Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Reginaldo Luis de Luna Junior

Melhoria da eficiência energética de comitês de classificadores de fluxo de dados para computação de borda

São Carlos
2024

Reginaldo Luis de Luna Junior

Melhoria da eficiência energética de comitês de classificadores de fluxo de dados para computação de borda

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas Distribuidos e Redes

Orientador: Hermes Senger

São Carlos
2024

Dedico este trabalho para minha primeira filha Esperança.

Agradecimentos

É com imenso prazer e gratidão que escrevo para expressar meus sinceros agradecimentos a todos que contribuíram para a entrega do meu trabalho de mestrado.

Em primeiro lugar, gostaria de agradecer à minha esposa, Débora Rocha, por todo o amor, apoio e paciência ao longo desses anos de estudos. Sem ela, eu não teria alcançado este importante marco em minha carreira. Enquanto nascia este trabalho, também nascia o amor da nossa vida, nossa pequena filha Esperança. Espero que você possa aproveitar o melhor que esta vida pode lhe dar.

Também sou profundamente grato ao meu orientador, Hermes Senger, que dedicou tempo e conhecimento para me ajudar a superar os desafios e me guiar na minha pesquisa. Seu comprometimento e incentivo foram fundamentais para o sucesso deste trabalho.

Gostaria de expressar minha admiração e gratidão a Guilherme Cassales, meu antecessor de pesquisa, que compartilhou comigo suas experiências e me deu valiosos insights para o desenvolvimento do meu trabalho.

Meu eterno agradecimento ao meu camarada Kaio Leal. Há uma frase de uma das músicas do Don L que diz "uma frase muda o fim do filme" e, aquela nossa conversa dentro do ônibus, voltando do nosso primeiro trabalho, foi a conversa que mudou o fim do filme, não só meu, mas de várias outras pessoas. Você sempre será lembrado como peça fundamental da minha biografia.

Além disso, não posso deixar de agradecer ao coletivo Quebradev: Gustavo Castilião, Marcos Eptaccio, Vitor e todos os outros colegas que fizeram parte da minha jornada acadêmica. Seus conselhos, apoio e amizade foram essenciais para me manter motivado e focado em alcançar meus objetivos.

Não poderia deixar de agradecer aos grandes cientistas que me ajudaram ao longo deste trabalho, seja com revisões de texto, apresentações ou por serem uma referência científica na contestação da realidade em que vivemos. Em especial, agradeço à brilhante cientista da computação Raquel Queiroz, autora de uma dissertação extraordinária sobre a utilização de Brain-Computer Interfaces e algoritmos genéticos, e também à Beatriz

Araújo, pelo trabalho e pela pesquisa crítica sobre o marxismo e o direito na subjetividade jurídica. Suas reflexões foram fundamentais para que eu pudesse ampliar minha visão e enxergar mais longe no horizonte.

À minha família, à minha mãe, Cleusa Rosa, por me apoiar em cada loucura minha desde a infância, ensinando-me a tentar mesmo que não consigamos, para que assim possamos aprender. Este ensinamento levo comigo para a vida. Aos meus irmãos, Jeferson e Jackson, por terem sido pioneiros na nossa difícil infância e me possibilitado estudar e ser o primeiro da família a ter uma graduação, pagando o ônibus para a faculdade e até uma alimentação quando eu não tinha dinheiro. E a todos os outros que estiveram comigo, dando apoio, incentivo e carinho, como minha cunhada Bárbara e meu pai, Reginaldo, que me ensinou que, além de tudo, o trabalho é o mais importante.

Gostaria de expressar meus agradecimentos às crianças deste país. Este trabalho é dedicado a vocês, que sempre instigam a curiosidade e merecem um futuro melhor. Em especial, gostaria de agradecer ao meu afilhado, Bernardo Luna.

Agradeço também ao rap, que sempre me inspirou e me motivou a buscar o melhor em mim. E, claro, à ciência e à universidade pública, que tornaram possível a realização deste trabalho e possibilitam o avanço científico em nosso país.

Por fim, gostaria de agradecer a todos aqueles que dedicam tempo e horas de estudo para o avanço científico. É graças ao esforço coletivo e ao compartilhamento de conhecimento que podemos avançar em busca de soluções para os desafios que enfrentamos.

"Direitos iguais para todos. Privilégios só para as crianças."
(Leonel Brizola)

Resumo

Computação de borda (CB) surgiu como uma arquitetura que pode ajudar a reduzir a demanda de energia e as emissões de gases de efeito estufa das tecnologias digitais. A computação de borda oferece baixa latência, mobilidade e ciência da localização para dispositivos sensíveis a atrasos, conectando os serviços de computação em nuvem aos usuários finais. Métodos de aprendizado de máquina (AM) têm sido cada vez mais utilizados em dispositivos na borda para classificação de dados e processamento de informações. Os comitês de classificadores tem demonstrado bom desempenho preditivo em problemas de classificação de fluxo de dados. A estratégia *mini-batching* melhora o reuso de dados dos caches na execução de comitês de classificadores em arquiteturas multi-core para a classificação de fluxos de dados online. A estratégia consiste em agrupar temporariamente dados de um fluxo e processá-los em conjunto. Como efeito, o *mini-batching* pode acelerar as aplicações e reduzir o consumo de energia. Neste trabalho, investigamos a fusão das etapas de treinamento e de classificação dos dados, trazendo ainda mais ganhos de reuso dos caches e melhorias no desempenho preditivo. Também comparamos *mini-batching* a duas estratégias que são suportadas pelo hardware de processadores multi-core utilizados em dispositivos de borda, que são a redução da frequência do *clock* e o desligamento de núcleos de processamento. Avaliamos as estratégias comparando seu desempenho e eficiência energética para a classificação de fluxos de dados usando seis algoritmos de comitês de classificação de última geração e quatro *datasets* de referência. Os resultados mostram que estratégias de *mini-batching* podem reduzir significativamente o consumo de energia em 95% dos experimentos, melhorando a eficiência energética em 96% em média e em 169% no melhor caso sobre as estratégias de hardware. Da mesma forma, a nova estratégia de *mini-batching* proposta melhorou a eficiência energética em 136% em média e 456% no melhor caso. Por fim, propusemos uma estratégia de otimização adaptativa e multi-objetivo para escolher dinamicamente o tamanho do mini-batching em função da ocupação da CPU das taxas de chagada de dados. A escolha do tamanho do *batch* usa o princípio de Pareto para otimizar tanto o tempo de resposta quanto o consumo de ener-

gia. Resultados mostram melhoria do consumo energético em 17 dos 24 casos avaliados. Já para a métrica de atraso, não houve uma redução significativa quando comparado a *batches* de tamanho 50 (apontado na literatura como uma boa escolha). Em resumo, a estratégia dinâmica oferece redução do consumo energético, sem perdas no tempo de execução.

Palavras-chave: Eficiência Energética, comitê de classificadores, fluxo de dados, computação de borda.

Abstract

Edge computing (EC) has emerged as an architecture that can help reduce the energy demand and greenhouse gas emissions of digital technologies. Edge computing offers low latency, mobility, and location awareness for delay-sensitive devices, connecting cloud computing services to end-users. Machine learning (ML) methods have increasingly been used on edge devices for data classification and information processing. Classifier ensembles have demonstrated good predictive performance in data stream classification problems. The strategy called *mini-batching* was previously proposed in the literature to improve cache data reuse when executing classifier ensembles on multi-core architectures for online data stream classification. The strategy involves temporarily grouping data from a data stream and processing them together. As a result, *mini-batching* can speed up applications and reduce energy consumption. However, the originally proposed *mini-batching* offers opportunities for further improvements. In this work, we investigate the fusion of the training and classification stages of the data, bringing more gains in cache reuse and predictive performance improvements. We also evaluate the *mini-batching* strategy compared to two strategies supported by the hardware of common multi-core processors used in edge devices: clock frequency reduction and processor core shutdown. We evaluate the strategies by comparing their performance and energy efficiency for data stream classification using six state-of-the-art classifier ensemble algorithms and four benchmark datasets. The results show that *mini-batching* strategies can significantly reduce energy consumption in 95% of the experiments, improving energy efficiency by an average of 96% and by 169% in the best case over hardware strategies. Similarly, the newly proposed *mini-batching* strategy improved energy efficiency by an average of 136% and 456% in the best case. Finally, we proposed an adaptive and multi-objective optimization strategy to dynamically choose the mini-batching size based on CPU occupancy and data arrival rates. The batch size choice uses the Pareto principle to optimize both response time and energy consumption. Results show an improvement in energy consumption in 17 of the 24 cases evaluated. However, for the latency metric, there was no significant

reduction compared to batch sizes of 50 (pointed out in the literature as a good choice). In summary, the dynamic strategy offers reduced energy consumption without losses in execution time.

Keywords: Energy efficiency, ensembles, data stream, edge computing.

Lista de ilustrações

Lista de tabelas

Lista de algoritmos

Lista de siglas

ARF Adaptive Random Forest

AT Arrival Time

CPU Central Processing Unit

CL Core Limiting

CL2 Core Limiting 2

CL4 Core limiting 4

DVFS Dynamic Voltage and Frequency Scalling

DPM Dynamic Power Management

FWI Full Waveform Inversion

FPT Finish processing time

GCC GNU Compiler Collection

IPS Instances per Second

ICTs Information and communication technologies

ICT Information and communication technologie

ILT Instance Living Time

JPI Joules per Instance

J Joules

LLVM Low Level Virtual Machine

LBag Leveraging Bagging

MB Mini Batching

MB-LF Mini Batching with Loop Fusion

MB-D Mini Batching Dynamic

MOA Massives Online Analysis

ML Machine Learning

MOOP Multi-Objective Optimization Problem

NL Negative Large

NM Negative Medium

NS Negative Small

OB Online Bagging - Oza Bag

OBADWIN Online Bagging with ADWIN

OBASHT Online Bagging with Adaptive Size Hoeffind Tree

PS Positive Small

PM Positive Medium

PL Positive Large

RD Reuse Distance

RF Random Forests

RTL Relation Theory Of Locality

RAM Random Access Memory

S Sequencial

SRP Streaming Random Patches

W Watt

VFH2 Voltage Frequency Half 2

VFH4 Voltage Frequency Half 4

VFS Voltage Frequency Scalling

ZE Zero

AIR Airlines

GMSC Give Me Some Credit

,

ELEC Electricity

COV Covertype

Sumário

Capítulo 1

Introdução

A demanda de energia elétrica para tecnologias de informação e comunicação do inglês Information and communication technologies (ICTs) cresce aproximadamente entre 6% e 8% por ano. Em 2018, as ICT consumiram cerca de 14% do uso global de eletricidade e esse número pode ultrapassar os 20% em até 2030 (ANDRAE; EDLER, 2015). No pior cenário, o que irá acontecer se não houver mudanças na eficiência energética das tecnologias de ICT é chegar em até 51% da eletricidade global utilizadas para esse fim até o ano de 2030. O mesmo estudo sugere, para o pior cenário, o uso da eletricidade de CT pode contribuir em 23% das emissões globais de gases do efeito estufa no mesmo ano.

A computação de borda é um paradigma de computação distribuída que permite que o processamento seja feito mais próximo da fonte de dados ou de usuários e aplicações que os consomem. Com o surgimento da internet das coisas (do inglês *Internet of Things* - IoT) e o desenvolvimento das redes 5G, a computação de borda é considerada a chave para reduzir latências de processamento das tecnologias de análise de dados, como reconhecimento facial, análise de dados em esteiras de produção, carros autônomos entre outras tecnologias. Este paradigma, além do processamento mais próximo da fonte de origem, possibilita economizar energia e tornar mais eficiente esse consumo (CAO et al., 2020).

Os algoritmos de aprendizado de máquina (*Machine Learning*) são os: supervisionado, não supervisionado, semi-supervisionado e por reforço, cada tipo de algoritmo tem sua complexidade e sua aplicação que deve ser analisada caso a caso, para os algoritmos supervisionados podemos separar em dois tipos de problemas de mineração de dados, sendo eles, classificação e regressão.

Podemos observar no dia a dia que a computação em borda e o processamento de fluxos de dados estão cada vez mais presentes. Esse processamento nos permite classificar

e tornar mais rápido essa tarefa, economizando recursos físicos, energéticos e reduzindo o tempo de espera, contando com o processamento mais próximo da fonte de origem muitas vezes (BIFET; GAVALDA, 2007).

Trabalhos recentes buscam formas de otimizar esses algoritmos de classificação reduzindo o consumo energético e também melhorando o seu desempenho computacional, dessa forma, tendo uma resposta mais rápida (CASSALES et al., 2020). O trabalho de Cassales et al. (2020) propôs a técnica do *mini-batching* para buscar uma melhoria de *software*, reduzindo o tempo de processamento e melhorando a eficiência energética. O *mini-batching* também organiza o acesso à memória e, como consequência, os dados estão mais próximos aos níveis de cache do processador. Outros trabalhos do mesmo autor (CASSALES; GOMES H, 2021; Cassales et al., 2022) investigaram a eficiência energética desta estrategia e também o tamanho de *mini-batching* ideal procurando um tamanho fixo para uma redução do consumo energético e também do tempo de processamento. Este presente trabalho é uma extensão das pesquisas já executadas em Cassales et al. (2020), Cassales et al. (2022), Cassales e Gomes H (2021).

Entre as técnicas que foram exploradas neste trabalho estão algumas que permitem o melhor uso das memórias *cache* do computador, possibilitando uma redução no consumo energético e um melhor desempenho, tais como a transformação de *loops*. Alguns trabalhos utilizam transformação diretamente da compilação do código (PALLISTER; HOLLIS; BENNETT, 2015). A transformação de *loops* permite que o acesso entre as hierarquias de memória seja mais eficiente, melhorando o desempenho computacional e também reduzindo o consumo de energia elétrica.

Outra técnica explorada é a adaptação dinâmica do *mini-batching*, em Das et al. (2014), estratégias de *batch* dinâmico utilizam conceitos de controle, buscando um equilíbrio entre o desempenho e consumo energético para um melhor processamento. Este presente trabalho propõe uma metodologia de processamento de um fluxo de dados com o tamanho de *batch* dinâmico. Utilizando um simples algoritmo de controle que adapta o tamanho do *batch* conforme a necessidade. Esse controle é feito através da taxa de ocupação da Central Processing Unit (CPU).

Algumas outras técnicas exploradas de hardware são utilizadas para buscar o equilíbrio entre desempenho e consumo energético, como *Dynamic Voltage and Frequency Scalling (DVFS)* e o *Dynamic Power Management (DPM)*, essas são estratégias conhecidas para aplicação no nível de hardware. DVFS é utilizado para otimizar a energia modificando a frequência utilizada no *clock* de um processador, ao reduzir a frequência deste, obtemos uma maior economia de energia. Todavia, ao fazer uma mudança no *clock* perdemos também a capacidade de processamento e para isso um estudo buscou aplicar as técnicas de otimização e utilizou o conceito de Pareto para buscar o melhor *trade-off* entre aplicações paralelas e a mudança no *clock* (COUTINHO et al., 2020). DPM, por sua vez, fornece possibilidades para o desligamento de componentes do hardware que não

estejam sendo utilizados, por exemplo, colocar um core em modo de *sleep* ou desligar completamente já que não será utilizado para determinado programa. Através dessa técnica é possível compreender quais recursos físicos do computador podem entrar em estados variados que consomem menos energia. Isso possibilita que tenhamos mais possibilidades de configuração para a execução de determinadas tarefas computacionais (BENINI; BOGLIOLO; MICHELI, 2000).

Em nosso estudo avaliamos o usa das técnicas de desligamento de componentes do hardware e redução da frequência dos processadores para buscar otimizações destas configurações, iremos analisar com melhorias nos algoritmos de classificação, como o *mini-batching* citado anteriormente, uma implementação de melhoria em seu acesso à memória através das técnicas de transformação de *loops*. E, por fim, será implementado a escolha dinâmica do tamanho limite do *batch* a ser processado a partir da quantidade de instâncias a serem processadas e do consumo da máquina pelo monitoramento de uso da CPU.

1.1 Objetivos e Desafios da Pesquisa

Este trabalho, tem como principal objetivo reduzir o tempo de atraso (*delay*) e reduzir consumo energético sem piorar a qualidade preditiva dos algoritmos de classificação, para dispositivos localizados na borda. Iremos utilizar algoritmos de classificação do tipo comitê para melhorar o poder computacional em processamento de instâncias e equilibro do consumo energético. Com base em trabalhos anteriores, investigaremos estas possibilidades no decorrer deste trabalho.

Tendo em vista que localidade de acesso aos dados da memória cache é um fator primordial para o alto processamento das instâncias dos comitês, diminuição do tempo de atraso (*delay*) e redução do consumo energético, esta pesquisa tem os seguintes objetivos:

1. Em trabalhos anteriores o *mini-batching* só havia sido aplicado nas etapas de treino e classificação separadamente. Investigamos possíveis benefícios de fazer a fusão dessas etapas (loops) com vistas a melhorar ainda mais o tempo de processamento das instâncias, reduzindo o *delay* e reduzindo o consumo energético destes processamentos.

2. Até então, *mini-batching* não havia sido comparado com outras estratégias implementadas no hardware dos processadores. Iremos comparar o uso do Mini Batching (MB) com técnicas como a limitação dos números de cores e/ou o controle da frequência do *clock* do processador.

3. Estudos anteriores analisaram o desempenho de *batchs* de tamanho estático. Investigamos uma estratégia de ajuste dinâmico do tamanho do *batch*, que seja capaz de se adaptar as variações da carga ao longo do tempo conforme a ocupação da CPU.

1.2 Organização do Trabalho

Esta dissertação está organizada da seguinte forma. Neste capítulo de introdução apresentamos para você o problema de pesquisa juntamente com uma visão geral de como os trataremos, além de mostrar na seção a seguir as publicações e reconhecimentos deste trabalho. No capítulo 2 trataremos da fundamentação teórica para embasar a pesquisa proposta, colocando uma visão geral sobre conceitos básicos utilizados nesta pesquisa para relembrar e facilitar a compreensão. O capítulo 3 apresenta os trabalhos relacionados utilizados como base bibliográfica. O capítulo 4 descreve detalhadamente a proposta de pesquisa que foi validada nesse estudo. capítulo 5 apresenta os experimentos e a discussão dos resultados obtidos. O capítulo 6 apresenta propostas de trabalhos futuros para essa pesquisa e então, apresentamos a conclusão dos experimentos realizados e do avanço obtido através do desenvolvimento desta pesquisa.

1.3 Publicações e Reconhecimentos

Essa seção mostra os trabalhos publicados ou aceitos em anais de eventos e conferências, além do reconhecimento do trabalho executado nesta pesquisa.

1. LUNA, Reginaldo; CASSALES, Guilherme; SENGER, Hermes. **Improving Performance and Energy Efficiency of the Classification of Data Streams on Edge Computing.** In: ESCOLA REGIONAL DE ALTO DESEMPENHO DE SÃO PAULO (ERAD-SP), 14. , 2023, São José dos Campos/SP. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2023 . p. 1-4. DOI: <https://doi.org/10.5753/eradsp.2023.231687>.

2. LUNA, Reginaldo, Senger, Hermes e Cassales Guilherme autores do artigo denominado **Improving Performance and Energy Efficiency of the Classification of Edge Computing**, foram escolhidos pela Comissão de Programa da 14ª Escola Regional de Alto Desempenho de São Paulo (ERAD-SP 2023), como autores do **MELHOR TRABALHO AO NÍVEL DE PÓS-GRADUAÇÃO** do evento.

3. Reginaldo Luna, Guilherme Cassales, Bernhard Pfahringer, Albert Bifet, Heitor Murilo Gomes, and Hermes Senger. 2024. **Mini-batching with Fused Training and Testing for Data Streams Processing on the Edge**. In 21st ACM International Conference on Computing Frontiers (CF'24), May 7–9, 2024, Ischia, Italy. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3649153.3649188>

1.4 Meios de Divulgação

1. Eficiência Energética em Comitê de Classificadores de Fluxo de Dados Online - Palestrante: Reginaldo Junior. São Paulo: Afroya Tech Conf, 05 nov. 2022.

Capítulo 2

Fundamentos teóricos do trabalho

Neste capítulo, serão apresentados os fundamentos teóricos e tecnológicos utilizados neste trabalho. Este capítulo está dividido da seguinte maneira: na seção 2.1 apresentamos a base teórica e o estado da arte do fluxo de dados; então na seção 2.3 apresentamos os algoritmos para classificação em fluxo de dados, na seção 2.6 é apresentado o modelo de redes conhecido como computação de borda, então apresentamos na seção 2.7 as técnicas de hardware para desempenho e redução do consumo energético e a estratégia de *mini-batching* na seção 2.8 e, por fim, discutimos a hierarquia de memórias em seção 2.9 e a localidade de dados em seção 2.10.

2.1 Processamento de fluxo de dados

O processamento de cenários dinâmicos com fluxo de dados apresenta desafios computacionais relacionados, como, por exemplo, a infraestrutura, a disponibilidade e o tempo de resposta (CANO; KRAWCZYK, 2019). Formalmente, o fluxo de dados é um conjunto de dados multidimensional, potencialmente ilimitado $x_1, x_2, ..., x_n$, chegando nos momentos $t_1, t_2, ..., t_n$ nos quais cada x_i é uma instância d-dimensional, notados por $x_i = (x_i^1 ... x_i^d)$ (AGGARWAL et al., 2003). Dado esse cenário potencialmente ilimitado de dados, não é possível armazenar todas as informações, de modo que o modelo de classificação deve mudar a cada novo dado processado. Com cada nova mudança de conceito, os conceitos anteriores podem ser esquecidos, outros conceitos podem ser aprendidos e conceitos existentes podem ser ajustados

Além disso, um fluxo de dados é comumente utilizado em ambientes de natureza dinâmica, onde a distribuição é não-estacionária e as alterações ocorrem continuamente. Esse fenômeno ficou conhecido como *concept drift* (TSYMBAL, 2004). De acordo com

Gama e Rodrigues (2009), o grande desafio no fluxo de dados é manter os modelos sempre atualizados com a acurácia, especialmente quando o aprendizado requer um incremento automático. Esses algoritmos utilizam o conceito de *concept-drift* associados a esse processo. Nesses casos, o modelo precisa descartar os antigos exemplos que já não refletem os dados presentes que estão sendo distribuídos e então adaptar a decisão ao novo modelo de dados.

