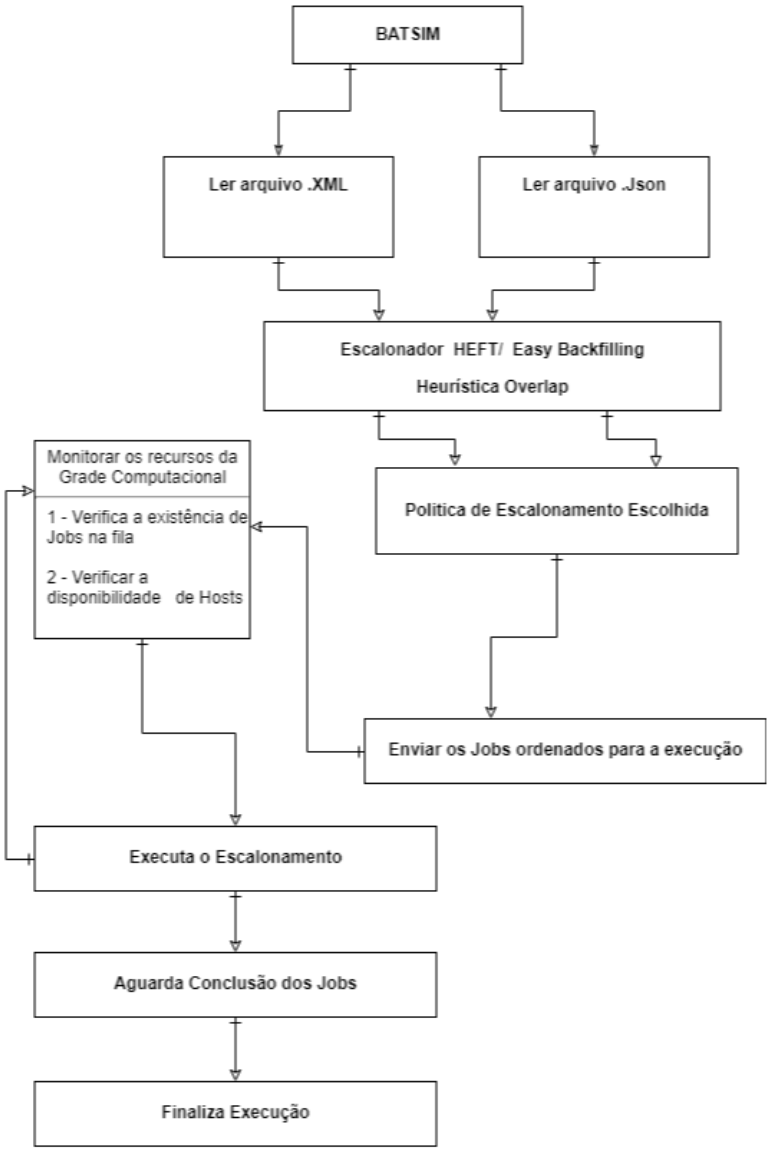


Figura 11 – Diagrama de Escalonamento dos algoritmos HEFT e Easy Backfilling. Fonte: Próprio Autor

O Diagrama apresentado na Figura 11, apresenta a sequência de execução dentro da ferramenta Batsim. Inicialmente o Batsim faz a leitura das plataformas utilizadas (arquivo .XML) e também os workloads desejados para a monitoração (arquivo .Json). Depois é escolhido o escalonador que executara naquela simulação. Cada escalonador tem suas próprias políticas de escalonamento, e por esse motivo, atua de maneira diferente em determinadas fases de escalonamento. 1 - *HEFT* que realiza a ordenação por ordem decrescente dos *jobs*; 2 - *Easy Backfilling* que realiza por ordem de chegada dos *jobs*.

Conforme ilustrado na Figura 11, após a escolha dos arquivos XML e Json, a ferramenta Batsim realiza a seleção da política de escalonamento, algoritmo *HEFT* ou *Easy Backfilling* representada nas Figuras 12a e 12b, onde observa-se como é o escalonamento de ambos os algoritmos utilizados para a pesquisa.

Figura 12 – Pseudocódigo para os e escalonamento dos Algoritmos *HEFT* e *Easy Backfilling* com heurística *overlap*

Algoritmo de escalonamento HEFT com Overlap
1. Para todo Job: 1. Calcular o custo médio de execução; 2. Calcular o custo médio de comunicação; 3. Calcular o valor de *rank*; 2. Ordenar as tarefas em uma lista de escalonamento em ordem decrescente, com base no valor de rank; 3. Enquanto a lista não estiver vazia: 1. Remover a primeiro Job; 2. Para cada processador disponível: 1. Calcular o tempo de conclusão da tarefa; 3. Selecionar o processador que conclui o Job em menor tempo; 4. Escalonar a tarefa em uma lista de Jobs escalonados; 4. Fim enquanto. 5. Enquanto a lista não estiver vazia: 1. Se existem recursos suficientes e se te < (tf - tf) + ((Δ* te) / 100): 1. Executar Job. 6. Fim enquanto.

(a) Pseudocódigo utilizado para a implementação do escalonador *HEFT*. Fonte: Próprio Autor

Algoritmo de escalonamento Easy Backfilling com Overlap
1. Para todo Job: 1. Ordenar as tarefas em uma lista de escalonamento em ordem de chagada; 2. Realiza o backfilling com os Jobs da fila 2. Para cada processador disponível: 1. Calcular o tempo de conclusão da tarefa; 3. Selecionar o processador que conclui o Job em menor tempo; 4. Escalonar a tarefa em uma lista de Jobs escalonados; 4. Fim enquanto. 5. Enquanto a lista não estiver vazia: 1. Se existem recursos suficientes e se te < (tf - tf) + ((Δ* te) / 100): 1. Executa Job. 6. Fim enquanto.

(b) Pseudo código utilizado para a implementação do escalonador *Easy Backfilling*. Fonte: Próprio Autor

Nas Figuras 12a e 12b encontra-se a descrição dos pseudocódigos utilizados para as implementações da heurística *overlap* nos escalonadores *HEFT*, que inicialmente calcula o custo médio da execução, depois executa a equação do *rank* para ordenar de forma decrescente os *jobs* a serem escalonados, verifica os processadores disponíveis para a execução, escalona os *jobs* e por fim, executa a equação 3.4 utilizada para a aplicação da heurística *overlap*. Para a utilização do escalonador *Easy Backfilling*, onde sua política de escalonamento é por ordem de chegada, o escalonamento dos jobs, se torna mais simples ao comparar com outros escalonadores.

Onde, para cada job na fila será realizado a verificação de processadores disponíveis que concluam a execução no menor tempo. Depois de escalonados, pode-se aplicar novamente a equação 3.4 para a utilização da heurística *overlap*.

Para essa implementação da heurística *overlap* em ambos os escalonadores, foram aplicadas as fórmulas descritas na seção 3.3, onde verifica-se o tempo de execução dos *jobs*, depois o tamanho do *job* no processador, os espaços em slots ociosos, e por fim, a aplicação do percentual de *overlap* desejado, analisando todas as informações relevantes encontradas no cabeçalho de cada *job*.

Depois de utilizado um dos escalonadores *HEFT*/ ou *Easy Backfilling*, os *jobs* ordenados ficam na fila para a execução, onde o escalonador é responsável por monitorar todos os recursos que estão disponíveis e a existência de *jobs* ainda na fila, executar o escalonamento, aguardar a conclusão dos *jobs* e finalizar a execução.

4.3 CONTRIBUIÇÃO EFETIVA NA APLICAÇÃO DA HEURÍSTICA *OVERLAP* NOS ESCALONADORES HEFT E EASY BACKFILLING

Uma das soluções encontradas para buscar reduzir os espaços ociosos que encontram-se nas execuções de *jobs* foi a utilização da heurística *overlap*, que busca sobrepor um percentual de cada *job* que está entre o espaço ocioso do *slot* de processamento.

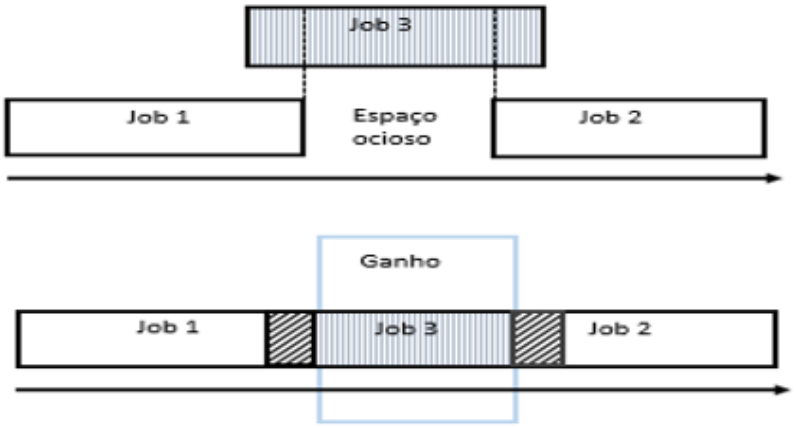


Figura 13 – Diagrama de Escalonamento dos algoritmos HEFT e Easy Backfilling. Fonte: Próprio Autor

Pode-se observar na representação da Figura 13, que o ganho efetivo da aplicação da heurística *overlap*, ocorre no momento em que o *job* executa no espaço que encontrava-se ocioso. Desta maneira observa-se que, quanto menor for o percentual de sobreposição, maior será o ganho efetivo na redução de consumo, sem afetar o desempenho. Pois quanto maior o percentual de *overlap* maior poderá ser, seu consumo

e o tempo de execução. Para a comprovação dos experimentos, foi realizada as simulações com a ferramenta Batsim em cenários e *workloads* simulando um ambiente real da Grid'5000.

4.4 SIMULAÇÃO

Quando trabalha-se com sistemas distribuídos, em larga escala de recursos heterogêneos, a simulação torna-se uma ferramenta adequada para a análise de algoritmos. Isto porque é eficaz em trabalhos que exigem um grande número de recursos e usuários envolvidos para sua execução (BUYYA; MURSHED, 2002). A motivação para a utilização de simulação é devido a facilidade de criação de um ambiente de testes, já que uma plataforma real requer um alto custo e demora, e também não é um ambiente controlável e de repetições para experimentação e avaliação de experimentos.

A simulação tem sido pesquisada e aplicada com sucesso por diversas pesquisas científicas, pois ela permite estudos de diversas métricas como desempenho, consumo, tempo de execução dentre outras, poupando assim tempo e custo (SHARMA; HSU; FENG, 2006).

4.4.1 FERRAMENTA DE SIMULAÇÃO BATSIM

Batsim é um simulador RJMS, open source, que permite simular o comportamento de um plataforma computacional na qual as cargas de trabalho são executadas de acordo com o escalonador. A lógica de simulação do *Batsim* é dividida em dois componentes, como mostra a Figura 14. O simulador orquestra a simulação, gerencia a plataforma e lida com entradas e saídas. O outro componente é responsável por tomar a maioria das decisões e pode, portanto, incluir partes de RJMS reais.

Os simuladores permitem modelar diversos tipos de recursos heterogêneos, como agregados, banco de dados e computadores de memória compartilhada. Os recursos podem ser modelados utilizando diferentes políticas de alocação para cada domínio, como espaço compartilhado (space-shared) e tempo compartilhado (time-share). Também é possível desenvolver sua própria política de alocação e utilizá-la nestas ferramentas (BUYYA; MURSHED, 2002).

A ferramenta de simulação *Batsim,* foi desenvolvida para a *Grid'5000* (GRID5000, 2018), baseando-se no Kit de ferramentas de simulação *SimGrid*. A Figura 15 é uma visão simplificada do seu funcionamento, onde o *Batsim* orquestra a simulação e gerencia os recursos, enquanto o componente de tomada de decisão decide o melhor caminho a seguir. Os dois componentes interagem através de um protocolo baseado em eventos. Esta separação clara permite usar procedimentos de tomada de decisões provenientes de RJMSs na simulação. (POQUET, 2017a)

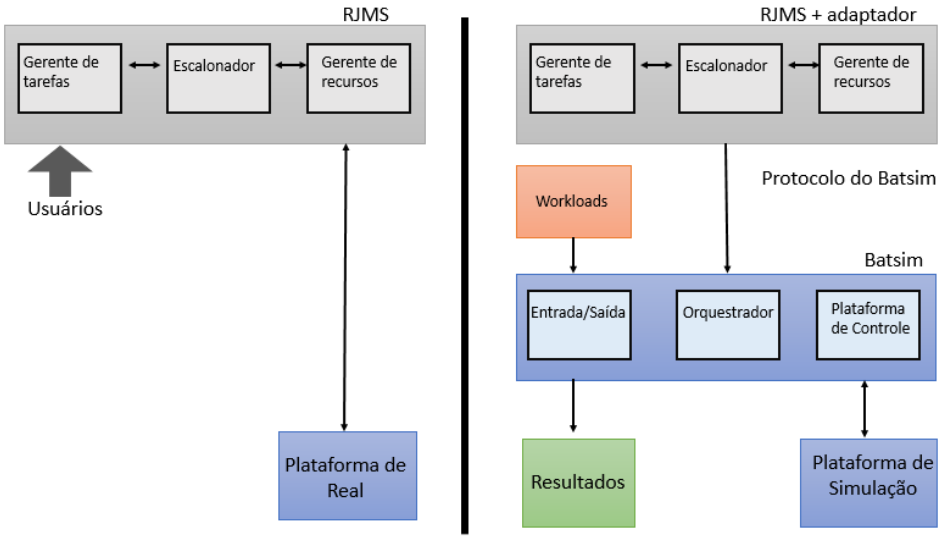


Figura 14 – Estrutura para simulação com o BatSim Real e Simulada. Fonte adaptada (POQUET, 2017b)

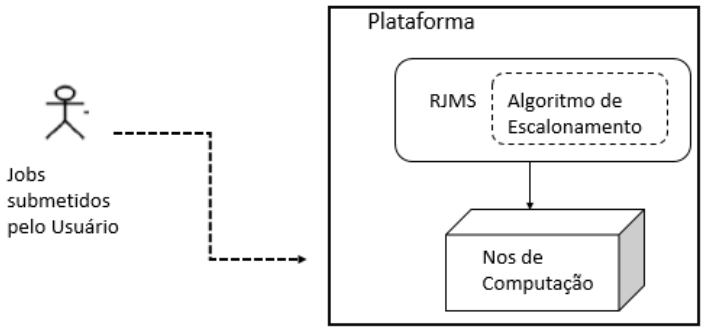


Figura 15 – Visão simplificada de uma plataforma semelhante a HPC. Usuários se comunicam com o RJMS, que orquestra como os recursos serão utilizados. Fonte adaptada (DUTOT et al., 2017)

A carga de trabalho é a principal entrada para o Batsim. Ela pode ser utilizada para definir o que o usuário deseja executar ao longo do tempo. O Batsim separa a carga de trabalho em dois conjuntos que são, trabalho e perfil. Trabalho define as solicitações do usuário, normalmente essa informação e utilizada pelo escalonador para tomada de decisões e o perfil define a informação que o simulador de plataforma usa para simular como o aplicativo deve ser executado (BATSIM, 2018b).

Para a simulação o *Batsim* utiliza a noção de perfil de trabalho, onde cada trabalho é associado a um perfil, mas um perfil pode ser associado a vários trabalhos. Cada perfil é definido pelo arquivo *JSON* da carga de trabalho (BATSIM, 2018b).

O componente *Batsim* demostrado na Figura 16 é responsável por simular os recursos computacionais, enquanto o componente *Decisão* é responsável por averi-

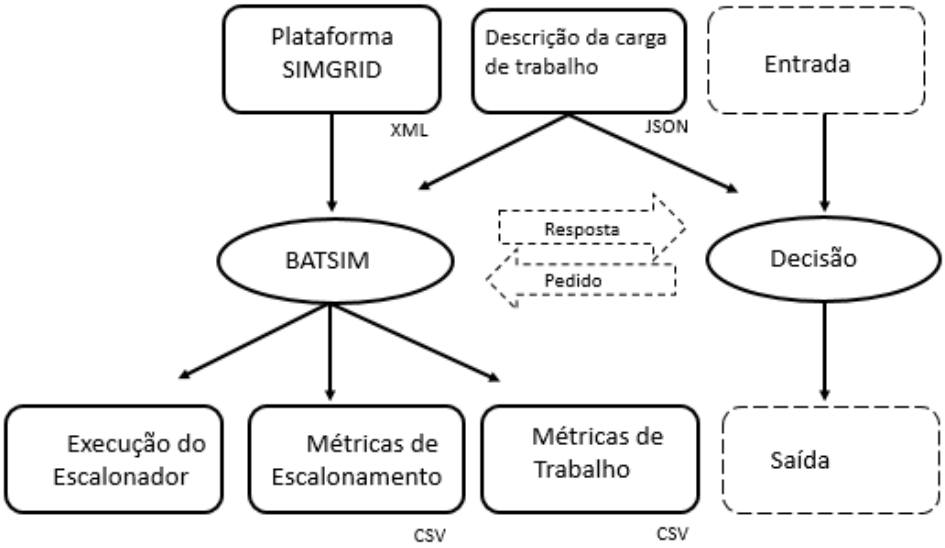


Figura 16 – Processos de simulação com a ferramenta BatSim. Fonte adaptada (DUTOT et al., 2015)

guar a melhor forma de economia de energia. As decisões de agendamento incluem a execução de trabalhos. As decisões relacionadas à energia incluem a alteração do estado de energia em uma máquina, ou seja, ligar ou desligar se necessário. O processo de submissão de trabalhos é responsável pela leitura de uma carga de trabalho, os colocando no momento certo da simulação. Para isso ele faz uma interação com os trabalhos em ordem crescente de tempo de submissão e envia uma mensagem para o servidor (DUTOT et al., 2015).

O *Batsim* utiliza um *plugin* para o cálculo de consumo energético importada do *SimGrid* [1],que tem quatro parâmetros (DUTOT et al., 2015):

1 - *Idle*: quando o host está ativo e em execução, mas sem nada para fazer;

2 - *OneCore*: quando apenas um núcleo está ativo, em 100%;

3 - *AllCores*: quando todos os núcleos do host estão em 100%;

4 - *Off*: quando o host está desligado.

Os valores dos parâmetros estão apresentados na Tabela 3.

Tabela 3 – Tabela utilizada para cálculo do consumo de energia elétrica pelo Batsim. Fonte Próprio Autor

Consumo do Host	***Estado dos núcleos dos Hosts***
100 Watts	**Idle**- quando seu host está funcionando, mas sem nada para fazer
120 Watts	**OneCore-** quando apenas um núcleo está ativo em 100%
200 Watts	**AllCores** - quando todos os núcleos dos hosts estão em 100%
10 Watts	**Off**- quando os hosts estão desligados

[1] <http://simgrid.gforge.inria.fr/simgrid/3.19/doc/platform.html>

Outra consideração relativa ao *plugin* importado pelo Batsim e relacionado a carga, é que a ferramenta *SimGrid* [2] também considera carga de 100% em suas execuções, pois toda vez que a CPU inicia ou para de fazer alguma coisa ela recalcula novamente a carga. Portanto, se um núcleo estiver com carga de 50% ao longo de um período, isso significa que ele está com carga 100% na metade do tempo e com carga de 0% no restante do tempo.

