UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Fernando Emilio Puntel

ANÁLISE DE DESEMPENHO DE ALGORITMOS DE ESCALONAMENTO APLICADOS A UM SISTEMA GERENCIADOR DE RECURSOS PARA EXECUÇÃO DE APLICAÇÃO OPERACIONAL DE COMPUTAÇÃO CIENTÍFICA

Santa Maria, RS
2019

Fernando Emilio Puntel

ANÁLISE DE DESEMPENHO DE ALGORITMOS DE ESCALONAMENTO APLICADOS A UM SISTEMA GERENCIADOR DE RECURSOS PARA EXECUÇÃO DE APLICAÇÃO OPERACIONAL DE COMPUTAÇÃO CIENTÍFICA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação.

ORIENTADORA: Prof.ª Andrea Schwertner Charão

Santa Maria, RS
2019

Puntel, Fernando Emilio
Análise de Desempenho de Algoritmos de Escalonamento aplicados a um Sistema Gerenciador de Recursos para Execução de Aplicação Operacional de Computação Científica / Fernando Emilio Puntel.- 2019.
73 p.; 30 cm

Orientadora: Andrea Schwertner Charão
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação , RS, 2019

1. Algoritmos de Escalonamento 2. SLURM 3. Computação Científica 4. Análise de desempenho 5. Cluster I. Schwertner Charão, Andrea II. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

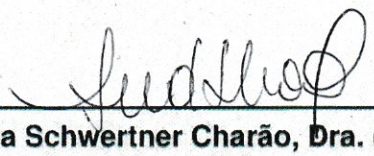
End. Eletr.: fepuntel@inf.ufsm.br

Fernando Emilio Puntel

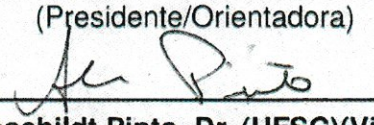
ANÁLISE DE DESEMPENHO DE ALGORITMOS DE ESCALONAMENTO APLICADOS A UM SISTEMA GERENCIADOR DE RECURSOS PARA EXECUÇÃO DE APLICAÇÃO OPERACIONAL DE COMPUTAÇÃO CIENTÍFICA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação.

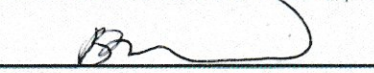
Aprovado em 15 de fevereiro de 2019:



Andrea Schwertner Charão, Dra. (UFSM)
(Presidente/Orientadora)



Alex Sandro Roschildt Pinto, Dr. (UFSC)(Videoconferência)



Benhur de Oliveira Stein, Dr. (UFSM)

Santa Maria, RS
2019

DEDICATÓRIA

Este trabalho é dedicado aos meus pais Loidir e Juceli, as minhas irmãs Bruna e Luiza e à minha namorada Gabriela.

AGRADECIMENTOS

Aos meus pais Loidir José Puntel e Juceli Gasparin Puntel pelo apoio, suporte e paciência neste período. As minhas irmãs Bruna Elisa Punte, Luiza Fernanda Puntel e ao meu cunhado Rouglas Felipe Strapazzon. À minha namorada Gabriela dos Santos de Souza pelo carinho, companheirismo e confiança. A minha orientadora, professora Dra. Andrea Schwertner Charão pelos ensinamentos nestes dois anos de mestrado, sempre me estimulando na busca de novos conhecimentos. Ao Dr. Adriano Petry do INPE, pela paciência e disponibilidade. À todos os professores que de alguma maneira contribuíram durante esta caminhada transferindo conhecimento para a minha formação. A UFSM, CAPES e ao PPGCC pela oportunidade de realizar o mestrado, contribuindo para meu crescimento acadêmico e pessoal. Sem vocês, nada disso seria possível.

"É muito melhor lançar-se em busca de conquistas grandiosas, mesmo expondo-se ao fracasso, do que alinhar-se com os pobres de espírito, que nem gozam muito nem sofrem muito, porque vivem numa penumbra cinzenta, onde não conhecem nem vitória, nem derrota."

(Theodore Roosevelt)

RESUMO

ANÁLISE DE DESEMPENHO DE ALGORITMOS DE ESCALONAMENTO APLICADOS A UM SISTEMA GERENCIADOR DE RECURSOS PARA EXECUÇÃO DE APLICAÇÃO OPERACIONAL DE COMPUTAÇÃO CIENTÍFICA

AUTOR: Fernando Emilio Puntel
ORIENTADORA: Andrea Schwertner Charão

No decorrer dos últimos anos a computação de alta performance é uma das principais áreas de pesquisa na computação, que tem com o propósito aumentar o desempenho dos ambientes computacionais e diminuir o tempo de execução das aplicações. Com uma gama de recursos computacionais distribuídos, o Sistema Gerenciador de Recursos (SGR) é o responsável por realizar o gerenciamento dos recursos computacionais, gerenciamento dos jobs para execução e dos usuários nestes ambientes. Para realizar o gerenciamento do ambiente de alto desempenho, o SGR possui alguns componentes essenciais para seu funcionamento, entre eles, um dos mais importantes é algoritmo de escalonamento, que define quando e onde cada job será executado. Contudo, a escolha do algoritmo de escalonamento para ambientes científicos que exigem resultados em tempo hábil e uma utilização otimizada dos recursos computacionais ainda necessita de mais pesquisas e experimentos. Este estudo analisa os aspectos do ambiente de computação distribuído quando utilizado quatro diferentes algoritmos de escalonamento aplicados ao sistema gerenciador de recursos SLURM, em um ambiente que executa uma aplicação científica de previsão ionosférica. São apresentados os pontos fortes e fracos de cada um dos algoritmos desenvolvidos e aplicados a um sistema gerenciador de recursos quando expostos a situações adversas em um ambiente que executa uma aplicação científica.

Palavras-chave: Algoritmo de escalonamento de jobs, SLURM, Sistema Gerenciador de Recursos, previsão ionosférica, computação científica

ABSTRACT

JOB SCHEDULING ALGORITHMS PERFORMANCE ANALYSIS APPLIED TO A RESOURCE MANAGEMENT SYSTEM OF OPERATIONAL SCIENTIFIC COMPUTING APPLICATION

AUTHOR: Fernando Emilio Puntel
ADVISOR: Andrea Schwertner Charão

During the last years, the High Performance Computing (HPC) is one of the main research areas in computing, with the proposal to increase the performance of high performance environments and decrease the application time execution. With lot distributed computational resources, the Resource Management System (RMS) is the response to the manager of computational resources, job manager and user manager in HPC. For perform the management in HPC, the RMS have some essential components for its operation, between them, one of the most important is a job scheduling, that defines when and where each job will execute. Yet, the job scheduling choice for scientific environments that require results in a timely manner and high utilization of the computational resources still need more researches and experiments. In this study, I analyzed the aspects of the distributed environment computational when we used four job scheduling different applied in resource management system SLURM, in an environment that running a scientific application of ionosphere forecasting. We present the strong points and weak points of each one job scheduling algorithm developed applied in resource management system when exposed in adverse situation in an environment that performs a scientific application.

Keywords: Job scheduling algorithm, SLUM, Resource Management System, ionosphere forecast, scientific computing

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWT	Average Waiting Time
CRS	Centro Regional Sul
CPU	Central Process Unit
$DAVS$	Data Assimilation and Visualization System
$EASY$	Extensible Argonne Scheduling system
$FCFS$	First Come First Served
$FIFO$	First in first out
GNU	General Public License
$GrADS$	Grid Analysis and Display System
GPS	Global Positioning System
HPC	High Performance Computing
$INPE$	Instituto Nacional de Pesquisas Espaciais
IRI	International Reference Ionosphere
kJ	kilojoule
$UFSM$	Universidade Federal de Santa Maria
$LLNL$	Lawrence Livermore National Laboratory
LSF	Loading Sharing Facility
MPI	Message Passing Interface
$NOAA$	National Oceanic and Atmospheric Administration
NPB	NAS Parallel Benchmarks
NAS	Numerical Aerodynamic Simulation
PBS	Portable Batch System
$RAPL$	Running Average Power Limit
RMS	Resource Management System
SFU	Solar Flux Units
SGR	Sistema Gerenciador de Recuros
SIP	Solar Irradiance Plataform

SJF	Shortest Job First
$SLURM$	Simple Linux Utility for Resource Management
$SUPIM$	Sheffield University Plasmasphere-Ionosphere Model
TEC	Total Eletronic Content

SUMÁRIO

1 INTRODUÇÃO

Ambientes de alto desempenho são comumente utilizados pelas mais diversas áreas de pesquisa. Áreas como previsões de clima espacial e simulação de fluídos físicos são exemplos de áreas que se beneficiam deste tipo de ambiente. Esses ambientes se tornaram muito atrativos por oferecerem processamento de alto desempenho e possibilitarem a obtenção de resultados mais rápidos (LIEU; FARHAT; LESOINNE, 2006). Desde os primeiros ambientes de alto desempenho, a performance das aplicações e uma alta taxa de utilização dos recursos computacionais estavam relacionados com a eficiência para obtenção de resultados. Com o grande avanço do poder de processamento dos computadores, a busca por atingir um melhor desempenho e obter resultados rápidos se tornaram fundamentais nestes ambientes, onde cada vez mais os resultados devem estar disponíveis em tempo hábil.

Dentre os ambientes de alto desempenho, o cluster é um dos ambientes que fornece às aplicações alto poder de processamento e respostas, por possuir computadores (normalmente chamados de nós) de alto desempenho interligados por uma rede de alta velocidade, a fim de trabalharem em conjuntos para resolver um problema, na maioria das vezes inviável de serem executados em um computador pessoal (STERLING; GROPP; LUSK, 2002).

Com uma grande quantidade de recursos computacionais distribuídos e trabalhando em conjunto, é necessária a utilização de um Sistema Gerenciador de Recursos (SGR), responsável por realizar diversas tarefas no ambiente de alto desempenho a fim de gerenciar os jobs da aplicação e obter um melhor rendimento e utilização dos recursos computacionais. Um SGR é formado por alguns componentes fundamentais para o seu funcionamento e gerenciamento do cluster e aplicação do usuário. O algoritmo de escalonamento é um dos principais componentes de um SGR, responsável por atender as solicitações de execução dos jobs dos usuários e utilizar os recursos computacionais distribuídos de forma eficiente (TALBY; FEITELSON, 1999) (MORENO; ALONSO-CONDE, 2004).

A busca por aumento na taxa de utilização dos recursos computacionais e a velocidade para obtenção dos resultados depende de diversos fatores, entre eles, um dos mais importantes é a escolha de um algoritmo de escalonamento aplicado a um SGR eficiente. A escolha de um algoritmo de escalonamento é uma das tarefas mais importantes para ambientes científicos e está relacionado com as necessidades de cada ambiente e aplicação. Ambientes de previsões de clima espacial são ambientes onde o algoritmo de escalonamento deve otimizar a ocupação dos recursos computacionais e otimizar a execução dos jobs a fim de obter dados em tempo hábil para utilização, caso contrário dados serão impróprios para uso comercial e/ou acadêmico (DONG; AKL, 2006).

Com diversas opções de algoritmos de escalonamento, é necessário que se escolha um que atenda as necessidades da aplicação e da arquitetura do ambiente de alto

desempenho onde o algoritmo será utilizado. Por conta da diversidade para realização de escalonamento, busca-se encontrar o algoritmo adequado para cada contexto. A escolha inapropriada de um algoritmo de escalonamento poderá acarretar em atraso para obtenção de resultados e uma baixa taxa de utilização de recursos computacionais (ABAWAJY, 2004) (ABAWAJY; DANDAMUDI, 2003).

Para realizar o escalonamento, existem algoritmos com técnicas de escalonamento mais simples como os algoritmos FIFO, SJF e *Roubin Round*, até algoritmos de escalonamento com técnicas mais sofisticadas, como por exemplo, algoritmos de escalonamento especializados em preencher lacunas de recursos computacionais que antes ficavam ociosas por conta de algumas políticas de escalonamento. Dentre os algoritmos, destacam-se o *Backfilling* (MU'ALEM; FEITELSON, 2001) que permite adiantar jobs da fila e algoritmos que utilizam técnicas de inteligência artificial para melhorar o rendimento dos recursos (YU et al., 2007).

Mesmo que exista uma diversidade de algoritmos de escalonamento aplicados a SGR, muitos são apresentados somente de forma teórica, avaliados quando expostos a ambientes simulados ou com jobs que utilizam cargas sintéticas. Observando a ausência de estudos que avaliam algoritmos de escalonamento em ambientes científicos, avaliar o desempenho dos algoritmos de escalonamento aplicados a um SGR em um cluster com uma aplicação científica é de suma importância, analisando aspectos primordiais para suprir as necessidades da aplicação.

Por conta disso, este estudo busca implementar e avaliar o desempenho de algoritmos de escalonamento já conhecidos aplicados ao Sistema Gerenciador de Recursos SLURM em um cluster. Para os experimentos, os algoritmos de escalonamento foram expostos a um cenário que utiliza jobs de uma aplicação científica de previsão ionosférica que é executada diariamente no Centro Regional Sul do Instituto Nacional de Pesquisas Espaciais e outro cenário com jobs de um grupo de *benchmarks*, com a intenção de avaliar a eficiência dos algoritmos em cenários distintos.

1.1 Objetivo Geral

Implementar e avaliar o desempenho de algoritmos de escalonamento aplicados ao Sistema Gerenciador de Recursos SLURM utilizando uma aplicação operacional de computação científica.

1.2 Estrutura do Texto

Além da introdução, este trabalho está organizado da seguinte forma. No capítulo 2 descreve-se ambientes de alto desempenho e sistemas gerenciadores recursos, com ênfase no SGR SLURM. Em seguida, no Capítulo 3 são descritos algoritmos de escalo-

namento e trabalhos relacionados. No Capítulo 4 são apresentados conceitos básicos de sistema de previsão ionosférica e apresentado o modelo de previsão ionosférica SUPIM utilizado no Instituto Nacional de Pesquisas Espaciais. No Capítulo 5 são apresentados os algoritmos de escalonamento utilizados neste trabalho, bem como a metodologia de implementação. Por fim, são apresentados os resultados obtidos por meio de experimentos realizados em um ambiente de alto desempenho utilizando um sistema de previsão ionosférica e um grupo de *benchmarks* no Capítulo 6, bem como as considerações que foram possíveis chegar com a análise dos experimentos e sugestões para trabalhos futuros no Capítulo 7.

2 AMBIENTES DE ALTO DESEMPENHO

Neste capítulo é apresentada a definição de ambientes de alto desempenho e seus componentes. Também é descrito Sistema Gerenciador de Recursos, um componente fundamental para o funcionamento destes ambientes, com ênfase no SGR SLURM.

2.1 Ambientes de Alto Desempenho

Com uma gama cada vez maior de dados para processamento, os ambientes de computação alto desempenho evoluíram muito no decorrer dos anos, possibilitando que as mais diversas áreas se beneficiem destes ambientes.

Ambientes de alto desempenho são comumente utilizados pelas mais diversas áreas de pesquisa, de desenvolvimento e áreas industriais. Desde aplicações que realizam previsões de camadas atmosféricas da terra (XIE et al., 2010), aplicações na área de geociência (LINDTJORN et al., 2011) até aplicações do mercado financeiro (ZHANG; SHI; KHAN, 2018). Estas e as mais diversas áreas são beneficiadas por ambientes de alto desempenho computacional, onde computadores com alto poder de processamento e memória trabalham de forma colaborativa, possibilitado o processamento de uma gama de informações e fornecendo resultados em tempo hábil.

Dentre as opções de computação de alto desempenho (do inglês *High Performace Computing* - HPC), destacam-se os seguintes ambientes: *cloud*, *grid* e cluster. Onde cada um delas possui suas características e a escolha pelo tipo de ambiente para processamento irá depender da aplicação e disponibilidade. *Cloud computing* é a tecnologia mais recente de HPC quando comparada aos demais ambientes de alto desempenho. A *cloud* permite que usuários que não tenham acesso a um cluster e/ou uma *grid* de forma física consiga executar sua aplicação em ambientes de alto desempenho alugados somente pelo período de tempo necessário para a execução. A principal característica é a escalabilidade, onde o usuário consegue obter mais ou menos poder de processamento apenas com apenas alguns cliques (BENTO, 2012).

Grid, ou Grade, é formado por computadores de alto desempenho que podem estar em locais geograficamente diferentes, onde os usuários podem executar a aplicação através de uma rede cooperativa. Para execução, o usuário acessa um computador gerenciador, responsável pelo gerenciamento dos nós de processamento e dos jobs das aplicações (GANDOTRA et al., 2011).

Um cluster computacional é um conjunto de computadores, comumente chamados de nó, que estão fisicamente interligados por uma rede de alta velocidade. O aspecto mais importante em um cluster é que os nós devem trabalhar em conjunto capazes de executarem coletivamente como um recurso único. Além dos nós de processamento, o cluster possui um nó administrador responsável por delegar funções e tarefas para cada

um dos nós de processamento.

Sterling (2011) define cluster como uma coleção de nós de processamento fracamente acoplados, isto deve-se ao fato de que a memória está distribuída entre os nós, diferente de uma configuração fortemente acoplado, onde todos os processadores possuem acesso e compartilham uma memória principal. Para (SLOAN, 2004) cluster é um conjunto de computadores que trabalham em conjunto para resolução de um problema, onde o cluster possui três componentes essenciais, uma coleção de computadores, uma rede de alta velocidade que interconecta os nós e um software que possibilita que os computadores compartilhem trabalhos com os demais nós do cluster.

Devido um cluster possuir recursos computacionais distribuídos, normalmente é necessário que um Sistema Gerenciador de Recursos (SGR) seja instalado e configurado no cluster, a fim de conseguir obter um melhor gerenciamento dos recursos, usuários e jobs. Atualmente existe um diversidade de SGRs disponíveis para instalação e configuração, cada um com características próprias e adequadas para os mais diferentes ambientes.

2.2 Sistemas Gerenciadores de Recursos

Um Sistema Gerenciador de Recursos (SGR) é um componente essencial para o funcionamento de um cluster ou *grid*, onde seu principal objetivo é realizar o gerenciamento dos recursos computacionais e tarefas dos usuários. Um SGR executa funções cruciais para o funcionamento do cluster, como escalonamento de jobs dos usuários, monitoramento de status dos nós de processamento e gerenciamento dos jobs. Para isso, um SGR ideal deve ser simples, eficiente e facilmente escalável (STERLING; GROPP; LUSK, 2002).

Um SGR em geral é utilizado para gerenciar a distribuição e execução de jobs em cluster. Tipicamente um SGR possuem três componentes: nó master: normalmente, executa a aplicação e possui o escalonador do SGR; nós computacionais: os nós computacionais executam os jobs e retornam os resultados; Recursos: recursos podem ser recursos de armazenamento, processamento, memória ou até rede de alta velocidade (KIERTSCHER; ZINKE; SCHNOR, 2013).

Além disso, é importante que o SGR seja altamente escalável, tolerante a falhas e portátil, permitindo assim que o sistema gerenciador consiga atender clusters com milhares de processadores de forma confiável e eficaz (JETTE et al., 2002). Atualmente existem diversos SGR com características distintas, atendendo as mais variadas arquiteturas e aplicações.

Dentre os mais conhecidos estão: o *Portable Batch System (PBS)* que é um sistema flexível que opera em sistemas UNIX multi-plataforma, que abrange cluster heterogêneos, supercomputadores e sistemas que necessitam computação paralela massivamente (HENDERSON, 1995); o *Loading Sharing Facility (LSF)* é um sistema proprietário para com-

putação de alto desempenho que pode ser utilizada tanto em ambientes UNIX quanto Windows e em redes com diferentes arquiteturas (ZHOU et al., 1993); o SGR Condor foi projetado especialmente para ambientes heterogêneos (LITZKOW; LIVNY; MUTKA, 1988). Além do SLURM, utilizado neste trabalho que é explicado de forma detalhada na subseção 2.2.1.

2.2.1 SLURM

O SGR SLURM (*Simple Linux Utility for Resource Management*) inicialmente foi desenvolvido pelo *Lawrence Livermore National Laboratory* (LLNL) [1] para gerenciamento de milhares de processadores. O SLURM foi projetado para que a configuração e instalação deste SGR fosse realizada de forma rápida e fácil, indiferente da arquitetura e tamanho do ambiente computacional (YOO; JETTE; GRONDONA, 2003).