Mesmo que o *concept drift* tenha uma extensa literatura que investiga seu funcionamento, a maioria dos trabalhos presume que a distribuição de dados não depende dos exemplos anteriores, ou seja, uma nova instância não depende da instância anterior do fluxo. Essa dependência é predominante em fluxos de dados provenientes de dispositivos de gravação de dados, tais como vigilância por vídeo e sensores. Logo, a dependência temporal adiciona mais um desafio à classificação do fluxo de dados (ŽLIOBAITĖ et al., 2015).

Algoritmos de mineração de dados evoluíram ao longo do tempo, especialmente nas últimas duas décadas. Anteriormente, predominavam os cenários conhecidos como *batch*, nos quais era possível utilizar lotes de dados, realizando diversas varreduras na base e nos modelos disponíveis, sem a necessidade de retreinamento dos modelos. No entanto, atualmente, a dinamicidade das informações permite que os dados sejam produzidos de forma contínua. A partir disso, consideramos que os modelos estão em constante mudança ao longo do seu processamento (OLIMPIO et al., 2021).

Em resumo, o fluxo de dados apresenta vários desafios para algoritmos de aprendizagem, incluindo, mas não se limitando a: uma quantidade enorme de exemplos, altas velocidades de chegada, exemplos rotulados limitados, recursos restritos (tempo e memória), novas classes, *concept-drift*, um trade-off entre precisão e eficiência, aplicativos distribuídos e dependências temporais (GOMES; READ; BIFET, 2019; AGGARWAL et al., 2003).

Dessa forma, a literatura atual apresenta diversas técnicas de otimização exclusivas para o processamento do fluxo de dados. Nós próximos tópicos, serão demonstrados alguns algoritmos do tipo comitês de classificadores específicos para a classificação em fluxo de dados, técnicas de otimização de desempenho e redução energética.

2.2 Comitês de classificadores

Um comitê de classificação, ou, em inglês, *ensemble*, consiste em diversos treinadores combinados para melhorar o desempenho preditivo do modelo. Quando desenhado um comitê de classificação, uma das suas características, segundo a literatura, é a diversidade dos tipos de treinadores, especialmente em relação aos erros de classificação. Isso significa que cada membro deve ser o mais único possível (POLIKAR, 2006).

Os comitês permitem que os resultados individuais de um conjunto de classificadores

possam ser combinados para classificar novas instâncias. Dessa forma, é possível obter uma maior variedade de modelos, de forma que, quando juntos, conseguem um desempenho preditivo maior do que quando são utilizados treinadores isolados (GAMA et al., 2004).

Os comitês de classificadores demonstram em estudos um desempenho preditivo frequentemente melhor que quando os classificadores individuais são utilizados (DIETTERICH, 2000). Por fim, um classificador com alta acurácia é quando a equação: $\frac{ICC}{n}$ possui uma alta taxa de acertos, onde ICC é o número de instâncias classificadas corretamente e n o total de instâncias processadas.

Podemos compreender melhor o alto desempenho preditivo dos comitês através da seguinte análise:

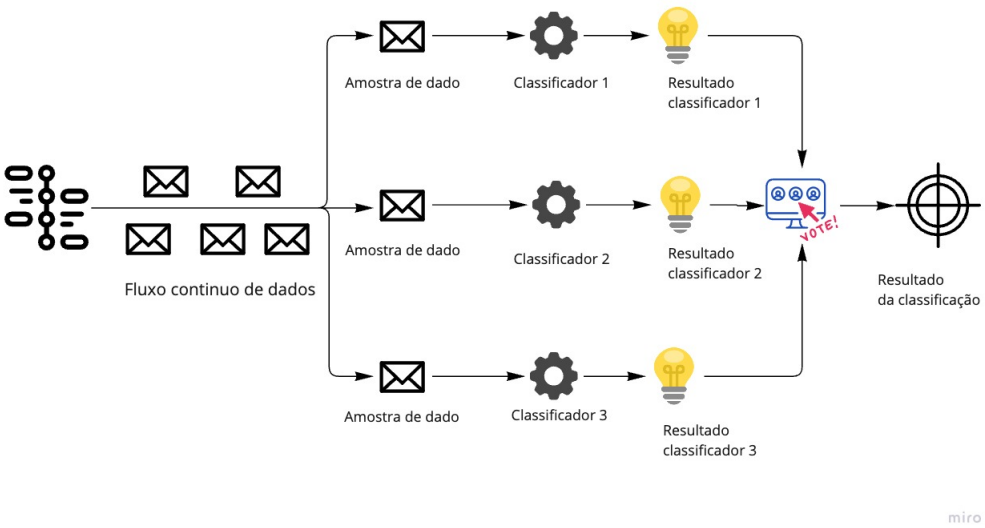
> Para entender porque a alta acurácia e a diversidade entre os classificadores é uma boa opção, imagine um conjunto de três classificadores: $C1, C2$ e $C3$, e considere uma nova instância x. Se os três classificadores forem idênticos (ou seja, não diversos), quando $C1(x)$ estiver errado, consequentemente $C2(x)$ e $C3(x)$ também estarão. No entanto, se os erros cometidos pelos classificadores não forem correlacionados, quando $C1(x)$ estiver errado, existe a probabilidade de $C2(x)$ e $C3(x)$ estarem corretos. Assim, uma simples votação majoritária entre os classificadores $C1, C2$ e $C3$ poderá classificar a instância x corretamente (OLIMPIO et al., 2021).

A literatura classifica três destes métodos de comitês de classificação (BREIMAN, 1996), (SCHAPIRE, 1999) e (WOLPERT, 1992), sendo que cada tipo de comitê de classificação tem uma forma de utilizar seus modelos e cálculos agregados para a construção de um melhor desempenho preditivo no processamento das informações.

Bagging (BREIMAN, 1996) é uma contração do nome *boostrap agreggation*. A técnica tem como principal objetivo resolver o problema de *overfitting*, que ocorre quando o modelo aprende demais sobre os dados de treino, tornando-se adequado apenas para esses dados e incapaz de generalizar para novos dados. Nesse caso, o desempenho do modelo é excelente nos dados de treino, mas cai drasticamente nos dados de teste (DOMINGOS, 2012). Originalmente desenhado para a classificação com modelos em árvores de decisão, o *bagging* também é usado para classificação e regressão. As saídas destes modelos são combinadas por média no caso de regresso ou em votação, quando utilizado para classificação (SEWELL, 2008). A Figura 1 mostra ilustra esse processamento.

Boosting (SCHAPIRE, 1999) é um modelo que, assim como o *bagging*, se baseia em uma média classificatória de todos os classificadores utilizados. Originalmente concebido para classificação e, posteriormente, utilizado como regressão de dados, o *boosting* inicialmente cria um aprendizado fraco em seu primeiro treinamento e, em seguida, melhora esse treinamento com os modelos agregados de forma aleatória até que se consiga chegar em um forte aprendizado (SEWELL, 2008). A Figura 2 ilustra seu funcionamento.

Figura 1 – Exemplo de funcionamento da técnica *bagging*



Fonte: O Autor

Já o último modelo de comitê de classificação conhecido na literatura é o *stacking* (WOLPERT, 1992), que introduz a ideia da possibilidade de mais de um modelo de comitê utilizado simultaneamente. Em outras palavras, é possível combinar o *bagging* e o boosting para obter uma melhora de acurácia na predição dos dados. Neste trabalho, será utilizado somente o modelo *bagging*.

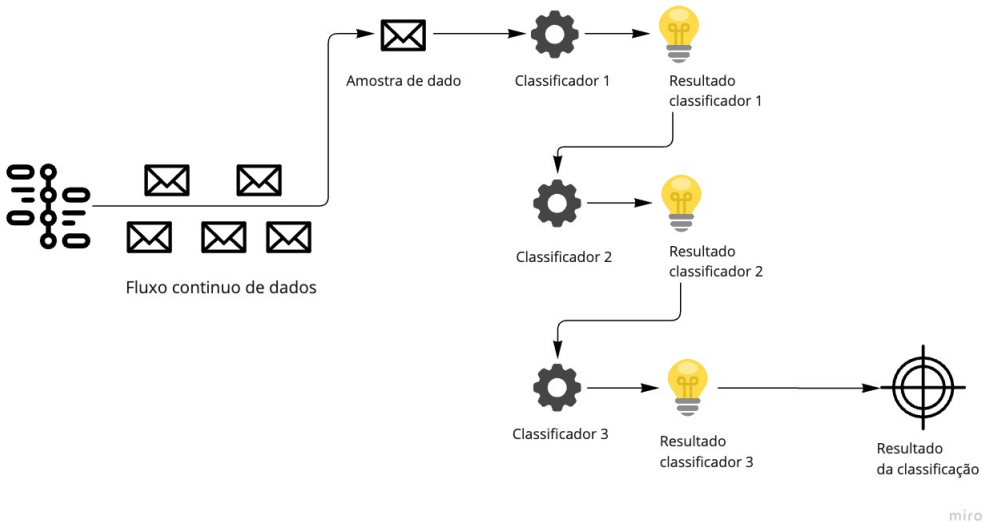
2.3 Algoritmos para classificação de fluxo de dados

É possível encontrar diversos algoritmos para classificação em fluxo de dados. Neste trabalho, focaremos naqueles que foram utilizados nos trabalhos relacionados para experimentação e análise (BIFET; HOLMES, 2010) (CASSALES, 2021). Esses algoritmos de classificação utilizam como base *Hoeffding Tree* (DOMINGOS; HULTEN, 2000).

De acordo com Domingos e Hulten (2000), *Hoeffding Tree* é uma estrutura de dados que permite construir árvores de decisão incrementais com base em um fluxo de dados em tempo real. Ao contrário de outras árvores de decisão, que exigem o processamento de todo o conjunto de dados para construir a árvore, *Hoeffding Tree* usa um teste estatístico chamado teste *Hoeffding* para decidir quando um nó pode ser dividido, permitindo que a árvore seja construída de forma incremental e com menor uso de recursos computacionais. Em resumo, a estrutura de dados e o algoritmo *Hoeffding Tree* permitem a construção de árvores de decisão incrementais com base em fluxos de dados em tempo real.

Esta seção detalha o funcionamento de um algoritmo utilizando a estrutura de dados

Figura 2 – Exemplo de funcionamento da técnica *boosting*



Fonte: O Autor

com o *Hoeffding Tree* e, posteriormente, os algoritmos de classificação que foram usados neste trabalho: *Oza Bagging with ADWIN*, *Leveraging Bagging*, *Oza Bagging with Adaptive Size Hoeffding Tree*, *Oza Bagging*, *Adaptive Random Forest* e por fim *Streaming Random Patches* que são construídos como comitês de classificação do tipo *Bagging*.

2.3.1 Hoeffding Tree

Hoeffding Tree é um algoritmo de classificação incremental que é capaz de se adaptar às mudanças de conceitos. Baseado em árvores de decisão, fornece soluções adequadas para diversos problemas computacionais.

Esse classificador é desenvolvido com base no modelo de árvores de decisão e consegue classificar dados contínuos utilizando o modelo *Hoeffding bound* (HOEFFDING, 1994). Esse modelo é a base matemática para que *Hoeffding tree* assuma que uma pequena amostra de dados pode ser suficiente para escolher o melhor atributo de classificação (HOEFFDING, 1994; MARON; MOORE, 1993). O conceito do *Hoeffding bound* afirma que, com probabilidade de $1-\delta$, a verdadeira média da variável é de pelo menos $r-\epsilon$, onde r é o valor médio calculado a partir das observações de tamanho n que são independentes e ϵ pode ser definido pela Equação:

$$\epsilon = \sqrt{\frac{R^2(ln(1/\delta))}{2n}} \tag{1}$$

A etapa de decisão ocorre quando um número de exemplos satisfaz o *Hoeffding bound*,

considerando o melhor atributo escolhido como o nó de divisão da árvore. Com esse algoritmo, não é necessária a utilização de armazenamento para a classificação de novos dados, pois cada instância é lida uma única vez. Isso torna a predição unitária capaz de se adaptar a cada modelo em caso de alterações na distribuição de dados (LEMAIRE; SALPERWYCK; BONDU, 2014).

O Algoritmo 1 apresenta o funcionamento detalhado do classificador *Hoeffding Tree*. Inicialmente é gerada uma árvore com uma única folha (Algoritmo 1 Linha 11). A seguir uma função irá analisar o ganho da informação $\bar{G}$ (Algoritmo 1 Linha 12). Abaixo podemos ver o ganho da informação pela entropia através da seguinte equação:

$$H(X) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i), \tag{2}$$

sendo p a probabilidade de observar o valor V_i.

Com a árvore inicializada, o procedimento segue os seguintes passos: a instância é difundida pela árvore, analisando cada nó até que a instância seja classificada ao chegar na folha. Feita a classificação, caso as instâncias analisadas no momento não sejam da mesma classe, é utilizada a função de ganho de informação $\bar{G}$ para cada uma dessas classes.

Dessas classes, duas características importantes são extraídas, o atributo com o maior ganho de informação $\bar{G}_{l1}$ (algoritmo 1 linha 21) é o primeiro, e o atributo com o segundo maior ganho de informação $\bar{G}_{l2}$ (algoritmo 1 linha 27) é o segundo. Em seguida, é realizado o cálculo de *Hoeffding bound* é executado. Se a diferença entre os resultados de $\bar{G}_{l1}$ e $\bar{G}_{l2}$ for menor que o novo *Hoeffding bound* calculado, é dividido o nó gerando duas novas folhas e os seus dados estatísticos são atualizados. Ao final da execução, é retornado o modelo com a árvore criada.

2.3.2 Online Bagging - Oza Bag (OB)

Online Bagging - Oza Bag (OB) é uma adaptação incremental do algoritmo original *Bagging*. Os autores Oza e Russell (2001) demonstraram como o processo de inicialização poderia ser adaptado para um processo de fluxo de dados utilizando o Poisson($\lambda = 1$) distribuído. Trata-se de uma distribuição de probabilidade discreta que descreve o número de eventos que ocorrem em um intervalo fixo de tempo ou espaço (CONSUL; JAIN, 1973). Em essência, em vez de modificar o valor original do treinamento, OB, o Poisson($\lambda = 1$) é usado para preencher os pesos a cada nova instância recebida, os pesos são entradas utilizadas para ajustar o modelo do classificador desta forma, possibilitando uma melhor predição dos dados. Estes pesos representam a quantidade de vezes que cada instância foi repetida durante a simulação de inicialização. A utilização desse algoritmo neste trabalho é essencial, pois é um algoritmo do tipo comitê de classificadores, que funciona de forma eficiente e com bons resultados em cenários com fluxo de dados.

Algoritmo 1 Pseudocódigo do *Hoefdding Tree*. Adaptado de (DOMINGOS; HULTEN, 2000)

Entrada:
S sequência de instâncias
X conjunto de atributos discretos
G(.) avaliação de ganho de informação
δ probabilidade mínima desejada para seleção de atributo do nó

Saída:
HT árvore de decisão gerada

HOEFFDINGTREE(S, X, G, δ)
Sendo HT uma árvore com uma única folha L_1 (raiz)
$X_1 = X \bigcup X_\emptyset$
Sendo $\bar{G}(X_\emptyset)$ o ganho de informação, $\bar{G}$ obtido a previsão da classe em S
Para cada classe y_k **Faça**
  Para cada classe x_{ij} de cada atributo $X_i \in X$ **Faça**
    $n_{ijk}(l_1) = 0$
Para cada exemplo $(x, y_k) \in S$ **Faça**
  Insira (X, y) em uma folha l usando HT
  Para cada (x, y_k) em $x|X_i \in X_i$ **Faça**
    Incremente $n_{ijk}(l)$
  Classifica l com classe majoritária entre os exemplos vistos em l
  Se observados até o momento em l não são todos da mesma classe **Então**
    Ganho de informação $\bar{G}_i(X_i)$ para cada atributo $X_i \in X_i - \{X_\emptyset\}$
    Sendo X_a o atributo com maior ganho de informação $\bar{G}_l$
    Sendo X_b o atributo com o segundo maior ganho de informação $\bar{G}_l$
    Hoeffding Bounds calculado por $\epsilon = \sqrt{\frac{R^2 ln(1/\delta}{2n}}$
    Se $\bar{G}_l(X_a) - \bar{G}_1(X_b) > \epsilon$ e $X_a \neq X_\emptyset$
      Atualize l por um nó interno dividido em X_a
      Para cada ramificação da divisão **Faça**
        Adicione uma nova folha l_m e então $X_m = X - X_a$
        $\bar{G}_m(X_\emptyset)$, portando $\bar{G}$ é resultado classe mais frequente em l_m
        Para cada y_k e cada $x_i j$ de cada $X \in X_m - X_\emptyset$
          $n_{ijk}(l_m) = 0$

Retorna HT

2.3.3 Leveraging Bagging (LBag)

Considerando a estrutura proposta em Oza e Russell (2001), o Bifet e Holmes (2010) propôs o LBag visando melhorar o trabalho de aleatoriedade nas instâncias. Com a técnica de re-amostragem do *Online Bagging* utilizando a distribuição do *Poisson*.

Os autores Bifet e Holmes (2010) propuseram aumentar os pesos das re-amostragens com um incremento no valor de λ para então fazer o cálculo da distribuição. Em uma segunda rodada de aprimoramento, a aleatoriedade é adicionada em um conjunto de saída usando estratégias para correção dos erros. Uma string binária de comprimento n é atribuída a cada classe, portanto, um novo conjunto de classificadores binários n são inicializados. Cada classificador é responsável por aprender um bit nessa *string* binária. No momento da chegada de uma nova instância x, ela então é atribuída a uma classe com o código binário mais próximo.

A motivação de utilizar um código aleatório contrário a um código determinístico é que cada membro do comitê irá prever uma função diferente, podendo reduzir assim os efeitos de correção entre os classificadores (BIFET; HOLMES, 2010). Posteriormente, a diversidade desses conjuntos é aumentada e os classificadores x e classe c são atribuídos em valores binários de uma forma uniforme, independente e aleatória. Metade exata das classes é mapeada para 0.

Portanto, a saída de um classificador em uma instância é a classe que possui mais votos em seu mapeamento binário de classes. Dessa forma, para lidar com a mudança de conceito, é utilizado o *Adaptive Windowing* (BIFET; GAVALDA, 2007) na implementação deste algoritmo.

2.3.4 Online Bagging with ADWIN (OBADWIN)

Neste trabalho, Oza e Russell (2001) implementam um algoritmo de *bagging* com o funcionamento online. Isso significa que o processamento é automático a partir de um novo dado, sem a necessidade de armazenamento para executar o re-processamento. Ainda segundo os autores, o algoritmo *Online Bagging* é capaz de melhorar a construção de novos algoritmos, pois utiliza a combinação da metodologia de duas técnicas de treino, o *OzaBag* com o *Adaptive Window*. A utilização dessa técnica implica que a média sobre os treinos executados pode ser considerada mais confiável, a partir da utilização da média atual do fluxo, a menos que uma alteração muito pequena ou recente ainda não seja visível no cálculo de estatísticas. Esse algoritmo representa um *trade-off* entre o custo computacional à medida que as árvores são incrementadas pelo detector de mudança. Isso implica que o OBADWIN terá um custo computacional maior, mas eventualmente o resultado preditivo é melhor, pois se beneficia com o detector de mudança.

2.3.5 Online Bagging with Adaptive Size Hoeffind Tree (OBASHT)

Uma quantidade significativa de estudos propôs melhorias para algoritmos de classificação *online* e o estudo Gomes et al. (2017) utiliza a técnica de Online Bagging - Oza Bag (OB) proposta inicialmente em Oza e Russell (2001) combinada com a árvore de tamanho adaptativo *Hoeffding Tree* (DOMINGOS; HULTEN, 2000). Como apresentado anteriormente, o estudo (DOMINGOS; HULTEN, 2000) implementa a estrutura de dados *Hoeffding Tree* baseado em árvores incrementais, que conseguem aprender através da quantidade massiva de dados.

No entanto, a partir do seu modelo tradicional, este algoritmo não está preparado para uma mudança de recebimento de dados, assumindo que eles chegam de forma constante. Além disso, o algoritmo é capaz de escolher um melhor atributo para a divisão da árvore com base em uma pequena amostra (BIFET; HOLMES; AL., 2009). O algoritmo OBASHT se destaca das implementações dos outros algoritmos por utilizar um número máximo de divisões. Quando o número total de divisões excede o máximo configurado anteriormente para a árvore, ele exclui os nós excedentes para manter o tamanho ideal.

Através da técnica de exclusão de nós redundantes, a estrutura de dados tende a ficar menor, permitindo uma adaptação mais rápida (BIFET et al., 2009). Quando maior essa estrutura, melhor será o desempenho durante períodos de processamento com pouca ou nenhuma execução. As árvores com o tamanho x são redefinidas cerca de duas vezes com mais frequência do que árvores com tamanho limite de $2x$. Isso cria velocidades diferentes para construção de conjuntos de árvores durante a reinicialização do treinamento. Um destaque importante no comportamento deste algoritmo é que as redefinições de árvores acontecerão o tempo todo, mesmo para aqueles conjuntos de dados estacionárias. No entanto, esse comportamento não tem destaque negativo no desempenho preditivo do algoritmo (BIFET; HOLMES; AL., 2009).

Podemos concluir que, quando uma árvore excede o tamanho máximo, os seguintes caminhos podem ser seguidos: os nós mais antigos e sobressalentes são deletados, incluindo a raiz e todos os seus filhos, exceto onde a divisão aconteceu, tornando o nó não excluído a nova raiz; todos os nós dessa árvore são deletados, iniciando uma nova árvore. A diversificação das árvores impacta positivamente o desempenho em comitês de classificadores (BIFET; GAVALDA, 2007).

2.3.6 Adaptive Random Forest (ARF)

Adaptive Random Forest (ARF) é uma adaptação do código original *Random Forest* (L, 2001) para classificação de um fluxo de dados. Random Forests (RF) usa a técnica *random subsets* que consiste em aumentar a diversidade entre os modelos básicos de classificação. A parte adaptável do ARF decorre da detecção de mudanças e estratégias de recuperação baseadas na detecção de alerta por árvore em um determinado conjunto.