4.4.2 CARACTERÍSTICAS DOS JOBS NO BATSIM

As plataformas estudadas neste trabalho com a ferramenta **Batsim** são compostas de recursos computacionais heterogêneo. Estes recursos disponíveis para a simulação, podem ter múltiplos estados de potência, com velocidade de computação e consumo associado a cada estado de energia (DUTOT et al., 2015).

A simulação que foi aplicada na dissertação é a RMJS, mencionada na seção 4.3.1. Conforme ilustrado na Figura 17, onde a principal contribuição dessa metodologia, é maximizar a simulação separando-a da plataforma do orquestrador, onde essa separação busca impedir que alguns trabalhos possam ser duplicados. Quanto ao método RMJS, pode ser facilmente adaptado para ser utilizado com qualquer algoritmo de escalonamento dentro da ferramenta *Batsim* (DUTOT et al., 2017).

Com as definições mencionadas no capítulo 2.5 , pode-se observar na Figura 18 como foi aplicado os escalonadores utilizados para o desenvolvimento do trabalho no ambiente de simulação Batsim dentro das fases de escalonamento. Ambos os escalonadores *HEFT* e *Easy Backfilling* respectivamente, passam pelas fases 1 e 2 depois realizam suas formas de escalonamento de *jobs* (utilizando a fórmula do rank e ordenando por chegada) e por fim finalizam com a fase 3, executando os trabalhos.

Quanto as características dos *jobs* que a ferramenta *Batsim* utiliza são: tempo de liberação, tempo de execução, tempo de processamento e número de recursos solicitados. Onde o número de recursos são especificados pelo usuário e não são conhecidos pelo escalonador antes do seu tempo de liberação. O tempo de processamento permanece desconhecido até que o *job* seja concluído (DUTOT et al., 2017). Dependendo do modelo utilizado, o tempo de processamento pode ser uma quantidade fixa de tempo enquanto o trabalho está em execução.

Outra fator interessante, é que o Batsim possui uma lista pré definida de dicionário de dados dos *jobs*, porém, essa lista pode ser alterada pelo usuário, acrescentando novos campos com informações que cada escalonador deseja para sua execução (BATSIM, 2018a). Os trabalhos pré definidos pela ferramenta possui os seguintes campos:

[2] <http://simgrid.gforge.inria.fr/simgrid/3.20/doc/>

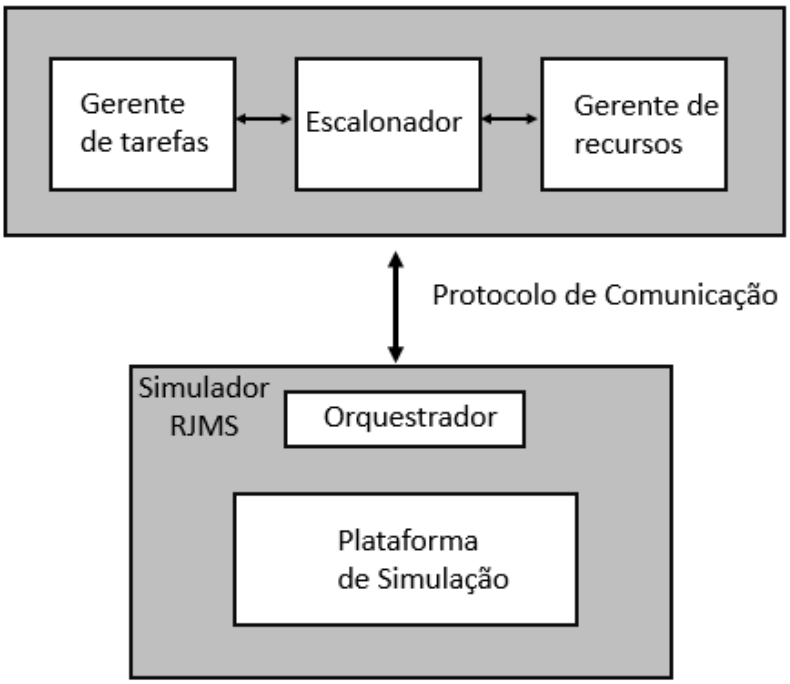


Figura 17 – Metodologia de Simulação RJMS utilizada para a realização das simulações apresentadas por (DUTOT et al., 2015). Fonte: Próprio Autor

- *id*: O identificador exclusivo do *job* (string).
- *subtime*: A hora em que a solicitação de *job* é emitida no sistema (float, em segundos).
- *res*: O número de recursos solicitados (inteiro positivo).
- *profile*: O nome do perfil associado ao *job* (string) - ou seja, a definição de como a execução do *job* deve ser simulada.
- *walltime*: O tempo máximo de execução do trabalho (float, em segundos). Qualquer trabalho que exceda seu tempo é morto pelo Batsim (campo opcional)

Os status dos *jobs* que o *Batsim* classifica no final de cada execução são os seguintes: *Jobs* executados com sucesso, ou seja sem nenhuma anormalidade até o final da execução, *Jobs* mortos e *Jobs* rejeitados. Onde o o *Job* é morto se atingir seu tempo de execução sem qualquer penalidade no escalonador, neste caso, simplesmente defini-se tempo de processamento igual ao tempo de execução. O *Job* rejeitado é quando ele excede o tempo de execução e ainda pode ocasionar danos ao escalonador. Os *Jobs* não são preemptivos, o que significa que, uma vez iniciado um trabalho não pode ser interrompido até a sua conclusão (DUTOT et al., 2017).

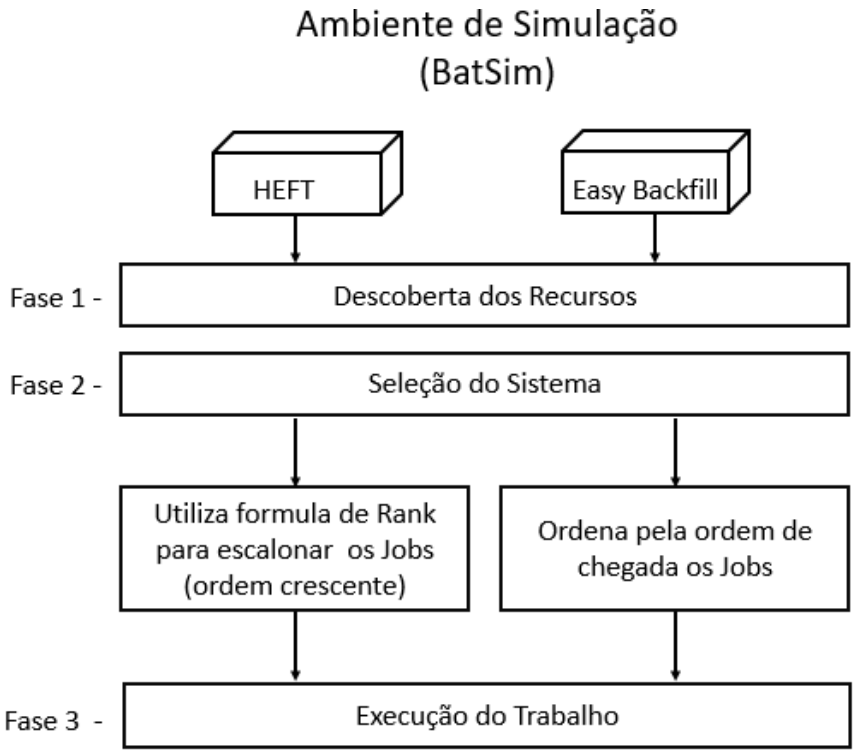


Figura 18 – Fases para o escalonamento de Jobs com *HEFT* e *Easy Backfilling*. Elaborado pelo autor

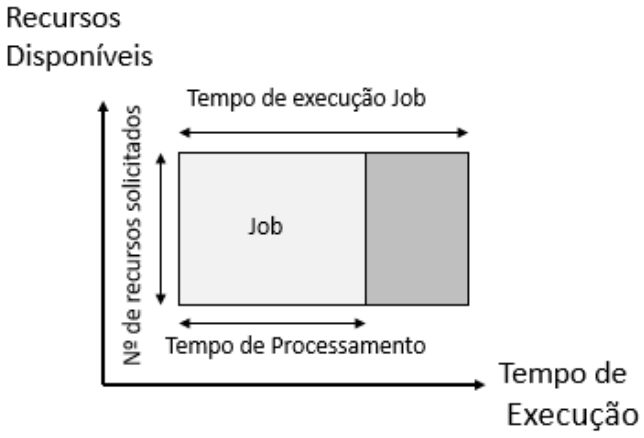


Figura 19 – Representação de aplicação e execução de um Job. Fonte Adaptada: (DUTOT et al., 2017)

A Figura 19 descreve as notações do *Job* principal dentro da ferramenta *Batsim*, delimitando graficamente o tempo de execução do Job, o tempo de processamento, tempo total de execução e o número de processadores solicitados (BATSIM, 2018a). Uma vez que os trabalhos foram executados, mais informações sobre eles podem ser definidos. Como resume a Figura 20. Foi denominado por $t_{inicial}$ a hora em que *Job* começa a ser executado, e t_{final} a hora em que ele conclui, portanto, (t_{final}

= $t_{inicial}$ + $t_{processameto}$). Já o tempo de espera do trabalho é (t_{exec} = $t_{inicial}$ - $t_{liberação}$). Para encontrar o T_{total} em que o *Job* permaneceu no sistema antes de ser executado, emprega-se o (T_{total} = t_{final} - $t_{liberação}$ = t_{exec} + $t_{processamento}$) denotando a quantidade total de tempo em que o *Job* ficou no sistema.

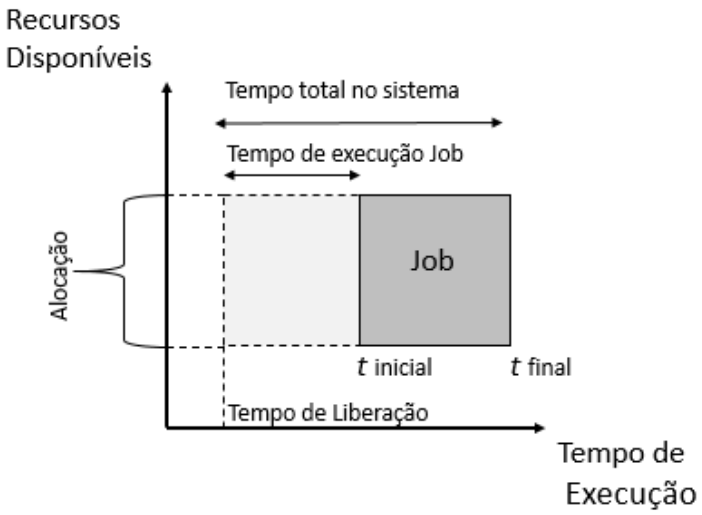


Figura 20 – Representação de algumas métricas de nível de job disponíveis após sua conclusão.Fonte Adaptada: (DUTOT et al., 2017)

4.5 CONSIDERAÇÕES PARCIAIS

Para buscar a solução do problema de consumo de energia elétrica para os ambientes de Grades Computacionais, foi utilizado a heurística *overlap* com a intensão de reduzir os espaços ociosos dos slots entre a execução de um *job* a outro. Esta heurística foi aplicada nos escalonadores *HEFT* e *Easy Backfilling*, com métodos de ordenação diferentes para verificar o consumo de ambos, com percentuais de sobreposição pré estabelecidos.

A aplicação da heurística *overlap* aconteceu no ambiente de simulação Batsim que é um simulador RJM, open sourse que permite simular uma infinidade de cenários. Sua entrada principal é a carga de trabalho estabelecida pelo usuário através de códigos, e o consumo de energia e médido através de *plugins* importados da ferramenta de simulação SimGrid.

5 SIMULAÇÃO E ANÁLISE DOS RESULTADOS

Neste capítulo serão apresentados os planos para a execução dos testes, ambiente de testes, descrição das experimentações e apresentação dos resultados obtidos com a aplicação da heurística implementada de *overlap* para os escalonadores *HEFT* e *Easy Backfilling*.

5.1 PLANO DE TESTES

O plano de teste, tem como principal objetivo especificar o que será realizado durante a experimentação demostrando sua consistência na apresentação dos resultados. Os testes foram feitos monitorando simulações, utilizando a ferramenta *Batsim*, onde a mesma já possui algumas cargas sintéticas, e também foram implementadas cargas específicas de testes para obtenção dos resultados, que baseiam-se em técnicas de avaliação de desempenho apresentadas por (JAIN, 1990).

Para analisar o comportamento da heurística *overlap* aplicados nos escalonadores *Easy Backfilling* e *HEFT*, foram elaboradas uma série de experimentos, cujo objetivo é realizar uma análise do consumo de energia, como pode ser observado na Figura 21.

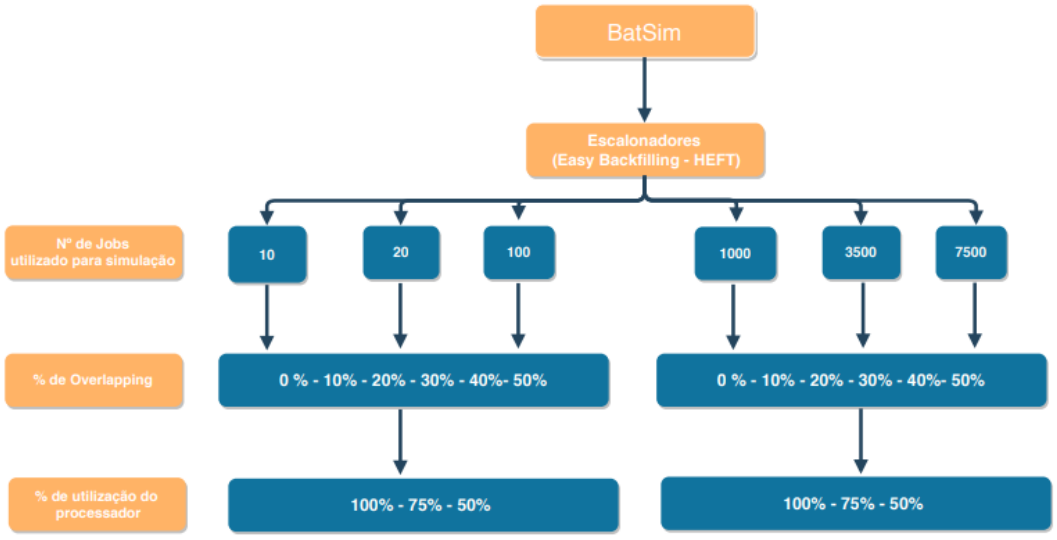


Figura 21 – Plano de testes realizados. Fonte: Próprio Autor

Para a simulação, optou-se por utilizar *workload* com um número aleatório de *jobs* que varia de 10 a 7.500, simulando cargas de um ambiente real. Foi utilizado o percentual de *overlap* entre 0% a 50%, pois em trabalhos similares como o de (RIOS

et al., 2011), que utilizou percentuais de até 100% de *overlap* seus resultados não foram satisfatórios, sugerindo sempre um *overlap* inferior a 50%.

Uma consideração relevante é que, a ferramenta para a simulação *Batsim* importa da ferramenta *SimGrid*[1] um *plugin* de tempo de utilização de núcleo da CPU, que considera carga de 100% em suas execuções, pois toda vez que a CPU inicia ou para de fazer alguma coisa ela recalcula novamente a carga. Portanto, se um núcleo estiver com carga de 50% ao longo de um período, isso significa que ele está com carga 100% na metade do tempo e com carga de 0% no restante do tempo. Para tanto, foram implementadas algumas modificações no código do *Batsim*[2], para realizar os experimentos com percentuais de 100%, 75% e 50% de utilização de núcleo da CPU, onde os *jobs* irão ocupar os recursos igualmente no tempo total de execução, porém a utilização de processamento será reduzida, para que a análise do consumo de energia seja verificada em todos os casos, apresentando resultados mais consistentes, já que nem sempre uma CPU está 100% executando com núcleos ativos.

Para a realização deste trabalho, foram executados 136 simulações, com os dois escalonadores implementados, analisando o consumo de energia (Joules), tempo de execução (segundos) e o status em que o *obs* está no final da execução, tendo como principal objetivo aplicar a heurística *overlap* com uma relevância no impacto de redução de consumo de energia elétrica em ambiente de Grade Computacional.

5.2 AMBIENTE DE TESTES

Qualquer estudo envolvendo simulação deve conter na plataforma a descrição do ambiente simulado. Como mencionado anteriormente, a ferramenta Batsim [3], utiliza *plugins* do *SimGrid* para a implementação e execução de vários módulos de suas simulações. As versões utilizada para as simulações com Batsim foi a 2.0.0 e do SimGrid 3.13.91 onde nela estão implementada as últimas alterações realizadas, inclusive a parte dos cálculos do consumo de energia elétrica.

Para o planejamento do ambiente de testes foi considerado a utilização de configurações de um ambiente real da Grid'5000, o *Nantes Econome*,[4], isso se deu para a validação dos resultados simulando um ambiente real, cujas configurações de hardwares são de 22 Hosts, Cpu 2 x Intel Xeon E5-2660 com 8 núcleos , 64 GiB de memória e armazenamento HDD de 2,0 TB e rede 10 Gbps. .

[1] <http://simgrid.gforge.inria.fr/simgrid/3.20/doc/>
[2] <https://hub.docker.com/r/tathi/batsim/>
[3] <https://http://www.simgrid.gforge.inria.fr/simgrid/3.20/doc/platform.html>
[4] <https://www.grid5000.fr/mediawiki/index.php/Nantes:Hardware>

5.3 ANÁLISE DOS RESULTADOS

Os experimentos e análises dos resultados são conduzidos individualmente para cada aplicação de *workload* com base no plano de testes apresentado na seção 5.1

5.3.1 Aplicação de *workload* de 10 *Jobs*

No primeiro cenário, as simulações com a carga de 100% de utilização de CPU apresentam-se na Figura 22. Essas informações foram monitoradas com *workload* de 10 obs. O maior impacto de redução de energia no escalonador *Easy Backfilling* foi com o *overlap* de 30%, obtendo uma redução no consumo de energia de 8% comparada com o escalonador sem aplicação de heurística. O menor desempenho no consumo de energia ocorreu com *overlap* de 40%, onde a redução foi de 1,82%.

