UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Wenderson de Souza Leonardo

Um Ambiente para Modelagem Integrada de Energia, Custo e Disponibilidade de Data Centers

Recife

2021

Wenderson de Souza Leonardo

Um Ambiente para Modelagem Integrada de Energia, Custo e Disponibilidade de Data Centers

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Gustavo Rau de Almeida Callou

Recife

2021

Dedico este trabalho a minha família.

Agradecimentos

Agradeço ao meu orientador Gustavo Rau de Almeida Callou, pela oportunidade de me juntar a ele em seus projetos através do Programa de Iniciação Cientifica. Bem como a meus colegas de curso, Adailson José, Caroline Gomes, Eliana França, Jadeilson Rocha e Leandro Cassio, que sempre estiveram lá para me ajudar quando precisei.

“A persistência é o caminho do êxito.”
(Charles Chaplin)

Resumo

Sustentabilidade tem recebido atenção crescente da comunidade científica, sendo o maior foco o da redução do consumo energético e também na manutenção de recursos não renováveis para as futuras gerações. Em paralelo, a expansão de paradigmas como o da computação nas nuvens, redes sociais e comércio eletrônico acabou por aumentar a demanda dos *data centers*. Nesse contexto, ferramentas que dão suporte a modelagem de arquiteturas de *data center* e que sejam capazes de computar métricas como a de disponibilidade, custo e consumo energético são de extrema importância. Esse projeto propõe o desenvolvimento de uma ferramenta com uma visão de alto nível para a modelagem de arquiteturas de *data centers* com a finalidade de se computar o consumo energético, disponibilidade e custo. No desenvolvimento dessa ferramenta, uma biblioteca de grafos está sendo utilizada para representar os componentes dessas arquiteturas, e a partir da qual será convertida internamente para scripts compatíveis com outra ferramenta, denominada, Mercury. A ferramenta proposta irá converter o modelo dessa visão de alto nível, e através da comunicação com o Mercury, serão computadas as métricas de interesse através dos formalismos RBD (*Reliability Block Diagram*), EFM (*Energy Flow Model*) e SPN (*Stochastic Petri Nets*). Além disso, algoritmos de otimização foram integrados a ferramenta proposta. Com o intuito de encontrar uma combinação de componentes para uma dada arquitetura de *data center* em uma fração reduzida de tempo, em comparação ao algoritmo de força bruta (algoritmo que testa todos os casos), a partir de uma lista de preestabelecida de componentes.

Palavras-chave: Disponibilidade, redes de Petri, Modelo de Fluxo de Energia, Otimização.

Abstract

Sustainability has received increasing attention from the scientific community, with a strong focus on reducing energy consumption and maintaining nonrenewable resources for future generations. In parallel, the expansion of paradigms such as cloud computing, social networking, and ecommerce has increased the demand for data centers. In this context, tools that support the modeling of data center architectures and compute metrics such as availability, cost, and energy consumption are extremely important. This project proposes developing a tool with a high level vision for modeling data center architectures to compute energy consumption, availability, and cost. This tool development uses a graph library to represent the architecture's components, and from which it will be internally converted to scripts compatible with another tool, called Mercury. The proposed tool will convert the model of this highlevel view, and through the communication with Mercury, the metrics of interest will be computed by formalism such as RBD(*Reliability Block Diagram*), EFM (*Energy Flow Model*) e SPN (*Stochastic Petri Nets*). Furthermore, optimization algorithms were integrated into the proposed tool. In order to find a combination of components for a given data center architecture in a reduced fraction of time, compared to the brute force algorithm (algorithm that tests all cases), from a pre-established list of components.

Keywords: Availability, Petri nets, Energy Flow Model, Optimization.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

RBD	Reliability Block Diagram
SPN	Stochastic Petri Nets
EFM	Energy Flow Model
MILP	Mixed Integer Linear Programming

Sumário

1 Introdução

Atualmente, devido ao aquecimento global, o aumento da poluição, e a falta de água potável, entre outros fatores ambientais, o mundo começou a entender a necessidade de se realizar mudanças comportamentais em relação ao meio ambiente. Essas mudanças têm como principal foco a redução do consumo de energia e do impacto ambiental. Dentro desse contexto, a comunidade científica e várias empresas iniciaram uma busca por métodos de desenvolvimento que atendam às atuais necessidades de energia, sem comprometer os recursos não renováveis. Dessa forma, o desenvolvimento sustentável passou não apenas a ser um foco para melhorias, mas sim uma exigência da sociedade (BELL; MORSE, 2000).

Em relação a área de Tecnologia de Informação (TI), o surgimento de novos paradigmas como a computação em nuvem e as redes sociais, entre outros, tem demandado uma maior capacidade das infraestrutura dos *data centers*. A computação em nuvem vem dirigindo essa nova tendência de utilizar aplicações online, tomando como base a Internet, ao prover software como serviço. Dessa forma, crescentes demandas aos *data centers* surgiram, e além de desempenho satisfatório, passou-se a exigir que tais ambientes devem prover os serviços com alta disponibilidade. A consequência direta de tal crescimento da demanda é o aumento no custo operacional e no impacto ambiental, principalmente, devido à redundância nas arquiteturas de equipamentos necessários para prover as operações dos servidores.

Grandes *data centers*, que reúnem servidores com alto poder de processamento, são considerados ameaças ao meio ambiente por tenderem a aumentar sua demanda em 66% até 2035 (Than; Thein, 2020). Na verdade, a maior demanda por energia é uma questão que tem impactado a forma como os sistemas dos *data centers* são concebidos, no sentido de que os projetistas precisam verificar vários *trade-offs* e selecionar a solução mais viável considerando a utilização de energia e outras métricas, como a disponibilidade.

É importante destacar que esforços vêm sendo realizados com o objetivo de tornar os *data centers* mais sustentáveis. Uma análise do consumo de energia em conjunto com a utilização de fontes renováveis e limpas são de fundamental importância para a redução do impacto ambiental desses sistemas. Sendo assim, *data centers* sustentáveis devem ser concebidos fazendo uso da menor quantidade possível dos mais apropriados equipamentos, os quais devem demandar o mínimo de energia (BASH et al., 2008).

A utilização de ferramentas para modelagem de tais arquiteturas é de extrema

importância no apoio aos projetistas de *data centers* na estimação do impacto ambiental, da disponibilidade e do custo associado às infraestruturas antes mesmo de serem implementadas. Porém, atualmente, os projetistas de *data centers* possuem poucos mecanismos e ferramentas para suporte a avaliação das infraestruturas desses ambientes. Todavia, essas ferramentas exigem que os projetistas possuam um prévio conhecimento sobre os modelos (RBD - *Reliability Block Diagram*, EFM - *Energy Flow Model* e SPN - *Stochastic Petri Nets*) utilizados, o que acaba por dificultar a sua utilização. Nesse contexto, este projeto tem por objetivo a proposição de um ambiente computacional para a modelagem de arquiteturas de *data centers* com suporte a modelos formais como RBD, EFM e SPN. Além disso, também é foco desta pesquisa a proposição de uma visão de alto nível que pode ser automaticamente convertida para a quantificação de métricas como: disponibilidade, custo e consumo energético. Outro objetivo é o de prover maneiras para que um projetista de *data center* avalie as infraestruturas, sem que necessite conhecer o formalismo desses modelos utilizados para obtenção das métricas de interesse.

O restante desse trabalho está dividido da seguinte forma. A Seção 2 mostra os trabalhos relacionados presentes na literatura. A Seção 3 apresenta a fundamentação teórica necessária para um adequado entendimento dessa pesquisa. Uma visão da ferramenta proposta é apresentada na Seção 4. A Seção 5 ilustra a aplicabilidade da ferramenta proposta através de um estudo de caso com um cenário real que modela arquiteturas de data centers. Esse estudo de caso faz uso do algoritmo genético implementado na ferramenta para otimizar os resultados obtidos através dos modelos. Por fim, a Seção 6 apresenta as conclusões e os futuros direcionamentos dessa pesquisa.

2 Trabalho Relacionados

Essa seção apresenta os trabalhos relacionados encontrados na literatura sobre modelagem de sistemas computacionais e, também, sobre o ferramental para suporte a tais modelos. Por exemplo, os autores propuseram em (Nguyen et al., 2019) uma modelagem hierárquica para a avaliação de disponibilidade e confiabilidade em redes de *data centers* em estrutura de árvore. O foco foi modelar as infraestruturas físicas dos *data centers*. Para se computar a disponibilidade e confiabilidade foram usados modelos matemáticos como redes de Petri estocásticas (SPN), redes estocásticas de recompensa (SRN) e cadeias de Markov (CTMC). Através de um sistema em três camadas, foi possível mostrar que a distribuição ativa de nós na rede pode melhorar a disponibilidade e a confiabilidade em sistemas de computação em nuvem.

Em (FERREIRA et al., 2020) foi proposto um algoritmo para melhorar a distribuição do fluxo energético das infraestruturas elétricas de *data centers*. Esse algoritmo, PLDA-D, é capaz de realizar automaticamente a distribuição dos fluxos nos modelos EFMs e, assim, otimizar o consumo energético do sistema. A ideia adotada foi a de priorizar na distribuição do fluxo os caminhos que fazem uso dos equipamentos com maior eficiência energética.

Em (Liu et al., 2016), os autores propuseram uma nova abordagem para modelagem do serviço de confiabilidade de *data centers* em nuvem. Para isso, foi proposto um método com dois estágios: pedido e execução. A confiabilidade foi calculada através do modelo baseado em uma infraestrutura simplificada. No entanto, não foi o foco dos autores a criação de um ambiente para auxiliar projetistas na otimização da disponibilidade, por exemplo.

Uma modelagem utilizando programação linear inteira mista (MILP) para o planejamento de capacidade e minimização do custo total de propriedade (CTP) de *data centers* verdes de alta disponibilidade foi proposta em (Tripathi; Vignesh; Tamarapalli, 2017). Os autores demonstraram que a integração com energia verde auxilia na minimização do CTP. Resultados obtidos demonstraram que foi possível melhorar a distribuição do servidor, cruzando os sites, e também se miniminizou o CTP.