Depois da sinalização dessa árvore, o conjunto cria e treina novamente um novo modelo, sem afetar as previsões do conjunto (L, 2001). Se esse alerta aumentar para um sinal de desvio, o comitê de classificação substitui a árvore associada pela nova árvore-base. No pior caso, o número de modelos em uma árvore no ARF pode ser no máximo o dobro do número de total de árvores devido às árvores-base. Contudo, como foi constatado em Gomes et al. (2017), a coexistência de uma árvore e sua árvore base são geralmente de curtos períodos durante a execução do algoritmo.

2.3.7 Streaming Random Patches (SRP)

Streaming Random Patches (SRP), de acordo com (GOMES; READ; BIFET, 2019), é um comitê de classificação especialmente adaptado para a classificação de fluxo de dados. Com a combinação do *Random Subspaces* e *Online Bagging*, SRP não é especificamente um *learner* como o *ARF* descrito anteriormente, mas inclui uma diversidade de mecanismos na base do *learner*. Em outras palavras, SRP usa a randomização global; em contrapartida, ARF utiliza a randomização local (GOMES; READ; BIFET, 2019). Os experimentos com foco no *Hoeffding tree* mostraram que o SRP consegue produzir árvores profundas mantendo o equilíbrio da diversidade de treino nos comitês de classificação com um ótimo custo computacional.

2.4 Massives Online Analysis (MOA)

Massives Online Analysis (MOA) é um *framework* que fornece implementações de algoritmos de classificação e execução em um fluxo de dados. MOA foi construído com o desafio de solucionar o problema de escalas nas implementações e no avanço do estado-da-arte quanto a execuções com *datasets* de tamanhos reais (BIFET; HOLMES, 2010).

De acordo com Bifet e Holmes (2010), esse *framework* fornece a possibilidade de evolução e adaptação dos seus algoritmos pela comunidade científica, por ser de código aberto. O código implementando no MOA propõe diversos tipos de algoritmos de classificação. Estes algoritmos são utilizados para pesquisa e avaliação, além de fornecerem classificação para um fluxo de dados implementado em cenários reais, que não estão sendo controlados para pesquisa.

MOA também fornece uma interface gráfica que diminui a curva de aprendizado a novos pesquisadores e entusiastas da área, além de uma documentação que fornece o passo a passo de como utilizar os principais algoritmos disponíveis em seu código. Segundo (BIFET; HOLMES, 2010), a implementação e disponibilização do MOA como um *framework* de classificação para fluxo de dados permitiu que o avanço do estado-da-arte se tornasse mais rápido devido a seu gerador de dados automático, beneficiando a possibilidade de comparação em relação aos dados de *datasets* reais com diversos algoritmos e cenários mais próximos da realidade.

2.5 Otimização Multi-Objetivo de Pareto

A otimização multiobjetivo *Multi-Objective Optimization Problem (MOOP)* tem um grupo de funções objetivos a serem otimizadas, onde se pode maximizar ou minimizar essa função. Assim, cada função precisa de regras para resolver o problema de forma viável. O desafio na otimização multiobjetivo é que a maioria desses objetivos são potencialmente conflitantes. Embora em problemas de otimização de objetivo único, a solução ótima seja geralmente facilmente definida, em multiobjetivo é mais difícil (ALHAMMADI; ROMAGNOLI, 2004).

A otimalidade de Pareto produz uma ferramenta para visualizar o trade-off entre os objetivos. Assim, muitos objetivos de função são mostrados, e uma decisão humana é necessária para expressar preferências entre as soluções alternativas e continuar do ponto em que as ferramentas matemáticas não podem nos apoiar (MIETTINEN, 1999).

As funções objetivo empregadas nos MOOP são em geral conflitantes entre si, uma função objetivo f_1 é conflitante com outra função f_2 quando não é possível melhorar o valor de f_1 sem a piora do valor da função de f_2 (ALHAMMADI; ROMAGNOLI, 2004).

Podemos ver um exemplo prático de objetivos conflitantes nos preços de automóveis. Os automóveis de menor consumo de combustível ou elétricos possuem um maior custo. Dessa forma, não há como escolher uma variável sem que a outra sofra consequência. Quando ambos os objetivos possuem a mesma importância, não é possível afirmar, por exemplo, que a redução do preço compensa na hora de escolher um carro que consuma mais combustível.

Em um MOOP, podemos empregar o conceito de dominância de Pareto para comparar duas soluções factuais de um problema. Dadas duas soluções x e y, dizemos que x domina a y. Assim, podemos dizer que existem, conforme o exemplo anterior, soluções melhores em menor consumo de combustível, mas não são melhores em custo e vice-versa. Portanto, podemos dizer que existem alternativas ótimas que são não-dominadas entre si nos objetivos de custo e consumo.

No campo da otimização, a otimalidade de Pareto é um conceito fundamental sobre as técnicas usadas, especialmente no contexto da otimização multiobjetivo. Este campo é considerado um processo matemático para procurar muitas alternativas que representem a solução ótima de Pareto. Em resumo, o ótimo de Pareto pode ser definido como um conjunto de soluções 'não inferiores' no espaço objetivo, onde nenhum dos objetivos pode ser melhorado sem sacrificar pelo menos um dos outros objetivos.

O conceito de otimização multi-objetivo, especificamente a otimalidade de Pareto, será utilizada neste trabalho a fim de otimizar o tamanho do *mini-batching*, com base em dois critérios, o tempo de processamento do tamanho do *mini-batching* (*delay*) e o consumo de energia.

2.6 Computação de Borda

Computação de borda, ou *Edge Computing*, do inglês, é o modelo de arquitetura utilizado para processamento de dados feito nas bordas da rede, ou seja, mais próximo da fonte de informação. Esse tipo de arquitetura destaca o rápido processamento e os modelos agnósticos de hardware que podem ser utilizados, tornando o processo mais diverso e também com maiores opções de poder de processamento e consumo energético (CAO et al., 2020).

É um novo modelo de computação que implementa recursos de computação e armazenamento (como *cloudlets*, micro data centers ou nós de computação em névoa) na borda da rede mais próxima de dispositivos móveis ou sensores (SATYANARAYANAN, 2017). Essa arquitetura computacional tem a capacidade de processar dados localmente, sem depender necessariamente de um serviço externo na nuvem, o que resulta em menor latência no processamento.

A computação de borda consegue reduzir os custos de rede de computação, evitar limites na largura de banda, diminuir os atrasos no envio e recebimento de informações e reduzir as falhas de serviços. Ao processar dados localmente, a computação de borda evita a necessidade de transferir grandes volumes de dados para *data centers* distantes, o que alivia a largura de banda da rede. Essa proximidade ao ponto de coleta de dados também diminui significativamente a latência, resultando em menores atrasos no envio e recebimento de informações. Além disso, ao operar de maneira distribuída e local, a arquitetura de borda reduz a dependência de uma única fonte central, diminuindo assim a vulnerabilidade a falhas de serviço. Essa arquitetura também oferece alta segurança e privacidade de dados, pois evita o envio de dados por redes abertas e públicas. Além disso, permite a análise de *big data* e a classificação de dados de maneira local, possibilitando a tomada imediata de decisões (SATYANARAYANAN, 2017). No entanto, a computação de borda geralmente oferece um poder computacional e capacidade de armazenamento limitados, já que esses dispositivos são, em sua maioria, portáteis (HAMDAN; AYYASH; ALMAJALI, 2020).

Conforme visto nas seções 2.2 e 2.1, a classificação de dados é uma forma de poder predizer cenários padrões a partir de um histórico anterior de dados. Porém, sua utilização em nuvem traz maior consumo energético e maior tempo de processamento, quando consideramos o atraso da rede e da classificação do dado. O desafio de utilizar pouco poder computacional para o processamento de dados pode ser resolvido com a combinação de algumas estratégias de melhoria no desempenho computacional e também em redução energética.

2.7 Estratégias de hardware para reduzir o consumo energético

Esta seção apresenta uma breve descrição das principais estratégias implementadas em hardware para equilibrar o desempenho computacional e o consumo energético. Processadores modernos implementam estratégias para economia de energia. Vários componentes que não estão em uso são automaticamente desligados para economizar energia. Além disso, vários processadores permitem reduzir ou aumentar a frequência do *clock* por software. A relação entre a frequência de *clock* e a potência consumida é quadrática: se dobramos a frequência do *clock*, o consumo tende a aumentar quatro. A seguir apresentaremos um resumo dessas técnicas.

2.7.1 Dynamic Voltage and Frequency Scalling (DVFS)

A técnica DVFS é uma técnica moderna utilizada para redução do consumo energético e também o controle de temperatura dos processadores (RIZVANDI et al., 2012). De acordo com (RIZVANDI et al., 2012), ela é empregada em dispositivos como notebooks e celulares para melhorar o desempenho da bateria. A utilização dessa estratégia também é comum em nós de computação de alto desempenho, visando esfriar e reduzir o consumo de energia (RIZVANDI et al., 2012).

De acordo com pesquisas realizadas por Feng, Ge e Cameron (2005), utilizando o *benchmark* SPEC CPU 2000, o consumo de energia da CPU de um nó durante uma execução é de aproximadamente 35% do total da energia consumida por esse nó. Porém, quando está em modo ocioso, esse consumo se reduz apenas para 15%. Os outros consumos são distribuídos entre memória, *coolers*, fonte de alimentação, interface de rede, disco e *chipset*.

A estratégia de hardware suportada por microprocessadores modernos é o Voltage Frequency Scalling (VFS), que permite ao usuário escolher entre algumas frequências de *clock* e tensões nas quais o microprocessador pode operar. Por exemplo, a frequência do *clock* pode ser ajustada usando *cpufreq-utils* (WIKI, 2023). Espera-se que o consumo de energia cresça em proporção quadrática com o aumento da frequência do *clock*, enquanto o desempenho da aplicação deverá aumentar linearmente. O objetivo é avaliar se essa técnica pode melhorar a eficiência energética, principalmente sob condições de baixa intensidade.

2.7.2 Dynamic Power Management (DPM)

O gerenciamento dinâmico de energia, do inglês, Dynamic Power Management (DPM) pode ser aplicado programaticamente para economizar energia (COUTINHO et al., 2020). O DPM abrange muitas técnicas para economizar energia e potência dinâmica. A maioria

dos microprocessadores hoje em dia pode desligar automaticamente componentes inativos para esse fim.

Se não houver instruções a serem executadas em um núcleo na CPU, este núcleo pode ser desabilitado (BENINI; BOGLIOLO; MICHELI, 2000). Outras técnicas incluem colocar memória DRAM e componentes de armazenamento em modos de baixo consumo de energia quando os dispositivos estevirem ociosos.

Por exemplo, as DRAMs têm uma série de modos de consumo de energia cada vez mais baixos para prolongar a vida útil da bateria em PMDs e *laptops*. Também têm havido propostas para discos que possuem um modo que gira mais lentamente quando não são estão em uso, para economizar energia.

Embora as estratégias DPM atuem de forma autônoma, estratégias de otimização podem ser empregadas para reduzir o número de núcleos ativos ou reduzir explicitamente a frequência do *clock* e economizar energia (COUTINHO et al., 2020).

Core Limiting (CL) é uma estratégia que limita a execução do aplicativo a um subconjunto dos núcleos de processamento disponíveis. Por exemplo, um aplicativo pode ser fixado em apenas dois dos quatro núcleos de um processador, enquanto os dois núcleos restantes são mantidos ociosos (acionando automaticamente mecanismos DPM) para reduzir o consumo de energia.

Essas configurações são aplicadas em níveis arquiteturais. No trabalho Benini, Bogliolo e Micheli (2000) são utilizados três exemplos de configurações possíveis para o gerenciamento de energia. Neste trabalho, iremos utilizar o pacote do Linux chamado *cset* (TSARIOUNOV, 2011). Com ele, é possível indicar ao sistema quais cores estarão disponíveis no hardware e dessa forma não teremos interferências na coleta de dados.

2.8 A estratégia mini-batching

Em essência, as técnicas de processamento em lote consistem em agrupar instâncias de um conjunto de dados e processá-las juntas. A técnica pode ser aplicada em diferentes contextos, para diferentes finalidades.

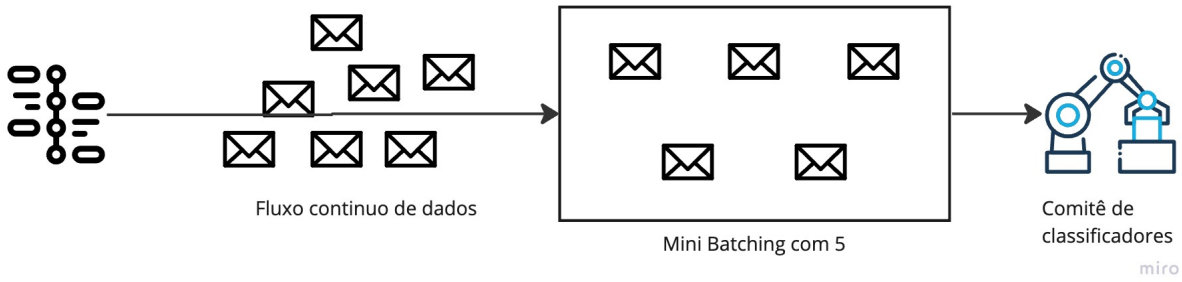
Mini-batching são comumente usados em *deep learning*, onde os dados de treinamento são divididos em subconjuntos menores chamados *mini-batching*. Ao usar lotes em vez de processar uma instância de dados por vez, o processo de treinamento se torna mais eficiente e com melhor reuso do *cache* da memória. No *Spark*, as instâncias de dados são agrupadas em pequenos lotes para recuperação eficiente de falhas (ZAHARIA et al., 2012). No entanto, o *mini-batching* estudado neste artigo difere dos cenários anteriores.

O *mini-batching* aqui se refere a uma otimização específica para melhorar a localidade dos dados em comitês de classificação, conforme definido em (CASSALES; GOMES H, 2021). A estratégia de *mini-batching* agrupa várias instâncias de dados de um fluxo em pequenos pedaços (chamados de *mini-batching*) para serem processados por cada *learner*

do conjunto. Cada *learner* é executado por uma tarefa paralela que itera em todas as instâncias de um *mini-batching*. Quando um *learner* é invocado, suas estruturas de dados são carregadas nos caches para processar a primeira instância de dados do *mini-batching* e reutilizadas para o processamento das próximas instâncias, ao contrário do processamento de uma única instância por vez, como no algoritmo sem a técnica do *mini-batching*).

Assim, o *mini-batching* reduz as perdas de cache e melhora o desempenho. Conforme comprovado em Cassales e Gomes H (2021), o *mini-batching* pode reduzir significativamente a métrica *reuse distance*, o que reduz o número de perdas de cache e o número de ciclos do processador, reduzindo assim o tempo de execução e a energia consumida para processar o fluxo de dados (Cassales et al., 2022). Neste trabalho iremos focar nesta técnica. A Figura 3 ilustra o funcionamento do *mini-batching*.

Figura 3 – Exemplo de funcionamento da técnica de mini-batching



Fonte: O Autor

Enquanto isso, podemos ver o Algoritmo 2 detalhando a técnica implementada em pseudocódigo. O algoritmo funciona da seguinte forma: na linha 3 o laço que itera sobre as instâncias é iniciado, então cada instância é adicionada em uma lista *B*. Ao chegar no limite esperado do tamanho para o *mini-batching*, se inicia o processamento de classificação dessas instâncias. Na linha 6, todo esse *mini-batching* é classificado e, nas linhas 7 a 9, cada *trainer* do comitê de classificadores (*ensemble*) recebe o resultado dessa classificação. As linhas 10 a 19 fazem o cálculo do novo peso para o treino com as próximas instâncias. Ambos os loops de repetição da linha 7 e da linha 10 foram construídos para serem executados em paralelo.

Esses trabalhos trazem uma perspectiva do *mini-batching* de tamanhos fixos, ou seja, pré-configurados durante a inicialização do programa responsável pelo processamento desses dados. Algumas outras implementações sugerem e estudam a possibilidade da utilização de *batchs* dinâmicos que possam se adaptar de acordo com algumas variáveis de processamento, como consumo de memória, uso da CPU, tempo de atraso entre o processamento de um *batch* e outro, entre outras variáveis. Os trabalhos de Liu et al. (2019),

Algoritmo 2 Implementação do mini-batching com paralelismo

```
1: Entrada: um ensemble E, numThreads, o fluxo de dados S, tamanho mini-batching
   L_mb
2: P = CreateServiceThreadPool(numThreads)
3: T = CreateTrainersColletions(E)
4: for para cada instancia I em um fluxo S do
5:    B.append(I)
6:    if B.size() == L_mb then
7:       E.classify(B)
8:       for cada trainer T em trainers faça em paralelo T do
9:          T_i.instances.append(B)
10:      end for
11:      for cada trainer T em trainers faça em paralelo T do
12:         for cada instancia I em B do
13:            W ← poisson(λ)
14:            W_inst ← I * k
15:            train_on_instance(W_inst)
16:         end for
17:         if change detected then
18:            B.clear()
19:         end if
20:      end for
21:   end if
22: end for
```

Das et al. (2014), Tyagi e Sharma (2020), Cheng et al. (2018) são alguns exemplos que fizeram essa investigação.

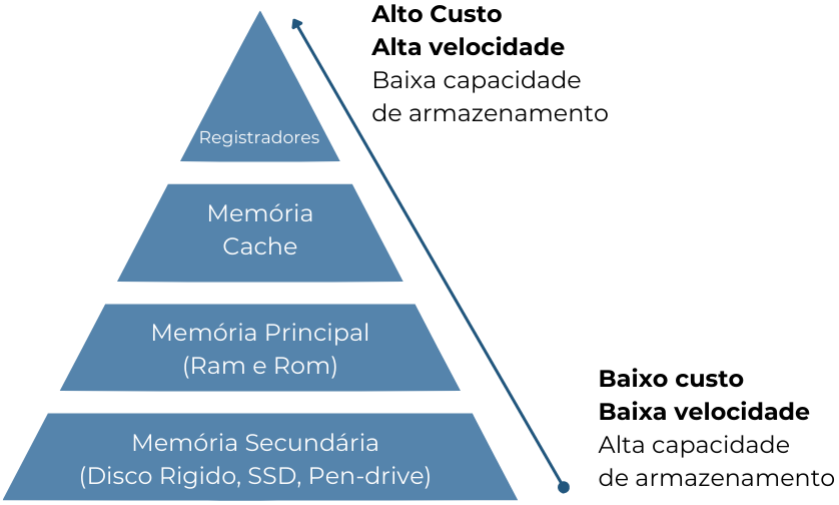
O trabalho de Cheng et al. (2018) pesquisa sobre a utilização de *micro-batchs* para o processamento de classificação de um fluxo de dados dinâmico utilizando a ferramenta *Spark*. São utilizadas, principalmente, as variáveis de taxa de saída e latência, as quais são investigadas para trazer o melhor desempenho quando utilizado o *micro-batch* de tamanho dinâmico. Esse tamanho é formado pela construção de um algoritmo de controle utilizado a partir da lógica *fuzzy*, uma estrutura construída onde possa ser suportado o recebimento destes dados e a escolha dinâmica do tamanho do *micro-batch*. A utilização de um algoritmo *fuzzy* especializado permitiu que o artigo calculasse o tamanho do *batch*. O artigo (DAS et al., 2014) propôs a metodologia para processar fluxos de dados com tamanho de lote dinâmico. Eles propõem um algoritmo de controle simples, porém robusto, que se adapta automaticamente aos tamanhos de lote conforme a situação exige. Os pesquisadores analisam o comportamento de alguns fluxos de carga de trabalho visando identificar um tamanho de lote melhor com base na latência e no *throughput*. Algumas variáveis são definidas para criar um algoritmo que modifica o tamanho do lote, como atraso de agrupamento, atraso de enfileiramento e tempo de processamento. Eles definem a latência como: a latência de ponta a ponta de uma carga de trabalho em streaming como a duração entre o momento em que um registro de dados entra no sistema e o momento

em que os resultados correspondentes a esse registro são gerados.

2.9 Hierarquia de memória

O conjunto de memórias é projetado hierarquicamente. Conforme podemos ver, na Figura 4 é apresentada uma sequência contendo a hierarquia de memórias existentes em um computador. O nível mais baixo possui uma maior capacidade de armazenamento, por ser mais barata, mas possui o maior tempo de acesso, enquanto o nível mais alto possui menor capacidade de armazenamento, porém maior velocidade de acesso.