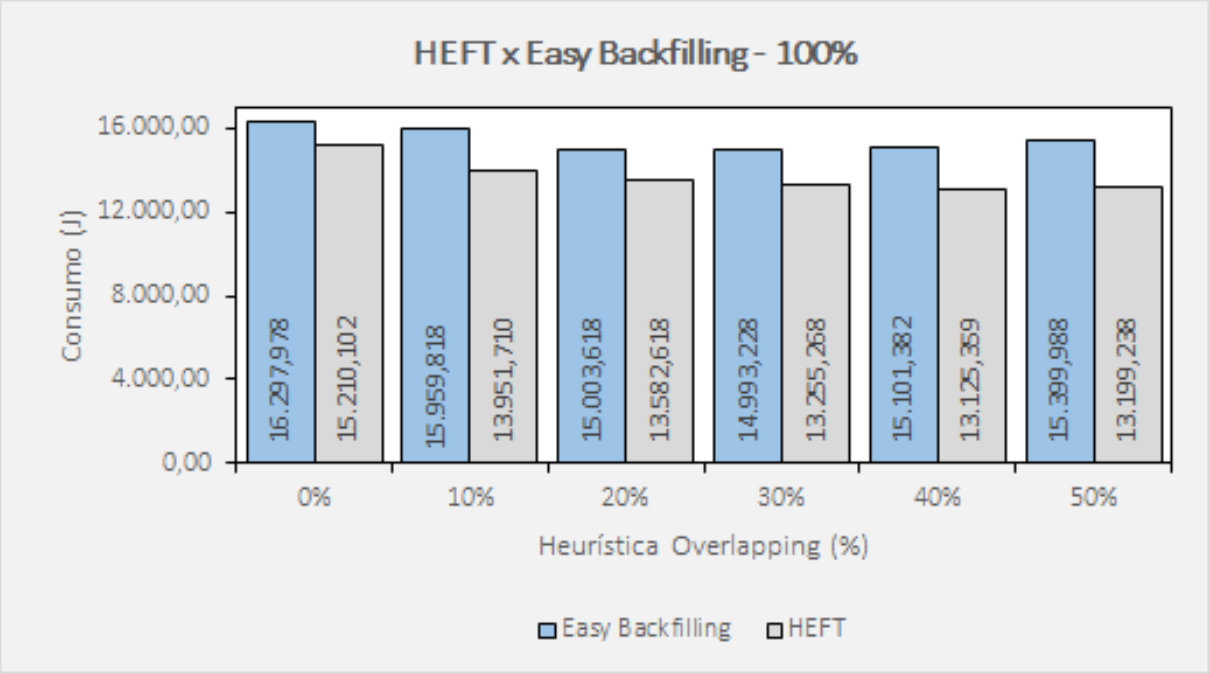


Figura 22 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 10 Jobs e 100% de CPU

No Escalonador *HEFT* o resultado com maior expressão foi com a utilização de *overlap* de 40%, gerando 13,05% de redução no consumo de energia. Enquanto sua menor performance não passou de 8,27% de redução com *overlap* de 10%. Desta maneira, se compararmos os dois escalonadores apresentados, neste cenário, pode-se observar que, mesmo com seu menor desempenho de consumo de energia, o *HEFT* com 13.951,710 Joules se sobrepôs sobre o *Easy Backfilling* com 15.959,818 Joules, isto se deve ao fato de que, em workloads com poucos *jobs*, o *HEFT* tem uma performance melhor de escalonamento.

A figura 23 apresenta os resultados das simulações com 75% de utilização de CPU, onde o maior impacto de redução da energia no escalonador *Easy Backfilling* aconteceu com o *overlap* de 30% que chega a um percentual de 11,51% de redução. O menor desempenho no consumo de energia foi com o *overlap* de 40%, onde a redução foi de somente 0,69%. Para os resultados obtidos nas simulações com o *HEFT*, pode-se destacar que, a maior redução de consumo de energia foi com *overlap* de 30% resultando em uma economia de 13,80%. Sua menor performance foi com *overlap* de 50% gerando 6,78% de redução de consumo. Sendo assim, observar que, o comportamento entre os escalonadores continuou o mesmo dos cenários anteriores, ou seja, o *HEFT* ainda continuou com uma margem maior de redução de energia de 1,99%, isto ocorreu considerando número de *workloads* gerados pequeno, pois em pesquisas que utilizaram ele como referência, foi relatado este tipo de comportamento em seus escalonamentos com cargas menores.

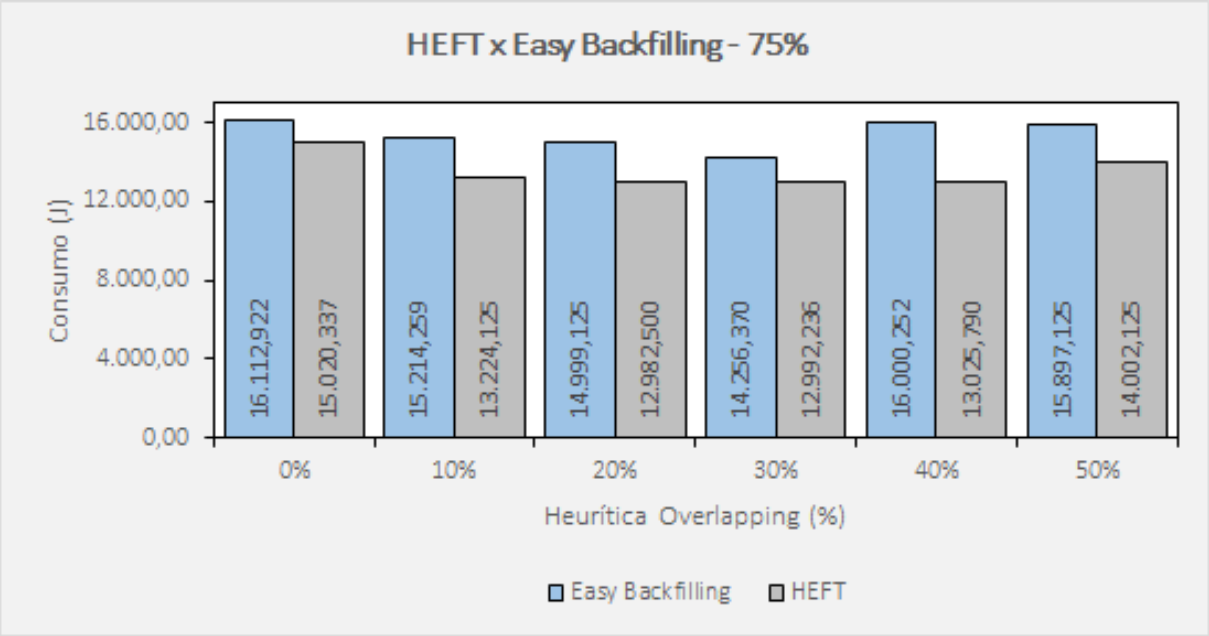


Figura 23 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 10 Jobs e 75% de CPU

Na Figura 24, apresenta-se a utilização de CPU de 50%, onde o maior impacto na redução de energia no escalonador *Easy Backfilling* aconteceu com *overlap* de 20% que chega a 11,55% de redução, e o menor desempenho no consumo de energia ocorreu com a *overlap* de 50%, onde a redução foi de somente 4,1%.

Para o escalonador *HEFT* o resultado com maior expressão aconteceu com a utilização de 20% de *overlap*, que chegou a 19,63% de redução, enquanto seu menor desempenho não passou de 4,37% de redução com *overlap* de 50%. Ao comparar os dois escalonadores apresentados, nestes cenários, pode-se observar que, mesmo

em seu pior desempenho do consumo de energia, o *HEFT* se sobrepôs sobre o *Easy Backfilling* consideravelmente, com um percentual de 8,08%.

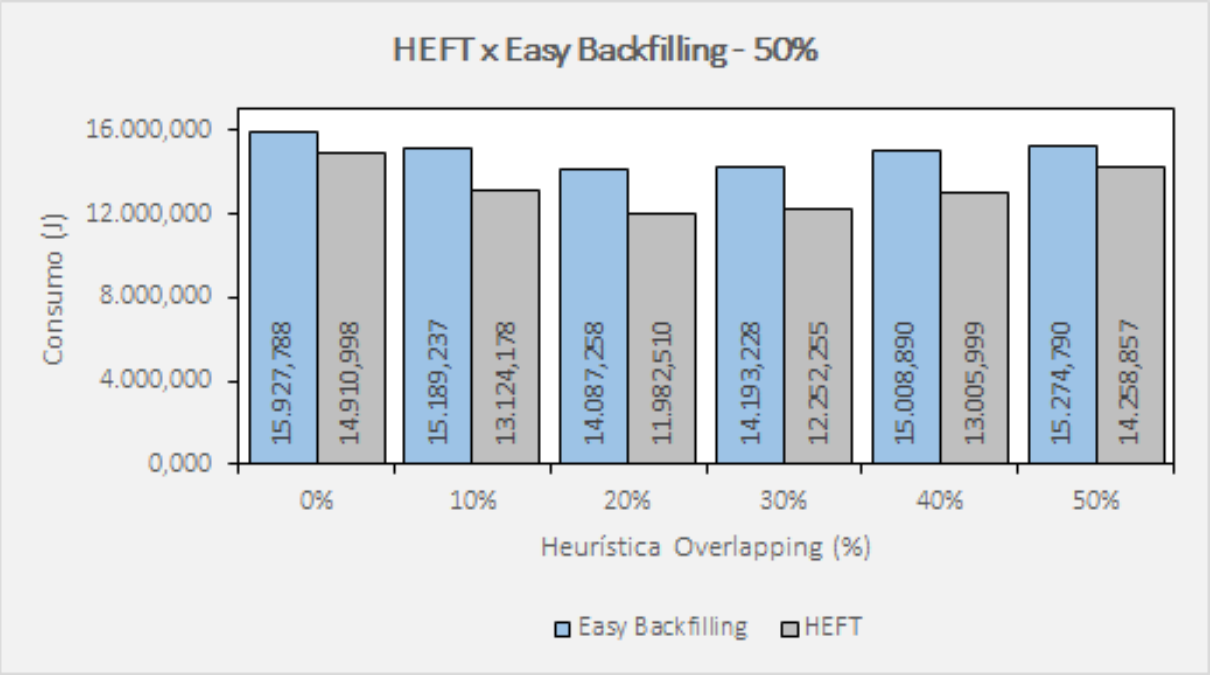


Figura 24 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 10 Jobs e 50% de CPU

A Tabela 4, tem como objetivo apresentar um resumo dos resultados encontrados com a aplicação da Heurística *Overlap* imprimindo o percentual de redução. Como pode-se observar, com *workload* de 10 *jobs*, o maior percentual de redução aconteceu utilizando o escalonador *HEFT*, com 20% de *overlap* e utilização de processador de 50%, esse percentual é calculado levando em consideração o consumo do escalonador sem a aplicação da heurística. Neste cenário o escalonador *HEFT* obteve os percentuais maiores de redução, que ocorreu devido a sua maneira de escalonamento, principalmente com *workloads* menores.

Tabela 4 – Apresentação dos maiores e menores resultados obtidos pela simulação com *workload* de 10 Jobs para o consumo de energia elétrica

	Maior Redução		**Menor Redução**	
	Overlap	**(%) Resultado**	**Overlap**	**(%) Resultado**
Easy Backfilling- 100%	30%	8,00%	40%	1,82%
Easy Backfilling- 75%	30%	11,51%	40%	0,69%
Easy Backfilling- 50%	20%	11,55%	50%	4,10%
HEFT - 100%	40%	13,05%	10%	8,27%
HEFT - 75%	30%	13,50%	50%	6,78%
HEFT - 50%	20%	19,63%	50%	4,37%

O tempo de execução apresentado na Figura 25 considera o tempo em que os *jobs* ocupam no processador. Desta forma, pode-se verificar que o *Easy Backfilling*

com *overlap* de 30% obteve um tempo de 35,551 segundos atingindo uma redução do tempo de execução de 3,92%, enquanto o *HEFT* com *overlap* de 40% executou os *jobs* em 34,452 segundos gerando uma redução no tempo de execução de 4,38%.

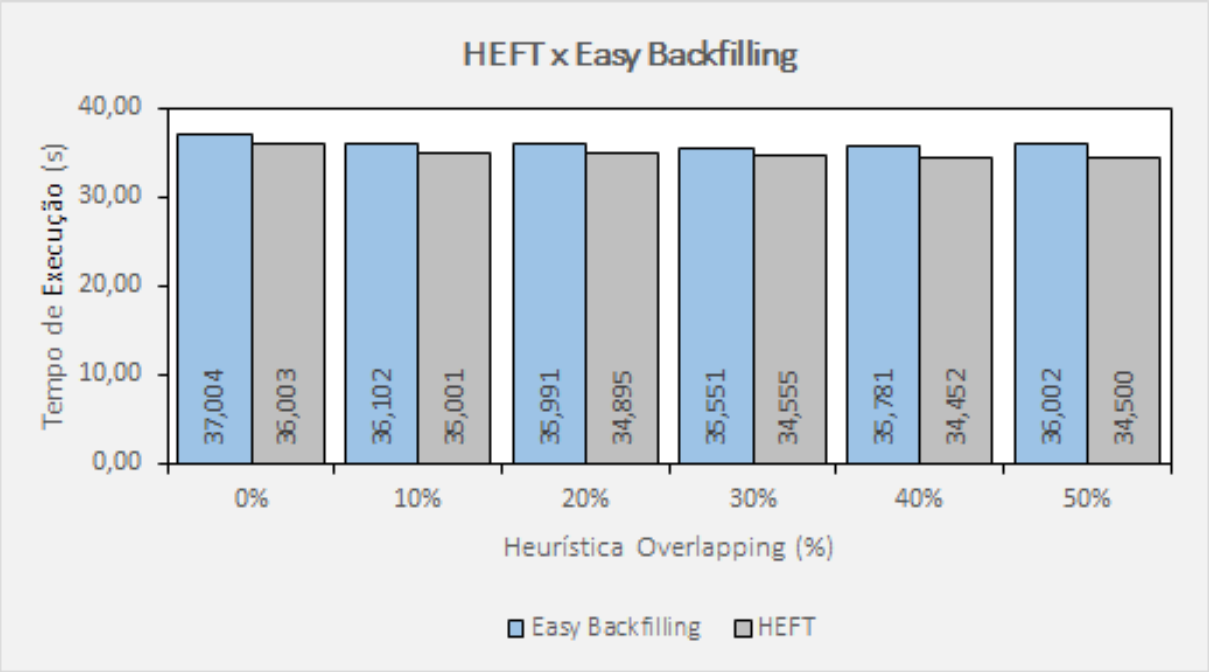
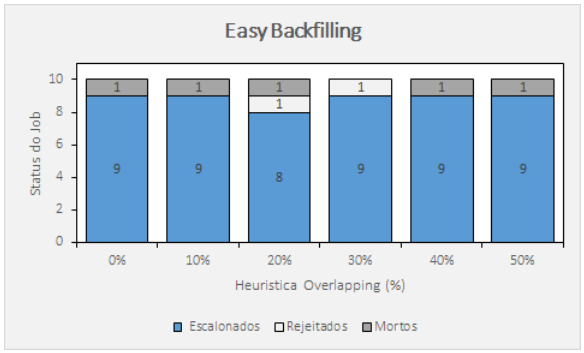


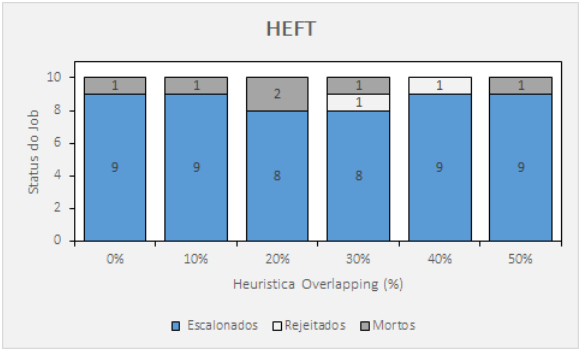
Figura 25 – Tempo de Execução dos Escalonadores com utilização da Heurística Overlap

A ferramenta Batsim também apresenta em suas simulações o status de cada *Job* escalonado, que pode ser, executados com sucesso, rejeitado pelo escalonamento e mortos, quando excede o tempo de execução, como já explicados na seção 4.3.2.

Figura 26 – Comparativo dos Status dos 10 Jobs escalonados pelo Batsim com os *Easy Backfilling* e *HEFT*



(a) Status do escalonamento executado pelo Easy Backfilling



(b) Status do escalonamento executado pelo *HEFT*

A Figura 26 mostra os quantidade detalhada dos *jobs* com seus status. Pode-

se perceber que o escalonador *Easy Bakfilling*, representado na Figura 26a obteve em média 10% dos *jobs* escalonados mortos ou rejeitados, o mesmo se aplica a Figura 26b. Com os resultados apresentados na Figura 25 e 26, é possível calcular a taxa de execução em *Jobs*/segundos corroborando que, para o *Easy Backfilling* com 30% de *overlap* obteve a melhor taxa de execução de 0,2531 Jobs/segundo e o *HEFT* com 40% de *overlap* gerou uma taxa de execução de 0,2612 Jobs/segundo.

