Para minimizar os custos operacionais de um sistema em grades, políticas de gerenciamento são utilizadas para coordenar o acesso aos recursos. Contudo, por se tratarem de sistemas complexos e dinâmicos, políticas tradicionalmente utilizadas são baseadas em regras fixas que não são otimizadas para os comportamentos do sistema. Neste contexto, o presente trabalho apresenta dois métodos baseados em *Deep Reinforcement Learning* para encontrar políticas de gerenciamento específicas para os padrões de utilização do sistema de modo a otimizar a eficiência energética. Com base em simulações, os resultados apontaram uma redução significativa no consumo de energia. Logo, foi possível concluir que, ao considerar o comportamento do sistema, políticas adaptativas tem maior potencial em sistemas dinâmicos e complexos.

Orientador: Maurício Aronne Pillon

Joinville, 2020

ANO
2020

LUCAS CAMELO CASAGRANDE | GERENCIAMENTO DE RECURSOS E PROCESSOS EM GRADES COMPUTACIONAIS ORIENTADO A ENERGIA COM *DEEP REINFORCEMENT LEARNING*

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

DISSERTAÇÃO DE MESTRADO

GERENCIAMENTO DE RECURSOS E PROCESSOS EM GRADES COMPUTACIONAIS ORIENTADO A ENERGIA COM *DEEP REINFORCEMENT LEARNING*

LUCAS CAMELO CASAGRANDE

JOINVILLE, 2020

UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
MESTRADO EM COMPUTAÇÃO APLICADA

LUCAS CAMELO CASAGRANDE

GERENCIAMENTO DE RECURSOS E PROCESSOS EM GRADES COMPUTACIONAIS ORIENTADO A ENERGIA COM *DEEP REINFORCEMENT LEARNING*

JOINVILLE

2020

LUCAS CAMELO CASAGRANDE

GERENCIAMENTO DE RECURSOS E PROCESSOS EM GRADES COMPUTACIONAIS ORIENTADO A ENERGIA COM *DEEP REINFORCEMENT LEARNING*

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: Dr. Maurício Aronne Pillon

JOINVILLE

2020

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Setorial do CCT/UDESC,

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

Casagrande, Lucas
Gerenciamento de Recursos e Processos em Grades Computacionais Orientado a Energia com Deep Reinforcement Learning / Lucas Casagrande. -- 2020.
121 p.

Orientador: Maurício Aronne Pillon
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2020.

1. Aprendizagem por Reforço. 2. Computação em Grade. 3. Gerenciamento de Recursos e Processos. 4. Escalonamento Orientado a Energia. I. Aronne Pillon, Maurício. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

Gerenciamento de Recursos e Processos em Grades Computacionais Orientado a Energia com *Deep Reinforcement Learning*

por

Lucas Camelo Casagrande

Esta dissertação foi julgada adequada para obtenção do título de

Mestre 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. Luis Carlos Erpen de Bona
UFPR

Joinville,SC, 18 de março de 2020.

Dedico este trabalho a Deus, pela vida e provento me fornecidos. Aos meus pais e familiares, pelo carinho e atenção me concebidos nos mais variados momentos. Ao meu orientador Dr. Maurício Pillon, por me ajudar a enfrentar os muitos desafios que surgiram neste trabalho. Ao meu professor Paulo Manseira, pelo suporte e visão de sabedoria que me foi compartilhada. Por fim, aos meus demais amigos e professores que me proporcionaram momentos de alegria e sabedoria.

AGRADECIMENTOS

Gostaria de agradecer a todos os envolvidos neste trabalho, em especial ao meu orientador Dr. Maurício Pillon e esposa Me. Edicarsia Pillon por me encaminharem e me suportarem no decorrer do mestrado e durante o período da minha graduação.

Ao meu professor e amigo Me. Paulo Manseira e família pelos mais valiosos aconselhamentos que poderia receber, os quais levarei pelo resto da vida.

Aos professores Dr. Charles Miers, Dr. Guilherme Koslovski e Dr. Nelson Gonzalez por suas valiosas contribuições e sugestões ao meu trabalho.

Aos meus professores Luiz Camargo, Rita Amorim, Robson, Guilherme, Jussara, e muitos outros, pela dedicação com que exercem seus trabalhos e por me ajudarem no conhecimento que tenho hoje.

Por fim, aos meus colegas e amigos pelos mais variados momentos de descontração.

"*If I have seen further it is by standing on the shoulders of Giants.*"

Isaac Newton

RESUMO

Grades computacionais introduzem uma abordagem que permite a composição de plataformas computacionais de larga escala através do compartilhamento de recursos entre instituições geograficamente distribuídas. Compartilhar recursos para obter maior poder computacional é atraente em termos financeiros, mas a complexidade e os custos operacionais podem inviabilizar o sistema caso políticas de gerenciamento eficientes não sejam adotadas. Por se tratarem de sistemas complexos e dinâmicos, políticas tradicionalmente utilizadas são baseadas em regras fixas que não são otimizadas para os comportamentos do sistema. Não obstante, detectar possíveis padrões de comportamento não é um processo trivial, pois o comportamento da carga de trabalho pode mudar repentinamente. Observado este cenário, o presente trabalho apresenta dois métodos baseados em *Deep Reinforcement Learning* (DRL) para encontrar políticas específicas para os padrões de utilização do sistema. Atuando no problema de *shutdown*, a proposta do DeepShutdown adota uma estratégia de *Off Reservation* (OR) e utiliza métodos de DRL para encontrar uma política que controle o desligamento dos recursos de modo a minimizar o desperdício de energia. Com este mesmo objetivo, a proposta do DeepScheduler é treinada com DRL para encontrar uma política de escalonamento adequada para os tipos de cargas de trabalho no sistema. Ambos os métodos são treinados e avaliados com os *traces* da GRID'5000 em um ambiente de simulação chamado de GridGym, que foi desenvolvido neste trabalho. Observando os resultados, foi possível obter uma redução média de 46% no desperdício de energia com uma quantidade similar de reinicializações de recursos comparando o DeepShutdown com políticas baseadas em regras. No contexto do DeepScheduler, comparado com políticas de escalonamento tradicionalmente utilizadas, foi possível obter uma economia de energia de até 7% com uma redução no tempo médio de espera de até 26%. Logo, é possível concluir que, ao considerar o comportamento do sistema, políticas adaptativas tem maior potencial em sistemas dinâmicos e complexos.

Palavras-chaves: Aprendizagem por Reforço, Computação em Grades, Gerenciamento de Recursos e Processos e Escalonamento Orientado a Energia.

ABSTRACT

Grid computing enables the composition of large-scale platforms through the sharing of resources among multiple geographically distributed institutions. Sharing resources to increase the computational power available is attractive, but the complexity and operational costs can impact the sustainability of the system if efficient management policies are not deployed. Given the dynamic and complex nature of grid systems, policies traditionally deployed are rule-based and are not optimized for the system usage patterns. Moreover, discovering system usage patterns is not straightforward because of the unexpected behavior changes in the workload. In this context, we propose two methods based on *Deep Reinforcement Learning* (DRL) to build system-specific policies. Acting on the shutdown problem, the DeepShutdown proposal adopts an *Off Reservation* (OR) strategy and uses DRL to find a shutdown policy that minimizes the energy wastes. With this same objective, the DeepScheduler proposal is trained with DRL to adapt a scheduling policy to the workloads being executed. Both methods are trained and evaluated with real workloads traces extracted from the GRID'5000 in a simulated environment developed in this work called GridGym. Based on experiments, with the DeepShutdown approach we observed a reduction of 46% on the averaged energy waste with an equivalent frequency of shutdown events compared to commonly used shutdown policies. In the DeepScheduler context, we observed an energy waste reduction of 7%, as well as average requests waiting time reduction of 26%, compared to traditionally deployed scheduling policies. Finally, by considering system usage patterns, adaptive policies have a higher potential to optimize the efficiency of complex and dynamic systems.

Key-words: Reinforcement Learning, Grid Computing, Resource and Job Management and Energy-Aware Scheduling.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A2C *Advantage Actor-Critic*
A3C *Asynchronous Advantage Actor-Critic*
ALE *Arcade Learning Environment*
CBF *Conservative Backfilling*
DL *Deep Learning*
DNN *Deep Neural Network*
DQN *Deep Q-Network*
DRL *Deep Reinforcement Learning*
DVFS *Dynamic Voltage and Frequency Scaling*
EASY *Extensible Argonne Scheduling sYstem Backfilling*
FCFS *First Come First Served*
FIFO *First In First Out*
FIQ Faixa Interquartil
FNN *Feedforward Neural Network*
GAE *Generalized Advantage Estimation*
GIS *Grid Information System*
GRU *Gated Recurrent Unit*
HPC *High Performance Computing*
LSF *Load Sharing Facility*
LSTM *Long Short-Term Memory*
MBox Machine Learning Box
MDP *Markov Decision Process*
ML *Machine Learning*
MuJoCo *MUlti-JOint dynamics with COntact*
NSCC *National Supercomputing Centre*
OR *Off Reservation*
OS *Opportunistic Shutdown*
P2P *Peer-to-Peer*
PC *Power Capping*
PPO *Proximal Policy Optimization*
pp-sld *Per-processor Slowdown*
PS *Proportional Shutdown*
QoS *Quality-of-Service*

RJMS *Resource and Job Management System*

RL *Reinforcement Learning*

SAF *Smallest estimate Area First*

SL *Supervised Learning*

TD *Temporal-Difference*

UL *Unsupervised Learning*

VO *Virtual Organization*

WCG *World Community Grid*

SUMÁRIO

1 INTRODUÇÃO

A computação em grades surgiu como um paradigma de computação distribuída que possibilitou a construção de plataformas computacionais de larga escala através da agregação de recursos geograficamente distribuídos. Para obter maior poder computacional, instituições definem uma *Virtual Organization* (VO) que irá estabelecer as regras de compartilhamento, descobrimento e alocação de seus recursos locais. Deste modo, os usuários destas instituições têm a seu dispor uma quantidade de poder computacional que excede a soma de suas partes (FOSTER; KESSELMAN, 2003; PRIMET; ANHALT; KOSLOVSKI, 2009). Não surpreendente, este paradigma ganhou popularidade na execução de aplicações paralelas de uso computacional intensivo e na condução de experimentos científicos em larga escala (POQUET, 2017).

Compartilhar recursos para construir plataformas computacionais de larga escala é atraente em termos de disponibilidade de recursos e acessível financeiramente, em termos de capital de investimento. Através da adição de novos provedores às VOs estabelecidas, um sistema em grades pode aumentar dinamicamente a disponibilidade de recursos em sua infraestrutura (GALIZIA; QUARATI, 2012). A maior disponibilidade de recursos possibilita explorar níveis mais profundos de paralelismos e aumentar a vazão do sistema. Logo, é natural que tais plataformas fossem construídas com uma visão de desempenho a qualquer custo (FENG, 2005).

A busca desenfreada por plataformas com alto poder de processamento é atraente em termos de desempenho, mas pode não ser sustentável sob aspectos financeiros. O consumo de energia de uma plataforma computacional possui um crescimento quase linear em função da utilização dos recursos (SUN et al., 2015). Não surpreendente, o custo energético representa uma despesa financeira significativa que afeta diretamente a sustentabilidade da plataforma. A maior demanda por energia pode limitar o tamanho da plataforma e tornar insustentável a operacionalização do sistema (DAYARATHNA; WEN; FAN, 2016). Por este motivo, estratégias de gerenciamento energeticamente eficientes estão entre as prioridades dos provedores de recursos e de seus administradores (TARPLEE et al., 2016; HINZ et al., 2018).

Otimizar a eficiência energética de uma plataforma não depende somente da escolha por componentes de *hardware* mais eficientes, mas também está relacionado às estratégias de gerenciamento implementadas no sistema (NESMACHNOW et al., 2013; ORGERIE; ASSUNCAO; LEFEVRE, 2014). Utilizado para otimizar e coordenar a utilização de uma grade computacional, o *Resource and Job Management System* (RJMS) é o mecanismo responsável pelo gerenciamento dos recursos e pro-

cessos. Neste contexto, processos são objetos oriundos das submissões dos usuários ao sistema enquanto recursos podem abranger desde um agregado de um provedor específico como toda a plataforma. Deste modo, o RJMS permite o gerenciamento do sistema como um todo, possibilitando a implantação de políticas e estratégias sobre diferentes níveis de controle, como: nível de aplicação, de escalonamento e de plataforma (POQUET, 2017).

A otimização do RJMS tem sido vastamente estudada na literatura e existem uma variedade de estratégias, cada qual atuando em um dos níveis de controle do RJMS. Como exemplo, a energia consumida por máquinas pouco utilizadas pode ser minimizada através do emprego de técnicas de *Dynamic Voltage and Frequency Scaling* (DVFS) no nível de aplicação. Com base nos padrões de utilização das aplicações, uma técnica de DVFS pode reduzir a frequência do processador e minimizar o consumo de energia quando a aplicação não está executando tarefas de uso computacional intensivo (YOUNG et al., 2013; HUANG; FENG, 2009).

Considerando o nível de escalonamento, técnicas de *Power Capping* (PC) e *energy budget* podem ser utilizadas para limitar o consumo de energia do sistema (BORGHESI et al., 2015; DUTOT et al., 2017). Políticas de escalonamento cientes do consumo energético também podem ser aplicadas para escalonar os processos levando em consideração o impacto no consumo de energia (SHI; ZHANG; ROBERTAZZI, 2017). Enquanto, no nível de plataforma, técnicas de *shutdown* podem ser utilizadas para aproveitar o tempo entre as execuções e minimizar a quantidade de recursos ociosos, através do seu desligamento (BENOIT et al., 2018; HIKITA; HIRANO; NAKASHIMA, 2008; TERZOPOULOS; KARATZA, 2013). Comparando as diferentes possibilidades, técnicas de *shutdown* estão entre as que possuem maior impacto esperado na economia de energia e, em conjunto com as técnicas de escalonamento, estão entre as soluções que possuem maior interesse pelos provedores (BATES et al., 2015).

Identificar os momentos que compensam o desligamento de um recurso computacional, de modo a não prejudicar o desempenho ou a expectativa de vida do hardware, depende tanto da carga de trabalho como da configuração de hardware da plataforma. Uma política agressiva de desligamento, que desliga os recursos imediatamente após a sua liberação, pode aumentar o consumo de energia em comparação a uma política suave, que espera um tempo antes de ordenar o desligamento. Para exemplificar esta situação, caso um recurso tenha um tempo de reinicialização de 15 minutos, distribuído entre inicializar e desligar, e o intervalo entre as submissões dos processos seja menor ou igual a este tempo, a aplicação de uma política agressiva de desligamento não compensaria em termos de desempenho. Ao invés de estar servindo um processo, o recurso permaneceria ocupado reinicializando por uma parte

considerável de tempo. Não obstante, a reinicialização constante do recurso pode não compensar financeiramente caso o seu custo, em termos de demanda de energia, seja superior ao custo quando ocioso (LIU et al., 2017; RAïS et al., 2018).

Tal situação também pode ser visualizada no contexto do escalonamento. Caso a política de escalonamento não seja sensível a energia, o escalonador pode aumentar o consumo de energia ao forçar a inicialização de recursos para aplicações com um curto tempo de execução. Neste contexto, atrasar a execução de alguns processos na expectativa que recursos sejam liberados prematuramente ou condensar a fila de processos na menor quantidade de recursos possível pode minimizar o desperdício de energia (POQUET, 2017; PINHEIRO et al., 2001). Contudo, elaborar tais políticas de gerenciamento, tanto de recursos (*e.g. shutdown*) como de processos (*e.g.* escalonamento), não é uma tarefa trivial devido a natureza dos sistemas em grades (ORGERIE; LEFÈVRE; GELAS, 2008).

Sistemas em grades possuem uma natureza dinâmica e complexa. As cargas de trabalho podem mudar de comportamento repentinamente, intensificando a complexidade em utilizar dados históricos para identificar padrões e elaborar políticas específicas para o sistema. Consequentemente, técnicas atualmente utilizadas são baseadas em regras simples e não levam em consideração as características da carga de trabalho ou do sistema. Utilizar o mesmo conjunto de regras em sistemas com padrões de utilização e configuração de plataforma distintas resulta em uma abordagem não otimizada (LEGRAND; TRUSTRAM; ZRIGUI, 2019). Deste modo, políticas adaptativas podem ser uma possível alternativa para desenvolver políticas específicas para os padrões de cada sistema (ORHEAN; POP; RAICU, 2018; KINTSAKIS; PSOMOPOULOS; MITKAS, 2019).

No contexto de controle adaptativo, *Reinforcement Learning* (RL) introduz métodos computacionais que possibilitam o treinamento de um agente através da interação com um ambiente para resolver processos de decisão. Em cada interação, o agente recebe uma observação sobre o estado interno do ambiente e, através de sua política, determina uma ação. A ação é executada no ambiente, influenciando o seu estado interno e gerando uma nova observação. Em conjunto com a nova observação, o ambiente fornece um estímulo (chamado de recompensa) ao agente que reflete a qualidade da ação executada. Com base neste estímulo, o objetivo do agente é encontrar as ações que maximizem a recompensa acumulada esperada (SUTTON; BARTO; WILLIAMS, 1992). Tal capacidade torna RL uma possível solução para otimizar políticas em função das observações que recebem.

Contudo, abordagens tradicionais de RL (*i.e.*, Q-learning e SARSA) possuem limitações que podem impossibilitar que o agente encontre uma política ótima devido a quantidade de estados e ações possíveis em um ambiente. Considerando a natureza

de um sistema em grades, tal estratégia pode não ser aplicada sem um trabalho manual na redução do escopo do problema (ORHEAN; POP; RAICU, 2018). Neste caso, técnicas de *Deep Reinforcement Learning* (DRL), que aplicam *Deep Learning* (DL) no contexto de RL, podem ser utilizadas para aproximar uma solução (MNIH et al., 2015).

Este trabalho propõe duas técnicas baseadas em DRL para encontrar políticas de gerenciamento específicas para os padrões do sistema de modo a otimizar a eficiência energética em grades computacionais. A primeira proposta, intitulada de DeepShutdown, busca otimizar o gerenciamento de recursos trabalhando no problema de desligamento. A segunda proposta, intitulada de DeepScheduler, busca otimizar o gerenciamento de processos trabalhando no problema de escalonamento. Ambas propostas possuem um objetivo comum, minimizar o desperdício de energia, mas trabalham em níveis de controle distintos. Deste modo, é possível observar a aplicabilidade de DRL no RJMS sobre cenários distintos.

Para avaliar o desempenho dos métodos propostos, é conduzida uma série de simulações utilizando como base as cargas de trabalho e configuração de *hardware* da GRID'5000, um sistema em grades para experimentos científicos que permanece em produção na França (BOLZE et al., 2006). Através destes experimentos, é analisado o impacto da carga de trabalho em diferentes heurísticas e como uma abordagem adaptativa pode resultar em uma política otimizada para o sistema. Não obstante, para lidar com as peculiaridades do treinamento com técnicas de DRL, também é desenvolvido uma extensão ao Batsim, um simulador de RJMS para grades computacionais, através de sua integração com o OpenAi Gym, um *framework* de RL. Chamado de GridGym, esta extensão simula o comportamento do RJMS com o Batsim e provê uma interface para a interação com algoritmos de RL.

Observando os resultados obtidos, foi possível atingir níveis consideráveis de economia de energia em ambas as propostas. Comparando o DeepShutdown com uma política agressiva de shutdown, foi observado uma economia média de energia de 11,7% com uma quantidade de reinicializações de recursos 17,9% menor. Em comparação a uma política suave, foi obtido uma economia de energia de até 46% com um acréscimo de 4% na quantidade de reinicializações. No contexto de escalonamento, o DeepScheduler foi capaz de reduzir em até 7% o consumo de energia e, ao mesmo tempo, em até 26% a média do tempo de espera em algumas cargas de trabalho quando comparado às políticas tradicionais. Logo, ao considerar as peculiaridades de cada carga de trabalho é possível maximizar a eficiência energética da plataforma através da otimização do seu gerenciamento.

1.1 OBJETIVO

A natureza de uma abordagem de DRL torna atrativa a sua adoção em sistemas dinâmicos e complexos. Estratégias baseadas em DRL podem aprender o comportamento do ambiente e considerar padrões não explorados pelas estratégias convencionais, baseadas em regras. Logo, é possível treinar um agente a encontrar uma política que se adapte aos padrões de utilização do sistema. O objetivo deste trabalho é dar um primeiro passo nesta direção.

Como principais contribuições, é possível citar:

- Através dos *traces* da GRID'5000, é demonstrado como submissões sequencias de processos podem influenciar o desempenho e a eficiência energética.
- São propostas uma política de escalonamento e uma de *shutdown* que exploram as propriedades e padrões das cargas de trabalho para otimizar a eficiência energética do sistema.
- É demonstrado como os problemas de *shutdown* e escalonamento podem ser modelados em um *Markov Decision Process* (MDP) de modo a possibilitar o emprego de métodos de RL.
- É conduzido uma série de experimentos para avaliar a adaptabilidade das políticas aprendidas pelos métodos propostos.
- É desenvolvido e disponibilizado uma extensão ao simulador Batsim (DUTOT et al., 2017) para lidar com as peculiaridades do treinamento com métodos de DRL. Esta integração (nomeada de GridGym), provê uma série de ambientes padronizados que modelam distintos problemas de gerenciamento de recursos e processos em grades computacionais.

1.2 MÉTODO DA PESQUISA

O método adotado neste trabalho é caracterizado pela adoção de uma pesquisa exploratória. Em uma primeira etapa, uma revisão sistemática é conduzida para levantar estudos sobre a adoção de RL em sistemas distribuídos. Com esta mesma finalidade, mecanismos de busca acadêmicos são utilizados para levantar pesquisas no âmbito de grades computacionais e sistemas de gerenciamento de recursos. O objetivo é entender as necessidades esperadas em estratégias de gerenciamento.

Em uma segunda etapa, estudos sobre DRL e suas aplicações são levantados com o objetivo de identificar possíveis limitações. Por fim, em posse de tais informações, hipóteses são levantadas e experimentos conduzidos para demonstrar

a efetividade de uma abordagem de DRL em um ambiente que representa o *modus operandi* de uma grade computacional.

1.3 ESTRUTURA DO TEXTO

O trabalho está estruturado da seguinte maneira. No Capítulo 2 é apresentada uma revisão da literatura sobre a composição de grades computacionais e sistemas de gerenciamento de recursos. Uma breve discussão sobre as abordagens de gerenciamento orientadas ao consumo de energia também é realizada em conjunto com a relevância deste tema. O estado da arte em DRL e os algoritmos de RL, que servem como base, são apresentados no Capítulo 3. Em conjunto com este tema, são levantados alguns trabalhos relacionados que visam aplicar DRL no contexto de gerenciamento de recursos e processos em sistemas distribuídos. No Capítulo 4 é apresentado a descrição dos métodos propostos e do processo de treinamento. No Capítulo 5 é descrito a metodologia de avaliação, sendo introduzido o GridGym e apresentado os resultados obtidos para então ser aberto uma discussão sobre o comportamento aprendido por ambas as propostas deste trabalho. Finalmente, as considerações finais, em conjunto com iniciativas para o aperfeiçoamento dos estudos abordados neste trabalho, são apresentadas no Capítulo 6.

2 GRADES COMPUTACIONAIS

A computação em grades é um paradigma de computação distribuída que possibilita a composição de plataformas de larga-escala através do compartilhamento de recursos entre múltiplas instituições. Conciliar os distintos requisitos e objetivos de cada organização participante em um único ambiente levanta desafios acerca da segurança, interoperabilidade e a transparência na distribuição. Deste modo, a implantação de protocolos padronizados e a otimização de seus procedimentos são necessários para garantir os níveis de eficiência esperados por seus administradores. O objetivo deste capítulo é dar uma visão geral sobre estes procedimentos e salientar os aspectos de sua estrutura que impactam na eficiência de sua operacionalização.

Com esta finalidade, o presente capítulo está organizado em cinco seções. Na Seção 2.1 é dada uma visão geral sobre a composição e o funcionamento de uma grade computacional de propósito geral. Na Seção 2.2 um modelo de arquitetura para grades computacionais é apresentado e discutido. Na Seção 2.3 é explicado como o gerenciamento de recursos é realizado e qual o seu impacto na eficiência do sistema. Na Seção 2.4 é levantado aspectos sobre o desperdício de energia em plataformas computacionais e as estratégias propostas para otimizar a eficiência energética. Finalmente, na Seção 2.5 é apresentado as considerações parciais.

2.1 INTRODUÇÃO

De forma análoga a uma malha energética, grades computacionais são contextualizadas através da agregação de recursos computacionais locais de múltiplas organizações que podem estar geograficamente distribuídas. Deste modo, o conceito de grade computacional abrange um conjunto de serviços, protocolos e padrões que objetivam permitir a coordenação destes recursos de modo seguro, dinâmico e transparente. Como resultado, organizações têm a seu dispor uma capacidade computacional maior que a soma de suas partes (FOSTER; KESSELMAN; TUECKE, 2001).

O conceito de grades computacionais não é um tópico recente. Sua origem data da década de 1990 como resultado de um esforço conjunto para construir ambientes de computação de alto desempenho. Ambientes largamente distribuídos que provessem acesso confiável, consistente e difundido a recursos computacionais locais, como computadores pessoais e estações de trabalho. Sobre um aspecto geral, seu objetivo principal era desenvolver a tecnologia necessária para lidar com os desafios advindos deste compartilhamento (FOSTER; KESSELMAN, 1998).

Coordenar a interligação de recursos computacionais e conciliar os interes-

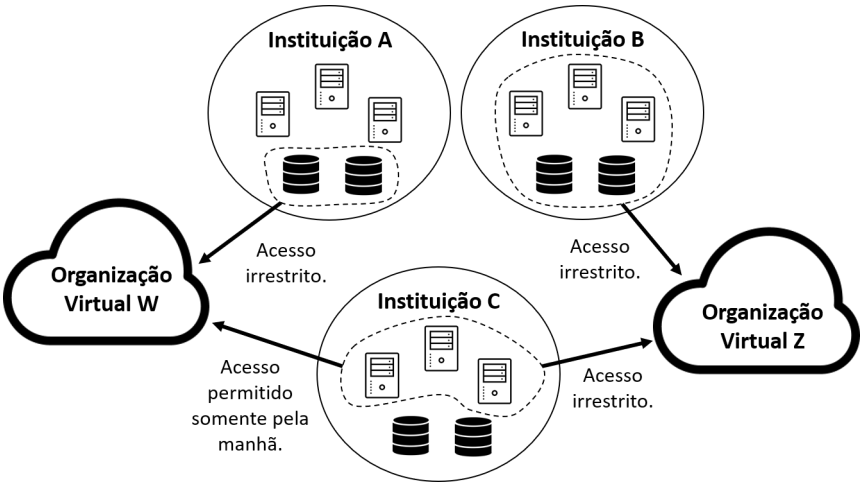
ses distintos de cada indivíduo e organização participante de uma grade não é uma tarefa simples. Para formar uma plataforma em grades, as organizações devem conceder o acesso remoto e direto aos seus recursos locais. Logo, é imprescindível que este *modus operandi* contenha políticas de utilização e de segurança para preservar os interesses individuais. Fica a critério de cada organização definir as condições de utilização de seus recursos, como também, quais destes podem ser compartilhados. Tal afirmação não implica necessariamente a utilização exclusiva de recursos computacionais, mas podem abranger recursos de armazenamento, de rede e sensores de monitoramento.

Devido as inúmeras possibilidades quanto aos objetivos e restrições de cada organização, é imprescindível que regras de compartilhamento sejam previamente estabelecidas e disseminadas entre os seus usuários. Através deste tipo de contrato, novos entrantes possuem as informações necessárias para utilizar e permitir o compartilhamento de seus próprios recursos. É neste contexto que as grades computacionais introduzem o conceito de *Virtual Organization* (VO) para formalizar o relacionamento do compartilhamento entre as organizações (FOSTER; KESSELMAN, 2003).

Na Figura 1 é ilustrado um exemplo da composição de uma VO. Neste exemplo são utilizadas três instituições (representadas pelas letras A, B e C) e duas VOs (representadas pelas letras W e Z) são formadas. A Instituição A participa da VO W enquanto a Instituição B participa da VO Z e a Instituição C participa das duas VOs. Com base nas definições de cada VO, a Instituição A permite que os participantes da VO W acessem os seus recursos de armazenamento sem qualquer restrição. A Instituição C compartilha somente os seus recursos computacionais e permite o seu acesso somente no período matutino. Na VO Z, a Instituição B permite o acesso completo a todos os seus recursos locais, sem restrições, e a Instituição C permite o acesso aos seus recursos computacionais sem restrição de período, como ocorre na VO W. Deste modo, a definição de uma VO possibilita visualizar como a plataforma está constituída e quais regras estão em vigor (FOSTER, 2002).

Usuários de uma plataforma em grades podem utilizar informações contidas na definição da VO para determinar os recursos para as suas aplicações. A Figura 2 ilustra a visão de um usuário sobre uma plataforma em grades e o procedimento padrão para a submissão de aplicações. Três instituições geograficamente distribuídas participam de uma única VO na qual são compartilhados todos os recursos, que estão organizados em agregados. Cada agregado contém um número arbitrário de máquinas com um ou mais processadores e núcleos. Tais recursos não necessariamente precisam ser homogêneos e a granularidade na qual estes podem ser alocados pode variar (*e.g.* núcleos, processadores ou máquinas). Por simplicidade, é considerado que os recursos são computacionais e cada usuário pode reservar um ou mais processa-

Figura 1 – Exemplo da organização de instituições em VOs.



Fonte: Adaptado de Foster, Kesselman e Tuecke (2001)

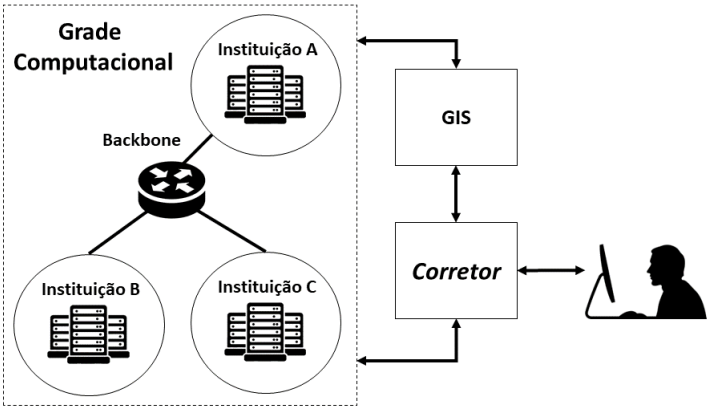
dores individualmente. Com esta finalidade, usuários interagem com a grade através de um intermediador chamado de corretor.

O objetivo do corretor é fornecer uma abstração de alto nível para que os usuários possam alocar os processadores das máquinas de modo transparente. Não obstante, é no corretor que os usuários vão monitorar e coletar os resultados de suas aplicações, reservar recursos e identificar os momentos em que estes estão disponíveis. Para obter estas informações, o corretor utiliza um serviço interno chamado de *Grid Information System* (GIS). Eventos que ocorrem na plataforma são transmitidos através do GIS para o corretor, que filtra e repassa estas informações para os usuários. Deste modo, a partir do corretor, os usuários visualizam o sistema como sendo um *pool* de recursos sob demanda (BUYYA; VENUGOPAL, 2004).

Seguindo este conceito geral, sistemas em grades podem tomar forma de maneiras distintas. Levando em consideração o tipo de aplicação esperada, uma grade pode ser construída e especificamente otimizada. Neste contexto, sistemas em grades podem ser divididas em três categorias, sendo: grades computacionais, grades de dados e grades de serviços (KRAUTER; BUYYA; MAHESWARAN, 2002; WILKINSON, 2009). Grades computacionais objetivam à execução de aplicações paralelas de alto desempenho (KURDI; LI; AL-RAWESHIDY, 2008). Aplicações de modelagem molecular e simulações são dois exemplos que podem melhor usufruir deste tipo de sistema (QURESHI et al., 2014). Visando atender aplicações de *Big Data* e mineração de dados, grades de dados possuem uma infraestrutura otimizada para lidar com grandes volumes de dados. Logo, este tipo de sistema é constituído por repositórios distribuídos que são otimizados e fornecem serviços de gerenciamento de dados (MAGOULÈS, 2009). Não menos importante, grades de serviços objetivam fornecer ser-

viços de alto nível sobre a infraestrutura. Como exemplo, este tipo de sistema pode implementar um ambiente colaborativo para os seus usuários (KRAUTER; BUYYA; MAHESWARAN, 2002).

Figura 2 – Visão geral de uma grade computacional.



Fonte: Adaptado de Buyya e Venugopal (2004)

Independe da categoria de um sistema em grades, interoperabilidade e transparência são dois desafios impulsionados pela complexidade da interligação de sistemas distintos. Não obstante, existem uma série de outros fatores que devem ser trabalhados, como por exemplo: mecanismos para o gerenciamento e alocação de recursos; mecanismos para a aplicação de políticas de segurança; e o desenvolvimento de aplicações e portais que providenciam uma interface para os seus usuários (FOSTER; KESSELMAN; TUECKE, 2001; BAKER; BUYYA; LAFORENZA, 2000).

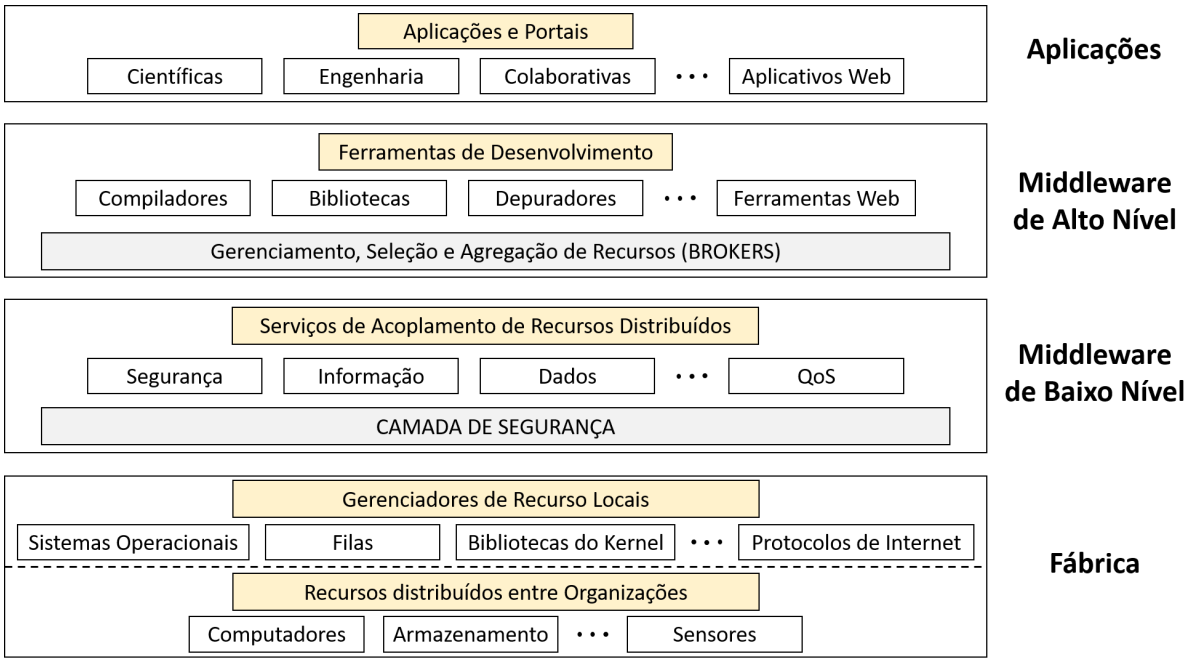
Devido às diferentes possibilidades quanto às possíveis implementações para cada tipo de sistema que pode fazer parte de uma grade, é quase inerente a necessidade de organizar estes sistemas em um algum modelo de arquitetura. De fato, organizando o problema em uma arquitetura é possível diminuir a complexidade como um todo e lidar com cada um destes desafios individualmente, seguindo a ideia de dividir para conquistar. Por este motivo, é importante conhecer a arquitetura de uma grade para entender como os seus elementos trabalham e como podem ser otimizados de modo a otimizar a eficiência do sistema.

2.2 ARQUITETURA

Para lidar com os desafios impulsionados pela implementação de um sistema em grades, acerca da integração entre sistemas não limitados a homogeneidade, a arquitetura de uma grade é organizada hierarquicamente em camadas. Cada camada é responsável por implementar uma série de componentes que fornecem um nível

de abstração, implementando serviços e protocolos, aos componentes das camadas superiores. Componentes de uma mesma camada compartilham características comuns e usufruem dos serviços das camadas inferiores para compor novos serviços. Seguindo esta lógica, na Figura 3 é possível visualizar a arquitetura de um sistema em grades dividida em quatro camadas, sendo: Fábrica, *Middleware* de baixo nível, *Middleware* de alto nível e Aplicações (BUYYA, 2002).

Figura 3 – Arquitetura geral de uma grade computacional.



Fonte: Adaptado de Buyya (2002)

Com base na arquitetura apresentada na Figura 3, os recursos locais dos provedores participantes da grade são representados na camada de Fábrica. Nesta camada é implementado abstrações sobre as operações locais de baixo nível. Logo, componentes desta camada objetivam abstrair o acesso as funcionalidades específicas de cada recurso da plataforma. Alguns exemplos destas funcionalidades, são:

- Identificação de estrutura interna do recurso e de seu estado atual, incluindo o estado da fila e a quantidade de trabalho remanescente.
- Identificação dos serviços implementados pelo fornecedor e de sua capacidade computacional ou de armazenamento.
- Início, controle e monitoramento da execução das aplicações.