Figura 4 – Conceito de memórias em 4 níveis, adaptado de (ZHANG et al., 2015)



Fonte: O Autor

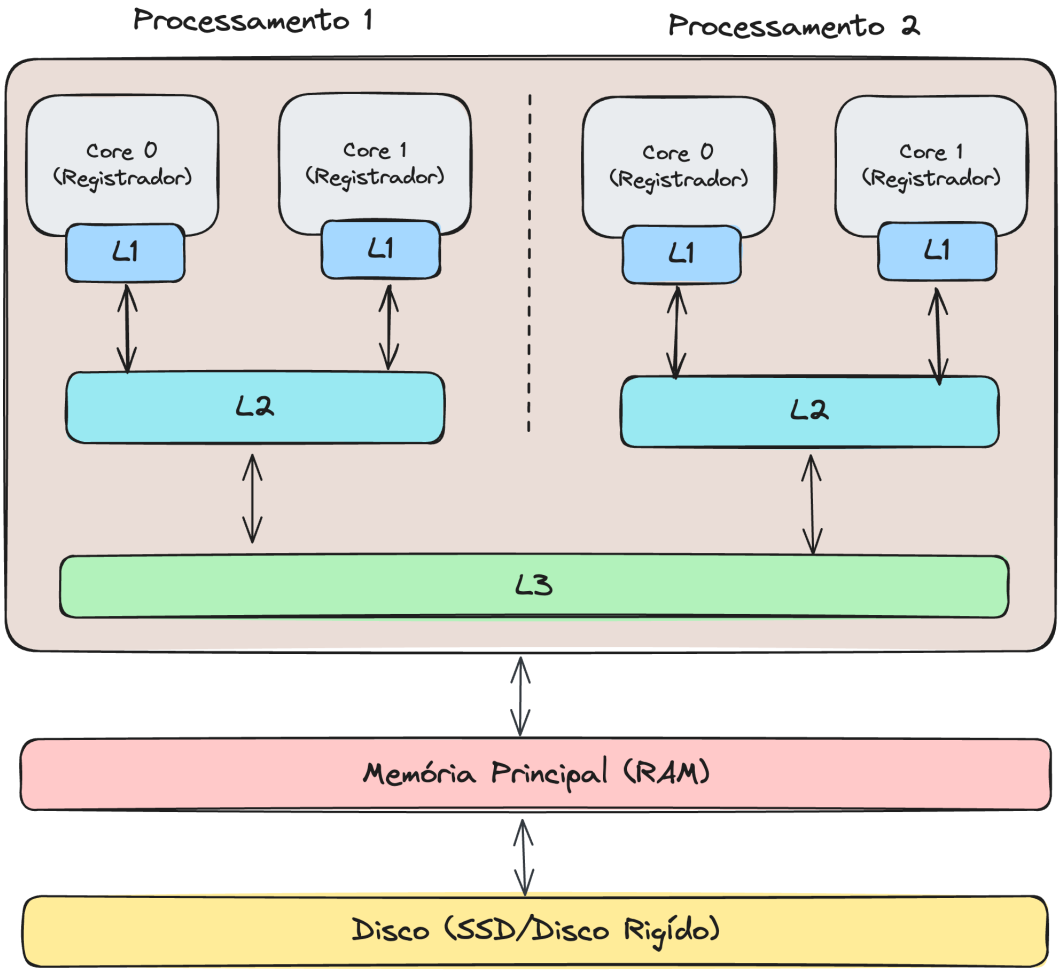
Na medida que avançamos na hierarquia de memória, três parâmetros aumentam. No primeiro, o tempo de acesso fica maior. Logo, os registradores da CPU podem ser acessados em um nanossegundo ou menos. As memórias cache demoram um pequeno múltiplo dos registrados da CPU. Acessos à memória principal normalmente levam 10 nanossegundos. Agora, vem um grande salto, pois o tempo de acesso a disco pode ser no mínimo 10 vezes mais lento para discos em estado sólido (TANENBAUM, 2009).

Os dados de um programa usados com mais frequência são mantidas no cache. Quando a CPU necessita de um dado específico, a arquitetura primeiro verifica as referências no cache para buscar esse dado. Somente se esse dado não estiver disponível, o processador vai buscar esse dado na memória principal. Assim, se uma porção desses dados estiverem disponíveis no cache, o tempo médio de acesso pode ser muito menor.

Essa hierarquia da memória é mostrada em termos de latência de acesso, como já vimos nas definições anteriores, cache (L1, L2 e L3), memória principal e disco (disco

rígido, memória flash e SSD). A Figura 5 ilustra o acesso à memória com os níveis de cache e de memória principal e secundária (ZHANG et al., 2015).

Figura 5 – Tipicamente, a hierarquia de memória contém dois chips de *cache*, o L1 único por core, o L2 que é compartilhado entre os processos, o terceiro nível, L3, compartilhado em todos as cores e, por último, a memória principal e o disco do computador.



Fonte: O Autor

O acesso à memória é dividida em níveis com objetivo de o acesso ser mais otimizado:

1. O nível mais próximo da CPU são os registradores: encontrados diretamente no processador, são a menor memória dentro desta arquitetura.

2. O primeiro nível, *cache* L1 também encontrado diretamente no processador, é a menor memória existente dentro desta arquitetura, com um tempo de acesso médio de 1 nanosegundo (ZHANG et al., 2015).

3. O segundo nível, *cache* L2: pode ser tanto encontrado tanto dentro no processador quanto fora dele. Foi projetado com o objetivo de aumentar o pouco armazenamento

existente na L1, com um tempo de acesso médio de 3 nanosegundos (ZHANG et al., 2015).

4. O *cache* L3 é compartilhado entre todos os núcleos. Este *cache* é maior em comparação com as caches L1 e L2 e é compartilhada entre os núcleos para permitir a comunicação eficiente entre eles, com um tempo de acesso médio de 10 nanosegundos (ZHANG et al., 2015).

5. Memória principal Random Access Memory (RAM): possui maior capacidade de armazenamento e um tempo de acesso médio de 15 a 90 nanosegundos (ZHANG et al., 2015).

6. Disco: possui a maior capacidade de armazenamento e um tempo de acesso médio de 100K de nanosegundos (ZHANG et al., 2015).

2.10 Localidade de dados

Localidade de dados ou principio da localidade é uma propriedade fundamental para compreendemos a otimização do desempenho do *hardware*, do *software* e de algoritmos (YUAN et al., 2019). Ainda segundo este autor, a localidade pode ser definida de várias maneiras, e diversas métricas relacionadas a ela foram propostas (YUAN et al., 2019). Outra definição possível é "a tendência dos programas agruparem referências de memórias em subconjuntos em um espaço de memória por longos períodos" (DENNING, 2003). Isso aumenta a velocidade entre o processamento do processador e memória (JACOB; WANG; NG, 2010). As localidades desempenharam um papel central para a otimização do desempenho dos sistemas operacionais ao longo das décadas.

De acordo com Stallings (2024) principio reflete na observação de que, no decorrer da execução de um programa, as referências a memória feitas pelo processador, tanto para instruções quanto para dados. Então os programas geralmente contém diversas sub-rotinas e laços interativas. Conforme visto na seção 2.9 o acesso à memória é recuperado em unidades de tamanhos diferentes variando de palavras individuais e grandes blocos de memória cache e segmentos muitos maiores de memória em disco.

Ainda segundo o (STALLINGS, 2024), podemos isolar a localidade de dados em duas formas básicas: a localidade temporal e a localidade espacial, onde a temporal refere-se a tendência de um programar em referenciar em breve aquelas unidades de memória referencias no passado recente. Por exemplo, quando um laço é executado, o processador executa o mesmo conjunto de instruções repetidamente. Constantes e variáveis e pilhas de trabalhado também são construções que levam esse principio. Enquanto a espacial, é a tendência de um programa referenciar unidades de memória cujos endereços estão próximos uns dos outros. Por exemplo, se uma unidade de memória x for referenciada no tempo t, é provável que as unidades de intervalo $x - k$ a $x + k$ sejam referenciadas em um

futuro próximo, para um valor relativamente pequeno de k. Isso reflete a tendência de acessar as instruções sequenciais de um programa. A localização espacial também reflete na tendência de um programa acessar localizações de dados sequencialmente, como ao processar a tabela de dados.

O uso do Reuse Distance (RD) foi utilizado para avaliar propostas como a do *mini-batching* para melhorar o desempenho e usos de recursos, por exemplo, o cache em compartilhamento de dados entre as implementações nos comitês de classificadores (CASSALES; GOMES H, 2021).

De uma perspectiva histórica, a localidade da memória tem sido estudada ao longo de décadas para otimizar a hierarquia da memória, sistemas operacionais, software e design de algoritmos, com avanços recentes em técnicas de medição (IBRAHIM; STROHMAIER, 2010)„ geração de rastreamento (SHEN; SHAW, 2008) e modelagem formal. No trabalho recente do Yuan et al. (2019) definiram a Relation Theory Of Locality (RTL) como um framework teórico para unificar as diversas medidas de memórias usado ao longo de cinco décadas de estudo e pesquisa. RTL disponibiliza um histórico matemático e categoriza as medidas em três diferentes tipos de localidades.

De acordo com Cassales e Gomes H (2021) podemos implementar a melhoria de localidade de dados de um processamento em um sistema fluxo de dados conforme o algoritmo 2. Cada algoritmo implementa um comitê de classificação composto por um conjunto de $l_i \in \varepsilon$. Nós os referenciamos por um *learner* individual que pode ser descrito como $L_i(1 \leq i \leq \varepsilon)$. Um fluxo de dados S é um conjunto potencialmente infinito de contáveis elementos $s \in S$. Cada elemento do fluxo $s :< v, t >$ consistem da relação da tupla v. Foi pressuposto que o tempo T é discreto, linearmente ordenado. Como o fluxo de dados é potencialmente infinito, foi assumido que o T é limitado no passado, mas não necessariamente no futuro. Assim, devido a limitações de memória e restrições de tempo de resposta, os algoritmos precisam processar incrementalmente elementos de dados recebidos em uma única passagem, realizando classificação e treinamento conforme os elementos de dados chegam.

Podemos ver N como uma sequência de referências de dados a memória demonstrado por $N = mt(1, ..., n)$, onde N é o tamanho dessa sequência de dados. Essa sequência pode ser acessada pelo conjunto de memórias distintas em m, esse conjunto de memória distintas pode ser demonstrado por $M = e_1...e_m$ onde m é o número de endereços de memória distintos em M. Esse modelo permite abstrair de quaisquer problemas de granularidade para que um item de dados possa ser uma variável, como um bloco de dados, uma página ou um objeto. Para ilustrar, podemos usar alguns exemplos de rastreamento compostos de apenas três elementos de dados $a; b; c$, incluindo aqueles que se repetem uma vez na mesma ordem (ou seja, abc, abc), na ordem oposta (ou seja, abc, cba), ou repetindo-os indefinidamente (ou seja, abc, abc,...) (CASSALES; GOMES H, 2021).

Em essência, a localidade de dados é relacionada para mensurar a localidade de cada

acesso à memória, segundo as cinco definições de localidade de acesso definidas por Yuan et al. (2019). Neste trabalho, utilizamos a definição da sequência de distância de reuso (*reuse distance sequence*) ou simplesmente RD (YUAN et al., 2019).

Segundo o trabalho recente do mesmo autor, podemos definir RD como "O número de dados distintos acessados desde o último acesso ao mesmo *datum*, incluindo o *datum* reutilizado". O RD é iniciado como ∞ em seu primeiro acesso. Para infinito, a distância de reuso é 1, pois é considerado o *datum* reutilizado e o máximo é m. Considere o exemplo, o RD da sequência é $\infty\infty\infty$ 333 para *abc abc* e $\infty\infty\infty$ 135 para *abc cba*.

Podemos utilizar o RD para analisar o comportamento de *caches*, permitindo avaliar a eficiência do uso da memória *cache* e identificar padrões de acesso que possam ser otimizados para melhorar o desempenho do sistema, através da implementação de técnicas de transformação de laços (BEYLS; D'HOLLANDER, 2001).

2.11 Considerações Finais

Neste capítulo, foram abordados os conceitos necessários para o entendimento deste trabalho. Primeiramente foi apresentada uma visão geral sobre o processamento em fluxo de dados e os algoritmos de classificação. Em seguida, explicamos o conceito sobre o framework utilizado para a classificação de dados, a conceituação geral da otimização multi-objetivo com Pareto e a experimentação dos algoritmos utilizados neste trabalho e então falamos sobre a arquitetura de computadores de computação em borda (*edge computing*). Então explanamos sobre as técnicas já utilizadas para melhoria do consumo energético em hardware e também em software especificamente para classificador de dados. Por fim, sumarizamos o conceito de memória e a localidade de dados. No próximo capítulo iremos discutir sobre os trabalhos relacionados utilizados como base para a elaboração desta pesquisa e as diferenças com o que foi proposto neste trabalho.

Capítulo 3

Trabalhos relacionados

Neste capítulo, apresentaremos os principais trabalhos relacionados a esta pesquisa. O objetivo é fornecer uma visão geral do estado da arte em estratégias para reduzir o consumo energético de comitês de classificadores em fluxos contínuos de dados. Na Seção 3.1, analisaremos os trabalhos que buscaram otimizar o consumo energético e processamento utilizando estratégias de *batching* no fluxo de dados. Na Seção 3.2, avaliaremos os trabalhos que utilizaram a otimização por meio das transformações de laços. Por fim, na seção 3.3, discutiremos os trabalhos que utilizaram técnicas de hardware para encontrar um equilíbrio entre desempenho e eficiência energética.

3.1 Técnicas para melhoria de desempenho e consumo energético em processamento de dados

O processamento em fluxo de dados é um desafio cada vez mais relevante em diversas áreas, desde a análise de dados de sensores em aplicações industriais até o monitoramento de redes sociais. Para lidar com esse desafio, têm sido propostas diferentes abordagens, como o uso de técnicas de *micro-batching* (ZAHARIA et al., 2012) (CARBONE et al., 2015) e *mini-batching* (CASSALES; GOMES H, 2021). Nesta seção, apresentaremos os principais trabalhos relacionados que abordam essas técnicas e suas contribuições para o processamento distribuído para o fluxo de dados.

O trabalho desenvolvido por (ZAHARIA et al., 2012) propôs a técnica denominada *Discretized Stream (D-Streams)*, que oferece uma aplicação funcional com eficiência na recuperação de falhas. A ideia central dessa técnica consiste em processar os dados em pequenos lotes, por meio de intervalos de tempo, melhorando o fluxo do processamento.

Para implementá-la, foi necessário solucionar dois desafios: a redução da latência (granularidade dos intervalos) e a rápida recuperação em caso de falhas, estas resolvidas através da implementação da técnica (*D-Streams*). Os resultados obtidos demonstraram que a utilização dessa técnica de processamento por lotes trouxe benefícios significativos na consistência dos dados processados e na recuperação de falhas.

Já o trabalho Carbone et al. (2015) disponibilizou um sistema de processamento de dados distribuído que consegue lidar com fluxos de dados em tempo real e conjuntos de dados em *batch*, o *Apache Flink*. O objetivo do sistema é muito similar ao *D-Streams* e visa melhorar a tolerância a falhas em fluxos de dados. Ao contrário de outras ferramentas que usam o *micro-batching* para processamento, o *Apache Flink* divide o processamento de dados em tarefas menores e as distribui em vários nós de um cluster. Isso permite que o *Flink* processe grandes volumes de dados de forma eficiente e confiável em ambientes de computação distribuída.

Os trabalhos citados propõem o processamento de um fluxo de dados em forma de *micro-batching* visando melhorar a confiabilidade e a tolerância a falhas durante o processamento. Contudo, esses trabalhos introduzem essa técnica ao nível de ferramenta, e não de aplicação, sem analisar o efeito do consumo energético no processamento dos dados. Em (Cassales et al., 2022) e (CASSALES; GOMES H, 2021) foi possível investigar ambas as questões.

O trabalho (CASSALES et al., 2020) propôs o *mini-batching* juntamente com o paralelismo, utilizando o processamento em pequenos lotes pré-configurados. Isso resultou em uma melhoria no acesso à memória *cache*, o que, por sua vez, acelerou significativamente o processamento conforme demonstrado no trabalho seguinte (CASSALES; GOMES H, 2021), que provou que o *mini-batching* oferece um escalonamento ótimo do acesso aos dados do cache.

Posteriormente, em (Cassales et al., 2022) os autores investigaram os benefícios do *mini-batching* na melhoria da eficiência energética, demonstrando que houve redução significativa do consumo energético. Em Cassales e Gomes H (2021) e Cassales et al. (2022) observaram-se perdas na qualidade preditiva devido ao atraso na votação dos *ensembles* em várias instâncias do processamento, resultando em uma leve diminuição no desempenho preditivo.

Outros trabalhos como Witten e Frank (2002) e Goodfellow, Bengio e Courville (2016) utilizaram a técnica do *mini-batching* em métodos baseados em descida de gradiente para otimizar o processamento do *deep learning*. Também é possível identificar trabalhos relacionados com o *mini-batching* para resolução de problema de inversão, como o trabalho de Kukreja et al. (2020) que investigou a possibilidade de utilizar o *mini-batching* para resolver o problema de inversão de onda completa, do inglês Full Waveform Inversion (FWI). O estudo mostra que o método proposto consegue reduzir significativamente a quantidade de memória necessária para executar o FWI, sem comprometer significativamente a

qualidade da solução.

Em geral, os trabalhos que implementam a técnica de *mini-batching* buscam melhorar o desempenho computacional. Os trabalhos relacionados a esta pesquisa buscam essa otimização sem investigar qual a relação que ela tem com a economia energética. O estudo que investigou com maiores detalhes essa relação foi o de (Cassales et al., 2022), porém, ainda assim, não considerou possibilidades da fusão dos laços de treinamento e de classifição para melhorar o acesso a memórias caches, na próxima seção iremos analisar alguns trabalhos relacionados que buscaram melhorar esse acesso através destas transformações.

3.2 Melhoria no acesso a memórias caches por meio de transformações de laços

Diferentes técnicas são empregadas buscando um melhor uso das memórias *caches* de um processador, buscando assim, melhor desempenho e redução no consumo energético. Dentre dessas técnicas, a transformação de laço tem um grande referencial teórico na literatura e com trabalhos feitos em que remetem aos anos 80, como Kuck et al. (1981). O objetivo do trabalho era buscar a otimização de desempenho reduzindo o tempo de execução desses laços. Recentemente, outros trabalhos utilizaram essa técnica visando a redução do consumo energético, como mostram os estudos de Ştirb e Ciocârlie (2016) e Pallister, Hollis e Bennett (2015).

As técnicas de transformação de laços nos permitem buscar soluções para determinar a melhor sequência de acesso de dados na memória de um computado. Diferentes técnicas de transformações são conhecidas, *loop fusion*, *loop unrolling*, *loop tilling* entre outros. Cada técnica nos fornece uma ferramenta para melhorar o acesso às memórias caches do computador, permitindo uma melhoria de desempenho e consumo energético. Ao longo desta seção iremos discutir como cada técnica foi empregada e quais foram os seus ganhos.

O trabalho Kuck et al. (1981) utiliza a análise de dependência de grafos para otimizar a execução de instruções de algoritmos e melhorar o acesso à memória, visando aprimorar o desempenho dos programas. Esse trabalho propõe técnicas para analisar a dependência de dados entre as instruções e, a partir dessa informação, otimizar a execução do código. Entre as técnicas propostas estão a distribuição de laços e a aplicação de transformações de laços, utilizadas em alguns casos específicos para otimizar o código. É importante ressaltar, entretanto, que o trabalho discute outras estratégias para lidar com dependências. Embora a otimização de código possa impactar no consumo energético de um sistema, o trabalho de Kuck et al. (1981) não tem como foco direto a redução de consumo de energia. Em relação aos resultados experimentais, o trabalho apresenta testes que avaliam a eficácia das técnicas propostas, tais como a redução do número de operações de leitura e escrita na memória e o tempo de execução dos programas. Em resumo, os autores pro-

puseram uma abordagem valiosa para análise e otimização de código, utilizando grafos de dependência para melhorar o desempenho de programas. As técnicas propostas, como a distribuição de laços e transformações de laços, podem ser utilizadas em alguns casos específicos, mas o trabalho apresenta diversas outras estratégias e técnicas para lidar com dependências de dados.

Em Pallister, Hollis e Bennett (2015) investigaram o consumo energético do código compilado sem otimizações de transformações de laços. O objetivo desse trabalho foi aplicar algumas técnicas de otimização de energia e desempenho na compilação de códigos utilizando a biblioteca GNU Compiler Collection (GCC) para reduzir o consumo de energia e manter o desempenho das aplicações. A partir das configurações de otimizações do compilador GCC, é possível analisar quais são as melhores opções, dado o grande número de opções possíveis. Os resultados demonstraram que mesclar algumas dessas configurações não surtiu maiores efeitos quando utilizadas isoladamente. Por esse motivo, foram realizados *benchmarkings* com várias dessas configurações para encontrar o melhor resultado em relação ao processamento e consumo energético. O trabalho demonstrou que não existe uma configuração universal de otimização energética e desempenho que sirva para todos os tipos de *benchmarks*. O tempo e a energia para computação de um determinado algoritmo são proporcionais, e configurações feitas ao nível de compilação tendem a reduzir o consumo energético de um algoritmo.

O artigo Ştirb e Ciocârlie (2016) apresenta como a transformação de laços pode ser utilizada para otimizar o consumo energético de programas compilados no Low Level Virtual Machine (LLVM). O estudo destaca a importância de se garantir que os laços possuam índices apropriados para que a transformação seja realizada corretamente pelo compilador. Esse trabalho conclui que essa técnica não só melhora o desempenho de processamento, mas também a eficiência energética. Os resultados demonstram que a técnica de transformação por *loop fusion* pode reduzir o consumo energético quando comparada à versão sem a transformação.

O artigo de Pallister, Hollis e Bennett (2015) são mostradas algumas técnicas para essas transformações de laços, tais como, *loop unrolling* e também o *loop fusion*, tais técnicas reduzem o número de instruções executadas para diminuir tempo de execução, desta forma, o algoritmo se torna mais eficiente tanto em processamento quanto em consumo energia, apesar de mostrar essas técnicas de otimização o artigo se limita em mostrar os resultados apenas para as configurações de otimização do próprio compilador GCC.

No trabalho de Brandolese et al. (2002), foram analisadas outras técnicas de otimização, como loop unrolling, *loop interchange* e *loop tiling*. Essas técnicas buscam melhorar o desempenho e o consumo energético através da melhoria no acesso à memória, da redução na dependência de dados e do aumento na velocidade de processamento.

Esses estudos são importantes para otimizar o desempenho e a eficiência energética em programas compilados no LLVM. No entanto, é necessário validar essas técnicas em

contextos específicos, como a classificação de fluxo de dados.

3.3 Otimização de desempenho e consumo energético com configurações específicas de hardware

O trabalho de Snowdon, Ruocco e Heiser (2005) demonstrou que a redução da tensão do processador nem sempre reduz o consumo de energia. Esse estudo utilizou diversos *benchmarks* para validar a hipótese com diferentes voltagens disponíveis no hardware do experimento. O trabalho mostrou que as voltagens menores consomem mais energia com um desempenho computacional pior do que as voltagens maiores.

O estudo de Coutinho et al. (2020) propõe um modelo estatístico que experimenta diferentes cenários de consumo, encontrando um equilíbrio entre desempenho computacional e energético. Para isso, o estudo utiliza aplicações com paralelismo para medir a eficiência dos resultados. As diversas combinações neste trabalho possibilitam a construção de um modelo eficiente e independente de hardware, que pode ser utilizado em qualquer dispositivo.