O SLURM é um sistema gerenciador de recursos *open source*, tolerante a falhas e altamente escalável. Como gerenciador de cargas de trabalho, basicamente o SLURM possui três funções principais: primeiramente, o SLURM fornece acesso exclusivo e/ou compartilhado de recursos para que o usuário execute sua aplicação por um período de tempo. A segunda tarefa, é oferecer uma estrutura para submeter, executar e gerenciar o job. E finalmente, realizar o gerenciamento dos recursos e uma fila de jobs pendentes para execução (ZHOU et al., 2013), (YOO; JETTE; GRONDONA, 2003). O SLURM foi totalmente projetado seguindo algumas métricas (YOO; JETTE; GRONDONA, 2003), (GEORGIOU, 2010):

- Simplicidade: o SLURM é simples o suficiente para que entusiastas entendam seu código fonte adicionem funcionalidades personalizadas para cada arquitetura e aplicação;
- *Open Source*: o SLURM é um SGR de código aberto sob a licença GNU (*General Public License* (GNU, 2015));
- Portabilidade: o SLURM é escrito na linguagem de programação C, com um mecanismo de configuração GNU *autoconf*;
- Independência da interconexão: O SLURM suporta comunicações sobre os protocolos UDP/IP e comunicações *Quadrics Elan3 e Myrinet*;
- Escalabilidade: O SLURM foi projetado para executar em ambientes computacionais com milhares de nós e processadores;
- Robustez: O SLURM consegue lidar com vários modos de falhas sem encerrar o job que está executando. Os jobs dos usuários podem ser configurados para que continuem a execução, mesmo que um dos nós apresente problema;

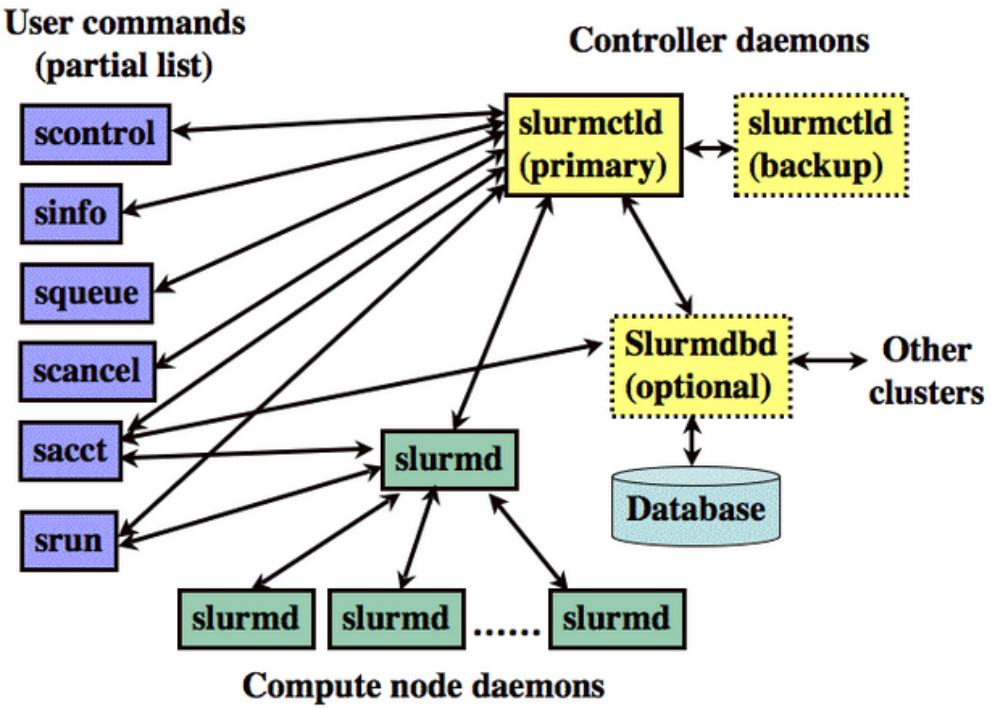
[1] https://www.llnl.gov/

- Segurança: O SLURM oferece *plug-in* de segurança que oferece tecnologia de criptografia para autenticação dos usuários a serviços;
- Sistema de administração amigável: o SLURM é configurado usando um arquivo de configuração simples e de fácil entendimento.

Atualmente o SLURM é uma das ferramentas mais populares para gerenciamento de cluster e jobs, estando presente em 6 dos 10 supercomputadores (SCHEDMD, 2017) na lista do TOP500 (TOP500, 2017), que lista os 500 supercomputadores com maior poder computacional do mundo. A ferramenta também é utilizada para gerenciamento no supercomputador Santos Dumount (LNCC, 2016), o supercomputador brasileiro melhor colocado no TOP500 [2].

A Figura 2.1 ilustra a arquitetura do SGR SLURM. O SLURM consiste em *SLURM controller* (ilustrado na Figura 2.1 como slurmctld) instalado e configurado no nó master, responsável pelo monitoramento e gerenciamento dos jobs e alocação de recursos, e um *Slurm Daemon* (ilustrado na Figura 2.1 como slurmd) instalado e configurado em cada um dos nós de processamento (SCHEDMD, 2017). Além do *Slurmdbd*, opcional para funcionando do SLURM, para gerenciamento de informações do ambiente computacional através de um banco de dados.

Figura 2.1 – Arquitetura do SLURM



Extraído de: (SCHEDMD, 2017)

[2] https://www.top500.org/

O *slurmd daemon* que executa em cada nó de processamento. Após iniciado no nó de processamento o *daemon slurmd* lê o arquivo de configuração, recupera informações salvas anteriores, informa o nó gerenciador que está ativo (pronto para receber novos jobs) e partir disso aguarda até que algum job seja delegado para execução; executa o job à ele submetido; retorna o status e aguarda novamente novos jobs (YOO; JETTE; GRONDONA, 2003) (BULL, 2010).

O *slurmctld* é responsável pelo gerenciamento de todos os demais jobs e é no nó onde é armazenado as informações de funcionamento do SGR. Quando inicializado, primeiramente o *slurmctld*, lê as informações do arquivo de configuração do SLURM. As informações de estado do nó de controle são gravadas no disco periodicamente para o sistema tolerante a falhas. Basicamente, o *slurmctld* possui três componentes principais: gerenciamento dos nós: monitora o estado de cada nó do cluster; gerenciamento das partições: agrupa os nós em partições e realiza o gerenciamento; e gerenciamento dos nós, onde aceita solicitações de jobs dos usuários e coloca em uma fila de prioridades para execução (YOO; JETTE; GRONDONA, 2003) (BULL, 2010).

Além de gerenciar os nós de processamento de forma individual, o SLURM permite que os nós sejam agrupados em partições, onde um conjunto lógico de nós são agrupados para o gerenciamento. Cada partição pode ser gerenciada de maneira diferente, onde cada uma pode possuir uma lista de espera, tamanho máximo do tamanho dos jobs, limite de tempo para execução, usuários autorizados para utilizar a partição, entre outros (YOO; JETTE; GRONDONA, 2003).

A configuração do SLURM no ambiente de alto desempenho é feita através de um arquivo de configuração, que contém informações necessárias para o seu funcionamento. Neste arquivo de configuração é possível realizar alterações conforme a necessidade do ambiente, como maneira de solicitação de recursos, configuração dos nós de processamento entre outros (PUNTEL; PETRY; CHARÃO, 2018). Através do arquivo de configuração do SLURM é possível selecionar a opção de escalonamento utilizando o algoritmo de escalonamento *Backfilling* conservador.

Além da configuração padrão, o SLURM também oferece mais de 100 plugins de 26 variedades diferentes disponíveis para instalação e configuração do cluster. Dentre as mais conhecidas estão plugins para topologias de rede, MPI (*Message Passing Interface*) e para sensores externos de temperatura, consumo de energia, entre outros (YOO; JETTE; GRONDONA, 2003).

3 ALGORITMOS DE ESCALONAMENTO

Neste capítulo será abordados conceitos inerentes a este trabalho no que diz respeito a algoritmos de escalonamento. Inicialmente são descritos conceitos básicos de algoritmos de escalonamento e em seguida os algoritmos mais comuns e os estudos relacionados ao trabalho proposto.

3.1 Algoritmos de escalonamento

No decorrer dos anos, algoritmos de escalonamento vem sendo muito investigados. A escolha de um algoritmo de escalonamento eficiente pode resultar em uma diminuição do tempo de resposta e um aumento de performance da aplicação e consequentemente rapidez para obtenção do resultado. Por conta disso, a escolha de um algoritmo de escalonamento em aplicações de alto desempenho é de grande importância para o desempenho tanto do software quanto do hardware (FEITELSON et al., 1997).

O escalonamento de jobs ou processos, é a atividade principal para atribuir à recursos computacionais programas para execução. Normalmente o objetivo do algoritmo de escalonamento consiste em que esta atribuição seja feita de maneira rápida e eficiente, minimizando assim gastos computacionais. Os algoritmos de escalonamento devem sempre ser projetados para atender as mais diferentes solicitações de processos ou jobs e para executarem em diferentes ambientes de alta performance.

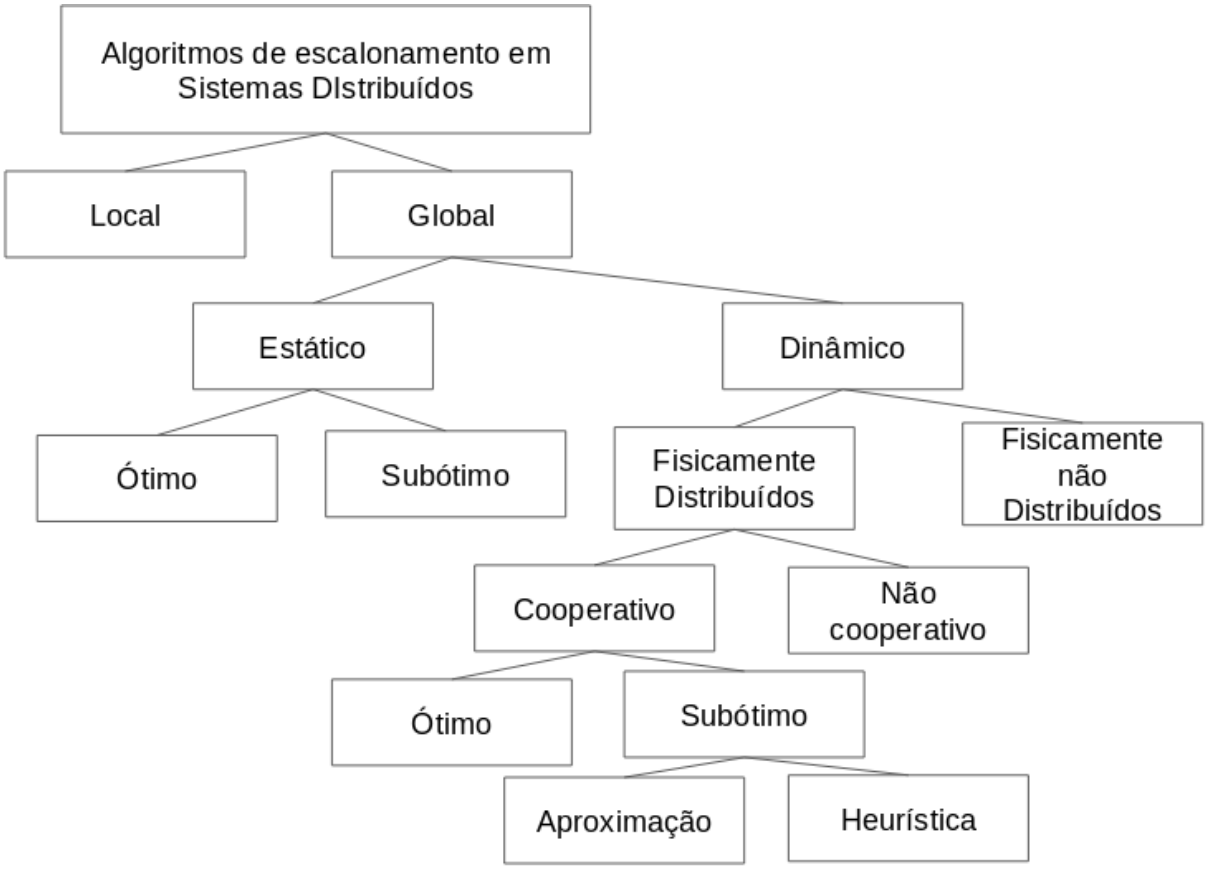
É sempre muito importante diferenciar em qual ambiente o algoritmo de escalonamento será utilizado. Este ambiente pode conter desde somente uma estação de processamento até ambientes com diversas estações de processamento distintas. Por conta disso, os níveis do escalonamento devem ser definidos de acordo com a aplicação e o ambiente de processamento (CARVALHO, 2013). A Figura 3.1 ilustra os níveis e subníveis de escalonamentos.

O escalonamento local é definido como um nível onde o escalonador de processos é responsável pelo gerenciamento da execução de tarefas no sistema operacional, monitorando questões como a execução dos processos, em qual dos processadores do computador a tarefa será submetida e executada, além da definição do tempo em que a aplicação irá utilizar o processador (TANENBAUM; FILHO, 1995).

O outro nível de escalonamento, definido como escalonamento global, consiste em definir e monitorar qual estação de trabalho (também conhecido como nó) é a mais adequada para execução do processo em questão. Neste nível, em sua maioria o objetivo principal é que o algoritmo de escalonamento consiga uma melhor ocupação dos recursos computacionais distribuídos (TANENBAUM; STEEN, 2007).

A nível de escalonamento global, destacam-se o escalonamento estático e dinâmico. No escalonamento estático, as atribuições para os processadores é definida no início

Figura 3.1 – Classificação dos níveis de escalonamento



Extraído e traduzido de: (CASAVANT; KUHL, 1988)

da execução, quando a estimativa do custo da aplicação já é conhecida. Já o escalonamento dinâmico é aplicado justamente quando a estimativa do total de processos e tempo de execução é desconhecida, ou em casos onde as tarefas são submetidas durante a execução, neste caso poucas informações são fornecidas para o escalonador, como requisições dos jobs e o ambiente em que irá realizar o escalonamento (ARABNEJAD; BARBOSA, 2014).

Em ambientes distribuídos, onde um algoritmo de escalonamento é aplicado a um sistema gerenciador de recursos, na sua maioria o algoritmo é exposto a situações dinâmicas, onde diversos usuários podem submetem suas solicitações de recursos e o ambiente de processamento pode ou não ser conhecido inicialmente, além dos recursos computacionais serem dinâmico, como no caso das Grids.

Normalmente o algoritmo de escalonamento é projetado para diminuição do tempo de execução e aumento da taxa de utilização dos recursos computacionais, fazendo com que os recursos fiquem ociosos o menor tempo possível, além de permitir que vários usuários utilizem os recursos do sistema de forma eficaz e os jobs sejam atendidos em tempo hábil para as diferentes aplicações (TALBY; FEITELSON, 1999).

Além do aumento da taxa de utilização de recursos computacionais, os algoritmos

podem visar outros objetivos, como: equidade entre o tempo de espera dos jobs (VASUPONGAYYA; CHIANG, 2005); minimização da latência ou o tempo resposta, minimização da utilização de recursos energéticos (GEORGIOU et al., 2015) ou limitação da potência dos processadores (GEORGIOU; GLESSER; TRYSTRAM, 2015). Em alguns casos, esses objetivos entram em conflito, por conta disso, os recursos computacionais utilizados como coeficiente para o escalonamento irá depender das necessidades de cada ambiente e aplicação.

Existem diversas técnicas para realizar o escalonamento de jobs e tarefas em HPC, o algoritmo de escalonamento FIFO (*First In First Out*), também conhecido como FCFS (*First Come First Served*) é um dos algoritmos mais comuns e a base para vários outros algoritmos de escalonamento considerados justos. No algoritmo clássico FIFO, os jobs são atendidos conforme a ordem de chegada na fila de execução, e caso não tenha recursos suficientes para a execução, o job deve aguardar até a sua requisição de recursos seja atendida, o que poderá acarretar em aumento de tempo de conclusão da aplicação e recursos computacionais ociosos, já que a técnica utilizada pelo FIFO não permite que jobs menores ou com menor prioridade sejam adiantados para execução. Na maioria das vezes o FIFO é utilizado somente em ambientes de experimentação, por ser considerado inadequado para a grande maioria das aplicações (MACHADO; MAIA, 2004).

Além do FIFO, o algoritmo de escalonamento SJF (*Shortest Job First*) também é comumente utilizados em aplicações e ambientes de testes, onde o algoritmo ordena os jobs de maneira crescente de maneira que os jobs com menores requisições ou prioridades menores executem primeiro (MAO; KINCAID, 1994). Outro algoritmo de escalonamento muitas vezes utilizado é o *Round Robin*, neste caso, todos os jobs sendo com maiores ou menores requisições ou prioridade possuem o mesmo tempo para processamento, buscando uma equidade entre os tempos de processamento (MITCHELL, 1998).

Contudo, em ambientes que demandam desempenho computacional e um tempo de execução eficiente, é preciso que os algoritmos de escalonamento mais sofisticados sejam utilizados e avaliados. Algoritmos de escalonamento mais sofisticados incluem técnicas aprimoradas para melhorar o desempenho em ambientes HPC, além de utilizarem informações contundentes em consideração para realizarem o escalonamento de jobs, como estado atual do ambiente de alta performance, técnicas de preenchimento, inteligência artificial entre outras. Também devem ser levados em conta diferentes informações, como a configuração do ambiente, seja o ambiente heterogêneo ou homogêneo, tipo de jobs que irá ser executado, bem como possíveis requisições dos jobs e requisitos fundamentais para a aplicação (OMARA; ARAFA, 2009).

3.2 Trabalhos relacionados

Diversos estudos apresentam diferentes estratégias de escalonamento em uma variedade de ambientes computacionais. A busca por diminuir o tempo de execução, tempo

de espera e consumo energético são alguns dos aspectos levados em consideração na escolha do melhor algoritmo de escalonamento para ambientes de alto desempenho com aplicações científicas . Selecionar uma estratégia de escalonamento mais adequada ainda é um grande desafio, dependendo de diversos fatores, como arquitetura da HPC, a aplicação em questão, entre outros aspectos, e com isso surgem as mais diversas estratégias para satisfazer cada aplicação.

Dentre as técnicas de escalonamento sofisticadas, estão algoritmos baseado no *backfilling*, que surgiu originalmente para estender a abordagem do algoritmo FIFO, buscando aumentar a eficiência dos recursos computacionais. Este tipo de algoritmo também é reconhecido por aumentar a satisfação dos usuários, pois permite que jobs menores ou com prioridades menores sejam adiantados da fila de execução, preenchendo assim, lacunas deixadas pelos jobs maiores (ARNDT et al., 2000) (JONES; NITZBERG, 1999).

No algoritmo *backfilling* clássico, também chamado de *backfilling* conservador, o algoritmo realiza adiantamentos de jobs com requisições menores ou com prioridade menores da fila de execução, desde que não atrase nenhum dos jobs que seja antecessor ao job a ser adiantado na fila para execução. A grande vantagem desde algoritmo é que permite que as decisões de programação sejam feitas na submissão do job, e assim, prever quando cada job será executado, fornecendo garantias aos usuários (HAMSCHER et al., 2000). Porém, mesmo com uma metodologia que permite o adiantamento de jobs menores, em alguns casos não é possível realizar o adiantamento por conta da metodologia de não permitir atraso dos jobs anteriores. Por conta disso, surgiram diversas vertentes baseadas no algoritmo de escalonamento *backfilling*, a fim de melhorar o desempenho de diferentes métricas.

Entre os algoritmos de escalonamento vertentes do *backfilling*, o algoritmo de escalonamento *EASY-backfilling* (*Extensible Argonne Scheduling System*) é o mais comum, onde também considera o adiantamento de jobs menores ou com prioridades menores, porém, apenas primeiro job da fila tem reservado o seu tempo para início. A proposta do *EASY-backfilling* é mais "agressiva", pois aumenta a taxa de utilização de recursos no momento, porém, o adiantamento de jobs pode vir a causar atrasos posteriormente (MU'ALEM; FEITELSON, 2001).

Assim como houve melhoras para o algoritmo *backfilling*, surgiram algumas adaptações para o *EASY-backfilling*, o algoritmo de escalonamento proposto por (TALBY; FEITELSON, 1999) oferece melhoras em relação ao *EASY-backfilling*, principalmente limitando um tempo de espera para todos os jobs, o que determina quanto tempo o job pode esperar até que sua requisição seja atendida pela escalonador, em experimentos realizados em simulador, o algoritmo proposto, apresenta melhoras significativas em relação ao *EASY-backfilling*. No estudo (SHAH; QURESHI; RASHEED, 2010), os autores propõem dois algoritmos baseados no *EASY-backfilling* para preenchimentos de lacunas de recursos computacionais com jobs menores, utilizando cargas de trabalhos sintéticas, os algoritmos

apresentaram um melhorar significativa em relação ao *EASY-backfilling*.

Dutot (2017) propôs um algoritmo de escalonamento baseado no algoritmo *backfilling* que leva em consideração o orçamento do consumo de energia durante um período de tempo. Mesmo com o objetivo de reduzir o consumo de energia, em muitas aplicações reais fica inviável utilizar um sistema que depende de orçamento energético, pelo fato, da aplicação necessitar do resultado em um tempo hábil e ficar aguardando para poder executar um job, pode representar um gasto maior ou dados resultantes já invalido.

Levando em considerações informações atuais do cluster, Kumar; Agarwal; Krishnan (2004) propõem um algoritmo de escalonamento baseado em lógica *Fuzzy*, que toma a decisão de escalonamento, baseado no estado dinâmico do cluster, como por exemplo carga de trabalho. O algoritmo tem como objetivo obter melhor taxa de utilização dos recursos computacionais e melhorar o balanceamento de carga utilizando inteligência artificial.

Também levando em consideração informações dinâmicas, o algoritmo proposto por Shmueli; Feitelson (2005) analisa os jobs faltantes para execução, onde os jobs não ficam em uma fila para execução, o algoritmo busca encontrar uma combinação de tarefas que, juntas, maximizem a utilização dos recursos computacionais. Os experimentos são realizados em um simulador para algoritmos de escalonamento com jobs com carga sintética e a comparação é feita em relação ao *EASY-backfilling*, onde o algoritmo proposto apresenta resultados superiores. Mesmo com resultados superiores em relação a outros algoritmos, ainda resta a análise do desempenho em ambientes reais.

