UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Marcelo Ferreira de Oliveira Filho

Reengenharia da gestão de cargas de trabalho no iSPD

SÃO JOSÉ DO RIO PRETO
2024

MARCELO FERREIRA DE OLIVEIRA FILHO

Reengenharia da gestão de cargas de trabalho no iSPD

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

São José do Rio Preto, 2024

O48r

Oliveira Filho, Marcelo Ferreira de
Reengenharia da gestão de cargas de trabalho no iSPD / Marcelo Ferreira de Oliveira Filho. -- São José do Rio Preto, 2024
42 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (UNESP), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Aleardo Manacero Junior

1. Cargas de Trabalho. 2. Simulação. 3. Grades Computacionais. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

UNIVERSIDADE ESTADUAL PAULISTA JÚLIO DE MESQUITA FILHO

MARCELO FERREIRA DE OLIVEIRA FILHO

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Aleardo Manacero Junior
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Profa. Dra. Renata Spolon Lobato
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Profa. Dra. Carina Alexandra Rondini
Universidade Estadual Paulista "Júlio de Mesquita Filho"

São José do Rio Preto, 2024

Agradecimentos

Agradeço ao meu orientador, Prof. Dr. Aleardo Manacero Junior, por aceitar o convite para me orientar e por sempre se mostrar disponível com grande dedicação para esclarecer dúvidas e oferecer conselhos e recomendações.

Agradeço a minha grande amiga Mariani, e todos os outros colegas que me incentivaram e apoiaram durante todos esses anos na graduação.

Por fim, agradeço também a todos os professores do campus que tive a oportunidade de conhecer durante esta jornada.

"Fica mais fácil.
Fica um pouquinho mais fácil a cada dia,
mas você tem que fazer todo dia.
Essa é a parte difícil.
Mas fica mais fácil".
(BOB-WAKSBERG, Raphael; Bojack Horseman)

Resumo

A Computação de Alta Performance (HPC) é essencial para resolver problemas complexos no âmbito cientifico, médico ou comercial. As grades computacionais são uma ótima solução para a HPC, fazendo com que diversas ferramentas que simulam essa infraestrutura sejam de suma importância para o estudo da área, como o *iconic Simulator of Parallel and Distributed Systems* (iSPD). O iSPD permite que o usuário utilize da ferramenta sem conhecimento aprofundado de programação, através de sua interface icônica intuitiva de fácil uso. Dentro de todos os componentes presentes no iSPD, há a partição da gestão de cargas de trabalho, que utiliza de parâmetros probabilísticos ou pré-definidos (*traces*) para gerar estas cargas. Este trabalho foca na refatoração deste componente para uma linguagem nova, bem como a implementação de uma nova funcionalidade que permitirá a definição dos parâmetros das cargas de trabalho, o que fará com que testes e execuções em diferentes situações no simulador sejam mais fáceis de serem comparadas, facilitando e melhorando a experiência do usuário ao utilizar o iSPD.

Palavras-chave: Cargas de Trabalho, Simulação, Grades Computacionais

Abstract

High Performance Computing (HPC) is essential for solving complex scientific, medical and commercial problems. Computational grids are a great solution for HPC, making several tools that simulate this infrastructure extremely important for studying the area, such as iSPD. iSPD allows users to use the tool without in-depth programming knowledge, thanks to its intuitive, easy-to-use iconic interface. Among all the components present in iSPD, there is the partition of workload management, which uses probabilistic parameters or predefined traces to generate these loads. This work focuses on refactoring this component into a new language, as well as implementing a new feature that will allow the parameters of the workloads to be defined, which will make it easier to compare tests and executions in different situations in the simulator, facilitating and improving the user experience when using iSPD.

Keywords: Workload, Simulation, Grid Computing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

HPC	*High Performance Computing*
iSPD	*iconic Simulator of Parallel and Distributed Systems*
GSPD	*Grupo de Sistemas Paralelos e Distribuídos*
GUI	*Graphical User Interface*
SWF	*Standard Workload Format*
GWF	*Grid Workload Format*
ROSS	*Rensselaer's Optimistic Simulation System*
RMSE	*Raiz do Erro Quadrático Médio*

Sumário

1 Introdução

A computação de alto desempenho ou HPC (do inglês *High-performance computing*) é uma ferramenta poderosa que permite a combinação de recursos computacionais para resolver problemas extremamente complexos. Ela é vital porque permite o processamento de grandes quantidades de dados e também a execução de cálculos complexos em velocidades que seriam impossíveis com computadores convencionais. Sendo assim, a computação de alto desempenho desempenha um papel fundamental na resolução de desafios complexos e na promoção do avanço tecnológico em diversas áreas, impulsionando a inovação e o desenvolvimento em escala global.

Sua importância é vista nos mais diversos domínios do conhecimento e nos campos de pesquisas científicas, destacando-se principalmente pela sua capacidade de simular fenômenos complexos e analisar vastos conjuntos de dados, tendo aplicações em áreas como desenvolvimento de automóveis e aeronaves, modelagem climática e *machine learning*. Na área médica, destaca-se a pesquisa genômica, em que é necessário alto poder computacional para executar sequenciamento e análise de DNA [Schmidt e Hildebrandt 2017]. A HPC também é utilizada na indústria, como por exemplo no setor de entretenimento e mídia, utilizando de computadores muito eficientes para edição e manutenção de efeitos visuais.

Devido a esta alta demanda da utilização de máquinas potentes e capacitadas, foi necessário o desenvolvimento de técnicas que otimizam recursos e ferramentas, com o objetivo de melhorar o desempenho e ter uma eficiência maior. Dentre elas existem os sistemas com arquiteturas paralelas e distribuídas, computação em nuvem, supercomputadores e também as grades computacionais, que são o foco deste projeto.

As grades computacionais são uma forma de computação distribuída em que uma rede de computadores promove alto desempenho através do compartilhamento de recursos entre as máquinas da rede [Wang, Jie e Chen 2018]. A capacidade que as grades computacionais possuem ocasionou sua vasta utilização, o que fez com que pessoas com baixo conhecimento de HPC encontrassem a área, levando assim a criação de ferramentas que simulam grades computacionais. As ferramentas SimGrid [Casanova, Legrand e Quinson 2008], GridSim [Buyya e Murshed 2002], GangSim [GangSim 2012], OptorSim [OptorSim 2012] e Bricks [Takefusa e Matsuoka 2000], e o iSPD (iconic Simulator of Parallel and Distributed systems) [Manacero et al. 2012] são notáveis ferramentas de simulação de grades computacionais.

O iSPD, desenvolvido pelo Grupo de Sistemas Pararelos e Distribuídos [GSPD 2012], tem como objetivo viabilizar e facilitar o gerenciamento e criação de escalonadores, modelos de grades e cargas de trabalho.

1.1 Motivação

Com o intuito de diminuir o tempo computacional da simulação, foi realizada a reengenharia do iSPD de forma que seja possível executar de maneira distribuída. Sendo assim, a reengenharia alterou o modelo de funcionamento baseado em memória compartilhada para um modelo de funcionamento em memória distribuída.

Assim, todo modelo de gestão de trabalho necessita de uma reengenharia para que passe a funcionar de maneira eficiente em memória distribuída.

1.2 Objetivo

Este trabalho tem como objetivo fazer uma atualização de uma parte importante para o funcionamento da ferramenta iSPD, sendo ela a gestão de cargas de trabalho, com o intuito de melhorar o seu desempenho e eficiência, executando de maneira distribuída.

Sendo assim a contribuição cientifica deste trabalho é a reengenharia da gestão de cargas de trabalho do iSPD para um novo modelo e a adição do módulo de carga de trabalho em traços [Silva 2012].

1.3 Metodologia

O projeto da reengenharia do gestor de cargas foi dividido nas seguintes etapas:

1. Revisão do que foi feito no iSPD sequencial;
2. Adaptação do modelo sequencial para o paralelo;
3. Testes;
4. Estudo dos resultados.

A primeira etapa é dedicada a uma revisão bibliográfica do que foi feito no iSPD sequencial, para o entendimento da estrutura do gestor de cargas de trabalho.

Na segunda etapa, será estudado como adaptar as estruturas sequenciais para o modelo distribuído e, em especial, ao protocolo de sincronização *Time Warp*, considerando o mecanismo de computação reversa do *framework* ROSS.

Na terceira etapa será realizada os testes para verificação se a carga de trabalho gerada está seguindo corretamente os modelos probabilísticos e o arquivo de traços.

Na quarta e última etapa, é fornecida uma análise sobre os resultados obtidos em relação à custo de armazenamento e desempenho computacional.

2 Trabalhos correlatos

Neste capítulo será mostrado trabalhos relacionados com o iSPD e com a reengenharia de carga, citando os principais simuladores de grades computacionais e modelos para geração de carga de trabalho em grades computacionais.

2.1 Simuladores

Esta seção trata dos simuladores de grades computacionais.

2.1.1 SimGrid

SimGrid é um simulador, criado em 1999, com o intuito de estudar algoritmos de escalonamento centralizados em plataformas computacionais distribuídas e heterogêneas [Casanova, Legrand e Quinson 2008]. Ele apresenta ferramentas para o estudo de sistemas alto desempenho, sistemas *peer-to-peer* e computação em nuvem.

A modelagem do SimGrid é através de um arquivo XML de nome `plataform.xml`, que configura os recursos da *grid* como máquinas e enlaces. Em outro arquivo XML de nome `deployment.xml` são modelos os processos que executarão em cada recurso especificado pelo arquivo `plataform.xml`. O simulador possui funções implementadas para ler e modelar a simulação através desses arquivos.

Um diferencial do SimGrid é que ele também permite uma configuração dinâmica da carga de trabalho descrita no arquivo XML através do uso de *traces*, adicionando maior verossimelhança à carga de trabalho simulada.

Sua principal limitação é que seu motor de simulação é completamente sequencial, o que afeta sua escalabilidade.

