UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Lucas Otaviano Larcher

Um Estudo Experimental *Streaming-Fog-Cloud*

Juiz de Fora

2023

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Larcher, Lucas.
Um Estudo Experimental Streaming-Fog-Cloud / Lucas Larcher. -- 2023.
91 p.

Orientador: Victor Ströele
Coorientadores: Mário Dantas, Felipe Costa
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2023.

1. Fog Computing. 2. Cloud Computing. 3. Data Stream Processors. 4. Data Stream. I. Ströele, Victor, orient. II. Dantas, Mário, coorient. III. Costa, Felipe, coorient. IV. Título.

Lucas Otaviano Larcher

Um Estudo Experimental Streaming-Fog-Cloud

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 01 de março de 2023.

BANCA EXAMINADORA

Prof. Dr. Victor Ströele de Andrade Menezes - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Mario Antonio Ribeiro Dantas - Coorientador
Universidade Federal de Juiz de Fora

Prof. Dr. Felipe Schneider Costa - Coorientador
Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

Prof. Dr. Luiz Maurilio da Silva Maciel
Universidade Federal de Juiz de Fora

PROPP 01.5: Termo de Aprovação COORD-PPG-CIÊNCIA-COMPUTAÇÃO 1234200 SEI 23071.906287/2023-40 / pg. 1

Prof. Dr. Douglas Dyllon Jeronimo de Macedo

Universidade Federal de Santa Catarina

Juiz de Fora, 13/04/2023.

Documento assinado eletronicamente por **Victor Stroele de Andrade Menezes**, **Professor(a)**, em 13/04/2023, às 10:10, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Luiz Maurílio da Silva Maciel**, **Professor(a)**, em 13/04/2023, às 13:11, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Mario Antonio Ribeiro Dantas**, **Professor(a)**, em 17/04/2023, às 11:32, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **DOUGLAS DYLLON JERONIMO DE MACEDO**, **Usuário Externo**, em 17/04/2023, às 13:49, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Felipe Schneider Costa**, **Usuário Externo**, em 19/04/2023, às 09:44, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Lucas Otaviano Larcher**, **Usuário Externo**, em 19/04/2023, às 11:14, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **1234200** e o código CRC **30A60733**.

AGRADECIMENTOS

Primeiramente agradeço a minha família, com muita enfase na minha mãe, meu pai e meus dois irmãos pelo suporte. Logo em seguida minha companheira, que eu encontrei na vida e que sempre me inspira... E a Nina. Agradeço o apoio dos poucos mas valiosos amigos que tenho.

Agradeço muito ao meu orientador, Victor, pela paciência e pelo tempo, assim como meus dois coorientadores, Mario e Felipe, que se mantiveram disponíveis e solícitos durante todo o trabalho. Agradeço novamente ao professor Felipe, por se dispor e contribuir, mesmo sem nenhum tipo de obrigação, o que mudou os rumos do trabalho.

Também agradeço ao Programa de Pós-Graduação, a UFJF e a todos os professores que contribuíram positivamente para a minha evolução como cientista e como pessoa.

“In a hole in the ground there lived a...” - J.R.R.Tolkien.

RESUMO

O aumento da produção de dados tem sido constante ao longo dos anos. O crescimento da produção de dados acaba gerando a necessidade de inovação de tecnologias no ramo de processamento de dados. As ferramentas de streaming de dados são softwares que tem como propósito lidar com grandes quantidades de dados com o objetivo de manter, gerir e analisar. Um problema quando se pensa em streaming de dados é o custo computacional para realizar tarefas para uma quantidade massiva de dados.Muito dessa dificuldade é devido a geração de uma grande quantidade de dados que, por vezes, o sistema não consegue lidar. Isso acontece pois o hardware disponível não tem poder computacional suficiente para lidar com todas as tarefas em tempo hábil. A solução mais básica envolve a utilização de mais hardware, até que seja suprida essa necessidade. Verifica-se também uma lacuna de recursos ociosos considerando os hardwares espalhados na rede que ainda não atingiram seu potencial máximo. Quando esses hardwares são habilitados para serem utilizados nas ”pontas”da rede, cria-se a disponibilidade para processar, armazenar e gerenciar os dados próximo ao elemento gerador de dados, diminuindo latência, aumentando a disponibilidade de hardware, disponibilizando mais poder computacional, entre vários outros benefícios. Essa é a *Fog*. O objetivo deste trabalho é experimentar na *Fog* através de um estudo voltado para o campo dos dados, verificando questões como a redução da sobrecarga gerada por fluxo de dados massivos, discutir o impacto da utilização das ferramentas de processamento de streaming de dados, verificar consumo de recursos e o poder computacional da plataforma, validar a utilização do ambiente como uma solução viável, entre outros. Neste trabalho foram realizados dois experimentos, sendo o primeiro fundamentalmente mais teórico, com a análise e experimentação na camada *Fog* sob a perspectiva do uso das Ferramentas de Processamento de Streaming de Dados (FPSD) e as limitações de recursos computacionais proporcionadas pelo hardware. O segundo experimento tenta solucionar um problema de conectividade no meio educacional entre alunos de ensino superior que estão alocados em polos educacionais e a *Cloud* gerida pela universidade. Nesse caso, a disponibilidade da conexão com a internet é bastante precária e a intervenção da utilização de um sistema em *Fog* pode sanar parte dos empecilhos para que a educação a distância torne-se mais eficiente e tenha melhor qualidade de experiência para os usuários. Os experimentos apresentam resultados animadores. Em ambos os casos a solução da *Fog* computacional atingiu as expectativas. No primeiro experimento, quando comparado com um sistema com mais recursos computacionais, o hardware se saiu bem e apresentou resultados além do esperado quanto a quantidade de elementos processados, enquanto os valores consumo de recurso seguiram comportamento semelhante. No segundo experimento a solução *Fog* se encaixa muito bem, resolvendo os principais problemas relacionados a qualidade de experiência do usuário, aumentando a disponibilidade e garantindo manutenção dos dados.

Keywords: fog computing, cloud computing, data stream processors, data stream.

ABSTRACT

The increase in data production has been constant over the years. The growth of data production genarates a need for innovation in technologies in the field of data processing. Data streaming tools are powerful software that aim to deal with large amounts of data to maintain, manage, and analyze. The biggest problem when thinking about data streaming is the computational cost to perform tasks for a massive amount of data. Part of this difficulty is ralated to a generation of a large amount of data that the system cannot handle. This happens because the available hardware does not have enough computational power to handle all tasks promptly. The most basic solution involves using more hardware until works. There is also a gap in idle resources, considering the hardware spread across the network that has not yet reached its maximum potential. When this hardware is enabled to be used at the "ends" of the network, availability is created to process, store and manage data close to the data generating element, reducing latencies, increasing hardware availability, providing more computational power and etc. This is Fog. The objective of this work is to promote the use of Fog through a study focused on the field of data, verifying issues such as reducing the overload generated by the flow of massive data, discussing the impact of using streaming data processing tools, verifying resource consumption and the computational power of the platform, validate the use of the environment as a viable solution, among others. In this work, two experiments were carried out, the first being fundamentally more theoretical, with the analysis of the Fog layer from the perspective of the use of Streaming Data Processing Tools (SDPT) and the limitation of computational resources provided by the hardware. The second experiment tries to solve a connectivity problem in the educational environment between education students who are allocated in educational centers and the Cloud managed by the university. In this case, the availability of the internet connection is quite precarious and the intervention of using a Fog system can remedy part of the obstacles for distance education to become more efficient and have the best quality of experience for users. The experiments showed encouraging results. In both cases, Fog's computational solution do what is expected. In the first experiment, when compared to a system with more computational resources, the hardware performed well and presented results beyond expectations regarding the number of elements processed, while the resource consumption values followed a similar behavior. In the second experiment, the Fog solution go very well, solving the main problems related to the quality of user experience, increasing availability, and ensuring data maintenance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

FPSD	Ferramentas de Processamento de Streaming de Dados
Fog	Computação em Fog
Cloud	Computação em Cloud
IoT	Internet of Things

SUMÁRIO

1 INTRODUÇÃO

Avanços na área da tecnologia da informação vieram para integrar e agregar em todos os setores, tanto para facilitar a vida da população quanto auxiliar nos meios de produção. Alguns desses avanços são paradigmas como a *Cloud* (MELL; GRANCE et al., 2011), Internet das Coisas (IoT) (XIA et al., 2012), indústria 4.0 (LASI et al., 2014), entre outros.

Essas novas tecnologias surgem por conta do crescimento no fluxo de dados gerados por usuários, sistemas e sensores. Neste contexto, existe um tipo de fonte específica categorizada como streaming de dados (GAMA; GABER, 2007), que são os fluxos de dados variados gerados por interações entre usuários, sistemas e sensores e por possuírem essas características tem maior complexidade para ser processado.

Uma streaming (ou fluxo) de dados é uma sequência de itens enfileirados ao longo do tempo. O intervalo de tempo é variável, assim como sua estrutura e disposição. O fluxo desses dados possui tamanho potencialmente infinito e assim se torna um problema complexo (SILVA et al., 2013). Como se trata de um volume de dados por vezes massivo e em tempo real, os métodos e abordagens tradicionais já desenvolvidos podem não ser eficazes para processá-los. Surge então a necessidade de descobrir novos métodos, cada vez mais eficientes, para resolver tal demanda (DING et al., 2015).

As tecnologias que utilizam de fluxos de dados podem encarar alguns novos tipos de problemas: (i) a falta ou limitação de recursos computacionais, como memória e tempo, bem como necessidades de processamento dos dados em tempo hábil; (ii) a distribuição de dados que ocorrem no fluxo ao longo do tempo, que pode impactar drasticamente no desempenho do modelo ou algoritmo usado; e (iii) os dados podem chegar tão rapidamente que processar todos os itens pode demorar ou às vezes até ser inviável (KRAWCZYK et al., 2017).

A análise destes grandes volumes dentro de um fluxo de dados é foco de muitos pesquisadores. Nos modelos de streaming de dados no ambiente IoT, por exemplo, o desafio está relacionado à velocidade do fluxo de entrada dos dados, desenvolvendo algoritmos que processem esses dados sob restrições muito rígidas de espaço e tempo (MORALES et al., 2016). Em alguns ambientes relacionados a cuidados com a saúde, há a necessidade de um monitoramento constante e um trabalho incessante em análise sobre valores vitais que são entregues através de uma streaming de dados (TOOR et al., 2020). Na indústria 4.0 a transmissão de dados massivos ocorre de forma natural devido a natureza inteligente dos processos. Neste cenário, dispositivos inteligentes tem o objetivo de aumentar o desempenho de máquinas industriais em aplicações, através de previsão de falhas, detecção de problemas de qualidade e a necessidade de realizar manutenção preventiva, reduzindo custos e melhorando a qualidade dos produtos (TA; LIU; NKABINDE, 2016).

Todo esse fluxo de dados precisa de um ambiente com poder computacional suficiente para seu processamento e análise. Uma solução amplamente adotada é a *Cloud Computing*), considerando seu potencial computacional utilizado para resolução de problemas diversos (SUNYAEV, 2020). Por definição, a computação em *Cloud* é um modelo que permite acesso ubíquo, prático e sob demanda a um conjunto compartilhado de recursos de computação configuráveis que podem ser provisionados rapidamente a qualquer momento e de qualquer local via Internet (MELL; GRANCE et al., 2011).

O crescimento e a variedade de aplicações na *Cloud* tem levado muitas empresas e usuários a realizarem tarefas online a maior parte do seu dia-a-dia. Como resultado da popularidade da web, os provedores querem garantir a disponibilidade de acesso à informação e serviço para seus usuários e a garantia de que as solicitações sejam processadas o mais rápido possível (RADOJEVIĆ; ŽAGAR, 2011). O problema surge quando o fluxo e o volume de dados é muito intenso, custando muito ou se tornando tão abundante que os sistemas se tornam incapazes de alcançar seus critérios mínimos de funcionamento, como *real-time*, tolerância a falhas, confiabilidade e disponibilidade. Surge então a necessidade de pensar em outras alternativas, como a utilização de sistemas de computação auxiliares, como a abordagem *Fog* (HONG; VARGHESE, 2019).

Ao invés de usar os grandes recursos computacionais dos servidores na *Cloud*, distantes do usuário e centralizados, há um movimento para empregar recursos descentralizados na borda de uma rede para processar dados mais próximos dos dispositivos do usuário, como em *smartphones* e *tablets*, conhecido como computação em *Fog*/*Edge*(HONG; VARGHESE, 2019). Os avanços das arquiteturas baseadas no paradigma *Fog Computing* vem possibilitando a resolução de problemas em diversos contextos, como as *Smart Homes, Smart Grids, Smart Vehicles, Health Data Management*, dentre outros (LAGHARI; JUMANI; LAGHARI, 2021; SINGH; SINGH; GILL, 2021).

A aplicação da computação em *Fog* no ambiente IoT pode resultar em maior eficiência, desempenho e redução na quantidade de dados transferidos para a *Cloud* para processamento, análise e armazenamento (ATLAM; WALTERS; WILLS, 2018). Portanto, os dados coletados próximos da borda, como sensores, são enviados para dispositivos de borda de rede para processamento e armazenamento temporário, em vez de enviá-los para a *Cloud*, reduzindo assim o tráfego e a latência da rede (WEN et al., 2017).

Se todo o volume de dados gerados pela streaming fosse enviado para a *Cloud*, o custo e o tempo para processamento desses dados seria inviável. Isso demonstra como é imprescindível a abordagem de processamento de dados em ambientes mais próximos dos dispositivos nos quais os dados são gerados.