Para encontrar esse modelo, é utilizado o princípio de Pareto como fórmula estatística, considerando as diversas variáveis encontradas durante o experimento. Coutinho et al. (2020) sugerem buscar o equilíbrio entre energia e desempenho computacional considerando diversas variáveis. A partir da utilização de *benchmarks* e configurações disponíveis nos processadores e no hardware utilizado para pesquisa, foram analisadas 6384 diferentes combinações. O trabalho encontrou um ponto de equilíbrio em que a energia economizada foi de cerca de 59.77%, 61.38%, e 17.7% quando utilizado com os governadores de frequência disponíveis no sistema, sendo estes governadores; *unix*, *performance*, *ondemand* e *powersave*, respectivamente, com uma melhora de desempenho ou mantendo o desempenho, quando comparado aos resultados sem essas configurações.

O estudo de Li et al. (2020) investigou a técnica de otimização de energia DVFS usando *Deep Learning* em um fluxo de dados. O objetivo é ajustar a frequência e a tensão dos processadores conforme a demanda de energia em tempo real, de modo a minimizar o consumo de energia sem comprometer o desempenho do sistema. O trabalho apresenta uma arquitetura de rede neural chamada *Sequence to Sequence LSTM Networks with Attention*, treinada em dados de energia e desempenho do sistema para prever a frequência e tensão ideais para cada carga de trabalho em tempo real. A partir da análise dos resultados, concluiu-se que a técnica proposta é eficaz em reduzir o consumo de energia em sistemas computacionais em um fluxo de dados, enquanto mantém um bom desempenho do sistema.

A investigação de otimização de energia com as configurações de hardware limitou a pesquisa em utilizar *benchmarks* genéricos e sem uma área de conhecimento específico, exceto no trabalho de Li et al. (2020), que utiliza a área de redes neurais para identificar

quando reduzir recursos de *hardware*. Contudo, esses trabalhos nos fornecem uma base para continuar a investigação de quais são os efeitos da utilização de menos recursos computacionais, voltagem menores ou maiores e desligamento de cores ociosos durante o processamento de um fluxo de dados.

3.4 Algoritmos de controle e batching dinâmico

Os algoritmos de controle têm sido a principal fonte para o desenvolvimento de novas tecnologias que precisam de autonomia. Esses algoritmos podem ser encontrados na indústria, comumente conhecidos como sistemas de automação, e também no consumo de bens de uso diário, como aspiradores de pó inteligentes (LUIZ et al., 1997).

Existem diversas formas e exemplos de algoritmos e sistemas de controle que encontramos no cotidiano, como semáforos, ligação das luzes de postes, e, na indústria, controle de vazão e temperatura das matérias-primas. Os algoritmos de controle conhecidos na literatura são: adaptativo, preditivo e *fuzzy* (LUIZ et al., 1997).

Em Bishop (2011), são apresentados o histórico destes sistemas de controle, as aplicabilidades e exemplos de uso. Os mesmos autores também apresentam os sistemas de controle com retroalimentação e explicam como são utilizados para que esses algoritmos possam se adaptar, tomar decisões baseadas em dados passados e predizer com precisão o comportamento esperado para uma situação futura similar.

O algoritmo *fuzzy* tem origem do estudo matemático de lógica ordinária. A lógica ordinária lida com declarações que podemos entender como verdade, enquanto os conjuntos lógicos *fuzzy* lidam com definições subjetivas ou relativas, como “alto”, “grande”, ou “lindas”. Dessa forma, é possível simular como os seres humanos tomam decisões, confiando em valores vagos ou imprecisos em vez de um valor falso, ou verdadeiro absoluto (CHENG et al., 2018).

Em Cheng et al. (2018) , é pesquisado sobre a utilização de *micro-batchs* para o processamento de classificação de um fluxo de dados dinâmico utilizando a ferramenta *Spark*. Utilizando as variáveis de taxa de saída e latência, investiga-se como trazer o melhor desempenho ao utilizar *micro-batches* de tamanho dinâmico. O tamanho deste *micro-batch* dinâmico é determinado por um algoritmo de controle *fuzzy*, que suporta o recebimento desses dados e a escolha dinâmica do tamanho do *micro-batch*. A utilização de um algoritmo *fuzzy* especializado permitiu que o artigo calculasse o tamanho do *batch*.

Para entender como calcular o *batch* os pesquisadores Cheng et al. (2018) propuseram primeiramente uma análise de caso a partir do consumo da CPU durante o processamento da classificação de dados. Com esta análise, notou-se que a CPU fica em média com apenas 26% de uso durante esse processamento, que o baixo paralelismo das atividades também melhorava o desempenho, aumentando a latência em 27% e diminuindo a taxa de vazão em 36%.

A implementação do *Batching* Dinâmico, descrita por (CHENG et al., 2018), utiliza a implementação de um algoritmo *fuzzy* que recebe duas entradas, $C(t)$ e $D(t)$, a entrada $C(t)$ reflete na taxa de vazão do tempo t e $t-1$ enquanto $D(t)$ reflete a latência do tempo t e $t-1$.

De acordo com resultado, o novo *batch* é calculado pelo *fuzzy* com 5 possíveis valores $f(+2), f(+1), f(0), f(-1), f(-2)$. É utilizada uma tabela de controle que corresponde a esses valores e as escolhas a partir dos resultados gerados nas equações anteriores.

O cálculo do *fuzzy* é capaz, a partir do histórico de interações e processamento, de identificar as novas variáveis naquele momento (tempo) e então predizer com o *fuzzy* seu novo *batch*. O novo *batch* por sua vez, é escolhido através da tabela pré-programada; os valores das colunas Negative Large (NL), Negative Medium (NM), Negative Small (NS), Zero (ZE), Positive Small (PS), Positive Medium (PM) e Positive Large (PL) são formas de programar o *fuzzy* para que ele possa escolher o melhor segundo da programação feita no seu algoritmo de controle.

Já em Das et al. (2014) propõem uma metodologia para processar fluxos de dados com tamanho de *batch* dinâmico. Eles propõem um algoritmo de controle simples, porém robusto, que se adapta automaticamente aos tamanhos de *batch* conforme a necessidade da situação. Os pesquisadores analisaram o comportamento de alguns fluxos de carga de trabalho visando identificar um tamanho de *batch* melhor com base na latência e no *throughput*. Algumas variáveis foram definidas para criar um algoritmo que altera o tamanho do *batch*, como atraso de agrupamento, atraso de enfileiramento (*delay*) e tempo de processamento. Eles definiram latência como "processamento de ponta a ponta de uma carga de trabalho em streaming, como a duração entre o momento em que um registro de dados entra no sistema e o momento em que os resultados correspondentes a esse registro são gerados". Com base nisso, a taxa de transferência é definida como "poder processar dados tão rapidamente quanto os dados que estão sendo recebidos".

Os pesquisadores então analisaram algumas hipóteses para criar um algoritmo de agrupamento dinâmico. Entre elas estão controles baseados em sinais binários, controles baseados em informações de gradiente, e a solução escolhida foi o Teorema do ponto fixo para contrações para encontrar o valor de agrupamento dinâmico com base em uma função $f(x)$, onde x é o intervalo de processamento do *batch* e y e é o tempo de processamento. Com base nesses dados, eles propuseram um algoritmo básico para calcular o próximo intervalo de *batch*.

A implementação do algoritmo consiste em um Gerador de Lote, um Controlador, um Gerador de Tarefas e um Processador de Tarefas. A partir dos resultados no processador de tarefas, o Controlador calcula um novo tamanho de *batch* com o algoritmo proposto.

Esses trabalhos trouxeram contribuições na área de melhoria no desempenho computacional e eficiência energética na classificação de dados em um fluxo de dados. O trabalho de Cheng et al. (2018) mostrou uma melhoria de 15% na eficiência energética e de 23% na

taxa de vazão, enquanto o de Das et al. (2014) demonstrou técnicas eficazes para melhoria nesses aspectos e também uma arquitetura e algoritmo genérico para utilização em outros estudos. Os trabalhos nos fornecem uma base teórica e de direção para a condução da pesquisa em relação à implementação do *mini-batching* com tamanho dinâmico e controle.

3.5 Considerações Finais

Foram apresentadas quatro subáreas de conhecimento relacionadas à otimização de desempenho e eficiência energética: o uso de *mini-batching* em algoritmos de classificação para fluxo de dados, a otimização de laços por meio de transformação e as estratégias de hardware para equilibrar desempenho e consumo de energia e a melhoria de *batchs* através da escolha dinâmica do tamanho. Na Tabela 1, é possível encontrar um resumo de todos os estudos realizados, incluindo seus objetivos, as técnicas utilizadas e as aplicações envolvidas.

Embora esses trabalhos tenham em comum o objetivo de otimização e a economia de energia, eles não exploraram as possibilidades de otimização por meio da transformação de laços em fluxo de dados com a implementação do *mini-batching*. Além disso, não investigaram como as configurações de hardware afetam o desempenho da classificação e do processamento em um ambiente de fluxo de dados.

Portanto, ainda há espaço para explorar as configurações de otimização de hardware e implementações de software para melhorar o desempenho e a eficiência energética. Os trabalhos relacionados demonstraram que é possível alcançar algoritmos com melhor desempenho e menor consumo de energia por meio da melhoria na localidade de dados e da configuração de hardware. Com base nessas informações, pretendemos investigar novas possibilidades para melhorar ainda mais a eficiência energética e o desempenho dos algoritmos de classificação em fluxo de dados.

Tabela 1 – Resumo das diferenças dos trabalhos relacionados e seus objetivos

Referência	Objetivo	Técnica Utilizada	Tipo de Aplicação
(ZAHARIA et al., 2012)	Tolerância a falhas e desempenho	*micro-batching*	Spark
(CARBONE et al., 2015)	Tolerância a falhas e desempenho	*micro-batching*	Flink
(CASSALES; GOMES H, 2021)	Localidade de dados	*mini-batching*	Comitês de classificadores
(GOODFELLOW; BENGIO; COURVILLE, 2016)	Localidade de dados	*mini-batching*	Gradientes Descendentes
(WITTEN; FRANK, 2002)	Localidade de dados	*mini-batching*	Gradientes Descendentes
(KUKREJA et al., 2020)	Redução do movimento dos dados, redução do *delay* e consumo energético	*mini-batching*	FWI
(KUCK et al., 1981)	Desempenho	Transformação de laços	Algoritmos em geral
(ŞTIRB; CIOCÂRLIE, 2016)	Desempenho e consumo energético	Transformação de Laços	Compiladores do LLVM
(PALLISTER; HOLLIS; BENNETT, 2015)	Desempenho e consumo energético	Transformação de Laços	Compiladores GCC
(BRANDOLESE et al., 2002)	Desempenho e consumo energético	Transformação de Laços	Algoritmos em geral
(COUTINHO et al., 2020)	Eficiência Energética	Redução de Frequência e Desligamento de componentes	Hardware
(LI et al., 2020)	Desempenho e consumo energético	Classificação de Dados	Hardware
(CHENG et al., 2018)	Desempenho e consumo energético	Classificação de Dados	Software
(DAS et al., 2014)	Desempenho	Classificação de Dados	Software
Este trabalho	Desempenho e consumo energético	Redução de Frequência, desligamento de componentes, *mini-batching*, *loop fusion*	Algoritmos de controle, Hardware e Software

Capítulo 4

Comparação entre estratégias de eficiência energética

Neste capítulo, será abordada a proposta de pesquisa para melhoria de desempenho e da eficiência energética de algoritmos de classificação de fluxos de dados que utilizam *mini-batching*, utilizando estratégias de hardware e de software. O uso de *mini-batching* para a melhoria dos comitês de classificadores para fluxos de dados foi estudada em Cassales et al. (2020), Cassales e Gomes H (2021), Cassales et al. (2022). Pretendemos contribuir com novas melhorias, bem como com os estudos comparativos entre estratégias de hardware e software. Resumidamente, os objetivos desta pesquisa são:

1. Implementar novas melhorias para a localidade dos acessos aos dados do *mini-batching* utilizando a fusão de laços das etapas de classificação e teste;

2. Avaliar se estratégias a limitação de *cores* (CL) em processadores e a mudança fixa em frequência (VFS) podem trazer benefícios para melhorar a eficiência energética dos comitês de classificadores de fluxos de dados;

Os trabalhos de Cassales e Gomes H (2021) e Cassales et al. (2022) propuseram a técnica do *mini-batching* para melhorar a localidade do acesso aos dados. Como consequência, um melhor desempenho e uma redução do consumo energético foi obtida. Nos melhores casos, foi possível ter uma aceleração no processamento de 12x e uma redução de 96% no consumo energético. Contudo, esses trabalhos não avaliaram estratégias de hardware como técnicas para otimizar ainda mais o consumo energético e o desempenho. Este trabalho propõe uma análise das otimizações de hardware para a classificação de um fluxo de dados. Com a utilização de ferramentas disponíveis nos sistemas operacionais, é

possível gerenciar *cores* e a frequência destas com objetivo de otimização de desempenho e consumo energético. Este trabalho também investiga a melhoria de localidade de dados do algoritmo de *mini-batching* através da transformação de laços.

Para isso, utilizaremos algumas ferramentas e técnicas, incluindo a implementação de *shell scripts* para mudanças nas voltagens disponíveis no processador e também a limitação de *cores*, adaptação de códigos do framework MOA e a transformação de loops em algoritmos de classificação de dados. Todo o código proposto pode ser consultado no *github* do projeto [1].

Por fim, com base nos resultados dos experimentos e na análise dos dados, será possível fornecer opções para configurar diferentes cenários de classificação em um fluxo de dados, otimizando-os para obter maior desempenho ou, preferencialmente, eficiência energética.

4.1 Otimizações no *mini-batching* por Software

Otimizações de software envolvem ajustes no código-fonte do programa para melhorar sua eficiência. Isso pode incluir mudanças em como o programa é estruturado ou como as operações são realizadas. Por exemplo, uma otimização de software pode envolver a redução do número de operações necessárias para executar uma determinada tarefa, o que pode levar a uma execução mais rápida e eficiente.

A partir do algoritmo de *mini-batching* proposto por (CASSALES; GOMES H, 2021), que apresenta dois laços não adjacentes com o mesmo índice, vamos propor uma transformação de código visando a melhoria de localidade dos acessos a dados. Uma estratégia promissora é a fusão de laços (*loop fusion*), mesclando os *loops* da linha 7 com a linha 10, que podem ser vistos no algoritmo 2. O resultado dessa transformação é mostrado no algoritmo 3.

Com base na fundamentação teórica e nos trabalhos relacionados, a implementação do Mini Batching with Loop Fusion (MB-LF) busca a otimização dos algoritmos de classificação de fluxo de dados, visando melhorar o desempenho e a eficiência energética. Através da melhoria na localidade de dados, esperamos que a otimização de transformação de laço feita aumente a eficiência do uso de caches, reduzindo a necessidade de novos acessos à memória. Como consequência, há uma melhora na aceleração do algoritmo e também no consumo energético.

[1] https://github.com/HPCSys-Lab/tecnhiques-reduce-consumption-energy

Algoritmo 3 Implementação do *mini-batching* com *loop fusion*

```
1: Entrada: um comitê de classificadores E, numThreads, um fluxo de dados S, tamanho do mini-batching L_mb
2: P = CreateServiceThreadPool(numThreads)
3: T = CreateTrainersColletions(E)
4: for cada instancia I no fluxo de dados S do
5:    B.append(I)
6:    if B.size() == L_mb then
7:       E.classify(B)
8:       for cada trainer T em trainers faça em paralelo T do
9:          T_i.instances.append(B)
10:         for each instance I in B do
11:            W ← poisson(λ)
12:            W_inst ← I * k
13:            train_on_instance(W_inst)
14:         end for
15:         if change detected then
16:            B.clear()
17:         end if
18:      end for
19:   end if
20: end for
```

4.2 Estratégia de Hardware para Redução do consumo energético

Neste trabalho, iremos aplicar estratégias de hardware visando buscar um melhor desempenho de *clock* de um CPU. Serão utilizados 2 e 4 núcleos com frequências mínima e máxima de 600MHz, além de 2 e 4 núcleos na mesma frequência, mas com 1200MHz. A versão sequencial original do trabalho (BIFET; HOLMES, 2010) também será utilizada para comparação de desempenho e consumo energético.

A Figura 6 mostra visualmente as estratégias de hardware que serão aplicadas para os experimentos. A imagem representa um processador com 4 núcleos (*cores*) que é o hardware utilizado neste trabalho. A configuração *Voltage Frequency Half 2 (VFH2)* e *Voltage Frequency Half 4 (VFH4)* representa a utilização da frequência 600MHz utilizando 2 e cores 4 respectivamente. Enquanto isso, o *Core Limiting 2 (CL2)* e *Core limiting 4 (CL4)* representa a utilização máxima da frequência nos cores, limitando apenas a disponibilidade dos cores, novamente em 2 e 4 cores.

Com os resultados destes experimentos, analisaremos os dados, de modo a comparar se a utilização paralela da classificação com estratégias de redução energética no hardware possui melhores resultados do que comparado às versões com *mini-batching* e à versão MB-LF.

Figura 6 – Representação visual das configurações aplicadas para limitação de *cores* e redução de frequência, onde o verde representa os *cores* ligados e cinza os *cores* desligados

Fonte: O Autor

4.3 Análise dos resultados

Neste capítulo, iremos avaliar os resultados dos experimentos gerados. Dividimos o capítulo por seção e subseções, a fim de facilitar a compreensão dos equipamentos utilizados para coletar os dados e a seção com a análise dos dados.

Os algoritmos escolhidos para a transformação de loops são: *loops* são: *Online Bagging*, *Oza Bag*, *OzaBag Adaptive Size Hoeffding Tree*, *Online Bagging ADWIN*, *Leveraging Bagging*, *Adaptive Random Forest* e *Streaming Random Patches*. Além disso, utilizaremos a forma sequencial do trabalho de Bifet e Holmes (2010) e a paralela implementada por (CASSALES; GOMES H, 2021). Para as alterações necessárias no hardware, utilizaremos as ferramentas *cpu-frequtils*, *numactl* e *cset*.

Os *datasets* escolhidos são frequentemente utilizados em *benchmarks* de aprendizado de máquina: *Airlines*, *GMSC*, *Electricity* e *Covertype*. Para aumentar a confiabilidade dos resultados, os experimentos foram executados múltiplas vezes, e a média dos resultados obtidos foi calculada.

Os dados de memória serão coletados utilizando a ferramenta de análise *perf-tools*, e o tempo de processamento será calculado com a ferramenta time. Além disso, utilizaremos o dispositivo de alta disponibilidade e alto desempenho *Yokogawa MW-1000* para coletar

dados de energia. Os sensores deste dispositivo coletam dados em Watt (W), sendo acessados por meio de conexão em rede.

4.3.1 O Ambiente de Hardware

O trabalho utilizou um *Raspberry PI 3 Model B*, um dispositivo simples com o modelo de arquitetura disponível na Tabela 2, para a execução dos experimentos. O ambiente de experimentação consiste em uma rede isolada conectada a um switch dedicado. Utilizamos o *Yokogawa MW-100*, um dispositivo escalável e de alto desempenho, que serviu como sensor para capturar o consumo em Watt (W).

Tabela 2 – Especificações de Hardware Rasperry PI 3

Processador	**Cortex-A53**
Cores/socket	**4**
Clock Frequency (GHz)	**1.2**
L1 cache (core)	**32 KB**
L2 cache (core)	**512 KB**
Memory (GB)	**1**

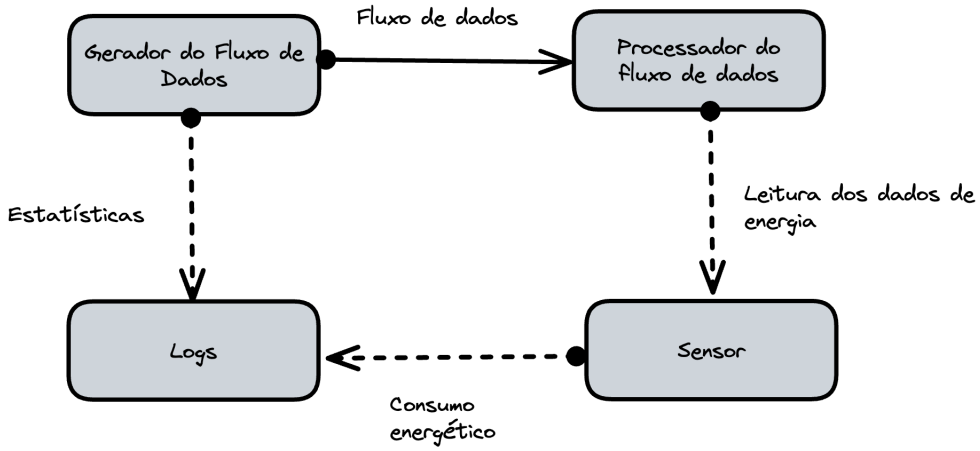
A Figura 7 ilustra o diagrama lógico do ambiente de experimentação. A máquina responsável por gerar o fluxo de dados é denominada "Gerador de fluxo de dados". Ela é responsável por carregar e enviar, via rede, as instâncias dos *datasets*, conforme a taxa de vazão configurada. Em resumo, o processo é realizado da seguinte forma: os dados são lidos a partir do disco, o *dataset* é carregado em memória e, em seguida, é aberta uma conexão com o processador do fluxo de dados para envio assíncrono dos dados. O gerador envia os dados em 4 pequenos *batches* com intervalos constantes. Isso acontece para que possamos simular o envio de informação em rede de forma intermitente. Por exemplo, uma taxa de 100 IPS enviará 20 instâncias a cada 200ms para o processador do fluxo de dados. Por fim, os logs são armazenados para serem utilizados como dados para a análise.

As técnicas foram implementadas através do *framework* MOA e testado em seis diferentes algoritmos de classificação, conforme descrito na seção 2.2 com quatro diferentes datasets, detalhados na subseção 4.3.2. A escolha destes algoritmos beneficia-se do fato de serem amplamente utilizados em muitos estudos na área de Machine Learning (ML) (BIFET; HOLMES, 2010).

4.3.2 Os datasets utilizados

Foram utilizados quatro *datasets* com acessos abertos e os detalhes das suas características podem ser vistos na Tabela 3.

Figura 7 – Diagrama lógico do ambiente de experimentação



Fonte: O Autor

Tabela 3 – Resumo características datasets.