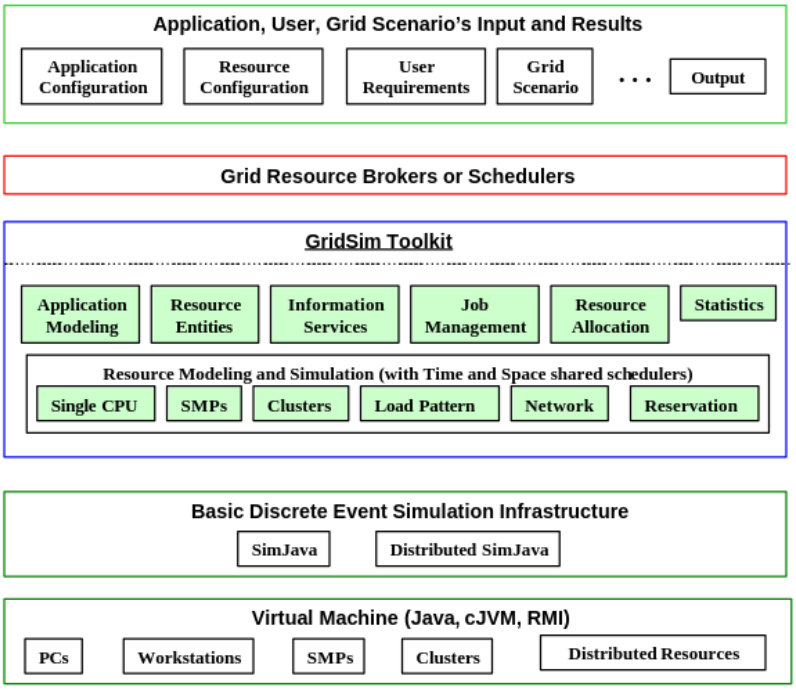
2.1.2 GridSim

GridSim é uma ferramenta para o estudo de algoritmos de escalonamento e configurações de *grids* para sistemas heterogêneos de larga escala [Buyya e Murshed 2002].

Sua arquitetura é em camadas, conforme ilustra a Figura 1, em que cada camada é construída através da camada abaixo. A primeira camada é a interface que conecta diretamente com o Java, implementando para processadores convencionais e multiprocessadores. A segunda camada apresenta uma infraestrutura de eventos discretos, sendo o SimJava um simulador de eventos discretos de propósito geral. A terceira camada apresenta as ferramentas do GridSim para modelagem da simulação de grades computacionais,

simulando através do simulador de eventos discretos de propósito geral apresentado pela camada abaixo. A quarta camada se preocupa com a simulação dos *resources brokers* e dos simuladores apresentados pelo GridSim. A quinta camada, por último, é a camada de aplicação e modelagem do GridSim, apresentado ao usuário final e construído através de todas as camadas abaixo.

Figura 1 – Arquitetura em camadas do GridSim



Fonte: [Buyya e Murshed 2002].

A criação de modelos é feita pela quinta camada da 'arquitetura, utilizando as classes e métodos fornecidos pelo GridSim. Portanto, é necessário o conhecimento da linguagem de programação e da documentação da ferramenta para a criação de modelos, não sendo amigável para leigos.

Além de ser usada para o estudo de algoritmos de escalonamento, o GridSim também permite analisar a influência de serviços de rede (como tráfego e roteamento) nas grades computacionais.

Diferente do SimGrid, o motor do GridSim apresenta paralelização da simulação, uma vez que cada componente pode ser executado por uma *thread*. Contudo, o uso de *threads* limita a escalabilidade devido à quantidade de memória necessária para gerenciar cada *thread*.

2.1.3 GangSim

GangSim é um simulador com o objetivo de estudar políticas de escalonamento em grades computacionais, em especial a relação entre escalonadores locais e escalonadores

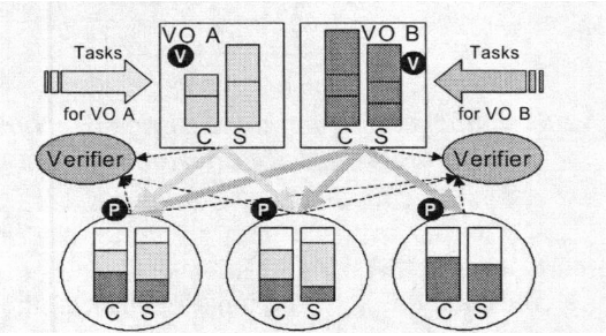
globais [GangSim 2012]. O simulador permite modelar usuários e Organizações Virtuais (VO) e suas capacidades para modelar políticas de uso; assim como estudar a escalabilidade e performance caso uma grade computacional decidir aumentar o número de sites e de usuários.

A arquitetura do GangSim é dividida em dois principais componentes:

- **Sites**: executarão as tarefas submetidas pelo usuário, possuindo escalonadores, número de CPUs, poder computacional de cada CPU, tamanho de espaço em disco e largura de banda dos canais.
- **Organizações Virtuais (VOs)**: usuários que submetem as tarefas e desempenham políticas de alocação de recursos. As tarefas de cada usuário são agrupadas em cargas de trabalho.

A Figura 2 descreve o funcionamento da alocação de recursos do GangSim. C e S identificam recurso de computação e recurso de armazenamento, respectivamente.

Figura 2 – Modelo de alocação de recursos no GangsSim



Fonte: [GangSim 2012]

A modelagem da plataforma e a especificação da carga de trabalho ocorre por meio de ferramentas específicas oferecias pelo GridSim, realizada através de *scripts*. Sendo assim, a modelagem não é muito amigável para usuários que não tem conhecimento da área.

2.1.4 Codes

Codes é um simulador construído sobre o *framework* ROSS, com finalidade de simular topologias de redes para sistemas de alta performance [Cope et al. 2011], em especial sistemas em nivel exa-escala. O simulador apresenta uma variedade de topologias de rede para simulação, entre elas *dragonfly*, *torus*, *fat-tree* e entre outras. Além da simulação de topologias de redes, Codes busca facilitar o estudo de sistemas de armazenamento em exa-escala.

Por ser construída sobre o *framework* ROSS, é possível a simulação paralela do seu motor. De fato, Codes é um simulador altamente escalável, suportando a simulação de modelos com até 2 milhões de eventos por segundo.

A desvantagem do Codes é a sua limitação de funcionalidades, não permitindo determinar políticas de escalonamento e também não permitindo uma construção flexível de topologias de rede, sendo limitado somente pelas topologias ofertadas pelo simulador. Apesar de permitir carga de traços de modelos reais, o simulador não simula as execuções das aplicações, simula apenas a execução das cargas de comunicação na topologia de rede.

2.1.5 Comparação entre os simuladores

A Tabela 1 apresenta uma comparação lado a lado entre os simuladores apresentados, incluindo o iSPD Java e o iSPD com a reestruturação realizada nesse trabalho.

Tabela 1 – Comparação entre os simuladores

	SimGrid	**GridSim**	**GangSim**	**Codes**	**iSPD**	**iSPD exa**
Linguagem de programação:	C	Java	Perl	C++	Java	C++
Modelagem do simulador:	Arquivos XML	Codificação direta	Scripts e interface web	Codificação direta	Interface Icônica e arquivos IML	Interface Icônica e arquivos JSON
Carga de trabalho:	Sintética	Sintética e traços	Sintética	Sintética e traços	Sintética e traços	Sintética e traços
Motor de simulação:	Sequencial	*Multithread*	Sequencial	Distribuído	Sequencial	Distribuído

Fonte: Elaborado pelo autor.

2.2 Geração de carga de trabalho

A modelagem da carga de trabalho é essencial para o estudo de um sistema a ser simulado. É comum ter conclusões errôneas sobre um sistema devido ao mau uso das cargas de trabalho [Jain 1991]. No geral, a carga de trabalho usada deve ser fiel à atividade do sistema no seu campo de atuação real, ou seja, fora da análise de performance ou da simulação [Lutteroth e Weber 2008].

Há dois tipos de cargas de trabalho em simulações: traços coletados de sistemas reais e cargas de trabalho sintética [Minh, Nam e Epema 2014]. A geração de carga de trabalho sintética é importante porque não é todas as vezes que traços de sistemas reais estão disponíveis ou ao menos existem [Galindo et al. 2009]. Para lidar com a falta de traços de sistemas reais, cargas de trabalho sintéticas são criadas a partir de um modelo apropriado [Iosup et al. 2007].

Uma carga de trabalho sintética consiste no uso de coleções de distribuições probabilísticas para descrever atributos da carga de trabalho, como tempo de execução, tamanho da tarefa, tempo de chegada e etc [Iosup et al. 2007]. Essa carga de trabalho sintética é representativa se a carga de trabalho simulada gera os mesmos resultados se submetida ao sistema real [Krishnamurthy, Rolia e Majumdar 2006].

No geral, a geração de carga de trabalho sintética tem as vantagens de serem reproduzíveis em larga escala, o que não ocorre com a carga baseada em traços justamente por estas serem inconsistentes e escassas.

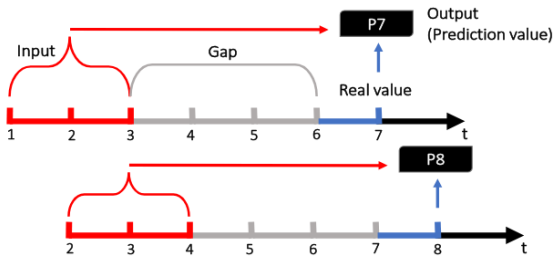
Entre os simuladores apresentados na seção anterior, todos possuem um gerador sintético de carga de trabalho. Contudo, o GridSim e o Codes também vem integrado com um gerador que utiliza traços de arquivos.

Para a geração sintética, é necessário utilizar bons modelos probabilísticos. A utilização de modelos clássicos como exponencial e uniforme muitas vezes não descrevem cargas reais, sendo necessário o uso de distribuições probabilísticas mais complexas como log-uniforme, hiper-exponencial e uniforme em dois estágios [Lo, Mache e Windisch 1998]. Contudo, a utilização das distribuições simples é essencial para testar o simulador sob carga conhecida e estável, como em distribuições uniformes.