(MELO; JUNIOR; CALLOU, 2020) se utiliza das redes de Petri Estocásticas (SPN) para aferir os efeitos de uma política de manutenção em *data centers*. São apresentadas duas abordagens distintas de manutenção, as quais têm por objetivo evitar ou reduzir possíveis interrupções do sistema por falha dos equipamentos. Elas são: Manutenção Corretiva (ocorre quando algum dispositivo falha) e Manutenção Preventiva (objetiva evitar a falha de um dispositivo). Os resultados deste trabalho evidenciam

que a aplicação de tais políticas eleva a disponibilidade do sistema.

(AUSTREGÉSILO; CALLOU, 2019) propuseram o uso de algoritmos genéticos para otimizar custo, impacto ambiental e disponibilidade da infraestrutura de energia elétrica de sistemas *data centers*. O objetivo é o de maximizar a disponibilidade e minimizar o custo total e a exergia operacional. Com o intuito de computar tais métricas, foram realizados dois estudos de caso para mostrar a aplicabilidade e validação da estratégia proposta. Além disso, também foi proposta uma estratégia de otimização baseada em algoritmos genéticos mono-objetivos. Em relação aos resultados obtidos, observou-se uma melhora significativa, ficando próximos ao ótimo, alcançado por um algoritmo de força bruta que analisou todas as possibilidades. Os autores também observaram que com o uso do algoritmo genético o tempo para obtenção das respostas foi significantemente inferior.

Em (RAMPAZZO; YAMAKAMI; FRANÇA, 2013) são descritas duas abordagens para auxiliar no Planejamento da Operação de Sistemas Hidrelétricos. Elas são Algoritmo Genético e Algoritmo Diferencial Evolutivo. Como resultado foi demonstrado que as abordagens apresentam grande potencial na resolução do problema. Foram obtidos não só um resultado muito próximo do ótimo como também um conjunto de soluções distintas de boa qualidade.

Diferente dos trabalhos anteriores, esse trabalho propõe uma ferramenta, que faz uso de uma interface grafica simplificada, para a modelagem de arquiteturas de *data centers*. A partir dessa interface, a ferramenta automaticamente converte a arquitetura montada para os modelos formais necessários para se computar as métricas de interesse (SPN, RBD, EFM). Além disso, o ferramental proposto também possui um módulo de otimização, onde o algoritmo genético foi implementado para otimizar as arquiteturas analisadas nesse trabalho. Os resultados dessa análise foram comparados com o algoritmo de força bruta. Por fim, a Tabela 1 ilustra uma comparação das estratégias de modelagem, das métricas adotadas e das técnicas de otimização utilizadas em cada artigo.

Tabela 1 – Características dos Trabalhos Relacionados.

artigo	Modelagem	Metricas	Otimização
(Nguyen et al., 2019)	SPN,SRN,CTMC	disponibilidade, confiabilidade	-
(FERREIRA et al., 2020)	EFM	consumo energético	PLDA-D
(Liu et al., 2016)	novo modelo de confiabilidade	confiabilidade	-
(Tripathi; Vignesh; Tamarapalli, 2017)	MILP	custo, disponibilidade	framework
(MELO; JUNIOR; CALLOU, 2020)	SPN	disponibilidade	-
(AUSTREGÉSILO; CALLOU, 2019)	RBD	disponibilidade	Alg. Genetico
(RAMPAZZO; YAMAKAMI; FRANÇA, 2013)	modelagem de hidrelétrica	geração hidrelétrica	Alg. Genetico, Alg. Diferencial Evolucionario
Este Trabalho	SPN,RBD, EFM	disponibilidade, custo, exergia	Alg. Genetico, Alg. Diferencial Evolucionario, NSGA-II

3 Fundamentação Teórica

Essa seção apresenta os conhecimentos necessários para um melhor entendimento desse trabalho. A seção inicia com a apresentação dos *Data centers* que são os ambientes modelados pela ferramenta proposta. Em seguida, apresentam-se as redes de Petri Estocásticas, os Diagramas de Bloco de Confiabilidade e os Modelos de Fluxo de Energia. Esses são os formalismos adotados pela ferramenta para se computar as métricas de interesse (custo, disponiblidade, por exemplo). A ferramenta proposta é capaz de converter a visão de alto nível proposta (ver Seção 4.1) para os modelos em RBD, SPN e EFM no padrão da linguagem de script do Mercury. Por fim, essa seção apresenta o algoritmo genético de otimização utilizado. Esse algoritmo tem por objetivo retornar uma solução para um problema pré-definido em um tempo reduzido.

3.1 *Data centers*

Os *data centers* são ambientes computacionais conhecidos por abrigar equipamentos responsáveis pelo processamento e armazenamento de informações imprescindíveis para a continuidade de negócios das mais diversas organizações, sejam elas empresas, instituições de ensino, indústrias, órgãos governamentais, hospitais, hotéis, entre outros (MARIN, 2011). *Data centers* são comumente utilizados como host para sites com o objetivo de processar transações comerciais, para proteger dados (por exemplo, registros financeiros, histórico médico de pacientes, gerenciar e-mails). Além disso, as organizações estão procurando por departamentos de TI com o objetivo de receber suporte em muitas fontes de negócios, como produtividade e receita. Portanto, os *data center* devem ter altos níveis de disponibilidade para suportar novos requisitos de tecnologia. O grande problema é que os *data centers* são ambientes dinâmicos onde equipamentos vão se tornando obsoletos e são substituídos; novos equipamentos, que demandam novas interfaces e novos tipos de conexões, são criados para atender as demandas dos novos clientes e das novas necessidades que surgem dia após dia (ZUCCHI, 2013).

Os *data centers* desempenham uma função vital na atualidade, dada a grande quantidade de informações digitais que eles armazenam. Para atender os padrões requeridos e as necessidades dos usuários, a infraestrutura de um *data center* deve atender a requisitos técnicos rigorosos para garantir a confiabilidade, a disponibilidade e a segurança, pois eles têm um impacto direto no custo e na eficiência. Uma infraestrutura com alta confiabilidade e disponibilidade deve possuir redundância do sistema, e por esse motivo, será mais caro e provavelmente irá gerar um maior consumo de

energia elétrica (Levy; Hallstrom, 2017).

Ao se projetar uma sala de *data center*, muitas infraestruturas devem ser analisadas como por exemplo: energia, refrigeração, conectividade, espaço, proteções (por exemplo, contra fogo) e monitoramento de temperatura. Além disso, os projetistas dos *data centers* devem se concentrar no aumento da produtividade e evitar o tempo de inatividade, e as estratégias de projeto são fundamentais para se atingir essas necessidades. De uma forma simplificada, as infraestruturas dos *data centers* podem ser compostas por: fontes de energia de reserva, que são responsáveis por suportar a carga elétrica do *data center* quando a fonte de energia primária falhar; infraestrutura de rede redundante para manter o sistema funcionando em caso de ocorrer uma falha de um dispositivo de rede; servidores de aplicações e de dados redundantes.

3.2 Redes de Petri

O conceito de redes de Petri surgiu em 1962 na tese de doutorado de Carl Adam Petri, na faculdade de Matemática e Física da Universidade Darmstadt, na Alemanha (Murata, 1989). As redes de Petri são uma técnica de especificação de sistemas que permite uma representação matemática utilizada para modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos (MACIEL P.; LINS, 1996).

A rede de Petri (PN) é composta por 4 elementos: lugares, transições, arcos e tokens. Os tokens presentes nos lugares representam o estado do sistema. As transições são ações realizadas para alterar o estado do sistema. Para uma transição está habilitada, é preciso que as pré condições sejam satisfeitas. Os arcos representam o fluxo de tokens pela rede, e a distribuição dos tokens ao longo da rede representa o estado em que o sistema se encontra. Graficamente, os lugares são representados por círculos, as transições por barras, os arcos por setas, e os tokens por pontos. A Figura 1 ilustra os elementos da rede de Petri.

Figura 1 – Elementos da rede de Petri.

A Figura 2 mostra um exemplo de uma PN que modela o funcionamento de um sistema. Nessa rede, os lugares representam o estado do sistema (*on* ou *off*) e as transições representam as ações que alteram o estado do sistema (*ligar* e *desligar*). A Figura 2 (a) representa o sistema ligado, pois possui um token no lugar *on*. Nesse momento, a única transição habilitada para ser disparada é a transição *desligar*. Após o disparo dessa transição, o modelo passará para o estado desligado, e irá possuir

um token no lugar *off* (Figura 2 (b)). Em seguida, a transição habilitada para disparar é a *ligar*, e o disparo dessa fará o sistema voltar ao estado de ligado como na Figura 2 (a). Existem muitas extensões das redes de Petri, e esse trabalho faz uso de uma extensão denominada redes de Petri estocásticas (SPN).

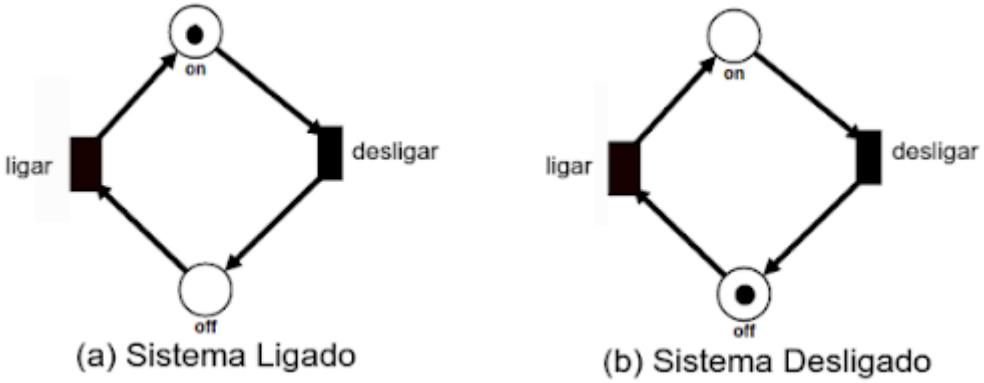

Figura 2 – Exemplo de uma rede de Petri.