A camada de Fábrica provê acesso às funcionalidades locais de cada recurso. Contudo, ainda é necessário integrar estas funcionalidades e permitir a sua comunica-

ção com os demais recursos. Com esta finalidade, *middlewares* de baixo e alto nível são implementados sobre a camada de Fábrica. No *middleware* de baixo nível é implementado serviços, como: gerenciamento de execução, gerenciamento de informação, gerenciamento dados, autenticação e autorização (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003).

Com base nas abstrações fornecidas pelo *middleware* de baixo nível, é possível implementar serviços que gerenciam um agregado de recursos e abstraem a sua operacionalização conjunta. Com este objetivo, o *middleware* de alto nível é definido e contém funcionalidades para lidar, com: o monitoramento dos recursos, escalonamento, execução e monitoramento dos processos. O *Resource and Job Management System* (RJMS) é implementado nesta camada, sendo o responsável por trabalhar cada funcionalidade relacionada a operacionalização dos agregados de recursos. A partir do RJMS, é possível desenvolver aplicações e portais web que permitem a geração de relatórios e a interação dos usuários com o sistema. Tais serviços são fornecidos na camada de aplicações, que interage com as camadas de *middleware* para prover serviços de alto nível (BUYYA; VENUGOPAL, 2004).

Seguindo esta arquitetura, um sistema em grades tem a sua complexidade distribuída em níveis de abstração. Cada nível provê uma abstração no acesso as funcionalidades dos recursos para permitir a sua integração e operacionalização conjunta através do RJMS. Logo, a partir deste mecanismo é possível gerenciar um agregado de recursos e distribuir o seu acesso entre os múltiplos usuários existentes. Assim, o RJMS exerce um papel crucial na eficiência de um sistema em grades. O modo em como os recursos são alocados e selecionados pode impactar o desempenho das aplicações e do sistema. Por este motivo, para otimizar a eficiência de um sistema em grade é necessário conhecer a estrutura que compõem um RJMS.

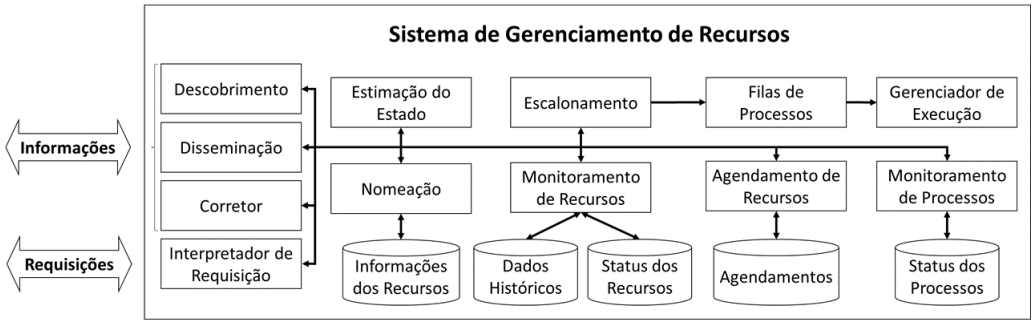
2.3 GERENCIAMENTO DE RECURSOS E PROCESSOS

Responsável pelos mecanismos de gerenciamento em um sistema em grades, o RJMS é considerado um componente chave para minimizar o custo operacional e otimizar o desempenho. Tal informação é respaldada pelas funcionalidades esperadas deste componente, como: descobrimento e alocação, em relação aos recursos; e escalonamento, monitoramento, execução e coleta dos resultados das aplicações, em relação aos processos (BUYYA; VENUGOPAL, 2004).

Dada a dinamicidade de um sistema em grades, RJMSs são sistemas naturalmente complexos. A heterogeneidade e distribuição dos recursos, distintos domínios administrativos e a escalabilidade são alguns dos fatores que acompanham o desenvolvimento de um sistema em grades. Neste contexto, o RJMS deve ser capaz de tra-

tar estes fatores de modo eficiente para não desperdiçar os recursos do sistema. Não obstante, estes sistemas devem ser adaptáveis para não só acomodar mudanças nas regras de compartilhamento, entre as instituições, mas também em sua infraestrutura. Logo, é compreensível que um RJMS seja modelado de forma similar à arquitetura de um sistema em grades. Seguindo este modelo, na Figura 4 é ilustrado uma visão geral da estrutura de um RJMS (KANDAGATLA, 2003).

Figura 4 – Visão geral da estrutura de um RJMS.



Fonte: Adaptado de Krauter, Buyya e Maheswaran (2002)

Como é possível observar na Figura 4, o RJMS é composto por componentes independentes que atuam de modo coordenado. Como exemplo, quando um usuário faz a requisição por um recurso é função do interpretador de requisição repassar as informações para o componente de agendamento, que detém informações sobre todas as reservas de recursos em seu repositório local. Neste mesmo contexto, enquanto o interpretador processa as requisições dos usuários, um corretor provê controle e acesso às informações sobre os recursos do sistema. Neste componente são implementadas interfaces para os processos de descobrimento e disseminação, responsáveis por fornecer uma visão global de todos os recursos que podem ser encontrados no sistema (KRAUTER; BUYYA; MAHESWARAN, 2002).

Para utilizar os recursos de um sistema em grades é necessário conhecer quais recursos estão disponíveis. Com tal finalidade, o processo de descobrimento realiza uma busca pela rede para encontrar os recursos que atendam aos requisitos definidos pelos usuários, em suas submissões (KANDAGATLA, 2003). É um mecanismo que trabalha em conjunto com o mecanismo de disseminação e espaço de nomes para obter informações atualizadas e os endereços dos recursos do sistema, respectivamente (MAHESWARAN; KRAUTER, 2000).

Através do interpretador de requisição e do corretor, usuários possuem a seu dispor todas as informações necessárias tanto para submeter novas aplicações como para coletar os seus resultados ou verificar o estado de execução. Contudo, estes componentes apenas fornecem uma abstração de alto nível para os usuários enquanto

componentes internos são responsáveis por providenciar os serviços necessários. Logo, um RJMS possui componentes internos, como: espaço de nomes, monitoramento de recursos e processos, agendamento, escalonamento e gerenciamento de execução (KRAUTER; BUYYA; MAHESWARAN, 2002).

No componente de espaço de nomes é definido as regras que irão ditar como os recursos serão nomeados e identificados em um sistema em grades (MESHKOVA et al., 2008). O componente de escalonamento é o responsável por determinar a ordem na qual os processos devem iniciar a sua execução e quais recursos devem ser alocados. Após a tomada de decisão, o componente de gerenciamento de execução é responsável por instanciar os processos nos recursos definidos no escalonamento (KRAUTER; BUYYA; MAHESWARAN, 2002). No decorrer da execução, componentes de monitoramento ficam a cargo de atualizar e persistir as informações sobre o estado atual dos recursos e dos processos que estão no sistema. Tais informações são utilizadas para medir o desempenho do sistema, através de diferentes métricas de desempenho, e para detectar possíveis falhas nos recursos (ZANIKOLAS; SAKELLARIOU, 2005).

Seguindo a ideia de dividir para conquistar, as funcionalidades do RJMS são distribuídas entre componentes internos e externos. Componentes internos implementam funções específicas de gerenciamento e controle de recursos e processos enquanto os componentes externos fornecem uma abstração de alto nível para os usuários. Contudo, o *modus operandi* de um RJMS em relação ao seu nível de controle sobre a plataforma depende dos objetivos dos administradores, que utilizam métricas de desempenho para avaliar a eficiência do sistema. Estas métricas podem quantificar o custo operacional e os ganhos de desempenho como função dos mecanismos adotados no RJMS (POQUET, 2017). Logo, para derivar métricas que enriqueçam às avaliações de desempenho do sistema, é necessário primeiro caracterizar uma submissão de aplicação para conhecer suas propriedades. Após esta descrição, o processo de escalonamento pode ser discutido como um meio para otimizar a eficiência do RJMS.

2.3.1 Caracterização de Processos

A caracterização dos processos submetidos ao sistema é um passo importante na implementação do RJMS pois o seu comportamento pode ser otimizado de acordo com o tipo de processo esperado. Um processo pode ser caracterizado quanto ao seu nível de flexibilidade e ao modo de execução, que determina o quanto este pode ser modificado pelo escalonador. Em relação a flexibilidade dos processos, quatro categorias são definidas, sendo (FEITELSON; RUDOLPH, 1996):

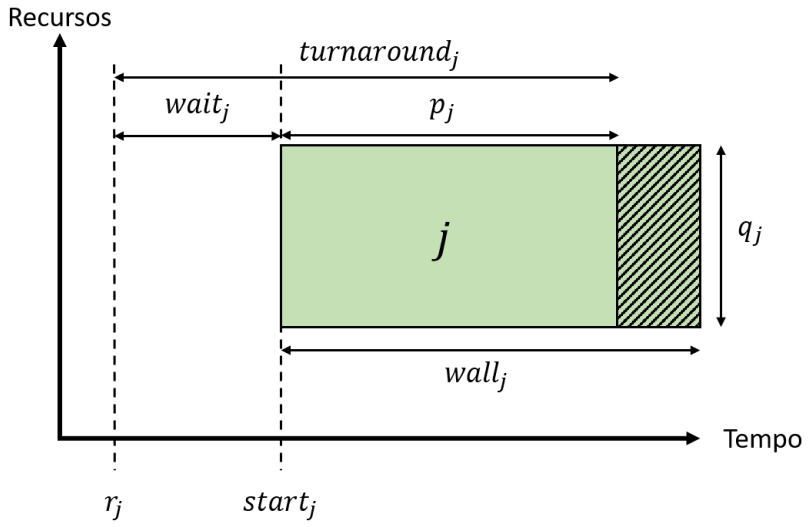
- Moldável: nesta categoria os processos determinam uma quantidade mínima de recursos que devem ser alocados. Desde que o processo não tenha iniciado sua execução, o escalonador pode decidir aumentar a quantidade de recursos alocados para maximizar o desempenho (SABIN; LANG; SADAYAPPAN, 2007).
- Maleável: processos maleáveis podem ter a sua alocação modificada dinamicamente pelo escalonador. A diferença em relação aos processos moldáveis está em que o maleável pode ser modificado após o início da execução do processo. Isto permite que o escalonador tenha liberdade para flexibilizar à alocação e aumentar a utilização do sistema quando necessário (BARSANTI; SODAN, 2006).
- Rígido: processos desta categoria não podem ter a sua alocação modificada pelo sistema. No momento da requisição, o processo estabelece uma quantidade fixa de recursos que deve ser mantida durante todo o seu tempo no sistema. Este é o tipo mais comum e utilizado em sistemas distribuídos (PRABHAKARAN, 2016).
- Em evolução: processos em evolução podem modificar a sua requisição dinamicamente, pedindo por mais ou menos recursos. Neste caso, o escalonador deve atender a requisição e modificar em tempo real os recursos alocados (PRABHAKARAN et al., 2015).

Em relação ao modo de execução de cada processo em um sistema em grades, é possível definir três categorias, sendo:

- Modo Interativo: usuários que executam processos em modo interativo recebem *feedback* da execução em tempo real, através do terminal (EMERAS et al., 2017).
- Modo *Batch*: processos que executam em *batch* são colocados na fila e iniciados somente quando existem recursos disponíveis.
- Modo Alocação: similar ao modo interativo. Contudo, neste modo o usuário pode manualmente executar o seu processo nos recursos que lhe foram alocados (YOO; JETTE; GRONDONA, 2003).

Processos podem divergir em relação ao seu nível de flexibilidade e modo de operação, assim como podem coexistir no sistema. Contudo, algumas características são compartilhadas entre todos os processos $j \in J$ de uma carga de trabalho J, sendo: a quantidade de recursos requisitada (q_j); a quantidade de tempo requisitada ($wall_j$), também chamado de *walltime*; o tempo de submissão (r_j); e o tempo de execução (p_j) (POQUET, 2017; LELONG; REIS; TRYSTRAM, 2018). Uma ilustração de um processo j descrevendo estas características, pode ser visualizada na Figura 5.

Figura 5 – Características de um processo em grades computacionais.



Fonte: Adaptado de Poquet (2017)

Com base nas características dos processos, métricas de desempenho podem ser derivadas e os mecanismos de gerenciamento do RJMS avaliados. Com esta finalidade, métricas tradicionalmente utilizadas, são (CARASTAN-SANTOS et al., 2019):

- Makespan C_{max}: um indicativo da vazão do sistema. Está métrica corresponde ao tempo requerido para completar uma carga de trabalho J. Formalmente definido, por:

$$C_{max} = \max_{j \in J}(start_j + p_j) - \min_{j \in J}(r_j) \tag{2.1}$$

  Sendo, $start_j$ o tempo de início de execução do processo j (LEUNG; SABIN; SADAYAPPAN, 2010).

- Média do Tempo de Espera: indica a média de tempo em que os processos permaneceram na fila. Formalmente definido, por:

$$\frac{1}{|J|} \sum_j wait_j \tag{2.2}$$

  Sendo, $wait_j = start_j - r_j$ o tempo de espera do processo j (ETINSKI et al., 2012a).

- Média do Tempo de *Turnaround*: indica a média de tempo em que os processos permaneceram no sistema. Formalmente definido, por:

$$\frac{1}{|J|} \sum_j turnaround_j \tag{2.3}$$

Sendo, $turnaround_j = wait_j + p_j$ o tempo em que o processo j permaneceu no sistema (PRABHAKARAN, 2016).

- Taxa de utilização: está métrica mostra a porcentagem de recursos que foram utilizados em um intervalo de tempo. Considerando o intervalo de tempo dado pelo makespan (C_{max}), está métrica pode ser formalmente definida, por:

$$\sum_j \frac{q_j \times p_j}{C_{max} \times |R|} \tag{2.4}$$

Sendo, $|R|$ o número total de recursos no sistema (XHAFA; ABRAHAM, 2010).

- Média de *Slowdown*: métrica que normaliza o tempo de espera com base no tempo de execução dos processos. Pode ser utilizada como um indicativo da justiça de uma política de escalonamento, pois segue a lógica que processos com tempos de execução longos (*e.g.* 1 hora) podem esperar mais na fila em comparação com processos curtos em tempo de execução (*e.g.* alguns minutos). Formalmente definido, por:

$$\frac{1}{|J|} \sum_j slowdown_j \tag{2.5}$$

Sendo, *slowdown* do processo j formalmente definido, por (FEITELSON, 2001):

$$\max(\frac{turnaround_j}{p_j}, 1) \tag{2.6}$$

- Média de *Slowdown* Delimitado: introduz um valor de corte no *slowdown* para minimizar a influência de processos com pouco tempo de execução. Formalmente definido, por:

$$\frac{1}{|J|} \sum_j bslowdown_j^{\pi} \tag{2.7}$$

Sendo, π o valor de corte e $bslowdown_j^{\pi}$ o *slowdown* delimitado do processo j, formalmente definido, por (FEITELSON; RUDOLPH, 1998):

$$\max(\frac{turnaround_j}{\max(p_j, \pi)}, 1) \tag{2.8}$$

- *Per-processor Slowdown*: em grades computacionais é possível executar processos paralelos em múltiplos recursos. Neste caso, o *slowdown* pode não ser a melhor métrica para avaliar a justiça do escalonador, pois não considera os recursos alocados para cada processo. Com esta finalidade, uma métrica que pode ser utilizada é a média do *Per-processor Slowdown* (pp-sld).

$$\max(\frac{wait_j + p_j}{q_j \times p_j}, 1) \tag{2.9}$$

Formalmente definido na Equação (2.9), o pp-sld considera que processos pequenos, em tempo de execução e em quantidade de recursos, devem aguardar menos na fila para executarem em relação aos demais processos. Logo, pode melhor representar o nível de justiça dos mecanismos de gerenciamento considerando processos paralelos (CARASTAN-SANTOS et al., 2019).

A partir da execução de um conjunto de processos em um sistema em grades é possível quantificar a eficiência do sistema através das inúmeras métricas apresentadas. Neste contexto, o RJMS é o mecanismo crucial que impacta diretamente no desempenho. O ato de escalonar processos e alocar recursos tem uma função primordial, dado que o início da execução de um processo é determinado pelo escalonamento enquanto o seu tempo de execução pode ser influenciado pelos recursos que lhe foram alocados. Logo, conhecer a etapa de escalonamento pode ser determinante para otimizar o gerenciamento dos recursos de um sistema em grades.

2.3.2 Escalonamento

O processo de escalonamento corresponde ao serviço de distribuir os recursos disponíveis entre os processos que estão aguardando na fila. Em outras palavras, é realizado um mapeamento entre recursos e processos seguindo uma política, que determina a sua tomada de decisão. Com esta finalidade, cada política de escalonamento utiliza um conjunto de métricas e informações dos processos alinhado as informações dos recursos disponíveis para ditar a ordem de execução (RODERO; GUIM; CORBALAN, 2009). Assim, é possível dividir a etapa de escalonamento em duas funções principais, sendo:

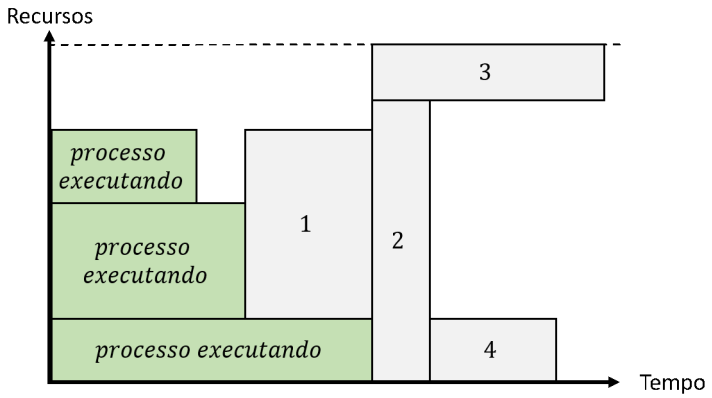
- Alocação de Recursos: Consiste em selecionar e alocar os recursos que atendem aos requisitos dos processos. Podem ser utilizados algoritmos simples, que selecionam os primeiros recursos encontrados através de um *matchmaking*, ou algoritmos mais complexos, como heurísticas que buscam uma combinação de recursos que podem minimizar uma métrica de desempenho, *e.g.* tempo de execução (SCHOPF, 2002; SMITH, 2004).
- Ordenamento da Execução: Consiste em determinar prioridades de execução para cada um dos processos na fila (LEUNG; SABIN; SADAYAPPAN, 2010).

A otimização de políticas de escalonamento é um tópico vasto que permanece em constante desenvolvimento. No contexto de sistemas em grades, algumas políticas tem sido consagradas como as mais bem sucedidas. Tais políticas utilizam um mecanismo de *backfilling* sobre a política de escalonamento *First Come First Served* (FCFS). O FCFS executa os processos respeitando a sua ordem de chegada, ou

seja, independentemente da quantidade de recursos disponíveis, nenhum processo pode passar a vez na fila (SRINIVASAN et al., 2002; POQUET, 2017).

Um exemplo do escalonamento de processos com a política FCFS pode ser visualizada na Figura 6. No tempo corrente, existem três processos executando enquanto quatro processos estão aguardando na fila. O primeiro e o segundo processo da fila não podem ser iniciados pois não existem recursos disponíveis. Enquanto o terceiro e o quarto processo, podem ser iniciados. Contudo, devido a política do FCFS, estes não podem iniciar antes do início dos processos prioritários na fila. Logo, é possível perceber que o escalonador utilizando a política FCFS não é eficiente em termos de utilização. Contudo, é uma política segura pois nenhum processo sofre de inanição, condição que ocorre quando um processo não consegue iniciar sua execução por ser sempre postergado (GOMEZ-MARTIN; VEGA-RODRIGUEZ; GONZALEZ-SANCHEZ, 2016).

Figura 6 – Escalonamento de processos com a política FCFS.



Fonte: Adaptado de Gomez-Martin, Vega-Rodriguez e Gonzalez-Sanchez (2016)

Conhecendo as vantagens e desvantagens do escalonamento com o FCFS, as estratégias de *backfilling* foram propostas. O objetivo é maximizar a utilização do sistema enquanto mantém a segurança do FCFS em relação à inanição. Com esta finalidade, mecanismos de *backfilling* buscam preencher os espaços disponíveis com processos que podem iniciar imediatamente, sem que o primeiro, ou os demais processos da fila, tenha o início de sua execução prolongada. Deste modo, alguns processos podem ser antecipados desde que não resultem na inanição de algum processo. O modo e a agressividade em que isto ocorre depende da estratégia adotada. Em geral, as mais populares, são: o *Conservative Backfilling* (CBF) e o *Extensible Argonne Scheduling sYstem Backfilling* (EASY) (ZOTKIN; KELEHER, 1999; LELONG; REIS; TRYSTRAM, 2018).

A política de escalonamento EASY foi desenvolvido para o IBM SP1, e poste-

riormente para o IBM SP2, um supercomputador do Laboratório Nacional de Argonne. Nesta política, o usuário deve fornecer uma estimação do tempo $wall_j$ em que seus processos vão utilizar os recursos requisitados. Quando o processo expira o seu $wall_j$ ele é automaticamente removido do sistema caso o seu término não seja antecipado pelo usuário. Deste modo, o escalonador consegue estimar o tempo em que os recursos estarão disponíveis, identificando lacunas que podem servir outros processos que não estejam no início da fila. Sob a condição de que os processos não atrasem a execução do primeiro processo da fila, o EASY vai desrespeitar o FCFS e permitir que processos sejam antecipados para preencher estas lacunas, reduzindo a quantidade de recursos ociosos. Processos da fila podem ser atrasados por este mecanismo. Contudo, não ocorre inanição pois em algum momento o processo será o primeiro da fila e é garantido que todos os processos vão eventualmente terminar a sua execução (MU'ALEM; FEITELSON, 2001; SRINIVASAN et al., 2002).

Algoritmo 1 EASY

```
1: for processo ∈ fila do
2:     if existe recursos para iniciar o processo then
3:         Inicie o processo;
4:         Remova o processo da fila;
5:     else
6:         retorne
7: processoPrioritario = retire o primeiro elemento da fila;
8: Reserve os próximos recursos para o processoPrioritario;
9: for processo ∈ fila do
10:    if existe recursos para iniciar o processo e a reserva do processoPrioritario não
       é violada then
11:        Inicie o processo;
12:        Remova o processo da fila;
13: Remova a reserva do processoPrioritario;
14: Adicione o processoPrioritario no início da fila;
```

Como descrito no Algoritmo 1, o EASY segue a política FCFS até que um processo da fila não tenha recursos suficientes para iniciar sua execução. Quando isto ocorre, o algoritmo vai reservar os primeiros recursos futuros que podem ser alocados para o processo prioritário e irá removê-lo da fila. No processo de reserva, o algoritmo consegue uma estimação de tempo em que o processo prioritário deve esperar para executar. Com base nesta estimação, o algoritmo começa a varrer os processos restantes na fila para identificar se algum destes pode iniciar a execução imediatamente. Caso exista um processo que possa ser iniciado e o seu tempo $wall_j$ não extrapole o início de execução do processo prioritário, caso seja utilizado um dos recursos reservados, a sua execução é iniciada imediatamente. Após todos os processos da fila serem verificados, o algoritmo então recoloca o processo prioritário no início da fila e

todo este processo é repetido (ZOTKIN; KELEHER, 1999; POQUET, 2017).

O EASY utiliza uma abordagem agressiva, postergando os processos da fila com exceção do primeiro para aumentar a utilização da infraestrutura. Tal abordagem aumenta a utilização do sistema. Contudo, dificulta a previsão do início dos processos que estão na fila. Visando esta dificuldade, a política de escalonamento CBF introduz uma abordagem conservadora que traz maior previsibilidade no escalonamento (GOMEZ-MARTIN; VEGA-RODRIGUEZ; GONZALEZ-SANCHEZ, 2016).

Diferente do EASY, no CBF nenhum processo pode atrasar a execução dos processos que estão na fila. Não restringir o atraso somente ao primeiro da fila pode diminuir as opções do escalonador, para aproveitar os recursos ociosos. Contudo, a maior previsibilidade permite que os usuários possam planejar com antecedência o início da execução dos seus processos, pois cada processo tem um tempo máximo de espera definido já no escalonamento. Não obstante, o desempenho do CBF em comparação ao EASY depende exclusivamente da carga de trabalho e da métrica avaliada. Logo, não é possível afirmar qual das políticas é melhor sem avaliar cada situação separadamente (MU'ALEM; FEITELSON, 2001).

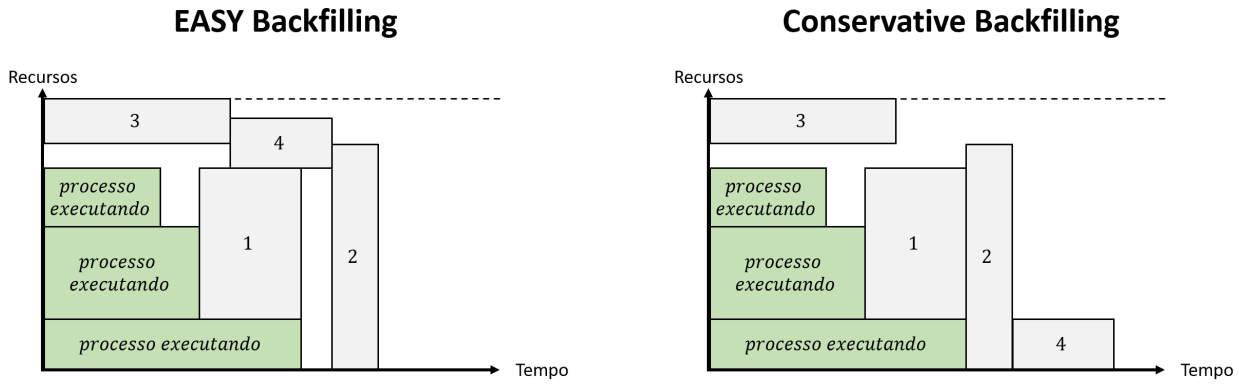
Algoritmo 2 CBF

```
1: for processo ∈ fila do
2:     recursos = busque na agenda o tempo t no qual existem recursos disponíveis para o processo;
3:     while recursos não estiverem disponíveis de t até o término processo do
4:         recursos = busque o próximo tempo t disponível;
5:     reserva = iniciando de t reserve os recursos até o término do processo;
6:     Adicione a reserva na agenda;
7:     if t == tempo atual then
8:         Inicie o processo;
```

O funcionamento do CBF está descrito no Algoritmo 2. Em cada tomada de decisão, o algoritmo vai pegar um processo da fila e buscar em sua agenda os próximos recursos que podem atender o processo durante todo o tempo requisitado. A agenda contém os intervalos de tempo em que os recursos do sistema estão reservados, semelhante a ideia de um diagrama de Gantt. Quando o algoritmo encontra este espaço, este então reserva os recursos para o processo. Caso os recursos estão disponíveis no momento da reserva, o processo é iniciado imediatamente, caso contrário o algoritmo repete este procedimento com o próximo processo da fila. Este modo de funcionamento garante que todos os processos da fila sejam executados em, no máximo, até o tempo definido no escalonamento. Caso algum recurso seja liberado antecipadamente, o escalonador pode optar por comprimir a agenda, antecipando a execução dos processos e preservando a ordem de execução, ou por refazer o processo de escalonamento (FEITELSON; Weil, 1998).

Na Figura 7 é ilustrado a diferença entre um escalonamento com o EASY e com o CBF. O cenário é o mesmo utilizado na Figura 6, contudo, o resultado difere em razão da política de escalonamento. Como é possível perceber, o EASY aumenta a utilização dos recursos atrasando o processo 2 para o processo 4 executar. Contudo, só seria possível ter uma previsão do início do processo 2 quando este for o primeiro da fila. Em contrapartida, o CBF apresenta maior previsibilidade pois a ordem definida durante o escalonamento é mantida, ou seja, o processo 2 irá, no pior caso, executar no tempo que lhe foi estimado. De qualquer modo, ambas as técnicas conseguem melhorar o desempenho do FCFS em termos de utilização (LELONG; REIS; TRYSTRAM, 2018; POQUET, 2017).

Figura 7 – Escalonamento de processos com as políticas EASY e CBF.



Fonte: Adaptado de Gomez-Martin, Vega-Rodriguez e Gonzalez-Sanchez (2016)

As políticas de escalonamento EASY e CBF utilizam informações adicionais sobre o tempo esperado de execução dos processos para determinar se é possível antecipar a sua execução. Deste modo, é possível aumentar a utilização dos recursos sem necessariamente reduzir o desempenho. Aumentar a utilização do sistema é interessante para melhor aproveitar os recursos disponíveis. Contudo, a maior quantidade de recursos ativos implica em uma maior demanda por energia, impactando os custos operacionais da plataforma. Deste modo, para ir além da otimização da eficiência em termos de utilização é necessário conhecer as variáveis que influenciam os níveis de eficiência energética de um sistema em grades.

2.4 EFICIÊNCIA ENERGÉTICA

A agregação de recursos com o objetivo de compor grades computacionais em larga escala é atrativo para a submissão de aplicações paralelas de alto desempenho. A maior disponibilidade de recursos possibilita explorar aplicações com níveis maiores de paralelismo, que pode resultar no aumento da vazão do sistema. Logo, é natural que a computação a qualquer custo predominasse na composição destes sistemas (FENG, 2005).

Este cenário começou a ser modificado na década de 2000, quando a eficiência energética passou a ser considerada uma métrica de desempenho. Foi identificado que a composição de plataformas maiores pode impactar significativamente os custos operacionais, tal qual pode limitar o seu crescimento. A maior quantidade de máquinas implica em uma maior demanda por energia. Como consequência, tem-se um aumento na geração de calor que irá demandar maior poder de resfriamento. Picos de calor podem aumentar a ocorrência de falhas nas aplicações além de poder diminuir a vida útil do *hardware*. Logo, o impacto nos custos operacionais, relacionados à aquisição, operação e manutenção de equipamentos para resfriamento, e os impactos ambientais, em função das emissões de CO2 resultantes da produção de energia, tem transformado a eficiência energética em uma tecnologia disruptiva no projeto de plataformas computacionais (FENG; CAMERON, 2007).

Com a finalidade de minimizar o consumo de energia, provedores aplicam estratégias de gerenciamento em diferentes níveis, sendo: *hardware*, aplicação e de plataforma. No nível de *hardware*, são exploradas novas arquiteturas e componentes de *hardware* mais eficientes. No nível de aplicação, estratégias trabalham no controle individual dos recursos e aplicações. No nível de plataforma, as estratégias atuam sob o sistema, no gerenciamento de um agregado de recursos. A escolha de cada estratégia depende dos objetivos dos administradores do sistema. Estratégias no nível de *hardware*, naturalmente, demandam maior capital de investimento. Enquanto estratégias no nível de aplicação e plataforma são baseadas em softwares e buscam maximizar a eficiência dos componentes atuais do sistema. Logo, é importante avaliar o custo e o impacto que cada estratégia representa (POQUET, 2017).

Em linhas gerais, a adoção de estratégias no nível de aplicação e plataforma, apresentam ser a opção que tem o maior impacto na eficiência energética de uma plataforma computacional ao menor custo de capital. Não obstante, provedores possuem maior interesse em técnicas baseadas em software com custo baixo de implementação. Soluções simples de gerenciamento que não comprometam a qualidade de serviço. Tal afirmação é respaldada pela utilização das políticas de escalonamento EASY e CBF em sistemas em produção, ambas heurísticas simples, mas com desempenho satisfatório e confiável. Por este motivo, o escopo é delimitado para as técnicas de gerenciamento de energia, como: *Dynamic Voltage and Frequency Scaling* (DVFS), *Power Capping* (PC), e *shutdown*. Estas técnicas podem trabalhar nos níveis de aplicação e de plataforma, são de interesse dos provedores e tem potencial para maximizar a eficiência energética de plataformas em produção (BATES et al., 2015).

Para melhor entender como estas técnicas trabalham, primeiro é necessário conhecer o perfil de consumo de energia padrão de um recurso computacional. Com base neste perfil, é possível identificar as fases que demandam maior consumo de

energia e entender as motivações que justificam à adoção de estratégias específicas. Portanto, é primeiro caracterizado o perfil de consumo de energia de um recurso computacional para serem apresentados técnicas que podem otimizar a eficiência energética de um sistema em grades.

2.4.1 Perfil de Consumo de Energia

Recursos computacionais possuem um perfil de consumo de energia que depende do seu estado atual. Considerando recursos como sendo máquinas de um agregado, cada recurso pode estar somente em um destes estados possíveis $S = \{computando, ocioso, off, on \rightarrow off, off \rightarrow on\}$ (ORGERIE; LEFÈVRE; GELAS, 2008). Recursos mudam de estado somente através de eventos externos, como um comando de desligamento ou uma requisição. Somente recursos no estado $ocioso$ é que podem iniciar a execução dos processos, que irá disparar um evento assim que mudar para o estado $computando$ (ASSUNÇÃO et al., 2012).

Recursos que estão ociosos podem ser desligados para economizar energia. Neste caso, os recursos passam por um estado de transição $on \rightarrow off$, que leva o tempo $t_{on \rightarrow off}$, para terminar no estado off. Para que recursos em off possam ser utilizados novamente, é necessário que sejam inicializados. Neste outro caso, os recursos passam para o estado de transição $off \rightarrow on$, que leva o tempo $t_{off \rightarrow on}$, para terminar no estado on (BENOIT et al., 2018).

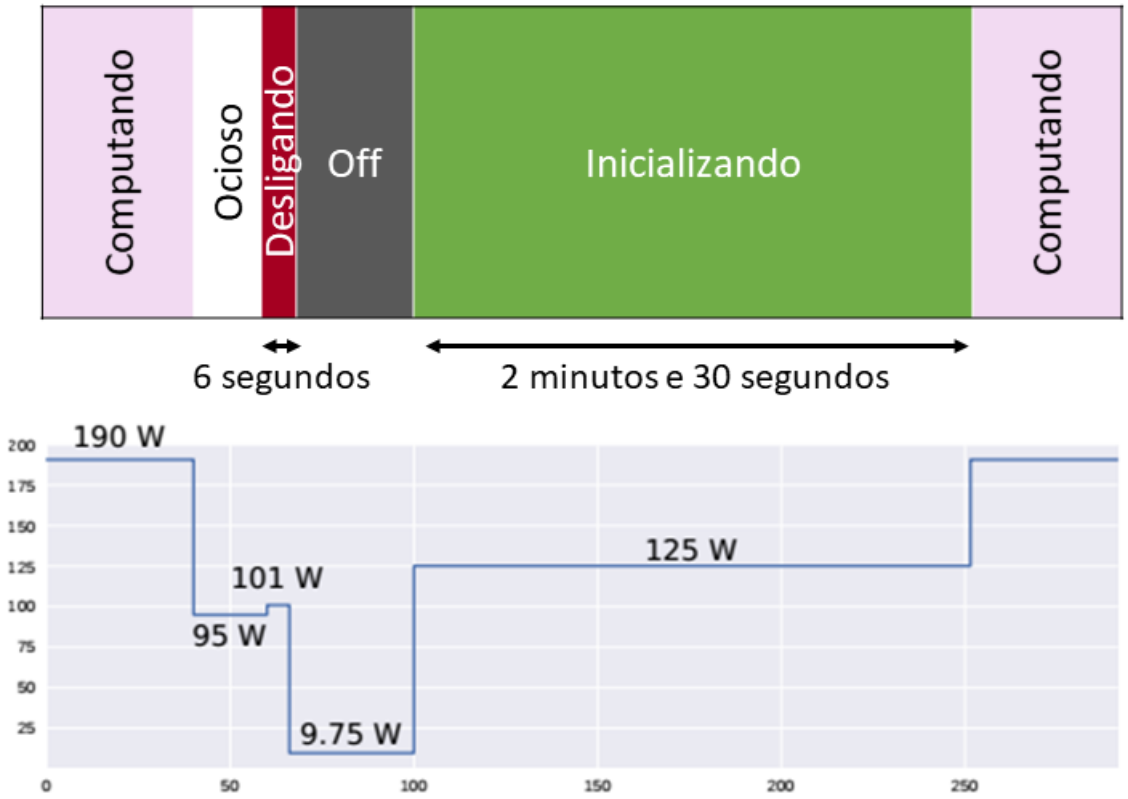
Cada estado de um recurso computacional possui um perfil de potência P, expressado em watts (W), que vai determinar o seu consumo de energia. O perfil de potência é expressado por $P = \{P_{computando}, P_{ocioso}, P_{off}, P_{on \rightarrow off}, P_{off \rightarrow on}\}$, sendo que $P(t)$ determina a potência que está sendo consumida no tempo t. Com base neste perfil, é possível estimar o consumo de energia de um recurso, expresso em joules (J), em um intervalo de tempo $[t_0, t_1]$. Formalmente, o consumo de energia pode ser determinado através da Equação (2.10) (POQUET, 2017):

$$\int_{t_0}^{t_1} P(t)dt \tag{2.10}$$

Na Figura 8 é possível visualizar um exemplo do perfil de consumo de um recurso computacional. No presente momento, o recurso está no estado computando e possui um consumo de 190 W. Após o recurso ser liberado pelo processo, este muda seu estado para ocioso, que possui um consumo de 95 W. Após um tempo, o recurso é desligado, levando 6 segundos com um consumo de 101 W. Quando desligado, o recurso possui um consumo de 9,75 W. Para iniciar a execução de um processo, o recurso é ligado novamente, o que consome 125 W por 2 minutos e 30 segundos.

Com base no perfil apresentado na Figura 8, é possível observar que a diferença de consumo entre um recurso ocioso e desligado é significativo. No entanto, o

Figura 8 – Perfil de consumo de energia de um recurso computacional.