5.3.2 Aplicação de *workload* de 20 *Jobs*

Para a execução do segundo cenário, foi aplicado um *workload* de 20 *jobs* para os experimentos, onde observou-se que o comportamento da heurística *overlap* com utilização de 100% de CPU, continuou semelhante ao cenário anterior, como mostra a Figura 27. Onde, percentual de ganho do energia do escalonador *Easy Backfilling* deu-se em 20% de *overlap* com 7,42% redução e o *HEFT* em 30% de *overlap* com 7,73% de redução. Considerando neste caso também que, os percentuais com menor impacto de redução de consumo de energia foi respectivamente o *Easy Backfilling* com 50% de *overlap* e 2,05% de redução e o *HEFT* com 50% de *overlap* e 6,33% de redução de energia

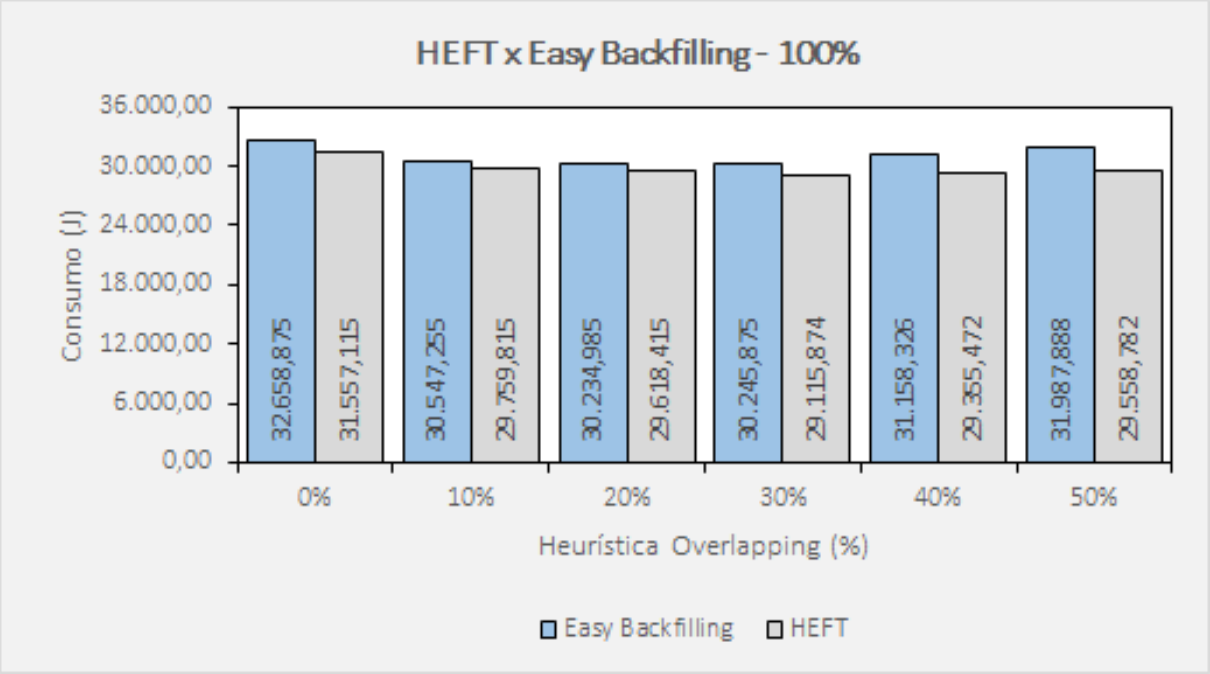


Figura 27 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 20 Jobs e 100% de CPU

Com a alteração na utilização de CPU em 75%, pode-se observar uma redução no consumo de energia, onde o percentual mais significativo do *Easy Backfilling* foi de 5,03% com *overlap* de 10% , e o*HEFT* de 8,44% com *overlap* de 10%. Em contrapartida, as menores reduções foram de 1,83% de redução com *overlap* de 40%

para *Easy Backfilling* e 5,04% de redução com *overlap* de 50% para o *HEFT*, como mostra a Figura 28

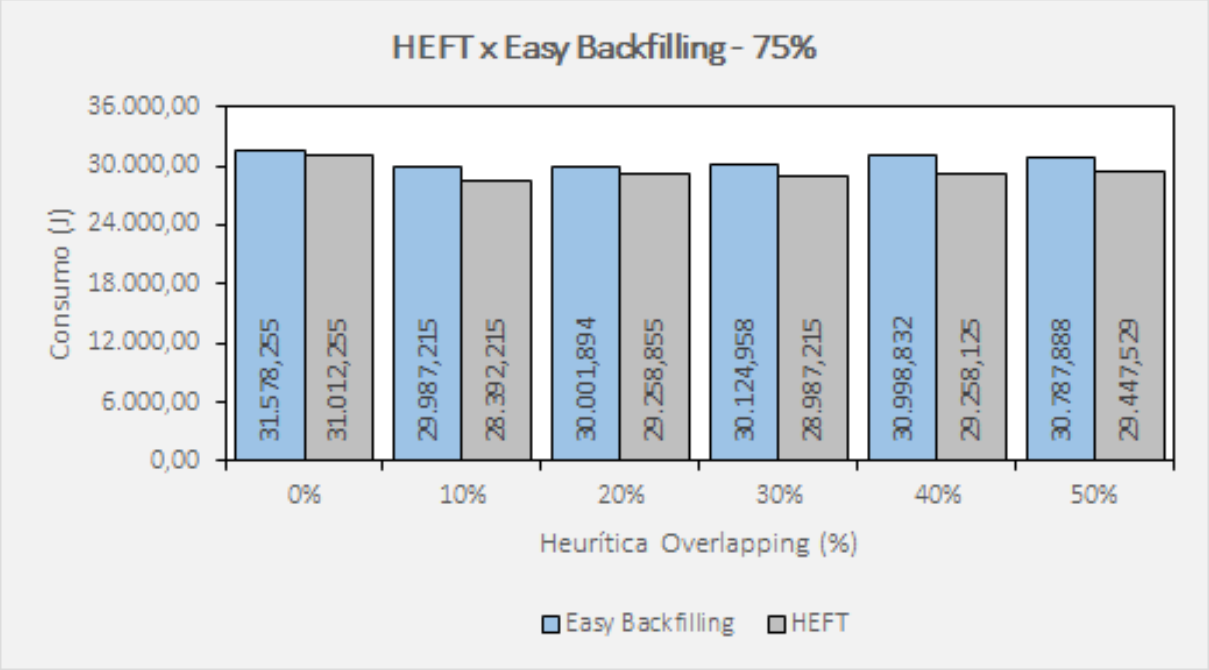


Figura 28 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 20 Jobs e 75% de CPU

Na Figura 29 encontra-se os resultados para 50% de utilização de CPU.

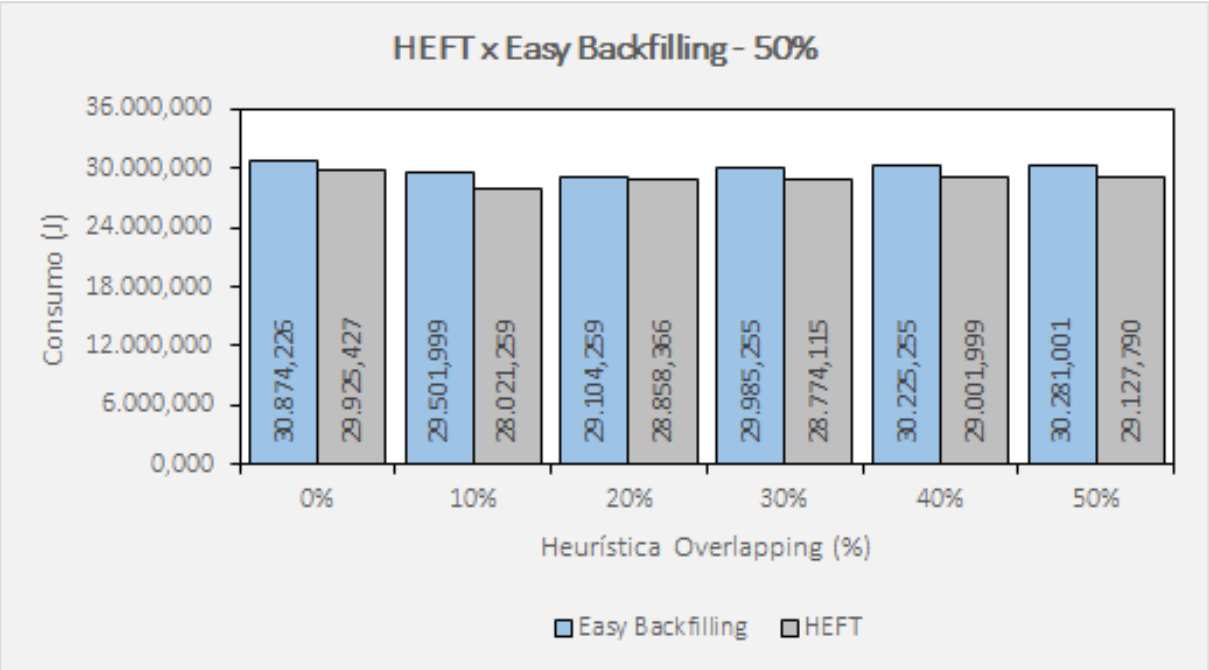


Figura 29 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 20 Jobs e 50% de CPU

Onde os melhores percentuais de redução encontrados foram no escalonador *Easy Backfilling* com *overlap* de 10% e 5,73 de redução e o *HEFT*, com *overlap* também de 10% e 6,3% de redução. Enquanto os menores percentuais obtidos em ambos os escalonadores, foram de *overlap* de 50% com 1,92% de redução no *Easy Backfilling* e o *HEFT*, com *overlap* de 50%, com redução de 2,66%. Assim como no cenário anterior, o *Heft* obteve melhores resultados na redução de consumo de energia, isto é devido seu método de escalonamento ser mais eficiente em cenários com a utilização de poucos *jobs*.

Tabela 5 – Apresentação dos maiores e menores resultados obtidos pela simulação com *workload* de 20 Jobs para o consumo de energia elétrica

	Maior Redução		**Menor Redução**	
	Overlap	**(%) Resultado**	**Overlap**	**(%) Resultado**
Easy Backfilling- 100%	20%	7,42%	50%	2,05%
Easy Backfilling- 75%	10%	5,03%	40%	1,83%
Easy Backfilling- 50%	10%	5,73%	50%	1,92%
HEFT - 100%	30%	7,73%	50%	6,33%
HEFT - 75%	10%	8,44%	50%	5,04%
HEFT - 50%	10%	6,30%	50%	2,66%

A tabela 5, demostrou que a utilização da heurística *overlap*, traz um ganho significativo para o consumo de energia elétrica, mesmo que, neste cenário o *workload* utilizado seja somente com 20 *jobs*. Sendo assim, o maior percentual de redução aconteceu utilizando o escalonador *HEFT*, com 10% de *overlap* e utilização de processador de 75% com 8,44% de redução.

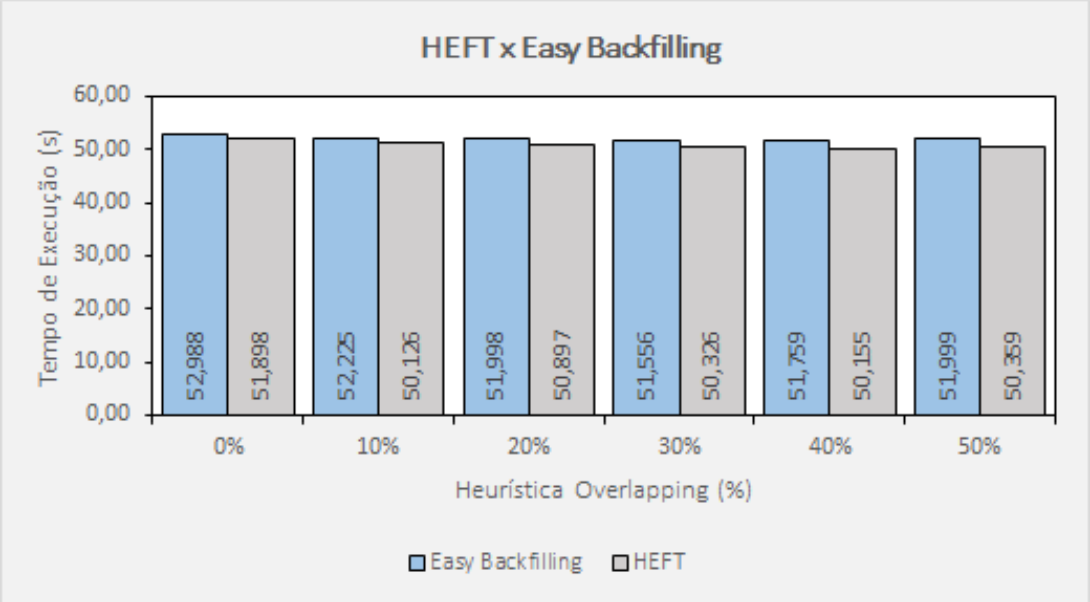
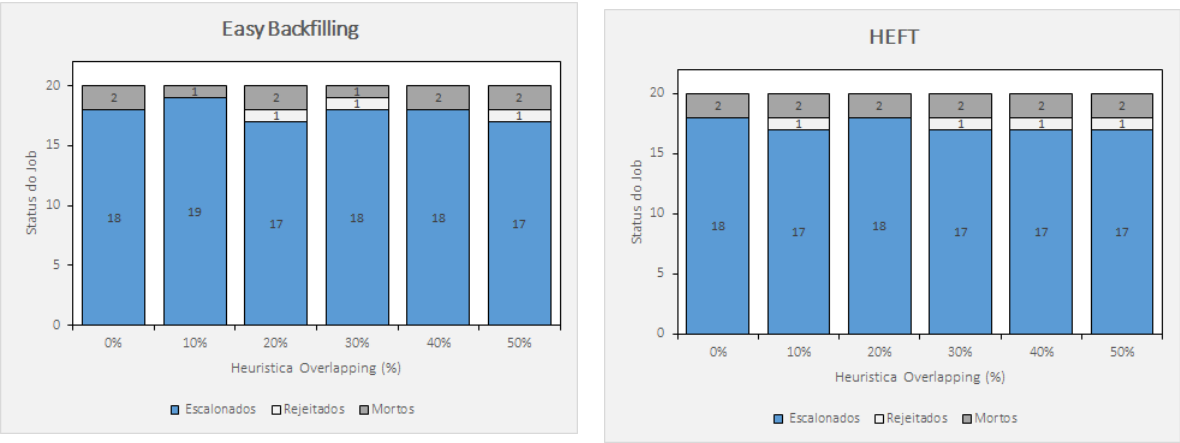


Figura 30 – Tempo de Execução dos Escalonadores com utilização da Heurística Overlap

Para o tempo de execução, pode-se observar que o *Easy Backfilling* obteve o tempo considerável, com 52,225 segundos e com *overlap* de 20% atingindo um percentual de redução de 1,45%, enquanto o *HEFT* conseguiu o tempo de 50,126 segundos com *overlap* de 10% chegando a percentual de redução do tempo de execução de aproximadamente 3,41% como pode ser observado na Figura 30

Figura 31 – Comparativo dos Status dos 10 Jobs escalonados pelo Batsim com os *Easy Backfilling* e *HEFT*



(a) Status do escalonamento executado pelo Easy Backfilling

(b) Status do escalonamento executado pelo *HEFT*

A Figura 31 apresenta status de cada *job* escalonado. Onde o percentual de *Jobs* mortos ou rejeitados fica em média de 10%, em ambos os escalonadores, tanto o apresentado na Figura 31a, quanto a Figura 31b. Com os resultados encontrados nas Figura 30 e 31, foram calculados as taxas de execuções em *Jobs*/ segundos, onde verificou-se que, ambos os escalonadores obtiveram a melhor taxa com 10% de *overlap*. O *Easy Backfilling* com 0,3636 Jobs/segundo e o *HEFT* gerou uma taxa de execução de 0,3391 Jobs/ segundo.

5.3.3 Aplicação de *workload* de 100*Jobs*

Quando aplica-se o *workload* de 100 *jobs* a uma utilização de processador de 100%, nota-se através da Figura 32, que o impacto de maior relevância para a redução de energia no escalonador *Easy Backfilling* acontece com *overlap* de 10% que chega a um percentual de 3,47% de redução. O menor o consumo de energia ocorre com *overlap* de 50%, onde a redução foi de 0,57%. Com os resultados encontrados utilizando o escalonador *HEFT* foram, melhor percentual de redução de energia aconteceu com *overlap* de 10% chagando a 2,20% de redução e o menor resultado foi de 50% de *overlap* com 1,71% de redução.

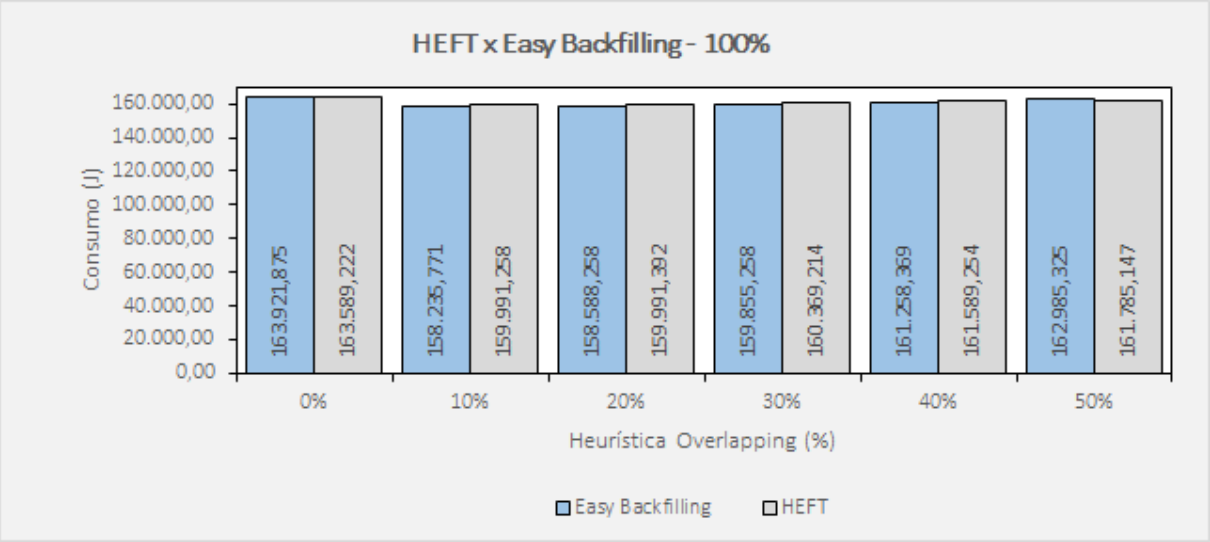


Figura 32 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 100 Jobs e 100% de CPU

Na redução de utilização de CPU de 75%, pode-se observar na Figura 33, que o percentual reduzido de energia neste cenário com o maior impacto ocorre no escalonador *Easy Backfilling* com *overlap* de 10% que chega a um percentual de 1,25% de redução. O menor desempenho no consumo de energia ocorreu com *overlap* de 50%, onde redução foi de -1,09%.

Para os resultados obtidos nas simulações com o *HEFT*, pode-se observar que, o maior impacto na redução de consumo de energia com *overlap* de 10% foi de 1,17% de redução, enquanto o menor desempenho aconteceu com *overlap* de 50% e consumo de -0,19% de redução.

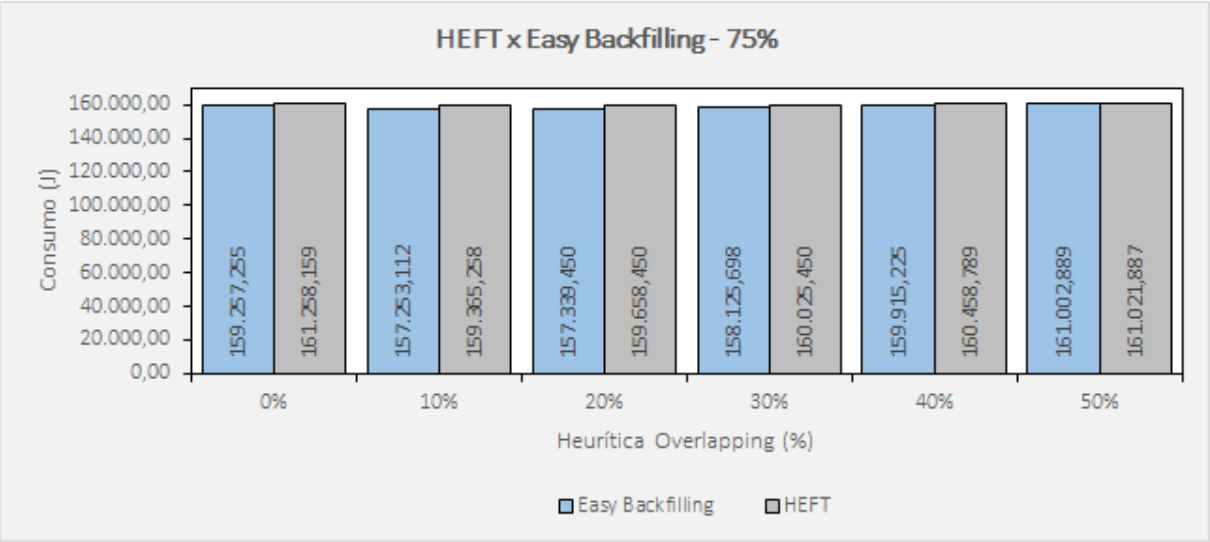


Figura 33 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 100 Jobs e 75% de CPU

Para a utilização de CPU de 50%, pode-se observar na Figura 34, que o maior impacto na redução de energia no escalonador *Easy Backfilling* ocorreu com *overlap* de 10%, chegando a 1,82% de redução, e o menor desempenho no consumo de energia ocorreu com a *overlap* de 40%, onde a redução foi de 0,85%. No escalonador *HEFT* o resultado com maior expressão aconteceu com a utilização de *overlap* de 10% que chegou a 1,35% enquanto seu menor desempenho não passou de -0,19% de redução com *overlap* de 50%.