No entanto, para tornar a geração sintética ainda mais realista, autores utilizam uma abordagem de criar a distribuição probabilística a partir dos traços, criando modelos que melhor se encaixam aos dados reais pelo uso do Método da Máxima Verossimilhança e outros artefatos estatísticos [Minh, Nam e Epema 2014].

Essa abordagem de gerar a carga sintética a partir dos traços se tornou ainda mais viável com o uso das Inteligências Artificiais. [Gao, Wang e Shen 2020] apresentam um modelo de geração de carga de trabalho a partir do treinamento de um modelo de Aprendizado de Máquina utilizando cargas reais classificadas de um sistema de Computação em Nuvem. Os autores avaliariam diversas estratégias baseadas em estatística e Aprendizado de Máquina utilizando o arquivo Google Cluster Trace, que contém cerca de 48 milhões de tarefas. Ao final, o trabalho melhora os resultados utilizando um algoritmo chamado de *m-gap Prediction*, extensão de um algoritmo de Aprendizado de Máquina, no qual os pontos temporais antes dos $n^th - m$ pontos são usados para predizer o valor da carga de trabalho no ponto n^th, conforme explicitado na Figura 3.

Figura 3 – Exemplo do algoritmo *m-gap prediction*



Fonte: [Gao, Wang e Shen 2020].

Ainda na técnica de Aprendizado de Máquina, [Wamba et al. 2017] utilizaram Redes Neurais para geração de cargas de trabalho utilizando traços para Computação em Nuvem. Os autores treinaram uma rede neural simples com 24 neurônios na camada de

entrada e uma camada oculta com 65 neurônios. Além da geração de carga de trabalho, o modelo também pode ser usado para realizar predições em tempo real de futuras cargas de trabalho que podem ser submetidas à rede de acordo com as que já foram submetidas.

Em suma, a modelagem da carga de trabalho é um aspecto essencial na simulação de sistemas, e a escolha entre traços reais e cargas sintéticas deve ser feita com cuidado. Enquanto as cargas sintéticas oferecem vantagens em termos de escalabilidade e reprodutibilidade, traços reais são fundamentais para garantir que a simulação seja verossímil.

Métodos que combinam cargas de trabalho reais e cargas de trabalho sintéticas oferecem vantagens significativas, como a reprodutibilidade e a escalabilidade, ao mesmo tempo em que preservam a verossimilhança da simulação. No entanto, a escassez de dados de traços reais de grades computacionais impacta negativamente a escalabilidade em cenários de exa-escala. Essa limitação de dados abrangentes compromete a geração de cargas em larga escala, especialmente quando são aplicados métodos de Aprendizado de Máquina, que dependem de grandes volumes de dados para um treinamento eficaz.

3 O Simulador iSPD

Neste capitulo será demonstrado em detalhes o simulador iSPD [Manacero et al. 2012], sua modelagem e a avaliação de grades computacionais, proporcionando um ambiente controlado para a experimentação e otimização de recursos distribuídos. Será ilustrado os seus componentes (Figura 4) e suas funcionalidades, dando enfoque para o componente de cargas de trabalho.

3.1 Visão Geral do iSPD

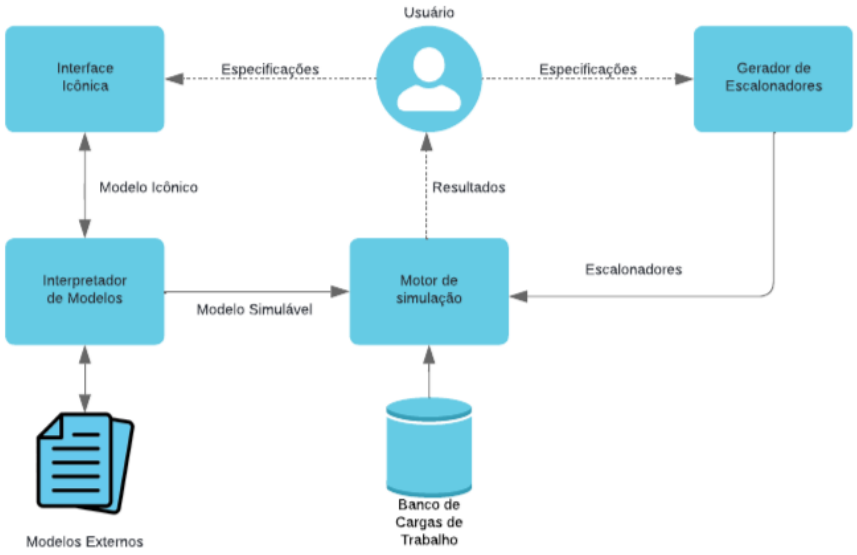
Como dito anteriormente, o iSPD (**i**conic **S**imulator of **P**arallel and **D**istributed systems) [Manacero et al. 2012] será o simulador utilizada para se fazer as simulações. Esse simulador tem como objetivo facilitar o processo de modelagem computacional utilizando-se de uma interface icônica. Atualmente, a sua versão disponibilizada a todos pelo site do GSPD [Homepage do GSPD] é feita em Java. É importante lembrar que este trabalho faz parte de um processo de atualização da ferramenta como um todo, recebendo mudanças até mesmo em sua linguagem base.

Perante a todos os pontos negativos dos simuladores descritos na seção anterior, o iSPD foi construído portando uma interface icônica, para fácil uso da ferramenta. Também, o iSPD tem como destaque importante o fato de que ele é bem versátil. Com a sua arquitetura que utiliza de duas linguagens para o processo de simulação [Menezes et al. 2012], se torna fácil de se desenvolver conversores de modelos vindos de outros simuladores, bem como converter do iSPD para outros, sendo assim um ponto extremamente positivo da ferramenta. Na Figura 4, podemos ver a arquitetura do iSPD.

A ferramenta é dividida majoritariamente entre os seguintes módulos:

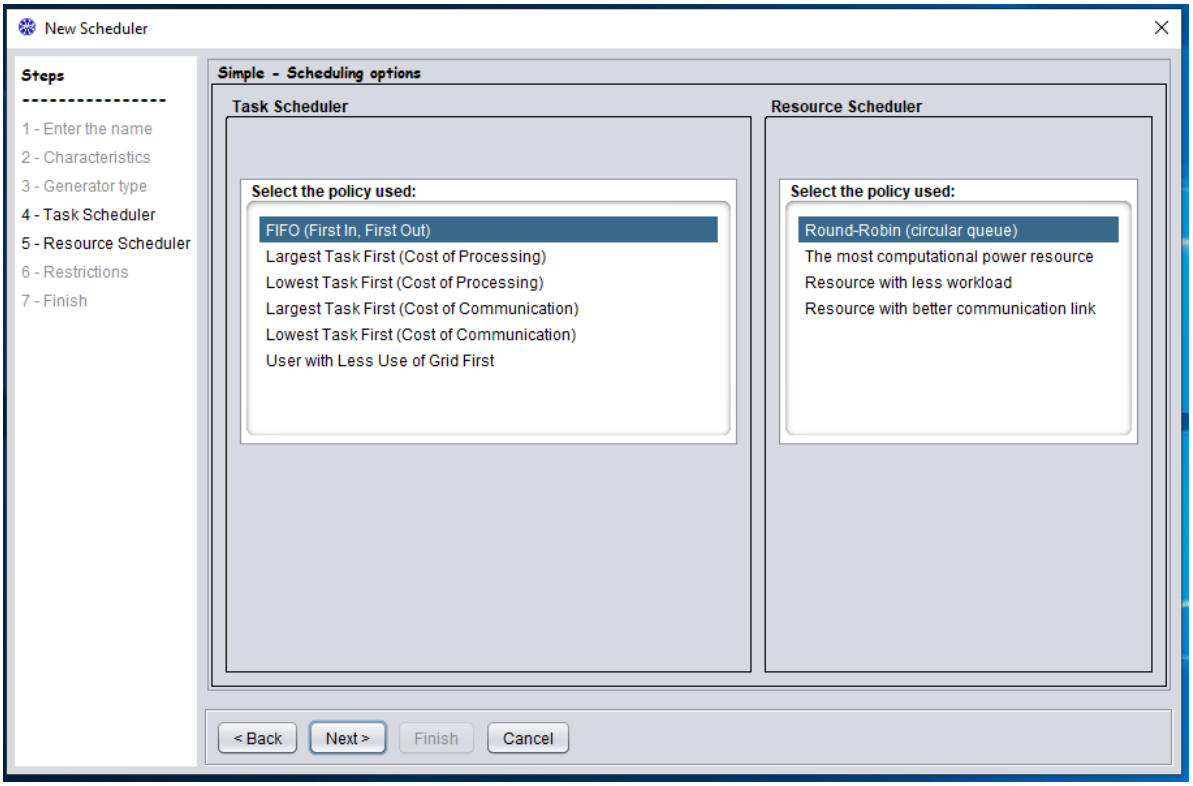
- **Interpretador de modelos internos e externos:** Esse módulo faz a interpretação do modelo icônico e o transforma em um modelo simulável, entregando-o para o motor. No caso de modelos externos ao iSPD, ele também faz a transformação deste modelo para a possível visualização na interface [Aoqui et al.].
- **Gerenciador e Gerador de Escalonadores:** Esses dois módulos simplificam tanto o gerenciamento dos escalonadores disponíveis quanto a criação de novas políticas de escalonamento. Juntos, esses sistemas não só administram os escalonadores para simulação, mas também facilitam a formulação de políticas escalonamento através de abordagens matemáticas simples e eficazes [Menezes et al. 2012]. A Figura 5 apresenta o Gerador de Escalonadores.

Figura 4 – Diagrama Estrutural



Fonte: Elaborado pelo autor.