Considerando os sistemas de dados, observa-se a evolução de como os dados são transmitidos, saindo dos *batches* (pacotes), caminhando para os *micro-batches* e, finalmente, chegando nas *streamings* de dados. Esse avanço aconteceu devido a evolução

de características da rede, permitindo uma transmissão melhor com menos atraso (ISAH et al., 2019).

As ferramentas utilizadas para grande volume de dados também se modificaram com o tempo. Baseado em algorítimos de divisão e conquista, o MapReduce (DEAN; GHEMAWAT, 2008), que é o algoritmo mais utilizado para sistemas de processamento de dados em massa, foi implementado primeiramente no software Apache Hadoop[1] para processar grandes quantidades de dados de forma paralela. A tecnologia foi evoluindo através de novos sistemas de dados, acompanhados também de novas ferramentas, como o Apache Storm[2], o Apache Spark[3] e, atualmente, o Apache Flink[4].

Com o intuito de viabilizar o processamento de streaming de dados com um menor impacto na *Cloud*, o uso do paradigma *Fog Computing* tem se mostrado promissor. Entretanto, os recursos da *Fog* são, normalmente, heterogêneos e dinâmicos em comparação com a *Cloud*, tornando o gerenciamento de recursos um desafio importante que precisa ser abordado(HONG; VARGHESE, 2019).

Surge então uma lacuna, o processamento de streaming de dados sobre a camada de Computação em *Fog*. Considerando o contexto da pesquisa neste trabalho, o intuito é responder as seguintes questões de pesquisa:

Questão de Pesquisa Primária (QPP) - *A cooperação Fog-Cloud é viável para redução de sobrecarga de processamento na Cloud quando do uso de ferramentas de processamento de streaming de dados em uma camada textitFog?*

Com essa questão é possível verificar de uma maneira abrangente como o uso da cooperação *Fog-Cloud* pode contribuir, a partir do resultado dos experimentos, com o cenário de processamento de streaming de dados. Assim, apresentamos a investigação dos seguintes pontos:

- É importante saber se os dispositivos na camada *Fog* conseguem realizar processamento de dados, mesmo utilizando sistemas computacionais diferentes. Tarefas essas que estão relacionadas a geração, manutenção, gerenciamento e processamento de dados. Dessa forma, vai ser possível compreender melhor o poder computacional que se tem disponível na camada *Fog* para sugerir, prever e aplicar essa solução em um cenário real.

- Considerando as FPSD, é interessante observar as soluções e resultados durante a realização das tarefas relacionadas ao processamento dos dados e verificar se estas

[1] https://hadoop.apache.org/
[2] https://storm.apache.org/
[3] https://spark.apache.org/
[4] https://flink.apache.org/

tarefas foram concluídas de forma satisfatória. Dessa forma, é possível avaliar a respeito de quais tipos de tecnologias e funcionalidades estão disponíveis para estes softwares. Com base nessas análises, também é possível mensurar a capacidade da camada de computação *Fog* em relação a uma camada de computação em *Cloud*. Assim, é possível gerir melhor suas capacidades e ter a habilidade de projetar o uso de tais tecnologias em um ambiente real.

- Esses resultados são importantes para compreender o poder da *Fog* e como esse tipo de aplicação pode auxiliar na realização de tarefas. Com esse entendimento, abre a disponibilidade de planejar novas aplicações e experimentos de forma com que seja possível realizar mais um salto tecnológico, assim como os sistemas do passado saíram dos mainframes para os clusters computacionais.

- É importante questionar se a camada *Fog* é eficiente para resolver um problema de aplicação no mundo real para validar a proposta de um sistema *Fog-Cloud*, avaliando e passando por questões como origem, processamento, gerenciamento e armazenamento.

Com a investigação desses pontos, é plausível responder a questão de pesquisa e avaliar o impacto que as aplicações que dependem de um processamento de dados de streaming provocam nos sistemas de processamento de dados, e verificar a viabilidade do paradigma *Fog Computing* para gerar uma menor dependência da *Cloud* computacional.

1.1 OBJETIVO

que seria investigar a viabilidade da cooperação Fog-cloud para redução da sobrecarga de processamento em cloud quando utilizando ferramentas de processamento de streaming na camada fog

O objetivo do trabalho é investigar a viabilidade da cooperação Fog-cloud através de experimentos utilizando ferramentas de processamento de streaming na camada *Fog* para compreender da capacidade e características envolvidas no processamento de streaming de dados na camada. O propósito é trazer mais interesse às aplicações executadas na *Fog* para reduzir a sobrecarga de trabalho na *Cloud*.

Assim, este trabalho propõe uma arquitetura de um framework baseada na cooperação *Fog-Cloud* para processamento de streaming de dados massivos na camada de computação em *Fog*, principalmente pensando nos diferentes hardwares que compõem esse ambiente. Durante o desenvolvimento da solução, a execução dos experimentos, e análise dos resultados, essas questões são respondidas, e a escalabilidade da proposta para processar os fluxos de dados é avaliada.

A partir de todas essas tecnologias apresentas que englobam fluxo de dados, computação na *Fog*, processamento de streaming de dados e outros, desenvolvemos soluções para dois experimentos que abordam esses conceitos. O primeiro verifica a viabilidade de uso das FPSD, monitorando o comportamento dessas ferramentas em uma camada *Fog* sobre algumas óticas distintas, como consumo de recursos computacionais, poder de processamento, etc. O segundo é um experimento considerando a aplicação dessas tecnologias no mundo real tentando resolver um problema que visa minimizar perdas de pacotes relacionados a perda de conectividade. A intenção é utilizar do poder da interação *Fog-Cloud* para resolver essa aplicação.

1.2 CONTRIBUIÇÕES

Assim, as principais contribuições da abordagem são resumidas em:

- Uma arquitetura baseada na cooperação *Fog-Cloud* para o processamento de streaming de dados;
- Execução de experimentos em um hardware típico de camada *Fog*;
- Resultados atingidos pelas experimentações das ferramentas na camada *Fog*.

1.3 ORGANIZAÇÃO

O trabalho inicia com o capítulo introdutório, que traz as primeiras impressões sobre o trabalho e alinha o caminho que o trabalho pretende percorrer durante sua execução. Nele é apresentada a contextualização do problema, a questão geral de pesquisa, o objetivo e as contribuições do trabalho.

No segundo capítulo é apresentada a base teórica necessária para a compreensão do trabalho como um todo. Os principais conteúdos deste capítulo passam por computação em *Cloud* e *Fog*, e streaming de dados. Depois da apresentação deste conteúdo, são apresentados alguns trabalhos que dialogam com os experimentos que são executados aqui.

O terceiro capítulo apresenta a solução proposta nesta dissertação. A arquitetura do framework é apresentada e suas camadas são descritas. As tecnologias utilizadas no desenvolvimento do framework também são descritas nesse capítulo.

No quarto capítulo a proposta é avaliada através da execução de dois experimentos. O primeiro visa analisar a viabilidade de uso de ferramentas de processamento de streaming de dados em uma camada *Fog*. Já o segundo, verifica a viabilidade do uso da solução para a resolução de um problema real.

Finalmente, o quinto capítulo traz as conclusões sobre o trabalho e os trabalhos futuros desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conteúdos necessários para que seja possível a compreensão da pesquisa realizada. Aqui são apresentados os conteúdos relacionados a computação em *Cloud*, computação em *Fog* e streaming de dados, respectivamente. Após a apresentação dos conteúdos bases, são apresentados trabalhos presentes na literatura que em sua execução se aproximam das resoluções apresentadas para os dois experimentos realizados neste trabalho.

2.1 CLOUD COMPUTING

A Computação em Nuvem, do inglês *Cloud Computing*, é um modelo ubíquo, sob demanda, que tem como objetivo prover um conjunto compartilhado de recursos de computação através de uma rede (como armazenamento, processamento, aplicações, serviços e redes). Com isso, consegue fornecer tecnologias que anteriormente não poderiam ser acessadas, de forma rápida e com o mínimo de burocracia possível (MELL; GRANCE et al., 2011).

A *Cloud* oferece economia quando há redução nos custos de operação, e, dessa forma, o retorno de um investimento é mais rápido. Do lado do provedor, isso leva a uma maior produtividade no provisionamento de serviços de infraestrutura (BRIAN et al., 2008). Dessa forma, as organizações podem se concentrar mais em suas competências essenciais e os serviços restantes podem ser terceirizados, podendo gerar melhores resultados do que os encontrados anteriormente (BRIAN et al., 2008).

Toda essa junção de tecnologias contribui para que um sistema se torne cada vez mais robusto e consistente. Isso não significa, necessariamente, maior complexidade, muito pelo contrário, pois certos detalhes de implementação são transparentes para o usuário, que pode se dedicar apenas na especificação de parâmetros que realmente interessam a ele (BRIAN et al., 2008).

A infraestrutura da *Cloud* consiste em unidades físicas, como servidores, CPU, armazenamento e rede (VISHWANATH; NAGAPPAN, 2010). Como a estrutura é acessada pela rede, pode existir uma quantidade considerável de saltos entre o usuário e o hardware. Dessa forma, podemos dizer que existe um atraso relacionado a esse tipo de serviço, podendo impactar negativamente a aplicação que depende exclusivamente da *Cloud*.

A Figura 1 exemplifica a diversidade de serviços disponíveis que são acessados através da rede, como espaço em disco, processamento de dados, entretenimento, etc. Ainda na Figura 1, é possível visualizar uma característica importante, a distância entre o usuário e o serviço, que gera retardo e maior concorrência de uso.

Atualmente, é possível observar Software, Infraestrutura, Plataformas e Serviços

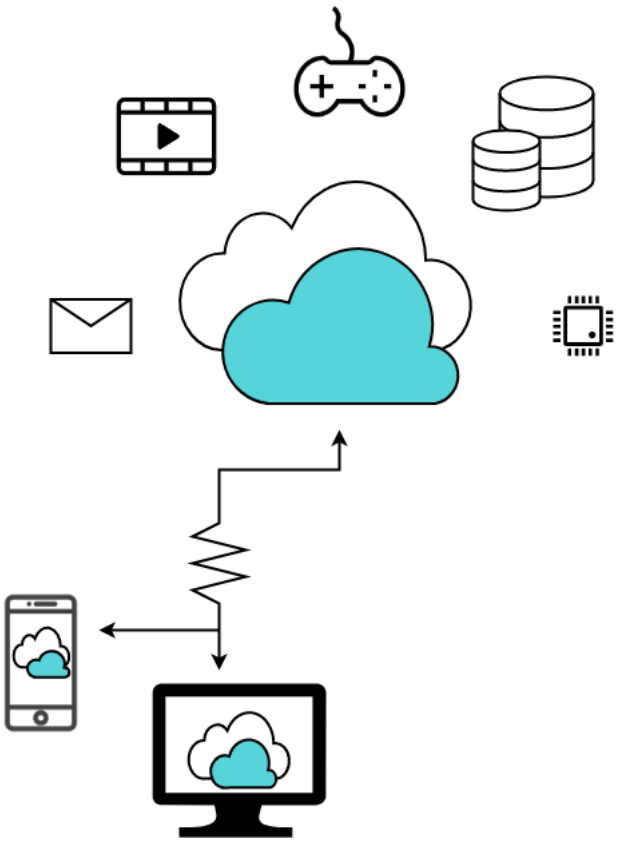

Figura 1 – Modelo de computação em *Cloud*.

disponibilizadas através da abordagem de computação em *Cloud*. Dessa forma, muitas aplicações já foram transferidas para a *Cloud* e a partir de uma conexão com a internet é possível realizar a maior parte das tarefas que o universo da computação pode oferecer (BRIAN et al., 2008).

2.2 FOG COMPUTING

A computação em *Fog* é introduzida como um paradigma emergente e inovador para estender os serviços em *Cloud*. Embora a computação em *Cloud* seja apresentada como um modelo que fornece acesso sob demanda e onipresente a um sistema compartilhado de recursos de computação e armazenamento. Os recursos de *Cloud* estão longe dos usuários e, como resultado, ela pode suportar, principalmente, serviços de alta latência.

A *Fog* pode estender esses recursos de computação e armazenamento incorporando uma camada de transição entre os dispositivos do usuário e a *Cloud*, levando a uma hierarquia de três camadas: camada de dispositivos dos usuários, camada *Fog* e camada de *Cloud*. A camada *Fog* intermediária consiste em um conjunto de hardwares como celulares, computadores de placa única, roteadores e gateways que estão distribuídos geograficamente e localizados o mais próximo possível dos dispositivos dos usuários (MELL; GRANCE et

al., 2011; GROUP et al., 2017).

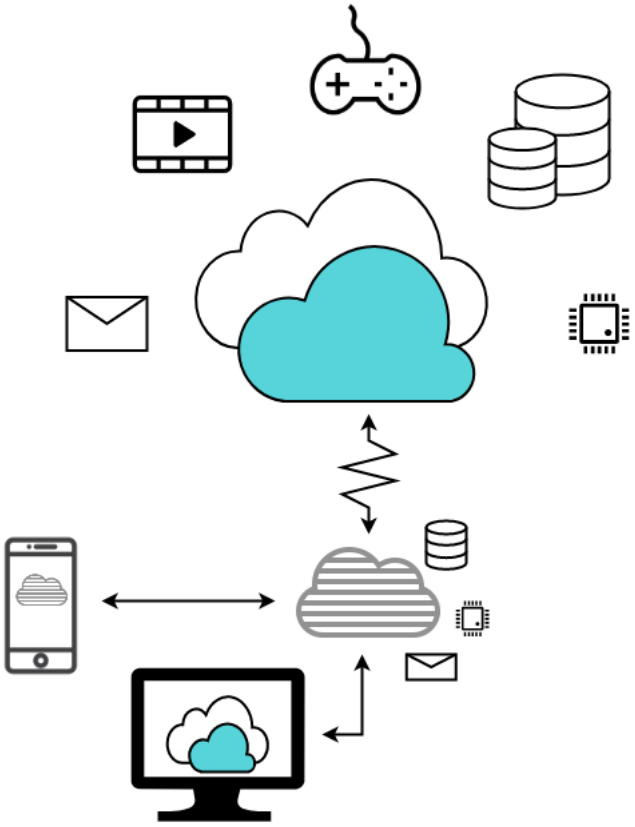

Figura 2 – Modelo de computação em *Fog*.