3.3 Redes de Petri Estocásticas

As redes de Petri estocásticas representam uma extensão das PNs apresentadas anteriormente, pois adicionam a possibilidade de se modelar o tempo em que o sistema leva para mudar de um estado para outro. Sendo assim, um elemento denominado por transição temporizada é adicionado nas SPNs. Essa transição temporizada utiliza a noção de tempo, de modo que o disparo da transição corresponde ao período de execução da atividade. A seguir são ilustrados 2 exemplos utilizando SPNs.

3.3.1 Modelo de Manutenção Corretiva

A Figura 3 ilustra o modelo criado para representar o modelo base, onde um time de manutenção corretiva é alocado para realizar a manutenção de um data center. Quando um equipamento falha, o token passa para o lugar OFF. Um token no lugar reparar representa o momento em que o equipamento se encontra em falha e foi submetido a uma manutenção corretiva. Contudo, para que este lugar seja alcançado, é necessário que uma equipe de manutenção esteja disponível (um token no lugar equipeManutencao). Pode-se observar que existem três transições no modelo: MTTR (tempo médio para reparar), MTTF (tempo médio para falhar) e acionarManutencao.

3.3.2 Modelo de Manutenção Preventiva

A Figura 4 representa o modelo base de manutenção corretiva, acrescido da possibilidade da realização de manutenção preventiva. Neste modelo ocorre a adição

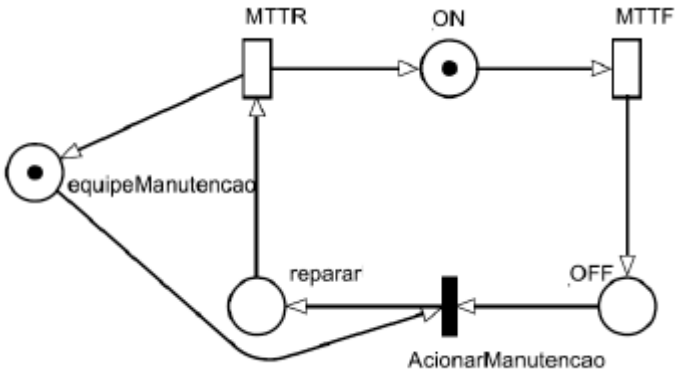

Figura 3 – Modelo Manutenção Corretiva.

de um novo lugar, em relação ao modelo de manutenção corretiva (Figura 5), REPARAR_PREV, para representar o momento da ocorrência da manutenção preventiva. A transição INICIAR_REPARO simula a periodicidade para a realização da manutenção preventiva. A transição TERMINAR_REPARO representa o tempo necessário para a realização da manutenção preventiva no dispositivo. A grande diferença da manutenção preventiva para a corretiva é que o MTTF do dispositivo na manutenção preventiva é reparado para o valor do equipamento novo.

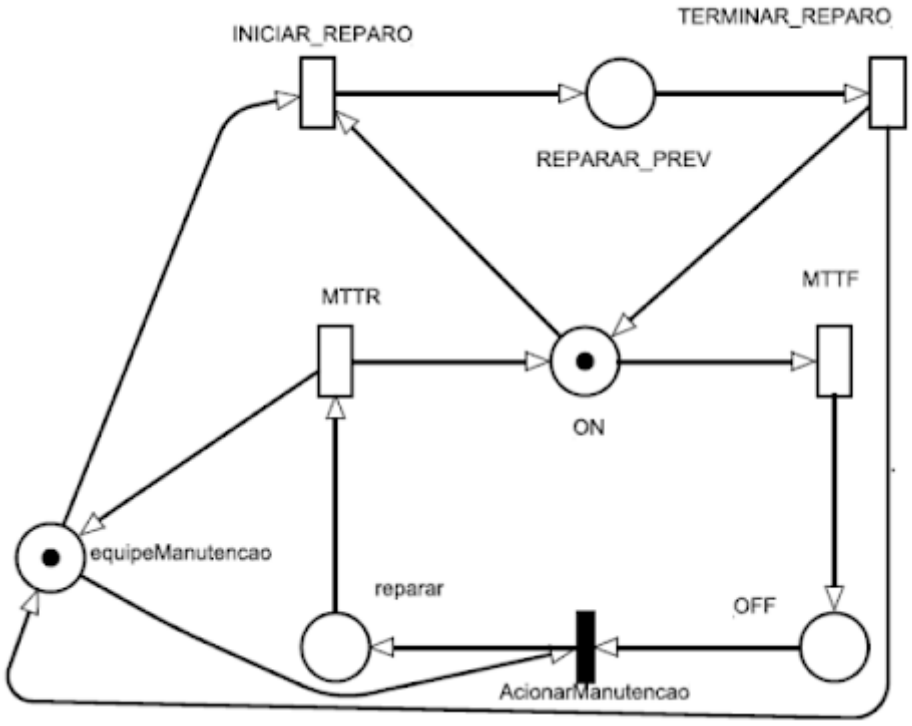

Figura 4 – Modelo Manutenção Preventiva.

3.4 Diagramas de Bloco de Confiabilidade

Os Diagramas de Bloco de Confiabilidade, do inglês Reliability Block Diagrams (RBD), foram inicialmente propostos como uma técnica para calcular a confiabilidade de sistemas. Atualmente, também são utilizados para calcular a disponibilidade. Em um RBD, os componentes são representados por blocos, interligados uns com os outros. Esses blocos podem representar composições: em série, em paralelo ou combinações dessas (TRIVEDI et al., 1996).

A Figura 5 ilustra dois exemplos de modelos em RBD. Na Figura 5 (a), os blocos são organizados em série; e na Figura 5 (b), os blocos são organizados em paralelo. No sistema em série, quando um componente falhar, o sistema todo irá parar. Já no sistema em paralelo, o sistema só irá falhar se ambos os blocos falharem.

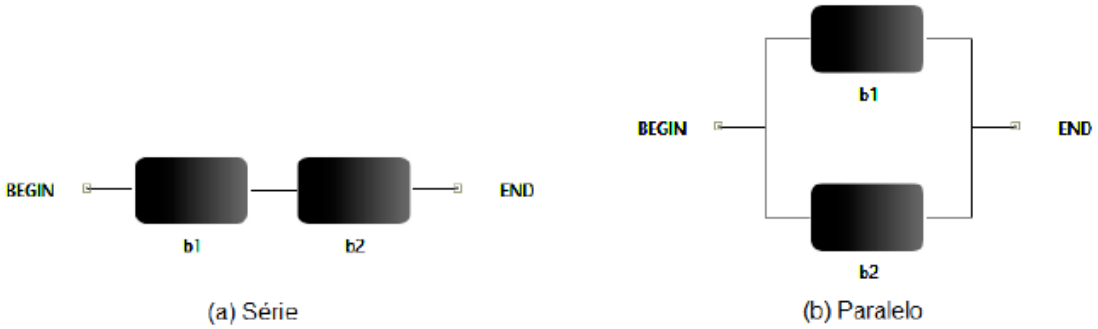

Figura 5 – Diagramas de Bloco de Confiabilidade.

3.5 Modelo de Fluxo de Energia

O Modelo de Fluxo de Energia (EFM) representa o fluxo de energia elétrica entre os componentes do *data center* sob análise. Esse modelo considera também a eficiência e a capacidade máxima que cada componente pode fornecer ou extrair (CALLOU et al., 2014a). O EFM é representado por um grafo acíclico dirigido em que os componentes são modelados como vértices e as respectivas ligações correspondem às arestas.

A Figura 6 ilustra um exemplo de um EFM que representa o fluxo entre componentes de um sistema de potência de data center. O peso existente nas arestas são utilizados para direcionar o fluxo. A representação gráfica do EFM tem um (1) por peso padrão nas arestas. Nesse modelo, os retângulos representam o tipo do dispositivo, e as etiquetas correspondem ao nome de cada dispositivo. O EFM pode ser utilizado para calcular a energia necessária para fornecer a potência demandada pelo ambiente de TI (representada no *TargetPoint1* da Figura 6). Além disso, o EFM também computa tanto o custo de aquisição como o custo operacional do ambiente sob análise.

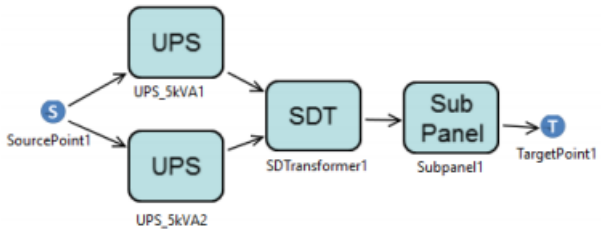

Figura 6 – Exemplo de um EFM.

3.6 Otimização

Otimização é uma técnica utilizada para se buscar encontrar a melhor solução possível respeitando as restrições dadas (RAO, 2009). As técnicas de otimização possuem um amplo leque de aplicações, levando em consideração que muitas empresas estam engajadas na resolução de problemas de otimização. Muitos dos problemas práticos e teóricos em engenharia, economia e planejamento podem ser modelados como problemas de otimização. Por exemplo, empresas de telecomunicações interessadas em otimizar o custo e a qualidade do serviço em seu projeto das redes de comunicação; empresas do mercado financeiro tem o objetivo de obter a melhor estratégia de alocação de ativos para otimizar o lucro e o risco; empresas de distribuição que visam otimizar a alocação das entregas (rotas) de forma a minimizar a distância percorrida pelos veículos (NOGUEIRA, 2015). Dentre as várias técnicas de otimização existentes, esse trabalho irá fazer uso do algoritmo genético, do diferencial evolutivo e do NSGAII.