Figura 5 – Gerador de Escalonadores

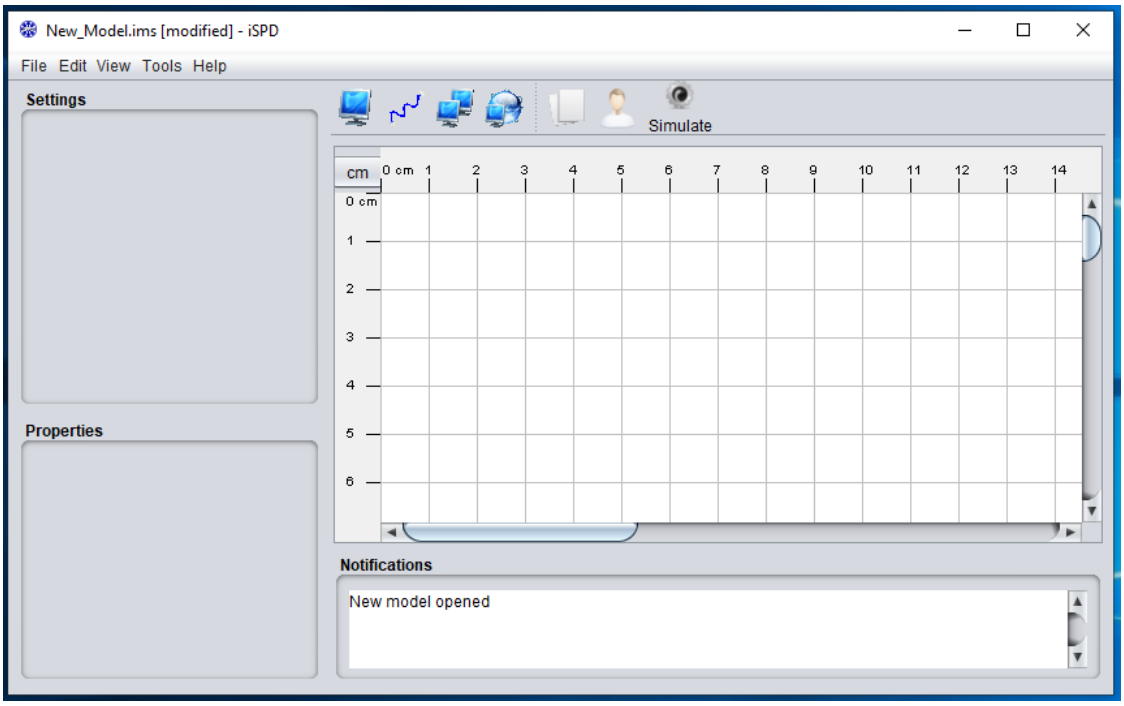


Fonte: Tela do iSPD Sequencial.

- **Motor de Simulação:** Esse módulo é o responsável pela simulação, executando após receber o modelo do interpretador. Após o final da simulação, ele também é o responsável por entregar os resultados ao usuário [Oliveira et al.].
- **Interface icônica:** Este módulo apresenta toda a iteração do usuário com a fer-

ramenta. É através desse módulo que é possível acessar todos os outros. É através da interface que podemos ver e manusear os nós do modelo icônico, que podem ser vistos como os pequenos ícones de computadores e setas [Guerra et al.]. Exemplo na Figura 6.

Figura 6 – Tela inicial do iSPD



Fonte: Tela do iSPD Sequencial.

3.2 Paralelização

A fim de melhor performance, o iSPD sofreu uma reengenharia para mudar de um sistema de memória compartilhada para um sistema de memória distribuída. Essa reengenharia foi necessária para que fosse possível executar a simulação em paralelo utilizando o protocolo *Time Warp* [Jefferson e Sowizral 1982], aumentando a escalabilidade do simulador. Além do modelo, foi trocada a linguagem de programação de Java para C++.

3.2.1 Time Warp

O *framework* utilizado para a execução paralela foi o ROSS [Carothers, Bauer e Pearce 2000], no qual foi construído utilizando os protocolos de sincronização *Time Warp* e *CMB*. Além do motor de execução, o *framework* fornece várias funções utilitárias para auxiliar na modelização do simulador, como funções probabilísticas e geradores de números aleatórios.

O *Time Warp* é um protocolo de sincronização que permite que eventos sejam revertidos caso ocorram violações de causalidade. Violações de causalidade ocorrem quando um evento que está sendo processado deveria ter sido processado no passado mas, por erros de sincronização, foi postergado. No *Time Warp*, essa violação é revertida e o estado retorna ao momento em que o evento pode ser executado corretamente.

O ROSS utiliza da técnica de Computação Reversa para restauração de estados dentro do *Time Warp*. Por conta disso, para cada procedimento é necessário escrever uma função para processamento direto e para processamento reverso, de modo que seja possível o cancelamento caso ocorra problemas de sincronização.

A escolha da linguagem C++ para a reengenharia foi, principalmente, pela integração nativa com a biblioteca MPI para execução distribuída da simulação. Além disso, o ROSS é feito com suporte a C, que pode ser facilmente adaptável para C++.

3.3 Considerações

Neste capitulo foi demonstrado o funcionamento geral do iSPD, ferramenta na qual o trabalho é feito. Foi apresentado a motivação de criação, bem como seu estado atual e suas funcionalidades, além de citar o projeto de paralelização do iSPD. No próximo capitulo será tratado especificamente de uma parte da arquitetura da ferramenta, o gestor de cargas de trabalho.

4 O Gestor de Cargas de Trabalho

Neste capítulo iremos aprofundar na funcionalidade especifica do iSPD que é foco deste projeto: o gestor de cargas de trabalho.

O gestor de carga de trabalho compreende algoritmos tanto para criação sintética de carga de trabalho, utilizando geradores, quanto à criação de carga de trabalho utilizando arquivos de traços fornecidos por sistemas reais.

4.1 Gerador de Carga de Trabalho

A geração de carga de trabalho é crucial no estudo de grades computacionais. A carga de trabalho é fundamental para estudar e prever diferentes configurações e condições de operação e como estas afetam o desempenho do sistema.

O gerador de carga de trabalho é utilizado para geração dos parâmetros das tarefas que serão executadas. As tarefas têm três parâmetros:

- **Tamanho de Processamento (MFLOP):** Custo computacional da tarefa;
- **Tamanho da Comunicação (MBITS):** Custo de comunicação da tarefa.
- **Offloading Computacional (%):** Porcentagem do custo computacional que será transferido para ser executado em GPU.

Dado esses parâmetros, foram projetados três geradores de valores:

4.1.1 Gerador Constante

Gerador no qual todas as tarefas terão os mesmos valores para os parâmetros. Sendo assim, para n tarefas, todas as n tarefas terão o mesmo tamanho de processamento, tamanho de comunicação e *offloading* computacional. Esse gerador constante é útil para teste de estabilidade e avaliar o sistema sob carga conhecida [Jain 1991].

4.1.2 Gerador Uniforme

Gerador no qual os parâmetros seguem distribuições uniformes independentes. Uma distribuição uniforme para dados contínuos, indo de um limite a até um limite b é dado por $f(x) = \frac{1}{b-a}$ Essa distribuição é usada para geração de dados aleatórios dentro de um certo limite.

Figura 7 – Processamento direto do gerador uniforme

```
1         proc_size = MinProcSize + tw_rand_unif(rng) * (MaxProcSize -
              MinProcSize);
2
3         comm_size = MinCommSize + tw_rand_unif(rng) * (MaxCommSize -
              MinCommSize);
```

Fonte: Elaborado pelo autor.

Figura 8 – Processamento reverso do gerador uniforme

```
1         tw_rand_reverse_unif(rng);
2         tw_rand_reverse_unif(rng);
```

Fonte: Elaborado pelo autor.

Os códigos para o processamento direto e o processamento inverso do Gerador Uniforme estão descritos nas Figuras 7 e 8, respectivamente.

O *framework* ROSS fornece as funções *tw_rand_unif* e *tw_rand_reverse_unif* para gerar números aleatórios usando a distribuição uniforme e para reverter essa geração, respectivamente. O número de vezes que a função para reverter a geração deve ser chamada deve ser equivalente ao número de vezes que a função de geração é chamada.

4.1.3 Gerador Uniforme Dois Estágios

Gerador cujos parâmetros seguem distribuições uniformes de dois estágios diferentes. Nesse gerador, é dado:

- l: limite inferior do primeiro intervalo
- m: limite superior do primeiro intervalo e limite inferior do segundo intervalo
- h: limite superior do segundo intervalo
- p: probabilidade de selecionar o primeiro intervalo

Primeiramente, é gerado um valor x a partir de uma distribuição uniforme. Se $x \leq p$, então o parâmetro recebe o valor de uma distribuição uniforme no intervalo (l, m). Caso contrário, o parâmetro recebe o valor de uma distribuição uniforme no intervalo (m, h).

Esse tipo de gerador tem maior flexibilidade e fornece cargas de trabalho sintéticas mais realistas [Law 2015].

Uma simplificação do código para processamento direto e processamento inverso do Gerador Uniforme de Dois Estágios estão descritos nas Figuras 9 e 10:

Figura 9 – Processamento direto do gerador uniforme em dois estágios

```
x = tw_rand_unif(rng)

if (x <= p)
{
    t = l +tw_rand_unif(rng) *(m-l);
}
else
{
    t = m + tw_rand_unif(rng) * (h-m);
}
```

Fonte: Elaborado pelo autor.

Figura 10 – Processamento reverso do gerador uniforme em dois estágios

```
tw_rand_reverse_unif(rng);
tw_rand_reverse_unif(rng);
tw_rand_reverse_unif(rng);
```

Fonte: Elaborado pelo autor.

4.2 Gerador do tempo de chegada

Esse gerador serve para modelar o tempo de chegada entre tarefas, nos quais o intervalo entre uma tarefa e outra é também baseada em modelos probabilísticos. [Jain 1991] e [Law 2015] definem modelos probabilísticos usados para geração de intervalos entre tarefas em simuladores. Entre estes, os que foram implementados para o iSPD são:

4.2.1 Distribuição constante

O tempo de chegada entre tarefas é fixo para todas as tarefas do modelo a ser simulado. Esse tempo de chegada é definido pelo usuário.