Na Figura 2 é possível verificar uma representação da estrutura da Computação em *Fog*. Nela existe o hardware presente na *Fog*, que está mais próximo do usuário e distante da *Cloud* e, além disso, apresenta menos concorrência por recursos uma vez que os recursos de hardware estão disponibilizados por região.

Outra característica da *Fog* é a escalabilidade, dada a facilidade de adicionar novos nós, mesmo que heterogêneos, a rede e o sistema mantém a capacidade de trabalharem em conjunto com facilidade (IORGA et al., 2018).

2.3 STREAMING DE DADOS

Com o crescimento massivo da produção de dados, as tecnologias utilizadas começaram a não performar o suficiente para cumprir suas respectivas tarefas. Dessa forma, novas tecnologias começaram a aparecer, como o MapReduce (DEAN; GHEMAWAT, 2008) e o Hive (THUSOO et al., 2009). O MapReduce é um algoritmo muito importante à implementação de processamento de datasets muito grandes, o algoritmo surge da ideia de dividir para conquistar (DEAN; GHEMAWAT, 2008). Para suprir as necessidades de armazenamento e gerenciamento surge o Hive que, com uma linguagem própria oriunda do SQL, realiza buscas com o objetivo de alimentar o MapReduce (THUSOO et al., 2009).

Essas tecnologias se tornaram muito eficientes, mas sofriam com a alta latência devido ao uso de pacotes (batch) de dados, pois estes tem um atraso intrínseco ao tempo de juntar os dados em um pacote. Agora havia a necessidade de realizar as operações com um tempo de resposta muito pequeno para alimentar sistemas e ambientes como o de pagamento de cartão de crédito, condições de tráfego, dados de um paciente, etc. (ISAH et al., 2019).

O que faz a streaming de dados ser diferente do modelo convencional de dados é a forma que os dados se apresentam para serem tratados, analisados e processados. Alguns dos preceitos mais relevantes que diferem da forma convencional são (SILVA et al., 2013):

- Os dados chegam de forma contínua;
- Não existe controle sobre a ordem em que os dados chegam e são processados;
- O tamanho da streaming é potencialmente infinito;
- Os objetos são descartados depois de serem processados ou depois de um período de tempo;
- Processo de geração de dados é desconhecido e não necessariamente segue um padrão.

As ferramentas de stream de dados contam com tecnologias semelhantes ao MapReduce para o processamento de dados. No entanto, possuem a capacidade de manter e processar elementos de forma sequencial, tratando a questão do atraso na entrega da informação (ISAH et al., 2019).

2.3.1 Janela de Tempo

As janelas de tempo são uma abordagem comumente utilizada para resolver problemas em que se tem a necessidade de analisar dados da streaming dentro de uma perspectiva que possua início e fim dentro do fluxo. Isso ocorre quando a visualização do dado dentro da streaming como um todo faz pouco ou nenhum sentido, mas quando analisado dentro de um contexto temporal ou quantitativo, gera informação (GAMA; GABER, 2007).

Existem diferentes modelos de janelas de tempo na literatura, mas o tipo mais relevante é a janela deslizante, que considera apenas um corte dos dados mais recentes. Nesse caso, o corte possui um tamanho n e a cada momento em que um novo dado chega, o dado mais antigo do corte é excluído para dar lugar ao recém chegado (GAMA; GABER, 2007). Outros exemplos seriam a janela inclinada (HAN; PEI; KAMBER, 2011), a janela de referências (GEHRKE; KORN; SRIVASTAVA, 2001) e janela amortecida (YUN et al., 2018).

2.3.2 Ferramentas de Streaming de Dados

Estes softwares surgem para lidar com os dados gerados de forma assíncrona e com o fluxo desordenado. As streaming de dados são sequências de dados, ilimitadas em tamanho, em tempo real e muitas vezes contém itens de dados multidimensionais, ou seja, tuplas (STONEBRAKER; ÇETINTEMEL; ZDONIK, 2005).

Existem tentativas de ajustar sistemas clássicos como Sistemas de Gerenciamentos de Bancos de Dados ou mecanismos de regras para gerenciar o processamento de fluxo de dados. No entanto, eles tendem a falhar em termos dos requisitos de processamento em tempo real.

Dessa forma, aplicações orientadas a processamento de fluxos são substancialmente diferentes das convencionais, atendendo aos critérios de processamento de grandes volumes e baixa latência. Ao criar um sistema de processamento de fluxo de dados, dentre os principais requisitos para processamento em tempo real estão (STONEBRAKER; ÇETINTEMEL; ZDONIK, 2005):

- *Processamento em fluxo.* Onde o sistema processa dados sem realizar busca e armazenamento com grande frequência em um banco de dados. Ao fornecer operações sem acesso a memória, as operações de armazenamento demoradas são omitidas no pipeline de processamento, o que resulta no aumento da eficiência do tempo do sistema.

- *Manipulação de imperfeições de fluxos.* Ao lidar com fontes imprevisíveis, a capacidade de se adaptar e lidar com imprevisibilidades é essencial em termos de cálculo de dados parciais, deficientes, atrasados ou desordenados.

- *Segurança e disponibilidade.* A ocorrência de falhas é uma preocupação crítica. Assim, os aplicativos baseados em fluxo precisam estar ativos e, em caso de falhas, devem ter a capacidade de se recuperar, garantindo a disponibilidade e a integridade dos dados.

- *Processamento distribuído.* Para lidar com a quantidade massiva de elementos, ter acesso a várias unidades de computação como multicores (BLAKE; DRESLINSKI; MUDGE, 2009) e clusters (BAKER, 2000) é muito importante em termos de ganho de escalabilidade.

- *Minimização do tempo de resposta.* Quando o aumento no número de volumes de dados no tempo é observado, o sistema deve se manter obtendo respostas com baixo tempo de resposta e de maneira contínua.

2.4 TRABALHOS RELACIONADOS

É importante, durante a execução de um trabalho, verificar o que foi produzido na literatura para evitar retrabalho e verificar as lacunas que ainda devem ser exploradas com a intenção de criar mais conhecimento fundamentado em repostas já obtidas por

outros pesquisadores. Dessa forma, nesse passo é importante encontrar trabalhos que se relacionem de alguma forma com a proposta desta dissertação.

Com isso, foram realizadas pesquisas na literatura à procura de trabalhos que sejam, de alguma forma, semelhantes ou que vão de encontro com as ideias abordadas nesta dissertação. Os trabalhos apresentados a seguir revelam os caminhos que os pesquisadores seguiram para resolverem problemas que possuem uma interseção com a arquitetura proposta e os dois experimentos realizados durante este trabalho.

Em Costa, Nassar e Dantas (2022), os autores discutem a relação entre o crescimento do volume de dados e a grande quantidade de sensores conectados na rede, o que viabiliza ambientes totalmente tecnológicos, como cidades inteligentes. A grande quantidade de sensores e dados também trazem outras questões importantes, como a necessidade de implementação de técnicas mais robustas capazes de selecionar fontes de dados confiáveis na rede e manter a baixa latência.

Os autores apresentam uma solução projetada para ser executada no ambiente *Fog* por meio do FOCUSeR, um método para avaliar os sensores da rede. Este método realiza a avaliação dos dados em tempo real, a partir de aprendizado online, para gerar um critério para o ranqueamento, possibilitando identificar falhas e anomalias em sensores no ambiente.

O sistema, apesar de não ser tão robusto quanto uma *Cloud*, apresenta bons resultados de acurácia e precisão e, com a utilização de um sistema em *Fog*, garante questões como baixa latência e disponibilidade de recursos. Os autores destacam que a *Fog* pode fazer um trabalho robusto e garantir baixa latência, que é imprescindível para diversos sistemas críticos.

Em Chintapalli et al. (2016), há um relato sobre processamento de dados em streaming, que vem ganhando atenção devido à sua aplicação em uma ampla gama de cenários. Com o objetivo de atender às crescentes demandas de processamento de dados de streaming, muitos mecanismos de computação foram desenvolvidos. No entanto, verifica-se a necessidade de benchmarks do mundo real que seriam úteis ao escolher a plataforma mais adequada para atender às necessidades de streaming em tempo real. Para resolver esse problema, os autores em Chintapalli et al. (2016) desenvolveram um benchmark de streaming para três FPSD consideradas representativas: Flink, Storm e Spark Streaming. Foi construído um pipeline de dados completo usando Kafka[1] e Redis para simular os cenários de produção do mundo real.

Os autores concluem que a competição entre sistemas de streaming quase em tempo real está esquentando e não há um vencedor claro neste momento. Cada uma das plataformas estudadas por eles tem suas vantagens e desvantagens. O desempenho é

[1] https://kafka.apache.org/

apenas um fator entre outros, como segurança ou integração com ferramentas e bibliotecas. Comunidades ativas para esses e outros projetos de processamento de big data continuam inovando e se beneficiando dos avanços umas das outras.

Os autores em Shahverdi, Awad e Sakr (2019) explicam sobre o mundo mais conectado, gerando uma enxurrada de dados a partir de vários hardwares (por exemplo, sensores) ou software no formato de fluxos de dados. O processamento em tempo real para quantidades tão grandes de dados de streaming é um requisito crucial em vários domínios de aplicativos, incluindo mercados financeiros, sistemas de vigilância, manufatura e cidades inteligentes.

Com a necessidade de novos testes de desempenho para avaliar versões mais recentes, novas funcionalidades, e o surgimento de novas ferramentas, aparece então a oportunidade de realizar novos benchmarks, assim como realizado anteriormente em Chintapalli et al. (2016). Em Shahverdi, Awad e Sakr (2019) é apresentado um extenso estudo experimental de cinco sistemas populares neste domínio: Apache Storm, Apache Flink, Apache Spark, Kafka Streams e Hazelcast Jet. Nele são relatadas e analisadas as características de desempenho desses sistemas. Além disso, é apresentado um conjunto de *insights* e lições importantes que surgiram com a condução dos experimentos.

Ambos os trabalhos entregam um sistema base de avaliação das ferramentas de streaming de dados, um complementando e atualizando o outro, e realizam uma análise de um sistema em *Cloud*. Nesta dissertação, o objetivo também é avaliar as ferramentas, mas com o foco na *Fog*, o que altera o que vai ser definido como fatores importantes de análise para esse ambiente. A relação do consumo de recursos computacionais em comparação à quantidade de elementos processados é um resultado muito mais importante para o estudo, por exemplo.

Em Battulga, Miorandi e Tedeschi (2020) os autores introduzem a infraestrutura de computação em *Fog* como uma alternativa para diminuir a carga de recursos de computação de provedores da *Cloud*, colocando dispositivos de borda mais próximos dos usuários finais e/ou fontes de dados, uma vez que sistemas e métodos para desenvolver e implantar um aplicativo em nós *Fog* não são tão maduros.

A FogGuru é uma plataforma para desenvolvimento e aplicações na camada de *Fog* com base na abordagem de streaming de dados através do motor da ferramenta Apache Flink. O objetivo é oferecer facilidades para um desenvolvedor criar um projeto na camada *Fog/Edge* para aplicações de processamento de streaming de dados para dar melhor suporte a elementos de Internet das Coisas.

O trabalho Battulga, Miorandi e Tedeschi (2020) detalha o funcionamento da arquitetura e fornece instruções de como o programador deve proceder e realizar um experimento. É apresentado um aplicativo de análise de tráfego baseado em *Fog* implantado

usando o FogGuru e executado em um nó de *Fog*, composto por um cluster de cinco Raspberry Pi com a utilização do Docker[2], através da funcionalidade do Docker Swarm[3], que permite trabalhar um cluster de contêineres.

Neste caso, o autor está interessado em transformar a *Fog* em um sistema facilitado de processamento de streaming de dados. Aqui a implementação é o foco e a preocupação é ter, antes de tudo, um sistema funcional e viável para o processamento de streaming de dados.

A automação de alto nível é comum em um ambiente Industrial de IoT (IIoT) (LEE et al., 2017). Dispositivos IIoT automatizados estão entre as tecnologias mais cotadas para a melhoria da eficiência operacional da fábrica. Eles ajudam a reduzir o tempo operacional e as despesas com pessoal e a aumentar a precisão operacional. Os dispositivos IIoT para fábrica inteligente são de vários tipos e são projetados para cumprir seus trabalhos designados para a fábrica de diversas maneiras.

Em Lee et al. (2017), os autores analisam os dados de fluxo que são gerados por máquinas em uma fábrica e como eles podem ser processados mais rapidamente, resultando em um tempo de inatividade menor e a previsão de falhas sendo feita mais cedo. Se um dispositivo IIoT for inteligente o suficiente para processar dados, ele poderá obter informações significativas processando os dados por si só. Também é possível utilizar outro hardware para coletar dados de pequenos sensores e processar os dados para tomar decisões. Para tratar este tipo de processamento de dados, que é realizado nas máquinas na fábrica, é utilizada a *Edge computing*, que está dentro do espectro da *Fog*.

Dispositivos *Edge* geralmente têm estruturas, propostas e arquiteturas diferentes de um hardware da *Cloud*, então é verificada a necessidade de saber qual mecanismo de processamento de dados pode ser usado para dispositivos de borda. Três mecanismos foram selecionados, o Apache Flink, Apache Storm e Apache Spark streaming. Eles foram instalados em um Raspberry Pi 3 e foi realizada a avaliação de suas latências e taxas de transferência.