Com uma gama de algoritmos, é preciso analisar quais algoritmos de escalonamento devem ser utilizados em diferentes ambientes HPC, conforme a necessidade. Em Qureshi; Shah; Manuel (2011) apresentam uma análise de algoritmos de escalonamento com características distintas utilizando diferentes Sistemas Gerenciadores de Recursos utilizando *benchmarks* com diferentes cargas de trabalho de trabalho. Neste estudo, foram analisados *throughout*, taxa de utilização das CPUs e memória e utilização da rede.

A nível de comparação para validação de algoritmos de escalonamento *backfilling* em ambientes reais, o trabalho de (WONG; GOSCINSKI, 2007) busca avaliar resultados obtidos por outras pesquisas que utilizaram simuladores para avaliar o desempenho dos algoritmos, além de apontar problemas existentes em simulações na literatura atual. O estudo avaliou algoritmos como *ARCA* e *EASY-backfilling* utilizando aplicações *MPI* conhecidas em um ambiente real. Nos experimentos realizados, o *EASY-backfilling* obteve um desempenho favorável quando comparado com os demais algoritmos analisados, e os autores recomendam mais experimentações de algoritmos de escalonamento em ambientes reais com a utilização de SGR.

Mesmo com uma gama de algoritmos de escalonamento propostos e analisados, dificilmente encontra-se comparações e avaliações de desempenho em ambientes reais, como cluster, utilizando aplicações científicas. A maioria dos algoritmos são analisados em simuladores e/ou ambientes que utilizem cargas sintéticas, o que pode passar des-

percebido obstáculos que podem ser encontrados em ambientes físicos com aplicações científicas.

Seguindo a linha de avaliar os algoritmos de escalonamento, a comparação de algoritmos de escalonamento aplicados a um Sistema Gerenciador de Recursos em ambientes reais com aplicações científicas é promissora, expondo os algoritmos em cenários distintos, buscando encontrar dificuldades e facilidades de cada um dos algoritmos quando submetidos a ambientes adversos, além disso, também busca-se analisar os algoritmos de escalonamento selecionados em um cenário controlado com jobs de benchmark.

4 SISTEMA DE PREVISÃO IONOSFÉRICA

Este capítulo apresenta conceitos básicos de previsão ionosférica, o modelo de primeiros princípios das camadas da Ionosfera e Plasmosfera e da Terra utilizado neste trabalho, bem o detalhamento da aplicação operacional da previsão ionosférica do Instituto Nacional de Pesquisas Espaciais (INPE).

4.1 Previsão ionosférica

A ionosfera é uma das áreas mais importantes e pesquisadas no que diz respeito à clima espacial. A camada da Ionosfera é de suma importância para nosso dia a dia, podendo interferir diretamente em sistemas tecnológicos utilizados, como o Sistema Global de Posicionamento (GPS) (GREWAL; WEILL; ANDREWS, 2007).

Por conta da sua importância, existem algumas maneiras de realizar a previsão e análise da ionosfera. Diversas pesquisas utilizam modelos empíricos baseados em física e dados observacionais para estimar o conteúdo eletrônico da ionosfera. Neste contexto, o modelo *Internacional Reference Ionosphere* (IRI) (MCNAMARA, 1984), (EZQUER; JAKOWSKI; JADUR, 1997) tem sido utilizado em diversos trabalhos, onde também é comparado o *Total Eletronic Content* (TEC) com outros modelos e observações.

Com isso, diversas comunidades científicas iniciaram as previsões ionosféricas utilizando técnicas de assimilação de dados combinadas com modelos físicos, resultando em maneiras confiáveis de estimar e prever sistemas dinâmicos da ionosfera. Com o avanço, diferentes técnicas e modelos de assimilação foram sendo desenvolvidos ao longo dos anos (ANGLING; CANNON, 2004), (BUST; GARNER; GAUSSIRAN, 2004) e (ANGLING; KHATTATOV, 2006).

O modelo *Sheffield University Plasmasphere-Ionosphere Model* (SUPIM) foi usado como ponto de partida para criação do programa Clima Espacial Brasileiro do Instituto Nacional de Pesquisas Espaciais. Diversas melhores foram sendo implementadas ao longo do tempo, também foi implementada uma técnica de assimilação de dados e testada a mesma utilizando dados de ionossonda. Os jobs do modelo SUPIM foram utilizados no experimentos deste trabalho, bem como jobs de assimilações de dados da previsão ionosférica.

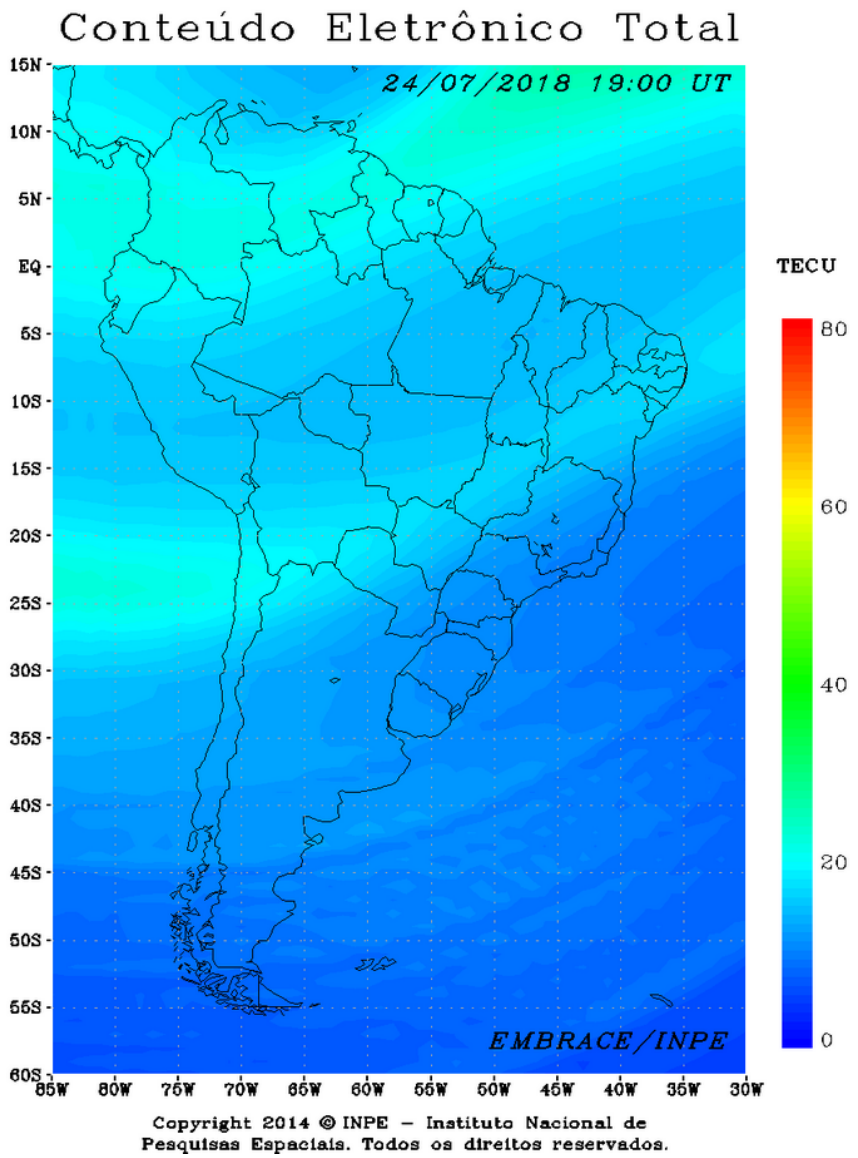
4.2 SUPIM

O SUPIM (*Sheffield University Plasmasphere-Ionosphere Model*) é um modelo utilizado para realização de previsões ionosféricas que é mantido pela INPE em parceria com Departamento de Matemática e Computação Aplicada, University of Sheffield - UK (PETRY, 2010). O SUPIM é um modelo de primeiros princípios das camadas da Ionosfera e

Plasmosfera da Terra. Este modelo fornece informações como densidade, velocidades alinhadas ao campo magnético e temperatura para os elétrons e íons. O SUPIM possui uma série de parâmetros de entrada para execução, que são basicamente: data/hora desejada para simulação, campo magnético, atmosfera neutra, vento, desvio vertical e informações sobre fluxos solares em diferentes frequências (EUV, F10.7 e F10.7A) (PETRY et al., 2014). O modelo foi desenvolvido na linguagem Fortran e todas as informações de entradas são obtidas através de previsões de irradiância solar *Solar2000* (TOBISKA et al., 2000) e por observações passadas ao modelo através de um código na linguagem *Shell Script*.

Além da execução do modelo SUPIM, a aplicação operacional possui um pré e pós processamento. E após a execução do modelo operacional de previsão ionosférica é realizado a geração de imagens com informações do conteúdo eletrônico total da ionosfera, como ilustrado na Figura 4.1.

Figura 4.1 – Mapa final da previsão ionosférica do INPE

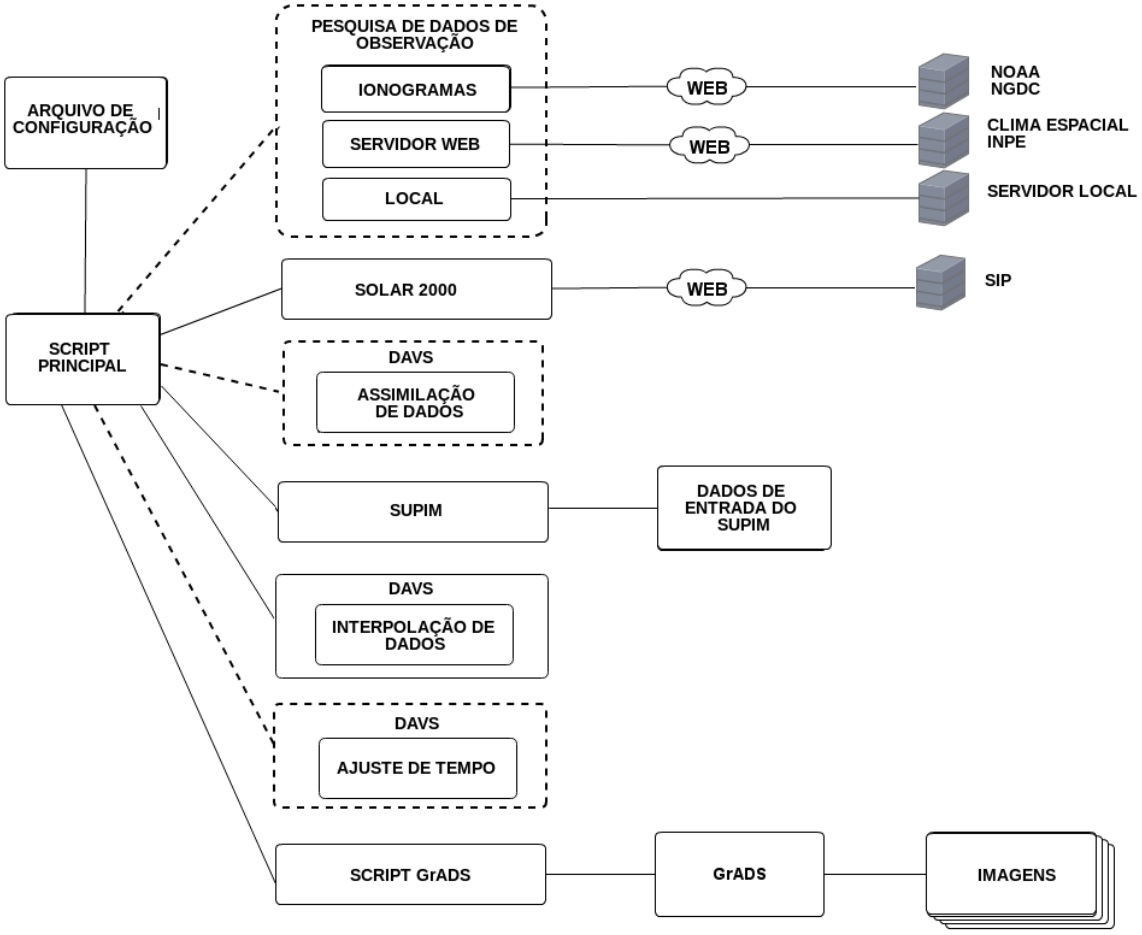


Extraído de: (INPE, 2018)

4.3 Execução operacional do sistema de previsão ionoférica

Além da execução do modelo SUPIM, é preciso realizar um pré e pós processamento para a previsão ionosférica. A Figura 4.2 ilustra o fluxograma da aplicação operacional do sistema de previsão ionosférica como um todo, onde através de um arquivo de configuração, contendo informações e variáveis de entrada para realizar a previsão é iniciada a execução através de um *script* principal, desenvolvido na linguagem *shell script*. É importante ressaltar que os itens com linhas pontilhadas são opcionais para a previsão ionosférica, já as demais etapas são primordiais para realização da previsão ionosférica.

Figura 4.2 – Fluxograma de execução do sistema de previsão ionosférica.



Extraído e traduzido de: (PUNTEL et al., 2015)

Como ilustrado na Figura 4.2 o pré-processamento consiste em busca de informações importantes para realização da previsão ionosférica. A etapa Pesquisa de Dados de Observação, opcional para previsão, consiste em realizar buscas e download de dados observacionais, estes dados podem ser obtidos através de três bases de dados: NOAA (*National Oceanic and Atmospheric Administration*)[1], Programa de clima espacial do INPE

[1] https://ngdc.noaa.gov/stp/iono/ionohome

[2] ou através de um servidor local.

Uma etapa essencial para continuidade da previsão ionosférica é o download dos dados de fluxo solar (na Figura 4.2 nomeado como *SOLAR 2000*). As informações de fluxo solar são obtidas através do software SIP (*Solar Irradiance Plataform*) da empresa *Space Environment Technologies* (EXELIS, 2018).

A próxima etapa é a primeira etapa do conjunto de execuções DAVS. A assimilação de dados é opcional, isso porque necessita de previsões ionosféricas anteriores do mesmo dia, além de dados observacionais reais para realizar a assimilação de dados.

Após as etapas de pré-processamento do sistema de previsão ionosférica, a próxima etapa é a execução do modelo SUPIM, que tem como dependência dados obtidos através do SOLAR2000, da assimilação de dados e download de dados observacionais, caso estas etapas tenham sido executadas. Para a execução do SUPIM são criados arquivos de entrada com variáveis contendo informações relevantes para a sua execução. O total de jobs do SUPIM irá depender da área global que deseja-se realizar a previsão ionosférica, pré-definida inicialmente no arquivo de configuração inicial. Por exemplo, a previsão ionosférica disponibilizada diariamente do site do clima espacial do INPE (INPE, 2018) é realizada somente para a área da América do Sul e a configuração é que cada job SUPIM execute referente a cada um dos 56 graus de longitude, neste caso, totalizando 56 jobs SUPIM. O exemplo de uma previsão ionosférica para a América do Sul também é ilustrado na Figura 4.1. Contudo, é possível realizar uma previsão ionosférica para todo o globo terrestre e com isso pré-definir que a granularidade seja maior, onde cada job SUPIM execute de 5 em 5 graus de longitude, por exemplo.

Com os dados resultantes do modelo SUPIM, a próxima etapa, essencial para a previsão ionosférica, é a interpolação dos dados. Neste caso o DAVS, Interpolação de Dados depende de dados oriundos das execuções do SUPIM para conseguir executar. Normalmente o total de jobs DAVS é de 24, onde cada job representa uma hora das próximas 24 horas, porém, em casos onde a assimilação de dados é realizada, é possível que este número de jobs seja menor.

A próxima etapa, também do grupo do DAVS, é o Ajuste de Tempo que verifica possíveis atrasos ou adiantamentos das previsões ionosféricas, tentando chegar ao mais próximo da realidade.

E a última etapa é a geração dos mapas 2D da previsão ionosférica, para isso é utilizado o sistema GrADS (*Grid Analysis and Display System*) (GRADS, 2018), para que as imagens sejam disponibilizadas no site do INPE (INPE, 2018).

Devido a aplicação operacional possuir diversas etapas para execução com jobs com diferentes cargas de trabalho é primordial realizar a escolha de um algoritmo de escalonamento de jobs eficiente para esta aplicação. Neste caso, a escolha do algoritmo de escalonamento de forma incorreta poderá acarretar em um atraso para obtenção da previ-

[2] http://www2.inpe.br/climaespacial/portal/pt/

são ionosférica, onde em alguns casos será preciso descartar os dados processados por já se estarem inválidos para utilização.

4.4 Detalhamento dos jobs da aplicação operacional

Os jobs que são utilizados nesta aplicação possuem detalhes que devem ser levados em consideração no escalonamento e tomada de decisões dos algoritmos. A aplicação é composta por dois grupos de jobs, nomeados SUPIM e DAVS.

O grupo de jobs nomeados SUPIM (*Sheffield University Plasmasphere-Ionosphere Model*) são jobs homogêneos que executam um modelo de primeiros princípios das camadas de Ionosfera e Plasmosfera da Terra. Este modelo fornece informações como densidade, velocidades alinhadas ao campo magnético e temperatura para os elétrons e íons. O SUPIM possui uma série de parâmetros de entrada para execução, que são basicamente: data/hora desejada para simulação, campo magnético, atmosfera neutra, vento, desvio vertical e informações sobre fluxos solares em diferentes frequências (EUV, F10.7 e F10.7A) (PETRY et al., 2014).

O modelo SUPIM foi desenvolvido na linguagem de programação Fortran e todas as informações de entradas são obtidas através de previsões solar de irradiância *SOLAR2000* (TOBISKA et al., 2000) e por observações passadas ao modelo através de um código na linguagem *Shell Script*. O modelo SUPIM apresenta uma instabilidade para convergência de dados, principalmente em períodos de baixa atividade solar e íons com temperaturas negativas, o que pode acarretar em problemas de convergência de dados do SUPIM, transformando-os em jobs heterogêneos. Como nos experimentos serão realizado a simulação ionosférica somente para a área da América do Sul, serão submetidos 56 jobs SUPIM, onde cada um representa um grau de longitude do território escolhido. Os jobs SUPIM necessitam de iteração físicas e químicas para buscarem condições de equilíbrio. Devido cada job representar um grau de longitude na área pré-selecionada, poderá ocorrer de alguns jobs terem um tempo de execução diferente dos demais, por este motivo foi escolhido um dia com fluxo solar estável, onde essa diferença de tempo dos jobs não influenciará na diferença de tempo entre os jobs (PETRY et al., 2014).

Os jobs nomeados como DAVS - Interpolação de Dados, são jobs homogêneos, que utilizam dados oriundos da previsão dos jobs SUPIM para realizar a interpolação dos dados. A interpolação de dados utiliza um técnica chamada “explosão de perfil", onde os pontos simulados são associados a pontos da grade geográfica próximos a ele. A partir da grade tridimensional resultada pela execução do DAVS é possível criar mapas bi-dimensionais com a estimativa do Conteúdo Eletrônico Total (do inglês *Total Eletronic Content* (TEC)) acumulando os valores das concentrações eletrônicas em altura, como ilustrado na Figura 4.1. O DAVS possui 24 jobs que representam cada uma das 24 horas do dia com a previsão do Conteúdo Eletrônico Total (PETRY; SOUZA; VELHO, 2011).

5 IMPLEMENTAÇÕES

Neste capítulo são apresentadas informações importantes a respeito dos quatro algoritmos de escalonamento que foram implementados e utilizados neste trabalho. Também são apresentadas as metodologias utilizadas em cada um dos algoritmos a respeito do funcionamento no cluster utilizado.

5.1 Algoritmos de escalonamento implementados

Foram implementados quatro algoritmos de escalonamento, o *FIFO*, *SJF*, *EASY-backfilling* e *Fattened backfilling*, apresentados nas próximas subseções. Todos os algoritmos foram desenvolvidos na linguagem *Shell Script*, seguindo a documentação de cada algoritmo.

Além do algoritmo FIFO, que respeita a ordem da fila de execução, foram implementados outros três algoritmos que utilizam técnicas que beneficiam o adiantamento de jobs com requisições ou prioridades menores da fila utilizando informações a respeito de tempo de processamento dos jobs e recursos computacionais disponíveis.

Apesar do SLURM fornecer total integração com os bancos de dados MySQL ou MariaDB, para os experimentos foi utilizado o MySQL de forma independente. Isto deve-se ao fato do cluster já possuir um banco de dados instalado, que anteriormente era utilizado por outro Sistema Gerenciador de Recursos. Foram criados dois banco de dados para controle de informações, um banco utilizado para controle de informações a respeito de recursos computacionais disponível e o outro com o controle dos jobs já submetidos e concluídos, ambos os bancos eram realimentados pelo algoritmo que estava executando.

Devido o ambiente de desenvolvimento utilizado ser utilizado de forma dedicada, sempre que os jobs foram colocados na fila de execução, todos os nós estavam com os recursos computacionais disponíveis, por conta disso, foi preciso adaptar os algoritmos de escalonamento para o problema em questão.

Por conta disso, todas as execuções iniciaram como o algoritmo FIFO, até que todos os recursos computacionais estivessem ocupados e a fila de execução ainda tivesse jobs para executarem. A partir do momento em que não houvesse recursos disponíveis para o próximo job da fila, as técnicas de cada algoritmo de escalonamento eram levadas em consideração. Para os algoritmos que necessitam analisar a fila como um todo, foi considerado o primeiro job da fila sempre o próximo job para executar.