4.2.2 Distribuição exponencial

A distribuição exponencial é a única distribuição contínua que tem a propriedade de que o tempo de chegada do evento anterior não ajuda a predizer o tempo de chegada do próximo evento [Jain 1991]. É largamente usada para simular tempos entre chegadas.

O ROSS fornece funções de geração de números utilizando distribuição exponencial *tw_rand_exponential(tw_rng_stream *g, double Lambda)* e também para números gerados por essas distribuições.

4.2.3 Distribuição de Weibull

A distribuição de Weibull tem a vantagem de conseguir mais acuradamente representar variações na taxa de chegada ao longo do tempo, podendo representar tempos de chegadas que diminuem com o tempo e tempos de chegadas que aumentam com o tempo, dependendo dos parâmetros passados [Law 2015].

O ROSS também fornece uma função para a distribuição de Weibull, *tw_rand_weibull(rng, m_Mean, m_Shape);*

4.2.4 Distribuição de Poisson

A distribuição de Poisson é usada para retornar o número de tarefas chegadas em um dado instante dado sua taxa média de chegada, sendo fornecido pelo ROSS na função *tw_rand_poisson(tw_rng_stream *g, double Lambda)*

4.3 Carga de trabalho baseada em traços

Além da caracterização da carga de trabalho utilizando os geradores, também há a possibilidade da caracterização utilizando traços. Traços consistem nos *logs* da execução de um sistema real, armazenado em um arquivo.

A modelagem da carga de trabalho utilizando traços de um sistema real permite maior realismo na geração de carga do simulador, permitindo a representação de ocasiões e situações que não podem ser sinteticamente geradas.

Há dois principais modelos de carga de traços para grades computacionais: o SWF (*Standard Workload Format* e GWF*(Grid Workload Format)* [Silva 2012]. Nesses modelos, é possível extrair, entre outras características:

- ID da tarefa;
- Status da tarefa;
- Tempo de execução;
- Proprietário da tarefa;
- Tempo de submissão

Dessas características, apenas o ID da tarefa, tempo de execução e tempo de submissão serão utilizados. Para trabalhos futuros, é necessário procurar modelos de traços que fornecem características da execução em GPU.

O trabalho foca na utilização da conversão do modelo SWF, que é o modelo de traços mais utilizado.

4.3.1 Conversão SWF

O primeiro passo é, tendo o arquivo de traços, converter para um modelo simulável dentro do iSPD.

Os dados do arquivo são representados dentro do simulador a partir da seguinte estrutura:

Figura 11 – Estrutura para armazenar dados gerados através dos traços

```
struct TraceWorkload
{
    int id;
    double procSize;
    double commSize;
    double arrival;
}

struct TraceWorkloadQueue
{
    int num_tasks;

    std::vector<struct TraceWorkload> queue;

    int remaining_tasks;
}
```

Fonte: Elaborado pelo autor.

Assim, utiliza-se um vetor que armazena as informações das tarefas. Cada tarefa é executada da posição 0 até a posição $num\_tasks - 1$, ordenados pelo tempo de criação conforme descrito no arquivo original SWF.

O tempo de processamento de cada tarefa, apesar de não ser oferecido no arquivo original, pode ser calculado através da média da capacidade de computação do modelo multiplicado pelo tempo de execução de cada tarefa, obtendo uma estimativa do tamanho computacional de cada tarefa. [Silva 2012].

O tempo de comunicação, por sua vez, não é oferecido no arquivo de traços e nem apresenta formas de calculá-lo por não oferecer informações sobre consumo de rede. Portanto, esse valor é gerado por um dos geradores sintéticos descritos na seção 3.1.

4.3.2 Estrutura de dados dos traços

A principal diferença entre o gestor de carga de trabalho com traços e o gestor de carga de trabalho com geradores é a necessidade de uma estrutura de dados para armazenamento das tarefas.

No uso de traços, essa estrutura é necessária pois não é possível gerar, somente no momento de execução, os parâmetros das tarefas dos traços, uma vez que elas não seguem

distribuições bem definidas. Já no uso de geradores, é possível gerar as tarefas apenas no momento de execução das mesmas, uma vez que utilizam de distribuições probabilísticas.

O uso da estrutura de dados para armazenamento de tarefas implica em um maior consumo de memória pelo simulador, o que pode afetar a escalabilidade do uso de traços para geração de carga de trabalho.

4.4 Considerações

Neste capítulo foi explicada a implementação do gestor de cargas de trabalho do iSPD após a reengenharia e com funcionamento de memória distribuída. Foi apresentado os geradores de carga de trabalho sintéticos e suas distribuições de probabilidade, com seus respectivos processamentos diretos e reversos. Foi também apresentado a implementação do sistema de arquivos de traços no iSPD, explicitando o conversor dos arquivos nos formatos respectivos para uma estrutura de dados para armazenamento dessas tarefas.

5 Resultados

Esse capítulo mostrará os resultados obtidos com esse trabalho.

5.1 Gerador de cargas

Conforme descrito na seção 3.1, a geração de carga de trabalhos é realizada a partir de um gerador de números aleatórios seguindo distribuições probabilísticas e limitantes definidas pelo usuário.

Para os testes, foram geradas 100.000 tarefas com tamanho de processamento variando de 100 até 10.000 MFLOPS. A geração do tamanho de comunicação e do *offload* computacional é feita da mesma forma. Esses valores representam uma abrangência desde tarefas simples até tarefas mais computacionalmente complexas. A amostra de 100.000 tarefas permite avaliar o gerador em um conjunto abrangente de dados e determinar se o gerador consegue manter as características estatísticas desejadas.

Para o cálculo da distância quadrática, é utilizada a Raiz do Erro Quadrático Médio (RMSE), pois essa métrica é amplamente utilizada para avaliar a diferença entre valores previstos e valores observados, penalizando fortemente os erros maiores. O RMSE é obtido através da equação:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

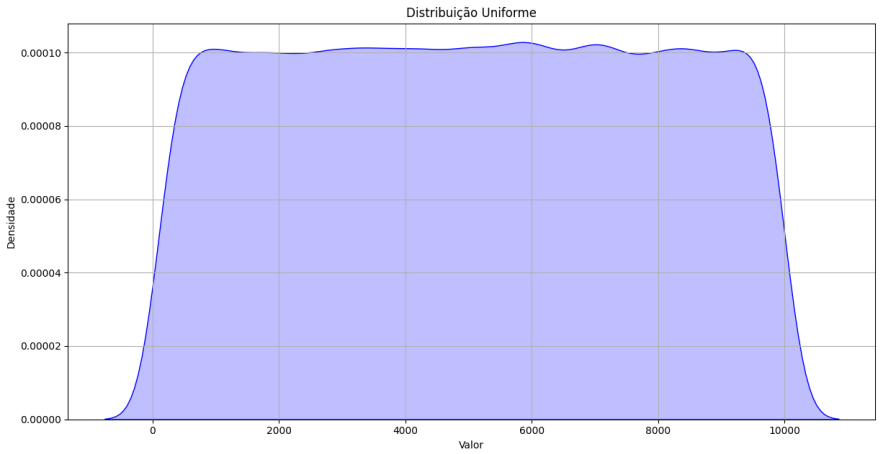
Na Figura 12 são apresentados os resultados obtidos com a geração de carga de trabalho uniforme.

O gráfico da distribuição uniforme apresenta o formato de uma distribuição uniforme, com a média dos dados sendo $5056,45$. A média teórica para a distribuição uniforme de 100 até 10.000 MFLOPS é de 5050. A distância quadrática entre a distribuição uniforme gerada e a distribuição uniforme teórica foi calculada em $70,01$. Esse valor pode ser considerado baixo, uma vez que a amostra dos dados é ampla, indicando que a distribuição gerada pelo gestor de carga de trabalho se aproxima de uma distribuição uniforme.

Os mesmos dados foram testados para a distribuição uniforme em dois estágios. A Figura 13 apresenta o gráfico do teste realizado.

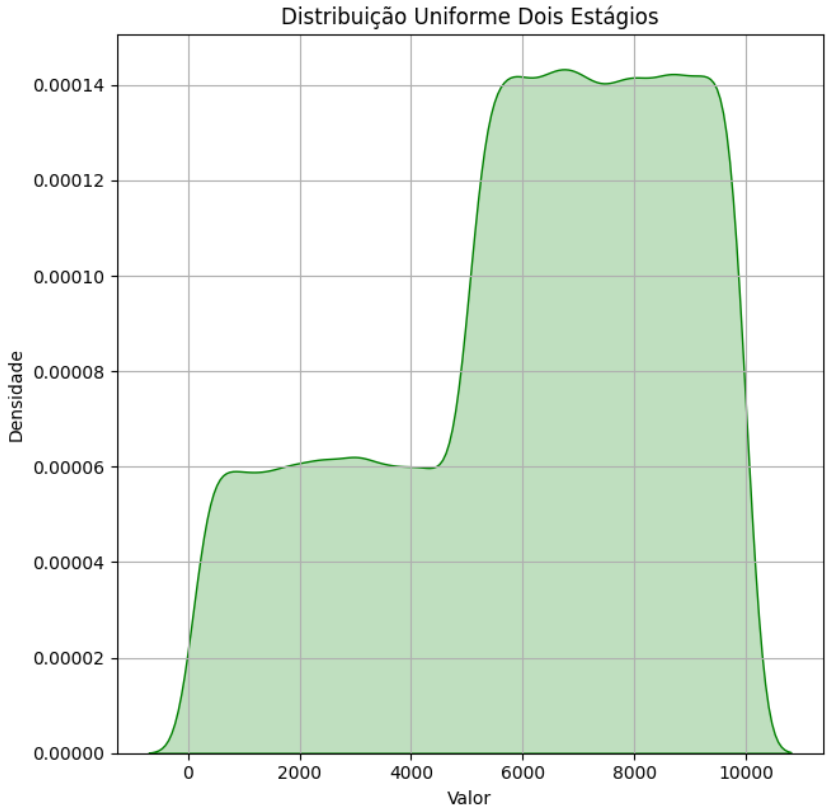
Sobre a distribuição uniforme em dois estágios, o valor intermediário foi 5050, que é exatamente o meio entre os limites, e a probabilidade de escolher o intervalo do primeiro estágio foi de $0,3$. A média teórica é a média ponderada entre o primeiro estágio e o segundo estágio, sendo:

Figura 12 – Geração de carga utilizando a distribuição uniforme com 100.000 tarefas variando de 100 até 10.000 MFLOPS



Fonte: Elaborado pelo autor.