Fonte: Adaptado de Poquet (2017)

mesmo não pode ser observado quanto ao consumo nos estados de transição (desligando e inicializando). O consumo durante as transições é superior ao consumo quando o recurso permanece ocioso. Logo, caso o recurso seja constantemente reinicializado, pode haver um aumento indesejável no consumo de energia. Por este motivo, técnicas de gerenciamento orientadas à energia devem levar em consideração o custo de suas ações para não diminuir a eficiência energética.

2.4.2 Estratégias Orientadas a Energia

Para otimizar a eficiência energética de uma plataforma computacional, o RJMS deve implementar estratégias eficientes de gerenciamento. A literatura contém um vasto acervo de trabalhos que objetivam otimizar esta métrica. Em linhas gerais, as estratégias mais comuns, adotadas no nível de aplicação e plataforma, são: DVFS, *shutdown* e PC (BATES et al., 2015).

Técnicas de DVFS objetivam a redução do consumo de energia através do controle dinâmico da frequência de operação do processador, que varia proporcionalmente à tensão de alimentação (GE; FENG; CAMERON, 2005). Deste modo, es-

tas técnicas são interessantes para explorar os tempos em que um processador está sendo pouco utilizado por uma aplicação, reduzindo a frequência para obter um menor consumo de energia. Portanto, este tipo de técnica atua no nível de aplicação, e precisa ser suportado pelo processador na camada de dispositivos de Fábrica da arquitetura (FENG, 2005).

Diminuir a frequência de operação do processador diminui o consumo de energia, contudo, sacrifica o desempenho. Ao operar em frequências menores, aplicações levam mais tempo para completar uma mesma tarefa. Tal comportamento pode, em algumas situações, introduzir o efeito oposto ao desejado, aumentando o consumo de energia (HSU; FENG, 2004). Logo, para obter sucesso no DVFS, é necessário conhecer a aplicação que está em execução ou utilizar circuitos auxiliares que permitem o monitoramento com precisão do processador (SNOWDON; RUOCCO; HEISER, 2005).

Ao invés de trabalhar no nível de aplicação, é possível reduzir o consumo através do gerenciamento dos recursos. Este é o caso das estratégias de *shutdown*, que objetivam minimizar o consumo de energia através do desligamento dinâmico dos recursos. Como apresenta o perfil de consumo da Seção 2.4.1, existe uma diferença significativa no consumo de energia entre um recurso ocioso e um desligado. Logo, é natural que o simples desligamento tenha um potencial significativo para reduzir o desperdício de energia dos recursos ociosos (HIKITA; HIRANO; NAKASHIMA, 2008; ORGERIE; LEFÈVRE; GELAS, 2008).

Técnicas baseadas no *shutdown*, buscam determinar quando os recursos devem ser desligados de modo a não prejudicar o desempenho do sistema. O consumo de energia decorrente da transição entre estados é custoso e apresenta riscos para os provedores. A adoção agressiva do *shutdown* pode diminuir o tempo de vida útil dos componentes de *hardware* no sistema, devido às sucessivas reinicializações. Não obstante, o tempo necessário para a completa transição entre os estados pode ser custoso para o desempenho do sistema, pois processos permanecem na fila enquanto seus recursos estão sendo inicializados (HERLICH; KARL, 2012).

Sobre uma perspectiva geral, o *shutdown* é a estratégia que tem maior potencial para reduzir o consumo de energia dos recursos (GEORGIOU; GLESSER; TRYSTRAM, 2015). Se os recursos não estão sendo utilizados com frequência, não faz sentido mantê-los ativos. É com base neste princípio que a técnica de *Opportunistic Shutdown* (OS), se baseia para determinar os momentos em que os recursos devem ser desligados. Caso um recurso tenha um tempo de ociosidade maior que o definido na política, ele é imediatamente desligado. Deste modo, o provedor pode variar o nível de agressividade do *shutdown*, variando o tempo de ociosidade (LAWSON; SMIRNI, 2005). Valores altos (*e.g.* 15 minutos) resultam em menos desligamentos, reduzindo

os riscos de falhas nos recursos. Em contrapartida, valores menores (*e.g.* 1 minuto) tem maior potencial em economizar energia, pois os recursos ficarão ociosos por menos tempo. Contudo, pode aumentar a quantidade de reinicializações caso o sistema tenha uma demanda frequente por recursos. Logo, o tempo de ociosidade de uma política de *timeout* deve ser definido com base na carga de trabalho do sistema, para reduzir a quantidade de transições e o seu impacto no desempenho (LIU et al., 2017)

Uma alternativa para reduzir a agressividade do *shutdown*, é utilizar a técnica *Proportional Shutdown* (PS). Neste caso, somente um percentual dos recursos é mantido desligado. Este porcentual pode ser dinâmico e se adaptar ao tipo de carga de trabalho atual no sistema. Seu conceito é baseado na ideia de *Off Reservation* (OR), na qual o algoritmo faz uma reserva de recursos, de modo similar ao processo de um consumidor, mas os mantém desligados. Caso o sistema receba uma rajada de requisições, o algoritmo pode reduzir o percentual de recursos reservados para aumentar a vazão (POQUET, 2017).

Uma terceira estratégia, consiste em realizar o gerenciamento de energia através do PC, que ocorre de modo similar ao DVFS no que diz respeito ao controle de energia do sistema. O DVFS atua diretamente no nível de aplicação enquanto o PC atua no nível de plataforma, lidando com um agregado de recursos (BATES et al., 2015). Estratégias de PC objetivam limitar a quantidade de energia que pode ser utilizada pelos recursos computacionais. Com esta finalidade, é definido um limite de consumo que não pode ser ultrapassado sobre nenhuma hipótese. Caso o consumo extrapole os limites e houver processos na fila, estes devem aguardar até que o consumo de energia volte para níveis inferiores ao limite (BORGHESI et al., 2015).

As técnicas baseadas em orçamentos dão um exemplo do funcionamento do PC. Neste caso, orçamentos são definidos contendo um limite máximo de consumo de energia permitido. Enquanto as aplicações estão executando e consumindo energia, o orçamento vai sendo reduzido, proporcionalmente, até chegar no seu limite ou até que seja renovado. Caso o orçamento chegue no seu limite, nenhum processo será executado até a sua renovação. Este *modus operandi* pode ser comparado ao modo como funciona a retirada de dinheiro de uma conta bancária. Caso não haja saldo disponível, não será possível realizar saques até que um novo depósito seja realizado. Não obstante, os saques estão limitados aos valores atuais na conta bancária (DUTOT et al., 2017).

Como observado, cada estratégia busca explorar um determinado comportamento para minimizar possíveis desperdícios de energia sobre diferentes níveis de controle. Contudo, a eficiência destas estratégias está intimamente relacionada aos tipos de cargas de trabalho no sistema e a configuração de *hardware* da plataforma. Em algumas situações, políticas orientadas a energia podem aumentar o desperdício

por não serem capazes de se adaptarem aos padrões de utilização do sistema. Deste modo, conhecer a composição do sistema e o funcionamento do RJMS implantado é fundamental para minimizar riscos e otimizar os custos operacionais com eficiência.

2.5 CONSIDERAÇÕES PARCIAIS

A computação em grades surgiu como um paradigma de computação distribuída capaz de agregar recursos computacionais de instituições geograficamente distribuídas. Desde então, sua tecnologia continua sendo otimizada através de novos mecanismos de segurança, descobrimento, descoberta e gerenciamento de recursos e processos.

Sistemas em grades têm atingindo popularidade em aplicações científicas e centros de pesquisa. Unir recursos para obter maior poder computacional é menos custoso em comparação à aquisição de novas máquinas. Algumas aplicações científicas, principalmente simulações, podem demandar tempo, exigir diferentes níveis de paralelismo, reprodutibilidade e configurações específicas. Grades computacionais atingem este objetivo fornecendo uma plataforma otimizada, reconfigurável e de larga escala. Não obstante, permitem que os recursos das instituições sejam melhor utilizados através de seu compartilhamento com outros membros.

Na literatura é possível encontrar diferentes grades computacionais, para nomear alguns: EGI, uma federação europeia entre provedores de recursos computacionais (EGI, 2019); *World Community Grid* (WCG), uma iniciativa da IBM para permitir que colaboradores compartilhem seus recursos ociosos (WCG, 2019); a XSEDE, sucessor da TeraGrid (XSEDE, 2019); e a a GRID'5000, uma grade computacional localizada na França e construída com o objetivo de providenciar uma infraestrutura de larga-escala altamente reconfigurável para o *deploy* de aplicações paralelas de alto desempenho (GRID'5000, 2019; ASSUNÇÃO et al., 2012).

Não obstante, também é possível citar alguns RJMSs mais conhecidos, como: o OAR, utilizado na GRID'5000 (CAPIT et al., 2005); SLURM, desenvolvido no Laboratório Nacional de Lawrence Livermore para gerenciar clusters Linux (YOO; JETTE; GRONDONA, 2003); *Load Sharing Facility* (LSF), desenvolvido no projeto Utopia da Universidade de Toronto e, posteriormente, adquirido pela IBM (ZHOU et al., 1993); e mais recentemente o Flux, voltado para ambientes de *High Performance Computing* (HPC) em larga escala (AHN et al., 2014).

Deste modo, é possível perceber que o tópico de grades computacionais e computação distribuída permanece ativo na comunidade científica. Contudo, determinar qual a melhor estratégia para um sistema não é uma tarefa simples. Políticas de escalonamento e técnicas de gerenciamento podem ser influenciados pelos padrões

de cargas de trabalho no sistema. No entanto, identificar estes padrões e adaptar as estratégias vigentes para corresponder a carga de trabalho atual não é uma tarefa trivial (LELONG; REIS; TRYSTRAM, 2018; ORGERIE; LEFÈVRE; GELAS, 2008). Assim, é possível observar que a elaboração de políticas adaptativas, em função da plataforma e dos padrões de carga de trabalho, parece ser o caminho a ser perseguido para otimizar a eficiência do sistema em termos de desempenho e custo operacional.

3 *REINFORCEMENT LEARNING*

Através de modelos matemáticos, *Machine Learning* (ML) introduz métodos capazes de treinarem algoritmos para resolver tarefas que abrangem, desde: encontrar padrões, aprender classificações e prever novas informações. Como parte da vasta área de ML, *Reinforcement Learning* (RL) introduz um estilo de aprendizagem peculiar. Ao invés de utilizar uma base de dados, em RL a aprendizagem ocorre através da interação com um ambiente. O ambiente modela a tarefa que deve ser realizada e define o objetivo, que serve como um guia para o agente tomar determinar as suas ações. Com as experiências coletadas durante a exploração do ambiente, o agente busca inferir quais ações melhor atingem o objetivo definido. Através deste princípio, métodos de RL foram propostos para otimizar a eficiência do treinamento dos agentes. O objetivo deste capítulo, é introduzir o conceito de RL para então serem apresentados métodos tradicionais e o estado-da-arte.

O presente capítulo está organizado em seis seções. Na Seção 3.1 é dada uma introdução geral sobre a área de RL. Na Seção 3.2 é discutido como um *Markov Decision Process* (MDP) é formulado, necessário para a aplicação de algoritmos de RL. Na Seção 3.3 são apresentados métodos clássicos que podem ser aplicados para resolver um MDP. Com base nestes métodos, o estado-da-arte em algoritmos de *Deep Reinforcement Learning* (DRL) são descritos na Seção 3.4. Na Seção 3.5 são apresentados trabalhos relacionados. Por último, na Seção 3.6 é apresentado as considerações parciais.

3.1 INTRODUÇÃO

Inspirado pelo modo como a aprendizagem biológica acontece, RL incorpora técnicas da teoria de controle para introduzir uma abordagem computacional que possibilita a aprendizagem a partir da interação com um ambiente. Teorias de aprendizagem sugerem que animais tendem a repetir um comportamento quando este é reforçado através de estímulos positivos e periódicos. RL utiliza este ideal para apresentar métodos que possam aprender de forma autônoma a resolver processos de decisão (SUTTON; BARTO, 2018).

No contexto de ML, a área de RL pode ser posicionada entre as áreas de *Supervised Learning* (SL) e *Unsupervised Learning* (UL). Abordagens de SL englobam técnicas que utilizam uma base de dados com respostas conhecidas e corretas. O objetivo destas técnicas é aprender uma função matemática que mapeie o conjunto de entrada com as saídas fornecidas, de modo que o algoritmo possa generalizar e

prever quando for apresentado novas situações. Em contrapartida, quando os dados não possuem as respostas esperadas, abordagens de UL são utilizadas. Neste caso, o objetivo é encontrar padrões estruturais a partir de características comuns entre os dados.

Similar a uma abordagem de UL, técnicas de RL também são treinados com dados sem respostas conhecidas e corretas. Contudo, um sinal de recompensa é utilizado como um indicativo da qualidade de cada tomada de decisão do algoritmo. Em comparação ao SL, abordagens de RL também aprendem a mapear as suas entradas (observações) com as saídas (ações). Contudo, o algoritmo não sabe se as ações executadas são as corretas ou as ótimas. O objetivo dos algoritmos de RL é explorar as ações disponíveis e aprender aquelas que os levem a estados satisfatórios, otimizando o objetivo definido. Deste modo, RL cobre um modo de aprendizagem específico e peculiar (LI, 2017).

Em uma abordagem de RL, todo o conhecimento é advindo através da interação com um ambiente, sendo um agente responsável por tomar decisões com base em observações para otimizar um objetivo. O objetivo é transmitido através de um sinal de recompensa, que indica a qualidade das ações executadas em cada uma das observações experimentadas. Observações encapsulam as informações sobre o estado interno do ambiente, sendo influenciadas pelas decisões do agente. Tomadas de decisão ruins ou similares vão refletir na qualidade das observações e podem restringir o comportamento do agente a um padrão específico, dificultando a exploração de caminhos alternativos que possam levar a melhores soluções. Logo, para otimizar o comportamento e não ficar preso em um ponto mínimo local, um algoritmo de RL precisa lidar com um dilema conhecido como *exploration-exploitation* (TOKIC, 2010).

O dilema *exploration-exploitation* divide as etapas de interação, entre agente e ambiente, em duas partes. A etapa de *exploration* consiste em explorar novas ações, que levam o agente ao desconhecido e incerto. Logo, nesta etapa o agente esta suscetível a seguir qualquer tipo de caminho, independente das recompensas recebidas e se aquelas ações são as melhores conhecidas. Na etapa de *exploitation*, o agente busca explorar os caminhos já conhecidos para reforçar o seu conhecimento e, assim, otimizar sua política. Deste modo, para obter novas experiências e, ao mesmo tempo, otimizar o seu comportamento, um agente deve balancear entre os atos de *exploration* e *explotation*. Contudo, equacionar este balanceamento não é uma tarefa trivial (KAELBLING; LITTMAN; MOORE, 1996; COHEN; MCCLURE; YU, 2007).

Outro problema que influencia algoritmos de RL está relacionado ao tempo em que as recompensas são fornecidas para o agente. Sinais de recompensa são valores numéricos que definem a qualidade das ações executadas pelo agente. A depender do problema, podem ser recebidas em cada tomada de decisão ou apenas em mo-

mentos de interesse. Como por exemplo, em um jogo de xadrez a recompensa pode representar o resultado da partida, sendo recebida somente ao seu final. Neste caso, ações tomadas no decorrer da partida, que potencialmente influenciaram o resultado, podem ter o mesmo peso que ações não tão sucedidas. Logo, demandará mais esforço e tempo para o agente identificar quais foram as ações que realmente influenciaram em sua vitória. Determinar como recompensar o agente é um problema complexo e conhecido como *temporal credit assignment* (OTTERLO; WIERING, 2012).

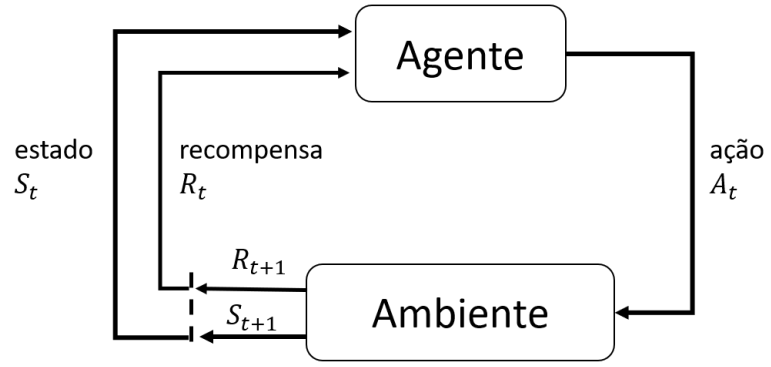
Métodos de RL utilizam uma série de estratégias para encontrar soluções para problemas de decisão e resolver cada um dos pontos levantados. Problemas de decisão são elaborados através da modelagem de uma tarefa em um *framework* conhecido como MDP, que envolve todos os elementos já citados, como: agente, ambiente, sinal de recompensa, ações e observações. Por este motivo, antes de serem descritos como os métodos de RL conseguem aprender a resolver tarefas, é necessário conhecer os elementos chaves que compõem um MDP.

3.2 *MARKOV DECISION PROCESS*

Um MDP permite que tarefas possam ser modeladas de tal modo que métodos de RL possam ser aplicados. É um *framework* que contém a formulação matemática para definir as regras que regem a interação entre o ambiente e o agente. Formalmente, um MDP pode ser definido por uma tupla (S, A, T, R), sendo: S define um conjunto finito de estados; $A \neq \emptyset$ é um conjunto finito de ações que podem ser executadas em cada estado do ambiente; $T : S \times A \times S \rightarrow [0, 1]$ é uma função de transição, que define as probabilidades do ambiente transitar para o estado $s' \in S$ quando o processo executa a ação $a \in A$ estando no estado $s \in S$; e $R : S \times A \times S \rightarrow \mathbb{R}$ é uma função de recompensa que retorna a qualidade da ação $a \in A$ quando executada no estado $s \in S$ (NG; RUSSELL et al., 2000).

A Figura 9 apresenta uma ilustração do formalismo de um MDP. Neste exemplo, o ambiente modela o problema enquanto o agente é responsável pela tomada de decisão. Quando a interação é iniciada, o ambiente fornece o seu estado atual $s \in S$ para o agente. Com base neste estado s, o agente utiliza uma política π para definir sua ação $a \in A$. A política π é responsável por mapear as observações às ações possíveis de serem executadas, sendo otimizada no decorrer do treinamento do agente. Quando a ação a_t, tomada no tempo t, é executada no ambiente, o seu estado interno é modificado seguindo a matriz de transição T. Por fim, o sinal de recompensa é calculado $R(s_t, a_t, s_{t+1})$ e fornecido para o agente no tempo $t + 1$ em conjunto com o novo estado atual s_{t+1}. Todo este processo segue até que uma condição de parada é alcançada.

Figura 9 – Dinâmica da interação em um MDP.



Fonte: Adaptado de Sutton e Barto (2018)

Seguindo este *framework*, métodos de RL buscam otimizar a política de modo a maximizar a sua recompensa acumulada esperada. Um agente que maximiza a sua recompensa imediata não, necessariamente, apresenta a melhor estratégia. Em algumas situações, ações imediatamente ruins podem levar o agente para estados que compensem o seu sacrifício, obtendo uma maior recompensa no futuro. Deste modo, ao invés de maximizar a recompensa imediata, o objetivo do agente é maximizar a recompensa acumulada esperada (chamada de retorno). A função de retorno, no seu modelo mais simples, é apresentada na Equação (3.1) (SUTTON; BARTO, 2018).

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{3.1}$$

Sendo, T o último passo dado no ambiente.

A Equação (3.1) é perfeitamente aplicável em problemas que tenham um limite na quantidade de interações entre agente e ambiente. Este é o caso quando o problema pode ser dividido em episódios. Cada episódio possui um estado terminal, sendo o ambiente reiniciado quando o agente visita este estado. Um exemplo deste caso seria o jogo de xadrez, quando o jogo acaba todo o processo tem de ser reiniciado ao seu estado inicial. Quando esta condição ocorre, é dito que o MDP possui um horizonte finito. Contudo, existem algumas situações em que este limite não é perfeitamente alcançável. Isto ocorre quando não existe uma possibilidade de parada, ou seja, o agente permanece em interação contínua. Nesta situação, é dito que o MDP possui um horizonte infinito e a Equação (3.1) não pode ser aplicada pois não irá convergir.

Para lidar com problemas de horizonte infinito, uma abordagem comum é adequar a Equação (3.1) adicionando um parâmetro de desconto γ. Esta adequação é dada na Equação (3.2), sendo $\gamma = [0, 1)$. Ao inserir este limite, uma série infinita converge desde que $\gamma < 1$. Não obstante, este parâmetro permite ponderar a importância entre recompensas futuras e as mais recentes. Ao determinar $\gamma = 0$, o agente irá ma-

ximizar a sua recompensa imediata $G_t = R_{t+1}$ e recompensas futuras são ignoradas. Ao utilizar γ com valores próximos de 1, as recompensas futuras terão influência no retorno, pois o peso de uma recompensa recebida no passo k corresponde à γ^{k-1}. Em todos os casos, ao utilizar o desconto, é dito que objetivo do agente é maximizar o retorno descontado esperado (BUŞONIU; BABUŠKA; SCHUTTER, 2010; OTTERLO; WIERING, 2012).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.2}$$

Para maximizar o retorno descontado esperado, o agente otimiza a sua política π usando as experiências coletadas durante a interação com o ambiente. Políticas determinam o comportamento do agente sobre as situações apresentadas e podem ser definidos como sendo uma função $\pi : S \rightarrow A$ que mapeia cada estado $s \in S$ a uma ação $a \in A(s)$. Métodos de RL divergem na estratégia adotada para otimizar a política. Uma abordagem comum consiste em utilizar uma estimativa do valor de cada estado visitado para avaliar e determinar quais ações devem ser reforçadas. Com esta finalidade, uma função valor $v_\pi(s)$ é definida para estimar a quantidade de recompensa que o agente pode esperar receber se este está no estado s e seguir a política π. Deste modo, é possível avaliar se a política π é melhor que a política π', sendo $\pi \geq \pi'$ somente se $v_\pi(s_0) \geq v_{\pi'}(s_0)$. Formalmente, a função valor está definida na Equação (3.3) (GOUBERMAN; SIEGLE, 2014).

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s], \forall s \in S \tag{3.3}$$

Do mesmo modo que é possível comparar duas políticas pelos valores de seus estados iniciais, é possível levar em consideração as ações executadas. Neste caso, é definida uma função valor de estado-ação que objetiva estimar a quantidade de recompensa que um agente pode esperar receber se este estiver em um estado $s \in S$ e tomar a ação $a \in A(s)$ seguindo a política π. Formalmente, esta definição é fornecida na Equação (3.4) (SEWAK, 2019).

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{3.4}$$

Como é possível observar, ambas funções valor apresentadas são estimadas com base nas experiências coletadas pelo agente, seguindo uma política π, no decorrer de sua interação com o ambiente. Logo, a eficiência e precisão com que estas estimações são realizadas vai impactar diretamente na capacidade de aprendizagem do algoritmo. Caso o algoritmo consiga estimar o valor ótimo de cada estado ou par estado-ação, então, a política ótima consiste em seguir uma abordagem gulosa. Contudo, estimar este valor pode não ser trivial em alguns problemas. Para exemplificar

estes cenários, na Seção 3.3 é apresentado alguns métodos que utilizam a Equação (3.4) para tentar encontrar uma política ótima. Não obstante, uma segunda abordagem que otimiza uma política diretamente, sem utilizar estas estimações, também é apresentada. Ambas as estratégias servem como base para a elaboração de algoritmos conhecidos como *actor-critic*, utilizado pelos métodos considerados o estado-da-arte em DRL.

3.3 MÉTODOS TRADICIONAIS

Sobre uma perspectiva geral, o objetivo dos métodos de RL é aprender a mapear cada situação à uma ação, de modo a maximizar a sua recompensa acumulada esperada. Nesta seção, alguns métodos tradicionais de RL são introduzidos. Tais métodos podem ser classificados de acordo com as estratégias adotadas para otimizar a política do agente. Deste modo, antes de serem discutidos os métodos tradicionais, é apresentado uma possível taxonomia.

3.3.1 Taxonomia

Cada método de RL difere no modo em como é realizado a otimização do comportamento do agente. Logo, é natural que possam ser classificados de acordo com a estratégia adotada. De modo geral, três aspectos podem ser analisados durante a classificação de um método de RL, sendo (SUTTON; BARTO, 2018):

- ***Model-free*** ou ***model-based***: Métodos que seguem uma abordagem *model-based* utilizam um modelo do MDP para estimar a política ótima. Em contrapartida, em uma abordagem *model-free*, a aprendizagem ocorre somente através das experiências coletadas. Logo, o agente não tem conhecimento sobre como o ambiente pode ser influenciado pela ação executada.

- ***Value-based*** ou ***policy-based***: Métodos classificados como *value-based* buscam encontrar a política ótima através da estimação do valor de cada estado ou par estado-ação. Abordagens do tipo *policy-based* buscam estimar diretamente a política ótima ao invés de usar a função valor. Neste caso, a política é parametrizada e seu vetor de parâmetros é otimizado de modo a maximizar a recompensa acumulada.

- ***On-policy*** ou ***off-policy***: No caso de um método que segue uma abordagem *on-policy*, a mesma política utilizada para determinar o comportamento do agente, em respeito a tomada de decisão, é utilizada durante a otimização, na estimação da função valor ou na atualização dos parâmetros da política. Enquanto na

abordagem *off-policy*, uma política é utilizada para determinar o comportamento e outra é utilizada na otimização (LAPAN, 2018, p. 139).

É possível observar que métodos de RL podem discernir drasticamente em como o agente é treinado com base nos estados visitados. Em alguns problemas, métodos *value-based* podem ser mais eficientes que métodos *policy-based* por ser mais simples estimar o valor de cada estado ao invés de otimizar a política diretamente. Contudo, existem problemas em que a política ótima pode ser probabilística. Neste caso, uma estratégia *policy-based* pode ser mais adequada por permitir a parametrização da política de modo a obter uma distribuição de probabilidades sobre as ações possíveis de serem executadas. Logo, para avaliar qual método é o mais indicado ao problema em questão é necessário conhecer as diferentes estratégias que podem ser classificadas de acordo com a taxonomia apresentada.

3.3.2 *Temporal-Difference Learning*

Temporal-Difference (TD) *learning* mescla características dos métodos baseados em programação dinâmica, otimizando a função valor com base em outras estimações, e dos métodos de Monte Carlo, aprendendo com as experiências advindas da interação com o ambiente. Deste modo, este tipo de estratégia possibilita que a aprendizagem ocorra a cada tomada de decisão e não dependa exclusivamente de um modelo do MDP (SEWAK, 2019).

A versão mais simples dos métodos de TD é conhecido como *one-step* TD, ou simplesmente TD(0). Neste método, as estimações da função valor são atualizadas imediatamente após a tomada de decisão, justificando o seu nome "*one-step*". Com tal finalidade, o TD(0) estima tanto o valor do estado atual como o valor do próximo estado para atualizar a função valor, técnica conhecido como *bootstrap*. Na Equação (3.5) é definido como o TD(0) atualiza a função valor.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.5}$$

Sendo, $V(S_t)$ a estimação do valor do estado atual; α a taxa de aprendizagem; R_{t+1} a recompensa recebida após a ação a_t ser executada no ambiente; γ o desconto; e $V(S_{t+1})$ a estimação do valor do próximo estado.

Ao analisar a Equação (3.5), é possível perceber que $TD_{target} = R_{t+1} + \gamma V(S_{t+1})$ pode ser interpretado como sendo o objetivo, o valor esperado do estado $V(S_t)$. Deste modo, $TD_{error} = [TD_{target} - V(S_t)]$ pode ser interpretado como sendo um resíduo. Como o agente não tem conhecimento sobre o quanto de recompensa futura é possível receber, este precisa explorar todas as alternativas para descobrir os valores ótimos de $V(S_t)$. Por este motivo é utilizado uma taxa de aprendizagem α. Possivelmente, os valores iniciais de $V(S_t)$ não representarão os seus valores reais. Logo, o

parâmetro α permite controlar a quantidade de atualizações que precisam ser realizadas para aproximar $V(S_t)$ de TD_{target}.

A partir da Equação (3.5), também é possível derivar uma função valor para estimar o valor de cada par estado-ação. Formalmente, esta adequação é definida na Equação (3.6) e utiliza uma tupla de cinco eventos $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, que dá nome ao método SARSA descrito no Algoritmo 3 (SUTTON; BARTO, 2018).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3.6}$$

Seguindo a taxonomia definida na Seção 3.3.1, SARSA é um método que pode ser classificado como *on-policy*, *model-free* e *value-based*. Tal classificação pode ser inferida a partir do Algoritmo 3. No início do treinamento, uma matriz $Q(s, a)$ é inicializada para todos os valores de $s \in S$ e $a \in A$. Depois, para cada episódio, é inicializado o estado inicial S e, utilizando um política π, é determinado uma ação A. Esta ação é executada no ambiente e o agente recebe uma recompensa R e um novo estado S'. Com base neste novo estado S', o agente determina sua próxima ação A' utilizando a mesma política π. A função valor é então atualizada com base na recompensa obtida e na estimação do retorno do próximo estado S'. Logo, o SARSA não necessita de um modelo do MDP (*model-free*), utiliza a mesma política para tomar decisões e atualizar a função valor (*on-policy*) e busca estimar o valor dos pares de estado-ação para encontrar a política ótima (*value-based*).

Algoritmo 3 SARSA

```
1: Inicialize Q(s, a) para todo s ∈ S e a ∈ A
2: for each episodio do
3:     Inicialize S
4:     Escolha A usando uma política π sobre S;
5:     for each passo do
6:         Execute a decisão A;
7:         Observe R e S';
8:         Escolha A' usando uma política π sobre S';
9:         Q(S, A) ← Q(S, A) + α[R + γQ(S', A') − Q(S, A)];
10:        S ← S'; A ← A';
11:        if S é um estado terminal then
12:            break;
```

Diferente do SARSA, uma segunda abordagem que pode ser adotada utiliza somente uma tupla de quatro elementos $(S_t, A_t, R_{t+1}, S_{t+1})$. Enquanto o SARSA decide suas ações (A e A') utilizando a mesma política π, esta abordagem pode utilizar uma política $greedy$ na atualização da função valor estado-ação e uma política π para tomar decisões. Logo, pode ser classificada como *off-policy*, *model-free* e *value-based*. Sua implementação é conhecida como *Q-learning* e está definida no Algoritmo 4 (WATKINS; DAYAN, 1992).

Algoritmo 4 *Q-learning*

Inicialize $Q(s, a)$ para todo $s \in S$ e $a \in A$
for each $episodio$ **do**
    Inicialize S
    for each $passo$ **do**
        Escolha A usando uma política π sobre S;
        Execute a decisão A;
        Observe R e S';
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$;
        $S \leftarrow S'$;
        if S é um estado terminal **then**
            $break$;

A principal diferença entre o Q-learning e o SARSA está na atualização da função valor. O Q-learning, utiliza uma política *greedy* para determinar a estimação do valor do próximo par estado-ação. Logo, os retornos dos pares estado-ação são atualizados com base no maior retorno estimado do próximo par estado-ação. Este detalhe pode diminuir o tempo necessário para o algoritmo convergir, contudo, não é possível generalizar pois depende do problema e em como este foi modelado em um MDP. Demais detalhes de implementação permanecem os mesmos, sendo a política ϵ-*greedy* utilizada para tomar decisões e lidar com o dilema *exploitation-exploration* (ABDALLAH; KAISERS, 2016; OTTERLO; WIERING, 2012).

Sobre uma perspectiva geral, métodos de TD utilizam estimações das recompensas esperadas para cada estado observado pelo agente no decorrer de seu treinamento. Tais estimações são otimizadas de acordo com a quantidade de informações obtidas durante a interação com o ambiente. Deste modo, o balanceamento entre *exploration-exploitation* é crucial para que o agente encontre os valores ótimos para cada estado ou par estado-ação. Contudo, existem problemas em que obter uma estimativa do valor dos estados não é um processo trivial. Isto pode ocorrer quando a política ótima não é determinística e um estado pode conter duas ou mais ações ótimas. Neste caso, buscar diretamente a política pode apresentar ser mais eficiente em comparação a estimar o valor de cada estado. Estratégia que é adotada nos métodos de gradiente da política.

3.3.3 Métodos de Gradiente da Política

Métodos de gradiente da política divergem dos métodos de TD por não necessitarem consultar uma função valor para otimizarem suas políticas. Neste caso, a política é parametrizada π_θ e o seu vetor de parâmetros θ é atualizado de modo a maximizar o retorno descontado esperado. Portanto, métodos de gradiente da política calculam o gradiente da política para determinar a direção em que os parâmetros de-

vem ser atualizados. Logo, a política escolhida deve ser diferenciável (SEWAK, 2019).

Seguindo este tipo de estratégia, é possível parametrizar a política de distintas maneiras. Uma abordagem tradicional, consiste em utilizar preferências numéricas para cada par estado-ação, dando forma a uma distribuição de probabilidades. Ações com maior preferência possuem uma maior probabilidade de serem escolhidas em cada tomada de decisão e o objetivo do agente é aumentar a preferência das ações que retornam o maior retorno esperado (SUTTON; BARTO, 2018). Assim, a política pode ser otimizada reforçando a probabilidade de uma ação A_t ser repetida no estado S_t de modo proporcional ao retorno esperado G_t e ao gradiente da política $\pi(A_t|S_t, \theta_t)$. A implementação desta abordagem é conhecida como REINFORCE e pode ser classificado como *on-policy*, *model-free* e *policy-based* (WILLIAMS, 1992). No Algoritmo 5 é apresentada uma possível implementação do REINFORCE.

Algoritmo 5 REINFORCE

1: Inicialize os parâmetros da política $\theta \in \mathbb{R}^{d'}$
2: **for each** $episodio$ **do**
3: Interaja por um episódio utilizando $\pi(\cdot|\cdot, \theta)$
4: **for** $t \in episodio$ **do**
5: $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
6: $\theta \leftarrow \theta + \alpha\gamma^t G \frac{\bigtriangledown\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$;

O método REINFORCE utiliza o retorno esperado em conjunto com o gradiente da probabilidade de executar a ação escolhida, normalizado pela sua probabilidade de ser escolhida, para atualizar os parâmetros da política. Deste modo, a probabilidade de as ações serem repetidas aumenta ou diminuí conforme o retorno obtido. Um segundo ponto sobre o método REINFORCE, está no fato de este precisar interagir sobre um episódio inteiro antes de atualizar sua política. Isto ocorre, pois, este método não utiliza a estratégia de *bootstrap*, observada no *Q-Learning* e SARSA, sendo necessário esperar o fim do episódio para poder estimar o retorno G. Tal abordagem, classifica este método como sendo um algoritmo de Monte Carlo (SUTTON et al., 2000).

Outro detalhe sobre o REINFORCE está no modo como o dilema *explotation-exploration* é conduzido. Como este método utiliza uma política probabilística, o dilema de *explotation-exploration* é tratado naturalmente. Isso ocorre pois, no início do treinamento, a política segue uma distribuição uniforme e todas as ações possuem a mesma probabilidade de serem escolhidas. Na medida com que a política é otimizada, estas probabilidades vão sendo modificadas de modo a aumentar as probabilidades de as melhores ações serem repetidas. Logo, desde que as atualizações sejam suaves e a política não se torne determinística, haverá uma chance de o agente realizar a fase de *exploration* em detrimento da fase de *exploitation* (LAPAN, 2018, p. 351).

Uma segunda estratégia baseada no gradiente da política, consiste em realizar o *bootstrap* utilizando a função valor. Diferente do REINFORCE, ao utilizar a função valor não é necessário esperar até o fim do episódio para atualizar a política. Deste modo, o algoritmo pode aprender em qualquer momento durante a interação e ser aplicado em problemas que não possuem um estado final alcançável. Técnicas que seguem esta abordagem são chamados de *actor-critic*, sendo o *actor* responsável por seguir a política e o *critic* responsável por avaliar as ações do *actor* através da função valor (KONDA; TSITSIKLIS, 2000).

Algoritmo 6 *One-step Actor-Critic*

Inicialize os parâmetros da política $\theta \in \mathbb{R}^{d'}$
Inicialize os pesos da função valor $\omega \in \mathbb{R}^{d'}$
for each $episodio$ **do**
    Inicialize S
    $I \leftarrow 1$
    while S não é terminal **do**
        Escolha A usando a política $\pi(\cdot|S, \theta)$;
        Execute a decisão A;
        Observe R e S';
        if S' é terminal **then**
            $\delta \leftarrow R - \hat{v}(S, \omega)$;
        else
            $\delta \leftarrow R + \gamma\hat{v}(S', \omega) - \hat{v}(S, \omega)$;
        $\omega \leftarrow \omega + \alpha^{\omega} I\delta \bigtriangledown \hat{v}(S, \omega)$;
        $\theta \leftarrow \theta + \alpha^{\theta} I\delta \bigtriangledown \ln \pi(A|S, \theta)$;
        $I \leftarrow \gamma I$;
        $S \leftarrow S'$;

Um exemplo inspirado no TD(0) e que representa a ideia dos métodos *actor-critic* é apresentado no Algoritmo 6. Neste caso, o *critic*, com o vetor de parâmetros ω, e o(s) *actor(s)*, com o vetor de parâmetros θ, são atualizados a cada tomada de decisão. Como esta abordagem não precisa esperar até o fim do episódio, de um modo geral, a aprendizagem tende a ser mais rápida. Não obstante, ambas as funções do *critic* e *actor* podem ser otimizadas paralelamente e em tempos distintos. Algoritmos que seguem esta estratégia podem ser classificados como sendo ambos *policy-based*, representado pelo *actor*, e *value-based*, representado pelo *critic* (SUTTON; BARTO, 2018).