Na conclusão, o Apache Flink e o Apache Spark Streaming apresentaram resultados concorrentes em todos os testes enquanto o Apache Storm apresentou desempenho inferior aos demais, além de exigir mais recursos de memória para executar seus componentes básicos, o que prejudicou a execução.

Esse trabalho se aproxima em partes do caminho que é percorrido nesta dissertação, mas a preocupação não está diretamente relacionada na informação que os dados carregam, o objetivo é descobrir mais sobre questões como o consumo de recursos computacionais por parte das ferramentas para um maior planejamento sobre a escolha da ferramenta,

[2] https://www.docker.com/
[3] https://docs.docker.com/engine/swarm/swarm-tutorial/

além é claro de avaliações em comparação com o camada da *Cloud*.

Nos países em desenvolvimento, o ensino superior é um veículo essencial do processo de desenvolvimento. Os Sistemas de Gestão da Aprendizagem surgem com um papel importante ao auxiliar no ensino superior. Em suma, o desenvolvimento das tecnologias, as infraestruturas de telecomunicações e a Internet têm um forte impacto no sector da educação, especialmente com o advento do ensino à distância que se torna cada vez mais presente.

Mais e mais instituições acadêmicas estão migrando para a educação a distância e hoje o Moodle é uma das ferramentas mais populares com esse objetivo. No entanto, no Moodle, os alunos devem estar conectados on-line o tempo todo para realizar atividades na plataforma. Além disso, a introdução ao ensino a distância nos países em desenvolvimento é feita com um conjunto de restrições como custos de conexão, cortes de energia e falha permanente de conectividade à Internet em algumas áreas.

Em Ngom, Guillermet e Niang (2012) os autores apresentam uma solução aprimorada para executar o Moodle no modo off-line para melhorar o aprendizado assíncrono. Esta solução permite que os alunos continuem as atividades de educação a distância nos casos em que a conexão com a Internet é altamente instável ou não existe. A solução apresentada utiliza configuração transparente e automática para o PC ou laptop do usuário final. Após a restauração da conexão, todas as atividades off-line são sincronizadas com a plataforma principal do Moodle.

Neste trabalho não existe a apresentação de um sistema claro de *Fog*, mas sim de uma ferramenta que atua em cada máquina individualmente. Apesar de apresentar alguns poucos elementos de uma solução em *Fog*, são estruturas de ensino diferentes.

Um material educacional bem projetado é muito importante para a implementação bem-sucedida do ensino a distância (IJTIHADIE et al., 2012). Professores e instrutores desempenham um papel importante em termos de concepção e construção de conteúdo de aprendizagem, tarefa que requer custos em termos de esforço, tempo e experiência. Um bom conteúdo de aprendizagem é provavelmente resultado de revisões recorrentes como resultado da experiência de ensino, bem como da avaliação das atividades dos alunos.

No caso de instituições de ensino superior em países em desenvolvimento como a Indonésia, o compartilhamento de recursos educacionais é um esforço altamente recomendado contra trabalhos de alto custo e redundantes. Compartilhar e reutilizar conteúdo de educacional sobre determinado assunto entre Sistemas de Gestão de Aprendizagem pode ser um dos métodos. Além disso, o ensino colaborativo pode fazer com que um conteúdo se desenvolva gradualmente ao realizar o compartilhamento de conteúdo.

Dessa forma, a capacidade de sincronizar o conteúdo entre as universidades é necessária. Por outro lado, a implementação típica de sistemas de educação a distância

pode não suportar essa funcionalidade, devido às restrições de infraestrutura de rede nos países em desenvolvimento, principalmente em algumas áreas, onde a rede tem pouca infraestrutura, gerando problemas como baixa largura de banda e até mesmo desconexões frequentes.

Em Ijtihadie et al. (2012), é apresentado um novo método de compartilhamento de conteúdo de educação a distância entre sistemas de gerenciamento de aprendizado distribuídos usando sincronização dinâmica de conteúdo. Este método também atende à necessidade de compartilhamento de cursos que apoia a atividade de ensino colaborativa. Além disso, essa abordagem foi projetada para atender às necessidades de compartilhamento de conteúdo em áreas com limitações de infraestrutura de rede em termos de largura de banda e disponibilidade.

A resolução aqui passa próxima da anterior, verificando qual é o melhor momento para que a atualização dos dados seja realizada. Aqui existe uma colaboração entre os diferentes sistemas de educação a distância, o que também torna a abordagem distante do que se projeta no segundo experimento, passando longe da questão da aplicação da camada *Fog*, que é onde esta nossa principal contribuição.

O trabalhos encontrados são importantes para refletir sobre soluções que foram desenvolvidas e sobre as tecnologias aplicadas em cada um dos trabalhos. Dessa forma, podemos ter o entendimento de como se deu a evolução da tecnologia na área. Outra percepção foi a de que a computação em *Fog* ainda é uma área de pesquisa recente na academia, e tem necessidade da produção de mais literatura para se criar uma base sólida de conhecimento, o que faz com que a tecnologia seja utilizada da sua melhor forma.

3 METODOLOGIA

Neste capítulo são apresentados os procedimentos metodológicos adotados para o desenvolvimento da pesquisa, sua caracterização e o percurso metodológico percorrido para atender aos objetivos definidos durante a pesquisa.

3.1 CARACTERIZAÇÃO DA PESQUISA

De acordo com Silva e Menezes (2005), norteado por GIL (1991), a caracterização da pesquisa normalmente é classificada quanto a: natureza, abordagem, objetivos e procedimentos. Dessa forma, do ponto de vista da sua **natureza**, esta pesquisa é classificada como uma **pesquisa aplicada**, ou seja, ela tem o objetivo de gerar conhecimento aplicado para prática dirigido à solução de problemas específicos. Esse tipo de solução envolve verdades e interesses locais.

A **forma de abordagem** é **quantitativa**, ou seja, é baseada em números. Através de quantidades é possível dar significado e traduzir números em respostas, resultados, opiniões e informações e assim classificá-las e analisá-las. Essa abordagem faz o uso de recursos e técnicas estatísticas (percentagem, média, moda, mediana, desvio-padrão, coeficiente de correlação, análise de regressão, etc.).

Os **objetivos** caracterizam uma **pesquisa explicativa**, pois se trata de identificar os fatores que determinam ou contribuem para a ocorrência dos fenômenos. Esse tipo de pesquisa aprofunda o conhecimento da realidade porque explica a razão, o "porquê" das coisas. Requer o uso do método experimental.

Do ponto de vista dos **procedimentos técnicos**, ela pode ser considerada uma **pesquisa bibliográfica**, que é quando é elaborada a partir de material já publicado, constituído de livros, artigos de periódicos e material disponibilizado na Internet e também é uma **pesquisa experimental**, que é quando se determina um objeto de estudo, selecionando as variáveis, definindo as formas de controle e de observação dos efeitos que esta variável produz no objeto.

3.2 PROCEDIMENTOS METODOLÓGICOS

Os dois experimentos seguem um fluxo de procedimentos semelhantes, que serão tratados aqui de forma geral e, quando necessário, especificados para cada um dos experimentos realizados no desenvolvimento do trabalho.

1. Análise do problema relacionado a cada problema apresentado;
2. Levantamento na literatura de trabalhos que, de alguma forma, se relacionam e corroboram com conhecimento inicial para o trabalho;

3. Definição de questões de pesquisa;
4. Proposta de solução relacionado à cooperação Fog-Cloud;
5. Escolha das métricas que respondem as questões de pesquisa de cada experimento;
6. Estudo de funcionamento de linguagens, ferramentas e ambiente para desenvolvimento das soluções;
7. Verificar e validar o ferramental para cada solução;
8. Configuração do ambiente para a execução das tarefas;
9. Testes de execução;
10. Execução dos experimento;
11. Análise dos resultados e conclusões.

O procedimento metodológico é valido para ambos os experimentos, devidamente utilizados para cada experimento, que serão apresentados com maior detalhamento dentro de cada caso e experimento. A construção da metodologia vem para nortear a pesquisa e facilitar o desenvolvimento e entendimento do processo.

4 ARQUITETURA PROPOSTA

Com a revisão da literatura, observa-se que a camada *Fog* é uma adição a um sistema de **Cloud**, e mesmo com limitações devido a seu tamanho e hardware, pode atender às mais variadas demandas. Os dispositivos presentes também são bastante heterogêneos, possuindo como exemplo, smartphones, microcomputadores, roteadores, entre outros.

Dentre os dispositivos mais utilizados na *Fog*, destacam-se os computadores de placa única (ISIKDAG, 2015; KAUP et al., 2018), que têm sido aprimorados cada vez mais nos quesitos desempenho, custo, consumo e tamanho. Essa tecnologia acompanha, principalmente, a expansão da **IoT**, que é uma tecnologia muito difundida atualmente. Este cenário tem gerado interesse da comunidade, aumentando o esforço em pesquisas na área.

Esses dispositivos possuem poder computacional aproximado a um smartphone. Em certos modelos é encontrando até mesmo versões de sistemas operacionais baseados em Linux, o que facilita o acesso a diversas tecnologias disponibilizadas através da plataforma, mas que também não garante que todas as tecnologias são transportadas. Existem casos em que certas modificações se tornam necessárias por dificuldades relacionadas à arquitetura computacional, recursos e etc.

Dadas as especificidades das aplicações que utilizam streaming de dados, vemos vantagens em uma arquitetura *Fog-Cloud* para a coleta e processamento desses dados, tais como a redução de saltos de rede, o processamento mais próximo do ponto de coleta dos dados, melhoria na capacidade de resposta geral e a redução de sobrecarga na *Cloud* para armazenamento e processamentos de dados. Assim, este trabalho verifica a viabilidade da cooperação *Fog-Cloud* como uma solução para processamento, armazenamento e gerenciamento de dados, em um contexto de streaming de dados.

Neste capítulo é proposta uma arquitetura, baseada na cooperação *Fog-Cloud*, para trazer o processamento e gerenciamento dos dados para a *Fog* e reduzir a sobrecarga da *Cloud*. Pode-se assumir que o nó *Fog* tem poder computacional suficiente para manipular e capturar partes dos dados. Após o pré-processamento e enriquecimento dos dados, estes são enviados para a *Cloud* para armazenamento final.

A proposta do framework vem para possibilitar, por exemplo, a execução de serviços com baixa latência, redução de gastos com recursos na *Cloud*, redução na concorrência por recursos, melhora na escalabilidade da solução, dentre outros. Conforme demostrado na Figura 3, a camada *Fog* é composta por um nó *Fog*, com autonomia e poder de processamento local, e aplicações *Fog*, que utilizam dos dados processados e enriquecidos pelo nó *Fog* para uma resposta mais rápida para o usuário. O principal desafio é possibilitar o processamento dos dados recebidos na camada *Fog*, considerando as particularidades das aplicações de streaming de dados, e este deve ser o foco desta solução. Como segundo

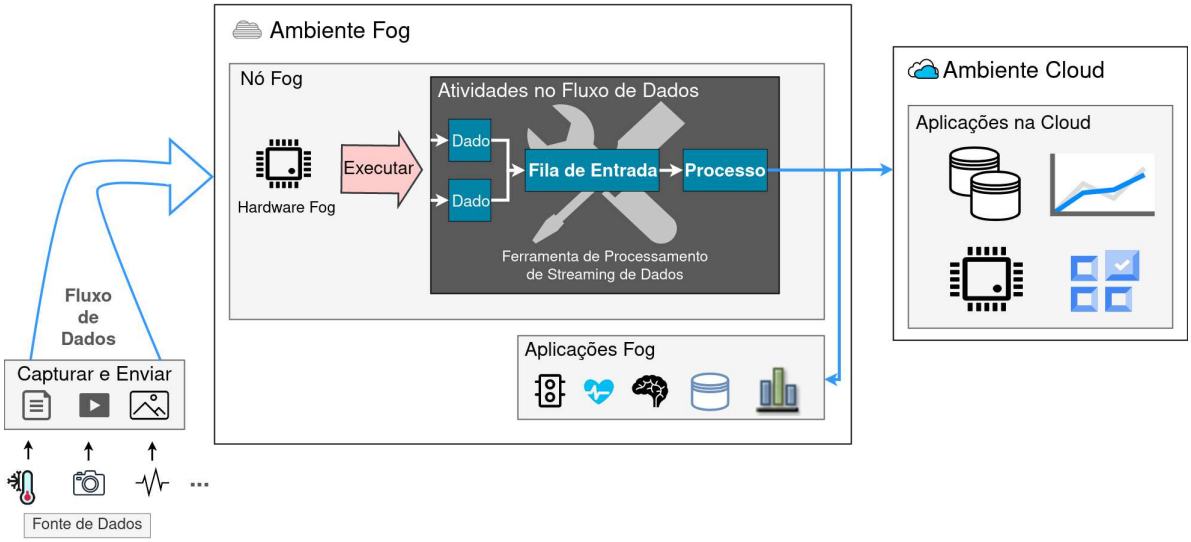

Figura 3 – Proposta de arquitetura com seus componentes e fluxo de dados.

ponto, é interessante que a solução seja escalável.

A Figura 3 fornece uma visão geral da camada *Fog*, com seus componentes principais e o fluxo que o dado deve percorrer dentro do sistema. No modelo foi definido um sistema de extração dos dados, em um primeiro momento, seguido da transformação através do processamento desses dados na *Fog*, para, por fim, realizar a entrega do resultado do processamento para a *Cloud*.

À esquerda é possível visualizar as fontes dos dados, que estão localizadas na borda e possuem como características principais a heterogeneidade e o volume de dados gerados. A arquitetura foi projetada para suportar fontes de dados diversas, como produção por parte de usuários, sensores, eventos, entre outros. Em geral, essas fontes de dados são referentes a dispositivos IoT.