Datasets	Airlines	GMSC	Electricity	Covertype
Instâncias	540K	150K	45K	581K
Features	7	10	8	54
Nominal feat	4	0	1	45
Normalizado	Não	Não	Sim	Sim

- O *dataset Airlines* prever se um determinado voo sofrerá atrasos, a partir de informações sobre o horário programado de partida. Assim, possui duas classes possíveis: atrasado ou não atrasou.

- O *dataset Electricity* foi coletado do *New South Wales Electricity* Market. Este *dataset* representa preços não fixos, que mudam de acordo com a demanda e essa mudança ocorre a cada 5 minutos. O *dataset* é usado para prever se o preço sobe ou desce.

- O *give me some credit (GMSC)* é uma pontuação de crédito do *dataset*, amplamente utilizado por bancos para clientes que solicitam crédito. Este *dataset* visa prever se um cliente se tornará inadimplente, com base em seu histórico de crédito, que pode ser positivo ou negativo.

- O tipo de cobertura florestal real para uma determinada observação (célula de 30 x 30 metros) foi determinado a partir dos dados do Sistema de Informação de Recursos (RIS) da Região 2 do Serviço Florestal dos EUA (USFS). Os dados estão em forma bruta (sem escala) e contêm colunas binárias (0 ou 1) de dados para variáveis independentes qualitativas (áreas, selvagens e tipos de solo).

4.3.3 Os algoritmos avaliados

Foram utilizados os seguintes comitês de classificação neste trabalho: *Online Bagging*, *Oza Bag*, *OzaBag Adaptive Size Hoeffding Tree*, *Online Bagging ADWIN*, *Leveraging Bagging*, *Adaptive Random Forest*, *Streaming Random Patches*. Todo o código nativo, implementação do *mini-batching* e MB-LF está disponível no *github* [2].

4.3.4 Análise dos Resultados

Nesta seção, serão detalhados os experimentos executados para verificar os resultados de eficiência energética, desempenho preditivo e desempenho computacional das estratégias de otimização em hardware e software já citados anteriormente, como gerenciamento manual da voltagem dos *cores*, limitando *cores* disponíveis e a transformação de *loops*. Dividiremos esta seção em 6 partes que visam identificar primeiro o desempenho ao acesso das memórias com a estratégia de transformação de *loops* implementada, a eficiência energética quando comparados em cada execução, a quantidade processada em vazão e consumo energético e, por fim, o desempenho preditivo de cada execução.

4.3.5 Medidas de avaliação

Para análise dos resultados, escolhemos nove medidas de avaliação, sendo elas: *speedup*, acesso à memória, *cache-references*, *cache-misses*, consumo médio de energia em W, acurácia, média de atraso (*delay*) nos processos, JPI e IPS. A seguir, mostraremos como cada medida é obtida.

O *speedup* é uma unidade de medida para avaliar a aceleração de um processamento, comparando com a versão sequencial. Comumente definido por S na seguinte equação:

$$S(n) = \frac{Ts}{Tn}, \tag{3}$$

sendo Ts é o tempo de processamento feito de forma sequencial e Tn é o tempo de execução paralela usando n tarefas. O resultado deste cálculo é o valor de aceleração que um algoritmo tem sobre o seu tempo sequencial (EAGER; ZAHORJAN; LAZOWSKA, 1989).

JPI é uma das unidades de medida para mensurar a relação do consumo energético. Antes da definição do JPI, é importante definirmos o consumo de energia. A ferramenta utilizada para captura dos dados de energia elétrica é expressa em W. Por outro lado, a energia é expressa em Joules (J). O valor do J é expresso pelo produto da potência W e do tempo (t), conforme mostrado na equação a seguir:

$$J = W \times t \tag{4}$$

[2] https://github.com/reginaldojunior/Parallel-Classifier-MOA/tree/loop-fusion-mini-batching

Como a fonte de alimentação pode variar, é necessária a obtenção dos dados com pequenos intervalos, visando calcular corretamente o consumo. Dessa forma, o consumo total pode ser expresso aproximadamente por:

$$E = \sum_{i=1}^{n} P_i \times t, \tag{5}$$

Sendo o n o número de amostras a serem calculadas.

Em geral, a eficiência energética é definida pela relação entre o número de instâncias processadas e a quantidade de energia gasta para esse processamento. Dessa forma, podemos definir o JPI a partir da seguinte equação:

$$JPI = \frac{E}{n}, \tag{6}$$

sendo n o número de instâncias processadas e E é a quantidade de energia gasta durante o processamento.

Outra medida de avaliação que iremos utilizar é o IPS. Essa medida avalia a quantidade de instâncias processadas em um determinado intervalo de tempo, sendo definida por:

$$IPS = \frac{n}{s}, \tag{7}$$

sendo n o número de instâncias processadas e s o tempo de execução do processamento.

Para a média de atraso $AvgDelay$ o em que ela é classificada até que o mini-batching complete o tamanho desejado e então seja processada. A média de atraso é utilizada para podermos ver o impacto causado pela estratégia de *mini-batching* e, a partir desta média, será possível ver se as implementações de hardware e software tiveram melhorias ou pioras ao serem utilizadas. A definição é dada por:

$$AvgDelay = \frac{\sum_{i=1}^{n} ILT}{n} \tag{8}$$

sendo ILT o Instance Living Time (ILT) calculado como $ILT = FPT - AT$ onde FPT é o Finish processing time (FPT) e o AT é Arrival Time (AT).

Por fim, a última medida de avaliação é composta pela equação de acurácia. A acurácia é o desempenho preditivo dos classificadores, sendo definida por:

$$Acuracia = \frac{ICC}{n}, \tag{9}$$

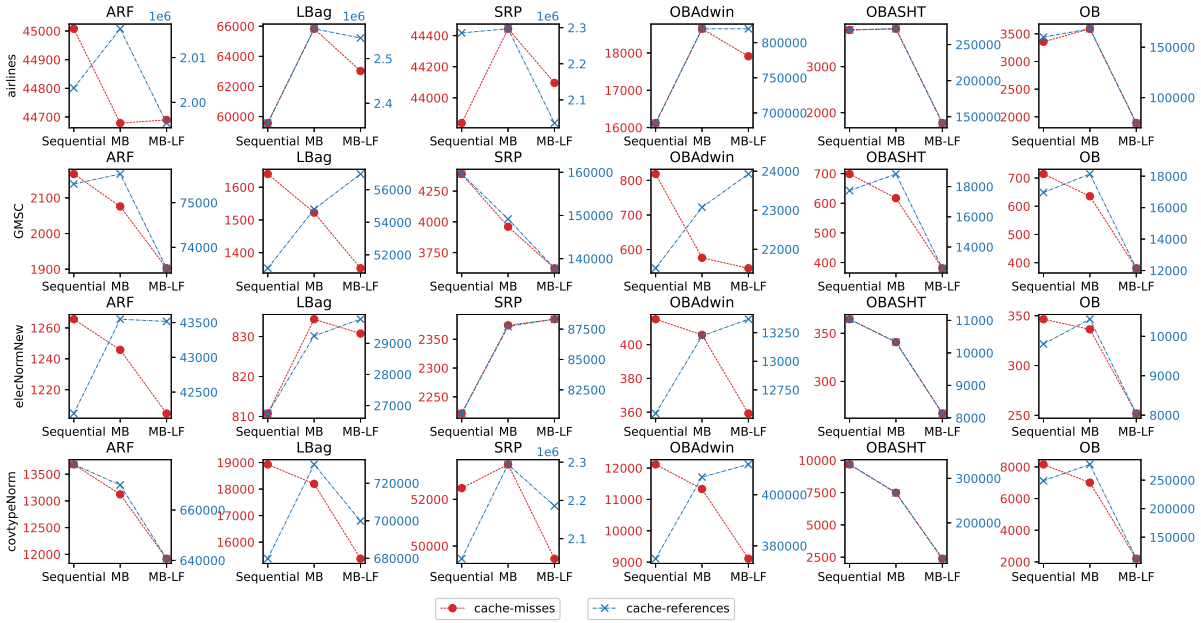
sendo ICC o número de instâncias classificadas corretamente e n é o número de instâncias processadas.

4.3.6 Melhorias no acesso à memória

A Figura 8 mostra o desempenho do acesso à memória (*cache-references*) e (*cache-misses*) com a implementação do algoritmo 3. A memória *cache-references* representa o acesso à memória mais próxima L1 e L2, enquanto o *cache-misses* a memória RAM. O resultado ótimo é quando os dois níveis diminuem, o *cache-references* e *cache-misses*, isso representa que está sendo utilizado menos recursos de memória. No melhor cenário o *cache-misses* é menor que o *cache-references* que representa que a informação está sendo buscada em quantidades menores na memória de maior custo computacional

Nos experimentos realizados com 24 execuções, a implementação com a transformação do laço só apresentou piora em 2 cenários. Em geral, o acesso às memórias *cache* foram melhorados, representando um bom resultado. Para entender esses valores as tabelas e gráficos abaixo irão mostrar visualmente essa tendencia. Na Tabela 4 é possível ver detalhadamente os valores para os acertos em *cache-misses* e *cache-references*.

Figura 8 – Gráfico com os resultados do acesso à memória olhando com dados do cache-misses e cache-references



Fonte: O Autor

4.3.7 Eficiência energética utilizando as estratégias de hardware e software

As estratégias de hardware utilizadas para redução de consumo energético foram mencionadas na seção 2.7, enquanto as de software podem ser encontradas na seção 2.8 e na seção 4.1 propomos a implementação de fusão dos laços de treinamento e classificação

Tabela 4 – Resultados do *cache references* e *cache misses* para os datasets *covTypeNorm* e *elecNormNew*.

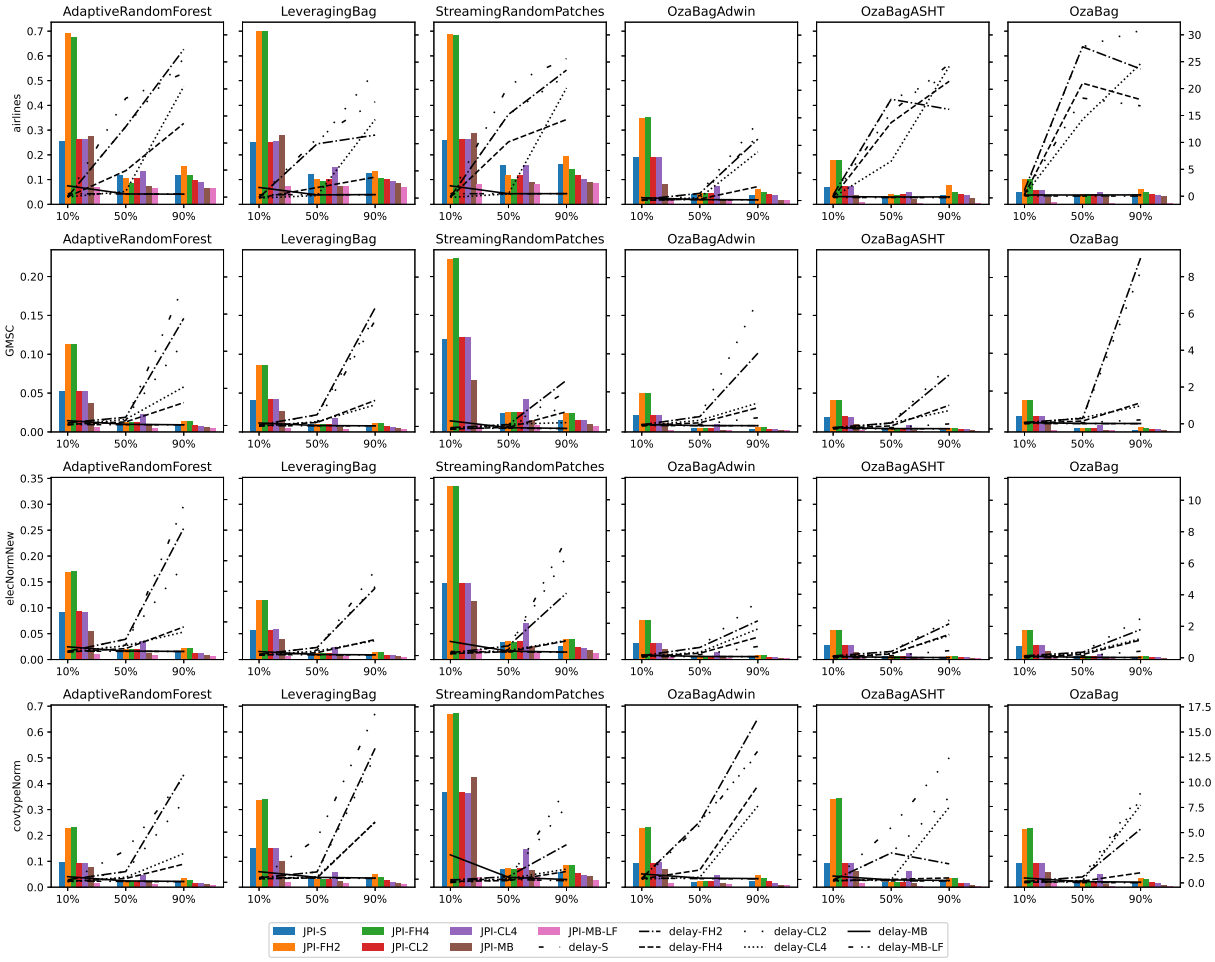
Dataset	Algoritmo	Metodologia	cache_references	cache_misses
covtypeNorm	ARF	MB	669,897.43	13,122.18
covtypeNorm	ARF	MB-LF	640,677.10	11,918.31
covtypeNorm	ARFSequential	Sequential	678,070.13	13,682.86
covtypeNorm	LBag	MB	730,019.51	18,196.36
covtypeNorm	LBag	MB-LF	699,858.36	15,375.02
covtypeNorm	LBagSequential	Sequential	679,802.79	18,929.39
covtypeNorm	OB	MB	277,561.49	7,006.85
covtypeNorm	OB	MB-LF	112,270.43	2,202.00
covtypeNorm	OBASHTSequential	MB	267,154.51	7,489.50
covtypeNorm	OBASHTSequential	MB-LF	119,360.25	2,382.40
covtypeNorm	OBASHTSequential	Sequential	331,368.78	9,680.78
covtypeNorm	OBAdwin	MB	406,924.55	11,331.64
covtypeNorm	OBAdwin	MB-LF	411,874.17	9,109.78
covtypeNorm	OBAdwinSequential	Sequential	374,971.02	12,116.11
covtypeNorm	OBSequential	Sequential	248,980.87	8,157.59
covtypeNorm	SRPSequential	MB	2,294,267.12	53,497.03
covtypeNorm	SRPSequential	MB-LF	2,185,666.34	49,457.15
covtypeNorm	SRPSequential	Sequential	2,047,501.26	52,481.53
elecNormNew	ARF	MB	43,548.64	1,245.79
elecNormNew	ARF	MB-LF	43,516.22	1,204.60
elecNormNew	ARFSequential	Sequential	42,192.10	1,265.64
elecNormNew	LBag	MB	29,240.90	834.34
elecNormNew	LBag	MB-LF	29,775.44	830.75
elecNormNew	LBagSequential	Sequential	26,746.16	810.74
elecNormNew	OB	MB	10,434.30	336.35
elecNormNew	OB	MB-LF	8,041.57	251.86
elecNormNew	OBASHTSequential	MB	10,333.30	340.77
elecNormNew	OBASHTSequential	MB-LF	8,134.80	266.67
elecNormNew	OBASHTSequential	Sequential	11,036.41	364.62
elecNormNew	OBAdwin	MB	13,221.17	405.77
elecNormNew	OBAdwin	MB-LF	13,367.87	359.21
elecNormNew	OBAdwinSequential	Sequential	12,543.59	415.05
elecNormNew	OBSequential	Sequential	9,804.87	346.53
elecNormNew	SRPSequential	MB	87,718.16	2,374.25
elecNormNew	SRPSequential	MB-LF	88,316.97	2,384.87
elecNormNew	SRPSequential	Sequential	80,552.49	2,221.08

buscando melhorar o acesso às memórias caches e consequentemente a redução do consumo energético. Iremos avaliar todas essas estratégias identificando qual delas possuem um melhor desempenho em relação ao consumo energético.

Na Figura 9 podemos observar os resultados para as execuções com as estratégias de hardware e software, onde o consumo de energia JPI está no eixo Y à esquerda, enquanto o atraso *delay* (em milissegundos) por instância aparece no eixo Y à direita, e o eixo X

representa os dados referentes a taxa de vazão do fluxo de dados que pode ser 10%, 50% e 90%. Os detalhes das configurações de parâmetros utilizados em cada execução podem ser vistos na Tabela 5.

Figura 9 – Gráfico com os resultados do JPI de todas as execuções, para a implementação de referência em MOA (S), escalonamento de frequência a 600MHz e 2 núcleos (FH2), 600 MHz e 4 núcleos (FH4), 1200MHz com 2 núcleos (CL2), 1200MHz com 4 núcleos (CL4), *mini-batching* (MB) e *mini-batching* com fusão de loop (MB-LF).



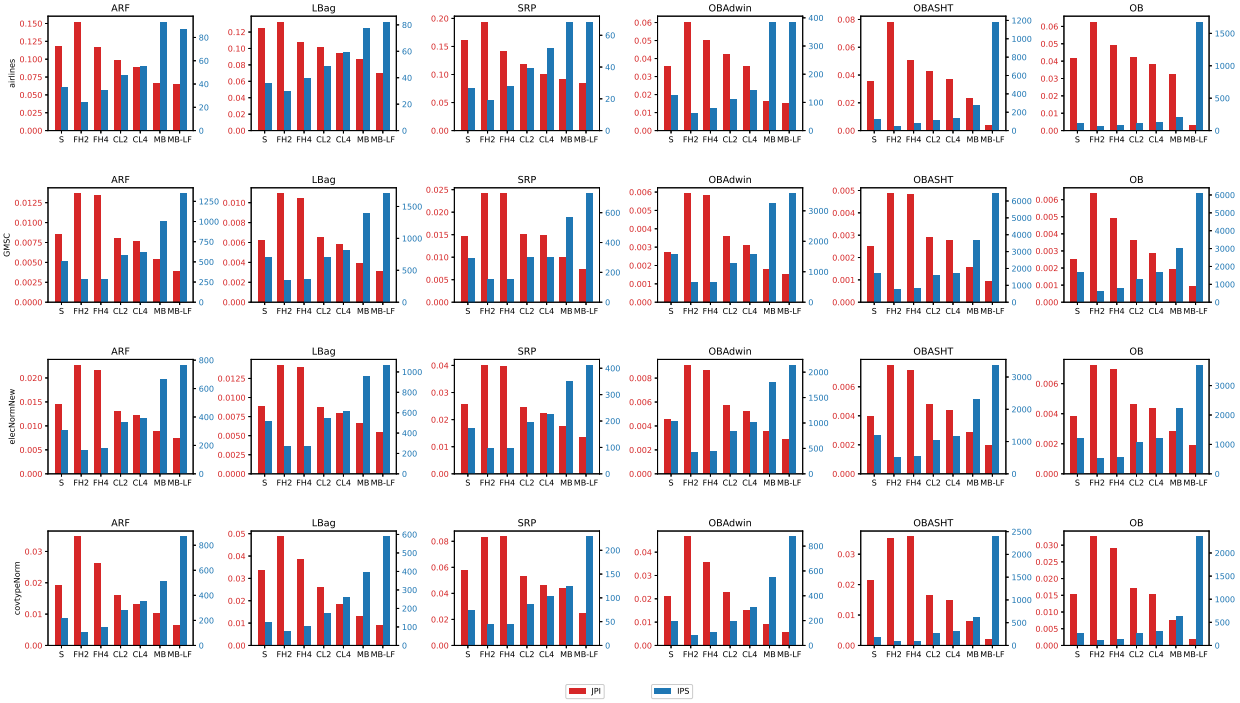
Fonte: O Autor

Quando observamos os gráficos em barras, é visível que as execuções do MB-LF apresentam os melhores resultados em todos os cenários em relação ao consumo energético geral, enquanto as execuções com redução da frequência do *clock* VFH2 e VFH4 possuem o pior desempenho energético. Os resultados para as execuções Sequencial (S), CL2 e CL4 possuem resultados muito próximos, não apresentando nenhuma melhoria quando comparados à versão sequencial e sem alterações de hardware. Torna-se evidente que as

adaptações de software como *mini-batching* e MB-LF possuem um resultado melhor em consumo energético quanto a estas outras estratégias.

Observando o JPI e o IPS, podemos notar uma tendência de melhoria quando são utilizadas as otimizações de software. Conforme pode ser visto na Figura 10, inicialmente o consumo de JPI é alto com o IPS baixo, mas com a aplicação das otimizações de software, esses gráficos mudam, apresentando um IPS maior e um JPI menor em comparação às outras execuções. É notável também que o MB-LF apresenta melhores resultados para todos os cenários.

Figura 10 – Gráfico com os resultados do JPI e IPS de todos os cenários



Fonte: O Autor

Comparando todas as execuções, é observável que o *mini-batching* melhorou energeticamente em 96% na média dos dados e no melhor caso teve uma melhora de 169%. Por outro lado, o MB-LF teve uma melhora de, 136% de melhoria no consumo energético em média e, no melhor caso de, 456%.

Podemos observar na seção anterior que há uma tendência em redução do acesso à memória nas estratégias de software aqui utilizadas que tornam o processamento com uma eficiência energética melhor do que quando não utilizadas essas técnicas, mesmo aplicando as estratégias de redução energética com hardware. Precisamos analisar se essas melhorias também apresentam bons resultados em aceleração de processamento e em qualidade preditiva. Esses tópicos serão aprofundados nas próximas seções.