A concepção de métodos de gradiente da política possibilitou a construção de política probabilísticas que podem encontrar políticas ótimas em problemas onde existem mais de uma ação ótima em alguns estados. Não obstante, a parametrização da política pode tomar forma de maneiras distintas, desde que seja diferenciável. Logo, técnicas de *Deep Learning* (DL) podem ser aplicadas e utilizadas conjuntamente com

os métodos tradicionais de RL. Como consequência, é possível resolver problemas complexos através de aproximações. Contudo, a adequação de um algoritmo de DL no contexto de RL levanta uma série de desafios que podem influenciar a eficiência da aprendizagem do agente. Desafios que são tratados através dos métodos conhecidos como DRL.

3.4 *DEEP REINFORCEMENT LEARNING*

Um dos principais problemas dos métodos tradicionais de RL, como SARSA e *Q-learning*, é a falta de escalabilidade. Em problemas complexos, que possuem um conjunto extenso de estados/ações, pode não ser possível encontrar a política ótima com estes métodos. Isto ocorrer pois estes métodos são abordagens tabulares, sendo as estimativas da função valor armazenadas em matrizes. Logo, se faz necessário realizar um trabalho manual na redução do escopo do trabalho ou buscar alternativas com aproximações (ARULKUMARAN et al., 2017; ORHEAN; POP; RAICU, 2018).

É neste contexto que técnicas de DL podem ser consideradas uma alternativa. Ao invés de utilizar matrizes, modelos de DL podem ser aplicados para aproximar uma função para estimar o valor dos estados, ou pares de estado-ação, em uma estratégia *value-based*. Do mesmo modo que, podem ser utilizados para representar a política em uma estratégia *policy-based*. Assim, é possível lidar com problemas complexos buscando a generalização, utilizando modelos de DL em conjunto com os métodos de RL, dando início aos métodos de DRL (MNIH et al., 2015; LI, 2017).

O objetivo desta seção é apresentar como os métodos clássicos de RL são estendidos para o contexto de DRL. A simples adoção de métodos de DL em um algoritmo de RL não garante uma aprendizagem eficiente. Logo, também é descrito as técnicas utilizadas para estabilizar o processo de treinamento e permitir que DL seja aplicado de modo eficiente.

3.4.1 *Deep Q-Network*

A aplicação de DL na estimação da função valor pode parecer simples. Como entrada, é utilizado as observações do ambiente e, como saída, tem-se o retorno descontado. Logo, seria o caso de coletar experiências para então treinar o método de DL a mapear as observações para os retornos estimados. Desde que as observações sejam independentes e distribuídas uniformemente, o modelo vai aproximar uma função que generalize para novas observações. Este caso se aplica aos problemas de SL, quando é conhecido as respostas corretas. Contudo, em problemas de RL, a aprendizagem ocorre online sendo as observações obtidas somente através da interação do agente com o ambiente.

Neste contexto, a aplicação de DL em RL levanta um desafio adicional. As experiências são coletadas sequencialmente, na medida com que a interação ocorre, e, portanto, não são independentes. Não obstante, novas observações são um reflexo da política do agente e das observações anteriores. Logo, a distribuição dos dados possivelmente não será uniforme (LAPAN, 2018, p. 205).

Outro detalhe que aumenta a complexidade em utilizar DL em problemas de RL, está relacionado ao modo como a função valor é otimizada para refletir o retorno esperado. Por exemplo, considerando o Algoritmo 4 que descreve a implementação do *Q-learning*, a função valor é atualizada por:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3.7}$$

Sendo, $Q(S_t, A_t)$ o retorno esperado quando a ação A_t é executada no estado S_t no tempo t; R_{t+1} a recompensa; γ o parâmetro de desconto; $\max_a Q(S_{t+1}, a)$ o maior retorno esperado para o próximo estado S_{t+1}; e α a taxa de aprendizagem.

Observando a Equação (3.7), é possível perceber que a estimação do valor do par estado-ação atual é atualizada utilizando uma estimação do retorno esperado para o próximo estado. Como o método Q-learning utiliza uma matriz para armazenar estas estimações, não haverá problema ao atualizar a política pois os dados são persistidos. Contudo, ao serem utilizados métodos de DL, sendo esta matriz substituída por um conjunto de parâmetros, a estimação de $Q(S_{t+1}, \cdot)$ pode ser influenciada conforme $Q(S_t, A_t)$ for sendo aproximado durante o treinamento. Isto ocorre pois ambas as estimações são realizadas pelo mesmo modelo, utilizando o mesmo conjunto de parâmetros. Logo, ao atualizar os parâmetros, novas estimações são obtidas para todos os pares de estado-ação. Como consequência, o treinamento não será estável pois toda vez que o modelo vai atualizar o vetor de parâmetros sua saída esperada é influenciada, gerando divergências (SUTTON; BARTO, 2018).

Para lidar com estes desafios, o método DQN introduz um conjunto de técnicas que possibilitam o treinamento estável de um modelo de DL em um algoritmo como o *Q-learning*. Para que as observações sejam independentes e distribuídas uniformemente, todas as transições experimentadas pelo agente, na interação com o ambiente, são armazenadas em um *buffer* chamado de *experience replay*. Assim, durante a otimização do modelo os dados são extraídos deste *buffer* seguindo uma distribuição uniforme. Deste modo, as observações possivelmente não serão sequenciais e não estarão correlacionadas, minimizando uma das fontes de instabilidade (SEWAK, 2019).

Para estabilizar o processo de treinamento, a segunda estratégia consiste em travar por um intervalo de tempo os pesos utilizados para estimar os retornos esperados dos próximos estados. Com esta finalidade, uma cópia dos parâmetros da função

Algoritmo 7 DQN

Inicialize um *buffer* $\triangle$ com capacidade N
Inicialize os pesos $\theta \in \mathbb{R}^{d'}$ da função ação-valor Q
Inicialize os pesos do alvo $\hat{Q}$ da função ação-valor com $\theta^{-} = \theta$
for each $episodio$ **do**
    Inicialize S
    for each $passo$ **do**
        Com probabilidade ϵ escolha uma ação A aleatória
        Caso contrário, $A \leftarrow \max_a Q(S, a, \theta)$
        Execute a ação A e observe R e S'
        Adicione a transição (S, A, R, S') em $\triangle$
        Pegue uma $amostra$ aleatória das transições armazenadas em $\triangle$
        for $(S_j, A_j, R_{j+1}, S_{j+1}) \in amostra$ **do**
            if S_{j+1} é terminal **then**
                $y_j \leftarrow R_{j+1}$
            else
                $y_j \leftarrow R_{j+1} + \gamma \max_a \hat{Q}(S_{j+1}, a, \theta^{-})$
            $\theta_{t+1} \leftarrow \theta_t + \alpha[y_j - Q(S_j, A_j, \theta_t)] \bigtriangledown Q(S_j, A_j, \theta_t)$
        A cada N passos, $\theta^{-} \leftarrow \theta$
        $S \leftarrow S'$

valor, chamado de *target network*, é mantida em paralelo. Toda vez que a função valor vai ser atualizada, a *target network* estima os retornos dos próximos estados mantendo os seus parâmetros constantes. Após um intervalo de tempo, os parâmetros da *target network* são então atualizados para refletir o conhecimento obtido. Deste modo, o alvo da função valor não varia após cada otimização, minimizando o segundo fator de instabilidade (MNIH et al., 2015).

Uma possível implementação para o método DQN pode ser visualizada no Algoritmo 7. Neste exemplo, as ações são escolhidas seguindo uma política ϵ-*greedy*, como observado no Q-learning mas, utilizando as estimações dada pelo modelo de DL. Demais diferenças implementam as estratégias propostas pelo DQN e que foram levantadas. Assim como o *Q-learning*, este algoritmo pode ser classificado como *off-policy*, *model-free* e *value-based*.

3.4.2 *Asynchronous Advantage Actor-Critic*

O *Asynchronous Advantage Actor-Critic* (A3C) é um método que segue o estilo *actor-critic* e, portanto, utiliza o gradiente da política para otimizar o seu comportamento. Contudo, diferente do método REINFORCE apresentado no Algoritmo 5, este método também utiliza uma função valor para realizar o *bootstrap* e estimar as vantagens das ações executadas pela política. Deste modo, pode ser classificado como *on-policy*, *model-free*, *value-based* e *policy-based*.

Um detalhe específico do A3C, está no modo em como o *critic* é utilizado para avaliar as ações dos *actors*. Neste método, as estimações do *critic* são utilizadas para verificar qual foi a vantagem das ações executadas. Deste modo, cada atualização no vetor de parâmetros da política é realizada, conforme (MNIH et al., 2016):

$$\theta_{t+1} = \theta_t + \alpha \hat{A}(s_t, a_t; \omega_t) \bigtriangledown_\theta \log \pi(a_t|s_t; \theta_t) \tag{3.8}$$

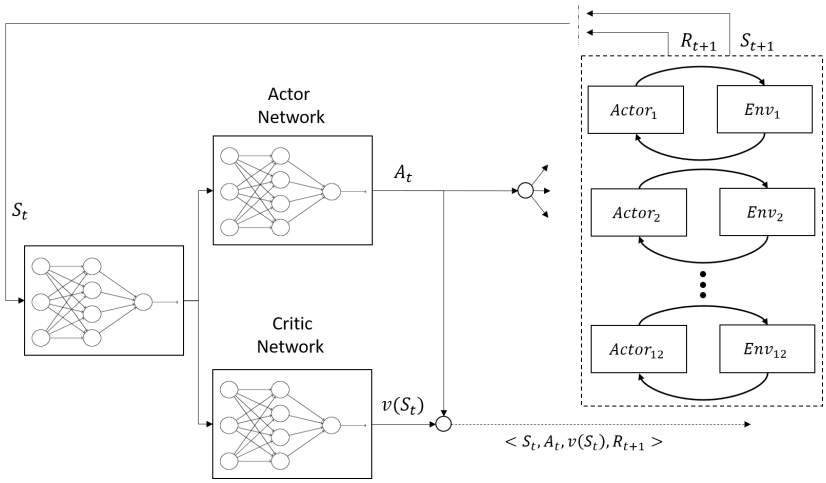
Sendo, θ_t os pesos da política no tempo t, ω os pesos da função valor e $\hat{A}(s_t, a_t; \omega_t)$ uma estimação da vantagem que pode ser obtido, por:

$$G_{t:t+n} - \hat{V}(s_t; \omega_t) \tag{3.9}$$

Sendo, $G_{t:t+n}$ a estimação do retorno truncado após n passos, chamado na literatura de *n-step return*, e $\hat{V}(s_t; \omega_t)$ a estimação do valor do estado atual (SUTTON; BARTO, 2018).

Em comparação ao DQN, o A3C não armazena as transições em uma memória pois, de um modo geral, suas implementações seguem uma abordagem *on-policy*. Deste modo, para lidar com a possível correlação nas observações, múltiplos *actors* são instanciados em paralelo, cada qual com uma instância própria do ambiente. Como resultado, cada *actor* vai coletar experiências possivelmente distintas ao mesmo tempo em que o desempenho da aprendizagem é potencializado pelo paralelismo. Contudo, este método não reutiliza os dados coletados. Após cada otimização, os dados são descartados e os *actors* coletam novas experiências utilizando a sua nova política (WANG et al., 2016).

Figura 10 – Dinâmica da interação de um algoritmo *actor-critic*.



Fonte: Adaptado de Sewak (2019)

Na Figura 10 é ilustrado um exemplo da dinâmica de interação de um algoritmo *actor-critic*, sendo a função valor e a política representados por uma *Deep Neural Network* (DNN) que possui algumas camadas ocultas compartilhadas para lidar

com as observações do ambiente. Este exemplo apresenta uma versão simplificada do método A3C, chamado de *Advantage Actor-Critic* (A2C). No A2C as atualizações nos parâmetros da política ocorrem de modo sincronizado enquanto no A3C este procedimento ocorre de modo dessincronizado. Como é possível observar no exemplo, múltiplos *actors* interagem com uma instância própria do ambiente e utilizam uma política parametrizada para determinar suas tomadas de decisão. Em paralelo, o *critic* estima a vantagem das ações executadas em cada um dos estados observados. Após n tomadas de decisão, a Equação 3.8 pode ser calculada e os pesos da política e da função valor atualizados de acordo (SEWAK, 2019).

3.4.3 *Proximal Policy Optimization*

A falta de eficiência em lidar com as experiências coletadas por um agente utilizando métodos de gradiente da política, como o REINFORCE e o A3C, é uma das justificativas para a proposta do método *Proximal Policy Optimization* (PPO). Até então, as trajetórias coletadas eram utilizadas uma única vez durante a otimização da política. A justificativa para não reutilizar as experiências está relacionado à função objetivo, utilizada para atualizar os parâmetros da política. Como observado na Equação (3.8), o vetor de parâmetros da política é atualizado proporcionalmente às vantagens estimadas das ações que foram executadas no ambiente. Neste caso, se a política for atualizada repetidas vezes com as mesmas experiências, a probabilidade de a mesma sequência de ações ser repetida pode aumentar ou diminuir na mesma proporção. Logo, pode ocorrer um sobre-ajuste na política, reduzindo a probabilidade de o agente explorar novos caminhos e encontrar a política ótima.

Para minimizar este possível sobre-ajuste, o PPO introduz uma função objetivo que possibilita reutilizar as experiências na atualização dos parâmetros. Formalmente, esta função é definida, como (SCHULMAN et al., 2017):

$$\hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{3.10}$$

Sendo, ϵ um parâmetro que delimita os valores de r_t, $\hat{A}_t$ a função vantagem e $r_t(\theta)$ retorna a razão entre a probabilidade de uma ação ser executada com os parâmetros atuais θ e a sua probabilidade de ser executada com os parâmetros antigos θ_{old}, que foram os utilizados na coleta de experiências. Formalmente, $r_t(\theta)$ é definida na Equação 3.11.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3.11}$$

Ao interpretar a Equação (3.11), é possível perceber que $r(\theta_{old}) = 1$. Logo, ao delimitar os valores de r_t, atualizações na política que aumentem demasiadamente a probabilidade de uma ação ser repetida, movendo r_t para fora do intervalo $[1-\epsilon, 1+\epsilon]$,

são desincentivadas. Como consequência, é possível utilizar as mesmas experiências coletadas durante a otimização sem, necessariamente, destruir a política (SCHULMAN et al., 2017).

Uma possível implementação do método PPO pode ser visualizado no Algoritmo 8. Assim como no método A3C, o PPO segue o estilo dos métodos *actor-critic* ilustrado na Figura 10. A principal diferença entre estes dois métodos está na utilização da função objetivo, demais detalhes permanecem. Na sua versão original, a função vantagem $\hat{A}$ utilizada é a *Generalized Advantage Estimation* (GAE):

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{3.12}$$

Sendo, $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ o resíduo da diferença temporal como já observado na Equação (3.5), γ o parâmetro de desconto e λ o parâmetro GAE, responsável por controlar o *trade-off* entre o viés e a variância (SCHULMAN et al., 2015).

Algoritmo 8 PPO

1: Inicialize os parâmetros da política $\theta \in \mathbb{R}^{d'}$
2: Inicialize os pesos da função valor $\omega \in \mathbb{R}^{d'}$
3: **for each** $iter$ **do**
4: **for each** $actor$ **do**
5: Colete T experiências usando a política $\pi_{\theta_{old}}$
6: Compute $\hat{A}_1, ...\hat{A}_T$
7: Otimize a política com a função objetivo por K épocas
8: $\theta_{old} \leftarrow \theta$

Ao permitir que múltiplas épocas de treinamento possam ser realizadas com as mesmas experiências coletadas, a eficiência na aprendizagem é intensificada e o algoritmo pode convergir em menor tempo. Como foi observado na descrição do Q-learning (na Seção 3.3.2), a estimação da função valor é aproximada do TD_{target} em pequenos passos que são controlados pela taxa de aprendizagem α. Realizar o *exploit* com uma maior frequência pode possibilitar com que o agente aproxime mais rapidamente a função valor. Contudo, caso o agente não tenha realizado a etapa de *exploration* o suficiente, o agente pode encontrar uma política sub-ótima ao invés da ótima.

Para lidar com este cenário, o método PPO calcula a diferença (r_t) entre a política utilizada para coletar as experiências e a política sendo atualizada. Com base nesta diferença é possível controlar a distância entre as políticas (através do parâmetro ϵ da Equação 3.10) e desincentivar atualizações agressivas. Logo, o método PPO apresenta ser uma alternativa possível para problemas complexos. Este é o caso encontrado nas políticas de gerenciamento implementadas no *Resource and Job Management System* (RJMS), que são problemas do tipo NP-difícil. Não surpreendente, é

possível encontrar trabalhos relacionados que já aplicaram DRL em sistemas distribuídos.

3.5 TRABALHOS RELACIONADOS

A eficiência energética de plataformas computacionais passou a ser uma métrica de interesse na década de 2000. O paradigma de desempenho a qualquer custo não é sustentável sobre aspectos financeiros e ambientais (FENG; CAMERON, 2007). Desde então, foram publicados estudos com foco em diferentes estratégias, como: *fine-grained power management* (ETINSKI et al., 2012b; MARZOLLA; MIRANDOLA, 2013); *coarse-grained power management* (DUTOT et al., 2017); escalonamento (FELLER; RILLING; MORIN, 2011); e *thermal management* (SAROOD; KALE, 2011).

Os primeiros estudos sobre estratégias de *shutdown* para economizar energia iniciam em 2001. Utilizando um algoritmo de distribuição de carga, foi possível economizar energia através da concentração dos processos em um menor número de recursos, resultando em menos recursos ativos (PINHEIRO et al., 2001). De modo similar, Muse (CHASE et al., 2001) utiliza um algoritmo guloso para adaptar dinamicamente o conjunto de recursos ativos de acordo com a demanda. Ambas as abordagens são similares às propostas deste trabalho, pois buscam adaptar o número de recursos ativos com base na demanda. Contudo, estes trabalhos não consideraram o custo da transição entre os estados de um recurso, condição que ocorre quando um recurso é desligado ou inicializado.

Como observado na Seção 2.4.1, o custo de transição pode ser custoso tanto em termos de energia como de desempenho, em razão do tempo requerido para completar a transição. Partindo deste entendimento, o ERIDIS trabalha no nível de plataforma para decidir os momentos em que os recursos podem ser desligados baseado em previsões sobre a carga de trabalho (ORGERIE; LEFÈVRE, 2011). O mecanismo utilizado para fazer as previsões utilizam a média dos períodos de inatividade em conjunto com a média das diferenças entre as previsões passadas e os valores reais.

De modo similar, o algoritmo Inertial Shutdown (POQUET, 2017, p. 85) adota uma estratégia de *Off Reservation* (OR) para adaptar dinamicamente o número de recursos ativos com base em estimativas sobre a falta de capacidade de resposta do sistema. Esta falta de capacidade é uma estimativa do tempo requerido para executar todos os processos da fila. Quando a capacidade de resposta está decaindo/aumentado, o algoritmo utiliza uma política para liberar/reservar recursos. Recursos reservados pelo Inertial Shutdown são desligados e não podem ser utilizados até que liberados, estratégia que também é adotada neste trabalho. Contudo, a principal diferença com este trabalho está na utilização de DRL para treinar um agente a otimizar o desli-

gamento dos recursos e o escalonamento dos processos. Logo, a parte de estimação é feita pelo agente e inferida através das experiências observadas no processo de treinamento.

A utilização de RL, em geral, não é uma exclusividade deste trabalho. Desde 1995 é estudado como treinar agentes para otimizar o gerenciamento de recursos (ZHANG; DIETTERICH, 1995; ZOMAYA; CLEMENTS; OLARIU, 1998). Não obstante, DRL pode ser visualizada como uma extensão à área de RL que utiliza métodos de DL para lidar com tarefas complexas. Sua principal adoção vem dos recentes avanços alcançados com as técnicas de DRL (MNIH et al., 2015), levantando perguntas sobre o seu desempenho ao lidar com problemas de gerenciamento de recursos.

Seguindo este contexto, na Tabela 1 é apresentado os principais trabalhos relacionados que aplicam RL e DRL. O DeepRM (MAO et al., 2016) é uma tentativa de construir agentes para lidar com a tarefa de escalonamento e minimizar o *slowdown*. O algoritmo é treinado com o método REINFORCE, similar ao utilizado neste trabalho. Contudo, o objetivo em questão e o modelo de ambiente diferem consideravelmente. Para exemplificar, no DeepRM a plataforma é modelada através de uma matriz RxT, sendo R o número de recursos e T a janela de tempo. A ideia é similar a um gráfico de Gantt, e cada processo na fila é também representado por esta matriz. Não obstante, o ambiente de simulação utiliza cargas de trabalho e plataformas sintéticas enquanto, neste trabalho, ambas são oriundas de sistemas reais.

O método REINFORCE também é utilizado para treinar um agente a escalonar tarefas que são organizadas em grafos acíclicos dirigidos (WU et al., 2018). Neste trabalho, o objetivo é selecionar qual processador deve executar a primeira tarefa da fila, de modo a otimizar o desempenho. O mesmo pode ser observado em Kintsakis, Psomopoulos e Mitkas (2019), que utiliza uma arquitetura de DNN chamada de *pointer network* para representar a política (VINYALS; FORTUNATO; JAITLY, 2015). Kintsakis, Psomopoulos e Mitkas (2019) objetiva otimizar o desempenho em sistemas de gerenciamento de *workflow* e também utiliza informações sobre a probabilidade de uma tarefa falhar, semelhante a ideia de nível de confiança adotada neste trabalho (Seção 4.3.1). Contudo, ambos os trabalhos não podem ser comparados com este pois a formulação do problema, as cargas e plataformas diferem drasticamente.

O algoritmo DRL-Cloud (CHENG; LI; NAZARIAN, 2018) é uma abordagem que utiliza métodos de DRL para minimizar o consumo de energia. É utilizado o método DQN para treinar um agente para lidar com o provisionamento de recursos e outro para lidar com o escalonamento de tarefas em computação em nuvem. Deste modo, tanto a carga de trabalho utilizada como o modelo de ambiente difere da abordagem adotada neste trabalho. Em grades computacionais, um recurso pode não ser compartilhado entre múltiplos usuários enquanto, na computação em nuvem, máquinas

virtuais de distintos usuários podem compartilhar o mesmo recurso.

Utilizando o mesmo método do DRL-Cloud, o DeepEE propôs maximizar a eficiência energética trabalhando na alocação dos processos e no controle da taxa de fluxo de ar. Para o treinamento e avaliação, tanto a carga de trabalho como a plataforma foram extraídas do *data center National Supercomputing Centre* (NSCC) localizado em Singapura. Através de simulações, os autores conseguiram minimizar entre $10-15\%$ o consumo de energia nos cenários avaliados (Ran et al., 2019). Resultado similar que também foi obtido utilizando os *traces* da NSCC e o mesmo método em uma abordagem que considera a temperatura dos processadores durante a alocação dos processos. Neste caso, os autores treinaram um segundo modelo para replicar o comportamento do *data center* utilizando redes neurais recorrentes do tipo *Long Short-Term Memory* (LSTM). Este modelo é utilizado tanto para alimentar o treinamento do agente como para ignorar as ações que possam resultar no superaquecimento dos processadores da plataforma (Yi et al., 2020). Ambas propostas apresentaram resultados sólidos que foram obtidos através de simulações alimentadas com dados reais. Contudo, por abordarem o domínio de *High Performance Computing* (HPC), as formulações adotadas para resolver o problema diferem das deste trabalho. Não obstante, o problema de desligamento não é considerado nestes trabalhos.

Tabela 1 – Características dos principais trabalhos relacionados.

Autor	Objetivo	Técnica	Simulador	Domínio
Yi et al. (2020)	Energia e Temp.	Deep Q-network	Ad-hoc	HPC
Ran et al. (2019)	Energia	Deep Q-network	Ad-hoc	HPC
Kintsakis, Psomopoulos e Mitkas (2019)	Desempenho	REINFORCE	Ad-hoc	Grid
Orhean, Pop e Raicu (2018)	Desempenho	Q-Learning e SARSA	WorkflowSim	Cluster
Wu et al. (2018)	Desempenho	REINFORCE	Ad-hoc	Cluster
Moghadam e Babamir (2018)	Custo de Comunicação	Q-learning	OptorSim	Cluster
CHENG, Li e Nazarian (2018)	Energia	Deep Q-network	Ad-hoc	Cloud
Liu et al. (2017)	Energia	Deep Q-network	Ad-hoc	Cloud
Mao et al. (2016)	Slowdown	REINFORCE	Ad-hoc	Cluster
Tong et al. (2014)	Tempo de Resposta	Q-learning	Ad-hoc	Grid

Combinando métodos de RL para lidar com a alocação de recursos e gerenciamento de energia, Liu et al. (2017) propôs uma abordagem hierárquica. No nível superior, é utilizado um *auto-encoder* para reduzir a dimensionalidade dos dados de entrada do DQN, que é utilizado para treinar o agente. No nível inferior, uma LSTM é treinada para estimar o tempo de chegada do próximo processo de modo a ser utilizado como entrada para um algoritmo que utiliza o método Q-learning para controlar o desligamento do recurso localmente. Ao invés de separar a estratégia em predição e controle e atuar no nível de recurso, neste trabalho foi adotado uma estratégia fim-à-fim que utiliza DRL para trabalhar no nível de plataforma, controlando todos os recursos.

No trabalho de Orhean, Pop e Raicu (2018) foi proposto a ferramenta Machine Learning Box (MBox) como um meio de fornecer soluções de escalonamento base-

ado em ML como serviço. Neste contexto, os métodos Q-learning e SARSA foram utilizados para avaliar a proposta em um ambiente que simula um sistema distribuído. Com esta finalidade, a ferramenta foi integrada com o WorkflowSim (CHEN; Deelman, 2012), que simula *workflows* em ambientes distribuídos. Contudo, o modelo de ambiente utilizado é limitado, sendo considerado que todas as tarefas possuem as mesmas propriedades e somente um processo pode executar por vez. Não obstante, não foi possível encontrar documentação sobre o MBox que permita a sua utilização em outros trabalhos.

Seguindo esta linha, Q-learning também é utilizado para escalonar tarefas de uso intensivo de dados em um ambiente baseado em agregados que seguem a ideia das grades de dados (Discutido na Seção 2.1). O método proposto segue uma arquitetura hierárquica, na qual um corretor global seleciona o agregado com o menor custo de comunicação e um corretor local, que utiliza o Q-learning, seleciona o nó do agregado (MOGHADAM; BABAMIR, 2018). O mesmo método também pode ser observado em Tong et al. (2014), que objetiva minimizar o tempo de resposta através da alocação de recursos. Logo, os objetivos e escopos adotados diferem da proposta deste trabalho.

Observado os trabalhos relacionados, foi possível identificar uma lacuna no contexto de políticas de gerenciamento para grades computacionais orientadas a energia. Não obstante, na maioria dos trabalhos foram utilizados simuladores ad-hoc, específicos para o ambiente modelado, e cargas de trabalhos sintéticas, que não representam o comportamento de um ambiente real. Deste modo, este trabalho apresenta dois métodos baseados em DRL para otimizar o consumo de energia em grades computacionais que são treinados e avaliados utilizando plataformas e cargas de trabalho reais.

3.6 CONSIDERAÇÕES PARCIAIS

A concepção do RL introduziu uma abordagem computacional que possibilita treinar um agente a desempenhar uma tarefa através da interação com um ambiente. Problemas de RL são modelados através da definição de MDPs, cujo objetivo é estabelecer as regras que regem o ambiente no qual o algoritmo interage. De forma autônoma, um agente aprende a resolver tarefas mapeando uma ação a cada situação observado no ambiente. Para tal, um sinal de recompensa é enviado em cada tomada de decisão de modo que o agente possa assimilar quais ações tem maior impacto positivo no objetivo em questão. Deste modo, o agente reforça a tendência de repetição das melhores ações para, então, tentar encontrar uma política ótima.

Neste capítulo, as principais técnicas de RL foram introduzidas. Métodos re-

centes que utilizam técnicas de DL para encontrar uma solução aproximada, dando origem ao termo DRL, também foram abordados. Como foi destacado, métodos de DRL podem ser aplicados quando um problema tem um espaço extenso de estados e ações. Neste caso, os métodos tradicionais de RL podem não ser capazes de encontrar uma política ótima, exigindo a redução do escopo do problema (ORHEAN; POP; RAICU, 2018).

A dinâmica utilizada pelos métodos de DRL para treinar um agente a encontrar uma política ótima torna suscetível a sua implementação em ambientes dinâmicos, como as grades computacionais. Através do treinamento, o agente pode encontrar uma política específica para o sistema, explorando os seus padrões de utilização de modo a otimizar o objetivo em questão. Como já foi levantado, uma política baseada em regras pode não apresentar a melhor solução sobre todas as cargas de trabalho submetidas em um sistema em grades. Contudo, a elaboração de políticas específicas não é trivial pois sistemas em grades são dinâmicos e complexos, a carga de trabalho pode mudar de comportamento em questão de dias ou semanas (LELONG; REIS; TRYSTRAM, 2018). Deste modo, métodos de DRL podem pavimentar um novo meio de construir políticas e técnicas de gerenciamento adaptativos.

4 *DEEP RESOURCE AND JOB MANAGEMENT*

O conceito de paradigma de computação distribuída materializado pela definição de grades computacionais, possibilitou a construção de plataformas em larga escala a partir do compartilhamento de recursos entre instituições geograficamente distribuídas. De modo a otimizar e coordenar a utilização dos recursos destas plataformas, mecanismos de gerenciamento são implementados sobre os diferentes níveis de controle do *Resource and Job Management System* (RJMS) (POQUET, 2017). Dada a dinamicidade e complexidade de sistemas em grades, mecanismos tradicionais são baseados em regras fixas e simples. Deste modo, por não levar em consideração os padrões de utilização na carga de trabalho, é obtida uma solução não otimizada (LEGRAND; TRUSTRAM; ZRIGUI, 2019).

Como apresentado na Seção 2.3, mecanismos baseados em regras fixas estão sujeitos a variações no seu desempenho em função da carga de trabalho. Técnicas de *shutdown*, que são implementadas para minimizar o desperdício de energia através do desligamento dos recursos em momentos oportunos, podem acabar por produzir o efeito oposto, aumentando o consumo de energia e degradando o desempenho. Este é um exemplo que pode ocorrer quando o intervalo entre as submissões de processos coincide com o tempo em que o comando de desligamento é emitido, forçando a reinicialização desnecessária dos recursos para atender os processos da fila (LIU et al., 2017; RAïS et al., 2018).

O mesmo pode ser observado no contexto do escalonamento. Ao fixar uma regra na política de escalonamento é possível que alternativas melhores, em termos de desempenho e energia, sejam ignoradas. Esta é a justificativa do *Extensible Argonne Scheduling sYstem Backfilling* (EASY) para substituir a política *First Come First Served* (FCFS) e aumentar a utilização do sistema, ignorando as prioridades dos processos. Contudo, não existem garantias que o EASY apresentará o melhor desempenho nas cargas futuras. Deste modo, conhecer o comportamento de uma carga de trabalho pode ser crucial para a otimização dos mecanismos de gerenciamento (VASILE et al., 2018; LEGRAND; TRUSTRAM; ZRIGUI, 2019).

Para verificar estes cenários, *traces* de alguns agregados da GRID'5000 foram coletados e uma análise nestes dados foi conduzida. Na Tabela 2 é apresentado algumas propriedades destes *traces*. Para cada *trace*, foram removidos os processos, que: não executaram; requisitaram uma quantidade inválida de recursos; ou possuem um tempo esperado de execução (*walltime*) inválido.

Analisando a Tabela 2, é possível perceber que os agregados possuem uma

Tabela 2 – Propriedades dos *traces* da GRID'5000 analisados.

Sítio	Agregado	Período	# Núcleos	# Processos	Processos Seq.	Util.
Lyon	Taurus	Set 12 - Out 19	168	81.925	54,1 %	51,7 %
Lyon	Hercule	Out 12 - Out 19	48	50.003	46,9 %	49,8 %
Lyon	Nova	Fev 17 - Out 19	368	42.171	52,9 %	58,3 %
Lyon	Orion	Out 12 - Out 19	48	66.005	39,7 %	62,7 %
Nancy	Graphite	Dez 13 - Out 19	64	60.786	56,0 %	47,3 %
Nancy	Grimoire	Jan 16 - Out 19	128	36.986	52,7 %	56,4 %
Nancy	Grisou	Jan 16 - Out 19	816	99.291	75,4 %	69,5 %
Nantes	Econome	Abr 14 - Out 19	352	66.557	66,8 %	57,6 %
Nantes	Ecotype	Jan 18 - Out 19	960	39.868	81,1 %	66,5 %

taxa média de utilização de 57%. O tamanho de cada plataforma (# Núcleos) varia consideravelmente dentro do intervalo $[48, 960]$. O mesmo pode ser observado na quantidade de processos executados em cada agregado, que possui uma amplitude de 62.305. A taxa de ocorrência de submissões de processos sequenciais também é significativa, atingindo $81,1\%$ da carga de trabalho do agregado Ecotype de Nantes. Em geral, a média de submissão deste tipo de processo ficou em $58,4\%$, indicando que a maioria dos processos submetidos a estes agregados são oriundos de submissões sequenciais.

Formalmente, submissões sequenciais são separadas por um curto intervalo de tempo (menor que 5 minutos neste trabalho). Como consequência, estes processos podem aumentar o consumo de energia da plataforma forçando a inicialização dos recursos que lhe foram alocados. Caso o processo tenha um tempo curto de execução, a energia consumida na inicialização e pelo período em que os recursos permanecem ociosos pode ser considerada um desperdício. Logo, é importante considerar a ocorrência destes processos na tomada de decisão de uma política de gerenciamento.

Para melhor entender o comportamento das cargas de trabalho, os agregados mais antigos de cada sítio (Taurus, Orion, Graphite e Econome) são selecionados para melhor análise. Com este objetivo, na Figura 11 é ilustrado a taxa de ocorrência de processos normais e sequenciais, sendo os processos agrupados pelo seu último status registrado no sistema (Terminado ou Erro). Os dados foram agrupados anualmente e os dados de anos incompletos foram descartados.

Observando a Figura 11, é possível perceber que a ocorrência de processos sequenciais não é sazonal. Todos os anos possuem uma taxa superior à 30% deste tipo de processo enquanto, em alguns anos, estes processos representam mais de 80% da carga (como em 2015 no agregado Econome). Não obstante, é possível perceber a alta taxa de processos que terminaram em um estado de erro. As cargas de trabalho são majoritariamente compostas por processos que falharam durante a execução. Este comportamento reforça a ideia de rajada de submissões, que podem

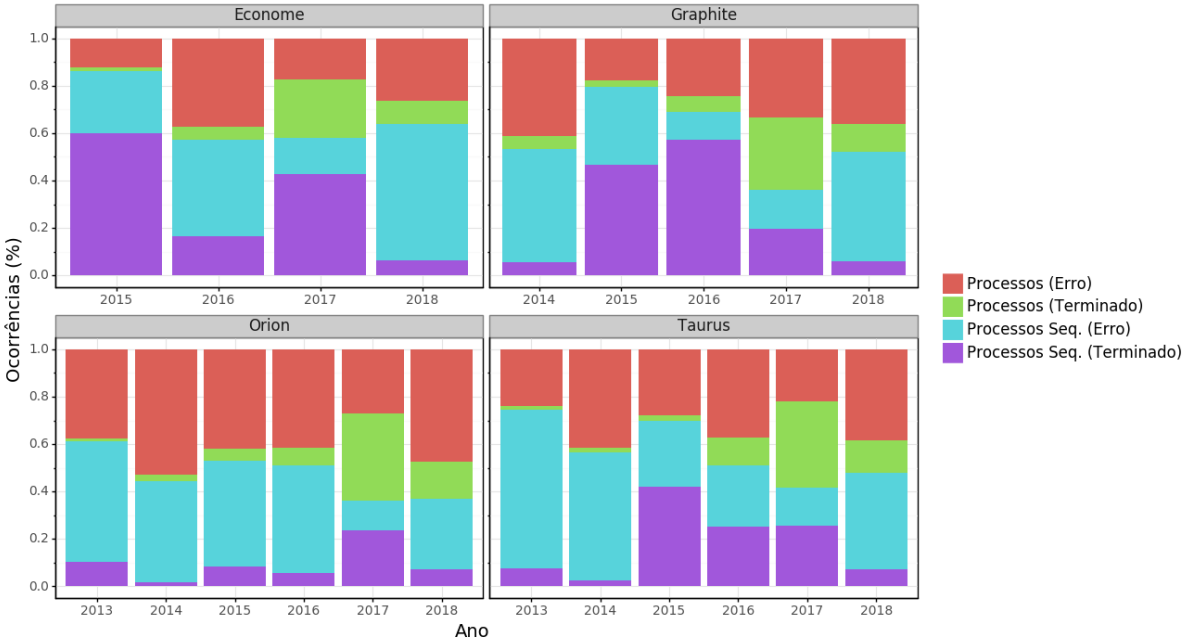


Figura 11 – Ocorrência de submissões sequenciais de processos nas cargas de trabalho dos agregados selecionados.

ser caracterizadas por repetidas submissões do mesmo processo. Caso o processo termine em erro, tais submissões são, em geral, repetidas em um curto período de tempo. Como consequência, o número de submissões sequenciais e de processos que terminam em erro irá aumentar. Logo, a frequência com que os processos falham não pode ser negligenciado.

Uma rajada de submissões pode ser perigosa para sistemas que adotam uma política de *timeout* agressiva para economizar energia. Se os recursos são desligados entre as submissões dos processos, o potencial de desperdício de energia é amplificado devido às possíveis reinicializações desnecessárias dos recursos. Por este motivo, ao considerar os padrões observados nas cargas de trabalho no processo de gerenciamento é possível obter um potencial significativo de economia de energia em comparação aos métodos heurísticos.