As streaming de dados produzidas na borda são encaminhadas para o nó *Fog*, mais especificamente para o **Atividades No Fluxo de Dados**, que é capaz de processar e lidar com o fluxo dos dados gerados. Esta etapa é responsável por garantir os requisitos das aplicações, tais como restrições temporais e ordem de chegada dos dados. Em geral, os dados recebidos pelo **Atividades No Fluxo de Dados** contêm informações sobre sua fonte, seu tipo, data e hora da geração desse dado, dentre outros. Dependendo do tipo da fonte (sensores, dispositivos IoT, ou outros sistemas) o dado enviado pode ser simples, como um valor numérico, ou mais complexo, como um metadado que carrega diversas informações.

O **Atividades No Fluxo de Dados** é composto por duas subetapas: **Fila de Entrada** e **Processo**. É necessário definir uma **Fila de Entrada** para receber, organizar e armazenar temporalmente os dados. A **Fila de Entrada** é responsável por "ouvir" e armazenar os dados coletados em um banco de dados temporário até que eles sejam

processados. Dessa forma, deve-se evitar perdas de dados que podem impactar nas aplicações. Hoje é comum utilizar os sistemas de publisher/subscriber assíncronos. Como grande expoente desse tipo de ferramenta, encontra-se o Apache Kafka.

Posteriormente, os dados ingeridos são processados, no componente **Processo**, por meio da filtragem, limpeza e processamento na fila. Ele identifica valores ausentes e valores inesperados, seja porque é de um tipo diferente ou porque é um valor discrepante. Além disso, os dados podem ser enriquecidos com informações como data/hora da chegada do dado na camada *Fog*, tempo de duração do processamento, data/hora de envio para a *Cloud*, etc.

Em questão de hardware, a *Fog* tem a característica de ser um ambiente com poder computacional reduzido. É comum nesse ambiente a utilização de microcomputadores, computadores de placa única (como o Raspberry Pi) e pequenos servidores (JOHNSTON et al., 2018; NASIR et al., 2019). Os dados que saem do componente de processamento são encaminhados para o **Ambiente Cloud**, onde são armazenados para serem acessados por outras aplicações.

A caraterística principal atrelada a *Fog* é permitir o processamento de dados próximo aos elementos de borda. Com isso, existem ganhos relacionados a agilidade e disponibilidade, além, é claro, de evitar sobrecarga de trabalho e consumo de recursos computacionais na *Cloud*.

4.1 TECNOLOGIAS ADOTADAS NA CAMADA FOG

O objetivo do trabalho é trazer contribuições para pesquisas em *Fog Computing* e, ao mesmo tempo, procurar desenvolver uma abordagem acessível e de fácil reprodução. Dessa forma, contribuir com ferramentas relevante e, ao mesmo tempo, facilitar processo de reprodução do trabalho diminuir a complexidade da validação do experimentos, assim como da implementação também.

Os hardwares que atuam na camada *Fog* são hardwares, em geral, com menores capacidades computacionais que um servidor em *Cloud*, como smartfones, microcomputadores, roteadores, assistentes virtuais, computadores de placa única, entre outros. Destes, o que geram maior interesse devido a custo, consumo, poder computacional e o tamanho são os computadores de placas únicas que fazem muitos sentido para gerenciar dados de sensores (ISIKDAG, 2015; KAUP et al., 2018). Os computadores de placa única são computadores completos, com poder computacional reduzido em comparação a um sistema convencional, mas que são muitos úteis para diversas tarefas. Estes computadores são, geralmente, usados em sistemas de controle, alarmes, sistemas de medidas, dentre outros.

Exemplos de computadores de placa são o Banana Pi[1], LattePanda[2] e o Raspberry Pi[3], que é um exemplo de computador de placa única, é originalmente já pensado para a educação e inclusão digital (RICHARDSON; WALLACE, 2012). Dessa forma, é possível considerar que esse é um hardware ideal para realizar esse tipo de experimento dado seu custo reduzido e os recursos computacionais que ele oferece e a proposta dessa plataforma.

Com relação à linguagem de programação, existem algumas que podem ser utilizadas em diversas arquiteturas e plataformas computacionais, que são as linguagens interpretadas. Alguns exemplos muito importantes são o Python[4] e Java (GOSLING; MCGILTON, 1995), que são poderosas linguagens de alto nível e multiplataforma. Essas linguagens contam com um interpretador em tempo real, o que permite executar códigos Java sem restrições de compilação.

Além das facilidades relacionadas a forma que a linguagem foi pensada, Java é comumente utilizada em ferramentas no âmbito de processamento de streaming de dados, possuindo inclusive versões desenvolvidas para a linguagem. Isso facilita a exploração dessas ferramentas e diminui a curva de aprendizagem de uma nova linguagem, apesar de existir ainda a curva de aprendizado relacionada a aprender uma nova ferramenta, por isso essa foi a linguagem principal utilizada, com auxilio do Python para tarefas como a geração de gráficos e processamento dos resultados.

Outro aparato tecnológico adotado para facilitar o desenvolvimento e experimentação foi a utilização de contêineres (AHMED; PIERRE, 2018). Os contêineres são uma evolução baseada na tecnologia das máquinas virtuais, que abstraíam e comportavam mais de uma máquina dentro de um único hardware, fornecendo uma plataforma para executar diferentes aplicações e serviços, como se fossem em máquinas distintas. O porém dessa tecnologia era o grande consumo de recursos, pois o hardware simulava todas as camadas de um sistema operacional, então a experiencia era semelhante a ter múltiplos sistemas, mas consumindo o recurso equivalente a múltiplos sistemas operacionais.

O contêiner trabalha essa questão de forma mais otimizada, compartilhando recursos e algumas camadas de sistema, mas separando os recursos de cada contêiner diferente. Essa medida facilita a construção de serviços em uma única máquina física e diminui drasticamente o consumo de recursos, criando a possibilidade de modularizar cada aplicação para a execução de uma determinada tarefa sem acrescentar tanto nos custos computacionais.

A virtualização auxilia a utilização dos contêineres, eliminando dificuldades relacionadas a configurações de ambiente, o que facilita muito a reprodução de experimentos tanto para um sistema em *Cloud* quanto para a plataforma ARM apresentada no Raspberry Pi

[1] https://www.banana-pi.org/
[2] https://www.lattepanda.com/
[3] https://www.raspberrypi.com/
[4] https://www.python.org/

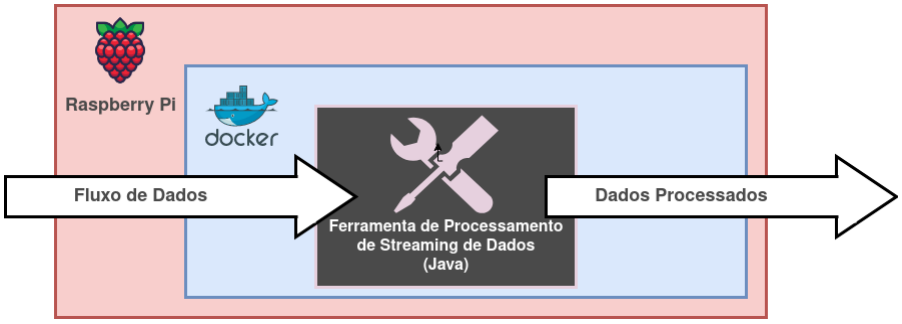

Figura 4 – Fluxo geral.

(AHMED; PIERRE, 2018; ALAM et al., 2018). Os softwares utilizados para contêineres são o ContainerD[5], CRI-O[6], Podman[7] e o mais famoso, difundido e conhecido deles, o Docker[8], que por esses motivos será o escolhido.

O fluxo básico dos dados implementado no framework está representado na Figura 4. Inicialmente, existe uma entrada de dados contínua e heterogênea chegando no Raspberry Pi, que está virtualizando uma instância de contêiner no Docker que mantém a execução de uma das FPSD. O dado é tratado e processado na ferramenta para que, posteriormente, seja encaminhado para seu destino, seja um banco de dados local ou na *Cloud*. Exemplos comuns de aproveitamento dessa saída de dados são a análise para tomada de decisão e armazenamento de dados enriquecidos em banco de dados.

Todo esse processo está sendo monitorado pelo software *Table of Processes* (top)[9]. O comando *top* apresenta a visualização em tempo real dos processos em execução no Linux e exibe as tarefas gerenciadas pelo kernel. O comando também fornece um resumo das informações do sistema que mostra a utilização de recursos, incluindo o uso de CPU e memória.

Como parte central do fluxo básico de execução do framework estão as ferramentas de processamento de streaming de dados (FPSD). A seleção das ferramentas é um fator importante para a implementação e avaliação do Framework. A escolha dessas ferramentas deve considerar a relevância para a academia e para a indústria.

Entre as ferramentas mais conhecidas para processamento de fluxo de dados temos por exemplo o Apache Spark, Apache Flink, Apache Storm, Apache Kafka Streams[10], Spring Cloud Data Flow[11], Amazon Kinesis[12] e o Google Cloud Dataflow[13]. Todas são

5 https://containerd.io/
6 https://cri-o.io/
7 https://podman.io/
8 https://www.docker.com/
9 https://man7.org/linux/man-pages/man1/top.1.html
10 https://kafka.apache.org/documentation/streams/
11 https://dataflow.spring.io/
12 https://aws.amazon.com/pt/kinesis/
13 https://cloud.google.com/solutions/smart-analytics

ferramentas poderosas e importantes, sendo algumas de grades empresas, outras de software livre, outras ainda com uma linguagem mais simples.

As ferramentas selecionadas foram três da ferramentas pertencentes à iniciativa Apache[14], que foram apresentadas nos dois trabalhos usados como base para os experimentos das FPSD (CHINTAPALLI et al., 2016) e (SHAHVERDI; AWAD; SAKR, 2019), além de bem sucedidas quando o cliente foi executado no ambiente do Raspberry Pi. As ferramentas escolhidas foram o Apache Spark, Apache Flink e Apache Storm. A escolha das ferramentas se deve também pela facilidade de implementação e a familiaridade com a linguagem Java. Características importantes relacionadas a essas ferramentas são:

Utilização em larga escala - É importante inserir ferramentas que sejam utilizadas dentro da academia e na indústria para que a avaliação traga maior contribuição. Dessa forma, é importante observar os trabalhos de pesquisa na área e tomar nota das ferramentas que estão sendo utilizadas para ter uma visão geral sobre o assunto e, dessa forma, poder elencar as ferramentas que são suscetíveis a entrar no trabalho.

Ferramenta aberta - É interessante priorizar a utilização de tecnologias livres e de código aberto (HIPPEL, 2001). Essa medida viabiliza a avaliação e reprodução do processo científico. As ferramentas de código aberto também possuem participação da comunidade, o que torna a ferramenta mais robusta com o tempo. Além disso, existem ferramentas abertas de interesse disponíveis no mercado e largamente usadas, como, por exemplo, as oferecidas pela Fundação Apache (SEVERANCE, 2012).

Multiplataforma - Uma vez que a camada *Fog* é caracterizada como um ambiente heterogêneo, as ferramentas serem classificadas como Multiplataforma é um ponto interessante de ser abordado. Essa tecnologia entra em acordo com a forma que a *Fog* se apresenta.

A base da maior parte das ferramentas relacionadas a processamento de streaming de dados é a funcionalidade chamada de MapReduce (DEAN; GHEMAWAT, 2008), que é um algoritmo de divisão e conquista, que separa os dados, processa e os junta ao final para encontrar o resultado. Essa característica torna o algoritmo altamente escalável, acelerando os processos, dependendo do problema. Este algorítimo ficou muito conhecido quando apresentado no software Apache Hadoop.

O projeto Apache Hadoop desenvolve software de código aberto para computação distribuída escalável. A biblioteca Hadoop é uma estrutura que permite o processamento distribuído de grandes conjuntos de dados em clusters de computadores usando modelos de programação simples. Ele foi projetado para escalar desde servidores únicos para milhares de máquinas, inclusive em máquinas heterogêneas. Em vez de depender de hardware para fornecer alta disponibilidade, a própria biblioteca foi projetada para detectar e lidar com

[14] https://www.apache.org/

falhas na camada de aplicação, fornecendo um serviço altamente disponível em um cluster de computadores.

Depois do Hadoop, surgiram diversas ferramentas que seguem propósitos semelhantes, com evoluções e adaptações para lidar com quantidades massivas de dados. Dentre elas, as FPSD, que lidam com dados massivos na forma de fluxo, como as ferramentas escolhidas também são enquadradas. A seguir um pouco mais sobre cada uma das ferramentas selecionadas.

4.1.1 Apache Storm

O Apache Storm foi pioneiro no desenvolvimento de sistemas de processamento de fluxos de dados distribuídos e tolerantes a falhas. Em dezembro de 2010, Nathan Marz teve a ideia de desenvolver um sistema de processamento de fluxo que pode ser apresentado como um único programa. Essa ideia resultou em um novo projeto chamado Storm. Ele se torna parte da Família Apache em 2014 (IQBAL; SOOMRO et al., 2015).

É um sistema de computação em tempo real distribuído gratuito e de código aberto. O Apache Storm facilita o processamento de fluxos de dados contínuos, fazendo para o processamento em tempo real o que o Hadoop fez para o processamento em lote. O Apache Storm é uma ferramenta simples e pode ser usado com qualquer linguagem de programação.

A ferramenta possui uma arquitetura Master/Slave onde o nó mestre é chamado de Superviewers e os nós de trabalho são chamados de Nimbus. O nó mestre é responsável por atribuir as tarefas aos diferentes nós de trabalho, além de monitorar a execução de todo o trabalho. Toda a comunicação de mensagens entre o Nimbus e os Supervisores é gerenciada por meio de um cluster Apache Zookeeper[15] (SHAHVERDI; AWAD; SAKR, 2019).