É importante destacar que sempre que um job conclui ou algum job é submetido, o primeiro job da fila é alterado, fazendo assim com que os algoritmos reconsiderem o primeiro job e possíveis adiantamentos de jobs menores. Nas próximos subseções os algoritmos implementados são apresentados de forma detalhada, além de apresentar uma comparação entre os algoritmos ao final deste capítulo.

5.1.1 *FIFO: First in first out*

Caracteriza-se algoritmo FIFO (*First-In-First-Out*) (MACHADO; MAIA, 2004), um modelo que atende os jobs ou processos em ordem de chegada, ou seja, o primeiro job da fila sempre será o primeiro a ser submetido para execução. Em algumas literaturas, o algoritmo FIFO também é conhecido como FCFS (*First-Come, First-Served*). Existem algumas desvantagens na utilização deste algoritmo, principalmente por parte dos jobs menores, onde o algoritmo FIFO não permite nenhuma flexibilização para que jobs com requisições menores possam executar em possíveis lacunas deixadas pelos demais job. Por este motivo, implementou-se o algoritmo FIFO a fim de analisar o desempenho da aplicação quando utilizado um algoritmo de escalonamento que não apresente benefícios para adiantamento de jobs menores para execução.

A Figura 5.1(a) ilustra um gráfico de *Gantt* do funcionamento do algoritmo de escalonamento FIFO. No gráfico é possível observar que alguns jobs já estão em execução e os demais jobs são executados conforme a ordem de chegada das solicitação de recursos.

5.1.2 *SJF: Shortest Job First*

O algoritmo SJF (*Shortest Job First*) (MAO; KINCAID, 1994) permite que jobs com menores requisições de recursos executem primeiro, desde que tenha recursos disponíveis para executar imediatamente. Este algoritmo não leva em consideração possíveis atrasos de jobs antecessores ao job em questão na fila de execução. O algoritmo SJF foi implementado a fim de analisar o desempenho da aplicação quando utilizado um algoritmo de escalonamento que adiante a execução de todos os jobs menores, não respeitando o tempo de execução dos jobs e a ordem da fila para execução.

O algoritmo SJF é muito utilizado em ocasiões onde não é necessário respeitar a ordem da fila de execução, pois o algoritmo leva em consideração as solicitações de recursos computacionais e acaba submetendo os jobs com menores requisições primeiro. Na Figura 5.1(b) é ilustrado um gráfico de *Gantt* do funcionamento do algoritmo de escalonamento SJF, onde é possível observar que a ordem de execução dos jobs é por ordem de tamanho da requisição de recursos.

5.1.3 *EASY-backfilling*

O algoritmo *EASY-Backfilling*, desenvolvido inicialmente pela *Lifka* para o supercomputador paralelo IBM SP1 (LIFKA, 1995), é um algoritmo baseado *backfilling* conservador e é comumente utilizado em cluster de alto desempenho. Diferente do algoritmo *backfilling* conservador que adianta a submissão de jobs com requisições menores ou com menores prioridades da fila de execução somente se o job não atrasar o início de nenhum job

sucessor a ele na fila, o algoritmo *EASY-Backfilling* tem como objetivo melhorar a taxa de utilização atual do cluster, sujeito a alterações na ordem da fila, permitindo que jobs com menor prioridade ou com menor solicitação de recursos possa ser submetidos primeiro, desde que não atrasem o primeiro job da fila para execução (MU'ALEM; FEITELSON, 2001). Devido a aplicação utilizar jobs com requisições diferentes, além do SJF que permite o adiantamento de jobs, foi implementado e avaliado o algoritmo *EASY-Backfilling*, por ser um algoritmo muito utilizado atualmente nos ambientes de alto de desempenho e apresentar resultados positivos quando adiantados jobs menores da fila de execução.

Como nos experimentos realizados, foi utilizado um cluster dedicado, os jobs são submetidos em ordem de chegada (igualmente ao algoritmo FIFO) e assim que o cluster não possuir mais recursos disponíveis para o primeiro da fila são aplicadas as técnicas do algoritmo *EASY-backfilling*. Assim, toda vez que um job é concluído é realizado as mesmas verificações do algoritmo *EASY-backfilling*, analisando se algum job pode ser adiantado sem o atraso do primeiro job da fila para execução. Outro caso, é quando o primeiro job da fila seja submetido, a primeira posição na fila de execução é alterada e com isso é verificado novamente se é possível adiantar a execução de algum outro job. Na Figura 5.1(c) é ilustrado um gráfico de *Gantt* do funcionamento do algoritmo de escalonamento *EASY-backfilling* onde é possível observar que utilizando a técnica de adiantamento de jobs menores foi possível que um dos jobs da fila fossem executados primeiro.

5.1.4 *Fattened backfilling*

O algoritmo *Fattened backfilling* é baseado no algoritmo *EASY-backfilling*. Este algoritmo não leva apenas em consideração a performance do escalonamento, mas também busca a equidade dos jobs em relação ao tempo de espera na fila. A proposta do algoritmo relata que o desempenho dos jobs não é somente o tempo de *throughput*, é preciso também que os jobs tenham um tempo de espera na fila coerente (GÓMEZ-MARTÍN; VEGA-RODRÍGUEZ; GONZÁLEZ-SÁNCHEZ, 2016).

No momento de verificação para adiantamento de algum job da fila é realizado os mesmos procedimentos do algoritmo *EASY-backfilling*, entretanto o algoritmo *Fattened backfilling* adiciona mais um coeficiente a medida de tempo para adiantamento dos job, nomeado *AWT* (*Average Waiting Time*) (GÓMEZ-MARTÍN; VEGA-RODRÍGUEZ; GONZÁLEZ-SÁNCHEZ, 2016). O algoritmo *Fattened backfilling* foi implementado e avaliado por ser muito semelhante ao *EASY-backfilling* e utilizar um coeficiente a mais no momento do adiantamento de jobs, possibilitando assim, que jobs menores tenham uma maior facilidade no momento do adiantamento dos jos quando comparado ao *EASY-backfilling*;

A Figura 5.1 apresenta de forma ilustrativa o comportamento dos quatro algoritmos que foram utilizados neste trabalho e como seria realizado o escalonamento, onde alguns jobs já estão em execução. A Figura 5.1(a) ilustra o algoritmo FIFO, a Figura 5.1(b) re-

presenta o algoritmo SJF, a Figura 5.1(c) o algoritmo EASY-backfilling e a Figura 5.1(d) algoritmo Fattened.

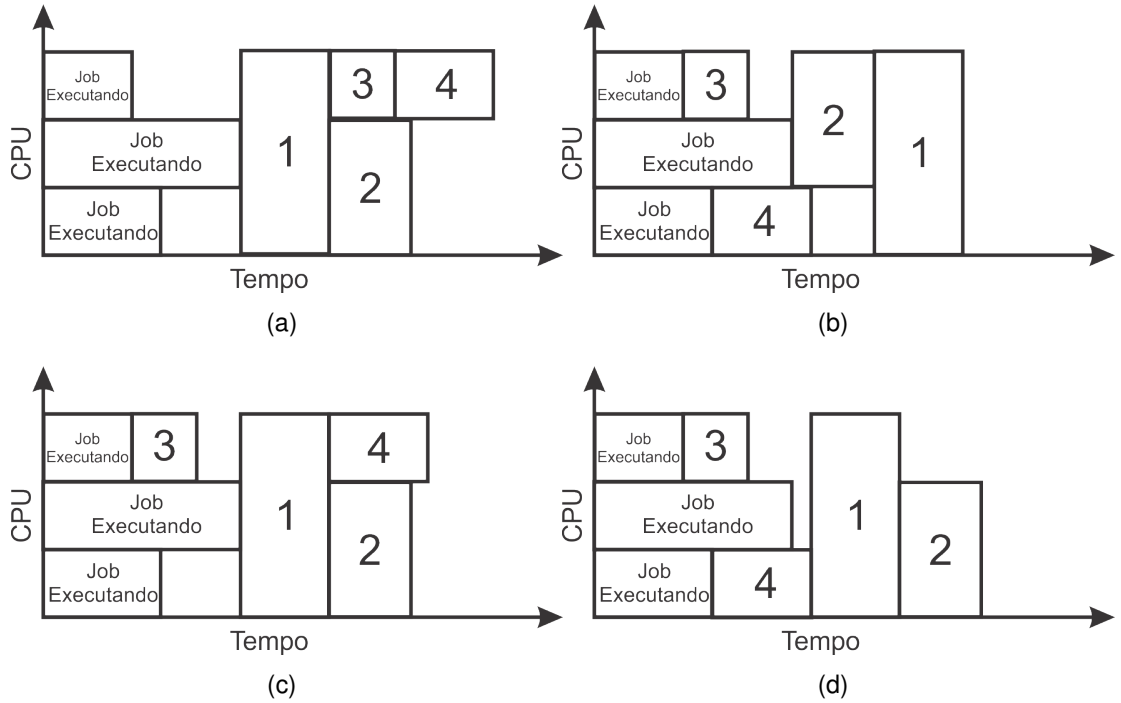


Figura 5.1 – Comparação ilustrativa dos algoritmos de escalonamento FIFO, SJF, EASY-backfilling e Fattened backfilling

6 RESULTADOS EXPERIMENTAIS

Neste capítulo são apresentadas as considerações iniciais sobre os experimentos, apresentando detalhes da plataforma e como foi realizada a configuração e os resultados experimentais utilizando os algoritmos de escalonamento aplicados ao Sistema de Gerenciamento de Recursos SLURM em um cluster. Os algoritmos foram expostos a um cenário com uma aplicação científica de previsão ionosférica e um cenário utilizando jobs de *benchmark* de fluídos físicos e químicos.

6.1 Considerações iniciais sobre os experimentos

Para avaliar o desempenho dos algoritmos de escalonamento de jobs utilizando jobs da previsão ionosférica projetou-se 3 cenários distintos, onde os jobs foram organizados de maneiras distintas na fila de execução. Além de avaliar os algoritmos em um ambiente controlado com jobs do *benchmark* NAS. Em ambos os casos foram analisados tempo de execução, taxa de utilização de CPU, tempo médio de espera e consumo médio de energia. Com os resultados buscou-se avaliar o desempenho dos algoritmos de escalonamento em um cluster com jobs e cenários distintos.

Para ambos os experimentos foram realizadas algumas alterações no arquivo de configuração do SLURM, configurações no cluster e configurado aplicações para medição das métricas de desempenho, as considerações sobre as configurações são apresentadas nas próximas subseção.

6.1.1 Descrição da plataforma

Os experimentos foram realizados em um cluster instalado nas dependências do Centro Regional Sul do Instituto Nacional de Pesquisas Espaciais (CRS-INPE). Durante os experimentos o cluster foi utilizado de forma dedicada a fim de garantir a confiabilidade dos resultados.

O cluster utilizado possui 5 nós de processamento e um nó de controle. Cada nó de processamento possui a seguinte configuração:

- 8 CPUs Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz;
- 74 GB de memória RAM;
- Sistema Operacional CentOS 6.4;
- Sistema Gerenciador de Recursos SLURM;

6.1.2 Arquivo de configuração do SLURM

Foi realizada a instalação e configuração do SLURM em todos os nós de processamento e de controle do cluster, documentada em um manual (PUNTEL; PETRY; CHARÃO, 2018). Juntamente com a instalação, o SLURM possui um arquivo de configuração que deve ser configurado conforme a plataforma e aplicação em que o SGR será utilizado.

O arquivo de configuração do SLURM é onde constam todas as informações de configuração do cluster e como o SLURM está configurado no nó master e nos nós de processamento. Por padrão o SLURM é configurado para que cada nó de processamento seja alocado somente para um único job ou para um grupo de jobs. Por conta das requisições dos jobs utilizados, foi preciso alterar as configurações padrões do SLURM, permitindo assim que os nós compartilhem os recursos computacionais com mais de um job simultaneamente, fazendo com que o job faça requisição por CPU, memória ou core. Por conta disso, as seguintes opções no arquivo de configuração foram alteradas:

- *SelectType*: por padrão esta opção é definida como *select/linear*, onde somente um job seria alocado para todo o nó. Para os experimentos, esta variável foi definida: *SelectType=select/cons_res*, permitindo o compartilhamento do nó;
- *SelectTypeParameters*: esta opção deve ser adicionada no arquivo de configuração do SLURM quando a variável *SelectType* é alterada. Para os experimentos, esta variável foi definida como: *CR_CPU_Memory*, onde os jobs podem solicitar recursos por CPU e memória;
- *DefMemPerCPU*: esta variável também deve ser incluída, pois caso, na solicitação de recursos o job não informar o total de memória, o valor definido nesta variável irá definir a quantidade de memória que será alocada para cada job. Em nosso caso foi definida como 5000 MB;

Com essas configurações é preciso que a requisição de recursos informe o total de CPUs necessárias para execução, caso contrário o SLURM irá alocar 1 CPU para cada job. Contudo, nos experimentos utilizando a previsão ionosférica, os jobs DAVS possuem requisições de 0.5 CPU, com isso além de alterações na forma das solicitações de recursos, também foi preciso adicionar a opção *OverSubscribe=FORCE* na configuração das partições, possibilitando que o nó receba um número maior de jobs do que o seu total de CPUs. Com essa alteração, a configuração dos nós ficou da da seguinte forma:

```
1 NodeName=no00 NodeAddr=192.168.100.250 CPUs=8 RealMemory=72466
2           Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
3 NodeName=no01 NodeAddr=192.168.100.249 CPUs=8 RealMemory=72466
4           Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
5 NodeName=no02 NodeAddr=192.168.100.248 CPUs=8 RealMemory=72465
```

```
6               Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
7  NodeName=no03 NodeAddr=192.168.100.247 CPUs=8 RealMemory=72466
8               Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
9  NodeName=no04 NodeAddr=192.168.100.246 CPUs=8 RealMemory=72466
10              Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
```

Já a configuração da partição ficou da seguinte maneira:

```
1  PartitionName=desenvolvimento Nodes=no0[0-4] Shared=YES
2           Default=YES OverSubscribe=FORCE MaxTime=INFINITE State=UP
```

As demais variáveis do arquivo de configuração padrão do SLURM não foram alteradas. O arquivo de configuração completo utilizado nos experimentos é apresentado no Apêndice A.

6.1.3 Medidores de desempenho

Para medição do consumo de energia, foi utilizado uma ferramenta integrada ao SLURM (SCHEDMD, 2018). Esta ferramenta permite a captação de dados energéticos por medidores do próprio nó ou medidores externos. Devido as CPUs dos nós possuírem medido energético integrado (*Running Average Power Limit* (RAPL)) (ROTEM et al., 2012) foi definido no arquivo de configuração que a leitura seria feita pelo medido interno e a cada 30 segundos o valor seria atualizado. Para a medição as seguintes variáveis foram adicionadas no arquivo de configuração do SLURM:

```
1  JobAcctGatherFrequency=energy=30,task=60
2  AcctGatherEnergyType=acct\_gather\_energy/rapl
3  AcctGatherNodeFreq=60
```

Para medição da taxa de utilização da CPUs foi utilizado a ferramenta *mpstat* (CILIENDO; KUNIMASA; BRASWELL, 2007) (CASALICCHIO; PERCIBALLI, 2017) e para taxa de utilização de memória que realiza a leitura *vmstat*(CILIENDO; KUNIMASA; BRASWELL, 2007), ambas realizam a leitura das informações a cada 10 segundos em um arquivo de *log*. Para medição do tempo de execução e tempo médio de espera foi utilizado a biblioteca *time* do próprio Linux.

6.1.4 Estimativa de tempo de execução dos jobs

Devido os algoritmos de escalonamento utilizados nos experimentos dependerem de informações a respeito do tempo necessário de processamento de cada tipo de job, foi preciso utilizar técnicas para realizar esta estimativa. Existem duas maneiras de definir o tempo de execução de cada um dos jobs, uma delas é solicitar para que o usuário especifique o tempo estimado de processamento do job no momento da submissão. Contudo, isto

costuma ser problemático, pois dificilmente os usuários conseguem especificar o tempo de execução exato ou aproximado de um job(ARNDT et al., 2000) (SKOVIRA et al., 1996).

Na maioria dos casos a estimativa de execução de jobs fornecidas pelos usuários são amplamente superestimadas, deixando recursos computacionais ociosos (GIBBONS, 1997), isto deve-se a um receio por parte do usuário para que seu job não seja interrompido durante a execução. Caso forneça tempo de processamento menor que o suficiente, acabará acarretando em um impacto na utilização dos recursos computacionais e no tempo da aplicação (WONG; GOSCINSKI, 2008). A segunda maneira é medir o tempo de processamento de um job e utilizar informações de tempos das últimas execuções do job para prever tarefas semelhantes no futuro. Uma boa técnica para estimativa de tempo de processamento, é utilizar a média de tempo das últimas duas execuções do job (TSAFRIR; ETSION; FEITELSON, 2007). Neste trabalho utilizou-se a média de tempo das últimas cinco execuções de cada job.

6.2 Experimentos utilizando sistema de previsão ionosférica

Apresentada as etapas da aplicação operacional de previsão ionosférica, para os experimentos foram escolhidas as etapas do SUPIM e interpolação de dados (DAVS), por serem etapas primordiais para a previsão ionosférica. Devido a assimilação de dados possuir dependência de dados oriundos do SUPIM, para os experimentos foram escolhidos dias distintos para cada job. O detalhamento dos jobs SUPIM e DAVS (interpolação de dados) são explicados na seção 4.4.

6.2.1 Metodologia dos experimentos

Para execução foram utilizados 56 jobs nomeados SUPIM e 24 jobs DAVS. Sendo que os 56 jobs SUPIM são homogêneos e cada job representa um grau de longitude da região da América do Sul (-35ºW até -85ºW), selecionada para realização da previsão ionosférica. E os 24 jobs DAVS, onde cada job representa cada uma das 24 horas do dia selecionado para geração de dados para os mapas.

Para execução dos 56 jobs SUPIM foi escolhido um dia onde os jobs sejam classificados como homogêneos. Como explicado anteriormente, na subseção 4.4, os jobs SUPIM dependerem do dado de fluxo solar para realizar a execução. Caso o dado de fluxo solar do dia escolhido para previsão ionosférica seja muito baixo o tempo de execução dos jobs SUPIM será distintos entre eles, podendo até acarretar na não conclusão de algum job SUPIM, invalidando assim o experimento (PETRY et al., 2014).

Para obter um grupo de jobs homogêneo para as execuções foi escolhido o dia 29 de janeiro de 2015, por possuir dados de fluxo solar considerado estável (F10.7 em 123.73 sfu (*solar flux units*) (CANADA, 2018)). Para os jobs DAVS foi escolhido o dia 28 de janeiro

de 2015, por já possuir dados do SUPIM para este dia. A escolha destes dias foi de forma arbitrária, os mesmos experimentos poderiam ser realizados em outros dias com dados de fluxos solares estáveis. Nos experimentos, todos os jobs possuem a mesma prioridade na fila de execução.

A Tabela 6.1 ilustra as requisições de CPU, memória e tempo de execução para os jobs SUPIM e DAVS. As requisições foram baseadas em dados de execuções antigas. Importante ressaltar que a requisição de 0.5 cpu para jobs DAVS é devido os jobs DAVS não apresentarem grande intensidade no período em que estão em execução, por conta disso é possível submeter até dois jobs DAVS por CPU.

Tabela 6.1 – Requisições dos jobs SUPIM e DAVS

Job	CPU	Memória (MB)	Tempo (s)
SUPIM	1	3072	3060
DAVS	0.5	10240	1050

A fim de avaliar a execução dos 80 jobs, foram projetados três cenários. Em cada cenário a fila de jobs foi organizada de forma que os algoritmos realizassem o escalonamento em cenários adversos. Além de uma fila de jobs representando um cenário intermediário, as filas foram projetadas simulando dois cenários extremos, um favorável para adiantamento dos jobs e outro desfavorável para adiantamento dos jobs, isso não representa que os cenários projetados realmente representem o melhor e o pior caso, mas sim o mais próximo destes extremos. Em cada um dos cenários os algoritmos de escalonamento foram executados 30 vezes.

Devido os jobs SUPIM necessitarem de um pré-processamento, o tempo de cada execução dos algoritmos e o início dos monitores de medição de consumo de energia, cpu e memória foram inicializados a partir do momento em que o primeiro job foi submetido. Durante os experimentos o cluster foi reservado para o uso de forma dedicada, garantido que o primeiro job da fila teria recursos computacionais disponível para executar e os monitores de desempenho apresenta-sem o resultado real.

6.2.2 Cenário Intermediário

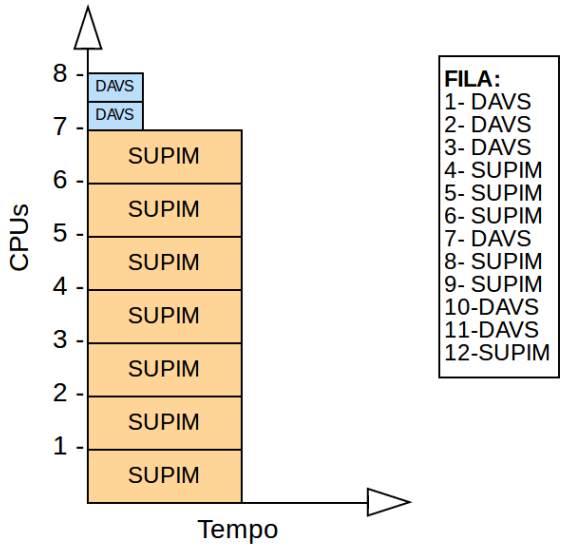
No cenário intermediário, a fila de jobs foi organizada de forma que os algoritmos de escalonamento em períodos conseguissem realizar o adiantamento de jobs menores para que pudessem executarem em lacunas de recursos computacionais deixadas pelos demais jobs, já em outros períodos o adiantamento de jobs não fosse trivial.