Neste contexto, o presente trabalho apresenta duas políticas de gerenciamento baseadas em *Deep Reinforcement Learning* (DRL) como uma alternativa para construir políticas específicas para os padrões de utilização do sistema. Atuando sobre dois níveis distintos de controle do RJMS, um método de *shutdown* (chamado de DeepShutdown) e outro de escalonamento (chamado de DeepScheduler) são treinados utilizando métodos de DRL sobre os *traces* da GRID'5000 analisados na Figura 11.

O presente capítulo está organizado da seguinte maneira. No decorrer da Seção 4.1 é descrito que tipo de sistema em grades é considerado neste trabalho e apresentado o seu formalismo. Na Seção 4.2 é formulado o problema do *shutdown* e descrita a estratégia do DeepShutdown. O mesmo ocorre na Seção 4.3, na qual o

problema de escalonamento e a estratégia do DeepScheduler são apresentados. Por último, na Seção 4.4 é apresentado as considerações parciais.

4.1 MODELO DO SISTEMA

Esta seção descreve o modelo de plataforma e de carga de trabalho utilizados para formular o problema do *shutdown* e do escalonamento. Seguindo a literatura especializada, esta descrição é baseada nos modelos adotados no Simgrid (CASANOVA et al., 2014), um simulador de grades computacionais, e no Batsim (DUTOT et al., 2017), um simulador de RJMS que utiliza o Simgrid para simular a plataforma computacional.

4.1.1 Modelo de Plataforma

A plataforma de um sistema em grades é composta por um conjunto de agregados de recursos organizados em sítios que podem estar geograficamente distribuídos. Cada sítio contém um ou mais agregados com um número arbitrário de recursos computacionais, de armazenamento ou de redes. No presente trabalho, considera-se somente os recursos computacionais, também referidos como nós na literatura. Cada nó é formado por um conjunto de processadores que possuem um ou mais núcleos. Usuários podem requisitar um ou mais agregados, nós, processadores ou núcleos para executarem suas aplicações. Assim, núcleos são simplesmente referenciados como recursos r, sendo caracterizados, por: (*i*) sua capacidade computacional (cpu_r), expresso em flop/s; (*ii*) seu estado atual (s_r); (*iii*) e seu perfil de consumo de energia (p_r), expresso em watts.

Neste trabalho, a plataforma segue o modelo da GRID'5000, e, portanto, é composta somente por agregados de recursos homogêneos. Deste modo, cada nó possui o mesmo número de recursos e cada recurso possui a mesma capacidade computacional e perfil de energia. Nós são independentes e possuem um conjunto único de recursos da plataforma. Neste contexto, nós somente podem ser desligados se (e somente se) todos os seus recursos estão ociosos. Do mesmo modo que, caso um nó seja inicializado, todos os seus recursos também serão inicializados. Como consequência, um nó só estará ocioso quando todos os seus recursos estão ociosos. Do mesmo modo que, um nó está trabalhando caso ao menos um de seus recursos esteja executando uma aplicação. Por simplicidade, a comunicação entre os recursos é desconsiderada e a transferência de dados entre processos paralelos ocorre imediatamente.

Como descrito na Seção 2.4.1, cada recurso $r \in R$ pode estar somente em um de seus estados possíveis $S = \{computando, ocioso, off, on \rightarrow off, off \rightarrow on\}$

no tempo t. Cada estado $s \in S$ é associado a um perfil de potência p_s, expresso em watts, que determina o seu consumo de energia. O estado de um recurso é modificado somente através de eventos externos, a partir do início da execução de um processo ou de um comando do RJMS.

Na Figura 12 é possível visualizar o modelo de recurso considerado, assim como as transições entre os estados e os seus custos, em termos de energia. Os dados utilizados neste exemplo são baseados nas informações extraídas no trabalho de Poquet (2017), que realizou um conjunto de experimentos na GRID'5000 para medir o consumo de energia real em cada um dos estados de um recurso.

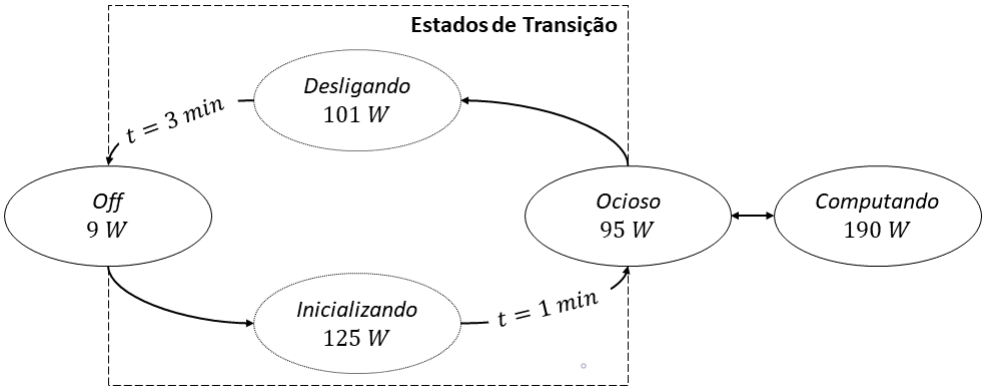


Figura 12 – Transições entre estados de um recurso e seu consumo relativo de energia.

Observando a Figura 12, é possível perceber que existe um tempo significativo para desligar e inicializar um recurso. Não obstante, estes estados de transição possuem um consumo de energia (de $101W$ e $125W$) superior ao seu consumo quando ocioso (de $95W$). Logo, o seu desligamento só irá compensar caso o recurso não seja inicializado logo em seguida.

4.1.2 Modelo de Processos

Processos são submetidos pelos usuários de modo online, através da submissão de aplicações, e podem ser definidos como um conjunto de trabalho que deve ser executado em um ou mais recursos computacionais (paralelamente). Cada processo $j \in J$ de uma carga de trabalho J é caracterizado, por:

- O tempo de submissão r_j (conhecido somente na submissão);
- O número de recursos requisitados q_j (conhecido somente na submissão);
- O tempo esperado de execução $wall_j$, informado pelo usuário (também chamado de *walltime*; e
- O tempo real de execução p_j (conhecido somente quando o processo finaliza).

Neste trabalho, é definido que a carga de trabalho é composta somente por processos do tipo rígido e que executam em modo *batch* (descrito na Seção 2.3.1). Portanto, o processo só pode iniciar sua execução quando a quantidade total de recursos requisitada estiver disponível. Do mesmo modo que, cada recurso alocado deve permanecer reservado até que o tempo esperado de execução do processo termine ou até que o processo libere seus recursos prematuramente. Como consequência, recursos não podem ser compartilhados entre processos distintos. Tal comportamento segue o padrão de utilização da GRID'5000. Usuários obtêm controle exclusivo dos recursos que lhe foram alocados.

De modo a reproduzir o comportamento de cada processo de uma carga de trabalho real extraída dos *traces* da GRID'5000, o perfil de execução de um processo é calculado em função do seu tempo real de execução e da capacidade computacional dos recursos que lhe foram alocados. O perfil de execução é o que determina a quantidade de computação realizada pelo processo. Obter tal informação requer instrumentar o processador enquanto um processo está em execução, o que aumenta a fidedignidade das simulações (GE; FENG; CAMERON, 2005). Contudo, as cargas de trabalho coletadas não possuem esta informação e reexecutar os processos com esta finalidade está além deste trabalho. Por este motivo, é considerado que cada processa usa 100% do processador. Logo, o perfil de execução de um processo j pode ser calculado pela Equação (4.1).

$$cpu_j = p_j \times cpu_a \tag{4.1}$$

Sendo, cpu_a é a menor capacidade computacional ($\min_{a \in A_j} cpu_a$) dos recursos que foram alocados (A_j) e p_j é o tempo real de execução do processo j. A justificativa por calcular o perfil de execução com base no menor poder computacional dos recursos alocados está relacionado a premissa de que os recursos são exclusivos de cada processo até finalizarem. Logo, processos paralelos, que pedem dois ou mais recursos, vão obter o mesmo tempo real de execução definido na carga de trabalho independente das diferenças nas capacidades computacionais dos recursos alocados.

4.1.3 Modelo de Energia

O modelo de energia utilizado neste trabalho segue o modelo descrito na Seção 2.4.1. Logo, o consumo de energia de uma plataforma $G(t)$ no tempo t, depende somente dos estados dos recursos. Formalmente, o consumo de energia e_r de um recurso r, expresso em joules, no intervalo de tempo $[t_0, t_1]$ é definido na Equação (2.10)

Como exemplo, seguindo o perfil definido na Figura 12, um recurso consome $e_r = 18.180$ joules para desligar enquanto consome $e_r = 7.500$ joules para inicializar. Do mesmo modo que, se este recurso permanecer ocioso por dez minutos, este terá

um consumo total de energia de $e_r = 57.000$ joules. Logo, a Equação (2.10) pode ser interpretada como um caso especial do modelo adotado no SimGrid (CASANOVA et al., 2014) quando os recursos ativos somente podem estar ociosos (com uma carga de 0%) ou executando (com uma carga de 100%).

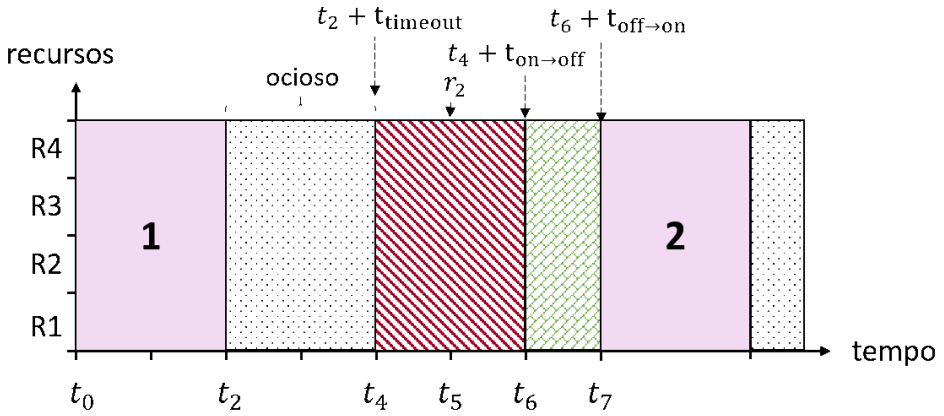
4.2 O PROBLEMA DO *SHUTDOWN*

O consumo de energia de recursos ociosos possui um impacto significativo no custo operacional de uma plataforma computacional. De modo a amenizar este custo, uma opção comum é desligar os recursos após um período de inatividade. Esta estratégia de *shutdown*, referida na literatura como *timeout* e, algumas vezes, como *opportunistic shutdown*, tem potencial para minimizar o desperdício de energia em períodos de ociosidade (DUTOT et al., 2017). Contudo, dependendo da carga de trabalho no sistema, esta mesma estratégia pode não só degradar o desempenho, mas, como também, aumentar o seu consumo de energia (ORGERIE; LEFÈVRE; GELAS, 2008).

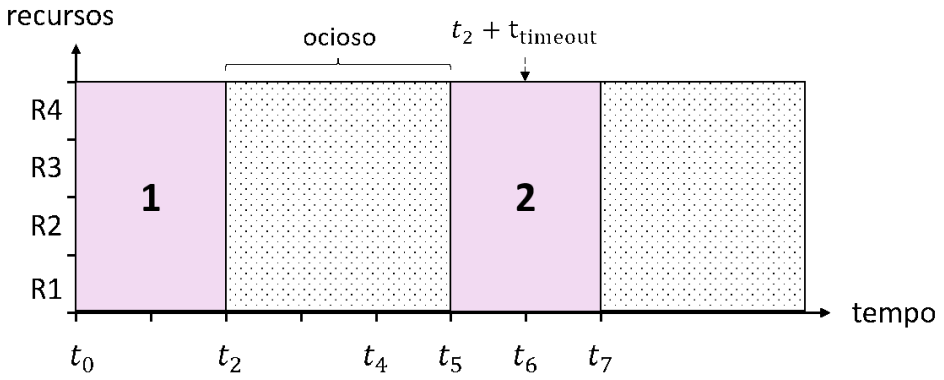
Para ilustrar como esta situação pode ocorrer, a Figura 13 apresenta dois cenários nos quais são aplicadas duas políticas de *timeout* em uma mesma carga de trabalho, cada política possui um tempo de ociosidade ($t_{timeout}$) distinto. Assume que $t_0, t_1, ... \in T$ são variáveis discretas que representam passos no tempo, r_j é o tempo de submissão do processo j, $R_1, R_2, ... \in R$ são os recursos computacionais da plataforma, $t_{on \to off}$ é o tempo requerido para um recurso ser desligado, $t_{off \to on}$ é o tempo de inicialização do recurso, P_s é o consumo de energia de um recurso que está no estado s e segue o perfil definido na Figura 12.

No primeiro cenário, ilustrado na Figura 13a, uma política de *timeout* com um tempo de ociosidade $t_{timeout} = 2t$ é implementada. No tempo t_0, todos os recursos estão sendo ocupados pelo processo 1 e nenhum processo está aguardando na fila. No tempo t_2, o processo 1 termina de executar e libera os recursos, que permanecem ociosos até o tempo t_4. Como $t_4 - t_2$ equivale ao tempo de ociosidade da política de *timeout*, os recursos são desligados. Os recursos começam o processo de desligamento que deve ser completado no tempo t_6. Contudo, no tempo t_5, o processo 2 é submetido ao sistema. Por não possuir recursos disponíveis atualmente, pois os mesmos estão sendo desligados, o processo 2 permanece na fila. Ao completar o desligamento, no tempo t_6, os recursos são inicializados para atender o processo 2. Os recursos completam a inicialização no tempo t_7 e o processo 2 pode iniciar sua execução. Neste cenário, a política de *timeout* aumentou tanto o tempo de espera do processo 2, degradando o desempenho do sistema, como o consumo de energia da plataforma, em virtude da reinicialização (desligar e inicializar) dos recursos.

Figura 13 – Ilustração do comportamento de duas políticas de *timeout* sobre uma mesma carga de trabalho.



(a) Estratégia de *timeout* com $t_{timeout} = 2t$



(b) Estratégia de *timeout* com $t_{timeout} = 4t$

Fonte: O Autor

A situação ocorrida no primeiro cenário pode ser mitigada através da adequação da política de *timeout* à carga de trabalho. Tal adequação é representada no segundo cenário, ilustrado na Figura 13b, na qual uma política de *timeout* com um tempo de ociosidade $t_{timeout} = 4t$ é aplicada. O exemplo inicia com a mesma observação, do tempo t_0 até t_2 os recursos estão sendo ocupados pelo processo 1. Após a liberação dos recursos pelo processo 1, estes permanecem ociosos até a submissão do processo 2 no tempo t_5. Neste caso, o tempo de ociosidade definido na política não é observada pois $t_5 - t_2 < t_{timeout}$. Logo, os recursos não são desligados e o processo 2 pode iniciar sua execução imediatamente.

Em relação ao primeiro cenário, a política do segundo cenário conseguiu reduzir o consumo de energia, pois o tempo de inatividade compensou o custo de reinicializar os recursos, ao mesmo tempo em que o processo 2 não foi atrasado. Caso nenhum outro processo seja submetido ao sistema, após ser observado o tempo de ociosidade definido na política, os recursos irão ser desligados de modo a economizar a energia. Contudo, não existem garantias que nenhum outro processo será submetido ao sis-

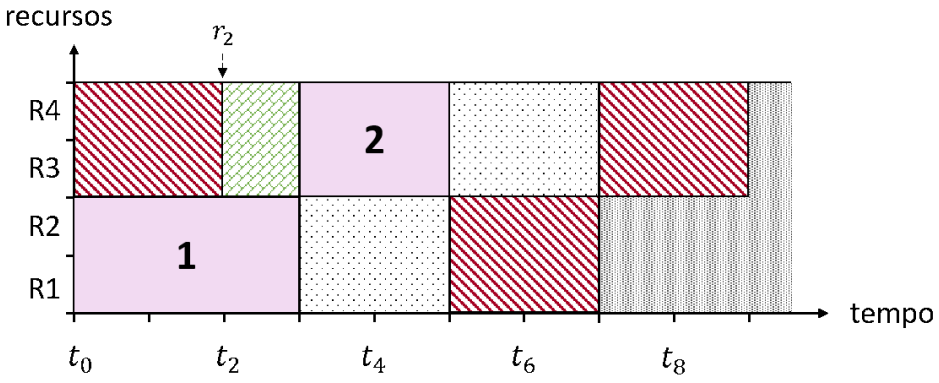
tema e o mesmo exemplo, ilustrado no primeiro cenário, ocorra também com esta nova política de *timeout*. Como consequência, o tempo de ociosidade deve ser calculado dinamicamente em função da carga de trabalho atual no sistema. Especificamente, a política deve ter informações sobre os processos futuros para ser possível determinar se compensa desligar os recursos no momento (RAïS et al., 2018). Contudo, obter ou prever tal informação é um processo complexo devido a dinamicidade dos sistemas em grades. Os padrões de utilização podem mudar repentinamente, em questão de semanas ou dias (LEGRAND; TRUSTRAM; ZRIGUI, 2019). Como alternativa, é possível aplicar uma estratégia de *Off Reservation* (OR), descrito na Seção 2.4.2.

Na Figura 14 é ilustrado um exemplo de como uma estratégia de OR pode apresentar um melhor comportamento em comparação a uma política de *timeout*. No primeiro cenário, ilustrado na Figura 14a, uma política de *timeout* com tempo de ociosidade de $t_{timeout} = 2t$ é aplicada. No tempo t_0, os recursos r_1 e r_2 estão sendo utilizados pelo processo 1 enquanto os recursos r_3 e r_4 estão sendo desligados. No tempo t_2, o processo 2 é submetido e pede dois recursos para execução. Como consequência, os recursos r_3 e r_4 são inicializados pois os demais estão ainda sendo utilizados pelo processo 1. No tempo t_3, os recursos terminam de inicializar e o processo 2 pode iniciar sua execução. Quando o processo 2 inicia, inesperadamente o processo 1 libera seus recursos, que permanecem ociosos até que o tempo de ociosidade da política seja observado novamente. A mesma situação ocorre nos recursos que foram alocados para o processo 2 quando este finaliza, no tempo t_5. Portanto, os recursos r_1 e r_2 são desligados no tempo t_5 enquanto os recursos r_3 e r_4 são desligados no tempo t_7. Neste caso, o custo de reinicialização dos recursos poderia ser minimizado caso o processo 2 fosse atrasado por $1t$, equivalente ao tempo de inicialização dos recursos r_3 e r_4. Contudo, o escalonador precisaria saber de antemão o tempo de execução real do processo 1.

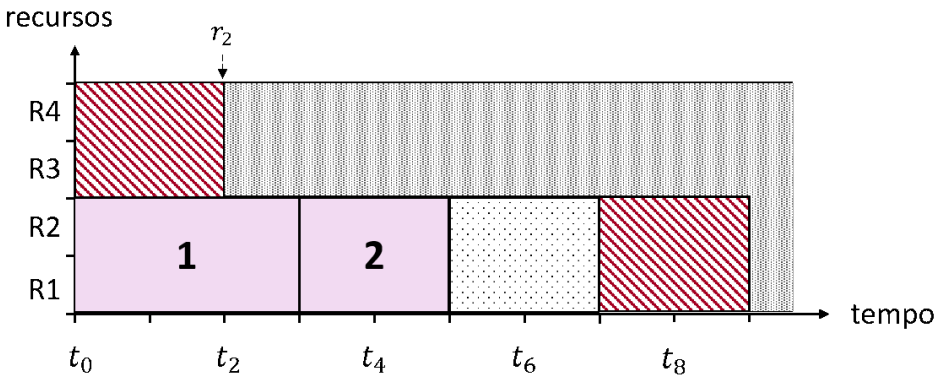
Ao invés de utilizar uma estimação do tempo de execução real dos processos ou tentar prever quando irá ocorrer novas submissões, é possível aplicar a estratégia de OR. Nesta estratégia, a política de OR reserva alguns recursos para si, impedindo a sua utilização por novos processos. Recursos reservados pelo OR são desligados imediatamente. Portanto, o OR pode ser interpretado como sendo um usuário que objetiva economizar energia, desligando os recursos que lhe foram reservados. Este comportamento é ilustrado na Figura 14b, no qual os recursos r_3 e r_4 são reservados pela política de OR. Neste cenário, como os recursos estão reservados pelo OR, o processo 2 é obrigado a esperar na fila até que os recursos r_1 e r_2 sejam liberados ou até que o OR libere os seus recursos, o que ocorrer primeiro. Deste modo, é possível evitar a reinicialização dos recursos r_3 e r_4, e assim economizar energia.

Uma possível vantagem deste método está relacionada a possibilidade de

Figura 14 – Ilustração do potencial de uma estratégia de OR.



(a) Estratégia de *timeout* com $t_{timeout} = 2t$



(b) Estratégia de *off reservation*

Fonte: O Autor

forçar o atraso de alguns processos até que compense o custo da inicialização dos recursos. Contudo, o problema nesta estratégia está justamente em determinar até que ponto compensa atrasar um processo, degradando o desempenho do sistema, para economizar energia. Como o tempo real de execução de um processo é desconhecido, não existem garantias em atrasar um processo na espera que recursos sejam prematuramente liberados. Como consequência, o tempo de espera na fila pode aumentar drasticamente a ponto de não compensar a economia de energia. De todo o modo, se aplicada adequadamente, tal estratégia tem potencial para atingir um maior nível de eficiência energética em comparação a uma política de *timeout*.

É neste contexto que o presente trabalho justifica a proposta do DeepShutdown, que utiliza DRL para treinar um agente à determinar, dinamicamente, a quantidade de recursos que devem permanecer reservados através da estratégia de OR. O objetivo é treinar o algoritmo em diferentes cargas de trabalho para aprender os momentos que compensem atrasar um processo, evitando reinicializações desnecessárias e sem comprometer o desempenho. Com esta finalidade, a formulação utilizada para definir o *Markov Decision Process* (MDP) e a descrição da estratégia adotada

pelo DeepShutdown são apresentados.

4.2.1 Formulação do Problema

O problema do *shutdown* consiste em determinar os momentos em que os recursos devem ser desligados com a finalidade de economizar energia. Integrar este problema com uma estratégia de OR adiciona uma camada de complexidade, pois o método deve também determinar por quanto tempo um processo deve ser atrasado. Processos são deliberadamente atrasados com a finalidade de mitigar a principal desvantagem de uma política de *timeout*, que ocorre quando o custo de reinicialização não compensa o custo de deixar o recurso ocioso ou desligado. Neste trabalho, é utilizado métodos de DRL para treinar um agente a realizar esta tarefa. Deste modo, o problema do *shutdown* é definido em um MDP.

Definido na Seção 3.2, um MDP define as regras que regem o relacionamento entre o agente (política) e o ambiente (RJMS). Dado um estado s_t no tempo t, o agente deve escolher uma ação $a_t \in A(s)$ que induz, no ambiente, uma distribuição de probabilidades $T(s_t, a_t)$ para os próximos estados. No próximo passo, o agente recebe o novo estado do ambiente em conjunto com um sinal de recompensa $R(s_t, a_t, s_{t+1})$, que representa a qualidade da ação a_t tomada no tempo t. Com base no sinal de recompensa, o agente busca selecionar a sequência de ações que maximize a sua recompensa acumulada esperada. Com tal finalidade, o agente otimiza sua política $\pi : S \rightarrow A$, reforçando a tendência de repetição das ações mais recompensadas em cada estado. Deste modo, conforme o número de interações/repetições aumenta, o agente vai assimilando quais ações o levam para os melhores estados que, consequentemente, otimizam o objetivo em questão.

Seguindo este *framework*, o problema do *shutdown* é formulado em um MDP, como:

State Space

O estado do ambiente é definido através da concatenação de n observações históricas do ambiente $H_t = [O_{t-n}, ..., O_{t-1}, O_t]$, do tempo $t - n$ até t. Uma observação O_t apresenta uma combinação do estado atual da plataforma, do estado da fila e do estado da simulação. No estado da plataforma é incluído informações sobre a quantidade de recursos em cada um de seus estados possíveis $[|r_{off}|, ..., |r_{computing}|]$. Deste modo, o agente consegue perceber quantos recursos estão sendo desligados ou quantos estão ociosos no momento. No estado da fila é incluído informações, sobre: a quantidade de processos na fila $|Q|$; uma promessa $prom_j$ do tempo em que o primeiro processo da fila deve iniciar sua execução, como utilizado pela política EASY descrita no Algoritmo 1; e um vetor de características dos primeiros k processos. O

vetor de características de um processo k é definido, como:

$$f_k = [q_k, wall_k, stretch_k, |j_{user}|] \tag{4.2}$$

Sendo, $|j_{user}|$ o número total de processos deste mesmo usuário no sistema (executando ou na fila) e o $stretch_k$ é a razão entre o tempo de espera e o tempo requisitado pelo processo k, definido na Equação (5.2). Por último, o estado da simulação apresenta informação sobre o tempo corrente de simulação. Assim, o agente pode inferir em quais horários o sistema está mais suscetível a receber uma rajada de submissões.

Action Space

De modo similar ao comportamento de um processo maleável no RJMS, o agente pode aumentar ou diminuir o tamanho de sua reserva através da requisição por recursos. O tamanho da reserva é o que determina a quantidade de recursos que serão mantidos reservados para o agente e, portanto, indisponíveis para alocação. Deste modo, o espaço de ações é dado por $\{\theta, 1, ..., G\}$, sendo que $a = G$ significa que o agente deseja reservar todos os recursos da plataforma e $a = \theta$ significa que o agente não deseja reservar recursos até a próxima tomada de decisão.

Quando o agente efetua uma reserva, os recursos são imediatamente desligados seguindo o modelo descrito na Seção 4.1.1. Recursos reservados não podem ser utilizados pelos processos até que sejam liberados pelo agente, através da diminuição do tamanho de reserva. Este comportamento é similar ao comportamento esperado quando um usuário pede por recursos. Com a finalidade de diminuir o tamanho do espaço de ações, é definido que o agente pode controlar somente os nós da plataforma. Como definido na Seção 4.1.1, nós são agregados de recursos e operações realizadas no nó são repassados para todos os recursos deste nó.

Reward Function

A função de recompensa é formulada de modo a guiar o agente através do objetivo principal: minimizar o desperdício de energia mantendo o desempenho sobre um nível pré-determinado. Com esta finalidade, a recompensa é formalmente definida na Equação (4.3).

$$R = -(E_{waste} + QoS) \tag{4.3}$$

$$E_{waste} = E_{idle} + E_{off \to on} + E_{on \to off} \tag{4.4}$$

Sendo, o desperdício de energia (E_{waste}), definido na Equação (4.4), corresponde ao consumo total de energia gasto enquanto os recursos estiveram ociosos

ou transitando entre ligado e desligado, desde a última tomada de decisão. O QoS é definido na Equação (4.5) e corresponde a uma métrica *job-centric*.

$$QoS = \sum_{j \in Q} \begin{cases} q_j & \text{if } wait_j \geq wall_j \times \tau \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Sendo, τ um parâmetro que controla a agressividade do algoritmo através da delimitação dos tempos de espera máximos desejados para cada processo na fila $j \in Q$.

A ideia principal é encorajar o agente à atrasar alguns processos na expectativa que recursos sejam liberados prematuramente de modo a evitar reinicializações desnecessárias, reduzindo o desperdício de energia. Em outras palavras, o agente deve encontrar as oportunidades que compensem o atraso extra, em alguns processos, em termos de energia e desempenho. Portanto, a recompensa pode ser interpretada como uma penalização proporcional, ao: número corrente de recursos ociosos; o número corrente de recursos transitando entre ligado e desligado; e o número total de recursos requisitados pelos processos com um tempo de espera que extrapole o limite definido pela métrica QoS.

4.2.2 Método Proposto

No núcleo de sistemas em grades, o RJMS é um mecanismo crucial para otimizar o gerenciamento da plataforma. Dentre suas responsabilidades é possível citar: reserva, alocação, escalonamento, execução e o monitoramento de processos e recursos. Tais procedimentos podem ser estrategicamente implementados em componentes independentes, que atuam de modo coordenado. Com base neste entendimento, a proposta do DeepShutdown pode ser visualizada como um componente adicional no RJMS. Na Figura 15 é possível visualizar como o DeepShutdown é integrado ao *workflow* de um RJMS.

O DeepShutdown é estrategicamente posicionado uma etapa anterior ao escalonamento, que decide a ordem de execução dos processos. Deste modo, antes de o escalonador tomar suas decisões, o DeepShutdown pode decidir quais recursos devem ser reservados para si. Em outras palavras, o DeepShutdown pode ser visto como um usuário com máxima prioridade, sempre sendo atendido primeiro em detrimento dos processos na fila.

Requisições do DeepShutdown são interpretadas do mesmo modo que as requisições dos usuários, descrito na Seção 2.3. Portanto, assim que o DeepShutdown reserva ou libera recursos, o algoritmo de escalonamento decide quais processos da fila vão executar nos recursos remanescentes. Após o escalonador finalizar seu procedimento, todo este processo é repetido e, na próxima tomada de decisão, o De-

epShutdown pode, novamente, aumentar ou diminuir a sua quantidade de recursos reservados.

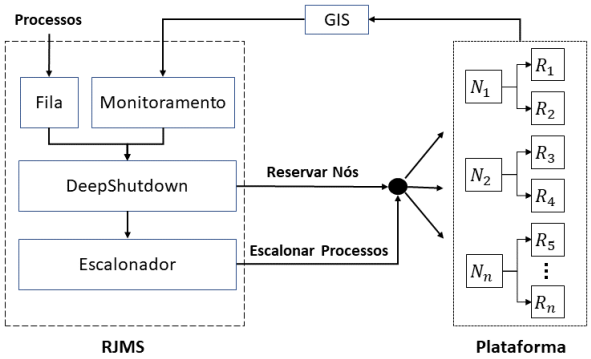


Figura 15 – Organização estrutural do DeepShutdown.

A escolha por tomar decisões um momento antes do algoritmo de escalonamento é para permitir que o DeepShutdown possa, indiretamente, controlar o escalonador. Este controle ocorre através da limitação do espaço de escalonamento que pode ser explorado para alocar os processos da fila. Ao reservar recursos, o DeepShutdown está interferindo nas opções de escalonamento. Conhecendo o comportamento do escalonador, caso o DeepShutdown preveja que o sistema irá receber uma rajada de requisições, é possível otimizar o escalonamento através do controle dinâmico da quantidade de recursos disponíveis para tal. Como consequência, é possível compactar os processos em um menor número de recursos, usando a função de recompensa como uma guia para determinar os momentos que compensem este comportamento. Não obstante, o algoritmo pode bloquear decisões de escalonamento que podem, potencialmente, impactar negativamente o seu objetivo.

Com o objetivo de treinar o agente, é utilizado o método *Proximal Policy Optimization* (PPO) descrito na Seção 3.4.3. O treinamento segue o estilo padrão dos métodos *actor-critic*, no qual um ou mais *actors* executam ações no ambiente e um *critic* avalia estas decisões. Neste trabalho, tanto os *actors* como o *critic* são representados por *Deep Neural Network*s (DNNs) que possuem a mesma arquitetura. Contudo, a diferença entre estas redes reside na camada de saída, sendo que a rede dos *actors* aplica uma função Softmax, para obter uma distribuição de probabilidades sobre as ações que podem ser executadas, e a rede do *critic* aplica uma função linear, para estimar as vantagens das ações executadas.

Na Figura 16 é ilustrado a arquitetura da DNN utilizada para a política (*actors*) e a função valor (*critic*). Esta arquitetura é composta por duas camadas internas, além da camada de entrada, que recebe as observações do ambiente, e da camada de saída. A primeira camada interna é uma rede neural recorrente do tipo *Long Short-Term Memory* (LSTM) com 128 unidades de memória e a segunda é uma *Feedforward*

Neural Network (FNN) com 64 unidades, sendo ligada na última saída do LSTM. Redes neurais recorrentes do tipo LSTM são conhecidas por lidar com sequências pois contém um estado interno que funciona como uma espécie de memória, permitindo que estas redes levem em consideração os dados predecessores (SUNDERMEYER; SCHLÜTER; NEY, 2012). Logo, dado ao tipo de observação esperado do ambiente que contém em seu estado n observações históricas, a escolha por uma LSTM é natural.

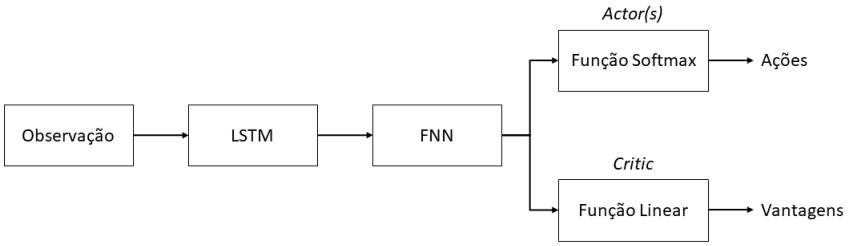


Figura 16 – Arquitetura da DNN utilizada no DeepShutdown.

Para reduzir a instabilidade no treinamento decorrente das estimações do *critic*, foi definido implementar a função *Generalized Advantage Estimation* (GAE) para quantificar a vantagem das ações em cada um dos estados observados. Formalmente, a função GAE é definida na Equação 3.12 (SCHULMAN et al., 2015). Com base nesta função vantagem, a função objetivo do PPO é calculada e os parâmetros são otimizados com o algoritmo Adam (KINGMA; BA, 2014). Demais detalhes permanecem os mesmos, como descrito na Seção 3.4.3.

4.3 O PROBLEMA DE ESCALONAMENTO

A otimização da eficiência de um sistema em grades, tanto em termos de custo operacional como de desempenho, está diretamente relacionada aos mecanismos de gerenciamento implementados no RJMS. Tais mecanismos trabalham em diferentes níveis de controle. O problema abordado pelo DeepShutdown trabalha, especificamente, no nível de plataforma, controlando os momentos em que os recursos devem ser e permanecer desligados. Conjuntamente com as técnicas de *shutdown*, o escalonamento é uma das estratégias de maior interesse dos provedores pois possuem um alto impacto esperado na eficiência do gerenciamento, tanto em termos de energia como de desempenho (BATES et al., 2015). Por este motivo, a otimização do escalonamento é também incluída neste trabalho.

Devido a complexidade natural dos sistemas em grades, políticas de escalonamento utilizadas são heurísticas simples e escaláveis. Tais heurísticas são baseadas em regras fixas e não levam em consideração os padrões de utilização do sistema. Como consequência, é possível observar uma perda de desempenho sobre certas

cargas de trabalho. Contudo, não existem garantias que o comportamento da carga permaneça constante nos próximos dias. De fato, as cargas de trabalho de um sistema em grades mudam em questão de dias e semanas, dificultando a elaboração de uma heurística que cubra todas as possibilidades. Deste modo, políticas de escalonamento atuais são baseadas no conhecimento que seus administradores possuem sobre o comportamento do sistema (LEGRAND; TRUSTRAM; ZRIGUI, 2019).

É neste contexto que o DeepScheduler, um método de escalonamento que utiliza DRL para aprender uma política específica ao comportamento observado nas cargas de trabalho, é proposto. Com base no treinamento, o agente pode reconhecer os padrões de utilização do sistema e adequar a sua política, dinamicamente, de modo a refletir na otimização do objetivo. Deste modo, é possível elaborar políticas de escalonamento específicas para cada sistema com base nos seus dados históricos de utilização. Nesta seção, o problema de escalonamento é formulado em um MDP para, em seguida, ser descrito o método proposto.

4.3.1 Formulação do Problema

Para ser possível aplicar métodos de *Reinforcement Learning* (RL), em geral, é necessário formular o problema em um MDP. Como descrito na Seção 3.2, um MDP define uma tupla (S, A, T, R). Seguindo este *framework* e considerando a matriz de transição (T) como desconhecida, o problema de escalonamento é formulado, como:

State Space

O estado do ambiente é formulado através da união do estado da fila s_q, da agenda de escalonamento do RJMS s_a, e de n observações históricas do estado da plataforma $s^p(t) = [s^p_{t-n}, ..., s^p_{t-1}, s^p_t]$. O estado da plataforma inclui informações, sobre: o número de recursos reservados, o número de processos na fila, o tempo de simulação corrente, e um indicativo do número de recursos que estão em cada um de seus estados possíveis, como descrito na Seção 2.4.1. Deste modo, o agente pode observar o tempo necessário para um recurso ser desligado e verificar a frequência que processos estão sendo submetidos ao sistema, no intervalo de tempo definido por n.

O estado da fila inclui informações sobre os k primeiros processos, sendo cada processo caracterizado, por: o número de recursos requisitados q_j; o tempo esperado de execução $wall_j$, fornecido pelo usuário; e um nível de confiança calculado a partir da média entre a razão do tempo real de processamento e o tempo esperado das últimas u submissões. Em outras palavras, o nível de confiança estima a probabilidade de um processo j executar menos que o tempo esperado de execução.

Por último, a agenda do RJMS inclui informações sobre os processos em exe-

cução, sendo caracterizados, por: o número de recursos alocados; o tempo restante de execução com base no seu tempo esperado; e uma estimação do tempo remanescente com base no nível de confiança.

Action Space

Durante o processo de escalonamento, o escalonador pode selecionar um ou mais processos da fila J para execução. Logo, o espaço de ações pode ser definido, como:

$$A = \{a | a \in \{\theta, 1, ..., |J|\}\} \tag{4.6}$$

Sendo, $a = 1$ indica que o agente deseja iniciar a execução do primeiro processo da fila e $a = \theta$ indica que nenhum processo deve ser escalonado nesta tomada de decisão.

De modo a permitir múltiplas decisões de escalonamento, o tempo de simulação é parado enquanto o agente está escalonando os processos. A simulação procede somente quando $a = \theta$ ou quando o agente toma uma decisão inválida, como quando o processo escalonado não possui recursos disponíveis para executar. Assim, a etapa de treinamento é simplificada através da redução do espaço de ações possíveis, pois não é necessário que o agente tome uma sequência de decisões em uma única tomada.