4.1.2 Apache Spark

O Spark surgiu como um projeto de pesquisa no AMPLab de Berkeley em 2009. Seu objetivo era observar as falhas do Hadoop MapReduce e desenvolver um sistema que conseguisse lidar com os dados de forma interativa e iterativa. A ferramenta se tornou *open source* em 2010 e entrou para a Fundação Apache em 2013. Desde o início, o Spark contou com uma comunidade muito ativa, contando com mais de 2.000 colaboradores, superando projetos como o próprio Hadoop (LUU, 2018).

O Apache Spark é um mecanismo multilíngue para executar engenharia de dados, ciência de dados e aprendizado de máquina em máquinas ou clusters de nó único, projetado especificamente para lidar com grandes quantidades de dados. Em termos de flexibilidade,

[15] https://zookeeper.apache.org/

o Spark fornece interfaces de programação de aplicativos (APIs) de alto nível em Java, Scala[16], Python, R[17] e shell interativo (GARCÍA-GIL et al., 2017).

Ao contrário da computação baseada em disco do Hadoop, o Spark executa a computação em memória, introduzindo sua estrutura chamada de dados distribuídos resilientes (Resilient Distributed Datasets em inglês, ou RDD), que funciona como um tabela. Como é possível armazenar resultados intermediários na memória, o sistema se torna mais eficiente para operações iterativas. Em termos de desempenho, o Spark é mais rápido que o Hadoop (GARCÍA-GIL et al., 2017).

O sistema Spark começou então a disponibilizar uma extensão de API que adiciona suporte para processamento de fluxo contínuo. Inicialmente, o streaming do Spark contava com o mecanismo de processamento de micro lote que coleta os dados que chegam dentro de um determinado período de tempo e executa um algoritmo no lote que possui os dados coletados. Durante a execução da tarefa em lote, o conjunto de dados para o próximo mini lote é coletado. Portanto, pode ser considerado como um mecanismo de processamento em lote com janela de tempo controlada para processamento de fluxo (SHAHVERDI; AWAD; SAKR, 2019).

4.1.3 Apache Flink

O projeto de pesquisa Stratosphere visava construir uma plataforma de análise de big data para a próxima geração, que permitiria analisar grandes quantidades de dados de forma gerenciável e declarativa. Em 2014, o Stratosphere se tornou open-source com o codinome Flink, como um projeto da Apache Incubator[18] e se tornou parte do projeto Apache Top Level no mesmo ano (ALEXANDROV et al., 2014).

O Apache Flink é uma ferramenta de processamento distribuído para computar fluxos de dados. O Flink foi projetado para ser executado em todo o tipo de ambientes de cluster, realizar cálculos com a mesma velocidade da memória e em qualquer escala. As APIs oferecem ao programador operadores genéricos como Join, Cross, Map, Reduce e Filter, o que difere o Flink do Hadoop MapReduce, que permite apenas que operadores complexos sejam implementados como uma sequência de fases de mapa e redução (RABL et al., 2016).

O estado da tarefa é sempre mantido na memória ou, se o tamanho do estado exceder a memória disponível, em estruturas de dados em disco com acesso eficiente. Portanto, as tarefas executam todos os cálculos acessando o estado local, produzindo latências de processamento muito baixas. O Flink garante consistência de exatamente um

[16] https://www.scala-lang.org/
[17] https://www.r-project.org/
[18] https://incubator.apache.org/

para o caso de falhas, que remove a possibilidade de um dado ser contabilizado mais de uma vez, verificando periodicamente e de forma assíncrona o estado local para o armazenamento.

4.1.4 Tabela Comparativa

A Tabela 1 sintetiza parte das informações sobre as FPSD selecionadas. Todas as ferramentas são importantes dentro do contexto desta pesquisa, contando uma história que ainda é recente. Considerando o desenvolvimento do framework proposto e as ferramentas selecionadas, no próximo capítulo iremos avaliar a proposta deste trabalho através de dois experimentos. Assim, pretendemos verificar a viabilidade da arquitetura, bem como analisar quais ferramentas são mais adequadas para o processamento de streaming de dados em uma camada *Fog*.

Tabela 1 – Ferramentas de processameto de streaming de dados.

Ferramenta	**Ano**	**Criador**	**Suporte de Linguagem**
Storm	2011	BackType	Java, Thrift
Spark	2013	UC Berkely	Java, Scala, Python, R
Flink	2015	TU-Berlin	Java, Scala, Python, SQL

5 AMBIENTE E RESULTADOS EXPERIMENTAIS

Neste capítulo são apresentadas as avaliações realizadas do framework proposto através da condução de dois experimentos. O primeiro objetiva avaliar a viabilidade técnica da solução e a performance das ferramentas para o processamento de streaming de dados em uma camada *Fog*. No segundo experimento o framework foi aplicado em um contexto real para avaliar a sua utilização em uma aplicação que necessita de uma solução viável e escalável.

Em síntese, esta avaliação pretende discutir, experimentar e verificar a utilização de soluções de aplicação de Ferramentas de Processamento de Streaming de Dados (FPSD) sobre a visão de um sistema de computação em *Fog* dentro de uma visão teórica e prática, e, dessa forma, instigar a comunidade científica a explorar este caminho.

5.1 EXPERIMENTO I

O primeiro experimento foi realizado através da execução de tarefas específicas considerando as soluções utilizadas no desenvolvimento da camada *Fog*. Neste experimento, foram utilizadas as ferramentas de processamento de streaming de dados e verificamos a capacidade de processamento da camada *Fog*. Para tal, ocorreram testes e coletas de dados sobre o comportamento do sistema durante a execução de tarefas pré-determinadas no âmbito de FPSD. Com os resultados foi possível mensurar o poder computacional de um sistema comum de *Fog* por meio de comparação com resultados obtidos em um ambiente considerado como *Cloud*.

5.1.1 Escopo da Avaliação

Apesar das FPSD já estarem presentes em sistemas computacionais nos últimos anos, existe aqui uma interrogação quando a proposta é o uso de outras plataformas. A arquitetura de um hardware na *Fog* pode diferir da arquitetura convencional X86[1,2] que é apresentada na maioria dos sistemas computacionais voltados a performance. A arquitetura ARM (JAGGAR, 1997), como a encontrada no Raspberry Pi, possui instruções diferentes a nível de máquina. Assim, o que é compilado em um sistema provavelmente não funciona no outro, pois a "receita de bolo está no idioma errado".

A primeira preocupação então é verificar o comportamento das ferramentas nesse ambiente. Um bom início é contar com a tecnologia das linguagens interpretadas (SCOTT, 2000), que funcionam em tempo de execução, pois transcrevem as instruções para linguagem

[1] https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/x86-architecture
[2] https://cs.lmu.edu/ ray/notes/x86overview/

de máquina durante este período. É claro que existe um custo, que normalmente é a perda de desempenho. Nesse sentido, temos linguagens como Python, R , e Java[3].

Outro questionamento é relacionado a incógnita de como essas ferramentas vão se comportar na camada *Fog*, que conta com recursos limitados. Nesse contexto, o experimento busca avaliar como as FPSD se comportam dentro de uma camada de *Fog* computacional. Como descrito no Capítulo 4, seção 4.1, o hardware escolhido é o **Raspberry Pi** (RICHARDSON; WALLACE, 2012), um dispositivo bastante utilizado pela comunidade científica neste contexto de estudo. O objetivo é avaliar o funcionamento das ferramentas em um hardware convencional de *Fog* e avaliar questões como consumo de recursos computacionais, resultados absolutos e complexidade de utilização. Posteriormente, é importante traçar um paralelo entre os resultados obtidos na *Fog* em comparação com uma sistema mais próximo de um ambiente *Cloud*.

5.1.2 Questões de pesquisa do Experimento I

Essa avaliação seguiu o modelo GQM (Goal - Question - Metric) (KITCHENHAM; CHARTERS, 2007), que determina que os objetivos do estudo devem ser definidos, seguidos das questões de pesquisa e métricas para avaliação das questões de pesquisa. O escopo desta avaliação e o Objetivo são descritos a seguir:

"**Analisar** as ferramentas de processamento de streaming de dados **com o objetivo de** verificar a viabilidade da sua utilização **em relação as** restrições computacionais **sob o ponto de vista de** um hardware não convencional **no contexto de** um ambiente computacional *Fog*."

Com base no escopo e no objetivo desta avaliação, foram definidas uma questão de pesquisa primária (QPP) e duas questões de pesquisa secundárias (QPS) foram derivadas:

QPP - ***O hardware disponível em uma camada Fog é capaz de suportar as operações de Ferramentas de Processamento de Streaming de Dados, considerando as limitações deste hardware?***

Essa questão avalia a competência do hardware utilizado em camadas *Fog* em entregar os recursos que sejam suficientes para executar as FPSD em conjunto com as tarefas propostas. A questão vai de encontro ao objetivo desse experimento, que é verificar o comportamento dessas ferramentas na ótica da camada *Fog*, que tem suas limitações com relação a hardware. Essa é a pergunta central desse experimento.

QPS1 - Como se dá o consumo dos recursos computacionais pelas FPSD na camada *Fog*?

[3] https://www.java.com/en/

Essa questão tem o objetivo de avaliar o consumo de recursos computacionais, como utilização de processador e memória, para avaliar como o hardware disponível em uma camada *Fog* lidou com as FPSD durante a execução do experimento.

A resposta a essa questão de pesquisa também serve de guia para o planejamento de um sistema semelhante, uma vez que, com o consumo de recursos utilizados e os requisitos de algum sistema semelhante é possível estimar a necessidade de hardware.

QPS2 - Qual tipo de comparação pode ser sugerida entre os resultados obtidos dentro de um sistema de computação *Fog* e uma camada de computação em *Cloud*?

Com essa questão pretendemos comparar, de certa forma, a execução das mesmas tarefas em uma camada *Fog* e em uma camada *Cloud*. Dada as particularidades dessas camadas, o objetivo não é identificar qual é a melhor, mas trazer indicadores que nos permitam analisar a performance de cada uma delas.

Segundo Kitchenham e Charters (2007), o fluxo de objetivos para métricas na metodologia GQM pode ser visualizado através de um grafo direcionado. O fluxo começa no nó objetivo e passa pelos nós que representam as questões, chegando aos nós métricos. A Figura 5 apresenta o fluxo neste experimento. As métricas definidas para responder as QP são:

M1: Consumo de CPU durante a execução das FPSD na *Fog*.

M2: Consumo de Memória durante a execução das FPSD na *Fog*.

M3: Volume (Quantidade) de elementos processados pelas FPSD na *Fog*.

M4: Análise comparativa do consumo de CPU durante a execução das FPSD na *Fog* e na *Cloud*.

M5: Análise comparativa do consumo de Memória durante a execução das FPSD na *Fog* e na *Cloud*.

M6: Análise comparativa do volume (Quantidade) de elementos processados pelas FPSD na *Fog* e na *Cloud*.

Na Figura 5 existe a relação entre as questões de pesquisa e as métricas adotadas na condução dos experimentos, tanto a nível de *Fog* quanto a nível de *Cloud*, ramificando a questão de pesquisa principal em duas secundárias, e, finalmente, definindo as métricas que respondem a essas questões.

Para analizar a QPP, é preciso responder como o sistema lidou com as FPSD, que é a QPS1, que pode ser avaliada através das métricas de 1 a 3. Com a QPS1 respondida,

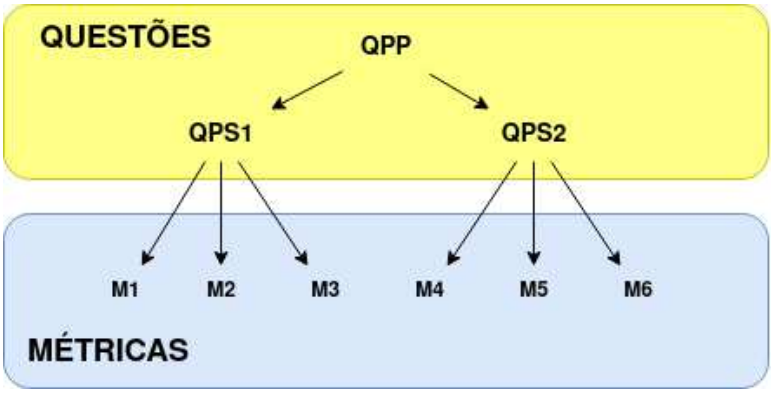

Figura 5 – Grafo direcionado representando a abordagem adotada nesta avaliação.

precisamos estabelecer uma conexão com um sistema convencional de *Cloud*. Isso é tratado quando as métricas de 4 a 6 são comparadas com os valores obtidos com as métricas 1 a 3, respondendo a QPS2.

5.1.3 Etapas de condução da Avaliação

Dois trabalhos foram selecionados para serem fonte de inspiração com os resultados obtidos com esta pesquisa. Os trabalhos de Chintapalli et al. (2016) e Shahverdi, Awad e Sakr (2019) realizam pesquisas no âmbito de avaliação de desempenho de FPSD em clusters computacionais, que são um ambiente mais convencional e é a estrutura adotada na *Cloud*. O objetivo é trazer uma nova visão destes experimentos para a camada *Fog* e verificar o poder computacional disponível dentro desse sistema. Como o hardware tem suas limitações, o consumo de recursos computacionais também é importante na nossa avaliação.

Com base nesses trabalhos Chintapalli et al. (2016), Shahverdi, Awad e Sakr (2019), foi desenvolvida uma metodologia semelhante a fim de manter um padrão e tornar o trabalho comparável, apesar das variações esperadas nos resultados em função das diferenças entre os hardwares utilizados.