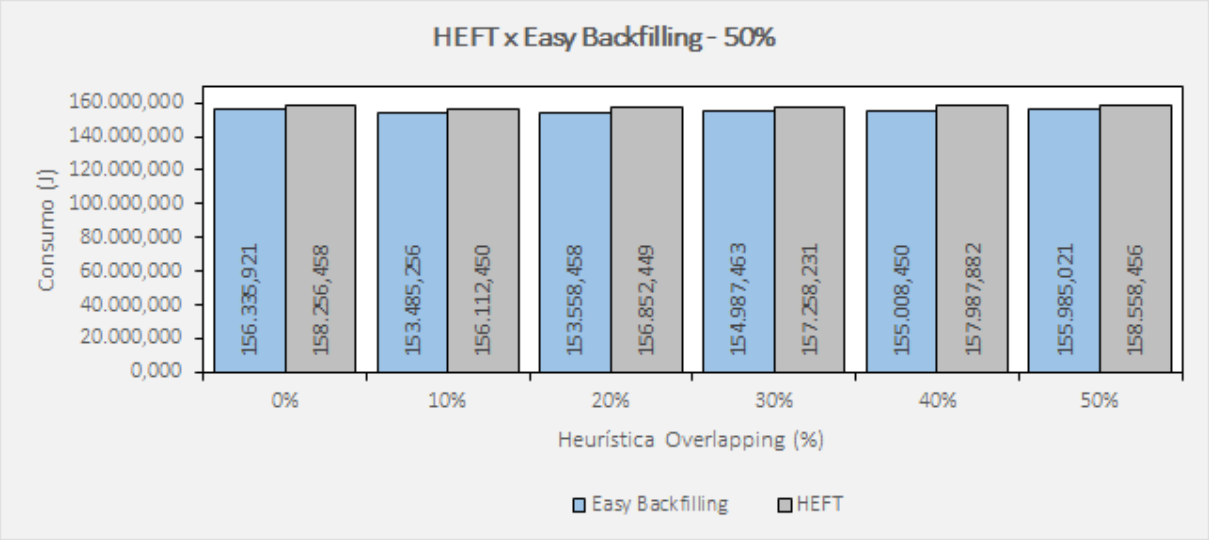


Figura 34 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 100 Jobs e 50% de CPU

Na Tabela 6, apresenta um resumo dos resultados obtidos com a aplicação da Heurística *Overlap* com os percentuais de redução. Como pode-se observar, com *workload* de 100 *jobs*, o maior percentual aconteceu utilizando o escalonador *Easy Backfilling*, com 10% de *overlap* e utilização de processador de 50%, considerando que o consumo de energia com essa carga é de 153.485,256 Joules, enquanto que, com 100% esse consumo passa a ser 163.921,875 Joules com o mesmo percentual utilizado.

Tabela 6 – Apresentação dos maiores e menores resultados obtidos pela simulação com *workload* de 1000 Jobs para o consumo de energia elétrica

	Maior Redução		**Menor Redução**	
	Overlap	**(%) Resultado**	**Overlap**	**(%) Resultado**
Easy Backfilling- 100%	10%	3,47%	50%	0,57%
Easy Backfilling- 75%	10%	1,25%	50%	-1,09%
Easy Backfilling- 50%	10%	1,82%	40%	0,85%
HEFT - 100%	10%	2,20%	50%	1,71%
HEFT - 75%	10%	1,17%	50%	0,14%
HEFT - 50%	10%	1,35%	50%	-0,19%

Quanto aos tempos de execução utilizado apresentados na Figura 35, foi considerado o tempo em que os *jobs* ocupam o processador, sendo assim, pode-se verificar que o *Easy Backfilling* obteve seu tempo melhor para a execução de aproximadamente 258,125 segundos com *overlap* de 10%, enquanto o *HEFT* conseguiu o tempo de 261,001 segundos com *overlap* de 20%.

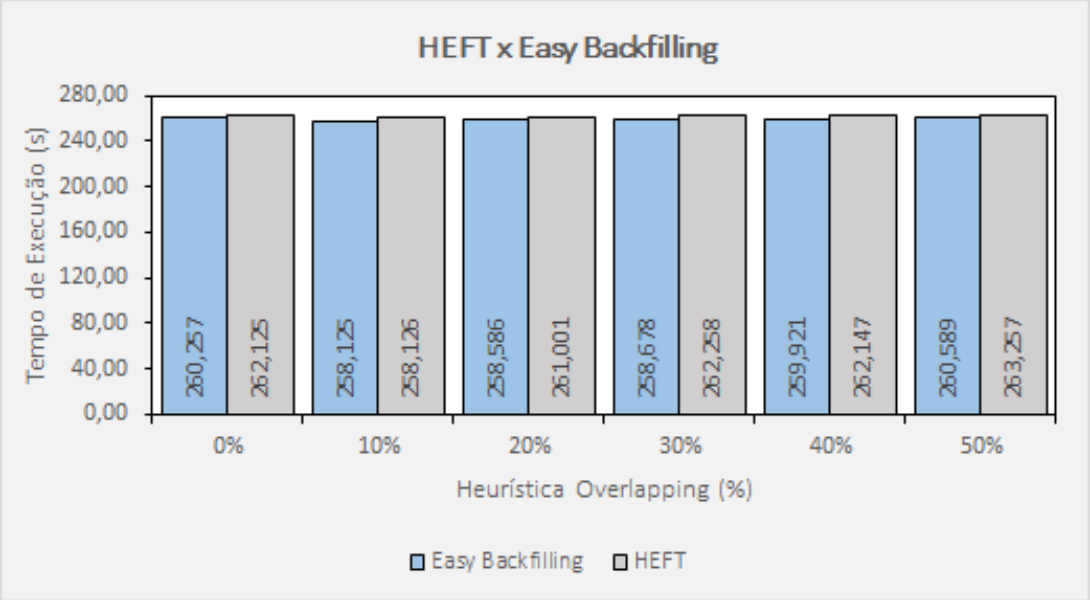
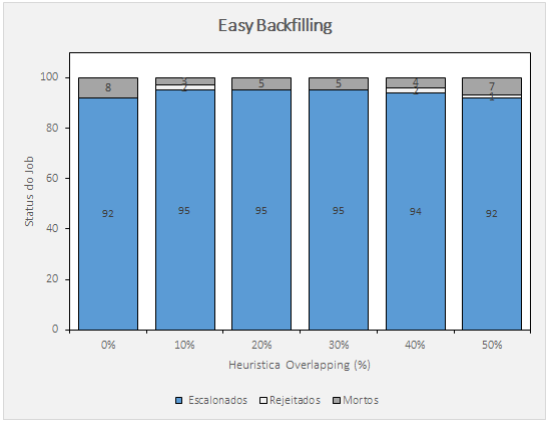


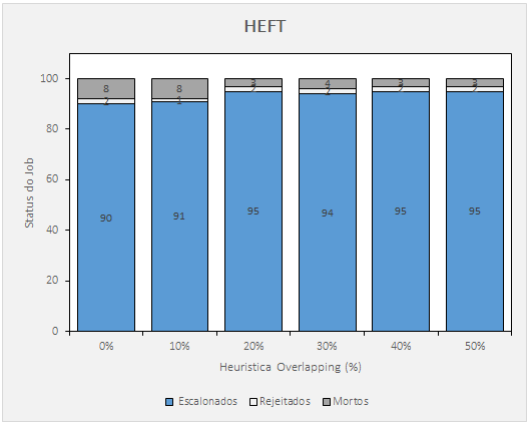
Figura 35 – Tempo de Execução dos Escalonadores com utilização da Heurística Overlap

Os status de cada *jobs* escalonado estão apresentados na Figura 36 e mostram as quantidade detalhada dos *Job* com seus respectivos status.

Figura 36 – Comparativo dos Status dos 100 Jobs escalonados pelo Batsim com os Easy Backfilling e HEFT



(a) Status do escalonamento executado pelo Easy Backfilling



(b) Status do escalonamento executado pelo *HEFT*

Pode-se perceber que o escalonador *Easy Bakfilling*, representado na Figura

36a obteve em média 5% dos *jobs* escalonados mortos ou rejeitados, o mesmo percentual se aplica a Figura 36b, porém observa-se que no escalonador *HEFT*, os números de *Jobs* foram mais expressivos, isso se da pela política de escalonamento que o escalonador realiza antes de direcionar para os recursos disponíveis.

Com os resultados encontrados nas Figura 35 e 36, foram calculados as taxas de execuções em Jobs/ segundos, onde verificou-se que, o *Easy Backfilling* obteve uma taxa de execução de 0,3682 Jobs/ segundo com o *overlap* de 10%, enquanto a taxa de execução do *HEFT* foi de 0,3639 Jobs/ segundo com *overlap* de 20%.

5.3.4 Aplicação de *workloads* de 1000 *Jobs*

Para a execução do quarto cenário, foi aplicado um*workload* de 1000 *jobs* para os experimentos, onde observou-se que, o comportamento da heurística *overlap* com utilização de 100% de CPU mudou. A Figura 37, demostra bem essa mudança, onde observa-se que o maior percentual de ganho do energia do escalonador *Easy Backfilling* deu-se em 10% de *overllaping* com 10,23% de redução e o *HEFT* com mesmo percentual de *overllaping* com 9,83% de redução. Considerando neste caso também que os percentuais com menores reduções de consumo de energia foram respectivamente o *Easy Backfilling* com 50% de *overlap* e 4,8% de redução e o *HEFT* com 50% de *overlap* e 4,52% de redução.

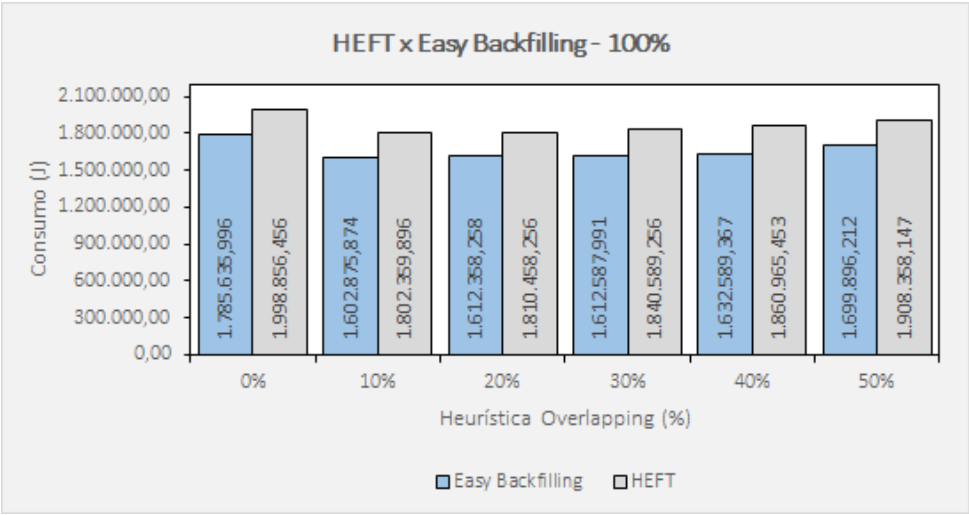


Figura 37 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 1000 Jobs e 100% de CPU

Na Figura 38 observa-se a mudança de utilização de CPU em 75% e mantei o mesmo tempo de utilização dos recursos e nota-se a redução do consumo de energia como pode ser observado. Neste cenário, o percentual mais significativo do *Easy Backfilling* foi de 10,08% com *overlap* de 10% , e o*HEFT* de 6,56% também com *overlap* de 10%. Em contrapartida, as piores reduções foram de 1,57% com *overlap* de

50% para *Easy Backfilling* e 1,06% com *overlap* de 50% para o *HEFT*, como mostra a Figura 38.

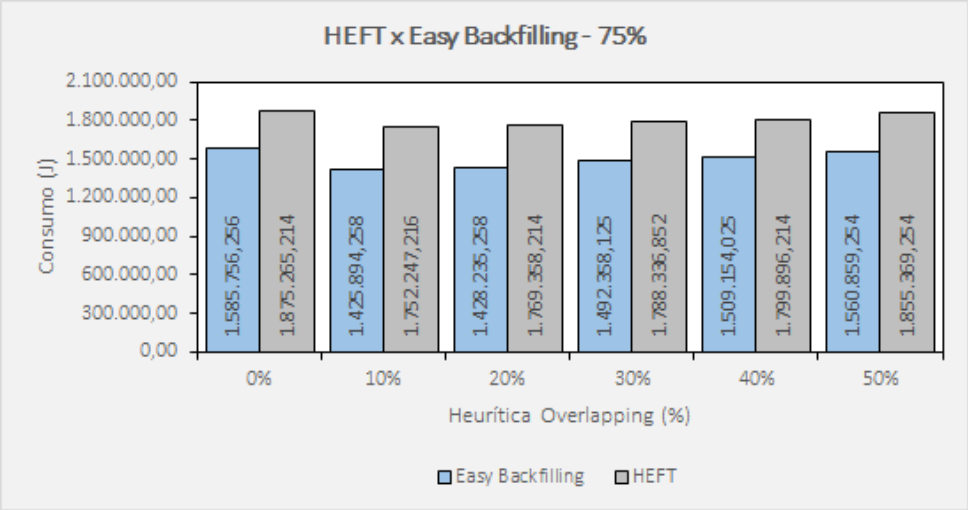


Figura 38 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 1000 Jobs e 75% de CPU

Na alteração de utilização de CPU de 50%, os melhores percentuais foram do escalonador *Easy backfilling* com *overlap* de 10% em 19,16% de redução e o *HEFT*,com *overlap* de 10% com 5,6% de redução, já os priores percentuais de *overlap* foram no *Easy Backfilling* com 50% de *overlap* e redução de 5,12% e o *HEFT*, com *overlap* de 50% com redução de 0,72%, como pode ser observado na Figura

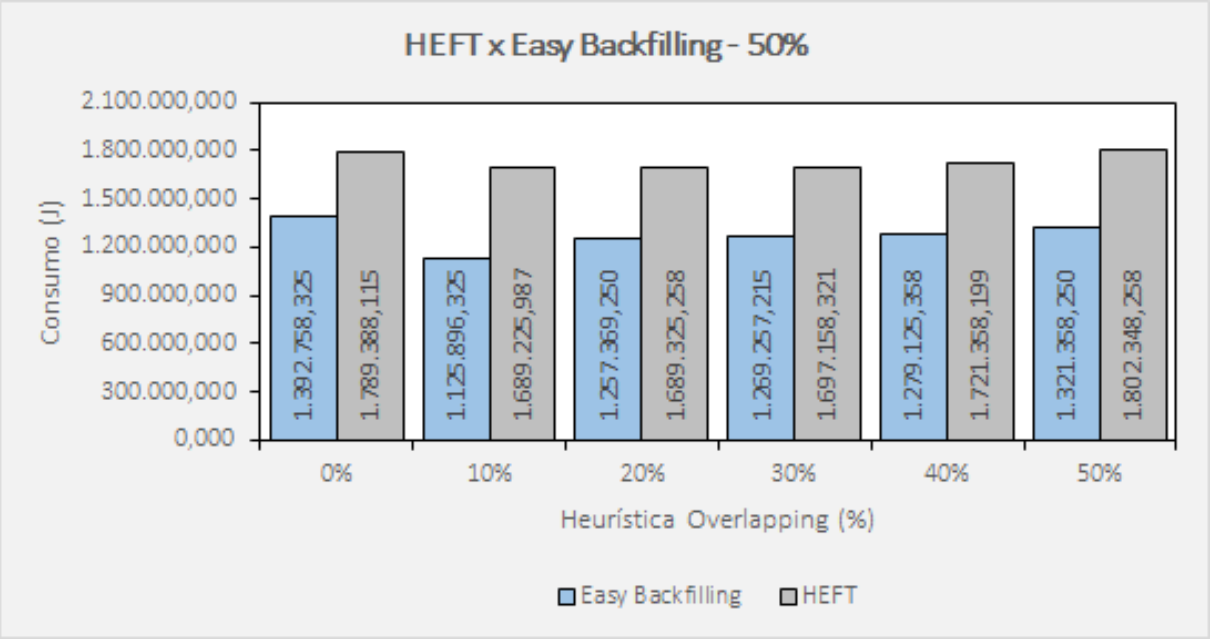


Figura 39 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 1000 Jobs e 50% de CPU

A tabela 7, apresenta os resultados obtidos com a aplicação da Heurística *Overlap* nos dois escalonadores utilizados. Como pode-se observar, com *workload* de 1000 *jobs*, o maior percentual de redução aconteceu utilizando o escalonador *Easy Backfilling*, com 10% de *overlap* e utilização de processador de 50%, considerando que o consumo de energia com essa carga é de 1.125.896,325 Joules, enquanto que, com 100% de utilização de CPU esse consumo passa a ser 1.602.875,874 Joules com o mesmo percentual de *overlap* aplicado.

Tabela 7 – Apresentação dos maiores e menores resultados obtidos pela simulação com *workload* de 1000 Jobs para o consumo de energia elétrica

	Maior redução		**Menor Redução**	
	Overlap	**(%) Resultado**	**Overlap**	**(%) Resultado**
Easy Backfilling- 100%	10%	10,23%	50%	4,80%
Easy Backfilling- 75%	10%	10,08%	50%	1,57%
Easy Backfilling- 50%	10%	19,16%	50%	5,12%
HEFT - 100%	10%	9,83%	50%	4,52%
HEFT - 75%	10%	6,56%	50%	1,06%
HEFT - 50%	10%	5,60%	50%	0,72%

Com tempo de execução demostrado na Figura 40, pode-se observar que o *Easy Backfilling* obteve o tempo de execução melhor em 1.462,895 segundos com *overlap* de 10% atingindo um percentual de redução de 13,23%, enquanto o *HEFT* conseguiu o tempo de 2.074,256 segundos com *overlap* de 10% chegando a percentual de redução do tempo de execução de aproximadamente 3,31%. Neste cenário o *Easy Backfilling* sobressaiu sobre o *HEFT* com uma vantagem em consumo de energia e tempo de execução.