A Figura 6.1 ilustra o gráfico de Gantt em um dos casos possíveis no cenário intermediário. O cenário possibilita que em alguns casos os algoritmos de escalonamento

conseguem adiantar jobs menores da fila de execução e outros casos a fila de execução acaba forçando com que os algoritmos submetam os jobs praticamente igual ao FIFO. Na Figura 6.1 é possível observar que alguns jobs já estão em execução, enquanto outros jobs aguardam por recursos computacionais na fila para execução. A partir da conclusão dos jobs em execução, os algoritmos irão tomar as seguintes decisões:

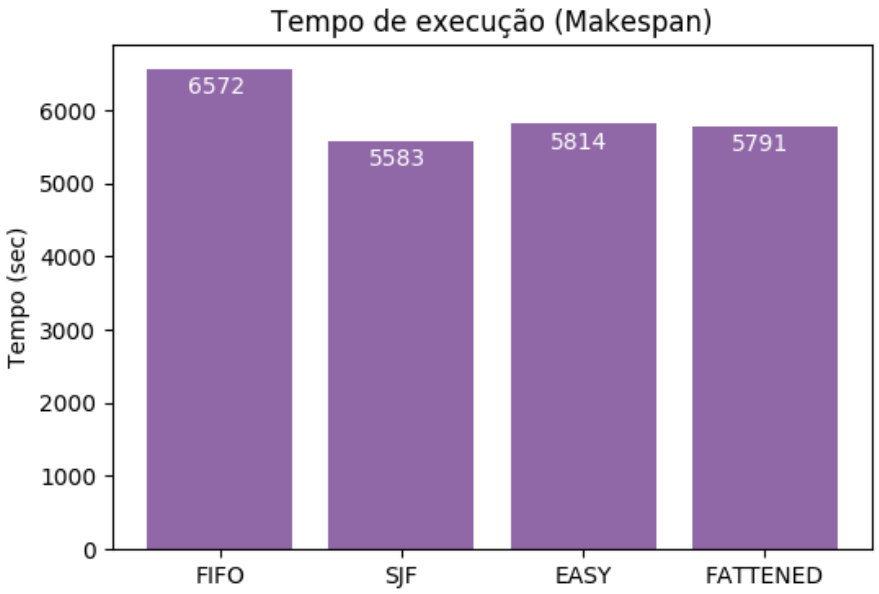
- Logo após a conclusão dos 2 jobs DAVS, os jobs DAVS (posições 1 e 2 da fila) serão submetidos e após seu término, o job 3 (DAVS) da fila é submetido;
- O primeiro job da fila de execução é o job SUPIM (anteriormente posição 4 da fila). A previsão para seu início é após a conclusão de algum job SUPIM que já está em execução. Neste caso, os algoritmos de escalonamento que realizam o adiantamento de jobs menores, poderão adiantar o job DAVS (posição 7 na fila) para execução junto com o job DAVS 3 da fila, pois não irá atrasar o início do job SUPIM;
- Para o restante dos jobs da fila não será possível realizar o adiantamento dos jobs por parte dos algoritmos de escalonamento.

Figura 6.1 – Caso intermediário



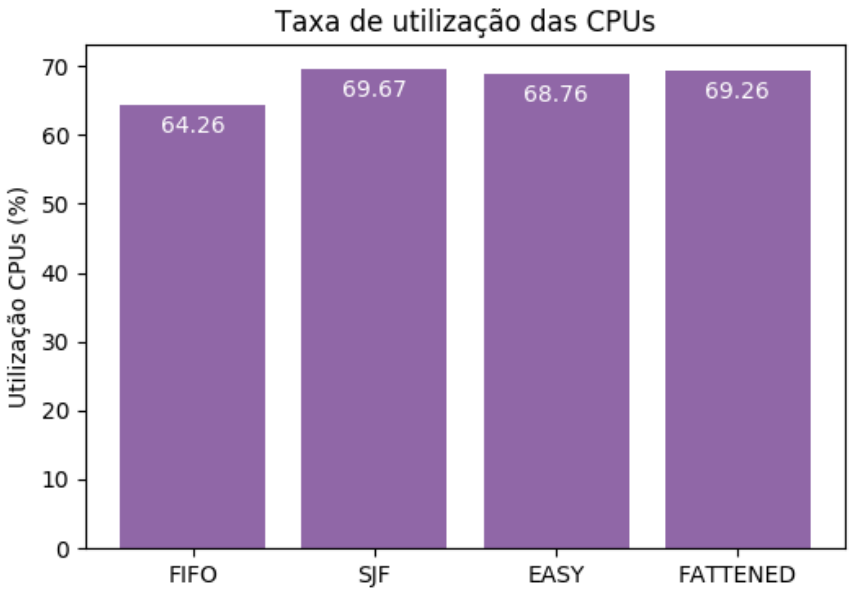
A Figura 6.2 apresenta o tempo de execução dos quatro algoritmos de escalonamento no cenário intermediário. É notório que as execuções com o algoritmo FIFO são prejudicadas, pelo fato do algoritmo não realizar o adiantamento de nenhum job menor da fila, realizado pelos demais. Comparando com os algoritmos que realizam o adiantamento, o algoritmo SJF obteve um resultado relativamente melhor, porém, ainda dentro de um margem muito pequena, isso deve-se ao fato deste algoritmo realizar a submissão dos jobs menores primeiro, diferente dos algoritmos *EASY-backfilling* e *Fattened-backfilling* que em muitos casos precisam respeitar a fila de execução.

Figura 6.2 – Tempo de execução no caso intermediário



A Figura 6.3 apresenta a taxa de utilização da CPUs comparando os quatro algoritmos de escalonamento no cenário intermediário. Assim como na comparação do tempo de execução o algoritmo FIFO apresentou um resultado inferior na comparação da taxa de utilização das CPUs. Já os demais algoritmos obtiveram resultados semelhantes para taxa de utilização de CPUs.

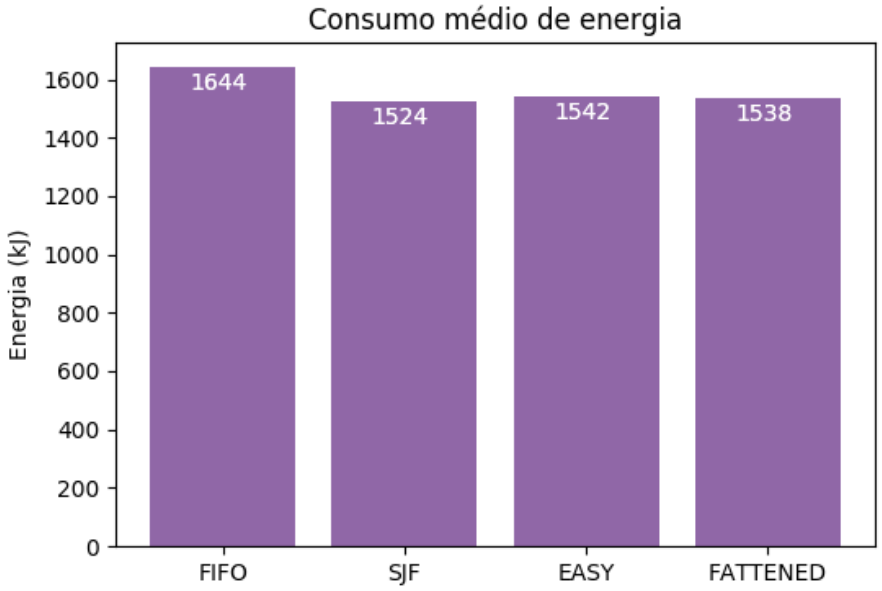
Figura 6.3 – Taxa média de utilização das CPUs no caso intermediário



A Figura 6.4 apresenta o consumo médio de energia, em *kJ*, utilizando os quatro algoritmos de escalonamento. Mesmo com alguns algoritmos apresentando um consumo

energético maior comparado aos demais a diferença é insignificante neste cenário.

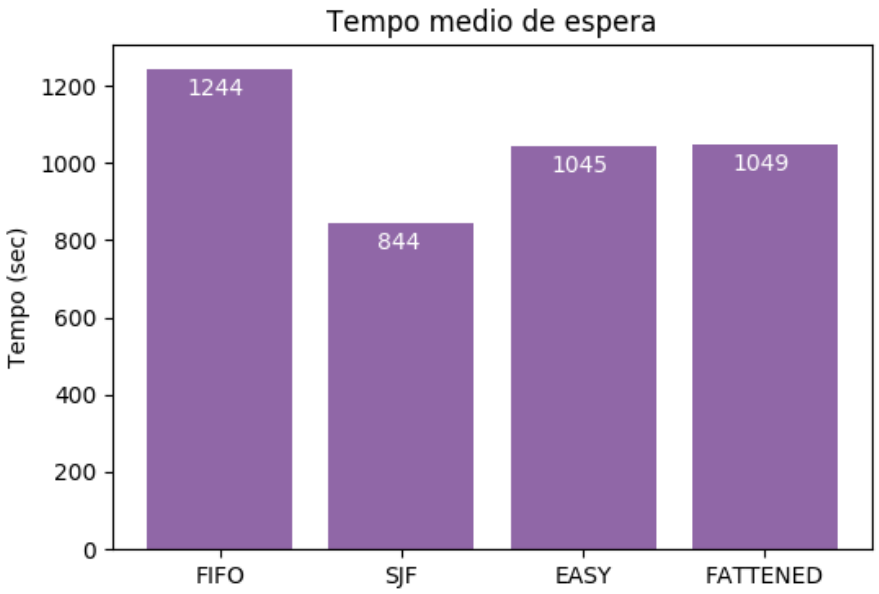
Figura 6.4 – Consumo médio de energia no caso intermediário



A Figura 6.5 ilustra o tempo médio de espera dos jobs na fila de execução. O tempo é calculado a partir do momento em que o job realiza a solicitação de recursos computacionais até que a requisição seja atendida pelo algoritmo de escalonamento. Neste cenário, o algoritmo que mais demorou para atender as requisições dos jobs da fila foi o algoritmo FIFO, isso é devido o algoritmo FIFO não realizar o adiantamento dos jobs menores da fila, sempre respeitando a ordem de chegada dos jobs. Em contrapartida, o algoritmo SJF obteve desempenho superior aos demais, devido ao fato do SJF realizar o adiantamento dos jobs menores sempre que tiver recursos computacionais disponíveis. Já os algoritmos *EASY-backfilling* e *Fattened-backfilling* obtiveram desempenho semelhantes, por utilizarem técnicas de escalonamento muitas parecidas para realizar o adiantamento de jobs menores sempre levando em consideração o não atraso do primeiro job da fila.

Mesmo com uma média do tempo de espera dos jobs na fila é preciso analisar o tempo de espera dos jobs individuais. A Tabela 6.2 apresenta os tempos de espera para os jobs SUPIM e DAVS de forma individual, além de apresentar novamente a média total para análise. É possível observar que quando utilizado os algoritmos SJF, *EASY-backfilling* e *Fattened-backfilling*, as requisições dos jobs DAVS (jobs menores) foram atendidas antes comparado ao algoritmo FIFO. Também é importante ressaltar que mesmo o algoritmo SJF obtendo uma melhora nos tempos de espera dos jobs DAVS, o tempo de espera dos jobs SUPIM foi levemente prejudicado comparando com os demais algoritmos. Isto é devido ao fato dos jobs DAVS serem todos executados primeiro, chegando a um ponto da execução onde a fila para execução é composta somente por jobs SUPIM, assim, os jobs SUPIM são executados sempre que possuírem recursos disponíveis não necessitando aguardar a

Figura 6.5 – Tempo de espera dos jobs na fila no caso intermediário



conclusão dos jobs DAVS.

Tabela 6.2 – Tempo de espera jobs individuais

Algoritmo	Tempo médio de espera	Tempo médio de espera jobs SUPIM	Tempo médio de espera jobs DAVS
FIFO	1244.31	1328.78	1205.82
SJF	844.26	1178.16	202.46
EASY	1045.36	1145.1	940.63
Fattened	1049.06	1131.33	932.3

Além das métricas apresentadas anteriormente, também foi analisada a taxa média de utilização de memória, porém, em todos os algoritmos a taxa de utilização de memória ficou em 8%.

A Tabela 6.3 apresenta os dados de desvio padrão para tempo de execução, taxa de utilização de CPU e consumo de energia após as 30 execuções de cada algoritmo no cenário intermediário.

A Tabela 6.4 apresenta os valores e o desvio padrão da média das execuções para a média do tempo de espera dos jobs na fila e dos jobs individuais.

Tabela 6.3 – Desvio padrão no caso intermediário

Algoritmo	Tempo de Execução(s)	Desvio Padrão	Utilização das CPUs(%)	Desvio Padrão	Consumo de Energia(J)	Desvio Padrão
FIFO	6572.10	237,06	64.26	1,49	1644861	83820
SJF	5583.26	95,81	69.67	2,21	1524696	50410
EASY	5814.93	142,25	68.76	1,42	1542491	19220
Fattened	5791.73	110,47	69.26	1,58	1538883	22720

Tabela 6.4 – Desvio padrão tempo de espera jobs no caso intermediário

Algoritmo	Tempo de espera (s)	Desvio Padrão	Tempo de espera jobs SUPIM	Desvio Padrão	Tempo de espera jobs DAVS	Desvio Padrão
FIFO	1244.32	30,27	1328.78	19,44	1205.82	75,20
SJF	844.26	43,53	1178.16	58,07	202,46	7,74
EASY	1045.36	48,65	1145.10	84.98	940.63	36,11
Fattened	1049.06	35,83	1131.33	76,32	932.3	44,46

6.2.3 Cenário Favorável

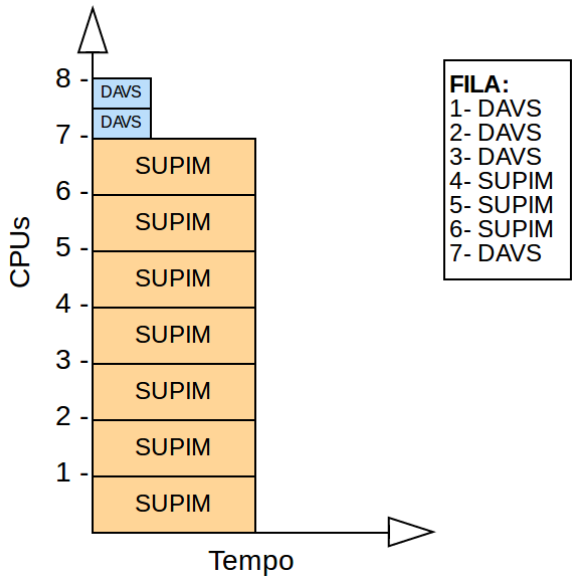
Neste cenário, os jobs DAVS e SUPIM foram organizados na fila de execução de forma que fosse facilitada o adiantamento dos jobs por parte dos algoritmos que utilizam este tipo de técnica. Os algoritmos SJF, *EASY-Backfilling* e *Fattened-backfilling* foram beneficiados neste cenários, pois a organização da fila proporcionava lacunas de recursos computacionais ociosos nos nós de processamento, onde seria possível realizar o adiantamento de jobs menores.

A Figura 6.6 ilustra um gráfico de Gantt em um caso do cenário favorável. Na Figura 6.6 é possível observar que alguns jobs já estão em execução, enquanto outros aguardam na fila para execução. A partir da conclusão dos jobs em execução, os algoritmos irão tomar as seguintes decisões:

- Após a conclusão da execução dos jobs DAVS, os algoritmos irão submeter os jobs DAVS (posições 1 e 2 da fila). Após o término das execuções dos dois jobs DAVS (antes submetidos), os algoritmos irão submeter o job DAVS (posição 3 da fila);
- Como o próximo jobs é um SUPIM e não possui recursos suficientes para sua submissão, os algoritmos que utilizam técnicas de adiantamentos dos jobs, irão submeter o job DAVS, da posição 7 da fila, pois não irá atrasar o início do primeiro job da fila;

- Estes casos acontecem durante toda a execução, o que facilita o adiantamento dos jobs DAVS.

Figura 6.6 – Caso favorável



A Figura 6.7 apresenta os tempos de execução dos quatro algoritmos de escalonamento. Neste cenário, o FIFO obteve desempenho inferior quando comparado com os demais algoritmos, devido ao fato da maneira como a fila de execução foi projetada. Outro fato importante é que os algoritmos *EASY-backfilling* e *Fattened-backfilling*, mesmo realizando o adiantamento de jobs somente em casos onde não houvesse atraso no primeiro job da fila, obtiveram desempenho similar ao SJF, algoritmo que não leva em consideração nenhum outro quesito a não ser recursos computacionais disponíveis.

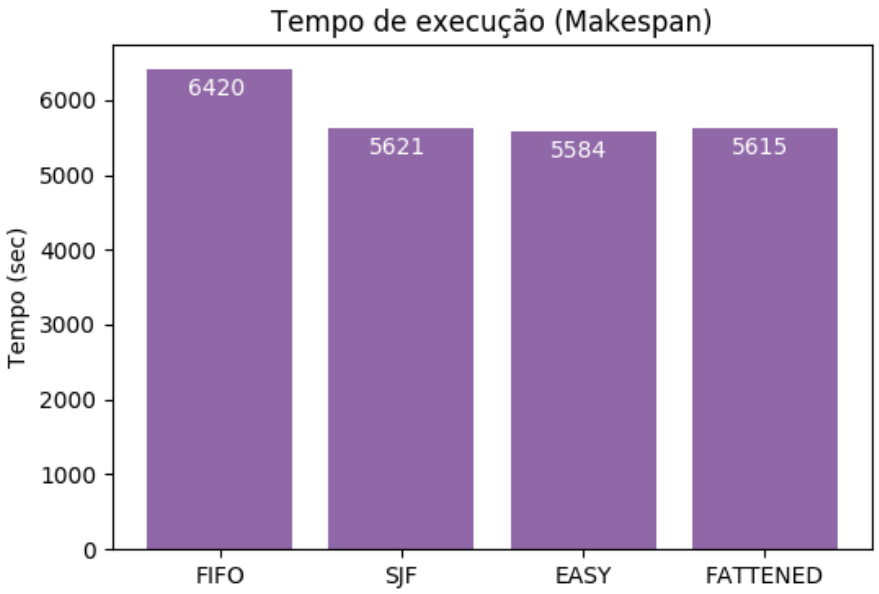
A Figura 6.8 ilustra a taxa média de utilização das CPUs, deixando clara a igualdade entre os algoritmos SJF, *EASY-backfilling* e *Fattened-backfilling* e a inferioridade do algoritmo FIFO neste cenário.

Assim como as demais métricas apresentadas neste caso, no consumo médio de energia o FIFO também foi o algoritmo que obteve desempenho inferior, onde os demais algoritmos obtiveram resultados semelhantes, ilustrado na Figura 6.9.

A Figura 6.10 apresenta os tempos médio de espera dos jobs na fila de execução. Como esperado, o algoritmo FIFO obteve um desempenho quase 50% inferior comparado aos demais. Mesmo os demais algoritmos obtendo resultados muito similares, o algoritmo SJF obteve um desempenho levemente melhor.

A Tabela 6.5 apresenta os tempos de espera dos jobs de forma independente. Importante destacar os tempos médio de espera para os jobs DAVS, onde o algoritmo SJF obteve um desempenho muito superior aos demais, pois o algoritmo submete os jobs com requisições menores primeiro. Neste cenário, é possível observar, mesmo com uma margem pequena, que o algoritmo *Fattened-backfilling* obteve um desempenho superior nos

Figura 6.7 – Tempo de execução no caso favorável



tempos de espera dos jobs DAVS (jobs menores) quando comparado ao *EASY-backfilling*. Representando que em alguns momentos, o *Fattened-backfilling* obteve maior sucesso no adiantamento de jobs menores devido a sua técnica de adiantamento.

Tabela 6.5 – Analise independente do tempo de espera dos jobs no caso favorável

Algoritmo	**Tempo médio de espera**	**Tempo médio de espera jobs SUPIM**	**Tempo médio de espera jobs DAVS**
FIFO	1218.43	1436.7	878.26
SJF	825.90	1171.16	146.96
EASY	883.15	1100.53	550.65
Fattened	890.90	1125.74	489.67

A Tabela 6.6 apresenta os dados de desvio padrão para tempo de execução, taxa de utilização de CPU e consumo de energia após as 30 execuções de cada algoritmo.

A Tabela 6.7 apresenta os valores e o desvio padrão para a média do tempo de espera dos jobs na fila de execução e a média e desvio padrão dos tempos de espera dos jobs de forma individual.

Além das métricas apresentadas, também foram analisados a taxa de utilização de memória, porém, com todos os algoritmos a taxa de utilização de memória ficou em 8%.