Reward Function

A recompensa foi formulada para orientar o agente a minimizar a quantidade de recursos ociosos enquanto mantém o desempenho médio dentro do esperado. Com esta finalidade, a função de recompensa é formulada como uma função linear ponderada de três métricas:

$$R = -\rho \times |N_{idle}| + \sigma \times QoS + \tau \times \mu \tag{4.7}$$

$$QoS = \sum_{j \in Q} \frac{-1}{wall_j} \tag{4.8}$$

Sendo, $|N_{idle}|$ a quantidade de recursos atualmente ociosos, QoS é formulado na Equação (4.8) e representa o número de processos na fila (Q) normalizado pelos tempos esperados de execução de cada processo ($wall$), μ é a taxa de utilização dos recursos, e os parâmetros ρ, σ e τ definem os pesos de cada métrica.

Ao interpretar a Equação 4.7, é possível observar que o agente é penalizado proporcionalmente ao número corrente de recursos ociosos ($|N_{idle}|$) e ao número normalizado de processos na fila. Deste modo, o objetivo é minimizar a quantidade de

recursos ativos enquanto reduz a média do *slowdown*, através da priorização dos processos menores. Não obstante, para encorajar o agente a escalonar os processos da fila, um bônus proporcional a taxa de utilização do sistema (μ), que corresponde ao número de recursos sendo utilizados atualmente, é fornecido.

4.3.2 Método Proposto

Como no DeepShutdown, a proposta do DeepScheduler utiliza o método PPO para treinar o agente a escalonar os processos do sistema. Assim, o DeepScheduler também segue o estilo dos métodos *actor-critic* e o seu treinamento ocorre de modo sincronizado. Não obstante, a mesma função vantagem GAE é utilizada. O objetivo deste trabalho não é avaliar técnicas de DRL, mas aplicá-las no contexto do RJMS em grades computacionais. Logo, outros métodos e parâmetros podem ser utilizados sem complicações em relação a possíveis conflitos nas implementações, sendo considerado como trabalhos futuros.

Devido as peculiaridades do problema de escalonamento, ao contrário da abordagem proposta no DeepShutdown (Seção 4.2.2), foi definido utilizar uma arquitetura de DNN diferente para as redes dos *actors* e do *critic*. No DeepScheduler, os *actors* e o *critic* possuem DNNs independentes, com pesos diferentes, mas que seguem a mesma arquitetura. Tal arquitetura é similar ao modelo *sequence-to-sequence* (seq2seq), composto por um *encoder* e um *decoder* (SUTSKEVER; VINYALS; LE, 2014). De modo a simplificar o treinamento, o *decoder* não é implementado e o estado oculto do *encoder* é transmitido diretamente para uma *FNN*, seguindo uma arquitetura de *sequence-to-element*. Logo, a configuração da DNN é composta por três camadas em paralelo que são formadas por *Gated Recurrent Unit*s (GRUs) com 64 unidades cada, que são concatenadas em uma FNN com 128 unidades (CHUNG et al., 2014).

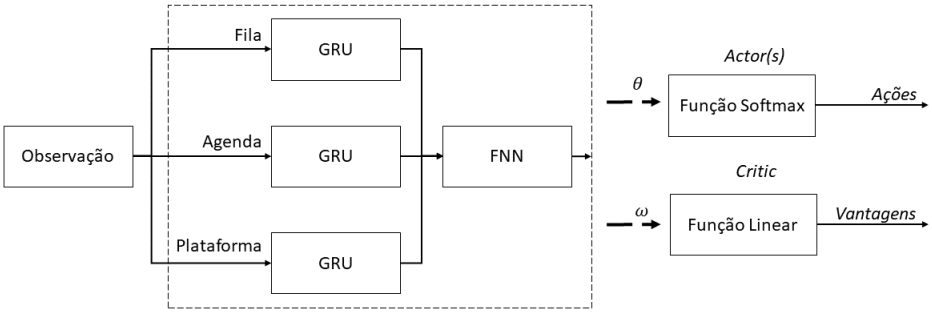


Figura 17 – Arquitetura da DNN utilizada no DeepScheduler.

Uma ilustração da arquitetura adotada no DeepScheduler é apresentada na Figura 17. Tal configuração de DNN é justificada pela formulação do problema. Cada camada de GRU é responsável por um tipo específico de observação do ambiente, sendo: uma sequência contendo características dos primeiros 20 processos da fila;

uma sequência contendo as características dos 20 primeiros processos a saírem do sistema, obtido da agenda do RJMS; e uma sequência contendo as últimas 20 observações do estado da plataforma. O limite do tamanho das sequências utilizadas neste trabalho não é imutável, podendo ser modificado sem complicações. A otimização dos parâmetros não está no escopo deste trabalho. Contudo, através de testes simples, tais valores apresentaram os melhores resultados. Não obstante, informações sobre o tempo necessário para os recursos mudarem de estado está incluso nas últimas 20 observações do estado da plataforma.

4.4 CONSIDERAÇÕES PARCIAIS

Devido a dinamicidade dos sistemas em grades, políticas de gerenciamento de recursos e processos têm sido, tradicionalmente, baseadas em regras fixas. Prever o comportamento das cargas de trabalho do sistema não é um processo trivial, justificando a utilização de soluções não otimizadas. Como uma alternativa para reverter este cenário e construir políticas otimizadas ao comportamento do sistema, duas políticas de gerenciamento foram propostas neste capítulo. A proposta do DeepShutdown objetiva ensinar um agente a utilizar uma estratégia de OR para minimizar o desperdício de energia advindo de reinicializações desnecessárias de recursos. De modo similar, a proposta do DeepScheduler busca compactar os processos no menor número possível de recursos ativos através do processo de escalonamento. Ambas as propostas utilizam métodos de DRL para treinar um agente a detectar padrões de interesse nas observações do ambiente, e assim otimizar o objetivo em questão.

Técnicas de RL tem sido utilizadas com sucesso em outros ambientes e áreas de estudo. Contudo, ainda faltam estudos acerca de seu comportamento no gerenciamento de recursos e processos em sistemas em grades. Um maior potencial na minimização do consumo de energia é passível de ser atingindo com políticas desenvolvidas especificamente para cada tipo de sistema e padrão de utilização. Deste modo, é necessário conduzir uma série de experimentos para validar este cenário e possibilitar uma discussão sobre o comportamento aprendido pelos agentes através da interação com os ambientes formulados pelos MDPs apresentados.

5 EXPERIMENTAÇÃO

Neste capítulo, a metodologia de avaliação adotada para avaliar o desempenho e o comportamento das propostas do DeepShutdown e do DeepScheduler é descrita. Com este objetivo, na Seção 5.1 é apresentado a arquitetura e mecânica do simulador utilizado para o treinamento e avaliação. O simulador é o responsável por reproduzir o comportamento do *Resource and Job Management System* (RJMS) em grades computacionais, sendo crucial para o treinamento dos métodos. Na Seção 5.2 é apresentado a configuração dos experimentos e de seus parâmetros, para permitir a reprodutibilidade dos resultados. Os resultados obtidos são apresentados na Seção 5.3. Com base nestes resultados, é iniciado uma discussão sobre o comportamento que foi aprendido em cada proposta na Seção 5.4. Por último, as considerações parciais são apresentadas na Seção 5.5.

5.1 SIMULADOR

Técnicas de *Reinforcement Learning* (RL) introduzem abordagens computacionais que possibilitam que um agente aprenda um comportamento, representado por uma política, através da experimentação com um ambiente. Dado o seu modo de aprendizagem, que segue uma abordagem de tentativa e erro, um agente pode levar um tempo considerado para conseguir aprender uma política. Este tempo pode depender de inúmeros fatores, como: complexidade do problema, tempo de resposta do ambiente e a técnica utilizada no treinamento. Sendo assim, é natural que um agente seja treinado através de simulações para posteriormente ser aplicado em um ambiente em produção.

Na literatura sobre RL, é possível encontrar simuladores específicos, normalmente utilizados para avaliar a eficiência de novos métodos de RL, como: o *Arcade Learning Environment* (ALE), que permite a simulação de jogos de Atari (BELLEMARE et al., 2013); e o *MUlti-JOint dynamics with COntact* (MuJoCo), um motor de física para simulações de corpo rígido (TODOROV; EREZ; TASSA, 2012). Contudo, o mesmo não pode ser observado no que diz respeito ao gerenciamento de recursos em grades computacionais.

Existem trabalhos sobre gerenciamento de recursos em grades computacionais e, de um modo geral, estes utilizam simuladores para avaliar o desempenho de seus algoritmos. Dentre os mais conhecidos estão o SimGrid (CASANOVA et al., 2014) e o GridSim (BUYYA; MURSHED, 2002). Ambos estes simuladores são responsáveis por simular o comportamento de uma plataforma computacional, modelando os

recursos. Contudo, são simuladores de propósito geral pois suportam uma variedade de domínios específicos, como: computação em nuvem, agregados, grades e *Peer-to-Peer* (P2P). Logo, as funcionalidades do RJMS devem ser implementadas sobre estes simuladores para poder ser avaliado o desempenho de seus mecanismos de gerenciamento.

Neste contexto, simuladores específicos para a avaliação de RJMSs foram propostos. O Batsim é um exemplo que utiliza o SimGrid para simular o comportamento de uma grade computacional (DUTOT et al., 2017). O objetivo do Batsim é servir como uma ferramenta para avaliação de algoritmos de escalonamento e facilitar a reprodutibilidade dos experimentos. Com tal finalidade, este abstrai o comportamento do RJMS e coordena a execução dos processos na plataforma que é simulada pelo SimGrid.

Semelhante ao Batsim, o simulador Alea também permite a avaliação de algoritmos de escalonamento. Para simular o comportamento da plataforma computacional, o Alea utiliza o GridSim. Sua arquitetura segue uma abordagem modular composta por uma coleção de componentes que se comunicam através de um protocolo de troca de mensagens orientado a eventos. O RJMS implementado neste simulador é robusto e possui uma coleção de algoritmos de escalonamento tradicionais, como o *Extensible Argonne Scheduling sYstem Backfilling* (EASY) e o *First Come First Served* (FCFS) (KLUSÁČEK; RUDOVÁ, 2010). Tanto o Batsim quanto o Alea são simuladores com um objetivo específico em sua concepção, permitir experimentos com o RJMS em sistemas distribuídos. Contudo, algoritmos de RL devem primeiro ser treinados para depois serem avaliados. Deste modo, se faz necessária a implementação de abstrações que permitam a sua integração com o *framework* esperado pelos métodos de RL.

A aplicação de RL no contexto de gerenciamento de recursos e processos em sistemas distribuídos não é novidade (SCHAERF; SHOHAM; TENNENHOLTZ, 1994; KIM; LEE, 1995; ZHANG; DIETTERICH, 1995). Sobre uma perspectiva geral, trabalhos que aplicam RL em sistemas distribuídos utilizam simuladores ad-hoc, desenvolvidos especificamente para o estudo sendo conduzido. Alguns exemplos de trabalhos podem ser observados na Tabela 1. Logo, cada trabalho implementa uma abstração própria que está acoplada ao método proposto. Além de dificultar a reprodutibilidade dos experimentos, esta abordagem não é adequada para permitir uma possível comparação entre as propostas.

Analisando este cenário, a plataforma Machine Learning Box (MBox) oferece serviços de escalonamento baseados em técnicas de *Machine Learning* (ML) para sistemas distribuídos. Esta plataforma foi desenvolvida em Java e contém uma coleção de interfaces que permitem a sua aplicação em simuladores de sistemas distribuídos.

Os algoritmos de ML, incluindo técnicas de RL, são incorporados através da biblioteca BURLAP e oferecidos como serviços (ORHEAN; POP; RAICU, 2018). Contudo, devido as escolhas de implementação e o escopo da proposta, ainda é necessário a integração com um simulador de sistemas distribuídos. Não obstante, tal implementação carece de documentação e da disponibilidade do código fonte. Deste modo, o desenvolvimento de um simulador de RJMS para grades computacionais integrado ao *framework* de uma abordagem de RL se faz necessário. Observada esta necessidade, a presente seção apresenta o GridGym, uma extensão que integra o simulador Batsim ao *framework* do OpenAi Gym (BROCKMAN et al., 2016).

5.1.1 Arquitetura

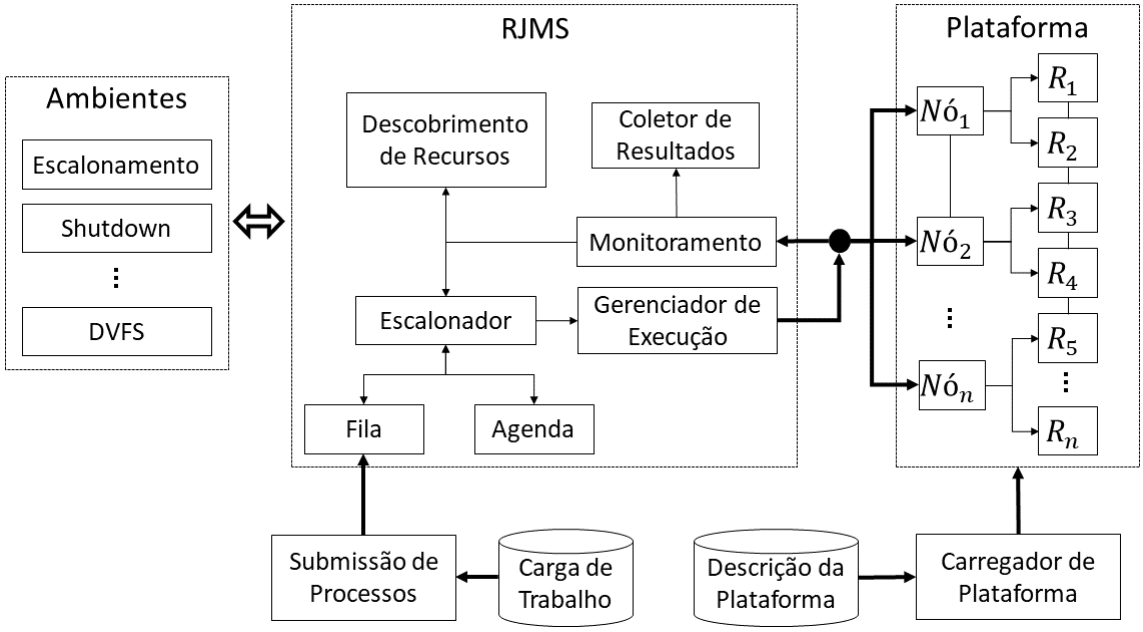
Um dos objetivos do GridGym é ser modular e extensível para cobrir e modelar os diferentes problemas encontrados no gerenciamento de recursos em sistemas distribuídos, como: escalonamento, alocação de recursos, economia de energia e entre outros. Com esta finalidade, a estrutura interna do GridGym é composta por uma coleção de componentes independentes que são coordenados por um módulo central.

Na Figura 18 é apresentado a estrutura interna do GridGym. Esta estrutura pode ser dividida em três componentes principais, sendo: Plataforma, RJMS e Ambientes. Destes componentes, o RJMS trabalha como sendo o módulo central e é responsável pela orquestração da simulação. Ambientes interagem com o simulador através do RJMS para obter informações sobre o estado do sistema e tomar decisões. Enquanto o RJMS interage com a plataforma para obter informações dos recursos e de sua topologia.

Para montar uma grade computacional virtual, a plataforma precisa de uma descrição do sistema. Nesta descrição são detalhados os recursos, contendo as suas propriedades e a composição dos nós. O formato desta descrição segue o padrão estabelecido no Batsim. Com base neste documento, o carregador de plataforma vai interpretar e montar a estrutura do sistema.

Além da descrição da plataforma, também é necessária uma descrição da carga de trabalho que vai ser utilizada na simulação. Ao invés de optar por um gerador de carga de trabalho, foi definido permitir a utilização de cargas reais. Portanto, a construção de uma carga de trabalho é feita a priori e segue o padrão definido no Batsim. Com base na carga de trabalho, o submetedor de processos vai ser responsável por simular o comportamento dos consumidores. Na carga de trabalho são definidos os tempos de submissão de cada processo, sendo esta informação utilizada para determinar o momento em que os processos devem ser submetidos ao sistema. Os processos permanecem desconhecidos pelo RJMS até a sua submissão.

Figura 18 – Principais componentes do GridGym.



Fonte: O Autor

Processos submetidos vão diretamente para a fila do RJMS. Com base nos processos da fila e na agenda, o escalonador determina como os processos são escalonados. A utilização do escalonador é opcional, sendo possível implementar um ambiente que desempenhe esta função através de métodos de RL. Contudo, o escalonador foi implementado para possibilitar a construção de ambientes que trabalham em conjunto com o escalonador. Este é o caso da estratégia adotada pelo DeepShutdown, descrito na Seção 4.2.

No interior do RJMS também são definidos componentes, para: gerenciar a execução, responsável por iniciar, controlar e terminar a execução dos processos no sistema; realizar o monitoramento, responsável por coletar informações sobre o estado dos recursos e dos processos em execução; coletar resultados, utilizado para obter métricas de desempenho durante e ao fim da simulação; e para o descobrimento dos recursos, responsável por identificar os recursos disponíveis no sistema.

Todos estes componentes são disponibilizados para a construção de ambientes através de uma interface padrão. O RJMS foi construído de um modo que este possa funcionar independentemente dos ambientes. Sendo assim, cada ambiente é livre para implementar qualquer componente do RJMS e modelar problemas distintos. Tal abordagem permite atingir diferentes níveis de granularidade no gerenciamento de recursos, pois o RJMS não está acoplado a um modo de operação específico.

5.1.2 Mecânica da Simulação

No decorrer da simulação, o módulo central é o responsável pela orquestração dos componentes. Todos os componentes interagem com este módulo seguindo uma abordagem orientada a eventos. Eventos são disparados em momentos de interesse, e enviados para a fila de mensagens do módulo central. Mensagens são consumidas sequencialmente, seguindo sua ordem de chegada. Caso não haja mensagens na fila, o módulo central permanece bloqueado. Deste modo, é possível utilizar dois modos de operação durante a simulação. O primeiro consiste em operar somente nos eventos do simulador, como quando um processo é submetido ou finalizado. Já o segundo consiste em operar periodicamente, após intervalos de tempo.

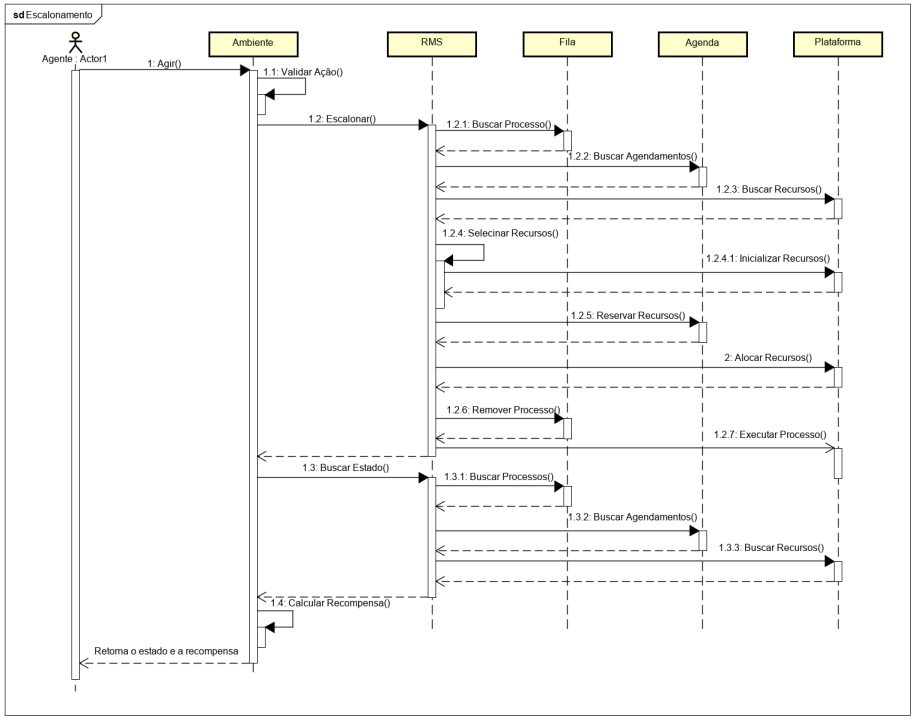
Para determinar o momento de finalizar a simulação, o componente responsável pela submissão de processos dispara um evento indicando que todos os processos da carga de trabalho já foram submetidos. Caso o módulo central não tenha outra operação agendada, a simulação é finalizada imediatamente. Caso contrário, a simulação será prolongada até que o módulo central não receba mais mensagens. Em todos os casos, é possível determinar um período de tempo máximo no qual o módulo central irá forçar a finalização da simulação, independentemente se houver operações agendadas, processos executando ou a serem submetidos. Deste modo, é possível avaliar os algoritmos com base em um intervalo de tempo fixo ou através de cargas de trabalho com tempos de simulação distintos.

Os algoritmos de RL, chamados de agentes, interagem com o GridGym através da definição de ambientes. Estes ambientes são responsáveis por modelar um problema em um *Markov Decision Process* (MDP). Com esta finalidade, os ambientes interagem com os componentes do RJMS e podem substituir algumas de suas funções. Na Figura 19 é apresentado um diagrama de sequência que descreve como esta integração, entre ambiente e RJMS, é conduzida. Neste exemplo, o ambiente modela o problema de escalonamento, sendo o agente responsável por escolher quais processos devem ser executados.

Observando a Figura 19, a interação é iniciada com a execução de uma ação pelo agente. Baseado nesta ação, o ambiente faz a validação e submete para o RJMS. Quando a ação é recebida pelo RJMS, todas as operações envolvidas no escalonamento são realizadas, com exceção da escolha de um processo da fila que é realizada pelo agente. Estas operações consistem em interagir com a fila, a agenda e a plataforma para obter os seus estados atuais. Com base nestas informações, o RJMS então seleciona os recursos usando um algoritmo de *matchmaking*.

As operações de selecionar, inicializar, reservar e alocar recursos são realizadas de modo independente pois os recursos podem estar desligados. Neste caso,

Figura 19 – Diagrama de sequência da interação entre um ambiente e o RJMS.



Fonte: O Autor

antes de serem alocados, eles precisam ser inicializados. Enquanto os recursos são inicializados, o processo não é retirado da fila. Caso os recursos já estejam disponíveis, o RJMS então remove o processo da fila e inicia sua execução.

Após o RJMS finalizar o escalonamento, o ambiente atualiza suas informações sobre o estado do sistema (fila, agenda e plataforma), calcula a recompensa para o agente e espera por uma nova tomada de decisão. Todo este procedimento é repetido até que a simulação seja finalizada. Neste caso, quem vai determinar os eventos que finalizam a simulação é o próprio ambiente. O procedimento padrão é terminar a simulação somente após o último processo da carga de trabalho finalizar a sua execução.

5.2 CONFIGURAÇÃO DOS EXPERIMENTOS

Nesta seção é descrito a metodologia utilizada para avaliar o desempenho dos métodos propostos. Inicialmente, as plataformas e cargas de trabalho utilizadas nos experimentos são apresentadas para, então, serem introduzidas e justificadas as métricas de avaliação. Por último, por se tratar de abordagens distintas, os ambientes de simulação do DeepShutdown e do DeepScheduler são detalhados de modo independente. Cada método, possui uma configuração de simulação específica.

5.2.1 Plataforma & Cargas de Trabalho

Para treinar e avaliar o comportamento dos métodos propostos, os *traces* da GRID'5000 apresentados na Figura 11 são utilizados como cargas de trabalho. Para permitir uma melhor discussão acerca dos resultados, estas cargas de trabalho foram separadas por dia e agrupadas em cinco grupos com base na ocorrência de submissões sequenciais. Na Tabela 3 é apresentado o resultado deste agrupamento, sendo os grupos de 1-4 compostos por cargas que possuem uma taxa de submissão de processos sequenciais entre $(0, 25\%)$, $[25\%, 50\%)$, $[50\%, 75\%)$ e $[75\%, 100\%)$, respectivamente, e o grupo 5 possui cargas que somente contém submissões sequenciais.

Tabela 3 – Número de cargas de trabalho para cada grupo definido com base na taxa de ocorrência de submissões sequenciais.

	Grupos				
	#1	#2	#3	#4	#5
Econome	259	520	504	271	39
Graphite	487	705	427	149	24
Orion	659	777	496	129	28
Taurus	441	947	612	130	22

As cargas apresentadas na Tabela 3 somente incluem os dias em que foram submetidos ao menos dois processos. O tempo nos *traces* é convertido de segundos para minutos, para diminuir o tempo de simulação. Não obstante, cada carga de trabalho é simulada de modo independente. Deste modo, a simulação ocorre em forma de episódios, no qual cada episódio representa um dia de operação do agregado e possui um total de 1440 passos. O simulador é reiniciado após cada simulação de modo que o comportamento em uma carga de trabalho não interfira nas demais.

Como consequência da utilização de cargas de trabalho reais, a plataforma simulada possui a mesma configuração e arquitetura de *hardware* dos seus respectivos agregados. As diferenças na capacidade computacional dos recursos de cada agregado são desconsideradas. Deste modo, a diferença entre as plataformas reside somente na quantidade total de recursos disponíveis e na composição dos nós, que determina quantos recursos cada nó possui. Por simplicidade, o perfil de consumo de energia dos recursos é o mesmo em cada plataforma e segue o perfil ilustrado na Figura 12, que foi apresentada na Seção 4.1.1.

O tamanho de cada plataforma é definido com base na quantidade de núcleos de cada um dos agregados selecionados, apresentados na Tabela 2. Já a quantidade de recursos para cada nó da plataforma é obtida através da distribuição dos núcleos, cada nó possui a mesma quantidade de núcleos. Deste modo, a simulação

da plataforma mantém as características das plataformas da GRID'5000[1] e permite a reprodução do comportamento das cargas de trabalho.

5.2.2 Métricas de Avaliação

O desempenho do DeepShutdown e do DeepScheduler é avaliado com base em diferentes métricas. Cada métrica provê uma análise sobre uma perspectiva, como a do administrador ou a do usuário. Com base na definição de processos de um sistema em grades apresentado na Seção 2.3.1, as métricas utilizadas para avaliação, são:

Desperdício de Energia

Definido na Equação 4.4, a quantidade total de energia desperdiçada tem um impacto direto na eficiência energética de cada política. Quando recursos estão ociosos ou transitando entre desligado e ligado, podem ser considerados inúteis para os provedores, pois não estão servindo um usuário ou economizando energia. Logo, o consumo de energia advindo desta situação é considerado um desperdício e sua minimização é de maior interesse dos provedores.

Quantidade de Transições

O objetivo desta métrica é avaliar a agressividade das políticas de *shutdown* e como o consumo de energia está relacionado ao número de reinicializações de recursos. Um número alto de transições indica que os recursos estão sendo desligados e/ou ligados com frequência. Tal comportamento não é desejado, pois pode ser prejudicial à saúde dos componentes de hardware. Não obstante, múltiplas inicializações simultâneas podem resultar na criação de pontos de calor, aumentando a probabilidade de erros nas aplicações em execução. Esta também pode ser considerada uma métrica de interesse dos provedores, pois usuários desejam executar as suas aplicações o mais rápido possível.

Tempo de Espera

De modo a balancear esta equação, métricas de interesse dos usuários também são consideradas. Sobre esta perspectiva, a primeira métrica utilizada é a média do tempo de espera dos processos. Formalmente, está métrica é definida na Equação (2.2) e pode ser considerada um indicador do desempenho do sistema.

[1] Informações sobre o *hardware* podem ser encontrados em <https://www.grid5000.fr/w/Hardware>

Slowdown

Um problema ao utilizar o tempo médio de espera dos processos na fila está em considerar que todos os processos possuem as mesmas propriedades. Logo, está métrica não serve como um indicativo do nível de justiça das políticas utilizadas no sistema. Com esta finalidade, uma métrica comumente utilizada é o *slowdown* (CARASTAN-SANTOS et al., 2019).

Formalmente definida na Equação (2.6), além do tempo de espera, o *slowdown* leva em consideração o tempo real de execução dos processos. Logo, processos com um tempo menor de execução terão maior influência em relação aos processos mais longos. Este comportamento ocorre, pois, o *slowdown* considera que processos curtos devem possuir um tempo de espera menor que processos longos. Portanto, pode ser considerada como uma métrica que quantifica a justiça do sistema.

Em detrimento do *slowdown*, também é possível utilizar o *slowdown* delimitado para medir o nível de justiça a média. Em algumas situações, é aconselhável utilizar esta métrica para minimizar a influência de processos com tempos de execução muito curtos (*e.g.* alguns segundos). Contudo, como a carga de trabalho utilizada neste trabalho foi previamente tratada, sendo o tempo convertido para minutos, não é necessário delimitar o *slowdown*.

Per-Processor Slowdown

A estratégia do *slowdown* é utilizar o tempo real de execução dos processos para normalizar o tempo de espera. Por este motivo, está métrica é mais eficiente que o tempo de espera para determinar o nível de justiça do sistema. Contudo, outras propriedades dos processos são ignoradas, como o número de recursos que são alocados. No problema de escalonamento em grades computacionais, processos podem requisitar um número arbitrário de recursos. Logo, não é justo tratar um processo que alocou todo o agregado do mesmo modo que um processo que pediu por somente um recurso (CARASTAN-SANTOS et al., 2019). Com esta finalidade, o *Per-processor Slowdown* (pp-sld), formalmente definido na Equação 2.9, pode ser utilizado em detrimento do *slowdown*.

Atraso

O modo em como o pp-sld é formulado pode fazer com que está métrica seja considerada uma boa medida da justiça das políticas de escalonamento. Contudo, está métrica pode não ser totalmente adequada para analisar o desempenho de uma política de *Off Reservation* (OR). Políticas de OR estão fundamentadas na ideia que é permitido atrasar a execução dos processos para economizar energia. Como con-

sequência, provavelmente o *slowdown* de uma política de OR será igual ou pior que o de uma política de *timeout*.

De modo a permitir uma comparação justa, é utilizado uma métrica que retorna o atraso na execução $delay_j$ de um processo j. Está métrica é formalmente definida na Equação 5.1.

$$delay_j = \begin{cases} wait_j - (wall_j \times \theta) & \text{if } \frac{wait_j}{wall_j} \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

O atraso na execução de um processo é um indicativo da quantidade de tempo em que um processo foi forçado a ficar em espera além do tempo permitido. Neste caso, é considerado que os processos podem ser atrasados até um limite máximo de tempo. Logo, o atraso na execução também serve como um indicativo da agressividade da política de OR.

Stretch

Os processos no sistema possuem um tempo real de execução desconhecido pelo RJMS. Logo, uma política de gerenciamento que considere o tempo esperado de execução, fornecido pelo usuário na submissão do processo, pode ser incorretamente avaliada por uma métrica que considere somente o seu tempo real. Por este motivo, também é levado em consideração o *stretch* de cada processo. Formalmente, o *stretch* é definido na Equação 5.2.

$$stretch_j = \frac{wait_j}{wall_j} \tag{5.2}$$

A principal motivação pelo uso do *stretch* está relacionado ao fato de uma política de OR considerar o tempo esperado de execução $wall_j$ de um processo j para avaliar se este pode ser atrasado. Não obstante, processos com um tempo real de execução curto podem influenciar negativamente o *slowdown*, penalizando uma política mesmo se o seu tempo de espera for relativamente baixo.

5.2.3 Ambiente do DeepShutdown

O desempenho do DeepShutdown é comparado com três políticas de *timeout* e uma política ideal de OR. As políticas de *timeout* desligam os recursos com diferentes tempos de ociosidade, sendo $t = [0, 1,$ e 5 minutos$]$. Estes tempos de ociosidade foram definidos de modo a explorar diferentes níveis de agressividade e avaliar como as métricas consideradas são impactadas. Não obstante, esta estratégia também permite avaliar o comportamento da política aprendida pelo DeepShutdown ao verificar a sua agressividade no desligamento dos recursos.

Para avaliar os ganhos possíveis de serem obtidos com uma política de OR que respeite os limites definidos nas métricas consideradas, também é utilizado uma política ideal de OR que conhece o tempo real de execução dos processos. Logo, esta política pode ser considerada fictícia pois tal informação não é disponibilizada. Contudo, com base nesta informação é possível determinar os momentos ideias para atrasar os processos na fila. A quantidade de recursos reservados por esta técnica só será reduzida quando os processos não podem ser atrasados, devido aos limites pré-estabelecidos. Assim, também é possível avaliar o quanto o DeepShutdown foi capaz de economizar energia utilizando somente o tempo esperado de execução fornecido na submissão dos processos.

A escolha por estas políticas oferece uma representação adequada das políticas de *shutdown* implementadas em sistema em produção (RAÏS; ORGERIE; QUINSON, 2016). De modo a realizar uma comparação justa, a mesma política de escalonamento e parâmetros são utilizados em todos os experimentos. A política de escalonamento implementada é o *Smallest estimate Area First* (SAF), que utiliza o mesmo mecanismo de *backfilling* do EASY definido no Algoritmo 1. Contudo, neste caso a fila de processos que é utilizada para realizar o *backfilling* é ordenada de modo ascendente com base no cálculo da área de cada processo. Formalmente, o cálculo desta área é definido na Equação 5.3 (CARASTAN-SANTOS et al., 2019).

$$f(j) = wall_j \times q_j \tag{5.3}$$

Em relação aos parâmetros utilizados no DeepShutdown, é instanciado 16 *actors* em paralelo para coletar experiências. O fator de desconto (γ) é definido em 0,99 enquanto o parâmetro GAE (λ) é definido em 0,95. O parâmetro ϵ da função objetivo (Equação 3.10) é 0,20. O algoritmo é treinado por 50 milhões de passos enquanto a política é otimizada a cada 1440 passos por 4 épocas. Deste modo, é possível dizer que o algoritmo é treinado após o fim de um dia de operação.

As observações do ambiente incluem o histórico das últimas 20 observações. De modo a limitar o tamanho da fila, é determinado que todas as políticas podem visualizar somente os primeiros 10 processos da fila. O parâmetro τ da função de recompensa é 0,50. O tempo de simulação é fixado em 1440 minutos, independe se ainda houverem processos a serem submetidos ou a serem finalizados. Cada política recebe a mesma observação do ambiente, a A única exceção é a política ideal de OR que utiliza o tempo real de execução dos processos. A cada simulação, o ambiente é completamente reiniciado de modo a não interferir nas próximas simulações.

Para verificar a generalização da política aprendida pelo DeepShutdown, os *traces* utilizados são divididos em dois grupos. Aproximadamente 80% dos dados são utilizados na fase de treinamento enquanto que os demais são utilizados somente na

avaliação. Durante a fase de treinamento, o agente seleciona uma carga de trabalho do grupo selecionado de modo aleatório, seguindo uma distribuição uniforme. Assim, o risco de sobre-ajuste à uma carga de trabalho específica é minimizado.

5.2.4 Ambiente do DeepScheduler

Para avaliar o desempenho da política encontrada pelo DeepScheduler, são utilizadas três políticas de escalonamento baseadas em regras, sendo: o First-Fit, que seleciona o primeiro processo da fila que pode ser iniciado imediatamente, desde que tenha recursos disponíveis; o EASY, previamente descrito na Seção 2.3.2; e o SAF, descrito na Seção 5.2.3.

Cada política de escalonamento, incluindo o DeepScheduler, possui a mesma visão do sistema. Para desligar os recursos é utilizado uma política de *timeout* com tempo de ociosidade definido em 5 minutos. A cada tomada de decisão, o tempo de simulação avança em um minuto ou, caso não haja processos na fila, até o próximo evento (*e.g.* submissão de um processo ou o término do desligamento de um recurso). Diferente da abordagem utilizada no DeepShutdown, no problema de escalonamento a simulação só termina quando todos os processos da carga de trabalho executarem.

A escolha de parâmetros do DeepScheduler é similar às escolhas do DeepShutdown. Na Tabela 4 é possível observar uma comparação entre os seus parâmetros. No processo de treinamento são instanciados 16 *actors* em paralelo, cada um com uma instância do ambiente de simulação. O parâmetro ϵ da função objetivo (Equação 3.10) é 0,20. O fator de desconto (γ) é definido em 0,97 enquanto o parâmetro GAE (λ) é definido em 0,95. O valor menor definido no fator de desconto está relacionado à influência das decisões de escalonamento futuras. As cargas de trabalho de sistemas em grades podem mudar de comportamento repentinamente. Logo, para minimizar os efeitos negativos desta dinamicidade, foi definido priorizar as decisões recentes.

Tabela 4 – Comparação entre os parâmetros de treinamento do DeepShutdown e do DeepScheduler.

	# *Actors*	γ	λ	ϵ	# Atualizações	# Épocas	Tamanho do *Batch*
DeepShutdown	16	0,99	0,95	0,20	34.722	4	1440
DeepScheduler	16	0,97	0,95	0,20	69.444	6	180

O algoritmo é treinado por 25 milhões de passos enquanto a política é otimizada a cada 360 passos em lotes de 180 experiências por 6 épocas, resultando em um total de 69.444 atualizações. Em relação aos pesos da função de recompensa (Equação 4.7), tanto o parâmetro ρ como o σ são definidos em 1 e o parâmetro τ é definido em 0,2. O valor do parâmetro τ é definido com base no tempo de ociosidade definido na política de *timeout*. Deste modo, decisões de escalonamento que resultem

na inicialização de recursos serão recompensadas caso o processo, que foi alocado, tenha um tempo de execução igual ou superior ao tempo de ociosidade.