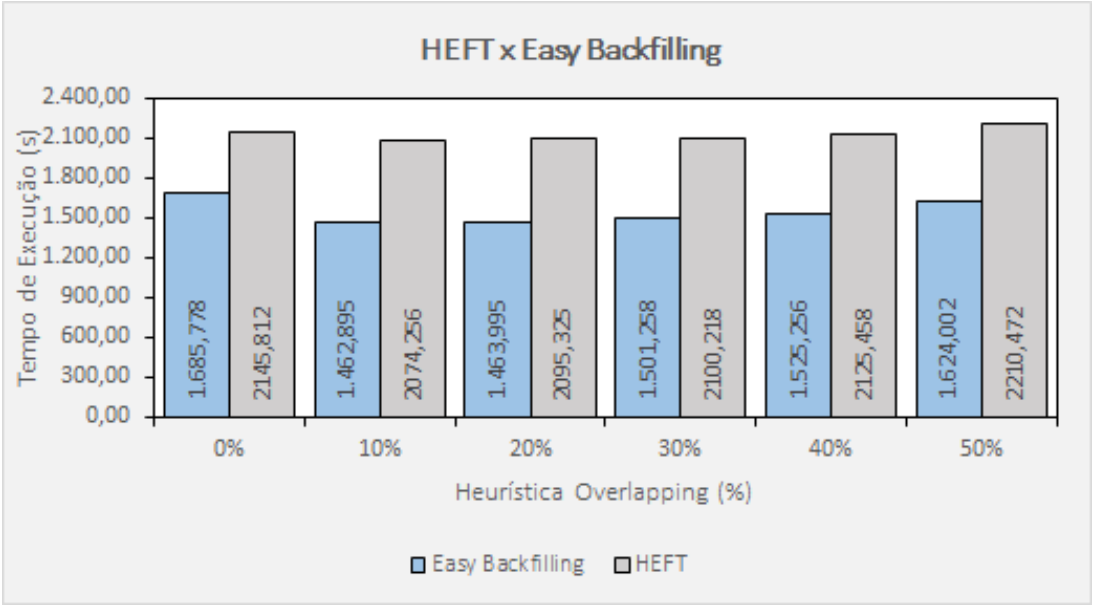
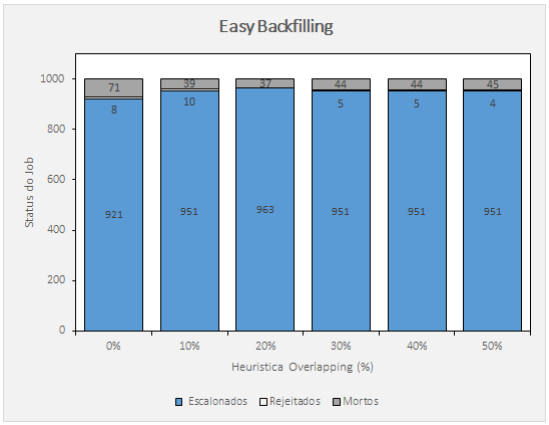


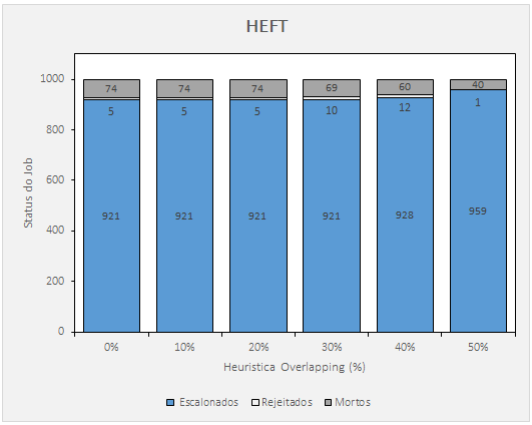
Figura 40 – Tempo de Execução dos Escalonadores com utilização da Heurística Overlap

A Figura 41 apresenta o status dos *jobs* escalonados, onde pode-se perceber que o percentual médio de *jobs* mortos ou rejeitados fica em média de 10%, apresentados na Figura 31a e Figura 31b.

Figura 41 – Comparativo dos Status dos 1000 Jobs escalonados pelo Batsim com os *Easy Backfilling* e *HEFT*



(a) Status do escalonamento executado pelo Easy Backfilling



(b) Status do escalonamento executado pelo *HEFT*

Desta forma, com os resultados encontrados nas Figura 40 e 41, foram calculados as taxas de execuções em Jobs/ segundos, onde verificou-se que, o *Easy Backfilling* obteve uma taxa de execução de 0,6582 Jobs/ segundo com o *overlap* de 20%, enquanto a taxa de execução do *HEFT* foi de 0,4440 Jobs/ segundo com *overlap* de 10%.

5.3.5 Aplicação de *workloads* de 3500 *Jobs*

Para a execução do quinto cenário, foi implementado um *workload* de 3500 *jobs* para os experimentos, onde observa-se que, o comportamento da heurística *Overlap* com utilização de 100% de CPU, obteve um percentual de ganho de energia com o escalonador *Easy Backfilling* em 10% de *overlap* com 7,01% de redução e o *HEFT* em 10% de *overlap* com 6,82% de redução.

Neste cenário, os percentuais com menores reduções de consumo de energia foram no escalonador *Easy Backfilling* com 50% de *overlap* e 0,51% de redução e o *HEFT* com 50% de *overlap* e -2,47% negativos a redução de energia, como mostra a Figura 42.

Alterando a utilização de CPU para 75% e mantendo o mesmo tempo de execução dos recursos, pode-se observar uma redução no consumo de energia. Onde, o percentual mais significativo do *Easy Backfilling* foi de 9,62% com *overlap* de 10% , e o*HEFT* de 3,64% com *overlap* de 10%. Em contrapartida, os piores percentuais

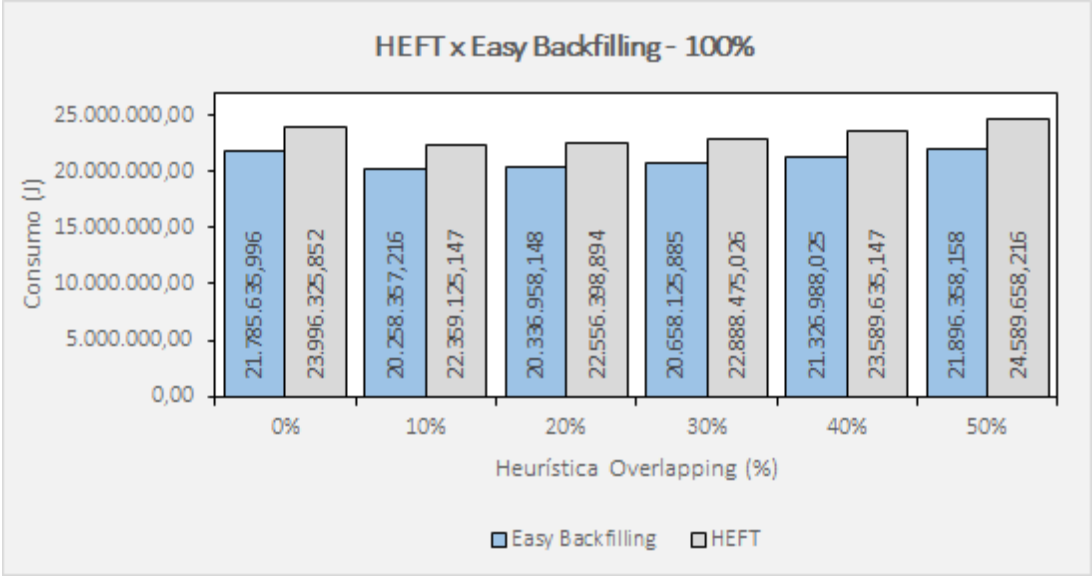


Figura 42 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 3500 Jobs e 100% de CPU

encontrados foram de 3,94% de redução com *overlap* de 50% para *Easy Backfilling*, e com redução de -2,75% com *overlap* de 50% para o *HEFT*, como demostra a Figura 43

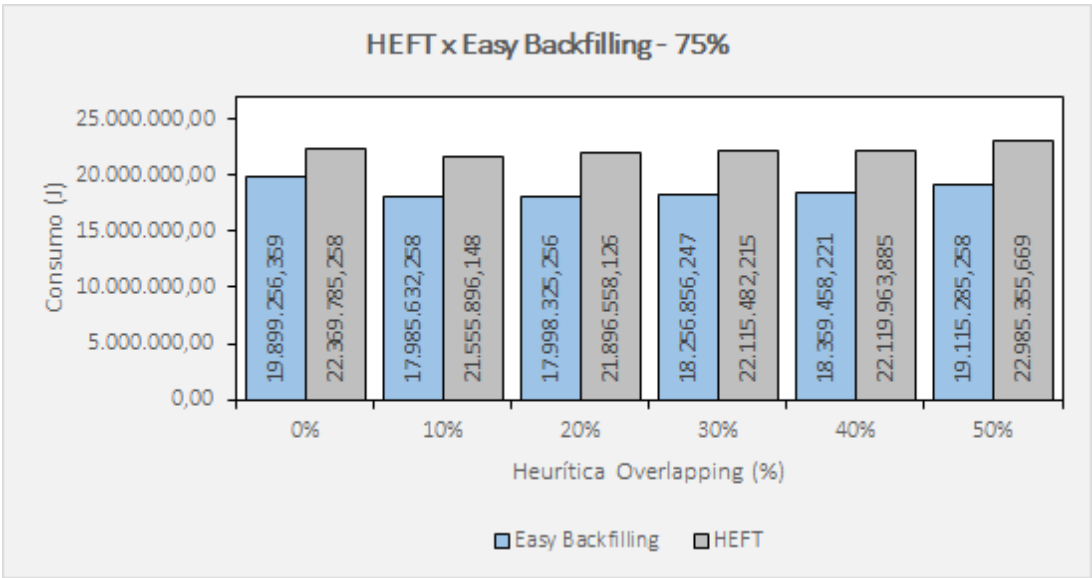


Figura 43 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 3500 Jobs e 75% de CPU

Quando o cenário utiliza 50% de CPU, os percentuais mais expressivos foram com o escalonador *Easy backfilling*, aplicando *overlap* de 10% com 7,8% de redução e o *HEFT* com *overlap* de 10% e 3,07% de redução. Os menores percentuais de redução foram com *overlap* de 50% e redução de 0,46%, e o *HEFT*, com *overlap* de 50% com redução de 1,99%, como apresenta a Figura 44.

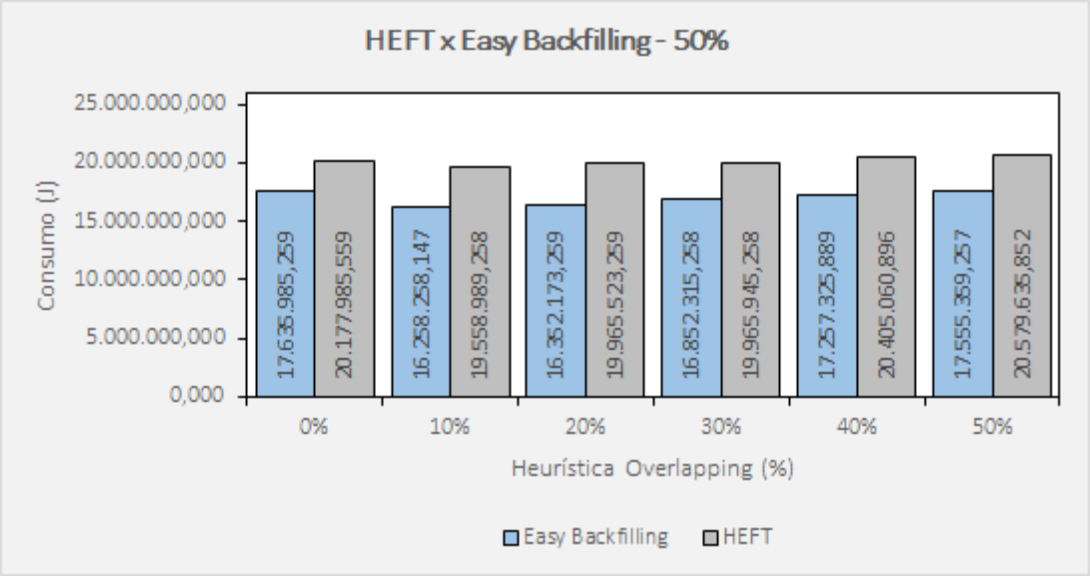


Figura 44 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 3500 Jobs e 50% de CPU

Como pode-se observar na Tebela 8, com *workload* de 1000 *jobs*, o maior percentual de redução aconteceu utilizando o escalonador *Easy Backfilling*, com 10% de *overlap* e utilização de processador de 50%, considerando que o consumo de energia com essa carga é de 16.258.258,147 Joules, enquanto que, com 100% de utilização de CPU esse consumo passa a ser 20.258.357,216 Joules com o mesmo percentual utilizado. A redução é significativa para o escalonador *Easy Backfilling*, em contrapartida, ao analisar o desempenho do consumo de energia do escalonador *HEFT*, conclui-se que, neste cenário sua utilização não apresentou uma redução significativa para ser o escalonador padrão em execução.

Tabela 8 – Apresentação dos maiores e menores resultados obtidos pela simulação com *workload* de 3500 *Jobs* para o consumo de energia elétrica

	Maior Redução		**Menor Redução**	
	Overlap	**(%) Resultado**	**Overlap**	**(%) Resultado**
Easy Backfilling- 100%	10%	7,01%	50%	0,51%
Easy Backfilling- 75%	10%	9,62%	50%	3,94%
Easy Backfilling- 50%	10%	7,80%	50%	0,46%
HEFT - 100%	10%	6,82%	50%	-2,47%
HEFT - 75%	10%	3,64%	50%	-2,75%
HEFT - 50%	10%	3,07%	50%	1,99%

Em alguns trabalhos que utilizaram o escalonador *HEFT*, percebeu-se que, quanto maior a carga utilizada, maior o tempo total de execução ao comparado com outros escalonadores, como apresentou os trabalhos de (ALIAGA; GOLDMAN, 2011) e (RIOS et al., 2014). Isto se da pela política de escalonamento com DAG, utilizada pelo *HEFT*, e já explanada na seção 3.1.1, onde pode acontecer um atraso no tempo total de execução, e no caso do presente trabalho, percebe-se um consumo maior de energia quando aplicado este escalonador em ambientes com um número de *jobs* maior para a execução, isto também é devido a política de escalonamento utilizada pelo *HEFT*.

O tempo de execução deste cenário, pode ser observado na Figura 45, onde o *Easy Backfilling* obteve seu melhor tempo de 40.258.357,2150 segundos com *overlap* de 10% atingindo um percentual de redução de 4,72%, enquanto o *HEFT* conseguiu o tempo de 44.875.635,8872 segundos com *overlap* de 10% chegando a percentual de redução do tempo de execução de 0,91%. Neste cenário, novamente o escalonador *Easy Backfilling* apresenta os melhores resultados de redução de consumo de energia.

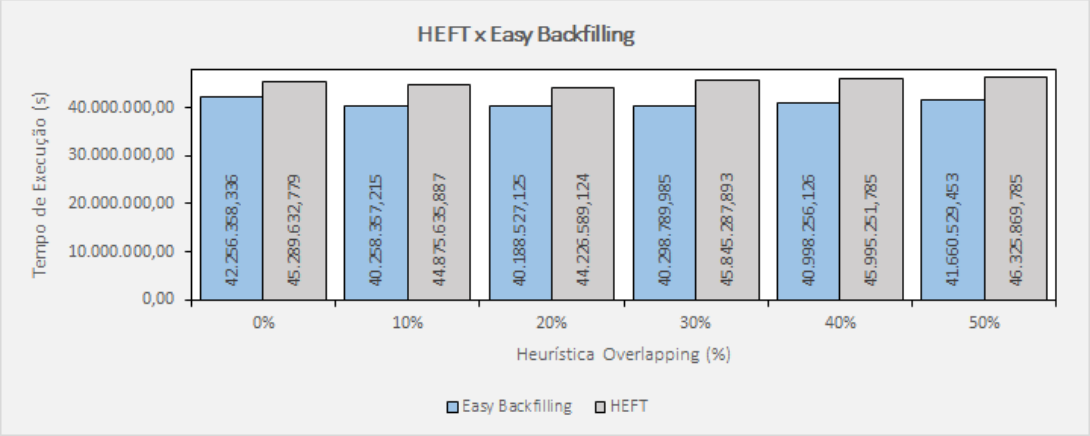
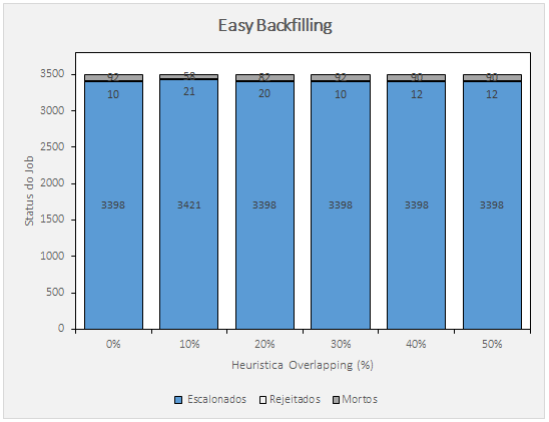


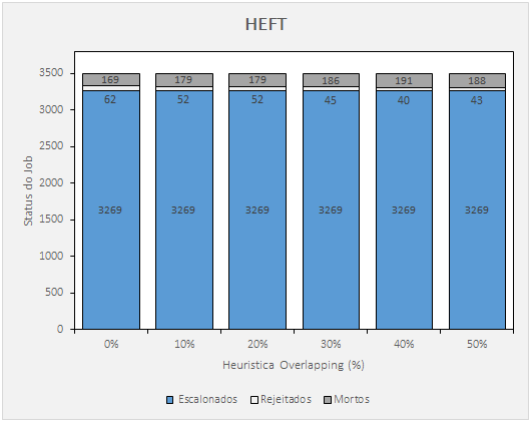
Figura 45 – Tempo de Execução dos Escalonadores com utilização da Heurística Overlap

A Figura 46 apresenta o status de cada *Job* escalonado, onde observa-se que o percentual médio de *jobs* mortos ou rejeitados fica em torno de 4%, em ambos os escalonadores, como o apresentado na Figura 46a, quanto o da Figura 46b. Para tanto, com os resultados encontrados nas Figura 45 e 46, foram calculados as taxas de execuções em Jobs/ segundos, onde o escalonador *Easy Backfilling* obteve uma taxa de execução de $8,49x10^{-5}$ Jobs/ segundo com o *overlap* de 10%, enquanto a taxa de execução do *HEFT* foi de $7,39x10^{-5}$ Jobs/ segundo com *overlap* de 20%.

Figura 46 – Comparativo dos Status dos 3500 Jobs escalonados pelo Batsim com os *Easy Backfilling* e *HEFT*



(a) Status do escalonamento executado pelo Easy Backfilling



(b) Status do escalonamento executado pelo *Heft*

5.3.6 Aplicação de *workloads* de 7500 *Jobs*

Na Figura 47, observa-se que, o percentual de ganho de energia do escalonador *Easy Backfilling* com utilização de CPU de 100%, foi com *overlap* de 10% e redução de 2,34%. Com o escalonador *HEFT* em 10% de *overlap* gerou uma redução de 5,33%. Neste cenário foi constatado que com a utilização de *overlap* de 50% para *Easy Backilling* e *HEFT*, gerou um acréscimo no consumo de energia de respectivamente -3,37% e -7,12%. Nessa situação o *HEFT* obteve um melhor desempenho utilizando a heurística de *overlap* de 10%, porém mesmo com menor desempenho com a heurística *overlap* de 10%, o *Easy Backfilling* demostrou maior redução no consumo de energia.