Figura 6.8 – Taxa média de utilização das CPUs

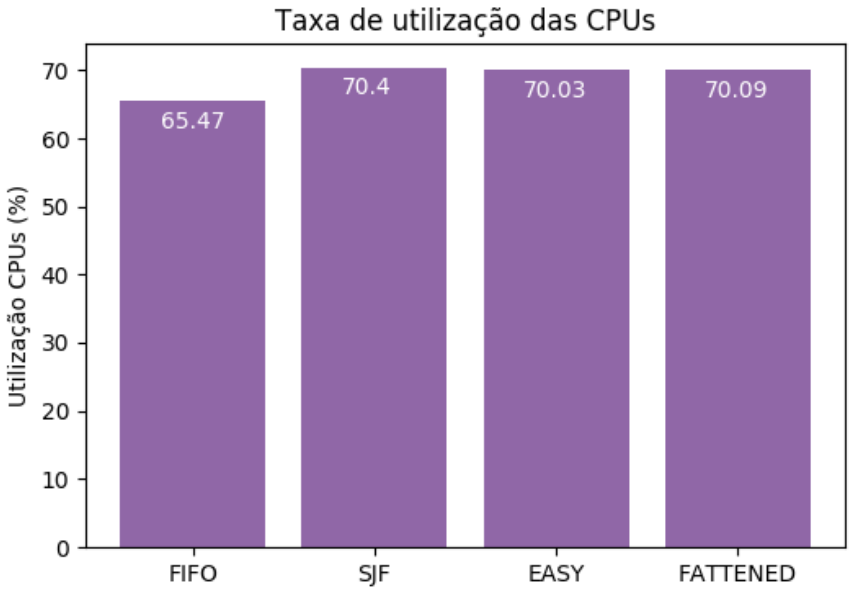


Figura 6.9 – Consumo médio de energia no caso favorável

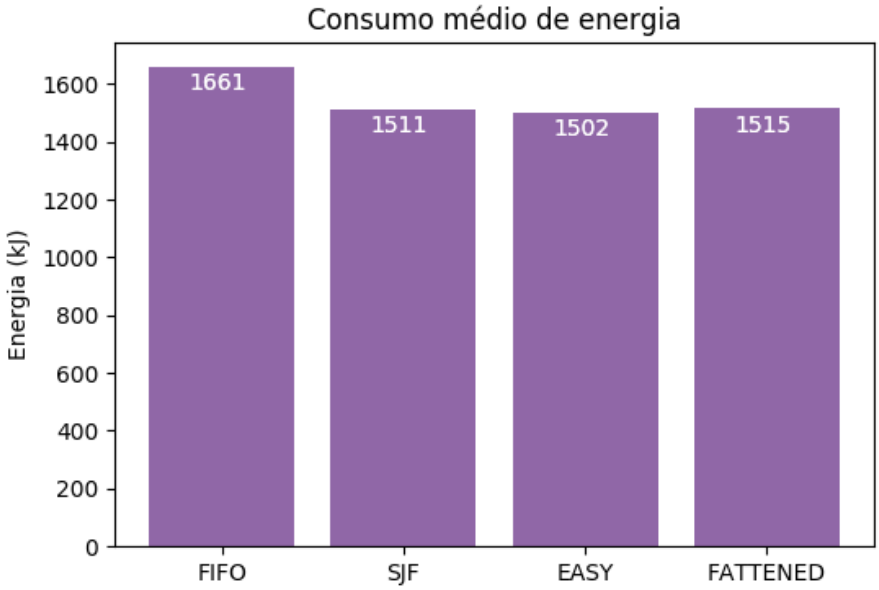


Figura 6.10 – Tempo de espera dos jobs na fila no caso favorável

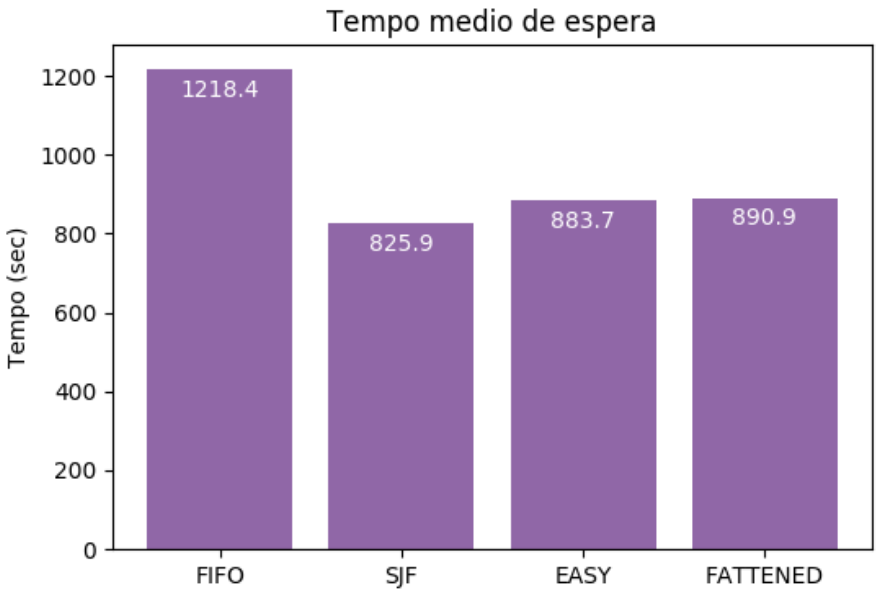


Tabela 6.6 – Desvio padrão no caso favorável

Algoritmo	Tempo de Execução(s)	Desvio Padrão	Utilização das CPUs(%)	Desvio Padrão	Consumo de Energia(J)	Desvio Padrão
FIFO	6420.50	189,67	65.47	1,75	1661222	31959
SJF	5621.22	120,89	70.43	2,37	1511585	22315
EASY	5584.40	105,43	70.03	2,28	1502042	30189
Fattened	5615.09	130,57	70.09	2,17	1515622	26842

Tabela 6.7 – Desvio padrão tempo de espera jobs no caso favorável

Algoritmo	Tempo de espera (s)	Desvio Padrão	Tempo de espera jobs SUPIM	Desvio Padrão	Tempo de espera jobs DAVS	Desvio Padrão
FIFO	1218.43	24,75	1436.70	35,97	878.26	19,15
SJF	825.90	29,42	1171.16	40,75	146,96	3,29
EASY	883.15	23,75	1100.53	46,38	550.65	77,06
Fattened	890.90	43,35	1125.74	39,93	489.67	41,33

6.2.4 Cenário Desfavorável

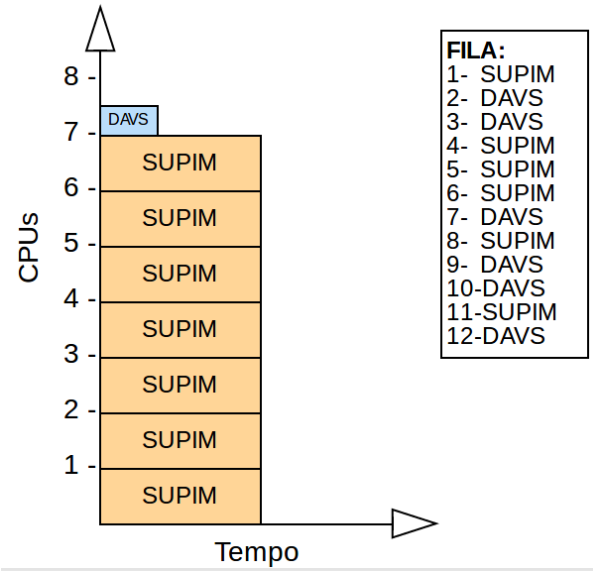
No cenário desfavorável, os jobs SUPIM e DAVS foram organizados de maneira que em momentos da execução os nós de processamento apresentariam lacunas de recursos computacionais ociosos, porém, devido a organização da fila de execução, os algoritmos

de escalonamento teriam dificuldade para o adiantamento de jobs menores.

A Figura 6.11 ilustra um gráfico de Gantt em um caso do cenário desfavorável. Na Figura é possível observar que alguns jobs já estão em execução, enquanto outros aguardam na fila para execução. A partir da conclusão dos jobs em execução, os algoritmos irão tomar as seguintes decisões:

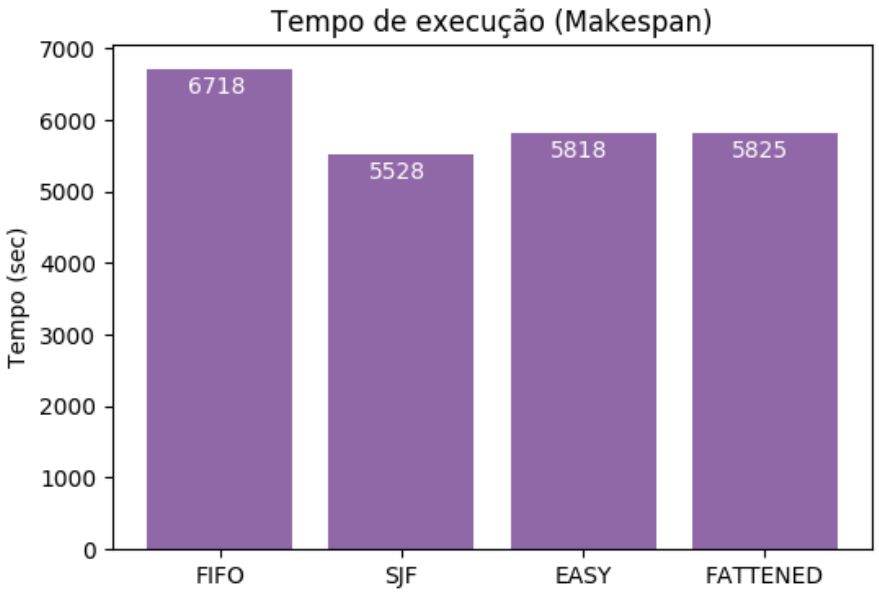
- Após a conclusão da execução do job DAVS, os algoritmos irão submeter o job SUPIM, primeiro da fila. Neste caso, o algoritmo *EASY-backfilling* não conseguiria adiantar o job DAVS (posição 2 da fila), pois iria atrasar, mesmo que alguns segundos ou minutos o início do primeiro job da fila (SUPIM);
- Após a conclusão dos jobs SUPIM, o job irá submeter os jobs praticamente em ordem de chegada, dificultando o adiantamento de jobs menores. Ao final da submissão, uma das CPUs também iria receber somente um jobs DAVS, impossibilitando o adiantamento de outro jobs DAVS.;
- Estes casos acontecem durante toda a execução, dificultando o adiantamento de jobs menores;

Figura 6.11 – Caso desfavorável



A Figura 6.12 ilustra os tempos de execução dos algoritmos de escalonamento no cenário desfavorável. Assim como apresentado nos experimentos anteriores, o algoritmo FIFO obteve desempenho inferior quando comparado aos demais algoritmos de escalonamento. Neste cenário, o SJF obteve desempenho melhor comparado ao *EASY-Backfilling* e ao *Fattened-Backfilling*, este resultado é devido ao fato de que mesmo em um cenário desfavorável, o SJF consegue realizar o adiantamento de jobs menores assim que algum

Figura 6.12 – Tempo de execução no cenário desfavorável



nó de processamento possuir recursos computacionais disponível preenchendo lacunas de recursos computacionais com mais facilidade.

As Figuras 6.13 e 6.14 apresentam a taxa média de utilização das CPUs dos nós de processamento e o consumo médio de energia, respectivamente. Em ambas as métricas o algoritmo de escalonamento FIFO obteve desempenho inferior, enquanto os demais algoritmos apresentaram resultados semelhantes.

O tempo médio de espera dos jobs na fila, ilustrado na Figura 6.15, apresenta um desempenho inferior do FIFO e um desempenho significantemente melhor para o SJF. Em relação ao *EASY-backfilling* e ao *Fattened-backfilling*, os algoritmos tiverem desempenho semelhante entre si, onde conseguiram realizar o adiantamento de somente alguns jobs, entretanto, não se aproximaram do desempenho do algoritmo SJF. Outro ponto importante, é que o algoritmo *Fattened-backfilling* utiliza técnicas justamente para conseguir obter desempenho superior ao *EASY-backfilling* no quesito de adiantamento de jobs, porém, quando exposto a um ambiente desfavorável, o algoritmo não apresentou um desempenho superior.

Assim como na média do tempo de espera, quando analisado os tempos de espera dos jobs de forma independente na Tabela 6.8, há igualdade entre os algoritmos *EASY-backfilling* e *Fattened-backfilling*.

Além das métricas apresentadas, também foram analisados a taxa de utilização de memória, porém, com todos os algoritmos a taxa de utilização de memória ficou em 8%.

A Tabela 6.9 apresenta os dados de desvio padrão para tempo de execução, taxa de utilização de CPU e consumo de energia após as 30 execuções de cada algoritmo.

A Tabela 6.10 apresenta os valores e o desvio padrão da média das execuções para

Figura 6.13 – Taxa média de utilização das CPUs no cenário desfavorável

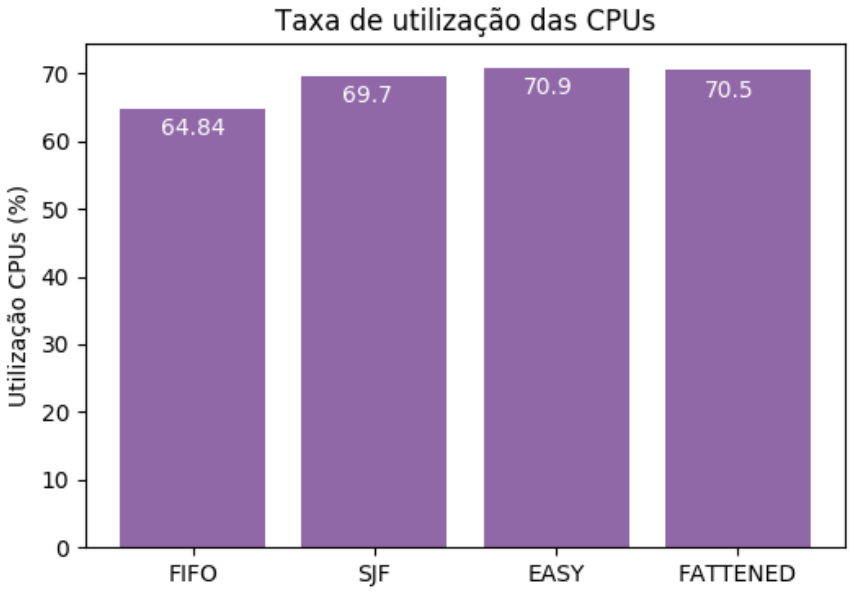


Figura 6.14 – Consumo médio de energia no cenário desfavorável

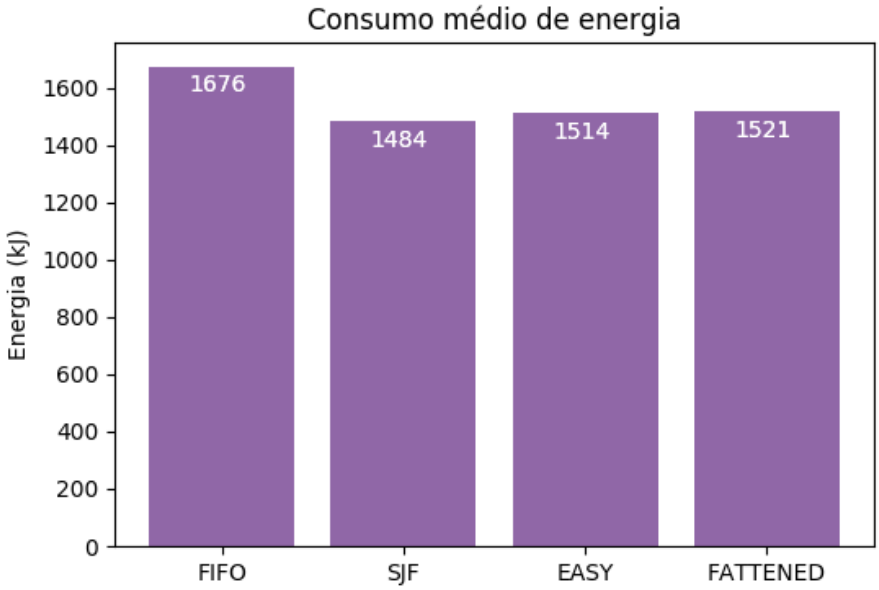


Figura 6.15 – Tempo de espera dos jobs na fila no caso desfavorável

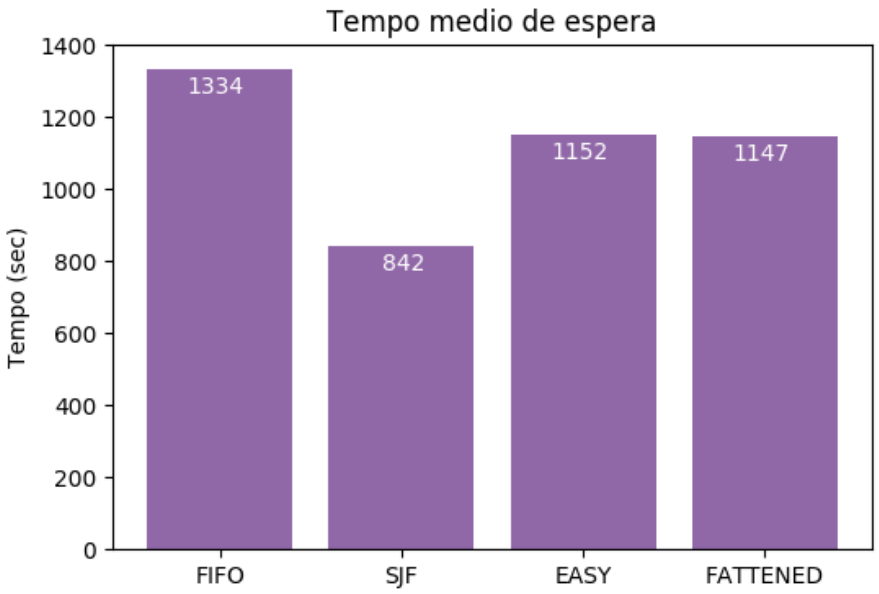


Tabela 6.8 – Analise independente do tempo de espera dos jobs no caso desfavorável

Algoritmo	Tempo médio de espera	Tempo médio de espera jobs SUPIM	Tempo médio de espera jobs DAVS
FIFO	1334.12	1134.84	1946.86
SJF	842.96	1023.76	641.73
EASY	1152.4	1093.66	1572.53
Fattened	1147.96	1061.53	1565.96

Tabela 6.9 – Desvio padrão no caso desfavorável

Algoritmo	Tempo de Execução(s)	Desvio Padrão	Utilização das CPUs(%)	Desvio Padrão	Consumo de Energia(J)	Desvio Padrão
FIFO	6718.56	201,67	64.84	2,32	1618342	35915
SJF	5528.43	120,89	69.74	1,47	1484850	22315
EASY	5818.10	146,15	70.96	1,74	1514125	27861
Fattened	5825.76	133,95	70.51	1,71	1521678	26842

a média do tempo de espera dos jobs na fila e dos jobs individuais.

6.2.5 Análise e discussão dos resultados

Nesta primeira etapa, foram analisados os desempenhos de quatro algoritmos de escalonamento de jobs aplicados ao sistema gerenciador de recursos SLURM em um clus-

Tabela 6.10 – Desvio padrão tempo de espera jobs no caso desfavorável

Algoritmo	Tempo de espera (s)	Desvio Padrão	Tempo de espera jobs SUPIM	Desvio Padrão	Tempo de espera jobs DAVS	Desvio Padrão
FIFO	1334.10	33,34	1134.86	18,94	1946.86	66,38
SJF	842.96	33,76	1023.76	29,85	641.76	29.85
EASY	1152.40	38,08	1093.66	134,84	1572.53	176,24
Fattened	1147.96	43,13	1061.53	108,50	1565.96	176,59

ter dedicado utilizando uma aplicação científica de previsão ionosférica que executa diariamente no Centro Regional Sul do Instituto Nacional de Pesquisas Espaciais.

Os algoritmos de escalonamento foram expostos a três cenários distintos, onde os jobs foram organizados na fila de execução em ordens diferentes. Primeiramente foram expostos a um cenário intermediário, onde os algoritmos em partes da execução tinham uma certa facilidade para conseguir adiantar jobs menores da fila e outras partes o adiantamento não era trivial. Os algoritmos também foram expostos a dois cenários extremos, um onde a ordem dos jobs facilitava o adiantamento e no outro a ordem dificultava o adiantamento de jobs menores, em lacunas deixadas por jobs maiores.

Em todos os cenários, o algoritmo que obteve desempenho inferior foi o algoritmo FIFO, isto é devido ao fato de que este algoritmo justamente não permite realizar o adiantamento de jobs menores para preencher lacunas deixadas pelos demais jobs. Por conta da sua proposta, o algoritmo FIFO dificilmente é utilizado para aplicações que demandam resultados em tempo hábil, em sua maioria utiliza-se o algoritmo FIFO para experimentações ou casos onde os jobs e os recursos são homogêneos.

Um dos pontos importantes analisado é o tempo médio de espera dos jobs na fila de execução, ou seja, o tempo total do momento em que o job solicita recursos computacionais até que a sua requisição é atendida pelo algoritmo de escalonamento. Devido o SJF executar os jobs menores primeiro, o tempo de espera destes jobs acaba influenciando na média do tempo de espera fazendo assim com que o resultado nos tempos médios de espera foram superiores aos demais. No caso favorável, os algoritmos *EASY-backfilling* e *Fattened-backfilling* obtiveram desempenho semelhante ao SJF, comprovando que em cenários onde a fila de execução está em uma ordem que possibilita o adiantamento dos jobs os algoritmos *EASY-backfilling* e *Fattened-backfilling* conseguem realizar operações de adiantamento de forma eficiente, tanto quanto um algoritmo que não leva em considerações os requisitos de atraso dos demais jobs da fila.