3.6.1 Algoritmos Genéticos

Os Algoritmos Genéticos (AG) são algoritmos de otimização baseados na lógica da seleção e genética natural. Embora randomizados, os algoritmos genéticos não são uma caminhada aleatória simples, visto que compartilham informações da população para criar um novo indivíduo. Algoritmos genéticos foram desenvolvidos por John Holland (HOLLAND, 1975), seus colegas e seus alunos na Universidade de Michigan. Os objetivos de sua pesquisa foram: abstrair e explicar rigorosamente os processos adaptativos dos sistemas naturais e projetar softwares de sistemas artificiais que possuam mecanismos de sistemas naturais significativos (GOLDBERG, 1989).

Os algoritmos genéticos foram propostos com o intuito de aplicar a teoria da evolução das espécies de Darwin na computação. Esses algoritmos usam conceitos da evolução biológica como genes, cromossomos, cruzamento, mutação e seleção, procurando aprofundar o conhecimento em processos de adaptação em sistemas naturais e, baseado neles, desenvolver sistemas artificiais (simulações computacionais) que mantenham a mesma lógica dos mecanismos originais (OLIVEIRA, 2005). A lógica

funciona de forma semelhante ao processo de cruzamento biológico de cromossomos que compartilham informação genética para criar um novo indivíduo. No fim, chegam a obter um indivíduo de alta adaptação, que não significa uma solução ótima, mas sim a “melhor solução” encontrada (GOLDBERG, 1989).

Um algoritmo genético passa pelas seguintes etapas (LUCAS, 2002):(i) codificação do indivíduo, fase onde se cria um indivíduo para representar uma solução do problema (PAPPA, 2002); (ii) inicialização, etapa que produz uma população de soluções aleatórias (indivíduos) para o problema a ser otimizado; (iii) avaliação, nessa etapa se verifica a aptidão dos indivíduos através de uma função *fitness*, ou função objetivo, para se estabelecer a qualidade de cada solução gerada, resultando numa pontuação para cada indivíduo; (iv) seleção, nessa etapa indivíduos são selecionados para a reprodução a partir de sua aptidão; (v) cruzamento, essa fase é responsável por recombinar caracterısticas das soluções, gerando novos indivíduos.

3.6.2 Algoritmo Diferencial Evolutivo

Algoritmo Diferencial Evolutivo (Differential Evolution - DE), proposto por (STORN; PRICE, 1995), é um algoritmo de otimização simples e eficiente, que tem recebido cada vez mais destaque no âmbito da otimização não linear com variáveis contínuas. Por ser uma versão melhorada ao Algoritmo Genético (Kenneth Price; Rainer M. Storn; Jouni A. Lampinen, 2005), segue a linha dos algoritmos que evoluem uma população de soluções. Assim sendo, esse algoritmo é classificado como um algoritmo evolutivo, embora não tenha qualquer inspiração em um processo evolutivo natural.

A mutação do DE utiliza conceitos matemáticos e estratégias heurísticas. Seguindo os passos do algoritmo: (i) deve-se definir os limites superior e inferior para cada parâmetro utilizado; (ii) a população inicial é gerada aleatoriamente dentro desse espaço de busca estabelecido, tendo seus valores gerados de maneira uniformemente distribuída. Isso tende a levar o algoritmo a realizar uma varredura por todo o espaço de busca (LACERDA, 2010); (iii) o algoritmo realiza a mutação e o cruzamento sobre os indivíduos. Para a mutação, são selecionados aleatoriamente três indivíduos diferentes da população e, em sequência, é realizada a operação conhecida como mutação diferencial: Mi = X1 + p*(X2-X3).

Nessa operação é realizada uma perturbação em um indivíduo X1, denominado vetor base. Tal perturbação consiste numa diferença vetorial entre dois outros indivíduos X2 e X3. Esse vetor é então multiplicado por um peso p (entre 0 e 1), que irá determinar o tamanho do passo. Para completar o processo de busca, o algoritmo realiza o cruzamento, processo que consiste em criar um novo indivíduo, a partir de dados copiados de outros dois. No caso do Algoritmo Diferencial Evolutivo,(iv) cada vetor da população atual é cruzado com um vetor mutante criado com o procedimento anterior;

(v) tendo o novo indivíduo sido gerado, sua função objetivo é calculada, e o resultado é comparado com o resultado da função do indivíduo correspondente da população atual; (vi) caso o novo indivíduo seja igual ou melhor a seu correspondente, o atual é substituído. Caso contrário, o indivíduo atual é mantido; (vii) uma vez que a nova população é criada, os processos de mutação, cruzamento e seleção são repetidos até que um critério de parada seja alcançado.

3.6.3 NSGA-II

O NSGA-II é um algoritmo evolutivo que se baseia na de seleção da nova população baseado em dois principais operadores: a Ordenação de Pareto e uma métrica de promoção de diversidade da população que é livre de parâmetros, chamada *crowding distance*. Depois de uma população inicial P ser criada com um tamanho pré-definido, então se começa o processo de otimização: (i) são gerados descentes a partir de P, formando uma população Q, com o mesmo tamanho de P; (ii) para que ocorra a otimização é necessario que apenas os melhores prossigam e os demais sejam rejeitados. Para isso, P e Q são unidos em uma população total Pt; (iii) são executados os passos deste algoritmo que faz uso de ordenação de Pareto. A Ordenação de Pareto (Non-dominated Sorting) consiste em classificar uma população Pt em d subconjuntos, também chamados ranques. Tal classificação ocorre quando os indivíduos de Pt são avaliados por um criterio de não dominancia entre si um a um. Em outras palavras, são separados aqueles que empatam com mais vitorias e menos derrotas, baseados nos criterios de avaliação; (iv) após a classificação dos indivíduos no primeiro subconjunto (ranque 1), o resto da população repete o processo até que se chegue ao último subconjunto (ranque d), que possui os indivíduos que são dominados por todos; (v) oncluída a classificação, metade da população é escolhida para a próxima geração. Essa escolha é baseada nos primeiros ranques formados; (vi) contudo, o último ranque em que há indivíduos a serem passados para a geração seguinte pode exceder a quantidade de vagas para completar o tamanho pré-definido da população. Para resolver este problema, os indivíduos desse ranque devem ser classificados com base no critério de *crowding distance*.

Tal critério avalia os indivíduos desse ranque em questão e numa comparação um a um define as distâncias entre eles. Concluída essa definição, são escolhidos os indivíduos com as maiores distâncias, numa quantidade equivalente ao que falta para completar o tamanho da população. Esse criterio serve, além de preencher o que falta, para escolher o indivíduos mais diferentes e, assim, evitar que o processo de otimização fique preso em ótimos locais. A Figura 7 retrata esse processo de rejeição do excedente populacional.

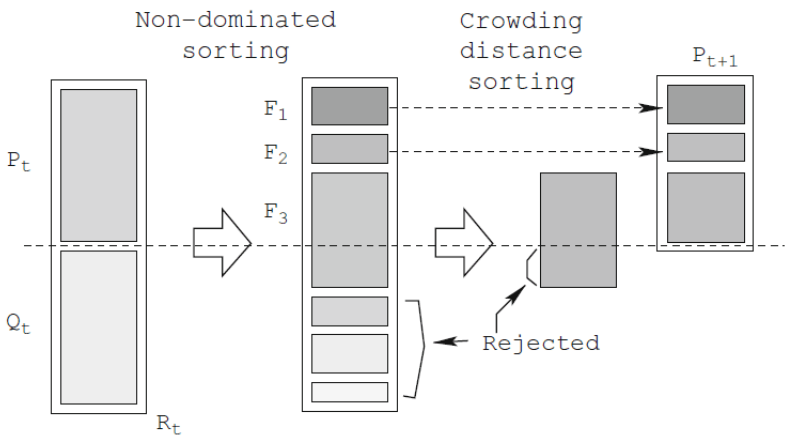

Figura 7 – Funcionamento do NSGA-II.

4 Uma Visão da Ferramenta

A ferramenta proposta surgiu com o intuito de interligar uma interface gráfica, com uma modelagem simplifica, proposta para representar arquiteturas de *data centers*, com os modelos formais responsáveis por calcular as métricas como disponibilidade, custo e consumo energético. A ferramenta foi desenvolvida em Java, utilizando as bibliotecas JAVA(Swing) e JGraphX. A Figura 8 mostra como é feita a integração da ferramenta proposta com outras ferramentas já existentes. Vale ressaltar que as avaliações dos modelos são realizadas a partir de outras ferramentas já existentes como, por exemplo, o CPN Tools (RATZER et al., 2003) e o Mercury (OLIVEIRA et al., 2017). É possível observar que a ferramenta foi implementada em camadas, existindo as camadas da visão de alto nível, a do parser, e os módulos de otimização e de manutenção. Esse trabalho apresenta uma visão geral do funcionamento da ferramenta proposta e, posteriormente, foca na apresentação detalhada do funcionamento da camada do parser.

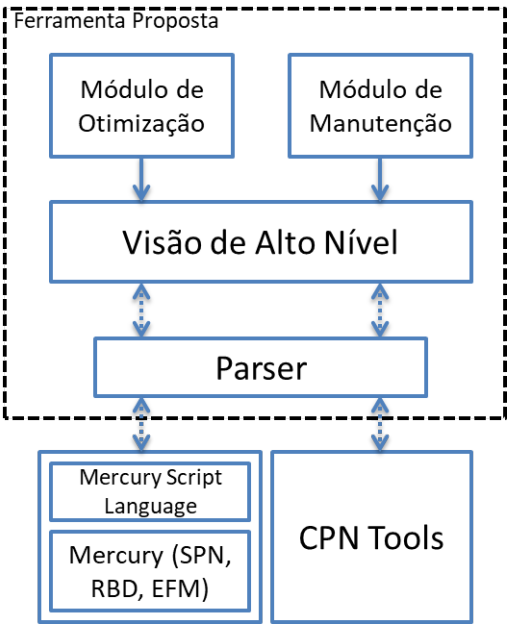

Figura 8 – Metodologia adotada.