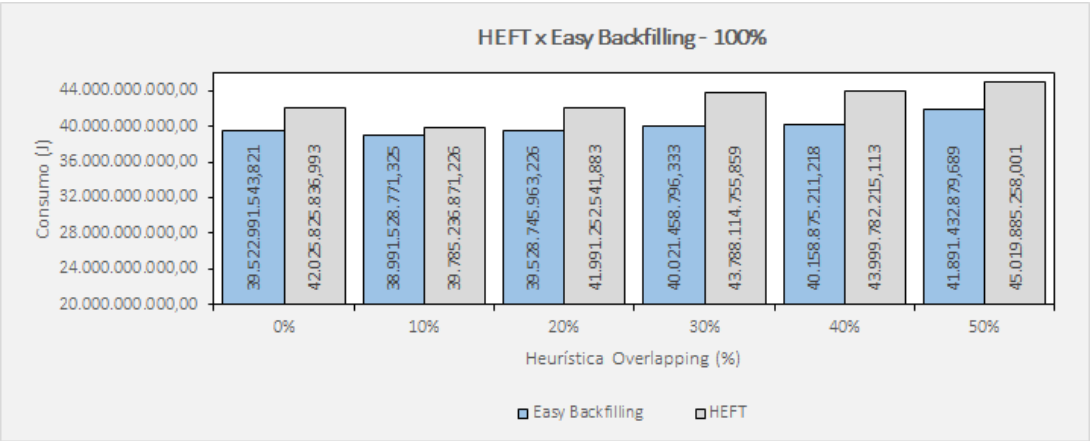


Figura 47 – Gráfico comparativo entre *HEFT* e Easy Backfilling com *workload* de 7500 Jobs e 100% de CPU

A Figura 48, apresenta a redução de utilização de CPU de 75%, onde o maior impacto de redução da energia no escalonador *Easy Backfilling* aconteceu com *overlap* de 10% a um percentual de 2,51% de redução. O menor desempenho no consumo de energia ocorreu com *overlap* de 50%, onde gerou um aumento no consumo de energia de -0,74%. Para os resultados obtidos nas simulações com o *HEFT*, pode-se destacar que a maior redução de consumo de energia ocorreu com *overlap* de 10% e 7,51% de redução, sendo que com 50% de*overlap* gerou um acréscimo de -1,36% no consumo de energia.

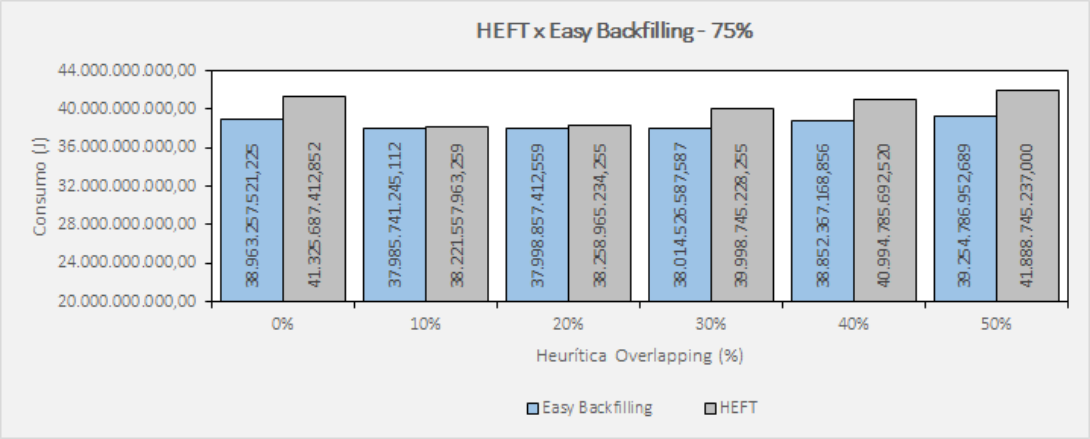


Figura 48 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 7500 Jobs e 75% de CPU

Para a utilização de CPU de 50%, pode-se observar na Figura 49 o consumo de energia, onde maior impacto na redução de energia no escalonador *Easy Backfilling* aconteceu com *overlap* de 10% que chega a 2,60% de redução, e o menor desempenho no consumo de energia ocorreu com *overlap* de 50%, onde gerou um acréscimo de consumo de -0,40%.

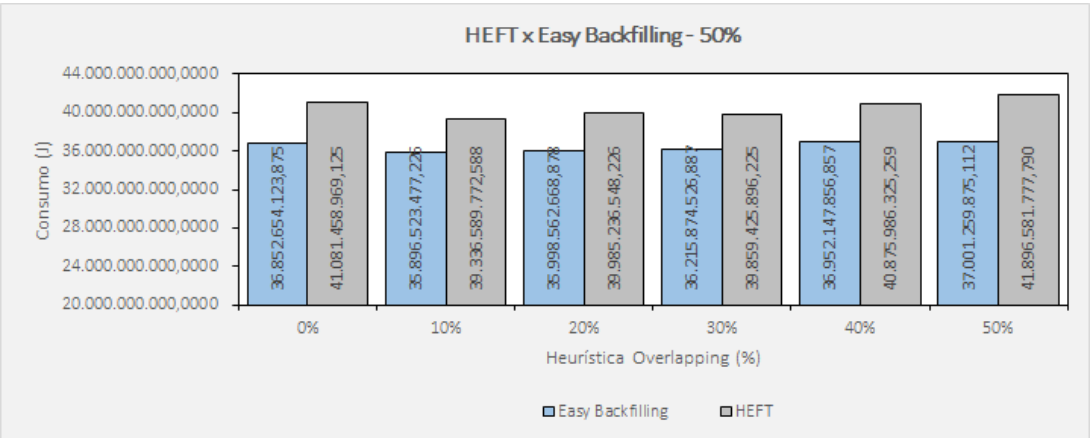


Figura 49 – Gráfico comparativo entre *HEFT* e *Easy Backfilling* com *workload* de 7500 Jobs e 50% de CPU

Com o Escalonador *HEFT*, a redução de energia de maior expressão aconteceu com a utilização de *overlap* de 10% que chegou a 4,25% de redução, enquanto que com 50% de *overlap*, ocorreu um aumento de -1,98% como observa-se na Figura 49.

A tabela 6 apresenta a utilização da heurística *overlap* com os principais consumos encontrados de energia elétrica para os escalonadores *Easy Backfilling* e *HEFT* em cenários de Grades Computacionais. Onde observa-se que, com *overlap* de 10 e 20% para ambos os escalonadores, obteve-se os melhores reduções no consumo de energia, e com *overlap* de 50% ocorreu um acréscimo no consumo de energia.

Tabela 9 – Apresentação dos maiores e menores resultados obtidos pela simulação com *workload* de 7500 Jobs para o consumo de energia elétrica

	Maior Redução		**Menor Redução**	
	Overlap	**(%) Resultado**	**Overlap**	**(%) Resultado**
Easy Backfilling- 100%	10%	2,34%	50%	-3,37%
Easy Backfilling- 75%	10%	2,50%	50%	-0,74%
Easy Backfilling- 50%	10%	2,60%	50%	-0,40%
HEFT - 100%	10%	5,33%	50%	-7,12%
HEFT - 75%	10%	7,51%	50%	-1,36%
HEFT - 50%	10%	4,25%	50%	-1,98%

Quanto ao tempo de execução, pode-se destacar que o *Easy Backfilling*, assim como o consumo de energia, obteve o tempo de execução melhor sobre o *HEFT*. Onde seu melhor desempenho foi de 94.259.825,2234 segundos com *overlap* de 10% atingindo um percentual de redução de 1,95%. Enquanto isso, o escalonador *HEFT* conseguiu o tempo de 95.991.258,9926 segundos com *overlap*g de 10% chegando ao percentual de redução do tempo de execução de 1,27% como mostra a Figura 50

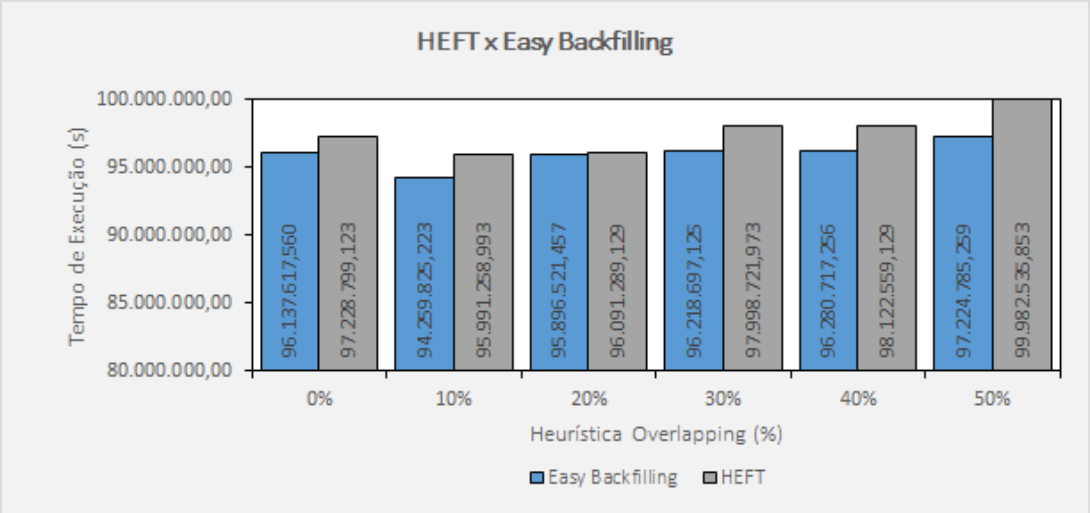
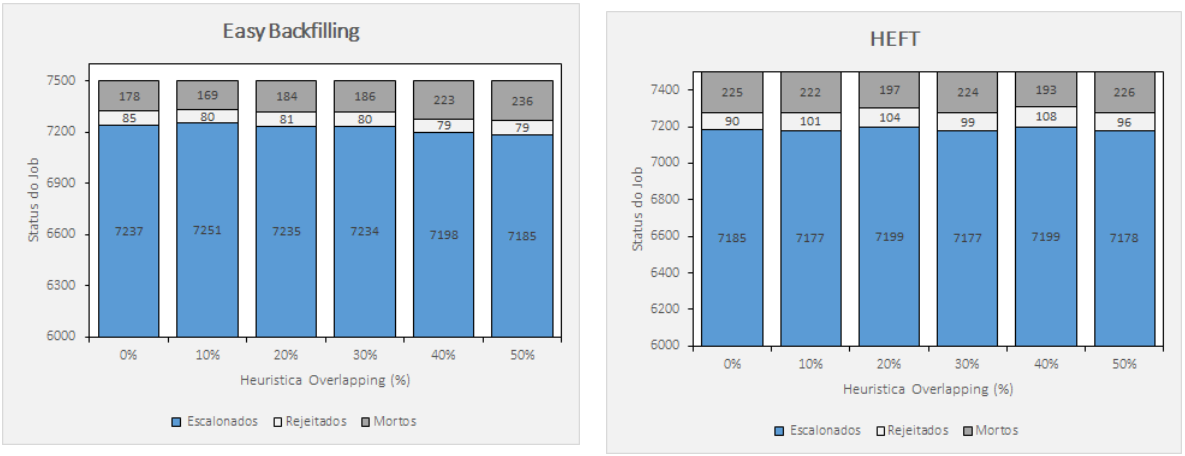


Figura 50 – Tempo de Execução dos Escalonadores com utilização da Heurística Overlap

A Figura 51 apresenta status de cada *Job* escalonado, onde nota-se que o percentual médio de *jobs* mortos ou rejeitados fica em média de 5%, em ambos os escalonadores, tanto o apresentado na Figura 51a, quanto na Figura 51b, isso acontece devido ao tempo excedido de execução de cada *jobs*. Sendo assim, com os resultados encontrados nas Figura 50 e 51, foram calculados as taxas de execuções em Jobs/ segundos, onde verificou-se que, o *Easy Backfilling* obteve uma taxa de execução de $7,69x10^{-5}$ Jobs/ segundo com o *overlap* de 10%, enquanto a taxa de execução do *HEFT* foi de $7,49x10^{-5}$ Jobs/ segundo com *overlap* de 20%.

Figura 51 – Comparativo dos Status dos 7500 Jobs escalonados pelo Batsim com os *Easy Backfilling* e *HEFT*



(a) Status do escalonamento executado pelo Easy Backfilling

(b) Status do escalonamento executado pelo *HEFT*

5.4 COMPILAÇÃO DOS RESULTADOS

Nesta seção serão apresentados os resultados que demostraram a maior redução de consumo de energia dentre os cenários implementados com utilização de CPU de 100%, 75% e 50%, simulados na ferramenta Batsim.

5.4.1 Compilação dos resultados com maior impacto no consumo de energia

A Figura 52 apresenta os resultados com maior impacto na redução de energia com a aplicação da heurística *overlap* com 100% de utilização de CPU em ambos os escalonadores. Pode-se observar que a cima de 100 *jobs*, as maiores reduções no consumo de energia foram com aplicação de*overlap* de 10%, o que significa que a redução do espaço ocioso no slot de processamento pode ser sobreposta com *jobs* que estão sendo executadas, sendo que com a aplicação de *overlap* superior a 10% ocorre uma sobrecarga de trabalho simultâneo podendo aumentar o consumo.

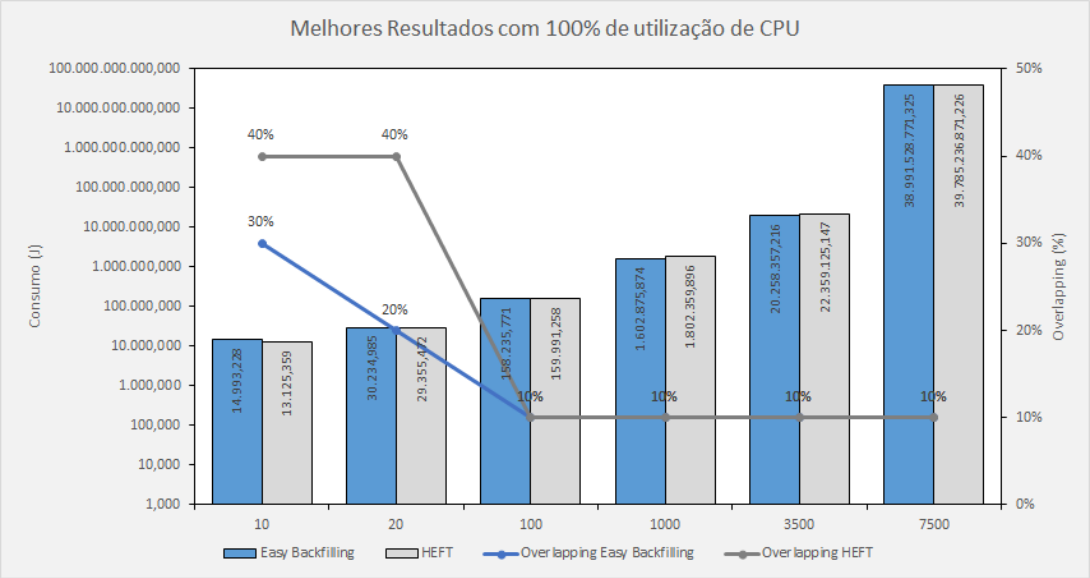


Figura 52 – Resultados com 100% de utilização de CPU e com maior impacto na redução de energia

Para os cenários aplicados com 10 e 20 *jobs*, com sobreposições de até 40%, demostraram maior redução no consumo de energia. Isto pode ocorrer devido ao baixo número de *jobs* na simulação, e uma maior disponibilidade de recursos na Grade Computacional.

Na Figura 53, com a utilização de CPU de 75%, o comportamento foi semelhante ao que aconteceu com as melhores performances encontradas com a utilização de CPU em 100%. Onde observou-se que, a partir de 20 *jobs* para ambos os escalonadores com 10% de *overlap* obteve-se as maiores reduções no consumo de energia.

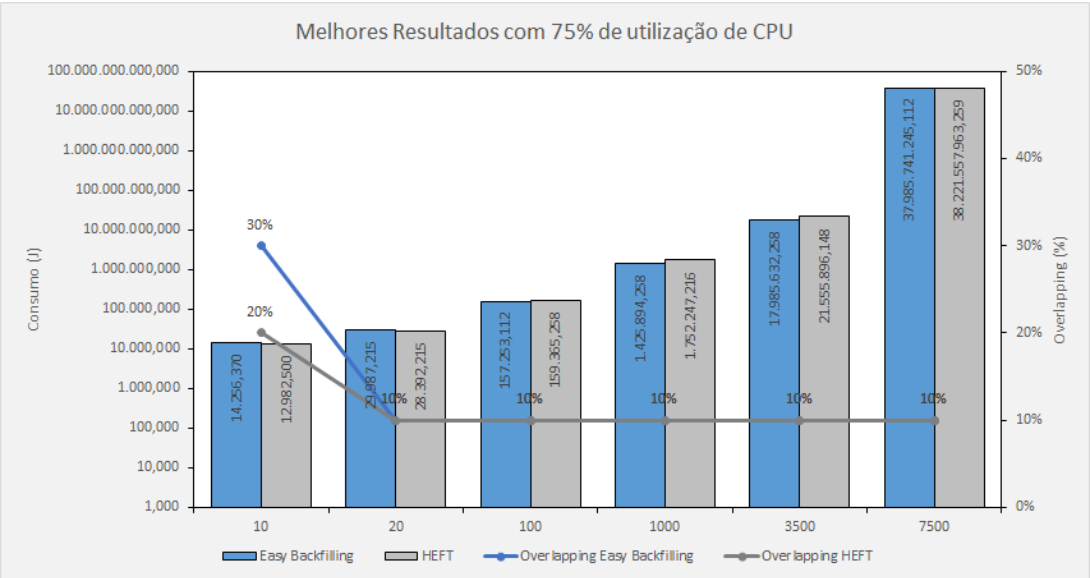


Figura 53 – Resultados com 75% de utilização de CPU e com maior impacto na redução de energia

Na Figura 54, com a utilização de CPU de 50%, pode-se observar que, para o escalonador *HEFT* a partir de *workloads* de 20 *jobs* tem um maior consumo de energia quando foi utilizado *overlap* de 10%. Para o *Easy Backfilling* as melhores performances no consumo de energia foram para *overlap* de 10% com utilização de *workloads* acima de 100 *jobs*.