Figura 13 – Geração de carga utilizando a distribuição uniforme de dois estágios com 100.000 tarefas variando de 100 até 10.000 MFLOPS



Fonte: Elaborado pelo autor.

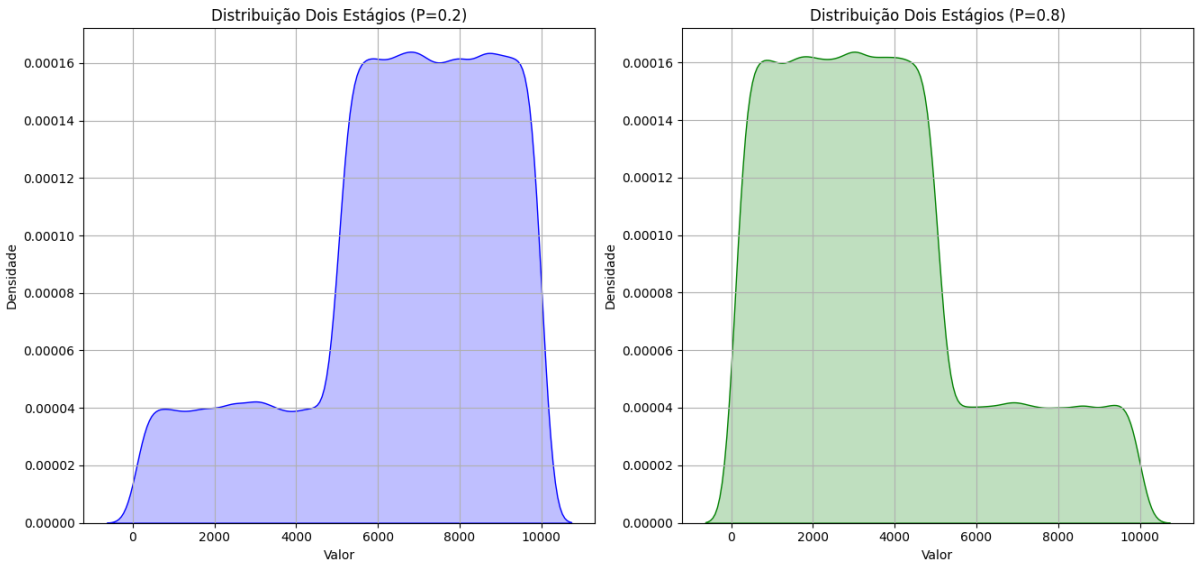
$$\mu_{\text{teórica}} = \left(0,3 \cdot \frac{100 + 5050}{2}\right) + \left(0,7 \cdot \frac{5050 + 10.000}{2}\right)$$

Assim, o valor da média teórica é de 6040, enquanto a média real dos dados foi de

6053, 82. A distância quadrática calculada foi de 96, 56, o que pode ser considerado baixo, devido à grande amostra de tarefas.

Para a distribuição uniforme em dois estágios, foram realizadas mais dois testes: um avaliando uma repartição mais à esquerda e outra avaliando uma repartição mais à direita, com as probabilidades do primeiro estágio sendo de 0, 8 e 0, 2 respectivamente. A Figura 14 apresenta os gráficos obtidos.

Figura 14 – Geração de carga utilizando a distribuição uniforme de dois estágios com 100.000 tarefas variando de 100 até 10.000 MFLOPS, com probabilidade do primeiro estágio sendo de 0,2 e de 0,8



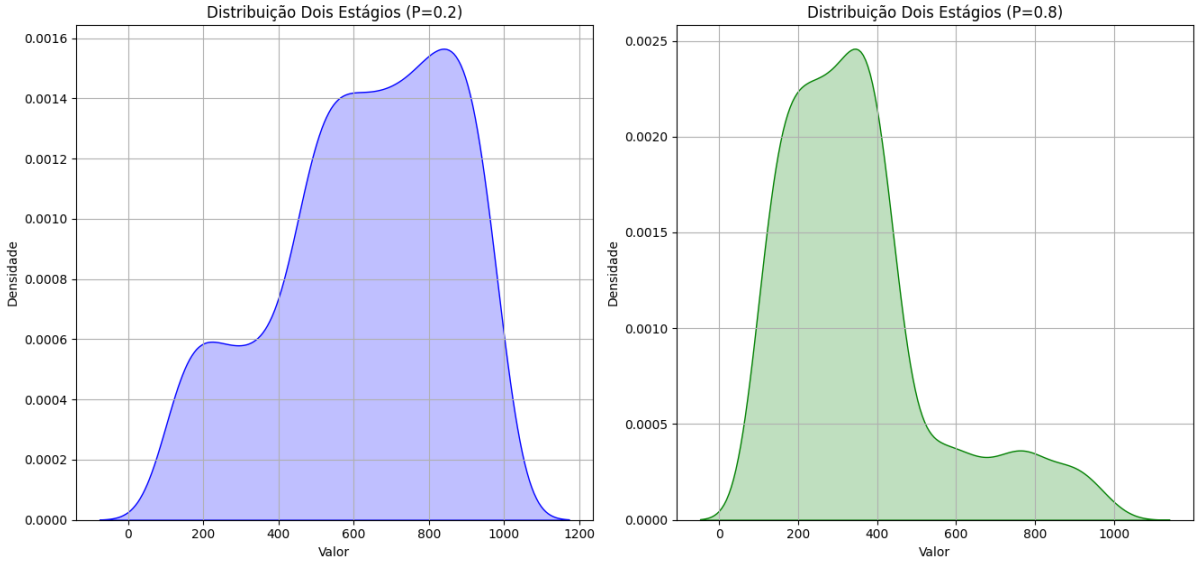
Fonte: Elaborado pelo autor.

Para a probabilidade do primeiro estágio sendo 0, 2, a média obtida pelo gerador foi de 6546, 531, sendo a média teórica de 6535, com distância quadrática 87, 86. Já para a probabilidade do primeiro estágio sendo de 0, 8, a média obtida pelo gerador foi de 3572, 47, sendo a média teórica de 3565, com distância quadrática 68, 98. Para ambos os casos, a distância quadrática entre os dados esperados e os dados obtidos é pequena em comparação a grande amostra de tarefas.

A fim de avaliar a geração de modelos menores, foi realizado um teste em um modelo de 1.000 tarefas variando de 100 a 1.000 MFLOPS. O gráfico da Figura 15 ilustra os resultados.

Para a probabilidade do primeiro estágio sendo 0, 2, a média obtida pelo gerador foi de 657, 922, sendo a média teórica de 662, 5, com a distância quadrática de 21, 2. Já para a probabilidade do primeiro estágio sendo de 0, 8, a média obtida pelo gerador foi de 381, 86, sendo a média teórica de 392, 5, e a distância quadrática obtida foi de 27, 99. A distância quadrática diminuiu de acordo com a menor amostra de tarefas e o menor intervalo de processamento, representando que o gerador também é utilizável em escala

Figura 15 – Geração de carga utilizando a distribuição uniforme e dois estágios com 1.000 tarefas variando de 100 até 1.000 MFLOPS, com probabilidade do primeiro estágio sendo de $0,2$ e de $0,8$



Fonte: Elaborado pelo autor.

menores.

Por conta do número de amostras ser menor, o formato do gráfico não é muito bem definido. Contudo, conforme demonstrado nos testes anteriores, o aumento do número de amostras mantém e reforça a característica estatística de se comportar como uma distribuição uniforme em dois estágios.

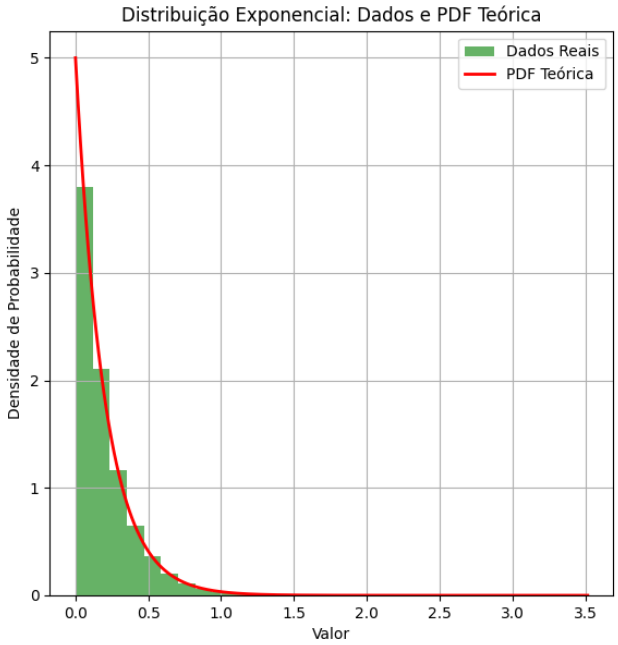
5.2 Gerador de tempo de chegada

Para a geração do tempo de chegada, foram utilizadas as distribuições probabilísticas descritas na seção 3.2. Os critérios estatísticos para avaliação serão os mesmos da sessão anterior, porém também são realizadas testes de aderência a Kolmogorov-Smirnov para distribuições contínuas e chi-quadrado para distribuições discretas. Os valores dos parâmetros foram escolhidos considerando um intervalo de valores baixo, gerando intervalos pequenos entre a chegada das tarefas, assim como ocorre em grades computacionais.