A camada do parser é responsável pela conversão do modelo gerado pela ferramenta proposta para os modelos formais. Essa conversão é feita de acordo com as métricas que se deseja avaliar. Por exemplo, se o projetista desejar computar a disponibilidade, o parser terá que realizar a conversão do modelo proposto na ferramenta para o formalismo RBD ou SPN. Essa conversão é feita através da representação na linguagem de *script* do Mercury (OLIVEIRA et al., 2017). Essa linguagem de *script* permite a avaliação dos modelos SPN, RBD e EFM sem a necessidade de se chamar a

interface gráfica. Um módulo de otimização também foi desenvolvido para poder realizar a otimização dos modelos criados. Existe ainda um módulo de manutenção, o qual contem o padrão para modelos em SPN de manutenção preditiva e corretiva que servem para fazer a representação de uma equipe de manutenção que fará os reparos na arquitetura montada.

É importante mencionar a aplicabilidade do ambiente proposto, tendo em vista que a partir dessa ferramenta será possível desenvolver métodos de otimização que fazem uso de formalismos distintos. Dessa forma, técnicas multiobjetivo de otimização que façam uso de forma integrada dos formalismos de redes de Petri coloridas (CPN), redes de Petri estocásticas (SPN), modelo de fluxo de energia (EFM), por exemplo, poderão vir a serem implementadas de forma automatizada com a ferramenta proposta. A seguir, mais detalhes serão fornecidos sobre a visão de alto nível e sobre o parser.

4.1 Visão de Alto Nível

A ferramenta desenvolvida conta com uma visão única da representação dos sistemas computacionais de interesse. O foco dessa pesquisa será o de utilizar diferentes formalismos para se computar diferentes métricas (custo, disponibilidade, consumo energético, etc). A Figura 9 mostra uma visualização da interface gráfica.

Figura 9 – Ferramenta Proposta.

Na Figura 10 é possivel observar uma simplificação dos 3 sistemas de um *data center* (potência, TI e refrigeração) modelados na ferramenta proposta. Nessa figura, cada um dos retângulos representa um equipamento (componente) de um *data center* e as arestas representam as ligações físicas entre eles. O ferramental proposto fornece suporte a conversões para modelos em SPN, RBD e EFM.

Cada um dos componentes representados possui um conjunto de informações internas mostradas na Figura 11. Essas informações são utilizadas para se calcular as métricas desejadas. Dentre os formalismos para os quais é possível se traduzir o modelo de alto nível, o EFM exige algumas informações para o cálculo de suas

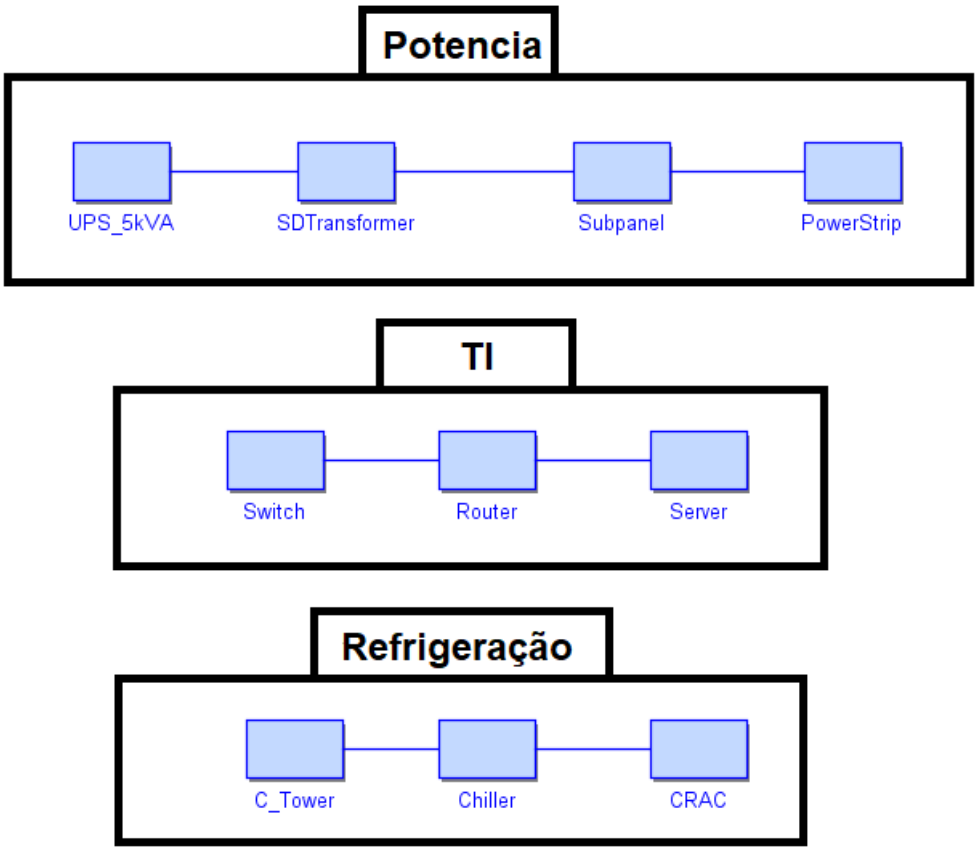

Figura 10 – Sistemas simplificados

métricas como a especificação da energia demandada pelo ambiente de TI, a eficiência energética, o custo de aquisição, e o tempo médio de falha (MTTF) de cada dispositivo.

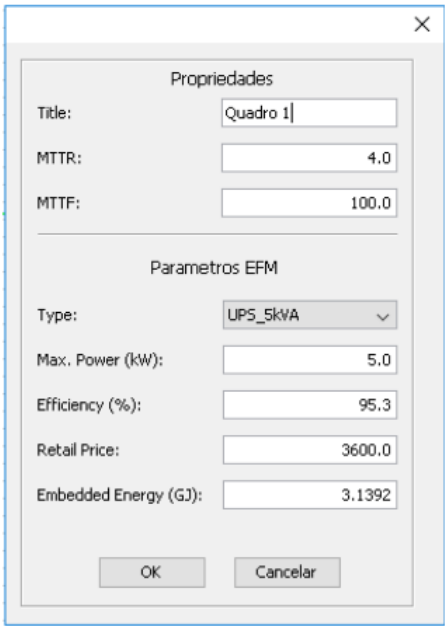

Figura 11 – Propriedades do Equipamento.

Essa ferramenta proposta traduz, automaticamente, o modelo de alto nível para os formalismos RBD, SPN e EFM no padrão adotado pela linguagem de *script* do Mercury. O formalismo a ser utilizado vai depender das métricas que se deseja computar e também da disposição dos equipamentos. A Figura 12 mostra um modelo representado no formato da linguagem de *script* do Mercury. A avaliação desse *script* é utilizada para se poder computar as métricas de interesse. Uma vez feita a avaliação, os resultados são obtidos e mostrados ao usuário.

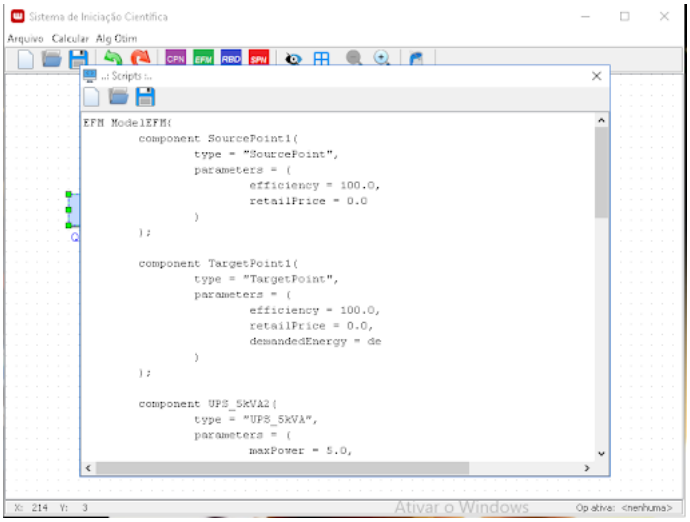

Figura 12 – Exemplo de *Script*.

Além disso, o Módulo de Otimização utiliza os resultados obtidos a partir desse mecanismo de avaliação para realizar os ajustes no modelo, ou seja, as melhorias na população. Esse processo pode ser repetido até se chegar ao critério de parada do algoritmo de otimização utilizado. No presente momento, o Módulo de Otimização conta com a implementação do Algoritmo Genético, DE, NSGAII. A seguir, mais detalhes são mostrados sobre o processo de conversão do modelo de alto nível para os diferentes formalismos utilizados.

4.2 Parser

O usuário ao utilizar a ferramenta escolhe se deseja calcular as métricas ou se deseja obter o *script* referente a arquitetura modelada. O parser é responsável por converter os modelos da visão de alto nível da ferramenta para os padrões dos ambientes *Mercury Script Language* e *CPN Tools*. Após a conversão, o parser retorna o *script* gerado para o usuário, ou manda-o para ser executado e então os resultados das métricas retornam para o usuário.

Este trabalho foca na conversão para o ambiente de modelagem com o Mercury, que dispõe da linguagem de *script* para representar os formalismos RBD, SPN e EFM

(utilizando apenas um target). Para realizar tal conversão, o parser faz uso de um algoritmo baseado em busca em profundidade. Basicamente, podemos dividir o parser em três módulos. Cada um desses módulos corresponde ao respectivo tradutor para o *script* dos formalismos RBD, SPN e EFM. Cada módulo constitui-se basicamente das mesmas partes para conversão no *script*, utilizando como entrada de dados o modelo de alto nível, gerando como saida o *script* no respectivo formalismo.

Afim de facilitar a explicação do processo de conversão, dividimos o padrão da linguagem de *script* do Mercury em três partes: início, meio e fim. Essa divisão em partes vale para todos os módulos (RBD, SPN ou EFM). Porém, existem algumas particularidades para cada formalismo. As partes de início e fim serão sempre constantes e independentes da arquitetura que está sendo modelada. Ressaltamos, todavia, que a parte de início e fim dependem para qual formalismo será convertido. No entanto, a parte intermediária (meio) sempre irá variar de acordo com a arquitetura sob análise. Assim, o parser fará a conversão da visão de alto nível para o padrão existente na parte intermediária e, em seguida, irá concatenar com as demais que compõe o *script* resultante. A seguir, detalhamos o algoritmo utilizado para fazer a geração dos *scripts* realizado pelo parser.