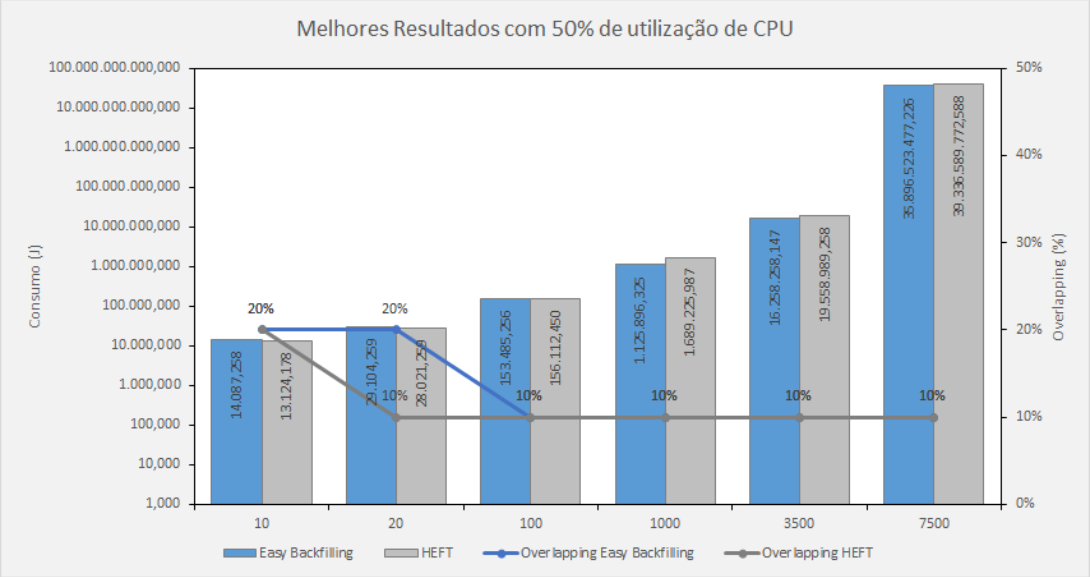


Figura 54 – Resultados com 50% de utilização de CPU e com maior impacto na redução de energia

5.4.2 Considerações Parciais

Com a aplicação da heurística *overlap* nos cenários estabelecidos com os workloads de 10, 20, 100, 1000, 3500 e 7500 *jobs* e sobreposição de 10%, 20%, 30%, 40% e 50% percebe-se uma melhora nos percentuais de redução de energia elétrica. Principalmente com a utilização do escalonador *Easy Backfilling* com sobreposição de 10% e workloads acima de 100. É notório que nestes cenários com número de *jobs* maiores o escalonador Easy Backfilling escalona de forma eficiente, tanto com a aplicação da heurística overlap, quanto sem, em comparação com o escalonador *HEFT*. Entretanto em cenários como o primeiro e o segundo com número de *jobs* menores, o *HEFT* obteve uma maior redução.

Ao comparar os resultados obtidos com a aplicação da heurística *overlap* nos cenários com utilização de CPU de 100%, 75% e 50% percebe-se que a maior redução de consumo foi com 50% de utilização de CPU. Identificando e analisando todos os resultados encontrados nas simulações, pode-se concluir que, a heurística *overlap*, aplicada para o escalonamento em Grades Computacionais reduz o consumo de energia elétrica.

6 CONSIDERAÇÕES FINAIS

A computação no ambiente de Grades é uma alternativa para a execução de aplicações paralelas e distribuídas que demandam um alto poder computacional. Essas aplicações são compostas de diversos *jobs* que têm por objetivo ser executadas buscando sempre um melhor desempenho (BORRO, 2014). No escopo de Computação em Grade o escalonamento é um desafio para atingir um bom desempenho no tempo de execução das aplicações. Para tanto, na literatura, foram propostos diversos algoritmos de escalonamento que buscam essa melhora no desempenho sem deixar de preocupar-se com a Computação Verde e a alta taxa de consumo de energia (AHMED et al., 2017).

Os algoritmos de escalonamento desenvolvidos para a Grade Computacional visam gerar um mapeamento eficiente dos *jobs* nos recursos disponíveis. Para isso, algumas características são levadas em consideração. Desta maneira, foi apresentado nesta dissertação dois escalonadores, *Easy Backfilling* e *HEFT* implementados com a heurística *overlap* (sobreposição de slots) com o intuito de amenizar a ociosidade dos recursos e consequentemente reduzir o consumo de energia elétrica.

A heurística *Overlap* foi implementada com os percentuais de sobreposição de 10%, 20%, 30%, 40% e 50% nos escalonadores, pois de maneira geral, a sobreposição de *slots* significa o compartilhamento de tempo no processador e a execução de dois ou mais *jobs*, ou seja, esta política permite que haja uma perturbação de um novo *jobs* naquelas previamente alocadas, visando a utilização de um *slot* que seria insuficiente para execução de um *jobs* mas que, de outro modo, permaneceria ocioso. Para esta análise foi utilizada a ferramenta de simulação *Batsim* e implementados alguns cenários destintos com cargas específicas buscando simular um ambiente real.

Para tanto, com o desenvolvimento do trabalho observou-se que, com a implementação da heurística de *overlap* em ambos os escalonadores, obteve-se ganhos de consumo de energia significativos. No entanto, pode-se destacar que, quando utilizava-se *workloads* menores, o escalonador *HEFT* obteve um desempenho energético melhor, porém quando *workloads* aumentam, o desempenho energético do *EASY Backfilling* destacava-se. Esse comportamento do *HEFT* entretanto, já seria o esperado, pois em trabalhos similares com a implementação do *HEFT* voltado para tempo de execução, seu comportamento foi exatamente igual, ou seja quanto maior a carga utilizada maior seu tempo de execução, se comparado a outros escalonadores (RIOS et al., 2014), (ALIAGA; GOLDMAN, 2011) e (GABRIEL, 2014).

Os resultados obtidos com os protótipos de implementação da heurística *over-*

lap aplicadados nos escalonadores *HEFT* e *Easy Backfilling* foram satisfatórios, pois de forma singular, conseguiram alcançar os objetivos de suas aplicações. De maneira geral a aplicação da heurística *overlap* de 10% independente do percentual de utilização de CPU, para ambos os escalonadores é o mais indicado para a diminuir a ociosidade dos recursos e consequentemente reduzir o consumo de energia. Sobreposições superiores a 10% pode ocorrer uma sobrecarga de trabalho simultânea, ocasionando menores reduções e até aumento no consumo de energia.

6.1 TRABALHOS FUTUROS

A comparação de algoritmos de escalonamento em ambientes de Grade Computacional ainda é um campo bastante extenso para pesquisa, pois além das dificuldades em aplicar os métodos em um ambiente real, se torna muitas vezes inviável pelo alto custo financeiro.

Como trabalhos futuros, pode-se sugerir a aplicação de mais algoritmos de escalonamento com a implementação da heurística *overlap* com cenários distintos e cargas maiores, já que existe uma infinidade de cenários disponíveis para a validação dos resultados deste estudo. Sendo assim, este estudo pode ser utilizado como base para outros trabalhos desenvolvidos por grupos de pesquisa que visão otimizar os recursos e o consumo de energia em ambientes de Grades Computacionais. Desta maneira espera-se avaliar a adaptação de decisões de escalonamento à medida em que as informações do ambiente são modificadas, e assim avaliar o impacto de estratégias utilizadas para tal finalidade.

REFERÊNCIAS

AHMED, M. et al. Power optimization of cfd applications on heterogeneous architectures. **International Journal of Computer Applications**, Foundation of Computer Science, v. 166, n. 8, 2017.

AHN, D. H. et al. Flux: A next-generation resource management framework for large hpc centers. In: IEEE. **Parallel Processing Workshops (ICCPW), 2014 43rd International Conference on**. [S.l.], 2014. p. 9–17.

ALIAGA, A. H. M.; GOLDMAN, A. **Estudo comparativo de escalonadores de tarefas para grades computacionais**. 2011.

ANDRIEUX, A. et al. Open issues in grid scheduling. **UK e-Science Report UKeS-2004-03, April 2004**, 2003.

ARTHI, T.; HAMEAD, H. S. Energy aware cloud service provisioning approach for green computing environment. In: IEEE. **Energy Efficient Technologies for Sustainability (ICEETS), 2013 International Conference on**. [S.l.], 2013. p. 139–144.

ASSUNCAO, M. D. D. et al. The green grid'5000: Instrumenting and using a grid with energy sensors. In: **Remote Instrumentation for eScience and Related Aspects**. [S.l.]: Springer, 2012. p. 25–42.

AVELAR, V. et al. Pue^{TM}: A comprehensive examination of the metric. **White paper**, v. 49, 2012.

BATISTA, D. M.; CHAVES, C. G.; FONSECA, N. L. da. Embedding software requirements in grid scheduling. In: IEEE. **Communications (ICC), 2011 IEEE International Conference on**. [S.l.], 2011. p. 1–6.

BATSIM. **Batsim-Jobs**. 2018. Disponível em: <https://batsim.readthedocs.io/en/latest/protocol.html#simulation-begins>.

BATSIM. **Batsim-Workloads**. 2018. Disponível em: <https://batsim.readthedocs.io/en/latest/input-workload.html#delay>.

BORRO, L. C. **Escalonamento em grades móveis: uma abordagem ciente do consumo de energia**. Tese (Doutorado) — Universidade de São Paulo, 2014.

BUYYA, R.; MURSHED, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. **Concurrency and computation: practice and experience**, Wiley Online Library, v. 14, n. 13-15, p. 1175–1220, 2002.

CAMPOS, P. P. V. **Um Algoritmo de Escalonamento para Redução do Consumo de Energia em Computação em Nuvem**. Tese (Doutorado) — Universidade de São Paulo, 2013.

CARVALHO, L. A. V. de; SANTANA, R. et al. A workflow scheduling algorithm for optimizing energy-efficient grid resources usage. In: IEEE. **Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on**. [S.l.], 2011. p. 642–649.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. **IEEE Transactions on software engineering**, IEEE, v. 14, n. 2, p. 141–154, 1988.

CHAWLA, S.; SALUJA, K. Enhanced job scheduling algorithm with budget constraints in computational grids. In: IEEE. **Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on**. [S.l.], 2016. p. 515–520.

DELVAZ, B. H.; BOVÉRIO, M. A. Ti verde. **Revista Interface Tecnológica**, v. 14, n. 1, p. 22–22, 2017.

DONG, F.; AKL, S. G. **Scheduling algorithms for grid computing: State of the art and open problems**. [S.l.], 2006.

DUTOT, P.-F. et al. Towards energy budget control in hpc. In: IEEE PRESS. **Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing**. [S.l.], 2017. p. 381–390.

DUTOT, P.-F. et al. Batsim: a realistic language-independent resources and jobs management systems simulator. In: SPRINGER. **Job Scheduling Strategies for Parallel Processing**. [S.l.], 2015. p. 178–197.

ENERGY, S. Energy star®. **History: ENERGY STAR**, 2011.

FERNANDES, F. : Um metodo de escalonamento baseado no comportamento de aplicações hpc para nuvens computacionais balanceando desempenho e eficiencia energetica. 102 f. Dissertação (Mestrado em Ciências Computacao) — Instituto Militar de Engenharia, Rio de janeiro, 2015.

FERNÁNDEZ-MONTES, A. et al. Smart scheduling for saving energy in grid computing. **Expert Systems with Applications**, Elsevier, v. 39, n. 10, p. 9443–9450, 2012.

FISCHBORN, M. et al. Computação de alto desempenho aplicada à análise de dispositivos eletromagnéticos. Florianópolis, SC, 2006.

FOSTER, I.; KESSELMAN, C. **The Grid 2: Blueprint for a new computing infrastructure**. [S.l.]: Elsevier, 2003.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. **The International Journal of High Performance Computing Applications**, Sage Publications Sage CA: Thousand Oaks, CA, v. 15, n. 3, p. 200–222, 2001.

FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. In: IEEE. **Grid Computing Environments Workshop, 2008. GCE'08**. [S.l.], 2008. p. 1–10.

FRANÇA, J. C. D. Método grasp baseado em computação verde aplicado ao escalonamento de recursos e tarefas em grades computacionais. 2014.

GABRIEL, P. H. R. **Uma abordagem orientada a sistemas para otimização de escalonamento de processos em grades computacionais**. Tese (Doutorado), 2014.

GAY, J.-S.; CANIOU, Y. **Simbatch: an API for simulating and predicting the performance of parallel resources and batch systems**. Tese (Doutorado) — INRIA, 2005.

GRID5000. **Escalonador OAR**. 2018. Disponível em: <http://oar.imag.fr/>.

HAN, G. et al. Resource-utilization-aware energy efficient server consolidation algorithm for green computing in iiot. **Journal of Network and Computer Applications**, Elsevier, v. 103, p. 205–214, 2018.

HIROFUCHI, T.; LEBRE, A.; POUILLOUX, L. Simgrid vm: virtual machine support for a simulation framework of distributed systems. **IEEE Transactions on Cloud Computing**, IEEE, v. 6, n. 1, p. 221–234, 2018.

ISMAIL, L. Performance versus cost of a parallel conjugate gradient method in cloud and commodity clusters. **INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY**, INT JOURNAL COMPUTER SCIENCE & NETWORK SECURITY-IJCSNS DAE-SANG OFFICE 301, SANGDO 5 DONG 509-1, SEOUL, 00000, SOUTH KOREA, v. 12, n. 11, p. 25–34, 2012.

JAIN, R. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: John Wiley & Sons, 1990.

KELEHER, P. J.; ZOTKIN, D.; PERKOVIC, D. Attacking the bottlenecks of backfilling schedulers. **Cluster Computing**, Springer, v. 3, n. 4, p. 245–254, 2000.

KIM, K. H.; BUYYA, R.; KIM, J. Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In: **CCGrid**. [S.l.: s.n.], 2007. v. 7, p. 541–548.

KLUSÁČEK, D.; CIRNE, W.; DESAI, N. **Job Scheduling Strategies for Parallel Processing: 21st International Workshop, JSSPP 2017, Orlando, FL, USA, June 2, 2017, Revised Selected Papers**. [S.l.]: Springer, 2018. v. 10773.

KLUSÁČEK, D.; RUDOVÁ, H. Alea 2: job scheduling simulator. In: ICST (INSTITUTE FOR COMPUTER SCIENCES, SOCIAL-INFORMATICS AND TELECOMMUNICATIONS ENGINEERING). **Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques**. [S.l.], 2010. p. 61.

KONDO, D. et al. Characterizing resource availability in enterprise desktop grids. **Future Generation Computer Systems**, Elsevier, v. 23, n. 7, p. 888–903, 2007.

KRAEMER, A. et al. Reducing the number of response time service level objective violations by a cloud-hpc convergence scheduler. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 30, n. 12, p. e4352, 2018.

KUMAR, P.; KUMAR, P.; KUMAR, V. Backfilling strategies for computational grid system load balancing. 2013.

KWOK, Y.-K.; AHMAD, I. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. **IEEE transactions on parallel and distributed systems**, IEEE, v. 7, n. 5, p. 506–521, 1996.

LELONG, J.; REIS, V.; TRYSTRAM, D. Tuning easy-backfilling queues. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. [S.l.], 2017. p. 43–61.

MÄMMELÄ, O. et al. Energy-aware job scheduler for high-performance computing. **Computer Science-Research and Development**, Springer, p. 1–11, 2012.

MÓR, S. D. et al. Eficiência energética em computaçao de alto desempenho: Uma abordagem em arquitetura e programaçao para green computing. **XXXVII Seminário Integrado de Software e Hardware-SEMISH**, p. 346–360, 2010.

MU'ALEM, A. W.; FEITELSON, D. G. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 12, n. 6, p. 529–543, 2001.

NETO, G. F. d. O. Proposta de conjunto de simulações para análise de desempenho de processadores superescalares e ensino de arquitetura de computadores. 2004.

ORGERIE, A.-C.; LEFÈVRE, L.; GELAS, J.-P. Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In: IEEE. **2010 14th IEEE international conference on parallel and distributed systems**. [S.l.], 2010. p. 171–178.

PASCUAL, J. A.; MIGUEL-ALONSO, J.; LOZANO, J. A. Locality-aware policies to improve job scheduling on 3d tori. **The Journal of Supercomputing**, Springer, v. 71, n. 3, p. 966–994, 2015.

PEREZ, F. J. R.; MIGUEL-ALONSO, J. Insee: An interconnection network simulation and evaluation environment. In: SPRINGER. **European Conference on Parallel Processing**. [S.l.], 2005. p. 1014–1023.

PONCIANO, L. et al. Análise de estratégias de computaç ao verde em grades computacionais oportunistas. 2010.

POQUET, M. Approche par la simulation pour la gestion de ressources. 2017.

POQUET, M. **Simulation approach for resource management**. Tese (Doutorado) — Université Grenoble Alpes, 2017.

RIOS, R. A. et al. Análise de heurısticas para escalonamento online de aplicaç oes em grade computacional. 2011.

RIOS, R. A. et al. Análise de heurısticas para escalonamento online de aplicaç oes em grade computacional. 2014.

RIOS, R. A. et al. Slot: uma ferramenta dinâmica para escalonamento global de aplicações em grades computacionais. Universidade Federal de São Carlos, 2008.

RODAMILANS, C. B. **Análise de desempenho de algoritmos de escalonamento de tarefas em grids computacionais usando simuladores.** Tese (Doutorado) — Universidade de São Paulo, 2009.

SCARAMELLA, J.; HEALEY, M. Service-based approaches to improving data center thermal and power efficiencies. **IDC White Paper**, 2007.

SCHOPF, J. M. A general architecture for scheduling on the grid. **Special issue of JPDC on Grid Computing**, Citeseer, v. 4, 2002.

SHARMA, S.; HSU, C.-H.; FENG, W.-c. Making a case for a green500 list. In: IEEE. **Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International**. [S.l.], 2006. p. 8–pp.

SILVA, M. R. et al. Ti verde–princípios e práticas sustentáveis para aplicação em universidades. **Anais do SIMPÓSIO BRASILEIRO DE SISTEMAS ELÉTRICOS–SBSE2010. Belém: Universidade Federal do Pará**, p. 1–6, 2010.

TANEMBAUM. **Sistemas Operacionais, Projeto e Implementação**. [S.l.: s.n.], 2012. 992 p.

TANG, Z. et al. An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment. **Journal of Grid Computing**, Springer, v. 14, n. 1, p. 55–74, 2016.

TEODORO, S.; CARMO, A. B. do; FERNANDES, L. G. Energy efficiency management in computational grids through energy-aware scheduling. In: ACM. **Proceedings Annual ACM Symposium on Applied Computing**. [S.l.], 2013. p. 1163–1168.

TOP500. **Supercomputadores**. 2018. Disponível em: <http://top500.org>.

TOPCUOGLU, H.; HARIRI, S.; WU, M.-y. Performance-effective and low-complexity task scheduling for heterogeneous computing. **IEEE transactions on parallel and distributed systems**, IEEE, v. 13, n. 3, p. 260–274, 2002.

WANG, J.; HAN, D.; WANG, R. A new rule-based power-aware job scheduler for supercomputers. **The Journal of Supercomputing**, Springer, v. 74, n. 6, p. 2508–2527, 2018.

WATANABE, E. et al. Algoritmos para economia de energia no escalonamento de workflows em nuvens computacionais. In: **32nd Brazilian Symposium on Computer Networks and Distributed Systems, Florianopolis, Brazil**. [S.l.: s.n.], 2014.

WERNER, J. et al. Uma abordagem para alocação de máquinas virtuais em ambientes de computação em nuvem verde. 2011.

XAVIER, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. In: IEEE. **Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on**. [S.l.], 2013. p. 233–240.

ZHAO, H.; SAKELLARIOU, R. Scheduling multiple dags onto heterogeneous systems. In: IEEE. **Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International**. [S.l.], 2006. p. 14–pp.