A Figura 16 foram apresentados os resultados dos tempos de chegadas para 100.000 tarefas, utilizando o parâmetro $\lambda = 0,2$ (considerando média dos dados 5) para a distribuição exponencial. A média teórica da distribuição exponencial é de $0,2$, e a obtida através dos dados foi de $0,19952$. A distância quadrática obtida foi de 32.44, o que sugere grande diferença entre a distribuição observada e a esperada em certo ponto. Para complementar, foi realizado o teste de aderência Kolmogorov-Smirnov (KS), considera todas as diferenças acumuladas entre as distribuições; obtendo o valor-p de 0.68. Sendo assim, apesar do

RMSA alto, não há evidências o suficiente para rejeitar a hipótese nula, ou seja, os dados observados pelo gerador são compatíveis com uma distribuição exponencial.

Figura 16 – Geração do tempo de chegada: distribuição exponencial

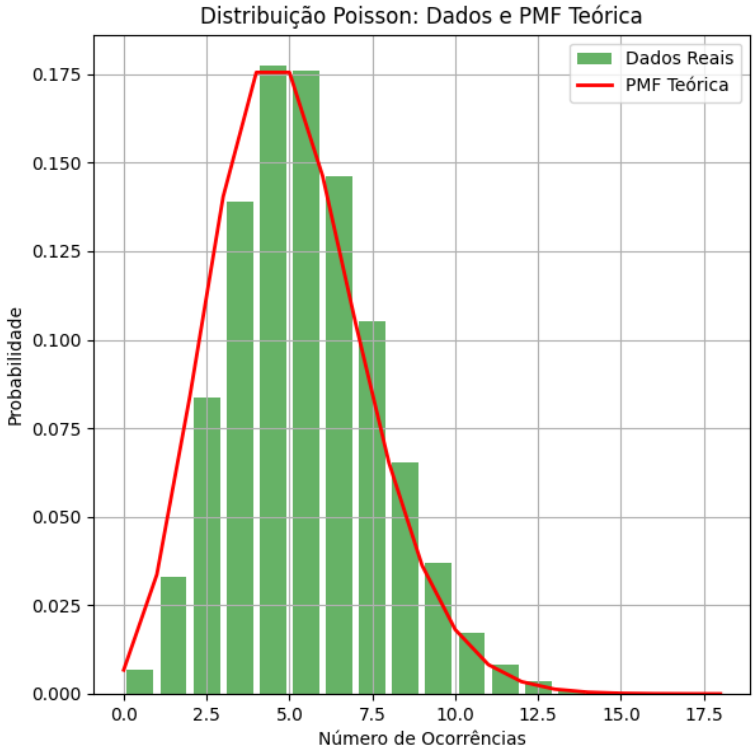


Fonte: Elaborado pelo autor.

A Figura 17 apresenta os resultados dos tempos de chegadas para 100.000 tarefas utilizando o parâmetro $\lambda = 5$. A média teórica da distribuição de Poisson com esse parâmetro é de 5, e a obtida com os dados gerados é a de $5,0014$. A distância quadrática medida é de 67.42, o que é alta . A fim de aprofundar a análise, foi realizado o teste de aderência chi-quadrado, para distribuições discretas, que resultou em um p-valor de 0.15, indicando que os dados estão em aderência com a distribuição de Poisson. Essa discrepância entre o RMSE e o chi-quadrado reflete que, em algumas categorias, a discrepância entre a frequência observada e a frequência esperada é grande, mas ainda sim essas diferenças não são suficientes para invalidar a hipótese de que os dados seguem uma distribuição de Poisson.

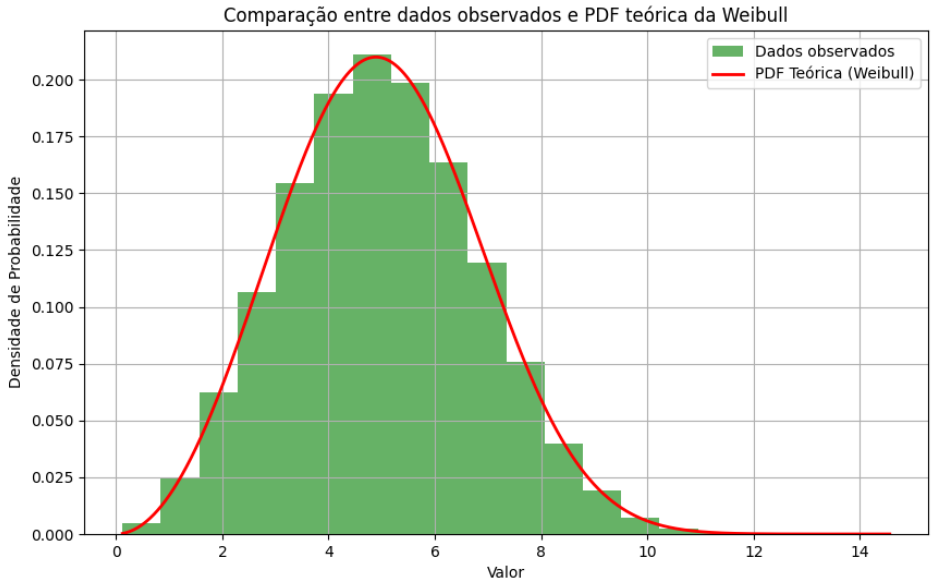
Na Figura 18 são apresentados os resultados dos tempos de chegadas para 100.000 tarefas utilizando o parâmetro $\lambda = 5$ e $k = 3$. A média teórica da distribuição de Weibull com esses parâmetros é de 5, e a obtida com os dados gerados é a de $4,997$. A distância quadrática obtida foi de $61,7$. Novamente, apesar da distância quadrática estar elevada, foi realizado o teste de aderência a Kolmogorov-Smirnov (KS) e o valor-p foi de $0,682$, reforçando que os dados observados são provenientes de uma distribuição de Weibull.

Figura 17 – Geração do tempo de chegada: distribuição de Poisson



Fonte: Elaborado pelo autor.

Figura 18 – Geração do tempo de chegada: distribuição de Weibull



Fonte: Elaborado pelo autor.

5.3 Conversor SWF

Os testes para o conversor SWF foram realizados utilizando os *logs* disponibilizados pela Universidade Hebraica de Jerusalém [1]. Foram empregados cinco arquivos distintos, cada um contendo configurações e quantidades de tarefas variadas, conforme apresenta a Tabela 2.

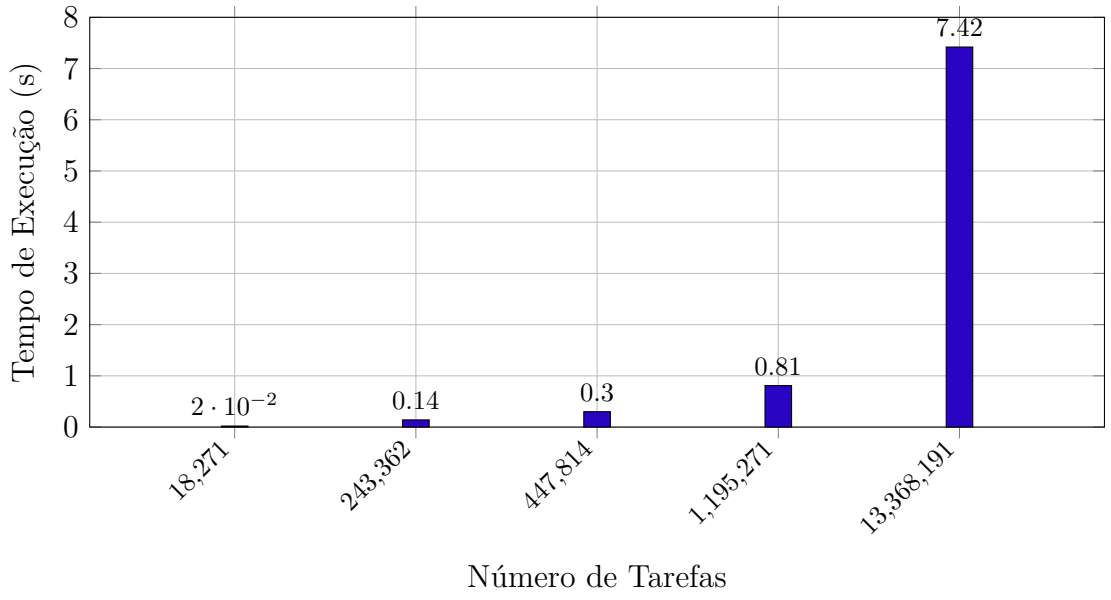
Tabela 2 – Tabela de arquivos SWF e quantidade de tarefas

Nome do Arquivo	**Quantidade de Tarefas**
NASA.swf	18,239
SDSC.swf	243,306
RICC.swf	447,814
SHARCNET.swf	1,195,242
IntelNetbatch.swf	13,368,191

Fonte: Elaborado pelo autor.

A conversão do arquivo de traços é realizada antes da simulação e, portanto, não afeta a execução da mesma. Contudo, conforme ilustrado na Figura 19, a conversão de arquivos de traços com informações de muitas tarefas adiciona um intervalo a mais no tempo de carregar o modelo simulado nas estruturas de dados, adicionando um tempo de carregamento para começar a simular, a partir do momento em que o usuário submete o modelo.