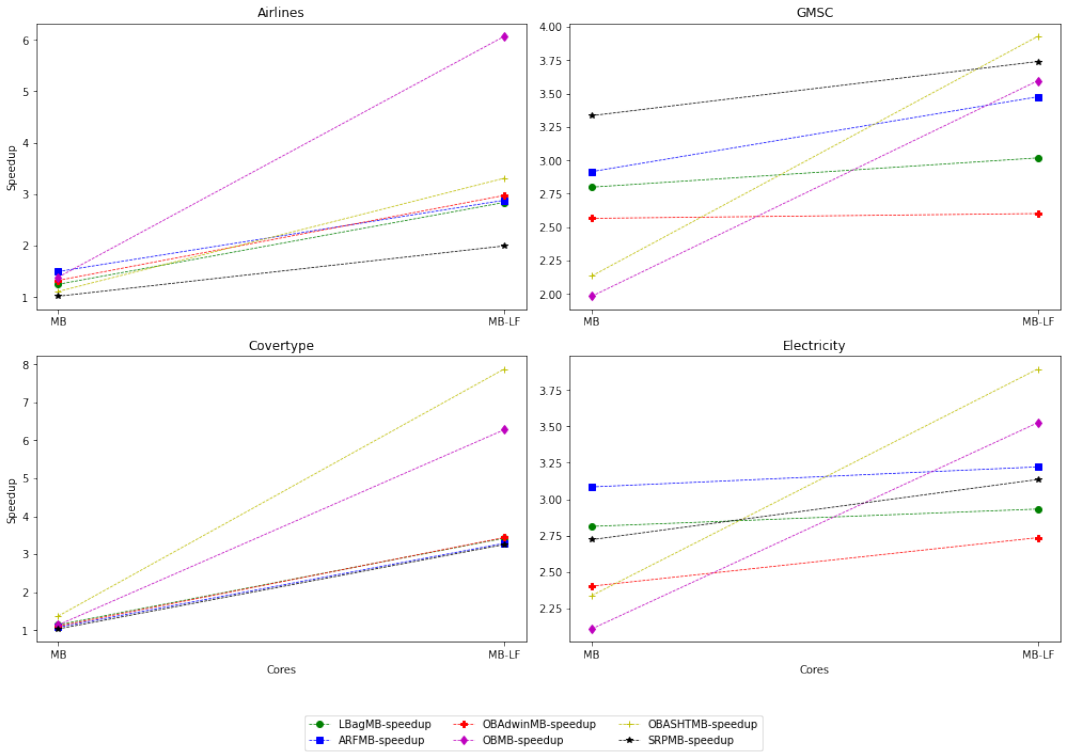
Tabela 5 – Resumo das configurações utilizados em cada experimento.

Sigla	Otimização de Software	Frequência do Clock	# of cores
S	Nenhum	1200 (máximo)	1
VFH2	Nenhum	600 (metade)	2
VFH4	Nenhum	600 (metade)	4
CL2	Nenhum	1200 (metade)	2
CL4	Nenhum	1200 (metade)	4
MB	Mini Batching	1200 (metade)	4
MB-LF	Mini Batching with Loop Fusion	1200 (máximo)	4

4.3.8 Aceleração do processamento (*speedup*)

Um dos resultados que precisamos analisar e observar a partir das alterações de software e hardware propostas neste trabalho é a aceleração do processamento computacional (*speedup*). A Figura 11 mostra os resultados obtidos de aceleração quando comparado o MB-LF com o *mini-batching*.

Figura 11 – Gráfico com os resultados de *speedup*



Fonte: O Autor

A versão sequencial não está disponível no gráfico para melhor visualizar os resultados em comparação à otimização do *loop fusion*, somente o *mini-batching*. O *mini-batching*, em geral, já se inicia com uma aceleração relativamente à versão sequencial, por exemplo.

Com o *dataset GMSC*, a otimização chega a 3.5x de aceleração quando comparada ao sequencial. Quando observamos os dados para o MB-LF, temos um desempenho médio de 4x de aceleração no processamento e de 8x no melhor caso. Isso representa um ótimo desempenho para a otimização proposta.

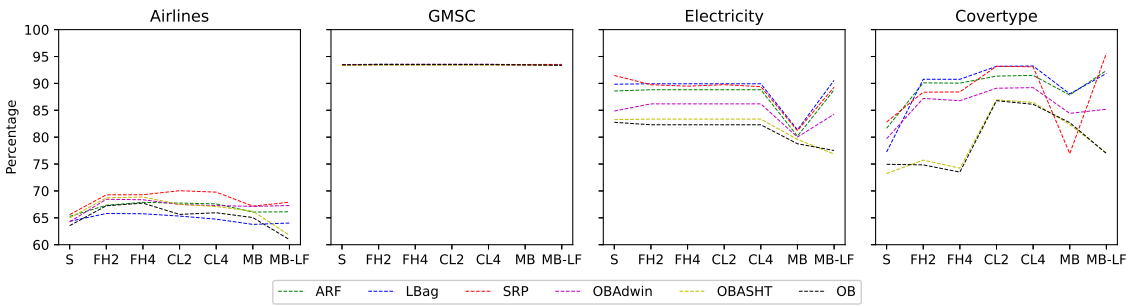
Podemos observar nos resultados para o caso de aceleração que o MB-LF teve ótimos resultados de otimização quando comparado à versão sequencial e à versão sem o *loop fusion*. Nesta análise, não consideramos o tempo de processamento para as otimizações de hardware, pois esses resultados, quando utilizam menos recursos, tendem a demorar mais do que quando são alocados mais recursos. Na próxima seção, iremos avaliar o desempenho preditivo dessa otimização, além de verificar o cenário para as otimizações de hardware.

4.3.9 Desempenho preditivo

Por fim, iremos analisar o desempenho preditivo para todas as versões de experimentos propostas. Essa informação nos fornecerá estratégias para podermos usar adequadamente cada solução proposta segundo o objetivo que queremos atingir. Visto que nas outras análises já pudemos perceber algumas perdas e ganhos, podemos então agora ter um equilíbrio com todas as variáveis importantes para o uso de classificação de fluxo de dados em computação de borda.

Na Figura 12 , é possível ver os resultados para todos os experimentos. As legendas das siglas podem ser vistas na Tabela 5. Começando pela análise de tendência, é visível que a mudança de hardware, com mais ou menos recursos, não afetam diretamente o comportamento da acurácia. Também pode-se observar que quando utilizado o recurso de otimização *mini-batching*, o gráfico tende a ter uma queda em vários cenários de execução. Em contrapartida, o cenário com MB-LF mantém significativamente o valor de acurácia quando comparado ao sequêncial e em alguns casos tendo até melhoria, como é o caso do *dataset airlines* com o algoritmo *OBADWIN*.

Figura 12 – Gráfico com os resultados da acurácia para todos os algoritmos e *datasets*



Fonte: O Autor

A análise de resultados para o desempenho preditivo mostra que a mudança de hardware não emite significativas mudanças na qualidade da predição, porém, quando utilizamos a otimização de software, pode acarretar perdas ou melhorias, como foi no caso do *mini-batching* e MB-LF respectivamente.

Capítulo 5

Mini-Batching com tamanho dinâmico

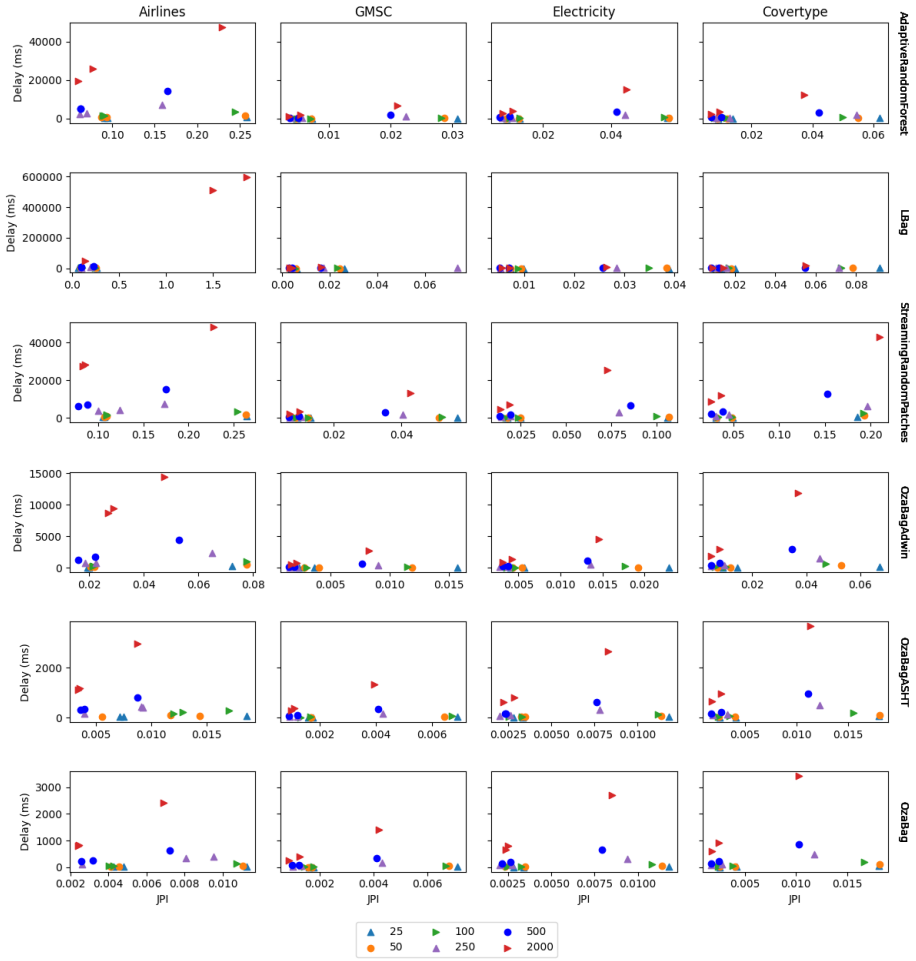
Neste capitulo será abordado a implementação do Mini Batching Dynamic (MB-D), buscando a redução do consumo de energia e o *delay*. Utilizamos otimização multi-objetivo de Pareto como padrão para a escolha do tamanho dinâmico do *batching* (ALHAMMADI; ROMAGNOLI, 2004; MIETTINEN, 1999). Com base também no Cheng et al. (2018) e Das et al. (2014), implementamos a arquitetura de experimentação. No decorrer deste capítulo, iremos discorrer e analisar os seguintes pontos:

1. Analisar, a partir da otimização de Pareto, quais são os melhores tamanhos de *batch* baseado nas porcentagens de vazão.
2. Utilizando a implementação do MB-LF iremos propor o uso do controle dinâmico do tamanho do *batch*.
3. Verificar se a utilização do MB-D e os resultados de *delay* e JPI são melhores em relação ao do MB-LF.
4. Verificar se os valores de acurácia não perdem qualidade em relação aos outros experimentos.

5.1 Implementação *mini-batching* dinâmico

A arquitetura e a implementação utilizada para execução de um fluxo de dados com o MB-D utilizou como base o MOA, um controlador de carga que se varia as porcentagens

Figura 13 – Resultado de Pareto pontos ótimos de configuração



Fonte: O Autor

de envio entre 10%, 50% e 90%. Em seguida, a partir da ocupação da CPU escolhemos os valores do MB-D conforme as regras na Tabela 6. Os valores desta tabela foram definidos conforme a fronteira de Pareto para denominar as soluções do conjunto Pareto-ótimo, a coluna "%" representa a ocupação da CPU, enquanto as colunas seguintes os prefixos dos *datasets*: Airlines (AIR), Give Me Some Credit (GMSC), Electricity (ELEC) e Covertype (COV), por exemplo, quando a execução estiver utilizando o algoritmo *Adaptive Random Forest*, na carga de 50% e no dataset ELEC o tamanho do *mini-batching* será de 500. Na Figura 13 podemos ver os resultados para as funções f_1 sendo essa representada pelo *delay* e a função f_2 o JPI. Para encontrar os seus pontos ótimos a partir da fronteira de Pareto, para tanto, utilizamos a fórmula de cálculo de distância euclidiana e os menores resultados foram escolhidos como o melhor ponto de configuração:

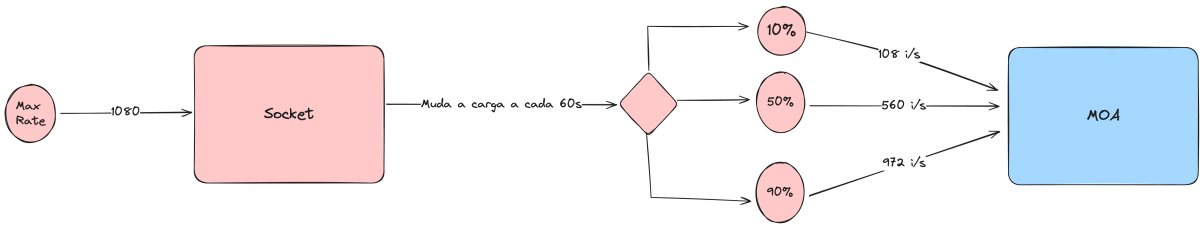
$$P = \sqrt{x^2 + y^2} \tag{10}$$

Podemos utilizar e configurar o algoritmo de decisão do MB-D a partir destes valores.

Algoritmo	%	AIR	GMSC	ELEC	COV
AdaptiveRandomForest	10%	250	500	500	100
	50%	500	500	500	500
	90%	250	500	500	500
LBagExecutor	10%	500	500	500	500
	50%	25	500	500	500
	90%	25	25	500	500
StreamingRandomPatchesExecutor	10%	250	250	250	25
	50%	500	250	500	500
	90%	500	500	500	500
OzaBagAdwinExecutor	10%	500	250	250	100
	50%	500	500	250	500
	90%	250	250	250	250
OzaBagASHTExecutor	10%	250	250	250	500
	50%	50	250	500	500
	90%	50	500	250	500
OzaBagExecutor	10%	500	250	250	250
	50%	25	500	250	500
	90%	250	250	250	500

Tabela 6 – Configurações de mini-batching dinâmico por algoritmo e conjunto de dados

Figura 14 – Ilustração do funcionamento entre o *Socket* até o MOA



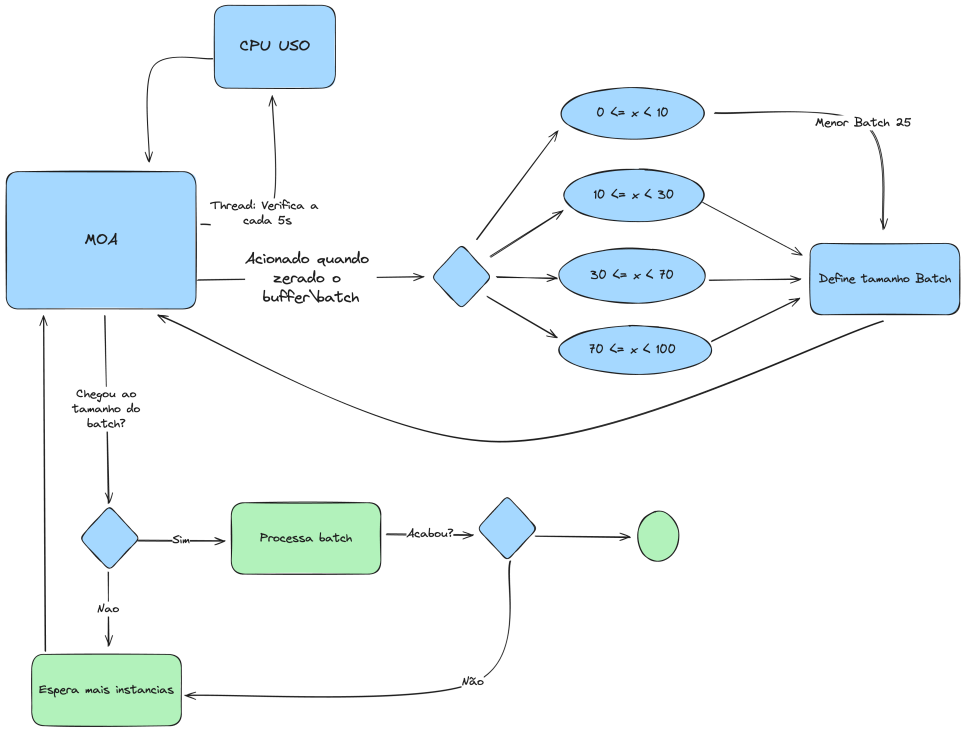
Fonte: O Autor

Já na Figura 14, podemos ver o funcionamento do experimento. O *socket* é o responsável por abrir uma conexão de rede para enviar para o MOA as instâncias a serem processadas e classificadas. Nesta arquitetura proposta, o *socket* escolhe entre três opções de carga aleatoriamente, sendo elas 10%, 50% e 90%. Ao definir esse valor, as instâncias são enviadas com essa carga durante 1 minuto e depois ocorre a troca para a nova carga aleatória, podendo ser repetida ou não.

Já na Figura 15 o MOA, é aberto uma thread que verifica o uso da CPU a cada 5 segundos a partir da carga média dos quatro núcleos disponíveis. Na CPU, é obtido o tamanho do *batch* a ser utilizado. Caso o valor do *batch* mude por conta de uma nova carga de CPU, o processamento é feito e um novo *buffer* de *batch* é iniciado.

A execução de cada algoritmo com *dataset* foi definido como 10 minutos e executado 10 vezes para podermos obter a média de todas as execuções. A execução do MB-LF

Figura 15 – Ilustração do MOA e a escolha do *batching* dinâmico



Fonte: O Autor

também tiveram modificações, nas execuções anteriores, utilizamos a execução da carga fixa em 10%, 50% e 90%, nos experimentos analisados neste capitulo utilizamos a execução aleatória destas cargas.

A definição da carga foi atribuída da seguinte forma: primeiramente, rodamos o programa com o MB-LF sem o *socket* carregando o *dataset* diretamente na memória, e então obtivemos o total de instâncias que cada algoritmo com *dataset* era capaz de processar por segundo. Através deste cálculo, obtivemos a taxa máxima para cada porcentagem.

A implementação teve as modificações que podem ser visualizados no algoritmo (4). Adicionamos a thread na linha 63 e 64, que verifica a taxa de ocupação da CPU; já na linha 65, foi adicionada a nova variável que obtém o novo tamanho do *mini-batching* a cada 5 segundos, e na linha 68 verificamos se o novo tamanho é diferente do atual. Caso seja, modificamos o valor atual na linha 69. Por fim, na linha 73, verificamos se as instâncias a serem processadas já são maiores ou iguais ao tamanho do *mini-batching* escolhido dinamicamente, caso seja, fazemos o processamento.

5.1.1 Eficiência energética e tempo de atraso (*delay*)

Nesta seção, serão discutidos os resultados de eficiência energética e *delay* da implementação do MB-D quanto comparados ao MB-LF de tamanho fixo 50, 500 e 2000. A

Algoritmo 4 Implementação do *mini-batching* dinâmico

1: **Entrada**: um comitê de classificadores E, $numThreads$, um fluxo de dados S, tamanho do *mini-batching* L_{mb}
2: P = CreateServiceThreadPool(numThreads)
3: T = CreateTrainersColletions(E)
4: Thread acionada a cada 5s
5: CPU_Occupation = CheckCPUOccupation()
6: L_{NewMB} = GetSizeMBByCPUOccupation(CPU_Occupation)
7: **for** cada instancia I no fluxo de dados S **do**
8: **if** L_{NewMB} != L_{mb} **then**
9: $L_{mb} = L_{NewMB}$
10: **end if**
11: B.append(I)
12: **if** $B.size() >= L_{mb}$ **then**
13: E.classify(B)
14: **for** cada trainer T em trainers faça em paralelo T **do**
15: T_i.instances.append(B)
16: **for** each instance I in B **do**
17: $W \leftarrow poisson(\lambda)$
18: $W_{inst} \leftarrow I * k$
19: $train\_on\_instance(W_{inst})$
20: **end for**
21: **if** change detected **then**
22: B.clear()
23: **end if**
24: **end for**
25: **end if**
26: **end for**

Figura 16 apresenta os resultados que iremos discutir.

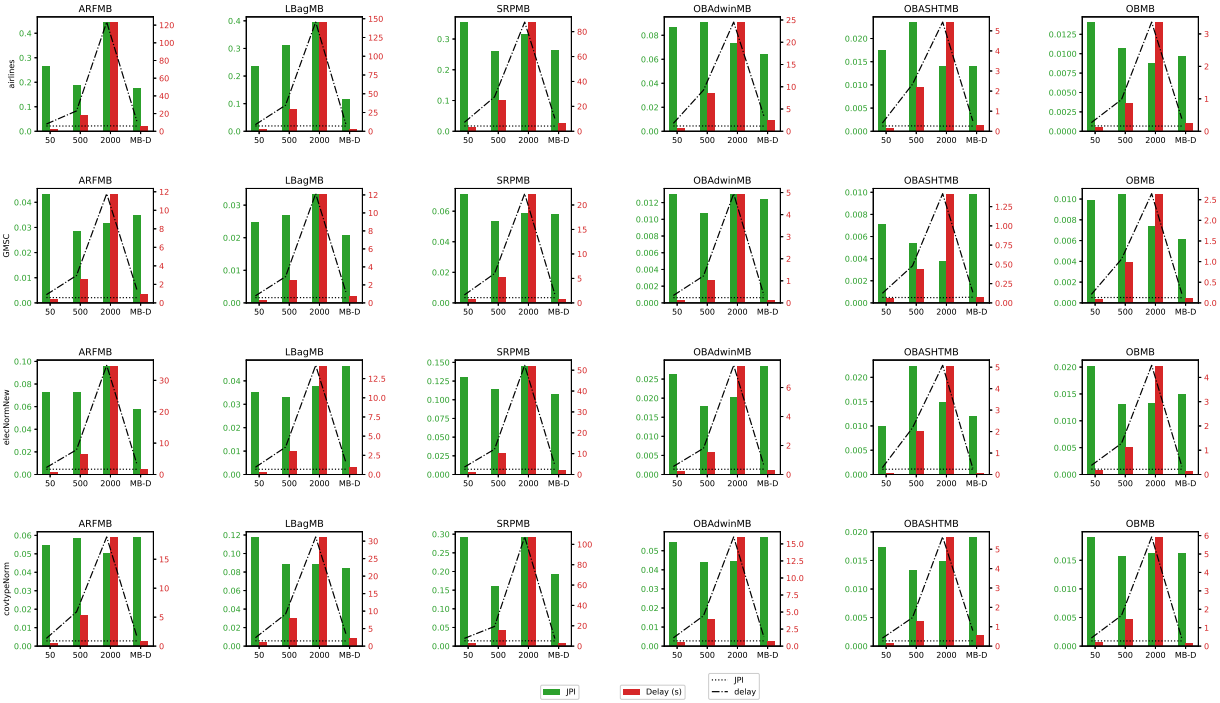
Primeiramente, iremos analisar os resultados referentes ao JPI já definidos no capítulo anterior. Os resultados do JPI têm uma tendência à queda, quando comparados ao tamanho fixo de 50, porém, com cenário de crescimento, quando comparados aos tamanhos 500 ou 2000. É possível ver um ganho de redução de consumo em 9 casos, quando comparamos a todos os tamanhos fixos, e um ganho de 16 casos, quando comparados ao melhor caso anterior, que era o tamanho 50.