4.2.1 Geração do *Script*

A geração do *script* da linguagem do Mercury inicia com a chamada do Algoritmo 1. Esse algoritmo recebe como parâmetro uma lista de vértices que representa os equipamentos que compõe a arquitetura em análise. O algoritmo inicia com a criação das variáveis *início* e *fim* (linha 1 e 2), responsáveis por armazenar as partes constantes da linguagem de *script* (cabeçalho e rodapé) no padrão adotado pela ferramenta Mercury.

O algoritmo procede com a inicialização da variável *caminhos* (linha 3) com uma *string* vazia. Esta variável é utilizada para armazenar os caminhos do vértice inicial até o final. Posteriormente, é criada a variável *equipamentos* que armazenará as informações necessárias, de cada equipamento, para o cálculo das métricas. Essas duas variáveis compõem a parte intermediária do *script*.

O algoritmo prossegue fazendo uma varredura a partir de todos os vértices passados como parâmetro (Linha 5). Em seguida, a Linha 6 chama o método *montarEquipamento*, cujo retorno será armazenado dentro da variável *equipamentos*. Esse método reescreve as informações do equipamento para o padrão de *script* do respectivo formalismo (ex., SPN, RBD, ou EFM). O passo seguinte, Linha 7, faz uma chamada ao método *identificaInicial* para poder identificar se o vértice i é um vértice inicial. Em caso afirmativo, gera-se os caminhos possíveis considerando esse vértice como ponto de partida. Esse método *gerarCaminhos* é baseado na busca em profundidade para po-

der percorrer e listar todos os caminhos existentes no grafo a partir do vértice passado como parâmetro. Os percursos gerados são retornados e armazenados na variável *caminhos*.

Por fim, o algoritmo retorna a concatenação das variáveis *inicio*, *equipamentos*, *caminhos* e *fim* (Linha 12). Esse retorno corresponde ao *script* já no padrão demandado pelo Mercury, enviando o *script* de volta para a ferramenta, seja direto para o usuário visualizar ou para que seja executado e calculadas as métricas referentes. A seguir, os métodos chamados pelo Algoritmo 1 serão detalhados.

Algorithm 1 *gerarScript(vertices)*

```
1:  inicio := textoCabecalho;
2:  fim := textoRodape;
3:  caminhos := "";
4:  equipamentos := "";
5:  for i in vertices do
6:      equipamentos := equipamentos + montarEquipamento(i);
7:      inicial := identificaInicial(i);
8:      if (inicial == true) then
9:          caminhos := gerarCaminhos("", i);
10:     end if
11: end for
12: return (inicio + equipamentos + caminhos + fim);
```

4.2.2 Montagem do Equipamento

O Algoritmo 2 *montarEquipamento* é responsável pela montagem do equipamento para cada formalismo desejado. Esse algoritmo recebe como parâmetro um vértice (equipamento) do grafo analisado. A Linha 1 inicia o algoritmo com a criação da variável *bloco* que recebe as informações do equipamento (ex., identificação). Em seguida, a variável bloco receberá os parâmetros de acordo com a grafia do formalismo (ex., SPN, RBD, EFM) ao qual será feita a conversão. Sendo assim, esse método é dependente do formalismo de destino. Por último, a variável bloco é retornada (linha 3).

Algorithm 2 *montarEquipamento(vertice)*

```
1: bloco := "eqp" + vertice.getId();
2: bloco := bloco+\n + vertice.getParametros();
3: return bloco;
```

4.2.3 Identificação de Vértice Inicial

O Algoritmo 3 *identificarInicial* identifica se o vértice recebido como parâmetro corresponde a um vértice inicial do modelo sob análise. Vale destacar que mais de um

vértice inicial pode está presente no modelo. Esse algoritmo testa o seguinte princípio, onde um vértice é considerado inicial se não temos nenhuma aresta chegando nele. Esse algoritmo começa (Linha 1) inicializando a variável *maximo* com a quantidade de arestas do equipamento (vértice) passado como parâmetro. Em seguida, a Linha 2 inicializa a variável booleana *ehInicial* com *true*. Essa variável será retornada ao final do algoritmo, indicando se corresponde a um vértice inicial ou não.

A seguir o valor zero é atribuido a variável *contador*. Depois, segue-se para um laço onde todas as arestas desse vértice são percorridas (Linha 4). A Linha 5 recebe uma aresta ainda não percorrida do vértice sendo analisado. A partir dessa aresta, é possível saber o vértice de destino dela. Então, a Linha 6 recebe dessa aresta o vértice do grafo alvo ou de destino. Em seguida, é verificado se o alvo da aresta corresponde ao vértice passado como parâmetro ao modelo. O leitor deve recordar que isso é feito para se verificar que um vértice inicial não pode ter nenhuma aresta chegando nele. Caso o alvo seja correspondente ao vértice, esse vértice não é um vértice inicial. Sendo assim, a variável *ehInicial* tem seu valor setado para *false* (Linha 8), encerrando a análise (Linha 9). Por fim, a variável *ehInicial* é retornada (Linha 13). Caso a Linha 13 retorne *true*, isso indica que o vértice passado por parâmetro corresponde a um vértice inicial. Caso contrário, não é um vértice inicial.

Algorithm 3 *identificarInicial(vertice)*

```
1:  maximo := vertice.getEdgeCount();
2:  ehInicial := true;
3:  contador := 0;
4:  while (contador < maximo) do
5:     aresta := vertice.getEdgeAt(contador);
6:     alvo := aresta.getTarget();
7:     if (alvo == vertice.getId()) then
8:        ehInicial := false;
9:        break;
10:    end if
11:    contador := contador + 1;
12: end while
13: return ehInicial;
```

4.2.4 Geração de Caminhos

O Algoritmo 4 tem como parâmetros de entrada o *percurso* feito a partir do vértice inicial até o vértice atual, e o vértice recebido como parâmetro do algoritmo corresponde ao vértice atual. Esse algoritmo inicia atualizando a variável *percurso* com a adição da representação do equipamento do *vertice* (linha 2). Após isso, as variáveis *maximo* e *ehFinal* são inicializadas (linhas 4 e 5). A variável *maximo* armazena a quantidade de ligações do *vertice*. Já a variável *ehFinal* corresponde a um booleano

utilizado para informar se o equipamento é ou não final. Em seguida, a variável *caminhos* (linha 6), a qual irá armazenar todos os percursos de um equipamento inicial até os equipamentos finais, é inicializada com uma *string* vazia.

O algoritmo prossegue com um laço que faz a verificação de todas as ligações desse equipamento(vértice). A Linha 8 recebe uma aresta do *vertice*. A Linha 9 recebe o equipamento alvo dessa aresta. Em seguida, é verificado se o alvo é o equipamento contido na variável *vertice* (linha 10). Em caso negativo, a variável *ehFinal* tem o valor setado para *false* (linha 11). O passo seguinte (linha 12) é setar a variável *caminhos* que recebe o retorno da chamada recursiva ao método *gerarCaminhos*. Nessa chamada, são passados como parâmetros as variáveis *percurso* concatenado com a *string* " $-->$ " (representação da ligação na linguagem de *script*) e *target* (alvo).

Caso ao terminar a análise de todas as ligações a variável *ehFinal* seja verdadeira (linha 16), significa que durante a recursão chegou-se a um vértice final. Logo, a variável *percurso* receberá uma quebra de linha (linha 17) e será retornada (linha 18). A volta da recursão irá setar a variável *caminhos* que receberá o acréscimo do último caminho percorrido. Ao final, todos os caminhos a partir desse *vertice* serão retornados.

Algorithm 4 *gerarCaminhos(percurs, vertice)*

1: $idVertice = vertice.getId();$
2: $percurso := percurso + \text{“}eqp\text{”} + idVertice;$
3: $contador := 0;$
4: $maximo := vertice.getEdgeCount();$
5: $ehFinal := True;$
6: $caminhos := \text{“”};$
7: **while** $(contador < maximo)$ **do**
8: $\quad edge := vertice.getEdgeAt(contador);$
9: $\quad target := edge.getTarget();$
10: $\quad$ **if** $not(target.getId() == idVertice)$ **then**
11: $\quad\quad ehFinal := False;$
12: $\quad\quad caminhos := caminhos + gerarCaminhos(percurso + \text{“} --> \text{”}, target) + \text{“}\backslash n\text{”};$
13: $\quad$ **end if**
14: $\quad contador := contador + 1;$
15: **end while**
16: **if** $(ehFinal == True)$ **then**
17: $\quad percurso := percurso + \text{“}\backslash n\text{”};$
18: $\quad$ **return** $percurso;$
19: **end if**
20: **return** $caminhos;$

5 Estudo de Caso

Este capítulo apresenta dois estudos de casos com o objetivo de ilustrar a aplicabilidade da ferramenta proposta. Nestes estudos, arquiteturas de *data centers* são avaliadas a fim de se otimizar métricas. Para validar tais otimizações foi utilizado um algoritmo que testa todas as combinações de itens, chamado de algoritmo de força bruta. A escolha da comparação com o algoritmo de força bruta se deve pelo fato de que esse algoritmo analisa todas as possibilidades possíveis, levando em consideração toda a base de equipamentos disponíveis para se montar as arquiteturas de *data center* que se deseja otimizar. Assim, esse estudo compara o tempo de execução e a eficácia do algoritmo genético implementado na ferramenta proposta.

Cinco arquiteturas típicas de *data centers* (AVELAR, 2011), denominadas nesse trabalho por A1, A2, A3, A4 e A5, com graus de complexidade diferentes, foram adotadas para os Estudos de Caso 1 (5.1) e 2 (5.2). A Figura 13 mostra essas cinco arquiteturas elétricas de *data centers* modeladas na ferramenta proposta. A partir desses modelos, pode-se conduzir a avaliação em SPN,RBD e EFM, por exemplo, para se obter métricas como disponibilidade, exergia operacional, entre outras.