5.3 RESULTADOS OBTIDOS

Ambos os métodos propostos neste trabalho objetivam minimizar o desperdício de energia. Contudo, cada abordagem atua sobre um nível de controle do RJMS. O DeepShutdown realiza o controle dos recursos, no nível de plataforma, enquanto o DeepScheduler atua no escalonamento. Deste modo, cada abordagem possui um MDP específico, impossibilitando a comparação entre seus resultados. Portanto, no decorrer desta seção os resultados obtidos com o DeepShutdown são apresentados primeiro para, em seguida, serem apresentados os resultados obtidos com o DeepScheduler.

5.3.1 DeepShutdown

Na Figura 20 é apresentado o desperdício de energia acumulado, por dia, do DeepShutdown em comparação com às demais políticas para cada um dos grupos de cargas de trabalho avaliadas. Os resultados foram normalizados utilizando como base o pior resultado obtido. Cada gráfico contém uma linha pontilhada vertical, separando os resultados obtidos com os dados de treinamento e de avaliação. O lado esquerdo apresenta os resultados para o conjunto de dados de treinamento enquanto o lado direito apresenta os resultados para o conjunto de dados de avaliação. Deste modo, é possível observar o comportamento do DeepShutdown sobre dados desconhecidos e analisar se o algoritmo é capaz de manter o desempenho esperado. Não obstante, em razão da política do DeepShutdown ser probabilística, os seus resultados são obtidos através da média de 4 repetições do mesmo experimento.

Analisando a Figura 20 é possível observar que as políticas de *timeout* são menos eficientes em termos de energia na medida que o tempo de ociosidade determinado na política aumenta. Uma política de *shutdown* agressiva (T(0)), que desliga os recursos ociosos imediatamente, apresenta uma economia de energia consideravelmente maior que uma política suave (T(5)). Contudo, a mesma observação não pode ser visualizada quando a comparação é realizada com uma estratégia de OR.

Em todas as cargas de trabalho, a política ideal de OR (OR*) e o DeepShutdown (DS) conseguiram economizar mais energia. Não obstante, em algumas cargas de trabalho, o DeepShutdown conseguiu ultrapassar o desempenho do OR*. Esta observação fica mais evidente nas cargas de trabalho dos agregados Orion e Graphite, que possuem as menores quantidades de recursos, e nas cargas dos grupos 4 e 5, que possuem a maior taxa de submissões sequenciais de processos.

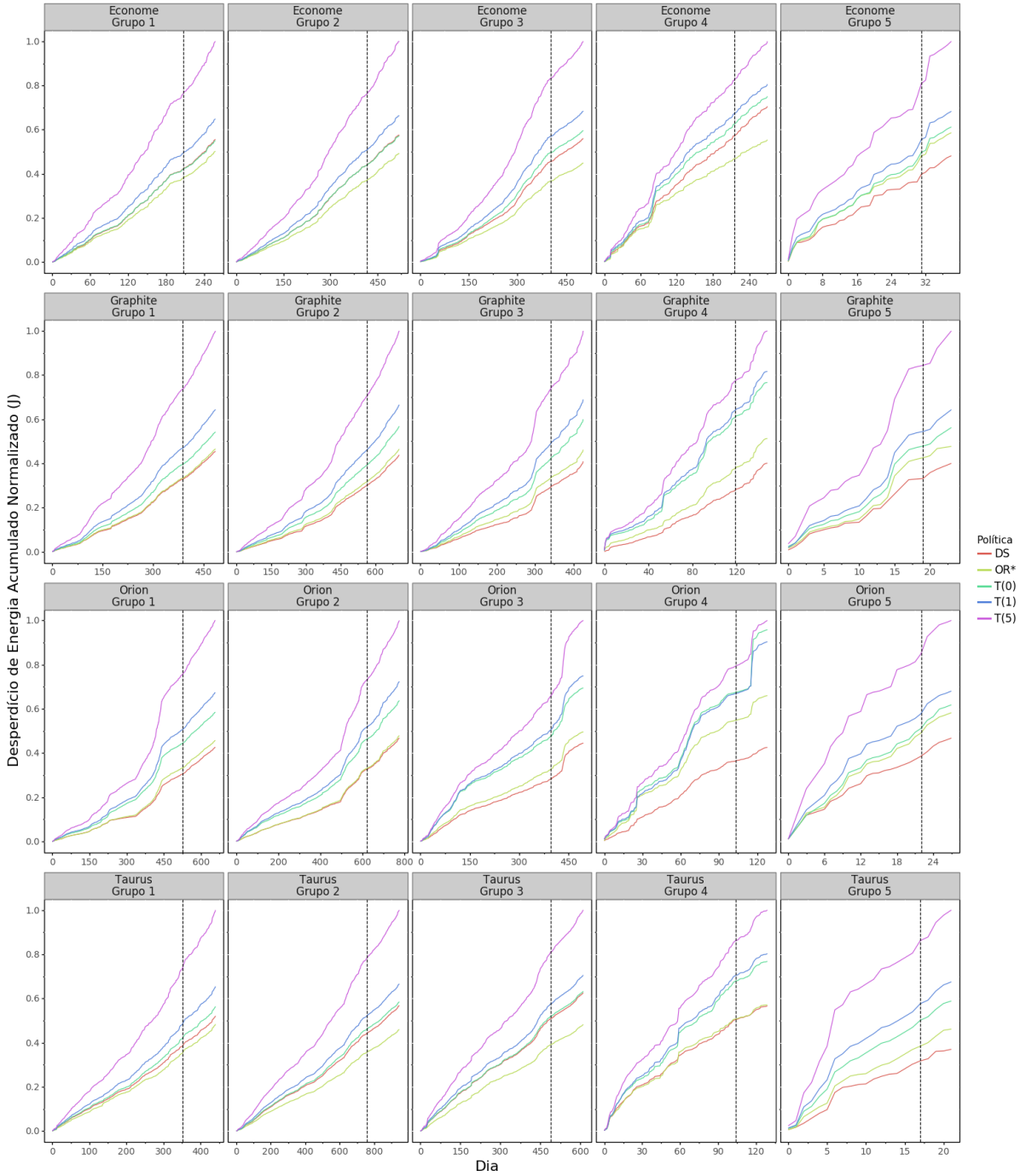


Figura 20 – Desperdício de energia acumulado e normalizado para cada grupo de carga de trabalho analisado.

Em todos os agregados, o DeepShutdown atingiu a melhor eficiência energética nas cargas do grupo 5. Contudo, nos grupos 1 e 2 do Econome e nas cargas do grupo 3 do Taurus, o desempenho do DeepShutdown é similar ao da política T(0). Este comportamento é um indicativo que o DeepShutdown está ignorando a métrica de *Quality-of-Service* (QoS) para alcançar uma maior economia de energia, principalmente nas cargas do grupo 5 e nas cargas do agregado Orion, pois o seu desempenho médio está consideravelmente superior ao desempenho da política OR*.

Analisando o desempenho do DeepShutdown nos dois conjuntos de cargas de trabalho, de treinamento e avaliação, não é possível observar uma perda de desempenho. Em algumas cargas de trabalho, como no grupo 4 do Orion e no grupo 5 do Taurus, o DeepShutdown apresentou melhores resultados nos dados de avaliação em comparação com às demais políticas. Contudo, nas cargas do Grupo 5 de Graphite, o DeepShutdown teve um desempenho pior em comparação ao OR*, mas obteve comportamento similar à política T(0). Logo, de um modo geral, é possível observar que o DeepShutdown conseguiu generalizar para lidar com cargas de trabalho desconhecidas, sem quedas abruptas no desempenho.

Analisando o desempenho médio por grupo, nas cargas do grupo 1 a política OR* economizou 2,5% mais energia em comparação ao DeepShutdown. Nesta mesma carga, o DeepShutdown economizou 13%, 25,4% e 51% mais energia em relação às políticas de *shutdown* T(0), T(1) e T(5), respectivamente. Um comportamento semelhante é observado também nas cargas do grupo 2. A política OR* conseguiu economizar 12,5% mais energia em relação ao DeepShutdown. O DeepShutdown teve uma queda de desempenho considerável nas cargas do Taurus deste grupo, justificando o aumento na diferença observada na comparação com o OR*. Contudo, o seu desempenho ainda foi superior que as demais políticas de *shutdown*.

Nas cargas do grupo 3, o DeepShutdown teve uma queda no desempenho mais acentuada nos agregados Taurus e Econome, os dois maiores em número de recursos, em comparação ao desempenho da política OR*. Como consequência, a política OR* apresentou um ganho de 18% no desempenho médio em relação ao DeepShutdown, mesmo perdendo por uma leve margem nos resultados dos agregados de Orion e Graphite. Em relação às demais políticas, o DeepShutdown economizou 10,8% mais energia que a política de *shutdown* mais agressiva T(0) e 44% em relação a política mais suave T(5).

Analisando os resultados obtidos nas cargas do grupo 4, é possível observar novamente um comportamento similar aos resultados obtidos nas cargas dos grupos com uma menor ocorrência de submissões de processos sequenciais. O DeepShutdown teve uma queda de desempenho considerável nas cargas do Econome, em relação ao OR*. Considerando os demais agregados, o DeepShutdown teve um desempenho médio superior a todas as políticas de *shutdown*. Em virtude do agregado Econome possuir a maior quantidade de recursos, resultando em um maior consumo de energia, o desempenho do OR* foi 12% superior ao DeepShutdown. Contudo, a diferença entre o DeepShutdown e as demais políticas de *shutdown* também aumenta. Em comparação a melhor política de *timeout* T(0), em termos de energia, o DeepShutdown foi capaz de economizar 18% mais energia.

Considerando que a ocorrência de submissões sequencias de processos nas

cargas de trabalho aumenta conforme o grupo em que estão agrupadas, é compreensível o aumento recorrente no desempenho do DeepShutdown em relação às políticas de *timeout*. Tal afirmação é reforçada pelos resultados obtidos nas cargas do grupo 5, sendo o DeepShutdown melhor que todas as demais políticas. Nestas cargas de trabalho, o DeepShutdown obteve uma redução média no consumo de energia de 18% em comparação ao OR*. Não obstante, foi possível observar um ganho de 26,7%, 34,8% e 56,1% em relação às políticas de *timeout* T(0), T(1) e T(5), respectivamente.

Analisando os resultados sobre outra perspectiva, na Tabela 5 é apresentado a média dos resultados por agregado em conjunto com algumas propriedades estatísticas. É possível observar, novamente, que o DeepShutdown apresenta a melhor eficiência energética em comparação às políticas de *timeout*. Comparado ao T(0), o DeepShutdown obteve uma redução média de 2,9% até 30% no desperdício de energia ao mesmo tempo que minimizou a quantidade de transições de estados dos recursos de 11.6% até 33.8%. Em relação à política mais suave T(5), foi possível observar uma diferença ainda superior. O DeepShutdown obteve uma redução de 40,6% até 56,5% no desperdício de energia com uma média de quantidade de transições similar. Considerando somente as cargas de trabalho de Graphite e Orion, é possível observar que o DeepShutdown obteve uma menor quantidade de transições que a política T(5), de 11% e 8.5%, ao mesmo tempo que minimizou o desperdício em 56,5% e 55,4%, respectivamente.

Tabela 5 – Desempenho do DeepShutdown (DS) em comparação às demais políticas analisadas.

		Atraso (min)				Desperdício de Energia (J)				# Transições				*Stretch* (min)			
		avg	std	min	max	avg	std	min	max	avg	std	min	max	avg	std	min	max
Econome	DS	**48,50**	116,77	0,0	1002,00	21738,40	20493,94	125,0	199485,5	93,50	80,38	1,0	797,00	0,22	1,03	0,0	26,68
	OR*	53,16	124,34	0,0	1002,00	**18076,86**	19474,91	125,0	223416,0	**84,96**	86,38	1,0	1044,00	0,21	1,07	0,0	26,68
	T(0)	49,69	123,29	0,0	1002,00	22392,20	23398,09	250,0	271352,0	106,12	106,36	2,0	1268,00	0,15	1,00	0,0	26,68
	T(1)	49,54	123,24	0,0	1002,00	25483,59	25960,63	250,0	329490,0	100,21	97,68	2,0	1260,00	0,15	0,99	0,0	26,68
	T(5)	49,48	123,54	0,0	1002,00	36624,22	32931,17	250,0	307125,0	85,35	72,73	2,0	720,00	**0,14**	0,99	0,0	26,68
Graphite	DS	**50,92**	116,47	0,0	770,00	**6712,60**	6219,47	125,0	51549,5	**30,63**	27,94	1,0	238,00	0,41	1,38	0,0	22,88
	OR*	54,27	123,59	0,0	770,00	7216,18	6587,65	125,0	72855,0	33,92	30,36	1,0	340,00	0,30	1,39	0,0	29,28
	T(0)	52,81	125,29	0,0	770,00	8950,24	8287,19	125,0	89024,0	42,20	38,47	1,0	416,00	0,26	1,41	0,0	29,28
	T(1)	52,99	125,43	0,0	770,00	10401,94	9381,74	125,0	87820,0	40,35	35,84	1,0	350,00	0,26	1,40	0,0	29,28
	T(5)	52,91	125,40	0,0	770,00	15434,66	13357,23	125,0	87452,0	34,48	28,82	1,0	208,00	**0,25**	1,41	0,0	29,35
Orion	DS	72,27	132,62	0,0	782,67	**6284,59**	6930,93	125,0	70165,0	**28,07**	30,69	1,0	291,25	0,54	2,10	0,0	66,24
	OR*	70,34	136,79	0,0	782,67	6763,39	7551,15	125,0	67003,0	31,84	34,79	1,0	302,00	0,41	2,04	0,0	63,95
	T(0)	67,23	139,61	0,0	782,67	8996,35	9892,78	250,0	146727,0	42,41	45,92	2,0	687,00	**0,30**	1,47	0,0	34,19
	T(1)	**67,14**	139,48	0,0	782,67	10081,97	10864,42	220,0	108591,0	38,95	40,81	1,0	356,00	**0,30**	1,47	0,0	34,19
	T(5)	67,39	139,76	0,0	782,67	14104,64	15307,39	220,0	125931,0	30,69	33,36	1,0	295,00	**0,30**	1,47	0,0	34,19
Taurus	DS	**48,31**	106,29	0,0	1149,00	16513,70	12811,54	125,0	134672,0	72,53	54,15	1,0	584,00	0,30	1,48	0,0	28,54
	OR*	53,60	113,04	0,0	1149,00	**13747,39**	12307,84	125,0	194121,0	**62,43**	53,19	1,0	904,00	0,29	1,46	0,0	29,51
	T(0)	49,99	114,82	0,0	1149,00	17441,70	14984,53	375,0	264884,0	82,10	68,27	3,0	1236,00	**0,22**	1,45	0,0	29,51
	T(1)	50,02	114,75	0,0	1149,00	19664,81	16836,47	375,0	271768,0	76,56	63,23	3,0	1032,00	**0,22**	1,44	0,0	28,72
	T(5)	50,23	115,25	0,0	1149,00	28714,38	22920,56	375,0	256368,0	65,29	49,51	2,0	612,00	**0,22**	1,44	0,0	29,12

Levando em consideração somente a política OR*, é possível observar que existem mais possibilidades para aumentar a eficiência energética atrasando a execução de alguns processos. O DeepShutdown apresentou melhor desempenho somente nas cargas de Graphite e Orion, cujo são os menores agregados em quantidade de recursos. Nas demais cargas de trabalho, a política OR* obteve uma redução no consumo de energia 20% superior à política aprendida pelo DeepShutdown enquanto obteve uma quantidade de transições menor em 10–16,1% Não obstante, a política

do OR* resultou em uma menor quantidade de transições em comparação a política de *timeout* mais suave T(5), alcançando uma economia de energia consideravelmente superior. Tais resultados apontam que, se for permitido atrasar a execução de alguns processos, uma abordagem de OR pode minimizar o desperdício de energia ao mesmo tempo em que possui um comportamento similar a uma política suave de *timeout*, em relação a quantidade de transições.

Observando os resultados sobre a perspectiva dos usuários, é possível notar que o DeepShutdown possui uma política mais agressiva que a política do OR*. No pior caso, a média de tempo do *stretch* aumentou em 36.6%. Contudo, estes valores permanecem inferiores ao valor de corte definido (descrito na Seção 5.2.3) em quase todas as cargas de trabalho. Em relação à política T(0), o Deepshutdown aumentou entre 46.6% e 80% o tempo médio do *stretch*. Estes resultados apontam o *trade-off* entre a minimização do consumo de energia e a maximização do desempenho. O DeepShutdown aumenta o *stretch* para obter uma maior economia de energia.

Não surpreendente, todas as políticas de *timeout* apresentaram resultados similares no *stretch*. Comportamento esperado pois não são uma abordagem de OR, e, portanto, não atrasam os processos deliberadamente. A pequena diferença observada entre estas políticas está relacionada ao tempo requerido para inicializar um recurso. Este comportamento também é possível observar no tempo de atraso dos processos. Em quase todas as cargas, o atraso médio foi inferior aos resultados do DeepShutdown. Não obstante, é possível observar que o DeepShutdown possui um tempo de atraso médio inferior ao tempo obtido com a política OR*. Tal comportamento indica que a política aprendida pelo DeepShutdown não está forçando o atraso dos processos por longos períodos de tempo após violar a QoS definida. Em outras palavras, o DeepShutdown está tentando minimizar o efeito negativo de uma abordagem de OR.

5.3.2 DeepScheduler

Na Figura 21 é apresentado o desperdício de energia acumulado e as médias acumuladas do tempo de espera e do pp-sld por dia para cada uma das cargas avaliadas. Como a política do DeepScheduler é probabilística, os seus resultados são a média de 10 repetições do mesmo experimento. A linha sólida horizontal nos gráficos representa a média/valor acumulado de cada uma das respectivas métricas enquanto que as linhas verticais separam os resultados obtidos com o conjunto de cargas de trabalho de treinamento (no lado esquerdo) e com o de avaliação (no lado direito). Para minimizar o efeito de valores atípicos nas métricas orientadas aos processos (tempo de espera e pp-sld), somente os resultados que estão entre os 10-90 percentis são considerados. Não obstante, os valores máximos e mínimos são representados pelas linhas pontilhadas horizontais.

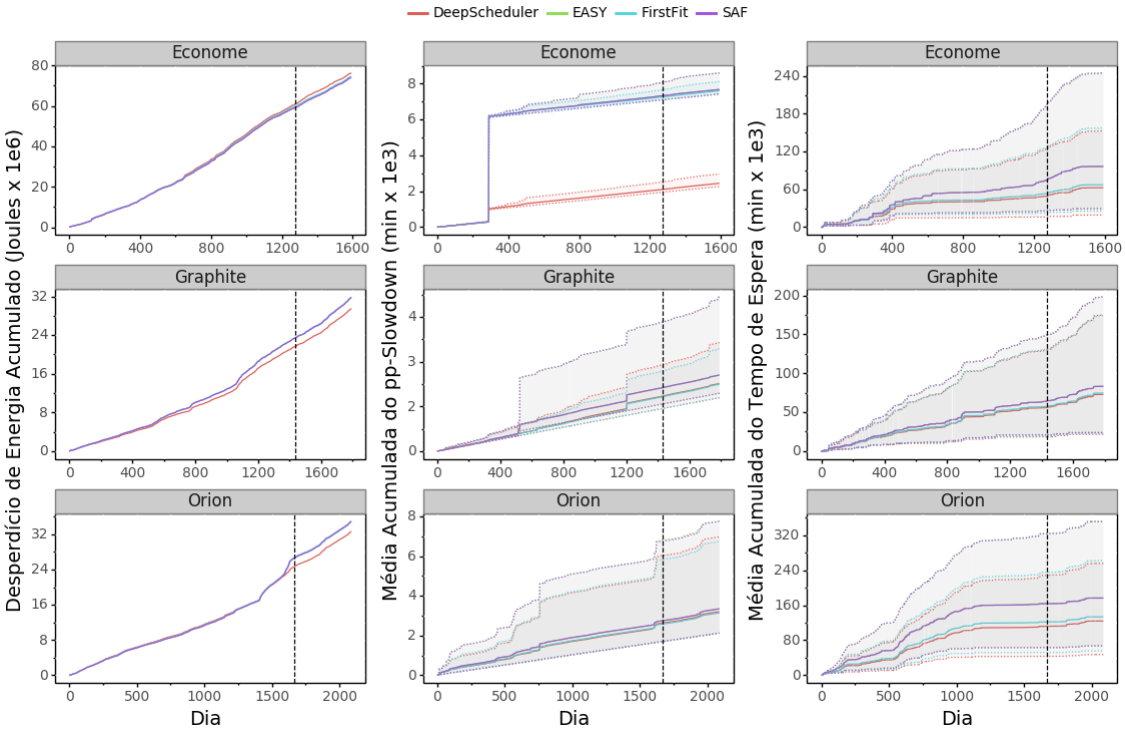


Figura 21 – Desperdício de energia acumulado e a média acumulada do tempo de espera e do pp-sld por dia.

Observando a Figura 21, o DeepScheduler foi capaz de atingir melhores coeficientes energéticos em quase todos as cargas de trabalho quando comparado às outras políticas de escalonamento. Nas cargas de Orion e Graphite, o DeepScheduler economizou 6,5–7,0% mais energia, respectivamente, enquanto que nas cargas de Econome obteve um acréscimo de 2,9% no desperdício de energia, quando comparado ao EASY e SAF. Como era esperado, a eficiência energética das políticas SAF, EASY e First-Fit é similar. Estas políticas não estão cientes do consumo de energia, portanto não é esperado uma mudança drástica de comportamento nesta métrica.

Comparando a média do tempo de espera, o DeepScheduler possui um comportamento similar à política First-Fit. Tanto as políticas SAF e EASY apresentaram resultados fracos e o First-Fit apresenta ser a melhor opção nas cargas de trabalho utilizadas. O DeepScheduler conseguiu aprender uma política que apresenta resultados ainda melhores que o First-Fit. Considerando a média, o DeepScheduler minimizou o tempo de espera médio em 5,8%, em comparação ao First-Fit, e em 27% em comparação ao SAF e EASY. Deste modo, é possível observar que utilizando uma política específica para o sistema é possível economizar energia sem perder desempenho, em termos de tempo de espera. Nos agregados Graphite e Orion, o DeepScheduler foi capaz de reduzir o desperdício de energia e, ao mesmo tempo, aumentar o desempenho do sistema.

Considerando o nível de justiça de cada política, é possível observar uma

drástica queda de desempenho das políticas SAF e EASY nas cargas do Econome. A principal justificativa por este comportamento está relacionada ao tipo de ordenamento da fila dos processos utilizados nestas políticas. Como ambas utilizam a política de ordenamento *First In First Out* (FIFO), caso o primeiro processo da fila requisitar todos os recursos do sistema por um longo período de tempo (*e.g.* 20 horas), o *slowdown* dos demais processos irá inflar esta métrica. Deste modo, para contornar esta situação é necessário mudar a política de ordenamento da fila e implementar outros mecanismos para prevenção de uma possível inanição, que ocorre quando um processo é atrasado indefinidamente. Tal implementação está além deste trabalho. Não obstante, nas cargas do Econome existem somente 9 dias que possuem uma carga superior à 300% da capacidade da plataforma, representando somente 1% de todos as cargas de trabalho. Logo, estes dias podem ser considerados valores atípicos. Contudo, foi decidido manter estas cargas para observar se o DeepScheduler seria capaz de contornar esta situação sem qualquer trabalho manual. De fato, o desempenho do DeepScheduler foi melhor no Econome enquanto apresenta um nível de justiça similar ao First-Fit nas demais cargas de trabalho.

5.4 DISCUSSÃO

Um aspecto importante relacionado a métodos treinados com *Deep Reinforcement Learning* (DRL) está em compreender o comportamento que foi aprendido. Com esta finalidade, esta seção apresenta algumas observações sobre as políticas aprendidas do DeepShutdown e do DeepScheduler.

5.4.1 A Política do DeepShutdown

A ideia central de uma abordagem de OR está em explorar as propriedades das cargas de trabalho de modo a encontrar os momentos que compensam atrasar a execução dos processos para aumentar a eficiência energética. Especificamente, é esperado que o algoritmo atrase as submissões sequenciais de processos que podem forçar reinicializações desnecessárias dos recursos que foram alocados. De modo a validar esta ideia, na Figura 22 são analisados os valores de *slowdown* dos processos em função do seu tempo de chegada, para cada uma das cargas de trabalho. Para reduzir o efeito de valores atípicos, uma transformação logarítmica é aplicada e os processos são agrupados de acordo com seu tempo de chegada.

Observando a Figura 22, é possível perceber um comportamento similar entre a política de OR* e a do DeepShutdown. Ambas as políticas apresentam uma amplitude maior nos valores de *slowdown* em comparação com as políticas de *timeout*. Tal comportamento já era esperado na política de OR*, mas também é possível perceber que o DeepShutdown aprendeu uma política mais próxima de uma abordagem

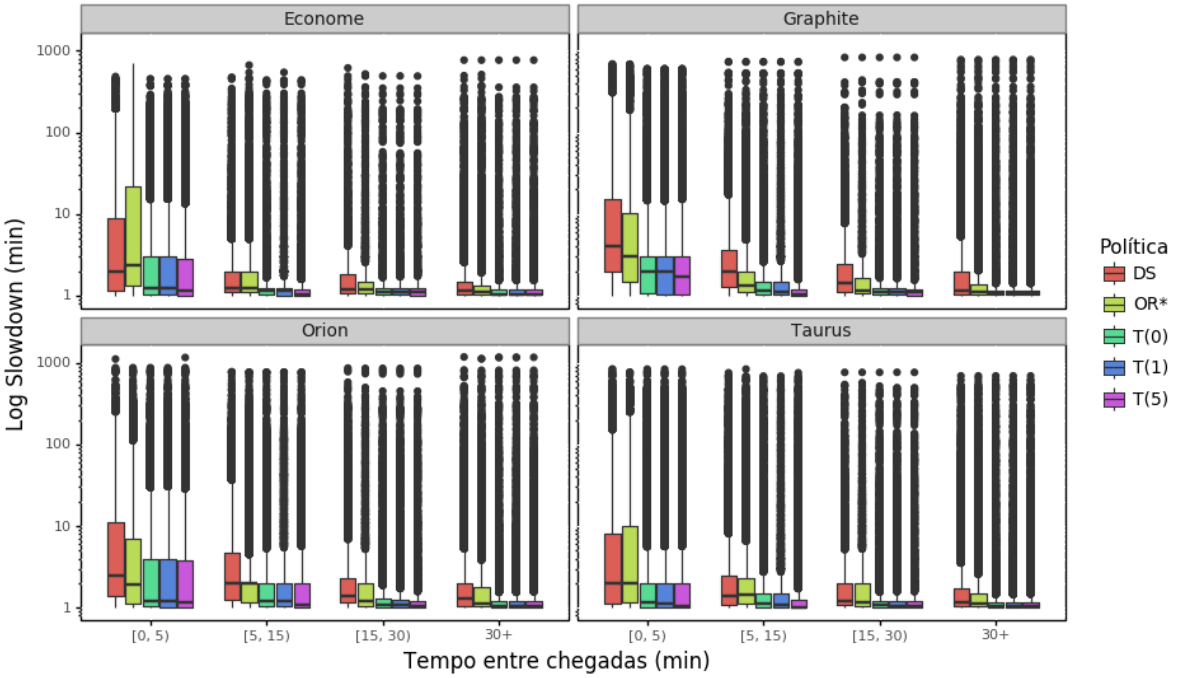


Figura 22 – Comparativo do *slowdown* em função do tempo de chegada.

de OR ao invés de uma política de *timeout*. Comparando o DeepShutdown com a política T(0), este comportamento fica mais evidente. O DeepShutdown exibe a maior amplitude no *slowdown* em quase todos os grupos de cargas de trabalho. O *slowdown* tem uma amplitude consideravelmente maior nos processos com menor tempo de chegada [0,5), mas eventualmente diminui conforme aumenta o tempo de chegada entre os processos. Logo, é possível observar que o DeepShutdown está priorizando o atraso das submissões sequencias de processos, assim como a política OR*.

Somente atrasar a execução de processos advindos de submissões sequencias é ineficiente para garantir uma maior economia de energia. Caso um processo desta categoria tenha um tempo real de execução longo (*e.g.* 1 hora), então não compensa forçar o atraso dos demais processos na fila. Neste caso, o custo em termos de desempenho é consideravelmente maior que o custo em inicializar os recursos para os processos da fila. Contudo, caso o tempo real de execução seja curto (*e.g.* alguns minutos), então o atraso extra pode compensar o custo das inicializações. Para verificar esta situação, é analisado o tempo real de execução dos primeiros 10.000 processos com os maiores valores de *slowdown* em cada carga de trabalho. O resultado desta análise é apresentado na Figura 23. Para diminuir o efeito de valores atípicos, novamente, uma transformação logarítmica é aplicada.

Observando a Figura 23, é possível perceber que o DeepShutdown está atrasando os processos com menor tempo real de execução quando comparado à política de *timeout* T(0). Este comportamento fica evidente nas cargas do agregado Orion, as quais o DeepShutdown também exibe a maior economia de energia. Também é pos-

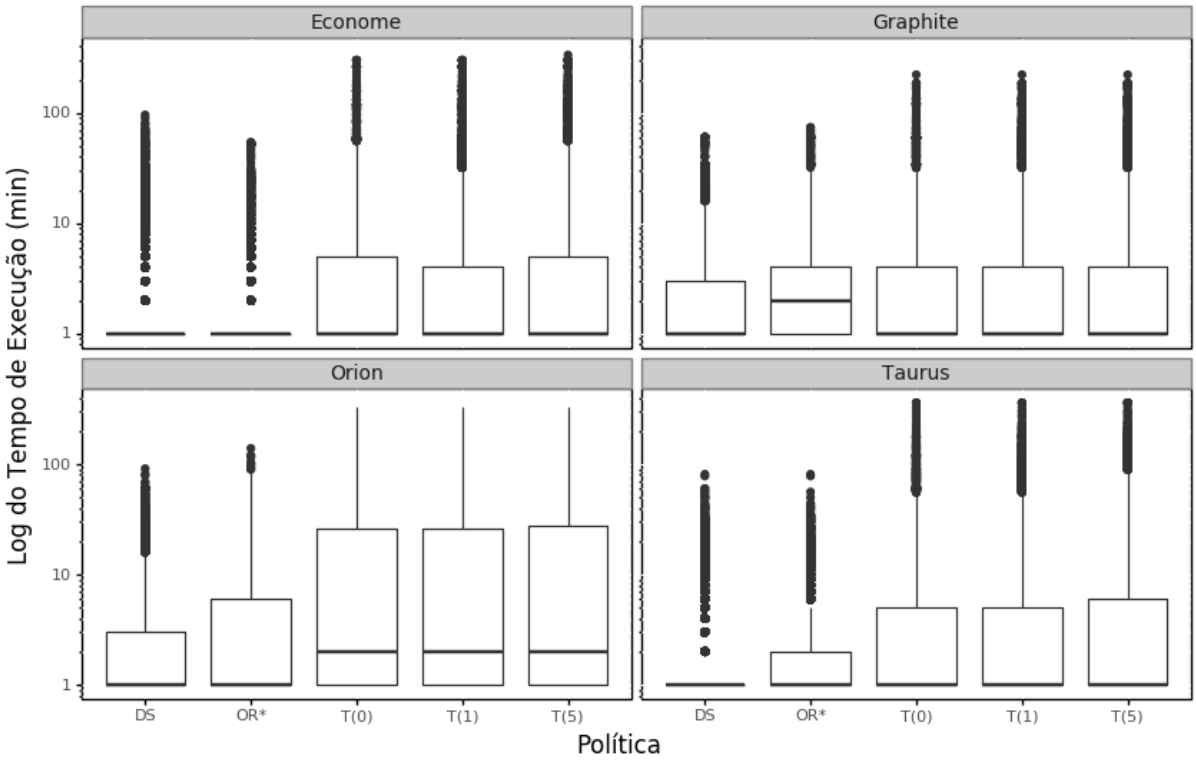


Figura 23 – Tempo real de execução dos 10k processos com os maiores valores de *slowdown*.

sível observar que o DeepShutdown é mais seletivo que a política OR*. Na maioria das cargas, a política OR* atrasou a execução dos processos com menor tempo de execução em relação às políticas de *timeout*. Contudo, o DeepShutdown foi capaz de selecionar os processos com maior precisão, principalmente, devido a sua abordagem mais agressiva. Atrasar os processos com o menor tempo real de execução, tem potencial para minimizar a quantidade de reinicializações dos recursos ao mesmo tempo que reduz o desperdício de energia. Logo, é possível observar que o DeepShutdown utiliza as propriedades das cargas de trabalho para tentar identificar os processos advindos de submissões sequenciais que possuem um tempo real de execução curto. Como consequência, o DeepShutdown atingiu melhores coeficientes em termos de economia de energia.

Para finalizar, é analisado o desempenho do DeepShutdown durante a fase de treinamento. Na Figura 24, é ilustrado a curva de aprendizagem para cada uma das cargas de trabalho. Cada ponto de dados do DeepShutdown é o resultado de uma média de 100 experimentos em cada carga de trabalho do conjunto de dados de treinamento. Para comparar a evolução do desempenho do DeepShutdown no decorrer do treinamento, é apresentado a média das recompensas obtidas com as políticas que apresentam as menores taxas de consumo de energia. Não obstante, como o DeepShutdown inicia o seu treinamento utilizando uma política aleatória, uma política que reserva os recursos aleatoriamente também é incluída.

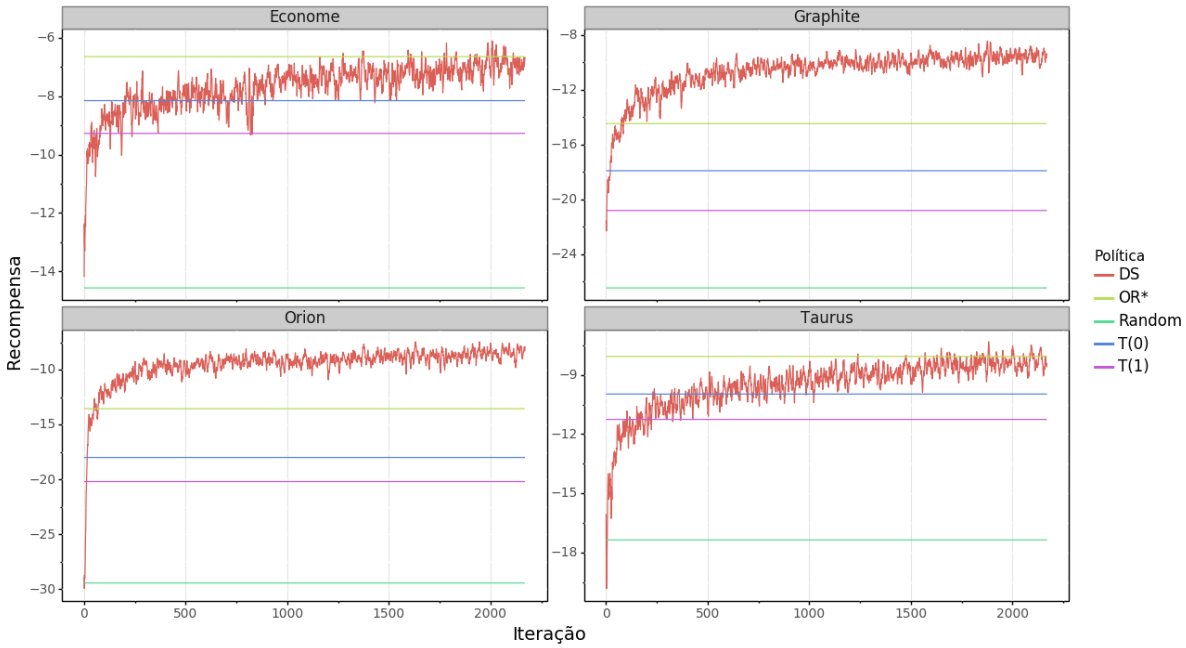


Figura 24 – Curva de aprendizagem do DeepShutdown.

Como é possível observar na Figura 24, o desempenho do DeepShutdown evolui conforme o número de interações aumenta. No começo do treinamento, é observado um desempenho inferior às demais políticas e seu comportamento é similar a uma política aleatória, como esperado. Contudo, na medida que o algoritmo obtém mais experiência, através da interação repetida com o ambiente, o seu comportamento é otimizado. Nos agregados com menor quantidade de recursos (Graphite e Orion), o DeepShutdown atingiu o melhor desempenho após 100 e 250 iterações. Contudo, nos demais agregados (Econome e Taurus), que são plataformas maiores, é possível observar que o DeepShutdown precisou de um maior número de iterações para aprender uma política que exibe resultados similares a política OR*, comparando somente a recompensa. Este comportamento está relacionado ao tamanho do espaço de estados possíveis, que aumenta conforme a quantidade de recursos na plataforma. Como consequência, o tempo de exploração e treinamento tende a ocorrer mais rápido em agregados menores.

A mesma variação observada na Tabela 5 também é observada nas curvas de aprendizagem, analisando a variação nas recompensas. Este comportamento indica que identificar submissões sequenciais de processos com tempo real de execução curto não é uma tarefa trivial. Analisando os agregados, além de exibir uma frequência semelhante de trabalhos sequenciais, cada experimento (chamado de episódio em RL) difere consideravelmente um do outro.

5.4.2 A Política do DeepScheduler

Para entender o comportamento aprendido pelo DeepScheduler, a Figura 25 apresenta a distribuição dos valores *slowdown* de todos os processos para cada uma das políticas de escalonamento. Os processos foram agrupados em 4 grupos, em função dos valores de *slowdown*.