A Figura 6 apresenta o fluxo de execução do experimento com cada uma das etapas, bem como as tecnologias adotadas na implementação da solução. As etapas são:

Etapa (i): Foi desenvolvido um produtor Kafka, que produz valores continuamente que devem ser consumidos pelo framework. Os elementos produzidos são números naturais dentro de um intervalo (de 0 a 100) e vão ser utilizados para em um algoritmo que conta ”campanhas”, que resumindo, é um mapeamento que conta a quantidade de elementos para cada número do intervalo dentro de uma janela de tempo.

Etapa (ii): Os dados massivos são alocados em um sistema *publisher/subscriber* da

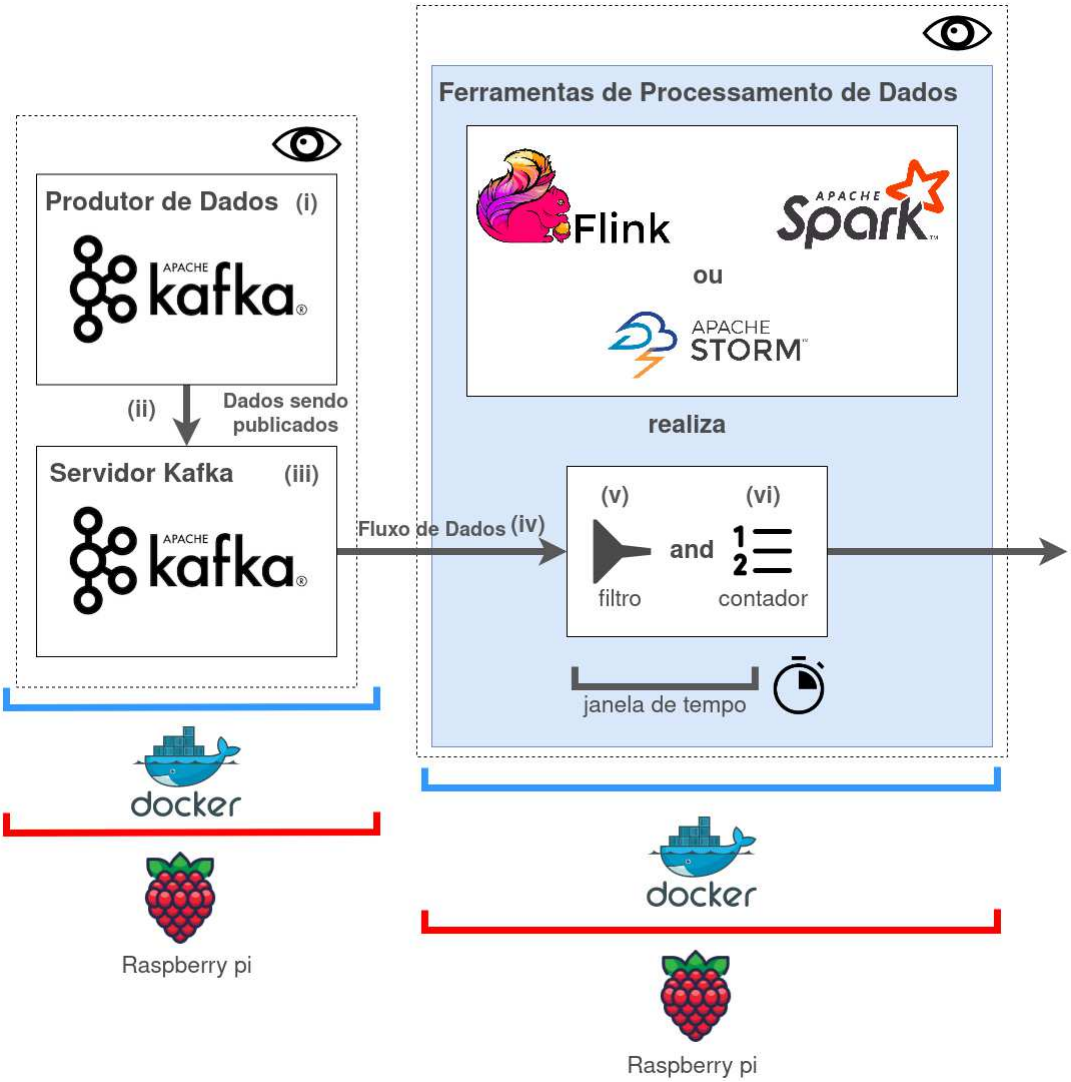

Figura 6 – Framework para o Raspberry Pi.

ferramenta Kafka para que sejam coletados pela ferramenta selecionada.

Etapa (iii): Os dados se acumulam de forma que o sistema Kafka seja capaz de alimentar o framework durante todo o período de execução com o maior fluxo possível.

Etapa (iv): A ferramenta selecionada entra em atividade, requisitando dados do sistema Kafka, recebendo o maior fluxo possível que o hardware do Raspberry Pi consegue suportar.

Etapa (v): O processo dentro da ferramenta selecionada verifica a validade do dado, ou seja, se ele pertence aos naturais dentro do intervalo determinado.

Etapa (vi): Finalmente agrupa os elementos, realizando a contagem de elementos dentro da janela de tempo estipulada.

Na Figura, o desenho do olho é para lembrar que durante todo o processo as ferramentas são avaliadas e dados são coletados através da ferramenta top (*Table of*

Processes) e armazenados em um arquivo, que pode gerar algum impacto nos resultados. Todo esse sistema está apoiado sobre um sistema Docker, que facilita a execução e a reprodução do experimento. O sistema de produção e manutenção de dados está alocado em um Raspberry diferente do de processamento de fluxo de dados, evitando esforço desnecessário e mantendo a maior disponibilidade de recursos possível para o framework.

As FPSD executam diferentes processamentos sobre os dados. Assim, para analisá-las de forma igualitária, foi desenvolvido um algoritmo que pudesse ser executado em todas as ferramentas selecionadas de forma simplificada. No ambiente das FPSD existe um exemplo base, que, nesse caso, se trata do "Word Count", que realiza a contagem de repetição de elementos dentro de uma streaming de dados. O algoritmo desenvolvido é bastante próximo do "Word Count", baseado nos trabalhos de (CHINTAPALLI et al., 2016) e (SHAHVERDI; AWAD; SAKR, 2019), como discutido anteriormente.

Cada ferramenta possui questões únicas de configuração e uma gama de funções próprias, tornando o código diferente pra cada uma, mesmo que em alguns casos sejam semelhantes. Mesmo assim, depois da tarefa de configuração, dada as devidas adaptações, a ideia base do algoritmo para resolver o problema é comum entre elas.

Algoritmo 1: Contador de Campanhas

```
1  enquanto houver entrada de fluxo de dados faça
2      enquanto estiver dentro da janela de tempo faça
3          Continue lendo;
4          se o dado for válido então
5              se a campanha já foi representada então
6                  Atualiza o valor de ocorrências de determinada campanha no para a
                    quantidade de aparições anterior com a adição de um;
7              senão
8                  Adiciona mais uma campanha ao mapa com uma aparição;
9              fim se
10         senão
11             Descarta o dado;
12         fim se
13     fim enquanto
14 fim enquanto
```

O pseudocódigo apresentado no Algoritmo 1 representa os passos básicos comuns a todas as ferramentas, que em alguns casos possuem uma estrutura própria para a otimização do algoritmo. O algoritmo é o "Hello World" para as FPSD, então uma implementação bem próxima do que foi desenvolvida para o problema é encontrada no site das ferramenstas, como exemplo, o que facilita a validação. Além disso, o algoritmo realiza as principais atividades que uma FPSD executa, o MapReduce e o controle de fluxo de dados.

O algoritmo se inicia com dois loops condicionais, com o primeiro deles (linha 1)

validando a repetição enquanto ainda houver um fluxo de dados ativo, e o segundo (linha 2) controlando a janela de tempo específica. É dentro de cada janela de tempo que as campanhas são contabilizadas.

A primeira linha logo após os dois loops condicionais (linha 3) continua a leitura dos dados, e na linha 4 temos uma instrução condicional que funciona como um filtro, verificando se o dado de entrada é válido como dentro de uma campanha.

No segundo condicional (linha 5) é verificado se já existe uma representação daquela campanha durante a janela de tempo. Se sim (linha 6) atualiza o contador para o número anterior com acréscimo de um e, senão (linha 7), o próximo passo (linha 8) é criar um novo contador de aparições para a nova campanha.

Finalizando o condicional mais interno (linha 9) e no próximo comando (linha 10) acontece a negação do filtro de validade da linha 4 da entrada de dados, que descarta o dado inválido (linha 11) e, finalmente, encerrando o condicional (linha 12). Por último são finalizados os dois loops condicionais (linha 13 e 14).

5.1.4 Resultado Experimentais na camada Fog

O hardware utilizado para a realização dos experimentos na camada de computação *Fog* foi um cluster com dois Raspberry Pi 4 b[4], possuindo 4GB de memória RAM e um processador Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz, conectados em uma rede local do tipo Gigabit Ethernet. A Figura 7 mostra uma foto do cluester de Raspberry Pis, que é o hardware utilizado.

A Figura 8 apresenta o modelo esquemático do cluster de Raspberrys Pi utilizado para a realização do experimento, Ele contém duas unidade de Raspberry Pi modelo 4b, um modelo 3[5] e um modelo 2[6]. Os dois modelos mais simples não se sairam bem ou não executaram as ferramentas de formas satisfatória, então apenas as duas unidades de Raspberry Pi modelo 4b foram utilizadas, por possuírem mais recursos computacionais.

Os dois modelos utilizam o Raspberry Pi OS - 64bits[7], que é baseado no Debian[8], uma distribuição do Linux (TORVALDS, 1999). Todos os códigos, algoritmos, configurações e informações necessárias para a reprodução do experimento estão disponibilizados no repositório no GitHub [9].

O experimento inicia com a produção de dados no produtor Kafka para uma instância do Kafka como servidor do tipo *publisher/subscriber* no mesmo Raspberry Pi.

[4] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[5] https://www.raspberrypi.com/products/raspberry-pi-3-model-b/
[6] https://www.raspberrypi.com/products/raspberry-pi-2-model-b/
[7] https://www.raspberrypi.com/software/
[8] https://www.debian.org/
[9] https://github.com/lucaslarcher/data_streaming_benchmark

Figura 7 – Cluster de Raspberry Pi.

Os dados são produzidos por um determinado período de forma que o montantes de dados sejam o suficiente para serem consumidos por qualquer ferramenta durante seu tempo de execução.

Um Raspberry Pi executa uma das ferramentas selecionadas, com o monitoramento da ferramenta top e guarda as informações relevantes. Essa execução tem a duração de dez minutos e o resultado é armazenado. A execução também gera um log reportando a quantidade de elementos processados a partir de um intervalo determinado. Os procedimentos são realizados sobre instâncias do Docker, para facilitar a execução e a reprodução dos testes, conforme descrito na Seção 5.1.3.

Os dados obtidos durante o experimentos dizem respeito ao consumo de CPU da ferramenta no sistema, ou seja, o quanto o processador está trabalhando, o consumo de

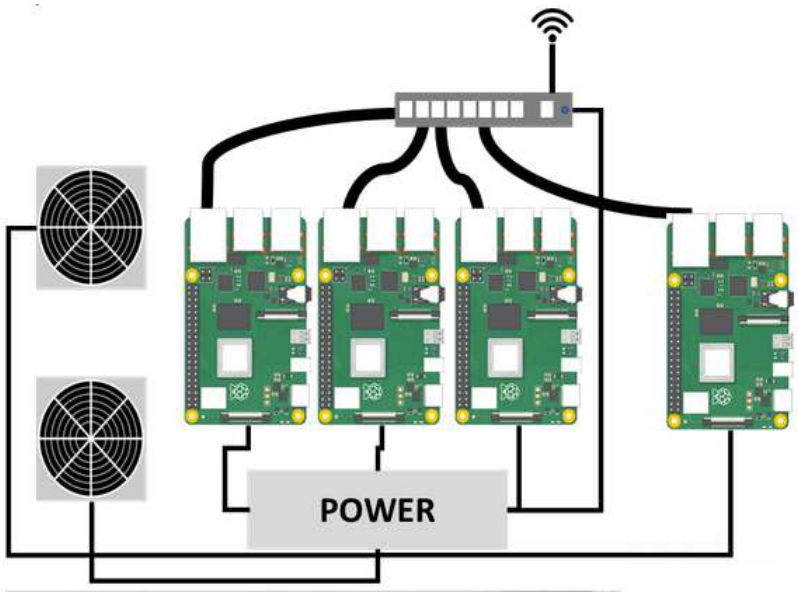

Figura 8 – Modelo Esquemático do Cluster.

memória RAM, que indica o quanto as operações consomem de memória, e a contagem de elementos processados durante todo o período (Métricas M1, M2 e M3). Com esses dados é possível observar como o Raspberry se comporta enquanto cada ferramenta está executando o processamento sobre os dados.

No gráfico apresentado na Figura 9 é possível analisar o consumo de CPU (Métrica M1). Em todas as ferramentas verificamos um consumo alto e constante de CPU, na maior parte do tempo acima do 80%. Isso revela o quanto essas tarefas são custosas para o Raspberry Pi utilizado na *Fog*.

Restringindo a Figura 9 para valores superiores a 80% temos a Figura 10, onde podemos observar quais ferramentas consomem mais CPU. Em ordem decrescente de consumo, o Flink é a ferramenta com maior consumo de CPU, seguido do Spark e por último o Storm, sendo o Storm o sistema mais instável nesse quesito e o Spark mais estável. Com essa primeira análise, podemos afirmar que é possível o uso do Raspberry para a execução das ferramentas, visto que todas foram capazes de executar a tarefa especificada. Porém, não é possível concluir se há uma ferramenta mais eficiente do que a outra visto que as três ferramentas tiveram um comportamento semelhante em termos de CPU.