É importante destacar a similaridade dessa representação de alto nível adotada pela ferramenta e a utilizada na literatura por projetistas de *data center* (AVELAR, 2011). A arquitetura A1 é composta pelos seguintes equipamentos: UPS, transformador (SDTransformer), painel de energia (Subpanel) e régua de tomadas (PowerStrip). A arquitetura A2 corresponde a arquitetura anterior acrescida de um UPS redundante e um chaveador estático (Static Transfer switch - STS). Na arquitetura A3 é adicionado um transformador(SDTransformer2). A arquitetura A4 considera a arquitetura anterior acrescida de um painel de energia redundante (Subpanel2). Por fim, A5 tem todos os caminhos redundantes.

Além disso, é importante destacar que foi criada uma base de dados, com 5 equipamentos de cada tipo, para simular a variedade de equipamentos do mundo real. Os parâmetros de entrada para cada componente dessa base foram gerados a partir dos intervalos mostrados na Tabela 2, os quais vêm do trabalho (CALLOU et al., 2014b).

Foram utilizadas 3 métricas nos estudos de caso a seguir: disponibilidade, exergia operacional e custo total, sendo os dois ultimos usados apenas no estudo 2 (5.2). Por ser uma probabilidade, a disponibilidade varia de 0 a 1, podendo também ser representada em porcentagem. Como as avaliações dos modelos apresentam dízimas muito próximas de 1, foi computada a disponibilidade em função da quantidade de 9s

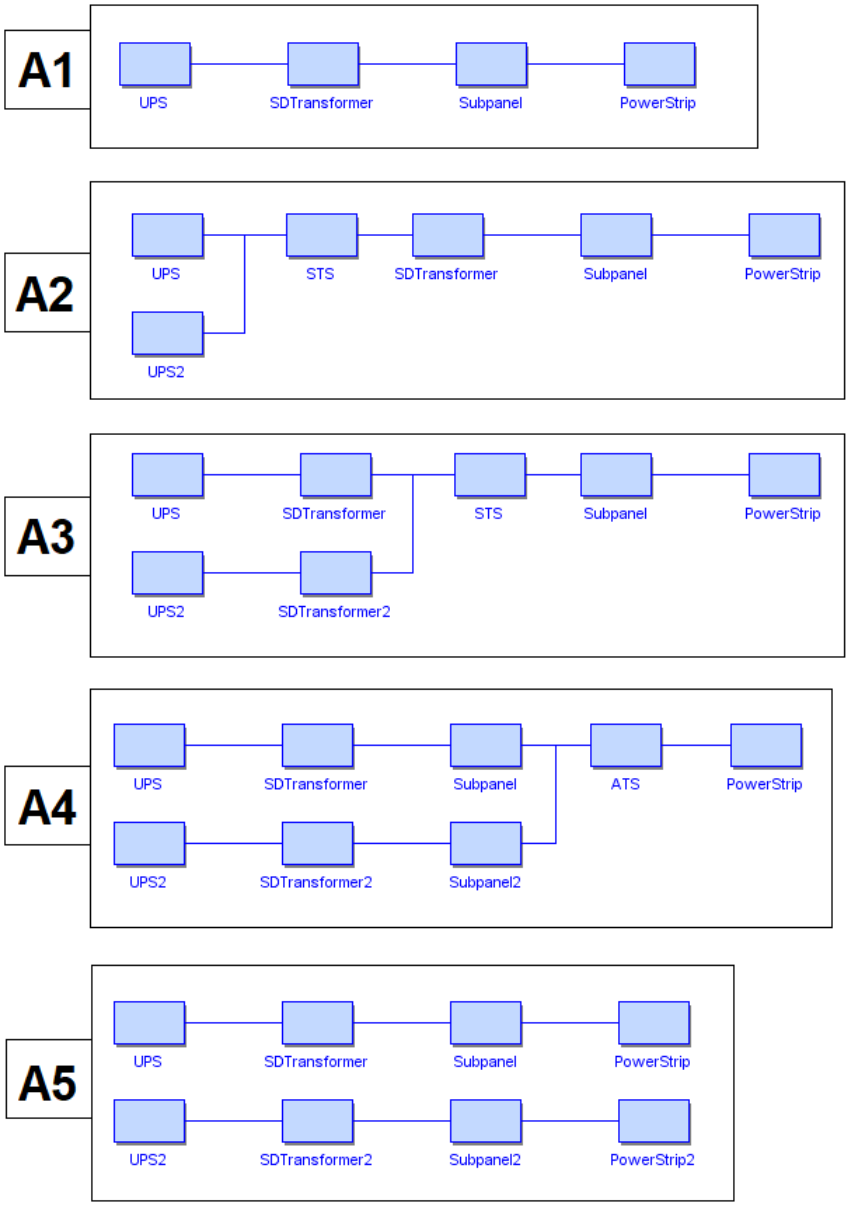

Figura 13 – Modelos das arquiteturas na ferramenta proposta.

(-log(1 - disp)). O custo total (custo de compra dos equipamentos + custo com consumo elétrico) é dado em dólares americanos (USD). A exergia operacional é dada em Joules (J).

Cada arquitetura é modelada na visão de alto nível, Figura 13, e internamente são criados os respectivos scripts (RBD, SPN e EFM) de acordo com a necessidade do caso. A partir da modelagem, cada indivíduo é ajustado a arquitetura durante a execução dos algoritmos de otimização para obter as métricas desejadas. Recebendo uma pontuação de acordo com uma função especifica em cada estudo de caso.

Tabela 2 – Parâmetro dos equipamentos adotados

Equipamentos	MTTF (Kh)	MTTR (h)	Eficiência (%)	Custo (USD)
UPS	[25; 75]	8,0	[90,5; 99,9]	[11250;18750]
STS	[24; 72]	6,0	[94,5; 99,9]	[600; 1000]
S. Down Transformer	[141,5; 423]	156,01	[93,5; 99,9]	[412,5; 687,5]
Subpanel	[152; 456]	2,4	[94,5; 99,9]	[150; 250]
Power Strip	[115,8; 345]	3,8	[94,5; 99,9]	[150; 250]

5.1 Estudo de Caso 1

Neste estudo foram utilizados o algoritmo genético e o algoritmo diferencial evolutivo. Foi escolhida uma função mono-objetivo, sendo essa a métrica disponibilidade. Os resultados obtidos por ambos os algoritmos foram comparados com o algoritmo de força bruta a fim de se poder averiguar a qualidade das soluções obtidas e, também, do tempo demandado de execução para cada estratégia.

A Tabela 3 mostra a comparação dos resultados obtidos através do algoritmo de força bruta e dos resultados alcançados a partir do algoritmo genético e do algoritmo diferencial evolutivo.

Tabela 3 – Comparativo entre algoritmo genético, dif. evolutivo e força bruta.

	Força Bruta		Alg. Genético		Alg. Dif. Evolutivo	
Arquit	Disp. (9s)	T. Exec.	Disp. (9s)	T. Exec.	Disp. (9s)	T. Exec.
A1	3,257	2,5s	3,254	2,2s	3,253	28,4s
A2	3,273	11,1s	3,271	1,23s	3,268	30,3s
A3	3,982	55,1s	3,957	1,27s	3,921	21,7s
A4	4,894	260,3s	3,93	1,29s	3,89	18,2s
A5	6,554	257,2s	6,427	1,31s	6,29	19,1s

Pode-se verificar que os resultados obtidos através da otimização com o Algoritmo Genético e pelo DE são muito próximos dos resultados obtidos por força bruta. No entanto, o tempo gasto para se executar cada uma das duas técnicas é bem distinto. À medida que a complexidade das arquiteturas aumenta, o tempo gasto para otimizar com força bruta cresce exponencialmente. No caso da otimização através do Algoritmo Genético, seu tempo de execução tende a permanecer aproximadamente o mesmo independente da complexidade.

Enquanto no Algoritmo Diferencial Evolutivo foi observado que em casos mais

simples o tempo de execução costuma ser maior, entretanto esse tempo tende a diminuir conforme aumenta-se a complexidade. Comparando com o Força Bruta, o Algoritmo Diferencial Evolutivo tem um tempo de execução maior em arquiteturas mais simples, o que é algo ruim. Mas à medida que a complexidade aumenta, o Algoritmo Diferencial Evolutivo supera o Força Bruta com um tempo exponencialmente menor.

Analisando ambos os dados obtidos dos algoritmos de otimização, Tabela 3, é possível notar que independente da arquitetura analisada, esses algoritmos alcançam resultados da disponibilidade com valores maiores que 95% do ótimo. Com um tempo de execução de menos de 3 segundos para o genético e 30 segundos para o DE, em contrapartida o força bruta chega a mais de 250 segundos (8 minutos) para as arquiteturas mais complexas. Comparando o tempo gasto para otimizar a arquitetura A5 utilizando Algoritmo Genético é aproximadamente 900 vezes menor, enquanto o Algoritmo Diferencial Evolutivo aproximadamente 450 vezes menor que o tempo gasto pela força bruta.

5.2 Estudo de Caso 2

Neste estudo foi trabalhada uma abordagem multiobjetiva com o intuito de otimizar as métricas disponibilidade, exergia operacional e custo total. Para tal foi escolhido o algoritimo NSGA-II. Mantendo o uso de um algoritmo de força bruta para a comparação e validação dos resultados do algoritmo utilizado.

A Tabela 4 mostra a comparação dos resultados obtidos através do algoritmo de força bruta e dos resultados alcançados a partir do algoritmo genético e do algoritmo diferencial evolutivo.

Tabela 4 – Comparativo entre NSGA-II e Força Bruta.