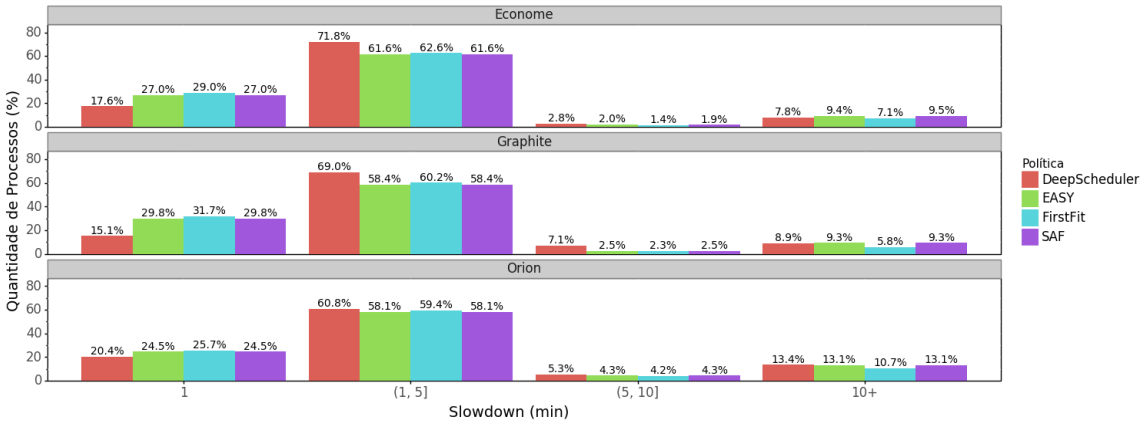


Figura 25 – Distribuição dos valores de *slowdown*.

Observando a Figura 25, é possível perceber que o DeepScheduler apresenta a maior concentração de processos com valores de *slowdown* entre 1–5 enquanto apresenta a menor concentração de processos com *slowdown* de 1. Uma característica das cargas utilizadas é a alta taxa de ocorrência de processos sequenciais que são submetidos dentro de 5 minutos que, em sua maioria, finalizam prematuramente. Logo, faz sentido atrasar alguns processos na esperança que processos executando possam liberar os seus recursos prematuramente de modo a minimizar o número de recursos ativos. Seguindo este princípio, o DeepScheduler atrasa mais processos que as demais políticas para otimizar a eficiência energética.

Atrasar alguns processos é interessante para evitar decisões imediatas e potencialmente ruins. Contudo, não existem garantias que um processo irá terminar prematuramente. Não obstante, esta estratégia deve ser empregada com cautela para não degradar o desempenho do sistema sem apresentar qualquer outra compensação. Para analisar quais processos o DeepScheduler está escolhendo atrasar, na Figura 26 é apresentado os valores de *slowdown* de todos os processos sequenciais. Para minimizar o efeito de valores atípicos só é considerando os valores menores dentro da $1,5\times$ Faixa Interquartil (FIQ)

Observando a Figura 26, é possível perceber que o DeepScheduler exibe os maiores valores de *slowdown* nos processos sequenciais enquanto o First-Fit apresenta os menores valores, em todas as cargas de trabalho. Logo, é possível argumentar que o DeepScheduler está esperando antes de tomar uma decisão de escalonamento e os processos sequenciais são os mais propensos a serem atrasados. Não

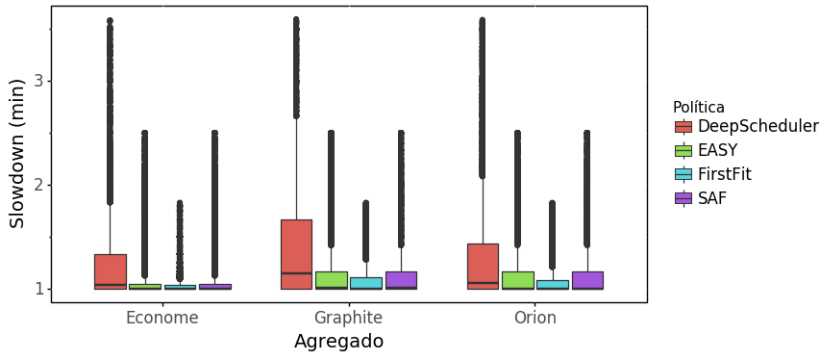


Figura 26 – Valores de *slowdown* de todos os processos sequenciais.

tomando decisões imediatas, é possível que o escalonador explore melhores alternativas de escalonamento que podem ocorrer quando um processo termina prematuramente ou quando um processo com um curto tempo de execução é submetido em seguida. O DeepScheduler foi capaz de aprender este comportamento, seguindo este princípio para economizar energia sem degradar o desempenho.

5.5 CONSIDERAÇÕES PARCIAIS

A dinamicidade observada em sistemas em grades aumenta a complexidade na elaboração de políticas de gerenciamento específicas para os comportamentos de um sistema. Observando este cenário, este capítulo conduziu um conjunto de experimentos utilizando os *traces* da GRID'5000 para avaliar o comportamento aprendido por dois métodos que utilizam DRL para aprender uma política de gerenciamento. O primeiro método, chamado de DeepShutdown, obteve uma economia de energia de até 56,5%, em comparação a uma política suave de *timeout*, através da aplicação de uma estratégia de OR no problema de *shutdown*. Não obstante, foi observada uma redução significativa na quantidade de inicializações e desligamentos de recursos em comparação a uma estratégia que desliga imediatamente os recursos ociosos.

Atuando no escalonamento dos processos, o segundo método chamado de DeepScheduler obteve uma redução no consumo de energia de até 7% quando comparado às políticas tradicionalmente utilizadas em grades computacionais. Ao levar em consideração o comportamento da carga de trabalho, o DeepScheduler também foi capaz de minimizar o tempo médio de espera dos processos na fila em até 26%. Logo, os resultados indicam que ao negligenciar possíveis comportamentos na utilização do sistema é possível que melhores alternativas de gerenciamento sejam ignoradas. Treinar agentes com métodos de DRL para aprender uma política através da interação com o ambiente que reproduz o comportamento do sistema usando seus dados históricos apresenta ser uma opção viável para contornar este cenário.

6 CONSIDERAÇÕES FINAIS

O consumo de energia é uma métrica crucial para a sustentabilidade de uma plataforma computacional de larga escala. O apetite crescente por maior poder computacional é saciado através da composição de plataformas cada vez maiores e complexas. Tais plataformas requerem soluções sofisticadas de gerenciamento para otimizar a eficiência na utilização dos recursos. O paradigma de desempenho a qualquer custo não é mais desejado, administradores estão em constante pesquisa para minimizar os seus custos operacionais. Mecanismos de gerenciamento dinâmico de energia podem tirar vantagem dos períodos em que os recursos são pouco utilizados ou ociosos para economizar energia. Recursos ociosos representam um desperdício de energia pois não estão servindo os usuários ou os provedores. Tal desperdício não pode ser negligenciado e diferentes estratégias são implementadas para minimizar o seu impacto no custo operacional de uma plataforma.

Neste trabalho, políticas de *shutdown* e de escalonamento foram exploradas. No contexto do problema de *shutdown*, foi apresentado a principal desvantagem destes métodos em função da carga de trabalho. Observado esta característica, foi apresentado uma alternativa que utiliza uma estratégia de *Off Reservation* (OR). Ao adotar esta estratégia, a política pode explorar a carga de trabalho para identificar processos que podem ser atrasados de modo a minimizar a quantidade de reinicializações dos recursos, economizando energia. Contudo, definir o comportamento de uma carga de trabalho não é um processo trivial pois sistemas em grades são dinâmicos e suas cargas de trabalho podem mudar de comportamento repentinamente. Com isto em mente, foram utilizados métodos de *Deep Reinforcement Learning* (DRL) para treinar um agente a realizar o desligamento dos recursos via OR. Chamado de DeepShutdown, o método proposto conseguiu aprender como e quando reservar recursos, e desliga-los, de modo a otimizar a eficiência em termos de custo de energia e desempenho. Os resultados obtidos apontaram que o DeepShutdown aprendeu um comportamento similar a uma estratégia de OR que utiliza um oráculo para obter informações reais sobre os processos. Processos com os menores tempos de execução foram os mais propensos a serem atrasados. Deste modo, o DeepShutdown conseguiu superar o desempenho de políticas de *timeout* baseados em regras fixas.

No contexto de escalonamento, o método DeepScheduler foi proposto como uma alternativa para construir políticas de escalonamento adaptativas. Utilizando métodos de DRL, um agente foi treinado para explorar o comportamento das cargas de trabalho de modo a encontrar uma política otimizada para o sistema. Os resultados indicaram que o DeepScheduler conseguiu diminuir o consumo de energia e, ao mesmo

tempo, aumentar o desempenho do sistema em comparação com políticas tradicionalmente utilizadas em produção. Não obstante, ao não executar decisões imediatas, melhores alternativas de escalonamento foram encontradas. Justificando o maior nível de justiça obtido com o DeepScheduler.

Motivado pela falta de ferramentas específicas, também foi desenvolvido um conjunto de ambientes que podem ser utilizados no treinamento de agentes com *Reinforcement Learning* (RL) no contexto de gerenciamento de recursos e processos em grades computacionais. Este ambiente, chamado de GridGym, serve como uma extensão que integra o Batsim ao *framework* do OpenAI Gym para lidar com as peculiaridades do treinamento de tais métodos. Através do GridGym foi possível reproduzir o comportamento de um sistema em produção, através de seus dados históricos, e realizar o treinamento dos métodos propostos. Observando os resultados obtidos, conclui-se que métodos baseados em DRL são uma alternativa viável para a composição de políticas especificas para cada sistema. Não obstante, ficou evidente que a exploração dos comportamentos de uma carga de trabalho não pode ser negligenciada na otimização da eficiência de um sistema em grades.

6.1 TRABALHOS FUTUROS

A aplicação de métodos de DRL no contexto de gerenciamento de recursos e processos em grades computacionais apresentou ser uma opção viável para compor políticas adaptativas. Contudo, um desafio para a aplicação destes métodos está na qualidade das informações utilizadas no treinamento dos agentes e na reprodutibilidade do comportamento real de um sistema nas simulações. Deste modo, como trabalhos futuros, experimentos com cargas de trabalho de outros sistemas devem ser conduzidos. Como consequência, outras configurações de plataformas também devem ser consideradas.

Neste trabalho foi optado por não levar em consideração a topologia de rede de uma plataforma. Esta abstração permitiu que os métodos fossem diretamente avaliados no problema em questão, sem interferências indiretas devido a latência na transferência de dados entre os nós. Contudo, um maior potencial poderia ser explorado caso o algoritmo estivesse ciente do custo advindo desta comunicação. Deste modo, a comunicação entre os nós deve ser considerada na modelagem das plataformas futuramente.

Por fim, a falta de padronização nos experimentos conduzidos para avaliar e comparar o desempenho entre políticas de gerenciamento dificulta a sua reprodutibilidade e compreensão dos resultados. O desenvolvimento do GridGym neste trabalho é fruto deste problema e não deve ser o último a ser desenvolvido. Portanto, é considerá-

vel que o GridGym seja estendido para abranger outros ambientes, como o problema do *Dynamic Voltage and Frequency Scaling* (DVFS), e modelos de sistemas distribuídos, como a computação em nuvem.

6.2 PUBLICAÇÕES

No decorrer do desenvolvimento deste trabalho, quatro artigos científicos foram elaborados e publicados em eventos regionais e internacionais, sendo:

- CASAGRANDE, L. et al. Aprendizado de máquina no gerenciamento de recursos da computação em nuvem: Uma revisão sistemática. In: **X Simpósio de Tecnologia da Informação da Região Norte e Noroeste do Rio Grande do Sul**. [S.l.:s.n.], 2018.
- CASAGRANDE, L.; PILLON, M. A. Deep reinforcement learning no problema de escalonamento de jobs em computação em grids. In: **XIX Escola Regional de Alto Desempenho da Região Sul (ERAD/RS)**, 2018.
- CASAGRANDE, L. et al. Algoritmo de escalonamento baseado em deep reinforcement learning para computação em grids. In: **Computer on the Beach 2019**. [s.n.], 2019.
- CASAGRANDE, L. et al. DeepScheduling: grid computing job scheduler based on deep reinforcement learning. In: **The 34-th International Conference on Advanced Information Networking and Applications (AINA-2020)**. [S.l.:s.n.], 2020.

Um quinto artigo foi submetido e está em espera por aprovação, sendo:

- CASAGRANDE, L. et al. Don't Hurry Be Green: Scheduling Servers Shutdown in Grid Computing with Deep Reinforcement Learning. **Journal of Grid Computing**, 2020. ISSN 1572-9184.

REFERÊNCIAS

ABDALLAH, S.; KAISERS, M. Addressing environment non-stationarity by repeating q-learning updates. **The Journal of Machine Learning Research**, JMLR. org, v. 17, n. 1, p. 1582–1612, 2016.

AHN, D. H. et al. Flux: A next-generation resource management framework for large hpc centers. In: **2014 43rd International Conference on Parallel Processing Workshops**. [S.l.: s.n.], 2014. p. 9–17. ISSN 0190-3918.

ARULKUMARAN, K. et al. A brief survey of deep reinforcement learning. **arXiv preprint arXiv:1708.05866**, 2017.

ASSUNÇÃO, M. D. de et al. The green grid'5000: Instrumenting and using a grid with energy sensors. In: DAVOLI, F. et al. (Ed.). **Remote Instrumentation for eScience and Related Aspects**. New York, NY: Springer New York, 2012. p. 25–42. ISBN 978-1-4614-0508-5.

BAKER, M.; BUYYA, R.; LAFORENZA, D. The grid: International efforts in global computing. 01 2000.

BARSANTI, L.; SODAN, A. C. Adaptive job scheduling via predictive job resource allocation. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. [S.l.], 2006. p. 115–140.

BATES, N. et al. Electrical grid and supercomputing centers: An investigative analysis of emerging opportunities and challenges. **Informatik-Spektrum**, Springer, v. 38, n. 2, p. 111–127, 2015.

BELLEMARE, M. G. et al. The arcade learning environment: An evaluation platform for general agents. **Journal of Artificial Intelligence Research**, v. 47, p. 253–279, jun 2013.

BENOIT, A. et al. Reducing the energy consumption of large-scale computing systems through combined shutdown policies with multiple constraints. **The International Journal of High Performance Computing Applications**, v. 32, n. 1, p. 176–188, 2018. Disponível em: <https://doi.org/10.1177/1094342017714530>.

BOLZE, R. et al. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. **The International Journal of High Performance Computing Applications**, v. 20, n. 4, p. 481–494, 2006.

BORGHESI, A. et al. Power capping in high performance computing systems. In: PESANT, G. (Ed.). **Principles and Practice of Constraint Programming**. Cham: Springer International Publishing, 2015. p. 524–540. ISBN 978-3-319-23219-5.

BROCKMAN, G. et al. **OpenAI Gym**. 2016.

BUŞONIU, L.; BABUŠKA, R.; SCHUTTER, B. D. Multi-agent reinforcement learning: An overview. In: **Innovations in multi-agent systems and applications-1**. [S.l.]: Springer, 2010. p. 183–221.

BUYYA, R. Economic-based distributed resource management and scheduling for grid computing. **arXiv preprint cs/0204048**, 2002.

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.

BUYYA, R.; VENUGOPAL, S. A gentle introduction to grid computing and technologies. **CSI Communications**, v. 29, 11 2004.

CAPIT, N. et al. A batch scheduler with high level components. In: **CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.** [S.l.: s.n.], 2005. v. 2, p. 776–783 Vol. 2.

CARASTAN-SANTOS, D. et al. One can only gain by replacing easy backfilling: A simple scheduling policies case study. In: **Cluster, Cloud and Grid Computing (CCGrid), 2019 19th IEEE/ACM International Symposium on**. [S.l.: s.n.], 2019.

CASANOVA, H. et al. Versatile, scalable, and accurate simulation of distributed applications and platforms. **Journal of Parallel and Distributed Computing**, Elsevier, v. 74, n. 10, p. 2899–2917, jun. 2014. Disponível em: <http://hal.inria.fr/hal-01017319>.

CHASE, J. S. et al. Managing energy and server resources in hosting centers. **ACM SIGOPS operating systems review**, ACM, v. 35, n. 5, p. 103–116, 2001.

CHEN, W.; Deelman, E. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In: **2012 IEEE 8th International Conference on E-Science**. [S.l.: s.n.], 2012. p. 1–8. ISSN null.

CHENG, M.; LI, J.; NAZARIAN, S. Drl-cloud: deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In: IEEE PRESS. **Proceedings of the 23rd Asia and South Pacific Design Automation Conference**. [S.l.], 2018. p. 129–134.

CHUNG, J. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. **arXiv preprint arXiv:1412.3555**, 2014.

COHEN, J. D.; MCCLURE, S. M.; YU, A. J. Should i stay or should i go? how the human brain manages the trade-off between exploitation and exploration. **Philosophical Transactions of the Royal Society B: Biological Sciences**, The Royal Society London, v. 362, n. 1481, p. 933–942, 2007.

DAYARATHNA, M.; WEN, Y.; FAN, R. Data center energy consumption modeling: A survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 1, p. 732–794, 2016.

DUTOT, P. et al. Towards energy budget control in hpc. In: **2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)**. [S.l.: s.n.], 2017. p. 381–390.

DUTOT, P.-F. et al. Batsim: A realistic language-independent resources and jobs management systems simulator. In: DESAI, N.; CIRNE, W. (Ed.). **Job Scheduling Strategies for Parallel Processing**. Cham: Springer International Publishing, 2017. p. 178–197. ISBN 978-3-319-61756-5.

EGI. 2019. Accessed: 2019-06-01. Disponível em: <https://www.egi.eu/>.

EMERAS, J. et al. Evalix: Classification and prediction of job resource consumption on hpc platforms. In: DESAI, N.; CIRNE, W. (Ed.). **Job Scheduling Strategies for Parallel Processing**. Cham: Springer International Publishing, 2017. p. 102–122. ISBN 978-3-319-61756-5.

ETINSKI, M. et al. Parallel job scheduling for power constrained hpc systems. **Parallel Computing**, v. 38, n. 12, p. 615 – 630, 2012. ISSN 0167-8191. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167819112000610>.

ETINSKI, M. et al. Understanding the future of energy-performance trade-off via dvfs in hpc environments. **Journal of Parallel and Distributed Computing**, v. 72, n. 4, p. 579 – 590, 2012. ISSN 0743-7315. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0743731512000172>.

FEITELSON, D. G. Metrics for parallel job scheduling and their convergence. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. [S.l.], 2001. p. 188–205.

FEITELSON, D. G.; RUDOLPH, L. Towards convergence in job schedulers for parallel supercomputers. In: **Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing**. London, UK, UK: Springer-Verlag, 1996. (IPPS '96), p. 1–26. ISBN 3-540-61864-3. Disponível em: <http://dl.acm.org/citation.cfm?id=646377.689507>.

FEITELSON, D. G.; RUDOLPH, L. Metrics and benchmarking for parallel job scheduling. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. [S.l.], 1998. p. 1–24.

FEITELSON, D. G.; Weil, A. M. Utilization and predictability in scheduling the ibm sp2 with backfilling. In: **Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing**. [S.l.: s.n.], 1998. p. 542–546. ISSN 1063-7133.

FELLER, E.; RILLING, L.; MORIN, C. Energy-aware ant colony based workload placement in clouds. In: **Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing**. Washington, DC, USA: IEEE Computer Society, 2011. (GRID '11), p. 26–33. ISBN 978-0-7695-4572-1. Disponível em: <http://dx.doi.org/10.1109/Grid.2011.13>.

FENG, W.; CAMERON, K. The green500 list: Encouraging sustainable supercomputing. **Computer**, v. 40, n. 12, p. 50–55, Dec 2007. ISSN 0018-9162.

FENG, W.-c. The importance of being low power in high performance computing. **Cyberinfrastructure Technology Watch Quarterly (CTWatch Quarterly)**, v. 1, n. 3, p. 11–20, 2005.

FOSTER, I. The physiology of the grid: An open grid services architecture for distributed systems integration. Citeseer, 2002.

FOSTER, I.; KESSELMAN, C. The globus project: A status report. In: IEEE. **Proceedings Seventh Heterogeneous Computing Workshop (HCW'98)**. [S.l.], 1998. p. 4–18.

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.

GALIZIA, A.; QUARATI, A. Job allocation strategies for energy-aware and efficient grid infrastructures. **Journal of Systems and Software**, v. 85, n. 7, p. 1588 – 1606, 2012. ISSN 0164-1212. Software Ecosystems.

GE, R.; FENG, X.; CAMERON, K. W. Improvement of power-performance efficiency for high-end computing. In: **Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11 - Volume 12**. Washington, DC, USA: IEEE Computer Society, 2005. (IPDPS '05), p. 233.2–. ISBN 0-7695-2312-9. Disponível em: <http://dx.doi.org/10.1109/IPDPS.2005.251>.

GEORGIOU, Y.; GLESSER, D.; TRYSTRAM, D. Adaptive resource and job management for limited power consumption. In: **2015 IEEE International Parallel and Distributed Processing Symposium Workshop**. [S.l.: s.n.], 2015. p. 863–870.

GOMEZ-MARTIN, C.; VEGA-RODRIGUEZ, M. A.; GONZALEZ-SANCHEZ, J.-L. Fattened backfilling: An improved strategy for job scheduling in parallel systems. **Journal of Parallel and Distributed Computing**, v. 97, p. 69 – 77, 2016. ISSN 0743-7315. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0743731516300788>.

GOUBERMAN, A.; SIEGLE, M. Markov reward models and markov decision processes in discrete and continuous time: Performance evaluation and optimization. In: ____. **Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems: International Autumn School, ROCKS 2012, Vahrn, Italy, October 22-26, 2012, Advanced Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 156–241. ISBN 978-3-662-45489-3. Disponível em: <https://doi.org/10.1007/978-3-662-45489-3_6>.

GRID'5000. 2019. Accessed: 2019-06-01. Disponível em: <https://www.grid5000.fr/>.

HERLICH, M.; KARL, H. Average and competitive analysis of latency and power consumption of a queuing system with a sleep mode. In: **2012 Third International Conference on Future Systems: Where Energy, Computing and Communication Meet (e-Energy)**. [S.l.: s.n.], 2012. p. 1–10.

HIKITA, J.; HIRANO, A.; NAKASHIMA, H. Saving 200kw and 200 k/year by power-aware job/machine scheduling. In: **2008 IEEE International Symposium on Parallel and Distributed Processing**. [S.l.: s.n.], 2008. p. 1–8.

HINZ, M. et al. A cost model for iaas clouds based on virtual machine energy consumption. **Journal of Grid Computing**, v. 16, n. 3, p. 493–512, Sep 2018. ISSN 1572-9184.

HSU, C.-H.; FENG, W.-C. Effective dynamic voltage scaling through cpu-boundedness detection. In: SPRINGER. **International Workshop on Power-Aware Computer Systems**. [S.l.], 2004. p. 135–149.

HUANG, S.; FENG, W. Energy-efficient cluster computing via accurate workload characterization. In: **2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid**. [S.l.: s.n.], 2009. p. 68–75.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237–285, 1996.

KANDAGATLA, C. Survey and taxonomy of grid resource management systems. **University of Texas, Austin, [Online**, p. 395–2003, 2003.

KIM, G. H.; LEE, C. S. G. Genetic reinforcement learning approach to the machine scheduling problem. In: **Proceedings of 1995 IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 1995. v. 1, p. 196–201 vol.1. ISSN 1050-4729.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KINTSAKIS, A. M.; PSOMOPOULOS, F. E.; MITKAS, P. A. Reinforcement learning based scheduling in a workflow management system. **Engineering Applications of Artificial Intelligence**, v. 81, p. 94 – 106, 2019. ISSN 0952-1976.

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

KONDA, V. R.; TSITSIKLIS, J. N. Actor-critic algorithms. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2000. p. 1008–1014.

KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Software: Practice and Experience**, Wiley Online Library, v. 32, n. 2, p. 135–164, 2002.

KURDI, H.; LI, M.; AL-RAWESHIDY, H. A classification of emerging and traditional grid systems. **IEEE Distributed Systems Online**, v. 9, n. 3, p. 1–1, March 2008. ISSN 1541-4922.

LAPAN, M. **Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more**. [S.l.]: Packt Publishing Ltd, 2018.

LAWSON, B.; SMIRNI, E. Power-aware resource allocation in high-end systems via online simulation. In: ACM. **Proceedings of the 19th annual international conference on Supercomputing**. [S.l.], 2005. p. 229–238.

LEGRAND, A.; TRUSTRAM, D.; ZRIGUI, S. Adapting batch scheduling to workload characteristics: What can we expect from online learning? In: **2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.: s.n.], 2019. p. 686–695.

LELONG, J.; REIS, V.; TRYSTRAM, D. Tuning easy-backfilling queues. In: KLUSÁČEK, D.; CIRNE, W.; DESAI, N. (Ed.). **Job Scheduling Strategies for Parallel Processing**. Cham: Springer International Publishing, 2018. p. 43–61. ISBN 978-3-319-77398-8.

LEUNG, V. J.; SABIN, G.; SADAYAPPAN, P. Parallel job scheduling policies to improve fairness: A case study. In: **2010 39th International Conference on Parallel Processing Workshops**. [S.l.: s.n.], 2010. p. 346–353. ISSN 0190-3918.

LI, Y. Deep reinforcement learning: An overview. **arXiv preprint arXiv:1701.07274**, 2017.

LIU, N. et al. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In: IEEE. **2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)**. [S.l.], 2017. p. 372–382.

MAGOULÈS, F. **Fundamentals of grid computing: theory, algorithms and technologies**. [S.l.]: CRC Press, 2009.

MAHESWARAN, M.; KRAUTER, K. A parameter-based approach to resource discovery in grid computing systems. In: BUYYA, R.; BAKER, M. (Ed.). **Grid Computing — GRID 2000**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 181–190. ISBN 978-3-540-44444-2.

MAO, H. et al. Resource management with deep reinforcement learning. In: **Proceedings of the 15th ACM Workshop on Hot Topics in Networks**. New York, NY, USA: ACM, 2016. (HotNets '16), p. 50–56. ISBN 978-1-4503-4661-0. Disponível em: <http://doi.acm.org/10.1145/3005745.3005750>.

MARZOLLA, M.; MIRANDOLA, R. Dynamic power management for qos-aware applications. **Sustainable Computing: Informatics and Systems**, v. 3, n. 4, p. 231 – 248, 2013. ISSN 2210-5379. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2210537913000206>.

MESHKOVA, E. et al. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. **Computer Networks**, v. 52, n. 11, p. 2097 – 2128, 2008. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S138912860800100X>.

MNIH, V. et al. Asynchronous methods for deep reinforcement learning. In: **International conference on machine learning**. [S.l.: s.n.], 2016. p. 1928–1937.

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

MOGHADAM, M. H.; BABAMIR, S. M. Makespan reduction for dynamic workloads in cluster-based data grids using reinforcement-learning based scheduling. **Journal of computational science**, Elsevier, v. 24, p. 402–412, 2018.

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**, v. 12, n. 6, p. 529–543, June 2001. ISSN 1045-9219.

NESMACHNOW, S. et al. Energy-aware scheduling on multicore heterogeneous grid computing systems. **Journal of Grid Computing**, v. 11, n. 4, p. 653–680, Dec 2013. ISSN 1572-9184. Disponível em: <https://doi.org/10.1007/s10723-013-9258-3>.

NG, A. Y.; RUSSELL, S. J. et al. Algorithms for inverse reinforcement learning. In: **Icml**. [S.l.: s.n.], 2000. v. 1, p. 2.

ORGERIE, A.; LEFÈVRE, L.; GELAS, J. Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In: **2008 14th IEEE International Conference on Parallel and Distributed Systems**. [S.l.: s.n.], 2008. p. 171–178.

ORGERIE, A.-C.; ASSUNCAO, M. D. d.; LEFEVRE, L. A survey on techniques for improving the energy efficiency of large-scale distributed systems. **ACM Computing Surveys (CSUR)**, ACM, v. 46, n. 4, p. 47, 2014.

ORGERIE, A.-C.; LEFÈVRE, L. ERIDIS: Energy-efficient Reservation Infrastructure for large-scale DIstributed Systems. **Parallel Processing Letters**, World Scientific Publishing, v. 21, n. 2, p. 133–154, jun. 2011. Disponível em: <https://hal.inria.fr/ensl-00618594>.

ORHEAN, A. I.; POP, F.; RAICU, I. New scheduling approach using reinforcement learning for heterogeneous distributed systems. **Journal of Parallel and Distributed Computing**, Elsevier, v. 117, p. 292–302, 2018.

OTTERLO, M. van; WIERING, M. Reinforcement learning and markov decision processes. In: **Reinforcement Learning**. [S.l.]: Springer, 2012. p. 3–42.

PINHEIRO, E. et al. Load balancing and unbalancing for power and performance in cluster-based systems. 2001.

POQUET, M. **Simulation approach for resource management**. Tese (Theses) — Université Grenoble Alpes, dez. 2017. Disponível em: <https://tel.archives-ouvertes.fr/tel-01757245>.

PRABHAKARAN, S. **Dynamic resource management and job scheduling for high performance computing**. Tese (Doutorado) — Technische Universität Darmstadt, 2016.

PRABHAKARAN, S. et al. A batch system with efficient adaptive scheduling for malleable and evolving applications. In: **2015 IEEE International Parallel and Distributed Processing Symposium**. [S.l.: s.n.], 2015. p. 429–438. ISSN 1530-2075.

PRIMET, P. V.-B.; ANHALT, F.; KOSLOVSKI, G. Exploring the virtual infrastructure service concept in Grid'5000. In: **20th ITC Specialist Seminar on Network Virtualization**. Hoi An, Vietnam: [s.n.], 2009.

QURESHI, M. B. et al. Survey on grid resource allocation mechanisms. **Journal of Grid Computing**, Springer, v. 12, n. 2, p. 399–441, 2014.

RAÏS, I.; ORGERIE, A.-C.; QUINSON, M. Impact of shutdown techniques for energy-efficient cloud data centers. In: CARRETERO, J. et al. (Ed.). **Algorithms and Architectures for Parallel Processing**. Cham: Springer International Publishing, 2016. p. 203–210. ISBN 978-3-319-49583-5.

Ran, Y. et al. Deepee: Joint optimization of job scheduling and cooling control for data center energy efficiency using deep reinforcement learning. In: **2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)**. [S.l.: s.n.], 2019. p. 645–655.

RAïS, I. et al. Quantifying the impact of shutdown techniques for energy-efficient data centers. **Concurrency and Computation: Practice and Experience**, v. 30, n. 17, p. e4471, 2018. E4471 cpe.4471. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4471>.

RODERO, I.; GUIM, F.; CORBALAN, J. Evaluation of coordinated grid scheduling strategies. In: **2009 11th IEEE International Conference on High Performance Computing and Communications**. [S.l.: s.n.], 2009. p. 1–10.

SABIN, G.; LANG, M.; SADAYAPPAN, P. Moldable parallel job scheduling using job efficiency: An iterative approach. In: FRACHTENBERG, E.; SCHWIEGELSHOHN, U. (Ed.). **Job Scheduling Strategies for Parallel Processing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 94–114. ISBN 978-3-540-71035-6.

SAROOD, O.; KALE, L. V. A 'cool' load balancer for parallel applications. In: **SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2011. p. 1–11.

SCHAERF, A.; SHOHAM, Y.; TENNENHOLTZ, M. Adaptive load balancing: A study in multi-agent learning. **Journal of artificial intelligence research**, v. 2, p. 475–500, 1994.

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

SCHULMAN, J. et al. High-dimensional continuous control using generalized advantage estimation. **arXiv preprint arXiv:1506.02438**, 2015.

SCHULMAN, J. et al. Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, 2017.

SEWAK, M. Policy-based reinforcement learning approaches. In: ____. **Deep Reinforcement Learning: Frontiers of Artificial Intelligence**. Singapore: Springer Singapore, 2019. p. 127–140. ISBN 978-981-13-8285-7. Disponível em: <https://doi.org/10.1007/978-981-13-8285-7_10>.

SHI, L.; ZHANG, Z.; ROBERTAZZI, T. Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud. **IEEE Transactions on Parallel and Distributed Systems**, v. 28, n. 6, p. 1607–1620, June 2017.

SMITH, W. Improving resource selection and scheduling using predictions. In: ____. **Grid Resource Management: State of the Art and Future Trends**. Boston, MA: Springer US, 2004. p. 237–253. ISBN 978-1-4615-0509-9. Disponível em: <https://doi.org/10.1007/978-1-4615-0509-9_16>.

SNOWDON, D. C.; RUOCCO, S.; HEISER, G. Power management and dynamic voltage scaling: Myths and facts. In: **Proceedings of the 2005 workshop on power aware real-time computing**. [S.l.: s.n.], 2005. v. 12, p. 1–7.

SRINIVASAN, S. et al. Characterization of backfilling strategies for parallel job scheduling. In: **Proceedings. International Conference on Parallel Processing Workshop**. [S.l.: s.n.], 2002. p. 514–519. ISSN 1530-2016.

SUN, D. et al. Re-stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. **Information Sciences**, v. 319, p. 92 – 112, 2015. ISSN 0020-0255. Energy Efficient Data, Services and Memory Management in Big Data Information Systems.

SUNDERMEYER, M.; SCHLÜTER, R.; NEY, H. Lstm neural networks for language modeling. In: **Thirteenth annual conference of the international speech communication association**. [S.l.: s.n.], 2012.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2014. p. 3104–3112.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

SUTTON, R. S.; BARTO, A. G.; WILLIAMS, R. J. Reinforcement learning is direct adaptive optimal control. **IEEE Control Systems**, IEEE, v. 12, n. 2, p. 19–22, 1992.

SUTTON, R. S. et al. Policy gradient methods for reinforcement learning with function approximation. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2000. p. 1057–1063.

TANGMUNARUNKIT, H.; DECKER, S.; KESSELMAN, C. Ontology-based resource matching in the grid - the grid meets the semantic web. In: **International Semantic Web Conference**. [S.l.: s.n.], 2003.

TARPLEE, K. M. et al. Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques. **IEEE Transactions on Parallel and Distributed Systems**, v. 27, n. 6, p. 1633–1646, June 2016.

TERZOPOULOS, G.; KARATZA, H. Performance evaluation and energy consumption of a real-time heterogeneous grid system using dvs and dpm. **Simulation Modelling Practice and Theory**, v. 36, p. 33 – 43, 2013. ISSN 1569-190X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1569190X13000695>.

TODOROV, E.; EREZ, T.; TASSA, Y. Mujoco: A physics engine for model-based control. In: IEEE. **2012 IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.], 2012. p. 5026–5033.

TOKIC, M. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In: SPRINGER. **Annual Conference on Artificial Intelligence**. [S.l.], 2010. p. 203–210.

TONG, Z. et al. Proactive scheduling in distributed computing - a reinforcement learning approach. **Journal of Parallel and Distributed Computing**, Elsevier, v. 74, n. 7, p. 2662–2672, 2014.

VASILE, M.-A. et al. Mlbox: Machine learning box for asymptotic scheduling. **Information Sciences**, Elsevier, v. 433, p. 401–416, 2018.

VINYALS, O.; FORTUNATO, M.; JAITLY, N. Pointer networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2015. p. 2692–2700.

WANG, Z. et al. Sample efficient actor-critic with experience replay. **arXiv preprint arXiv:1611.01224**, 2016.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992.

WCG. 2019. Accessed: 2019-06-01. Disponível em: <https://www.worldcommunitygrid.org/>.

WILKINSON, B. **Grid computing: techniques and applications**. [S.l.]: CRC Press, 2009.

WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. **Machine Learning**, v. 8, n. 3, p. 229–256, May 1992. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00992696>.

WU, Q. et al. Adaptive dag tasks scheduling with deep reinforcement learning. In: VAIDYA, J.; LI, J. (Ed.). **Algorithms and Architectures for Parallel Processing**. Cham: Springer International Publishing, 2018. p. 477–490. ISBN 978-3-030-05054-2.

XHAFA, F.; ABRAHAM, A. Computational models and heuristic methods for grid scheduling problems. **Future Generation Computer Systems**, v. 26, n. 4, p. 608 – 621, 2010. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X09001782>.

XSEDE. 2019. Accessed: 2019-06-01. Disponível em: <https://www.xsede.org/>.

Yi, D. et al. Efficient compute-intensive job allocation in data centers via deep reinforcement learning. **IEEE Transactions on Parallel and Distributed Systems**, v. 31, n. 6, p. 1474–1485, 2020.

YOO, A. B.; JETTE, M. A.; GRONDONA, M. Slurm: Simple linux utility for resource management. In: FEITELSON, D.; RUDOLPH, L.; SCHWIEGELSHOHN, U. (Ed.). **Job Scheduling Strategies for Parallel Processing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 44–60. ISBN 978-3-540-39727-4.

YOUNG, B. D. et al. Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment. **The Journal of Supercomputing**, v. 63, n. 2, p. 326–347, Feb 2013. ISSN 1573-0484. Disponível em: <https://doi.org/10.1007/s11227-012-0740-7>.

ZANIKOLAS, S.; SAKELLARIOU, R. A taxonomy of grid monitoring systems. **Future Generation Computer Systems**, v. 21, n. 1, p. 163 – 188, 2005. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X04001190>.

ZHANG, W.; DIETTERICH, T. G. A reinforcement learning approach to job-shop scheduling. In: CITESEER. **IJCAI**. [S.l.], 1995. v. 95, p. 1114–1120.

ZHOU, S. et al. **Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems**. [S.l.], 1993.

ZOMAYA, A. Y.; CLEMENTS, M.; OLARIU, S. A framework for reinforcement-based scheduling in parallel processor systems. **IEEE transactions on parallel and distributed systems**, IEEE, v. 9, n. 3, p. 249–260, 1998.

ZOTKIN, D.; KELEHER, P. J. Job-length estimation and performance in backfilling schedulers. In: **Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No.99TH8469)**. [S.l.: s.n.], 1999. p. 236–243. ISSN 1082-8907.