Figura 19 – Gráfico número de tarefas e tempo de execução



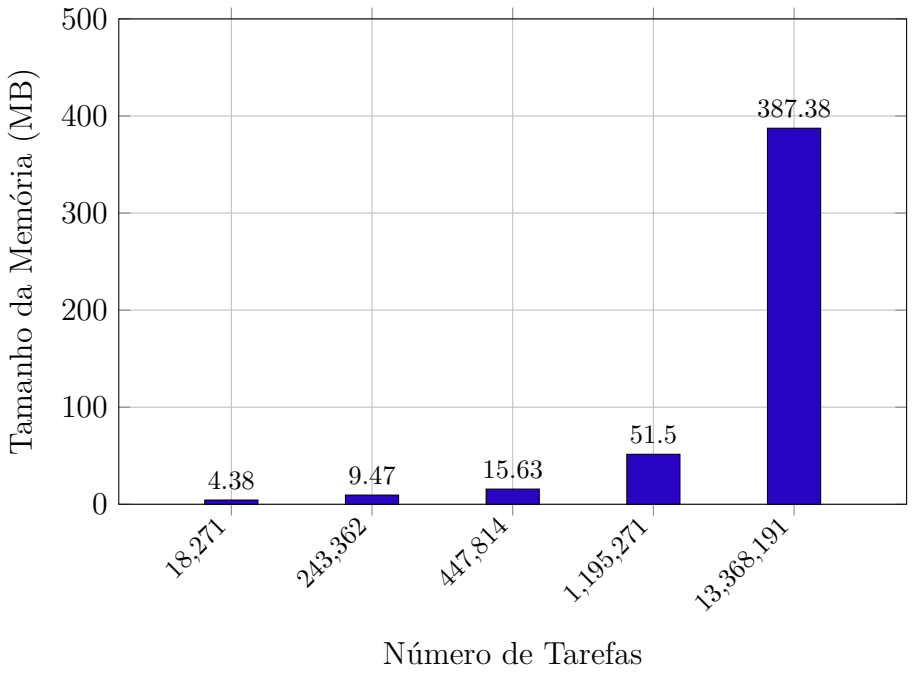
Fonte: Elaborado pelo autor.

Além do aumento no tempo de execução, há um aumento no uso da memória, por conta da estrutura de dados que armazena as informações das tarefas do arquivo no

[1] https://www.cs.huji.ac.il/labs/parallel/workload/logs.html

simulador, conforme explicado na seção anterior. Para sistemas de grande escala, com mais de 100 milhões de tarefas, a estrutura de dados para que armazena os traços ocupa a escala de Gigabytes de memória; se tornando inadequado a execução em computadores simples. A Figura 20 apresenta o consumo de memória dos arquivos utilizados para o teste.

Figura 20 – Gráfico número de tarefas e consumo de memória (em MB)



Fonte: Elaborado pelo autor.

5.4 Considerações

Neste capítulo foi apresentado os resultados obtidos por esse trabalho, primeiro apresentando o resultado do conversor de arquivos SWF para uma estrutura de dados dentro do motor e discutindo sua performance; e depois validando os geradores dos parâmetros das tarefas com as distribuições de probabilidade.

6 Conclusões e Trabalhos Futuros

Neste trabalho foi proposta e implementada a reengenharia do Gestor de Tarefas para o simulador de grades computacionais iSPD. Primeiro, foi feita a reengenharia dos geradores de carga de trabalho e geradores de tempo de chegada, de forma que seja possível a geração de tarefas utilizando as principais distribuições probabilísticas usadas em simulação [Law 2015]. Após, foi realizado a construção da simulação por arquivos de traços convertendo do modelo SWF para uma estrutura de dados que armazenasse as informações das tarefas.

Assim, o trabalho realizado enriqueceu o gestor de carga de trabalho do iSPD e expandiu seu domínio para simulação utilizando cargas de trabalhos através de traços de sistemas reais.

Em continuação desse trabalho, segue as seguintes sugestões para trabalhos futuros:

- **Traces de sistemas de grande porte:** Implementar suporte para arquivos de traços que contenham informações detalhadas sobre GPU e que representem sistemas em nível de exa-escala, permitindo a simulação de ambientes computacionais ainda mais complexos.
- **Geração de arquivos de traços utilizando Inteligência Artificial Generativa:** Dado que muitos arquivos de traços de sistemas reais são difíceis de obter ou são confidenciais, o desenvolvimento de algoritmos de Aprendizado de Máquina Não Supervisionado para a criação de arquivos de traços realistas poderia beneficiar a comunidade, oferecendo uma alternativa para estudos e simulações.
- **Estrutura de dados para otimização de espaço:** A estrutura de dados utilizada para armazenamento das tarefas não é aplicável em largas escalas, uma vez que o consumo de memória cresce em grande proporção conforme aumenta o número de tarefas. Sugere-se o uso de uma estrutura de dados mais avançada, como uma Árvore B+ para armazenamento das tarefas.

Referências

AOQUI, V. et al. Interpretador de modelos externos para simulador de grades computacionais. *I Escola Regional de Alto Desempenho de São Paulo*, v. 400, p. 1–2. Citado na página 22.

BUYYA, R.; MURSHED, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, Wiley Online Library, v. 14, n. 13-15, p. 1175–1220, 2002. Citado 3 vezes nas páginas 14, 16 e 17.

CAROTHERS, C.; BAUER, D.; PEARCE, S. Ross: a high-performance, low memory, modular time warp system. In: *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation*. [S.l.: s.n.], 2000. p. 53–60. Citado na página 24.

CASANOVA, H.; LEGRAND, A.; QUINSON, M. Simgrid: a toolkit for the simulation of application scheduling. In: IEEE PRESS. *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. [S.l.], 2008. p. 1–12. Citado 2 vezes nas páginas 14 e 16.

COPE, J. et al. Codes: Enabling co-design of multilayer exascale storage architectures. In: ACM. *Proceedings of the Workshop on Emerging Supercomputing Technologies*. [S.l.], 2011. v. 2011. Citado na página 18.

GALINDO, H. E. S. et al. Synthetic workload generation for capacity planning of virtual server environments. In: *2009 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2009. p. 2837–2842. Citado na página 19.

GANGSIM. *GangSim*. 2012. Disponível em: <[invalidURLremoved]>. Citado 2 vezes nas páginas 14 e 18.

GAO, J.; WANG, H.; SHEN, H. Machine learning based workload prediction in cloud computing. In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. [S.l.: s.n.], 2020. p. 1–9. Citado na página 20.

GSPD. *GSPD's homepage*. 2012. <http://www.dcce.ibilce.unesp.br/spd/>. Disponível em http://www.dcce.ibilce.unesp.br/spd/. Citado na página 14.

GUERRA, A. I. et al. Plataforma de simulação de grades computacionais: Interface icônica. *I Escola Regional de Alto Desempenho de São Paulo*, v. 400, p. 1–2. Citado na página 24.

HOMEPAGE do GSPD. <https://www.dcce.ibilce.unesp.br/spd/internos.php>. Accessed: 2024-05-26. Citado na página 22.

IOSUP, A. et al. On grid performance evaluation using synthetic workloads. In: FRACHTENBERG, E.; SCHWIEGELSHOHN, U. (Ed.). *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 232–255. ISBN 978-3-540-71035-6. Citado na página 19.

JAIN, R. *The art of computer systems performance analysis : techniques for experimental design, measurement, simulation, and modeling*. New York: J. Wiley, 1991. ISBN 0471503363. Citado 3 vezes nas páginas 19, 26 e 28.

JEFFERSON, D.; SOWIZRAL, H. A. *Fast Concurrent Simulation Using the Time Warp Mechanism: Part I, Local Control*. Santa Monica, CA: RAND Corporation, 1982. Citado na página 24.

KRISHNAMURTHY, D.; ROLIA, J. A.; MAJUMDAR, S. A synthetic workload generation technique for stress testing session-based systems. *IEEE Transactions on Software Engineering*, v. 32, n. 11, p. 868–882, 2006. Citado na página 19.

LAW, A. M. *Simulation Modeling & Analysis*. 5. ed. New York, NY, USA: McGraw-Hill, 2015. Citado 4 vezes nas páginas 27, 28, 29 e 40.

LO, V.; MACHE, J.; WINDISCH, K. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. p. 25–46. ISBN 978-3-540-68536-4. Citado na página 20.

LUTTEROTH, C.; WEBER, G. Modeling a realistic workload for performance testing. In: *2008 12th International IEEE Enterprise Distributed Object Computing Conference*. [S.l.: s.n.], 2008. p. 149–158. Citado na página 19.

MANACERO, A. et al. Ispd: An iconic-based modeling simulator for distributed grids. In: *Proceedings of the 45th Annual Simulation Symposium*. San Diego, CA, USA: Society for Computer Simulation International, 2012. (ANSS '12). Citado 2 vezes nas páginas 14 e 22.

MENEZES, D. et al. Scheduler simulation using ispd, an iconic-based computer grid simulator. In: *2012 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.: s.n.], 2012. p. 000637–000642. Citado na página 22.

MINH, T. N.; NAM, T.; EPEMA, D. H. J. Parallel workload modeling with realistic characteristics. *IEEE Transactions on Parallel and Distributed Systems*, v. 25, p. 2138–2148, 2014. Disponível em: <https://api.semanticscholar.org/CorpusID:15727158>. Citado 2 vezes nas páginas 19 e 20.

OLIVEIRA, P. et al. O motor de uma plataforma de simulação de grades computacionais. *I Escola Regional de Alto Desempenho de São Paulo*, v. 400, p. 1–2. Citado na página 23.

OPTORSIM. *OptorSim*. 2012. Disponível em: <http://optorsim.sourceforge.net/>. Citado na página 14.

SCHMIDT, B.; HILDEBRANDT, A. Next-generation sequencing: big data meets high performance computing. *Drug Discovery Today*, v. 22, n. 4, p. 712–717, 2017. ISSN 1359-6446. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1359644617300582>. Citado na página 14.

SILVA, D. T. da. *Implementação de um Banco de Traces de Cargas de Trabalho para o iSPD*. 2012. Citado 3 vezes nas páginas 15, 29 e 30.

TAKEFUSA, A.; MATSUOKA, S. Bricks: A simulator for shared-memory multiprocessors and its application to evaluation of programming models. In: IEEE. *Proceedings. 13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing (Cat. No. 01EX470)*. [S.l.], 2000. p. 283–290. Citado na página 14.

WAMBA, G. M. et al. Cloud workload prediction and generation models. In: *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. [S.l.: s.n.], 2017. p. 89–96. Citado na página 20.

WANG, L.; JIE, W.; CHEN, J. *Grid computing: infrastructure, service, and applications*. [S.l.]: CRC Press, 2018. Citado na página 14.