	Força Bruta				Alg. NSGA-II			
Arqt	Disp.	Exergia (J)	Custo (USD)	T. Exec.	Disp.	Exergia (J)	Custo (USD)	T. Exec.
A1	3,277	32,83	15400	6s	3,224	19,78	15250	4,2s
A2	3,25	47,18	27710	31,1s	3,27	24,27	27354	5,23s
A3	3,881	52,05	28542	88,1s	3,92	26,581	27947	6,27s
A4	3,95	32,05	28774	506,3s	3,9	29,41	27995	7,29s
A5	6,432	42,85	27640	884,2s	6,446	38,32	27349	8,31s

Nota-se que os resultados obtidos através da otimização multiobjetivo com o NSGA-II se aproximam muito dos resultados obtidos pelo força bruta. No entanto, o tempo gasto para se executar cada uma das duas técnicas é bem distinto. A medida que a complexidade das arquiteturas aumenta, o tempo gasto para otimizar com força bruta cresce exponencialmente. Já com o uso do NSGA-II, o tempo gasto permanece

aproximadamente o mesmo, e significativamente inferior ao tempo gasto pela força bruta. Observando a Tabela 4, pode-se perceber que o tempo gasto para se otimizar as arquiteturas utilizando algoritmo genético foi sempre inferior a 10 segundos.

6 Conclusão

Devido a crescente demanda de toda sociedade global por serviços de Internet, a disponibilidade dos *data centers* que dão suporte a tais serviços vem se tornando cada vez mais indispensável. Não apenas usuários em seu dia-a-dia, mas também empresas de todo porte dependem, de maneira ininterrupta, que esses serviços estejam sempre em funcionamento para realizar suas operações. Isso significa que qualquer interrupção no serviços de Internet pode causar grandes prejuízos a muitas empresas, simultaneamente, deixando a economia passível de sofrer impactos. Nesse contexto, esse trabalho fez a proposição de uma ferramenta para auxiliar projetistas a modelar arquiteturas de *data centers*, a partir de uma visão de alto nível, sem a necessidade de se conhecer o formalismo a ser usado para a obtenção das métricas de interesse, por exemplo, disponibilidade.

A ferramenta proposta permite a tradução do modelo de alto nível para os formalismos RBD, SPN e EFM, utilizando a linguagem de script do Mercury. A partir do desenvolvimento do ferramental, foi possível progredir para outra etapa do projeto. Essa fase correspondeu à integração da ferramenta com algoritmos de otimização, e que utilizam os mesmos formalismos. Dois estudos de caso foram apresentados mostrando a aplicabilidade da ferramenta já utilizando as técnicas de otimização, o primeiro faz uma comparação entre o algoritmo genético e o diferencial evolutico, onde foi possível observar a qualidade das solução obtidas em tempo bem reduzido se comparado ao algoritmo de força bruta; e o segundo foi a comparação multi-objetivo entre o NSGA-II e o força bruta, onde o NSGA-II mostrou resultados muito próximos do força bruta porém com um tempo de execução bem reduzido.

Em relação a trabalhos futuros, deseja-se implementar outros métodos de otimização, bem como a integração de um módulo de previsão de falhas. Além disso, também iremos realizar a modelagem de outras arquiteturas para a otimização de mais métricas e averiguar possíveis novas integrações com outras ferramentas para modelagem de *data centers*. Outro objetivo que se tem em mente é abranger a modelagem para a área de computação em nuvem.

Referências

AUSTREGÉSILO, M. S. S.; CALLOU, G. Stochastic models for optimizing availability, cost and sustainability of data center power architectures through genetic algorithm. *Revista de Informática Teórica e Aplicada*, v. 26, n. 2, p. 27–44, 2019. Citado 2 vezes nas páginas 14 e 15.

AVELAR, V. Comparing availability of various rack power redundancy configurations. APC White Paper 48, 2011. Citado na página 33.

BASH, C. et al. The sustainable information technology ecosystem. In: . [S.l.: s.n.], 2008. p. 1126 – 1131. ISBN 978-1-4244-1700-1. Citado na página 11.

BELL, S.; MORSE, S. Sustainability indicators: Measuring the immeasurable. *Journal of Rural Studies*, v. 16, 07 2000. Citado na página 11.

CALLOU, G. et al. An integrated modeling approach to evaluate and optimize data center sustainability, dependability and cost. *Energies*, Multidisciplinary Digital Publishing Institute, v. 7, n. 1, p. 238–277, 2014. Citado na página 20.

CALLOU, G. et al. An integrated modeling approach to evaluate and optimize data center sustainability, dependability and cost. *Energies*, v. 7, n. 1, p. 238–277, 2014. ISSN 1996-1073. Disponível em: <https://www.mdpi.com/1996-1073/7/1/238>. Citado na página 33.

FERREIRA, J. et al. An algorithm to optimise the energy distribution of data centre electrical infrastructures. *International Journal of Grid and Utility Computing*, Inderscience Publishers (IEL), v. 11, n. 3, p. 419–433, 2020. Citado 2 vezes nas páginas 13 e 15.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675. Citado 2 vezes nas páginas 21 e 22.

HOLLAND, J. H. *Adaptation in Natural and Artificial System*. [S.l.]: University of Michigan, 1975. Citado na página 21.

Kenneth Price; Rainer M. Storn; Jouni A. Lampinen. *Differential Evolution*. 3. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 2005. ISBN 978-3-540-31306-9. Citado na página 22.

LACERDA, A. S. *Proposta de um Algoritmo Evolucionário Nebuloso para Solução de Problemas de Otimização Multiobjetivo*. Tese (Doutorado), 06 2010. Citado na página 22.

Levy, M.; Hallstrom, J. O. A new approach to data center infrastructure monitoring and management (dcimm). In: *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 17.

Liu, Y. et al. Service reliability modeling of the it infrastructure of active-active cloud data center. In: *2016 Prognostics and System Health Management Conference (PHM-Chengdu)*. [S.l.: s.n.], 2016. p. 1–7. Citado 2 vezes nas páginas 13 e 15.

LUCAS, D. *Algoritmos genéticos: uma introdução*. 2002. Universidade Federal do Rio Grande do Sul, . Apostila elaborada sob a orientação de Luis Otavio Alvares, para a disciplina de Ferramentas de Inteligência Artificial. Citado na página 22.

MACIEL P.; LINS, R. D. C. P. R. F. *Introdução às redes de petri e aplicações*. [S.l.: s.n.], 1996. Citado na página 17.

MARIN, P. S. *Data Centers-Desvendando Cada Passo-Conceitos, Projeto, Infraestrutura Física e Eficiência Energética*. São Paulo - SP:: Érica, 2011. Citado na página 16.

MELO, F. F.; JUNIOR, J. S.; CALLOU, G. de A. Evaluating the impact of maintenance policies associated to sla contracts on the dependability of data centers electrical infrastructures. *Revista de Informática Teórica e Aplicada*, v. 27, n. 1, p. 13–25, 2020. ISSN 21752745. Disponível em: <https://seer.ufrgs.br/rita/article/view/RITA_27_V1_13>. Citado 2 vezes nas páginas 13 e 15.

Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541–580, April 1989. ISSN 0018-9219. Citado na página 17.

Nguyen, T. A. et al. Reliability and availability evaluation for cloud data center networks using hierarchical models. *IEEE Access*, v. 7, p. 9273–9313, 2019. Citado 2 vezes nas páginas 13 e 15.

NOGUEIRA, B. *Exploração multiobjetivo do espaço de projeto de sistemas embarcados de tempo-real não críticos*. Tese (Doutorado) — Universidade Federal de Pernambuco. Centro de Informática, 2015. Citado na página 21.

OLIVEIRA, D. et al. Advanced stochastic petri net modeling with the mercury scripting language. In: *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*. New York, NY, USA: ACM, 2017. (VALUETOOLS 2017), p. 192–197. ISBN 978-1-4503-6346-4. Disponível em: <http://doi.acm.org/10.1145/3150928.3150959>. Citado na página 25.

OLIVEIRA, H. *Algoritmo evolutivo no tratamento do problema de roteamento de veículos com janela de tempo*. 2005. Monografia (Graduação em Bacharelado em Ciências da Computação), Departamento de Ciências da Computação, Universidade Federal de Lavras. Citado na página 21.

PAPPA, G. *Seleção de atributos utilizando algoritmos genéticos multiobjetivos*. Dissertação (Mestrado) — Programa de Pós Graduação em Informática Aplicada da Pontifícia, 2002. Departamento de Estatística e Informática. Citado na página 22.

RAMPAZZO, P. C. B.; YAMAKAMI, A.; FRANÇA, F. O. de. Algoritmo genético e evolução diferencial para a resolução do problema de planejamento hidrelétrico. In: LOPES, H. S.; RODRIGUES, L. C. de A.; STEINER, M. T. A. (Ed.). *Meta-Heurísticas em Pesquisa Operacional*. 1. ed. Curitiba, PR: Omnipax, 2013. cap. 19, p. 307–324. ISBN 978-85-64619-10-4. Citado 2 vezes nas páginas 14 e 15.

RAO, S. *Engineering Optimization: Theory and Practice: Fourth Edition*. [S.l.]: John Wiley and Sons, 2009. ISBN 9780470183526. Citado na página 21.

RATZER, A. V. et al. Cpn tools for editing, simulating, and analysing coloured petri nets. In: AALST, W. M. P. van der; BEST, E. (Ed.). *Applications and Theory of Petri Nets 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 450–462. ISBN 978-3-540-44919-5. Citado na página 25.

STORN, R.; PRICE, K. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, v. 23, 01 1995. Citado na página 22.

Than, M. M.; Thein, T. Energy-saving resource allocation in cloud data centers. In: *2020 IEEE Conference on Computer Applications(ICCA)*. [S.l.: s.n.], 2020. p. 1–6. Citado na página 11.

Tripathi, R.; Vignesh, S.; Tamarapalli, V. Optimizing green energy, cost, and availability in distributed data centers. *IEEE Communications Letters*, v. 21, n. 3, p. 500–503, 2017. Citado 2 vezes nas páginas 13 e 15.

TRIVEDI, K. et al. Reliability analysis techniques explored through a communication network example. 12 1996. Citado na página 20.

ZUCCHI, A. A. W. *Construindo um data center*. 2013. Revista USP, no. 97, pp. 43– 58. Citado na página 16.