Quando analisamos o *delay* ou tempo de atraso, observamos uma perda ou empate na maior parte dos casos quando comparados ao tamanho 50. No entanto, há um ganho considerável quando comparados aos tamanhos fixos de 500 ou 2000.

Os resultados nos mostra que é possível ganhar em eficiência energética se estivermos dispostos a sacrificar um pouco o tempo de atraso quando utilizamos a estratégia de MB-D. Essa estratégia permitiu um ganho de 100,33% na eficiência energética e 182% no melhor caso. Em termos de *delay*, houve uma melhoria de 21,32% na média e um máximo de 44,76%.

Também houve um ganho em relação ao atraso e ao consumo energético em 17 dos 24 casos testados quando comparados ao tamanho do *mini-batching* puro de tamanho fixo 50 e, de 24 dos 24 casos para o consumo energético quando comparados ao *batchs* de tamanhos 500 e 2000, já para o tamanho 50 não houve melhorias significativas quando verificados somente a variável de consumo energético JPI.

Figura 16 – Resultado do MB-D com as configurações do Pareto



Fonte: O Autor

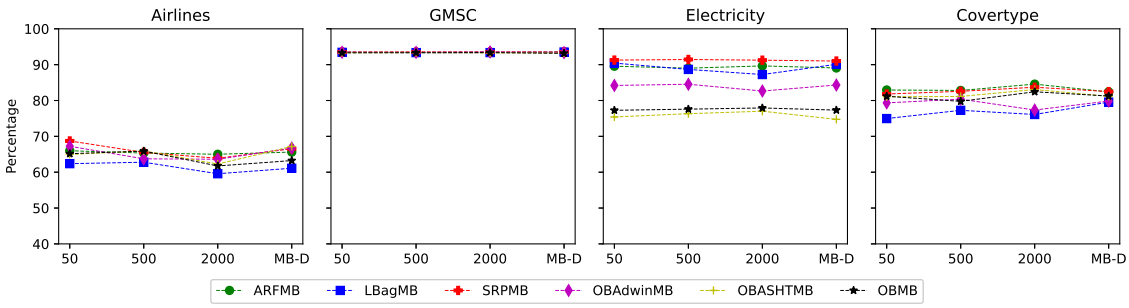
5.1.2 Desempenho Preditivo

Como último ponto de análise dos resultados do MB-D, precisamos verificar se o desempenho preditivo se manteve em relação aos experimentos anteriores, ou se teve queda. Esses dados colaboram para certificar que a implementação dinâmica, além de propor melhorias em relação ao desempenho de processamento e de energia, também pode predizer corretamente a classificação dos dados.

Os resultados preditivos da implementação do MB-D não apresentaram grandes diferenças quando comparados à implementação anterior MB-LF de tamanho fixo. Isso significa que a utilização de uma estratégia dinâmica de *mini-batching* também pode classificar corretamente os dados conforme outras estratégias já analisadas.

Podemos concluir, com base na análise do desempenho preditivo, que a implementação dinâmica não traz nenhum ganho ou perda significativa na qualidade de predição dos

Figura 17 – Resultado do MB-D de predição com as configurações do Pareto



Fonte: O Autor

algoritmos e *datasets* utilizados.

Capítulo 6

Conclusão

Comitês de classificadores são uma técnica que oferece maior confiabilidade na predição de classificações, pois utilizam uma diversidade de classificadores. Essa diversidade é aproveitada para aumentar a precisão e robustez das previsões, combinando os resultados de múltiplos classificadores individuais. Embora os comitês sejam populares por produzirem resultados altamente precisos, muitos aspectos de sua implementação eficiente ainda precisam ser estudados. No entanto, poucos trabalhos exploram melhorias na eficiência e no desempenho desses algoritmos.

O *mini-batching* é uma técnica desenvolvida anteriormente a este trabalho, que demonstrou benefícios notáveis na melhoria da localidade de acesso aos dados, resultando em melhor desempenho no processamento e redução do consumo energético, com mínimas perdas no desempenho preditivo. O presente trabalho estende a pesquisa sobre a aplicação de *mini-batching* em comitês de classificadores, seja buscando novas melhorias quanto efetuando a comparação com outras estratégias. Este trabalho oferece três principais contribuições ao estudo do *mini-batching* em comitês de classificadores.

A primeira contribuição foi investigar a fusão das etapas (loops) de classificação e treinamento. O *mini-batching* puro, proposto em (Cassales et al., 2022), melhorou a eficiência energética (ou seja, a quantidade de trabalho realizado pela mesma energia consumida), isso posto, nossa nova estratégia de *mini-batching*, que funde as fases de classificação e treinamento, melhorou a eficiência energética em 136% em média e 456% no melhor caso, em relação ao *mini-batching* original.

Como segunda contribuição, comparamos o uso do MB, uma estratégia implementada em software com as estratégias VFS e CL implementadas no hardware. A conclusão obtida foi de que nossos experimentos mostram que o *mini-batching* supera as estratégias de hardware em termos de desempenho e eficiência energética em 95,7% dos casos testados,

indicando que funciona para todos os algoritmos de *bagging*.

Por fim, nosso trabalho estende a literatura fazendo o controle dinâmico do tamanho do MB. Como contribuição, implementamos um controle dinâmico do tamanho do MB de forma inédita, utilizando fronteiras de Pareto para uma otimização multi-objetivos: redução da latência e do consumo de energia. Os resultados mostram ganhos de tivemos melhora de eficiência energética em 136.00% em média, e 456.00% no melhor caso no uso do *loop fusion* em relação ao *mini-batching* original. Isso representa uma melhoria de 17 dos 24 casos testados na redução do consumo energético, já para os casos *delay* houve uma queda em todos os testados quando comparados ao *batchs* de tamanhos 500 e 2000, já para o tamanho 50 não houve melhorias significativas.

6.1 Trabalhos Futuros

Durante a construção deste trabalho, as tentativas de cobrir o máximo de casos possíveis e analisar todas as possibilidades com o *mini-batching* tornaram-se limitadas, abrindo espaço para possíveis continuações com base nos resultados obtidos até o momento. Nesta seção, apresentamos algumas direções potenciais para futuras pesquisas.

Controlador do *mini-batching* dinâmico com *fuzzy*: O trabalho de Cheng et al. (2018) utilizou o algoritmo *fuzzy* como base para a escolha do tamanho do *batch*, conforme o SLA definido por eles. Os experimentos e conclusões desse estudo fornecem muitos insumos para a aplicação das mesmas técnicas em computação de borda utilizando o MOA. Também é possível explorar outras variáveis de verificação. Neste trabalho, utilizamos a taxa de utilização da CPU, mas é possível considerar o tempo de atraso para o processamento, o consumo de memória, as taxas de acerto de *cache-misses* e *cache-references* e, a partir desses valores, definir um algoritmo que possa predizer qual é o melhor tamanho de *batch* a ser utilizado.

Utilização de algoritmos genéticos para o controle do tamanho do *mini-batching*: Algoritmos genéticos são técnicas inspiradas na evolução natural para resolver problemas de otimização e busca. Eles podem ser aplicados ao controle do tamanho do *mini-batching* para encontrar configurações ótimas que maximizem a eficiência energética e o desempenho do sistema. Utilizando algoritmos genéticos, é possível explorar um amplo espaço de soluções, adaptando dinamicamente o tamanho dos *batches* com base em diversas métricas de desempenho, como utilização da CPU, tempo de atraso para o processamento, consumo de memória e taxas de *cache-misses* e *cache-references*.

Com base nessas possibilidades, é possível explorar novas adaptações e técnicas para aprimorar ainda mais o desempenho computacional e a eficiência energética de dispositivos IoT e de computação em borda.

A implementação de controladores de *mini-batching* dinâmicos, seja utilizando algoritmos *fuzzy* ou genéticos, oferece caminhos promissores para a otimização. A combinação

dessas técnicas com outras métricas de desempenho, como tempo de processamento, consumo de memória e taxas de *cache*, pode levar a soluções mais adaptativas e eficientes.

Referências

AGGARWAL, C. C. et al. A framework for clustering evolving data streams. In: **Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29**. [S.l.]: VLDB Endowment, 2003. (VLDB '03), p. 81–92. ISBN 0127224424.

ALHAMMADI, H. Y.; ROMAGNOLI, J. A. Process design and operation: Incorporating environmental, profitability, heat integration and controllability considerations. In: **Computer aided chemical engineering**. [S.l.]: Elsevier, 2004. v. 17, p. 264–305.

ANDRAE, A. S. G.; EDLER, T. On global electricity usage of communication technology: Trends to 2030. **Challenges**, v. 6, n. 1, p. 117–157, 2015. ISSN 2078-1547. Disponível em: <https://www.mdpi.com/2078-1547/6/1/117>.

BENINI, L.; BOGLIOLO, A.; MICHELI, G. D. A survey of design techniques for system-level dynamic power management. **IEEE Transactions on very large scale integration (VLSI) systems**, IEEE, v. 8, n. 3, p. 299–316, 2000.

BEYLS, K.; D'HOLLANDER, E. Reuse distance as a metric for cache behavior. In: CITESEER. **Proceedings of the IASTED Conference on Parallel and Distributed Computing and systems**. [S.l.], 2001. v. 14, p. 350–360.

BIFET, A.; GAVALDA, R. Learning from time-changing data with adaptive windowing. In: SIAM. **Proceedings of the 2007 SIAM international conference on data mining**. [S.l.], 2007. p. 443–448.

BIFET, A.; HOLMES, e. a. Moa: Massive online analysis, a framework for stream classification and clustering. In: PMLR. **Proceedings of the First Workshop on Applications of Pattern Analysis**. [S.l.], 2010. p. 44–50.

BIFET, A.; HOLMES, G.; AL., P. et. New ensemble methods for evolving data streams. In: ACM. **Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.], 2009. p. 139–148.

BIFET, A. et al. Improving adaptive bagging methods for evolving data streams. In: SPRINGER. **Asian conference on machine learning**. [S.l.], 2009. p. 23–37.

BISHOP, R. C. D. R. H. **Modern control systems**. [S.l.: s.n.], 2011.

BRANDOLESE, C. et al. The impact of source code transformations on software power and energy consumption. **Journal of Circuits, Systems, and Computers**, World Scientific, v. 11, n. 05, p. 477–502, 2002.

BREIMAN, L. Bagging predictors. **Machine learning**, Springer, v. 24, n. 2, p. 123–140, 1996.

CANO, A.; KRAWCZYK, B. Evolving rule-based classifiers with genetic programming on gpus for drifting data streams. **Pattern Recognition**, Elsevier, v. 87, p. 248–268, 2019.

CAO, K. et al. An overview on edge computing research. **IEEE access**, IEEE, v. 8, p. 85714–85728, 2020.

CARBONE, P. et al. Apache flink: Stream and batch processing in a single engine. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, IEEE Computer Society, v. 36, n. 4, 2015.

CASSALES, G. et al. Improving parallel performance of ensemble learners for streaming data through data locality with mini-batching. In: IEEE COMPUTER SOCIETY. **22nd International Conference on High Performance Computing and Communications**. [S.l.], 2020. p. 138–146.

Cassales, G. et al. Balancing Performance and Energy Consumption of Bagging Ensembles for the Classification of Data Streams in Edge Computing. **IEEE Transactions on Network and Service Management**, dez. 2022.

CASSALES, G.; GOMES H, e. a. Improving the performance of bagging ensembles for data streams through mini-batching. **Information Sciences**, Elsevier, v. 580, p. 260–282, 2021.

CASSALES, G. W. **Improving data locality of bagging ensembles for data streams through mini-batching**. 108 p. Tese (Doutorado) — Universidade Federal de São Carlos, São Carlos, 2021.

CHENG, D. et al. Adaptive scheduling parallel jobs with dynamic batching in spark streaming. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 29, n. 12, p. 2672–2685, 2018.

CONSUL, P. C.; JAIN, G. C. A generalization of the poisson distribution. **Technometrics**, Taylor & Francis, v. 15, n. 4, p. 791–799, 1973.

COUTINHO, A. D. et al. Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems. **Energies**, Multidisciplinary Digital Publishing Institute, v. 13, n. 9, p. 2409, 2020.

DAS, T. et al. Adaptive stream processing using dynamic batch sizing. In: **Proceedings of the ACM Symposium on Cloud Computing**. [S.l.: s.n.], 2014. p. 1–13.

DENNING, P. J. Great principles of computing. **Communications of the ACM**, ACM New York, NY, USA, v. 46, n. 11, p. 15–20, 2003.

DIETTERICH, T. G. Ensemble methods in machine learning. In: SPRINGER. **International workshop on multiple classifier systems**. [S.l.], 2000. p. 1–15.

DOMINGOS, P. A few useful things to know about machine learning. **Communications of the ACM**, ACM New York, NY, USA, v. 55, n. 10, p. 78–87, 2012.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: ACM SIGKDD. **Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2000. p. 71–80.

EAGER, D. L.; ZAHORJAN, J.; LAZOWSKA, E. D. Speedup versus efficiency in parallel systems. **IEEE transactions on computers**, IEEE, v. 38, n. 3, p. 408–423, 1989.

FENG, X.; GE, R.; CAMERON, K. W. Power and energy profiling of scientific applications on distributed systems. In: IEEE. **19th IEEE International Parallel and Distributed Processing Symposium**. [S.l.], 2005. p. 10–pp.

GAMA, J. et al. Learning with drift detection. In: SPRINGER. **Brazilian symposium on artificial intelligence**. [S.l.], 2004. p. 286–295.

GAMA, J.; RODRIGUES, P. P. An overview on mining data streams. **Foundations of Computational, Intelligence Volume 6**, Springer, p. 29–45, 2009.

GOMES, H. M. et al. A survey on ensemble learning for data stream classification. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 50, n. 2, p. 1–36, 2017.

GOMES, H. M.; READ, J.; BIFET, A. Streaming random patches for evolving data stream classification. In: IEEE. **2019 IEEE International Conference on Data Mining (ICDM)**. [S.l.], 2019. p. 240–249.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: MIT press, 2016.

HAMDAN, S.; AYYASH, M.; ALMAJALI, S. Edge-computing architectures for internet of things applications: A survey. **Sensors**, MDPI, v. 20, n. 22, p. 6441, 2020.

HOEFFDING, W. Probability inequalities for sums of bounded random variables. In: **The collected works of Wassily Hoeffding**. [S.l.]: Springer, 1994. p. 409–426.

IBRAHIM, K. Z.; STROHMAIER, E. Characterizing the relation between apex-map synthetic probes and reuse distance distributions. In: IEEE. **2010 39th International Conference on Parallel Processing**. [S.l.], 2010. p. 353–362.

JACOB, B.; WANG, D.; NG, S. **Memory systems: cache, DRAM, disk**. [S.l.]: Morgan Kaufmann, 2010.

KUCK, D. J. et al. Dependence graphs and compiler optimizations. In: **Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages**. [S.l.: s.n.], 1981. p. 207–218.

KUKREJA, N. et al. Lossy checkpoint compression in full waveform inversion. **Geoscientific Model Development Discussions**, v. 2020, p. 1–26, 2020. Disponível em: <https://gmd.copernicus.org/preprints/gmd-2020-325/>.

L, B. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

LEMAIRE, V.; SALPERWYCK, C.; BONDU, A. A survey on supervised classification on data streams. In: SPRINGER. **European Business Intelligence Summer School**. [S.l.], 2014. p. 88–125.

LI, B. et al. Online dvfs using deep learning: Sequence to sequence lstm networks with attention. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 39, n. 5, p. 966–977, 2020.

LIU, B. et al. Accelerate mini-batch machine learning training with dynamic batch size fitting. In: IEEE. **2019 International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 2019. p. 1–8.

LUIZ, C. C. et al. Controle adaptativo versus controle fuzzy: Um estudo de caso em um processo de nível. **SBA Controle & Automação**, v. 8, n. 2, p. 43–51, 1997.

MARON, O.; MOORE, A. Hoeffding races: Accelerating model selection search for classification and function approximation. **Advances in neural information processing systems**, v. 6, 1993.

MIETTINEN, K. **Nonlinear multiobjective optimization**. [S.l.]: Springer Science & Business Media, 1999. v. 12.

OLIMPIO, G. et al. Intrusion detection over network packets using data stream classification algorithms. In: IEEE. **2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)**. [S.l.], 2021. p. 985–990.

OZA, N. C.; RUSSELL, S. J. Online bagging and boosting. In: PMLR. **International Workshop on Artificial Intelligence and Statistics**. [S.l.], 2001. p. 229–236.

PALLISTER, J.; HOLLIS, S. J.; BENNETT, J. Identifying compiler options to minimize energy consumption for embedded platforms. **The Computer Journal**, Oxford University Press, v. 58, n. 1, p. 95–109, 2015.

POLIKAR, R. Ensemble based systems in decision making. **IEEE Circuits and systems magazine**, IEEE, v. 6, n. 3, p. 21–45, 2006.

RIZVANDI, N. B. et al. Multiple frequency selection in dvfs-enabled processors to minimize energy consumption. **Energy-Efficient Distributed Computing Systems**, Wiley Online Library, p. 443–463, 2012.

SATYANARAYANAN, M. The emergence of edge computing. **Computer**, IEEE, v. 50, n. 1, p. 30–39, 2017.

SCHAPIRE, R. E. A brief introduction to boosting. In: CITESEER. **Ijcai**. [S.l.], 1999. v. 99, p. 1401–1406.

SEWELL, M. Ensemble learning. **RN**, v. 11, n. 02, p. 1–34, 2008.

SHEN, X.; SHAW, J. Scalable implementation of efficient locality approximation. In: SPRINGER. **International Workshop on Languages and Compilers for Parallel Computing**. [S.l.], 2008. p. 202–216.

SNOWDON, D. C.; RUOCCO, S.; HEISER, G. Power management and dynamic voltage scaling: Myths and facts. In: CITESEER. **Workshop on Power Aware Real-time Computing, New Jersey, USA**. [S.l.], 2005. v. 31, p. 34.

STALLINGS, W. **Arquitetura e organização de computadores: projetando com foco em desempenho**. 11. ed. [S.l.]: Grupo A, 2024. E-book. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 13 jun. 2024.

ŞTIRB, I.; CIOCÂRLIE, H. Improving performance and energy consumption with loop fusion optimization and parallelization. In: IEEE. **2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI)**. [S.l.], 2016. p. 000099–000104.

TANENBAUM, A. **Organização estruturada de computadores**. [S.l.]: Pearson Prentice Hall, 2009.

TSARIOUNOV, A. **manage sets of cpus**. 2011. <https://manpages.org/cset-set>. [Online; accessed 20-December-2023].

TSYMBAL, A. The problem of concept drift: definitions and related work. **Computer Science Department, Trinity College Dublin**, Citeseer, v. 106, n. 2, p. 58, 2004.

TYAGI, S.; SHARMA, P. Taming resource heterogeneity in distributed ml training with dynamic batching. In: IEEE. **2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)**. [S.l.], 2020. p. 188–194.

WIKI, T. **How to use CPU Freq Utils**. 2023. <https://www.thinkwiki.org/wiki/How_to_use_cpufrequtils#What_is_it.3F/>. [Online; accessed 19-December-2023].

WITTEN, I. H.; FRANK, E. Data mining: practical machine learning tools and techniques with java implementations. **Acm Sigmod Record**, ACM New York, NY, USA, v. 31, n. 1, p. 76–77, 2002.

WOLPERT, D. H. Stacked generalization. **Neural networks**, Elsevier, v. 5, n. 2, p. 241–259, 1992.

YUAN, L. et al. A relational theory of locality. **ACM Transactions on Architecture and Code Optimization (TACO)**, ACM New York, NY, USA, v. 16, n. 3, p. 1–26, 2019.

ZAHARIA, M. et al. Discretized streams: An efficient and {Fault-Tolerant} model for stream processing on large clusters. In: **4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12)**. [S.l.: s.n.], 2012.

ZHANG, H. et al. In-memory big data management and processing: A survey. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 27, n. 7, p. 1920–1948, 2015.

ŽLIOBAITĖ, I. et al. Evaluation methods and decision theory for classification of streaming data with temporal dependence. **Machine Learning**, Springer, v. 98, p. 455–482, 2015.

Anexos

ANEXO A

Ambiente de Experimentação

Neste capitulo de anexos descreveremos os componentes do ambiente de experimentação. Na seção A.1 descrevemos o dispositivo de coleta de dados de energia e na seção A.2 o dispositivo responsável pelo processamento do fluxo de dados.

A.1 Yokogawa MW-100

O *Yokogawa MW-100* é um sistema de aquisição de dados modular e portátil, projetado para aplicações de monitoramento e registro de dados em tempo real. Sua flexibilidade, precisão e robustez o tornam ideal para diversas indústrias, incluindo manufatura, energia, pesquisa e desenvolvimento, e controle ambiental. Na Figura 18 é possível ver a foto do sistema em funcionamento no *rack* do laboratório de pesquisa na Universidade Federal de São Carlos no Departamento de Computação do campus de São Carlos.

A.2 Raspberry Pi 3

O *Raspberry Pi 3* é um microcomputador de baixo custo, desenvolvido pela Raspberry Pi Foundation. Ele é amplamente utilizado em educação, projetos de *hobby* e aplicações industriais leves devido à sua versatilidade, compactação e comunidade de suporte robusta. Na Figura 19 é possível ver a foto do microcomputador em funcionamento no *rack* do laboratório de pesquisa na Universidade Federal de São Carlos no Departamento de Computação do campus de São Carlos.

Figura 18 – Hardware para coleta de energia Yokogawa MW-100

Fonte: O Autor

Figura 19 – Hardware para processamento do fluxo de dados Rasperry PI 3

Fonte: O Autor