De forma geral o algoritmo *Fattened-backfilling* obteve exito na redução do tempo de espera dos jobs menores na fila de execução, assim como é sua proposta. E mesmo o algoritmo não prezando por métricas de desempenho, como taxa média de utilização de

CPU e tempo de execução, nos cenários apresentados, o algoritmo obteve desempenho semelhante ao *EASY-backfilling*.

Contudo, mesmo expondo os algoritmos em cenários distintos com jobs de uma aplicação científica, os resultados não apresentaram diferenças significativas, salvo pela comparação com o algoritmo FIFO. A igualdade dos resultados é devido ao fato da aplicação científica SUPIM-DAVS utilizar somente dois tipos de jobs com diferentes solicitações de recursos e um número relativamente baixo de jobs para realizar o escalonamento. Por conta disso, é necessário expor estes algoritmos aplicados ao SLURM em ambientes com jobs com requisições heterogêneas, assim, conseguindo observar as propostas de cada algoritmo.

6.3 Experimentos em um ambiente controlado

Devido os resultados utilizando os algoritmos de escalonamento na aplicação de previsão ionosférica não apresentarem diferenças significativas a ponto de realmente identificar as técnicas utilizadas pelos algoritmos, nesta seção iremos avaliar os algoritmos em um ambiente controlado, utilizando cargas de trabalhos distintas, a fim das diferenças de escalonamento ficarem evidentes.

Para experimentação foram escolhidos jobs do NAS (*Numerical Aerodynamic Simulation*) *Parallel Benchmarks* (NPB) (NASA, 2018). NPB é um conjunto de programas projetados para avaliar o desempenho de supercomputadores paralelos. Basicamente os *benchmarks* são derivados de aplicações de dinâmica de fluídos (BAILEY et al., 1991).

No NPB1 estão inclusos 8 aplicações que imitam computação e transferência de dados. Estão inclusos no pacote de *benchmark* do NAS1 (BAILEY et al., 1991), (OGURA, 2011):

- IS (*Integer Sort*): *benchmark* paralelo que realiza a ordenação de dados baseado no algoritmo *bucket sort*, exigindo comunicação entre os processos paralelos;
- EP (*Embarrassingly Parallel*): este *benchmark* estima limites superiores aos que podem ser atingidos para o desempenho de ponto flutuante em um computador paralelo;
- CG (*Conjugate Gradient*): este *benchmark* é utilizado para calcular uma aproximação ao menor valor próprio de uma matriz simétrica positiva;
- MG (*Multi-Grid on a sequence of meshes*): o *benchmark* MG utiliza o método *multigrid* V-ciclo que calcula a solução da equação de *Poisson*;
- FT (*Discrete 3D fast Fourier Transform*): *benchmark* resolve equações parciais 3D utilizando FFT espectral.

- BT (*Block Tri-diagonal solver*): parte essencial para dinâmica de fluidos computacionais, este *benchmark* resolve um sistema sintético de equações não lineares;
- SP (*Scalar Penta-diagonal solver*): solução de múltiplos sistemas independentes de equações dominates diagonalmente e não diagonalmente;
- LU (*Lower-Upper Gauss-Seidel solver*): *benchmark* iterativo para resolução de problemas lineares, utiliza o método *symmetric sucessive over-relaxation* para resolver problemas de um sistema diagonal.

6.3.1 Metodologia dos experimentos

Para este experimento foram utilizadas os *benchmarks* SP (*Scalar Penta-diagonal solver*), MG (*Multi-Grid on a sequence of meshes*), FT (*Discrete 3D fast Fourier Transform*) e BT (*Block Tri-diagonal solver*). Após cinco execuções, foi definido as requisições de cada um dos jobs, apresentado na Tabela 6.11. A escolha dos *benchmarks* e do total de CPUs para cada um foi feita de forma arbitrária, somente para forçar o algoritmo a realizar o escalonamento com jobs heterogêneos, é possível realizar os experimentos com outros *benchmarks* da NAS e com diferentes requisições.

Tabela 6.11 – Requisições jobs *Benchmark* NAS

Job	CPU	Memória (MB)	Tempo (sec)
BT	2	1280	180
FT	2	5132	240
MG	4	26624	420
SP	6	5132	680

Para os experimentos foram utilizados 110 jobs, destes 34 jobs BT, 34 jobs FT, 25 jobs MG e 17 jobs SP. A quantidade de jobs com menores requisições foi maior justamente para que os algoritmos de escalonamento conseguissem realizar o maior número de adiantamentos de jobs possível. Assim como nos experimentos utilizando jobs SUPIM e jobs DAVS, o cluster foi utilizado de forma dedicada. Os algoritmos foram expostos a um cenário com os jobs para realizar o escalonamento, cada algoritmo de escalonamento foi executado 30 vezes.

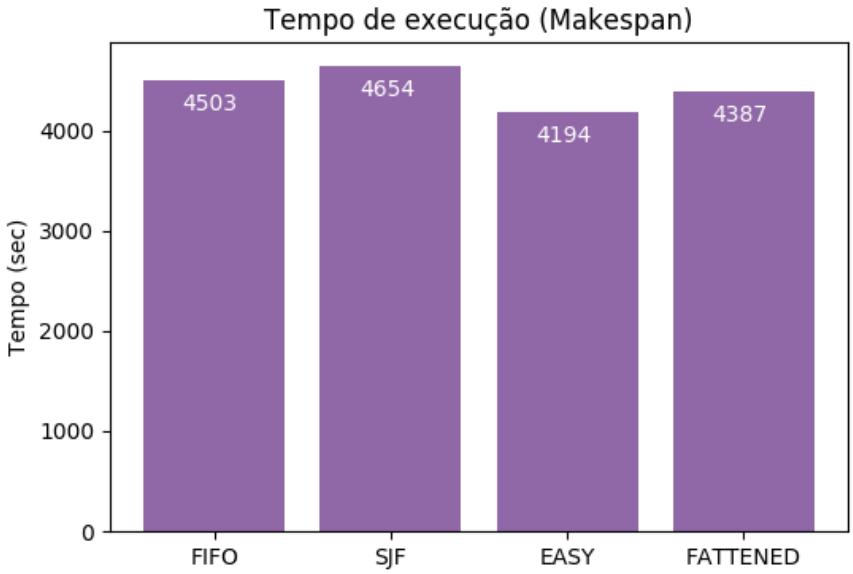
Para execução, nos jobs BT, FT e MG foram utilizados os dados padrões de entrada, já no SP o numero de interações foi alterado para 160 e o trabalho do problema foi alterado para 256x 256x 256. A alteração nos dados de entrada dos jobs SP foi devido a observação de que com os dados padrões de entrada, os jobs concluíam em um tempo muito baixo e como o objetivo deste experimento é justamente realizar o escalonamento com jobs com

requisições distintas foi preciso forçar nos parâmetros iniciais para obter jobs com tempos e requisições maiores.

6.3.2 Análise e discussão dos resultados

O tempo de execução utilizando os quatro algoritmos de escalonamento é ilustrado na Figura 6.16. Neste cenário, é possível observar que o algoritmo *EASY-backfilling* obteve um desempenho de aproximadamente 10% melhor em relação ao SJF, isto é devido ao fato de que o algoritmo *EASY-backfilling* respeitou a fila de execução e executou os jobs maiores em sua ordem e realizou o escalonamento de jobs menores para executarem em lacunas de recursos computacionais deixados pelos demais jobs. Diferente do algoritmo SJF que acaba adiantando todos os jobs menores, resultando em um atraso para os jobs com requisições maiores executarem, já que devido suas solicitações de recursos não permitem que utilizem lacunas de recursos computacionais deixadas pelos jobs menores. Também, neste cenário o FIFO obteve desempenho melhor, mesmo que reduzido, em relação ao SJF justamente por executar os jobs maiores em seu tempo e não atrasá-los o início. Já o algoritmo Fattened-backfilling obteve desempenho como o esperado, já que é um algoritmo que não preza exatamente pelo desempenho e sim pela média de tempo de espera.

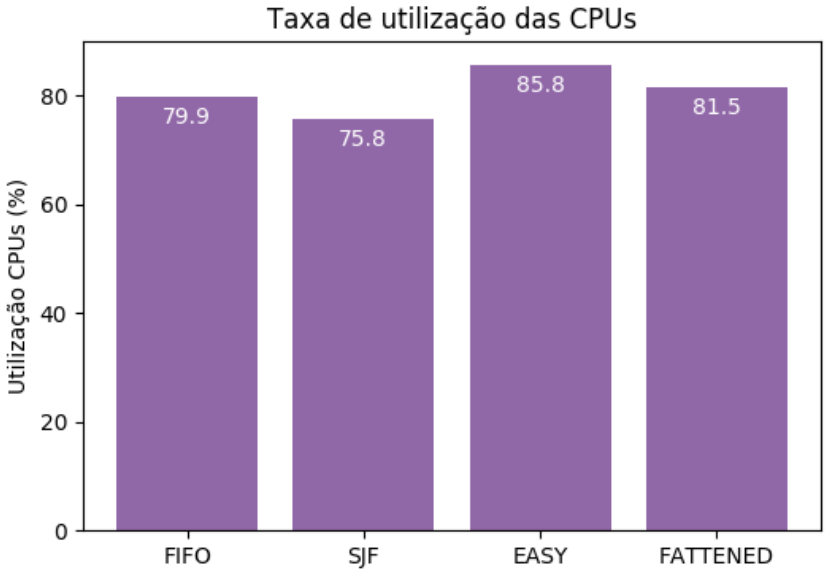
Figura 6.16 – Tempo médio de execução no cenário controlado



A Figura 6.17 ilustra os dados da taxa de utilização das CPUs. Nesta métrica os algoritmos *EASY-backfilling* e *Fattened-backfilling* apresentaram um melhor desempenho quando comparado com os demais algoritmos. No caso do *EASY-backfilling* o aumento da taxa de utilização das CPUs em aproximadamente 13% quando comparado ao SJF. Já

o algoritmo *Fattened-backfilling* mesmo não buscando a melhora na utilização de CPUs apresentou um resultado superior ao SJF. O desempenho inferior no algoritmo SJF foi resultado da técnica utilizada, pois como o algoritmo atrasa a submissão dos jobs com requisições maiores, acaba que ao final das execuções um número pequeno de jobs SP esta executando em alguns nós. A partir de observações foi possível observar que em alguns casos somente um ou dois jobs maiores estavam executando por nó de processamento, resultando assim em uma baixa na taxa de utilização de CPUs.

Figura 6.17 – Taxa de utilização das CPUs no cenário controlado



Quando comparado a taxa média de utilização de memória, ilustrado na Figura 6.18, é possível observar que o algoritmo *EASY-backfilling* obtém resultado superior aos demais, apresentando um resultado melhor de aproximadamente 20% quando comparado ao algoritmo SJF.

Quando comparado o consumo médio de energia, apresentado na Figura 6.19 , os algoritmos *EASY-backfilling* e *Fattened-backfilling* apresentaram um consumo inferior comparado aos demais, porém, pouco significativo.

Na Figura 6.20 são apresentados os tempos médio de espera na fila dos jobs. Neste caso é possível observar que mesmo que o algoritmo SJF adiantando todos os jobs menores para executarem primeiro, o algoritmo Fattened-backfilling apresentou um desempenho superior, isto é por conta de utilizar um método onde permite que mais jobs menores ou com prioridades menores consigam ser adiantados.

A Tabela 6.12 apresenta os tempos médios de espera total e de forma individual. Importante ressaltar que o algoritmo SJF obteve desempenho consideravelmente superior ao *EASY-backfilling* somente quando comparado os jobs menores (FT e BT). Quando comparado os algoritmos *EASY-backfilling* e *Fattened-backfilling*, é possível observar que

Figura 6.18 – Taxa média de utilização de memória

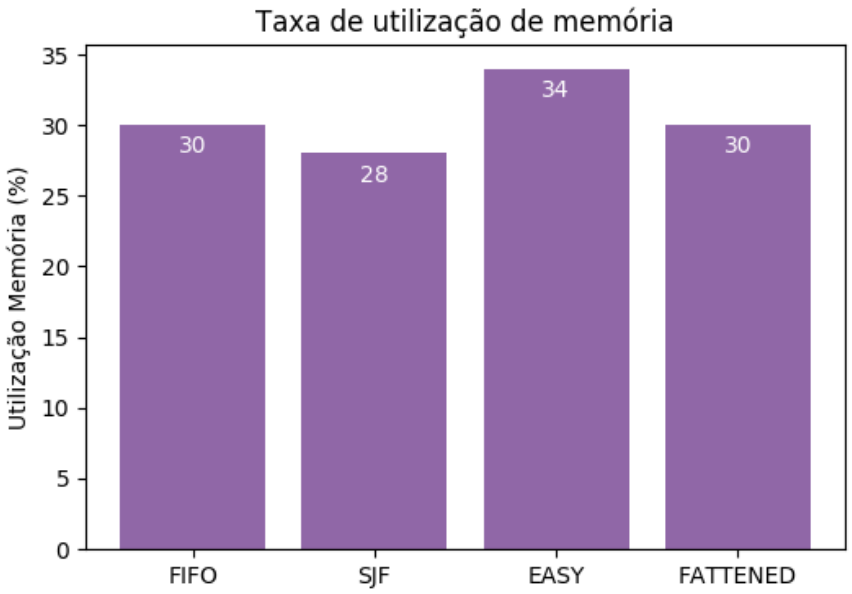
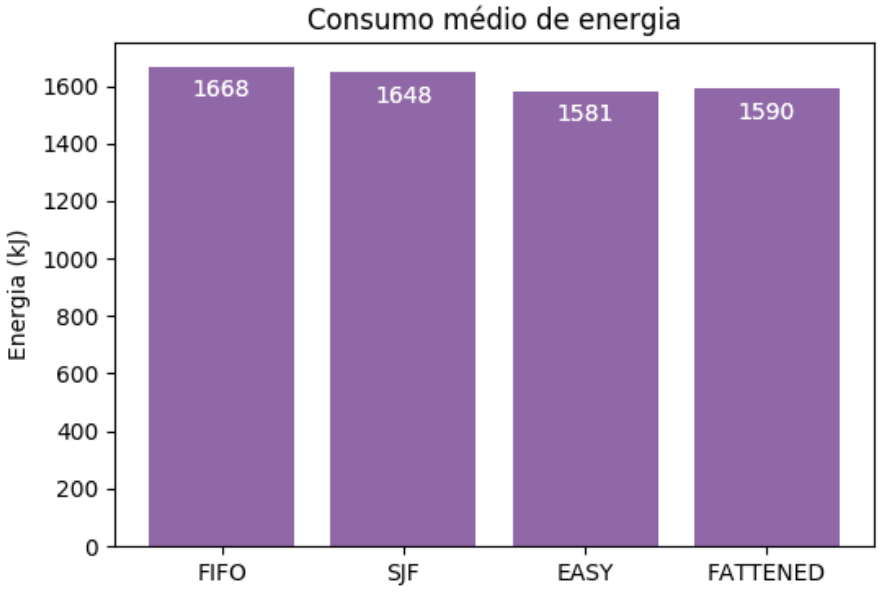


Figura 6.19 – Consumo médio de energia no cenário controlado



o *Fattened-backfilling* obteve desempenho inferior somente no tempo de espera dos jobs SP, justamente porque o algoritmo possibilita que os jobs menores tenham uma maior facilidade para o adiantamento, reduzindo assim, consideravelmente os tempos de espera dos jobs MG, FT e BT.

De forma geral, o algoritmo FIFO obteve desempenho inferior aos demais neste cenário. Um ponto a destacar-se é o fato de que o algoritmo *Fattened-backfilling* mesmo não tendo como objetivo principal a melhora nas métricas de desempenho computacio-

Figura 6.20 – Tempo médio de espera dos jobs na fila no cenário controlado

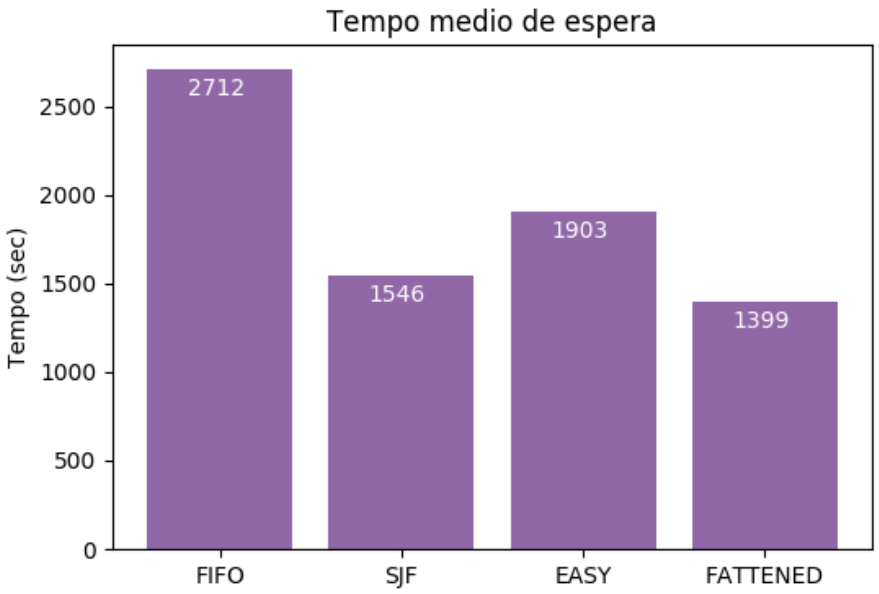


Tabela 6.12 – Tempo de espera jobs individuais

Algoritmo	Todos os jobs	SP	MG	FT	BT
FIFO	2712	1486	2134	3181	3360
SJF	1546	1966	1974	1253	1357
EASY	1903	1432	2005	1907	2106
Fattened	1399	1912	1657	1144	1246

nal, o algoritmo obteve resultados semelhantes ao *EASY-backfilling*. Já o algoritmo SJF foi prejudicado neste experimento, justamente por dar preferência para os jobs menores, atrasando assim o início dos jobs maiores, o que acaba causando um atraso para conclusão dos jobs e consequentemente diminuição na taxa de utilização das CPUs, já que quando os jobs maiores estiverem executando, não há jobs menores na fila de execução para processamento nas lacunas de recursos computacionais.

A Tabela 6.13 apresenta os dados de desvio padrão para tempo de execução, taxa de utilização de CPU e consumo de energia após as 30 execuções de cada algoritmo.

A Tabela 6.14 apresenta os valores e o desvio padrão da média das execuções para a média do tempo de espera dos jobs na fila, dos jobs SP e dos jobs MG.

A Tabela 6.15 apresenta os valores e o desvio padrão da média das execuções para a média do tempo de espera dos jobs FT e BT.

Tabela 6.13 – Desvio padrão no cenário controlado

Algoritmo	Tempo de Execução(s)	Desvio Padrão	Utilização das CPUs(%)	Desvio Padrão	Consumo de Energia(J)	Desvio Padrão
FIFO	4503.36	93,05	79.92	0,68	1668266	27080
SJF	4656.06	137.57	78.85	1,39	1648928	25340
EASY	4194.80	79,97	85.82	0,56	1581356	17458
Fattened	4387.53	128,12	81.52	0,69	1590218	20708

Tabela 6.14 – Desvio padrão da média dos tempos de espera e dos jobs SP e MG no cenário controlado

Algoritmo	Tempo de espera (s)	Desvio Padrão	Tempo de espera jobs SP	Desvio Padrão	Tempo de espera jobs MG	Desvio Padrão
FIFO	2712.73	60,68	1486.40	29,54	2134.06	54,61
SJF	1546.5	49,22	1966.96	96,72	1974.03	109,09
EASY	1903.90	20,81	1432.90	21,66	2005.83	27,27
Fattened	1399.43	112.82	1912.93	33,78	1657.56	75,75

Tabela 6.15 – Desvio padrão do tempo de espera dos jobs FT e BT no cenário controlado

Algoritmo	Tempo de espera jobs FT	Desvio Padrão	Tempo de espera jobs BT	Desvio Padrão
FIFO	3181.46	80,04	3360.76	73.03
SJF	1253.16	82,53	1357.53	95,91
EASY	1907.66	32,52	2106.13	34,04
Fattened	1144.16	183,26	1246.16	212.77

7 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo implementar e avaliar o desempenho de algoritmos de escalonamento aplicados ao Sistema Gerenciador de Recursos SLURM, em dois cenários utilizando jobs distintos. Em um ambiente científico que executa diariamente um sistema de previsão ionosférica e em outro cenário com um grupo de *benchmarks* do NAS (*Numerical Aerodynamic Simulation*). Os experimentos foram realizados em um cluster do Centro Regional Sul do Instituto Nacional de Pesquisas Espaciais.

Após a implementação e realização dos experimentos utilizando os algoritmos FIFO, SJF, *EASY-backfilling* e *Fattened-backfilling*, constatou-se um melhor desempenho do algoritmo SJF em diversas métricas analisadas, isto deve-se a técnica de escalonamento utilizada pelo algoritmo. Nos cenários favorável e desfavorável os algoritmos *EASY-backfilling* e *Fattened-backfilling* realizaram o adiantamento de jobs de forma eficaz, conseguindo em algumas métricas desempenho semelhante ao SJF.

Mesmo que os algoritmos de escalonamento apresentaram desempenhos semelhantes quando expostos a cenários adversos em uma aplicação científica, os resultados não apresentaram de forma realmente clara as propostas utilizadas por cada um dos algoritmos de escalonamento avaliados. Por conta disso, além da utilização de jobs de uma aplicação científica, os algoritmos foram expostos a um cenário controlado com jobs de um *benchmark*.

Quando expostos a uma variedade maior de requisições dos jobs, observou-se com mais clareza os resultados das técnicas utilizadas por cada um dos algoritmos. Principalmente do algoritmo *Fattened-backfilling* onde foi possível observar com clareza a principal finalidade do algoritmo que busca uma equidade entre os tempos de espera dos jobs. Além disso, neste caso, o tempo de execução do *EASY-backfilling* foi superior comparado aos demais, isto deve-se ao fato de que o algoritmo realizou de forma eficiente os adiantamentos de jobs menores, preenchendo lacunas de recursos computacionais. Neste cenário, também percebeu-se uma queda de rendimento dos algoritmos de escalonamento FIFO e SJF que utilizam técnicas menos sofisticadas, em muitos casos não conseguindo otimizar a utilização dos recursos computacionais.

Analisando o desempenho dos algoritmos de escalonamento de forma geral observou-se que o desempenho da aplicação melhorou significativamente quando utilizado algoritmos de escalonamento que realizam o adiantamento de jobs para executarem lacunas de recursos computacionais ociosos quando comparados a algoritmo de escalonamento menos sofisticados. O que favorece muito o desempenho e a eficiência para obtenção dos resultados da aplicação científica de previsão ionosférica e no cenário controlado. Mesmo que neste cenário científico o desempenho foi favorável quando utilizado estes algoritmos é preciso que se analise os requisitos de cada aplicação e plataforma para escolha do melhor algoritmo de escalonamento.

Como trabalhos futuros, sugere-se a avaliação destes e de outros algoritmos de escalonamento aplicados a sistemas gerenciadores de recursos em ambientes de computação científica diversos, compreendendo os aspectos de cada aplicação e suas necessidades. Recomenda-se a implementação de outros algoritmos de escalonamento aplicados ao SGR SLURM utilizando as diretivas de preparação do SLURM e do ambiente de alto desempenho para realização dos experimentos.Além disso, sugere-se que estes e outros algoritmos de escalonamento sejam implementados e aplicados a outros sistemas gerenciadores de recursos disponíveis, além de avaliar outras métricas no ambiente computacional, como *cloud* e *grid*.

REFERÊNCIAS BIBLIOGRÁFICAS

ABAWAJY, J. H. Dynamic parallel job scheduling in multi-cluster computing systems. In: SPRINGER. **International Conference on Computational Science**. Disponível em: https://link.springer.com/chapter/10.1007/978-3-540-24685-527–34.

ABAWAJY, J. H.; DANDAMUDI, S. P. Parallel job scheduling on multicluster computing systems. In: IEEE. **Proceedings IEEE International Conference on Cluster Computing**. Disponível em: https://ieeexplore.ieee.org/abstract/document/1253294, 2003. p. 11.

ANGLING, M.; CANNON, P. Assimilation of radio occultation measurements into background ionospheric models. **Radio Science**, Wiley Online Library, v. 39, n. 1, 2004.

ANGLING, M.; KHATTATOV, B. Comparative study of two assimilative models of the ionosphere. **Radio science**, Wiley Online Library, v. 41, n. 5, 2006.

ARABNEJAD, H.; BARBOSA, J. G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. **IEEE Transactions on Parallel and Distributed Systems**, v. 25, n. 3, p. 682–694, March 2014. ISSN 1045-9219.

ARNDT, O. et al. A comparative study of online scheduling algorithms for networks of workstations. **Cluster computing**, Springer, v. 3, n. 2, p. 95–112, 2000.

BAILEY, D. H. et al. The nas parallel benchmarks. **The International Journal of Supercomputing Applications**, Sage Publications Sage CA: Thousand Oaks, CA, v. 5, n. 3, p. 63–73, 1991.

BENTO, A. **Cloud Computing Service and Deployment Models: Layers and Management: Layers and Management**. [S.l.]: IGI Global, 2012.

BULL. **SLURM V2.2 User's Guide**. 1. ed. BULL atos technologies, 07 2010. Acessado em 15 de outubro de 2018. Disponível em: <https://support-q.bull.com/documentation/byproduct/infra/sw-extremcomp/sw-extremcomp-com/>.

BUST, G.; GARNER, T.; GAUSSIRAN, T. Ionospheric data assimilation three-dimensional (ida3d): A global, multisensor, electron density specification algorithm. **Journal of Geophysical Research: Space Physics**, Wiley Online Library, v. 109, n. A11, 2004.

CANADA, N. R. Monthly averages of solar 10.7 cm flux. Acessado em 20 de julho de 2018, 2018. Disponível em: <http://www.spaceweather.gc.ca/solarflux/sx-5-mavg-en.php>.

CARVALHO, E. A. d. Uma analise sobre as metricas para escalonamento dinamico de processos em simulacao distribuìda usando protocolos otimistas. 2013.

CASALICCHIO, E.; PERCIBALLI, V. Measuring docker performance: What a mess!!! In: **Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion**. New York, NY, USA: ACM, 2017. (ICPE '17 Companion), p. 11–16. ISBN 978-1-4503-4899-7. Disponível em: <http://doi.acm.org/10.1145/3053600.3053605>.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. **IEEE Transactions on Software Engineering**, v. 14, n. 2, p. 141–154, Feb 1988. ISSN 0098-5589.

CILIENDO, E.; KUNIMASA, T.; BRASWELL, B. **Linux performance and tuning guidelines**. Disponível em: https://lenovopress.com/redp4285.pdf: IBM, International Technical Support Organization, 2007.

DONG, F.; AKL, S. G. **Scheduling algorithms for grid computing: State of the art and open problems**. http://ftp.qucis.queensu.ca/TechReports/Reports/2006-504.pdf, 2006.

EXELIS. **Scientific Data Visualization Software form ITTVIS**. EUA, Acessado em 27 de agosto de 2018. 2000. Disponível em: <http://www.spacewx.com/solar2000.html>.

EZQUER, R.; JAKOWSKI, N.; JADUR, C. Predicted and measured total electron content over havana. **Journal of Atmospheric and Solar-Terrestrial Physics**, v. 59, n. 5, p. 591 – 596, 1997. ISSN 1364-6826. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1364682696000491>.

FEITELSON, D. G. et al. Theory and practice in parallel job scheduling. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. Disponível em: https://link.springer.com/chapter/10.1007/3-540-63574-2_14, 1997. p. 1–34.

GANDOTRA, I. et al. Cloud computing over cluster, grid computing: a comparative analysis. **Journal of Grid and Distributed computing**, v. 1, n. 1, p. 1–4, 2011.

GEORGIOU, Y. Contributions for resource and job management in high performance computing. **Universite de Grenoble, France**, Citeseer, 2010.

GEORGIOU, Y. et al. A scheduler-level incentive mechanism for energy efficiency in hpc. In: IEEE. **Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on**. Disponível em: https://ieeexplore.ieee.org/abstract/document/7152527, 2015. p. 617–626.

GEORGIOU, Y.; GLESSER, D.; TRYSTRAM, D. Adaptive resource and job management for limited power consumption. In: IEEE. **Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International**. Disponível em: https://ieeexplore.ieee.org/abstract/document/7284402, 2015. p. 863–870.

GIBBONS, R. A historical application profiler for use by parallel schedulers. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. Disponível em: https://link.springer.com/chapter/10.1007/3-540-63574-2_16, 1997. p. 58–77.

GNU, G. Gnu general public license. **http://www. gnu. org/licenses/gpl. html**, Acessado em: 25 de novembro de 2018, 2015.

GÓMEZ-MARTÍN, C.; VEGA-RODRÍGUEZ, M. A.; GONZÁLEZ-SÁNCHEZ, J.-L. Fattened backfilling: An improved strategy for job scheduling in parallel systems. **Journal of Parallel and Distributed Computing**, Elsevier, v. 97, p. 69–77, 2016.

GRADS. **Grid Analysis and Display System (GrADS)**. EUA, Acessado em 27 de agosto de 2018. 2000. Disponível em: <http://iges.org/grads/>.

GREWAL, M. S.; WEILL, L. R.; ANDREWS, A. P. **Global positioning systems, inertial navigation, and integration**. Second Edition: John Wiley & Sons, 2007.

HAMSCHER, V. et al. Evaluation of job-scheduling strategies for grid computing. In: SPRINGER. **International Workshop on Grid Computing**. Disponível em: https://link.springer.com/chapter/10.1007/3-540-44444-0_18, 2000. p. 191–202.

HENDERSON, R. L. Job scheduling under the portable batch system. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. Disponível em: https://link.springer.com/chapter/10.1007/3-540-60153-8_34, 1995. p. 279–294.

INPE. Tec supim (previsão). http://www2.inpe.br/climaespacial/portal/tec-supim-previsao/. Acessado em 31 de julho de 2018, 2018. Disponível em: <http://www2.inpe.br/climaespacial/portal/tec-supim-previsao/>.

JETTE, M. et al. Survey of batch/resource management related system software. **Lawrence Livermore National Laboratory**, 2002.

JONES, J. P.; NITZBERG, B. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. Disponível em: https://link.springer.com/chapter/10.1007/3-540-47954-6_1, 1999. p. 1–16.

KIERTSCHER, S.; ZINKE, J.; SCHNOR, B. Cherub: power consumption aware cluster resource management. **Cluster computing**, Springer, v. 16, n. 1, p. 55–63, 2013.

LIEU, T.; FARHAT, C.; LESOINNE, M. Reduced-order fluid/structure modeling of a complete aircraft configuration. **Computer methods in applied mechanics and engineering**, Elsevier, v. 195, n. 41-43, p. 5730–5742, 2006.

LIFKA, D. A. The anl/ibm sp scheduling system. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. Disponível em: https://link.springer.com/chapter/10.1007/3-540-60153-8_35, 1995. p. 295–303.

LINDTJORN, O. et al. Beyond traditional microprocessors for geoscience high-performance computing applications. **Ieee Micro**, IEEE, v. 31, n. 2, p. 41–49, 2011.

LITZKOW, M. J.; LIVNY, M.; MUTKA, M. W. Condor-a hunter of idle workstations. In: IEEE. **Distributed Computing Systems, 1988., 8th International Conference on**. Disponível em: https://ieeexplore.ieee.org/document/12507, 1988. p. 104–111.

LNCC. Santos dumont architecture. Acessado em 20 de julho de 2018, 2016.

MACHADO, F. B.; MAIA, L. P. **Arquitetura de sistemas operacionais**. LTC: LTC, 2004.

MAO, W.; KINCAID, R. K. A look-ahead heuristic for scheduling jobs with release dates on a single machine. **Computers & operations research**, Elsevier, v. 21, n. 10, p. 1041–1050, 1994.

MCNAMARA, L. Prediction of total electron content using the international reference ionosphere. **Advances in Space Research**, v. 4, n. 1, p. 25 – 50, 1984. ISSN 0273-1177. Disponível em: <http://www.sciencedirect.com/science/article/pii/0273117784904708>.

MITCHELL, M. **An introduction to genetic algorithms**. [S.l.]: MIT press, 1998.

MORENO, R.; ALONSO-CONDE, A. B. Job scheduling and resource management techniques in economic grid environments. In: RIVERA, F. F. et al. (Ed.). **Grid Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 25–32. ISBN 978-3-540-24689-3.

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

NASA. Nas parallel benchmarks. Acessado em 20 de novembro de 2018, 2018. Disponível em: <https://www.nas.nasa.gov/publications/npb.html>.

OGURA, D. R. **Uma metodologia para caracterização de aplicações em ambientes de computação nas nuvens.** 2011. Tese (Doutorado) — Universidade de São Paulo, Disponível em: http://www.teses.usp.br/teses/disponiveis/3/3141/tde-12032012-114518/pt-br.php, 2011.

OMARA, F. A.; ARAFA, M. M. Genetic algorithms for task scheduling problem. In: **Foundations of Computational Intelligence Volume 3**. Disponível em: https://link.springer.com/chapter/10.1007/978-3-642-01085-9_16: Springer, 2009. p. 479–507.

PETRY, A. **Análise e paralelização do código SUPIM**. São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2010. 35 p. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m19@80/2010/07.19.13.43>. Acesso em: 11 set. 2018.

PETRY, A.; SOUZA, J. R. de; VELHO, H. F. de C. Sistema para previsao operacional da dinÂmica da ionosfera baseado no modelo supim. 2011.

PETRY, A. et al. First results of operational ionospheric dynamics prediction for the brazilian space weather program. **Advances in Space Research**, Elsevier, v. 54, n. 1, p. 22–36, 2014.

PUNTEL, F. E.; PETRY, A.; CHARÃO, A. S. **Instalação e configuração do sistema de gerenciamento de recursos SLURM em um cluster**. São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2018. 32 p. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m21c/2018/08.29.16.54>. Acesso em: 26 set. 2018.

PUNTEL, F. E. et al. Preliminary evaluation with ensemble prediction for the supim model. In: PAN-AMERICAN CONGRESS ON COMPUTATIONAL MECHANICS, 1. (PANACM), 2015, Buenos Aires. **Proceedings...** International Center for Numerical Methods in Engineering (CIMNE), 2015. v. 1, p. 1322–1330. ISBN 9788494392825. Disponível em: <http://congress.cimne.com/PANACM2015/frontal/doc/EbookPANACM2015.pdf>. Acesso em: 27 ago. 2018.

ROTEM, E. et al. Power-management architecture of the intel microarchitecture code-named sandy bridge. **Ieee micro**, IEEE, v. 32, n. 2, p. 20–27, 2012.

SCHEDMD. Slurm commerical support and development. Acessado em 20 de julho de 2018, 09 2017. Disponível em: <https://www.schedmd.com/>.

____. Slurm energy accounting plugin api. Acessado em 19 de novembro de 2018, 2018. Disponível em: <https://www.slurm.schedmd.com/acct_gather_energy_plugins.html>.

SHAH, S. M. H.; QURESHI, K.; RASHEED, H. Optimal job packing, a backfill scheduling optimization for a cluster of workstations. **The Journal of Supercomputing**, v. 54, n. 3, p. 381–399, Dec 2010. ISSN 1573-0484. Disponível em: <https://doi.org/10.1007/s11227-009-0332-3>.

SKOVIRA, J. et al. The easy—loadleveler api project. In: SPRINGER. **Workshop on Job Scheduling Strategies for Parallel Processing**. Disponível em: https://link.springer.com/chapter/10.1007/BFb0022286, 1996. p. 41–47.

SLOAN, J. D. **High Performance Linux Clusters**. with OSCAR, Rocks, OpenMosix, and MPI: A Comprehensive Getting-Started Guide: "O'Reilly Media, Inc.", 2004.

STERLING, T. L.; GROPP, W.; LUSK, E. **Beowulf cluster computing with Linux**. [S.l.]: MIT press, 2002.

TALBY, D.; FEITELSON, D. G. Supporting priorities and improving utilization of the ibm sp scheduler using slack-based backfilling. In: IEEE. **Parallel Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings**. Disponível em: https://ieeexplore.ieee.org/abstract/document/760525, 1999. p. 513–517.

TANENBAUM, A. S.; FILHO, N. M. **Sistemas operacionais modernos**. [S.l.]: Prentice-Hall, 1995.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: principles and paradigms**. [S.l.]: Prentice-Hall, 2007.

TOBISKA, W. K. et al. The solar2000 empirical solar irradiance model and forecast tool. **Journal of Atmospheric and Solar-Terrestrial Physics**, Elsevier, v. 62, n. 14, p. 1233–1250, 2000.

TOP500. https://www.top500.org. Acessado em 20 de julho de 2018, 09 2017.

TSAFRIR, D.; ETSION, Y.; FEITELSON, D. G. Backfilling using system-generated predictions rather than user runtime estimates. **IEEE Transactions on Parallel and Distributed Systems**, v. 18, n. 6, p. 789–803, June 2007. ISSN 1045-9219.

VASUPONGAYYA, S.; CHIANG, S.-H. On job fairness in non-preemptive parallel job scheduling. In: CITESEER. **IASTED PDCS**. doi=10.1.1.118.1939, 2005. p. 100–105.

WONG, A. K.; GOSCINSKI, A. M. The impact of under-estimated length of jobs on easy-backfill scheduling. In: IEEE. **Parallel, Distributed and Network-Based Processing,16th Euromicro Conference on**. Disponível em: https://ieeexplore.ieee.org/abstract/document/4457142, 2008. p. 343–350.

WONG, A. K. L.; GOSCINSKI, A. M. Evaluating the EASY-backfill job scheduling of static workloads on clusters. **2007 IEEE International Conference on Cluster Computing**, p. 64–73, 2007. ISSN 1552-5244.

XIE, J. et al. High-performance computing for the simulation of dust storms. **Computers, Environment and Urban Systems**, Elsevier, v. 34, n. 4, p. 278–290, 2010.

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.

YU, K.-M. et al. A fuzzy neural network based scheduling algorithm for job assignment on computational grids. In: SPRINGER. **International Conference on Network-Based Information Systems**. Disponível em: https://link.springer.com/chapter/10.1007/978-3-540-74573-0_55, 2007. p. 533–542.

ZHANG, P.; SHI, X.; KHAN, S. U. Quantcloud: Enabling big data complex event processing for quantitative finance through a data-driven execution. **IEEE Transactions on Big Data**, IEEE, 2018.

ZHOU, S. et al. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. **Software: practice and Experience**, Wiley Online Library, v. 23, n. 12, p. 1305–1336, 1993.

ZHOU, X. et al. Exploring distributed resource allocation techniques in the slurm job management system. **Illinois Institute of Technology, Department of Computer Science, Technical Report**, Citeseer, 2013.

APÊNDICE A – ARQUIVO DE CONFIGURAÇÃO DO SLURM

Neste apêndice é apresentado o arquivo de configuração do SLURM no cluster do Centro Regional Sul do Instituto Nacional de Pesquisas Espaciais.

```
1  # slurm.conf
2  # Arquivo de configuracao do SLURM
3  # Configuracao do CLUSTER
4  # Centro Regional Sul do Instituto Nacional de Pesquisas Espaciais
5  #
6  # Programa de Pos-Graduacao em Ciencias da Computacao
7  # Universidade Federal de Santa Maria
8  #
9  # Fernando Emilio Puntel
10 # 2018
11 #
12 ControlMachine=nogw.localdomain
13 ClusterName=cluster
14 ControlAddr=10.10.6.15
15 #
16 MailProg=/bin/mail
17 ReturnToService=0
18 SlurmctldPort=6817
19 SlurmdPort=6818
20 SlurmdSpoolDir=/var/spool/slurmd
21 SlurmUser=supim-davs
22 AuthType=auth/munge
23 StateSaveLocation=/var/spool
24 SwitchType=switch/none
25 TaskPlugin=task/none
26 MpiDefault=none
27 SlurmctldPidFile=/var/run/slurmctld.pid
28 SlurmdPidFile=/var/run/slurmd.pid
29 ProctrackType=proctrack/pgid
30 #
31 #
32 # TIMERS
33 SlurmctldTimeout=300
34 SlurmdTimeout=300
35 InactiveLimit=0
36 MinJobAge=300
37 KillWait=30
38 Waittime=0
39 #
40 #
41 # SCHEDULING
```

```
42 SchedulerType=sched/backfill
43 SelectType=select/cons_res
44 SelectTypeParameters=CR_CPU #CR_CPU_Memory
45 DefMemPerCPU=5000
46 FastSchedule=0
47 #PriorityType=priority/multifactor
48 #PriorityDecayHalfLife=14-0
49 #PriorityUsageResetPeriod=14-0
50 #PriorityWeightFairshare=100000
51 #PriorityWeightAge=1000
52 #PriorityWeightPartition=10000
53 #PriorityWeightJobSize=1000
54 #PriorityMaxAge=1-0
55 #
56 #
57 # LOGGING AND ACCOUNTING
58 AccountingStorageType=accounting_storage/filetxt
59 AccountingStoreJobComment=YES
60 JobCompLoc=/var/log/slurmData.log
61 JobCompType=jobcomp/filetxt
62 JobAcctGatherType=jobacct_gather/none
63 SlurmctldDebug=3
64 SlurmctldLogFile=/var/log/slurmctld.log
65 SlurmdDebug=3
66 SlurmdLogFile=/var/log/slurmd.log
67
68 JobAcctGatherFrequency=energy=30,task=60
69 AcctGatherEnergyType=acct_gather_energy/rapl
70 AcctGatherNodeFreq=60
71
72 #EnergyAccountingType=energy_accounting/rapl
73 #EnergyAccountingNodeFreq=30
74 #
75 #
76 # COMPUTE NODES
77 NodeName=no00 NodeAddr=192.168.100.250 CPUs=8 RealMemory=72466
78 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
79 NodeName=no01 NodeAddr=192.168.100.249 CPUs=8 RealMemory=72466
80 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
81 NodeName=no02 NodeAddr=192.168.100.248 CPUs=8 RealMemory=72465
82 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
83 NodeName=no03 NodeAddr=192.168.100.247 CPUs=8 RealMemory=72466
84 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
85 NodeName=no04 NodeAddr=192.168.100.246 CPUs=8 RealMemory=72466
86 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
87 PartitionName=desenvolvimento Nodes=no0[0-4] Shared=YES Default=YES
88 OverSubscribe=FORCE MaxTime=INFINITE State=UP
```