Analisado o consumo de memória RAM (Métrica M2) utilizado pela ferramenta através da Figura 11, temos um cenário bem diferente. Nesse caso, o consumo de memória RAM é abaixo dos 40% para as ferramentas, com o Spark estando abaixo dos 20% e o Flink e o Storm muito próximos. Dentre as ferramentas, o Flink deveria ser o maior consumidor neste aspecto, pois sua característica principal é trabalhar sobre a memória RAM. É interessante pontuar a quantidade de memória que o Spark está consumindo, que

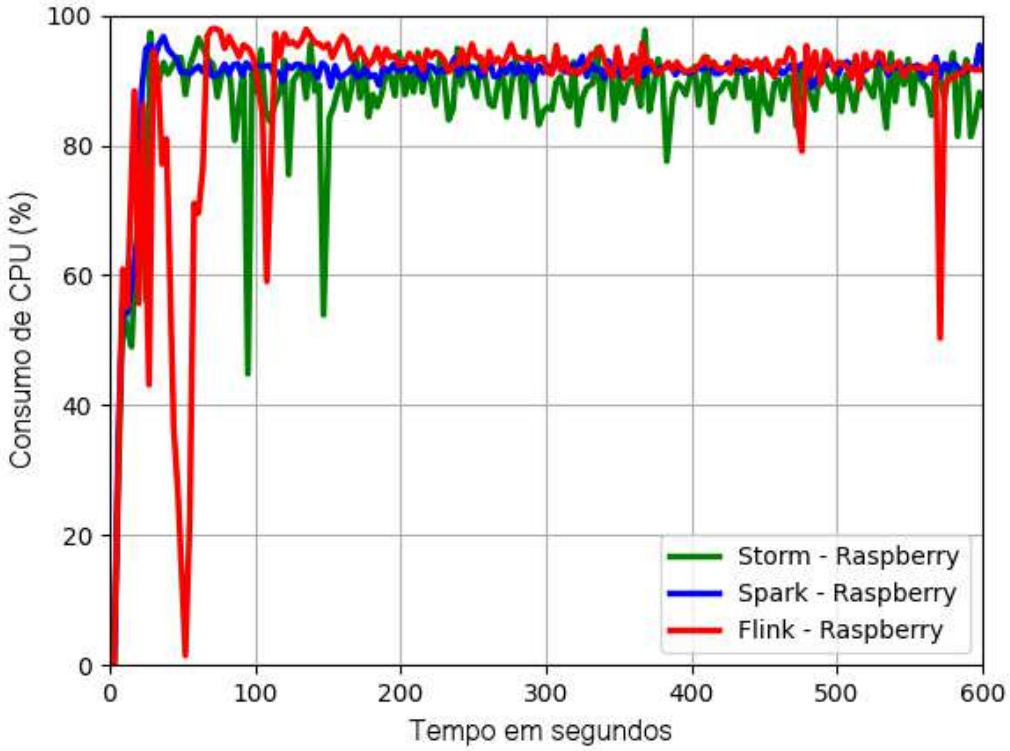

Figura 9 – Consumo de CPU no Raspberry Pi.

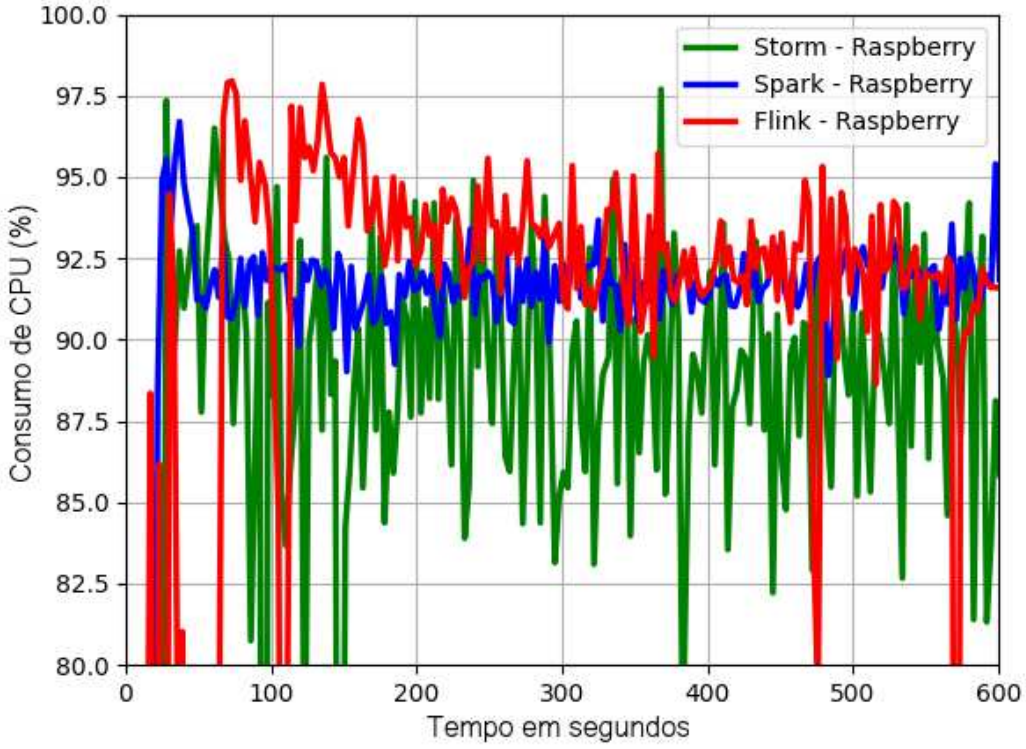

Figura 10 – Consumo de CPU no Raspberry Pi (Acima de 80%).

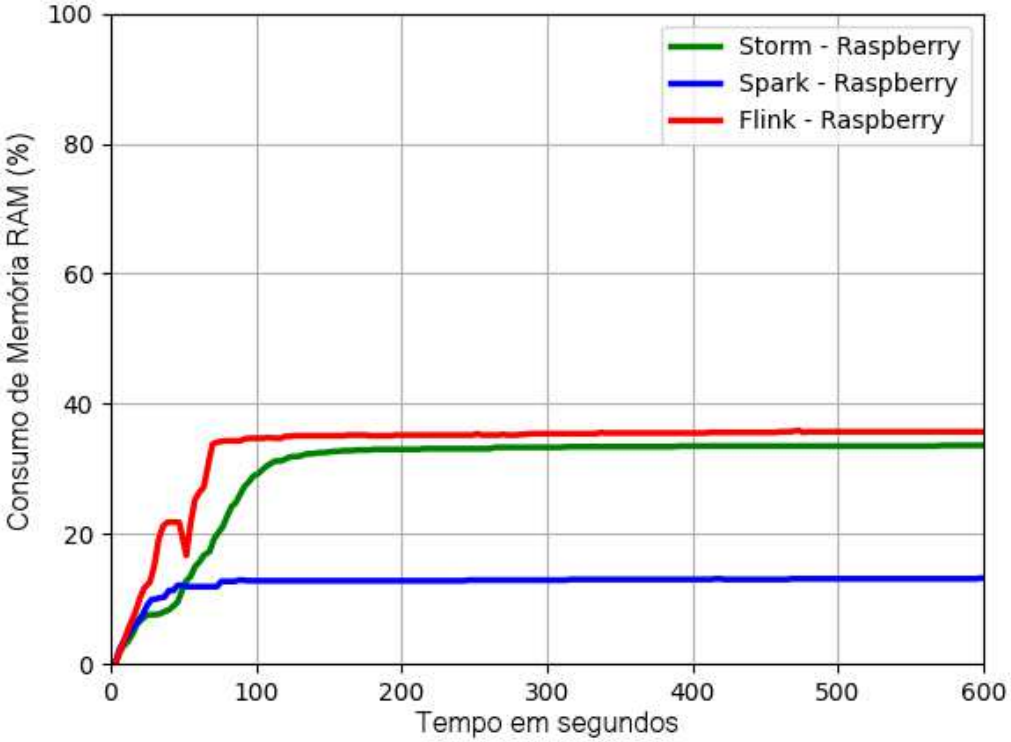

Figura 11 – Consumo de Memória RAM no Raspberry Pi.

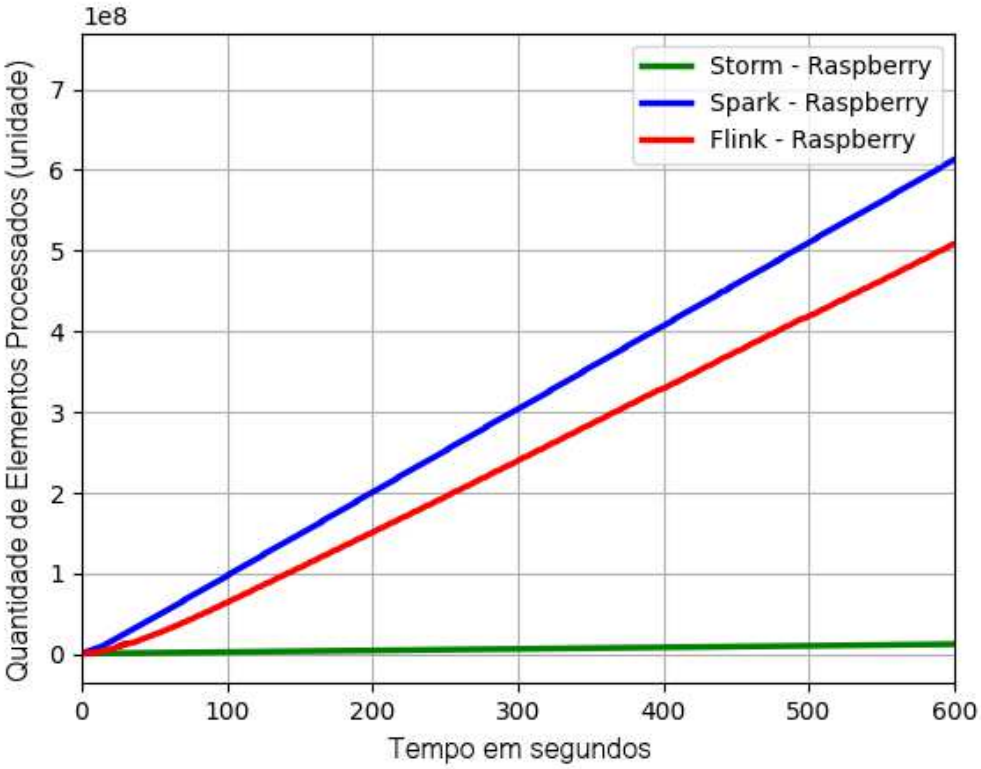

Figura 12 – Elementos Processados no Raspberry Pi.

está bem abaixo das outras ferramentas e pode ser um bom indicativo dependendo das condições e dos diferentes hardwares presentes na *Fog*.

Durante a análise do número de elementos processados (Métrica M3), apresentado na Figura 12, verificamos a diferença entre a quantidade de elementos processados durante o tempo de processamento de cada uma das ferramentas. Na frente são apresentados o Spark e o Flink, com o Spark na frente com uma pequena margem de vantagem.

A ferramenta Storm atinge a casa de dezenas de milhões de elementos processados, bastante abaixo das outras duas ferramentas, que por sua vez chegam a centenas de milhões de elementos processados. Os resultados apresentados na execução do Storm foram verificados em múltiplas configurações e se mantiveram.

Nesse momento, é importante lembrar que no comparativo de consumo de CPU, não houve uma ferramenta que se sobressaiu sobre as outras e, quanto ao consumo de memória RAM, o Spark foi melhor em relação às outras ferramentas. O resultado do Spark no quesito memória RAM e elementos processados o torna melhor resultado para o experimento.

É importante reiterar que todas as execuções foram revalidadas por pelo menos de três execuções, variando a quantidade de raspberrys executando ao mesmo tempo em paralelo e diferentes modelos de raspberry, sendo eles o 2, 3 e 4, para avaliar o resultado que seria melhor apresentado como resposta. A partir destes resultados não foi verificado irregularidade ou variação significativa em relação aos resultados obtidos.

Quando a tarefa foi realizada simultaneamente em dois Raspberrys Pi 4 b, recebendo os dados de mesmo sistema e os resultados se repetiram novamente, o que dá indícios a respeito da escalabilidade do sistema, dependendo da aplicação realizada. O resultado final corresponde a um sistema executando em Raspberry Pi 4 b, executando a mesma tarefa com as devidas bibliotecas e arquitetura de cada ferramenta.

5.1.5 Resultados Experimentais na camada Cloud

Com intuito de responder a **QPS2**, utilizamos as mesmas ferramentas e o Algoritmo 1 para replicar o experimento conduzido anteriormente. Porém, foi utilizado um hardware com potencial de processamento consideravelmente superior ao utilizado na *Fog*. Para tal, neste experimento, foi utilizada a infraestrutura da Rede Integrada de Pesquisa em Alta Velocidade (RePesq)[10] em um primeiro momento, mas devido as especificações de hardware disponibilizados para o trabalho no ambiente, a escolha foi feita por um sistema mais poderoso que o o grupo de pesquisa tem acesso, onde contamos com um Ryzen 7 1700[11] com 8 núcleos e 16 threads e 16GB de memória RAM. Como os resultados relacionados

[10] https://www.repesq.ufjf.br/
[11] https://www.amd.com/pt/products/cpu/amd-ryzen-7-1700

à CPU e memória RAM estão avaliados percentualmente, é possível realizar mais uma análise comparativa, verificando como o consumo de recurso das ferramentas acontece em cada um dos ambientes.

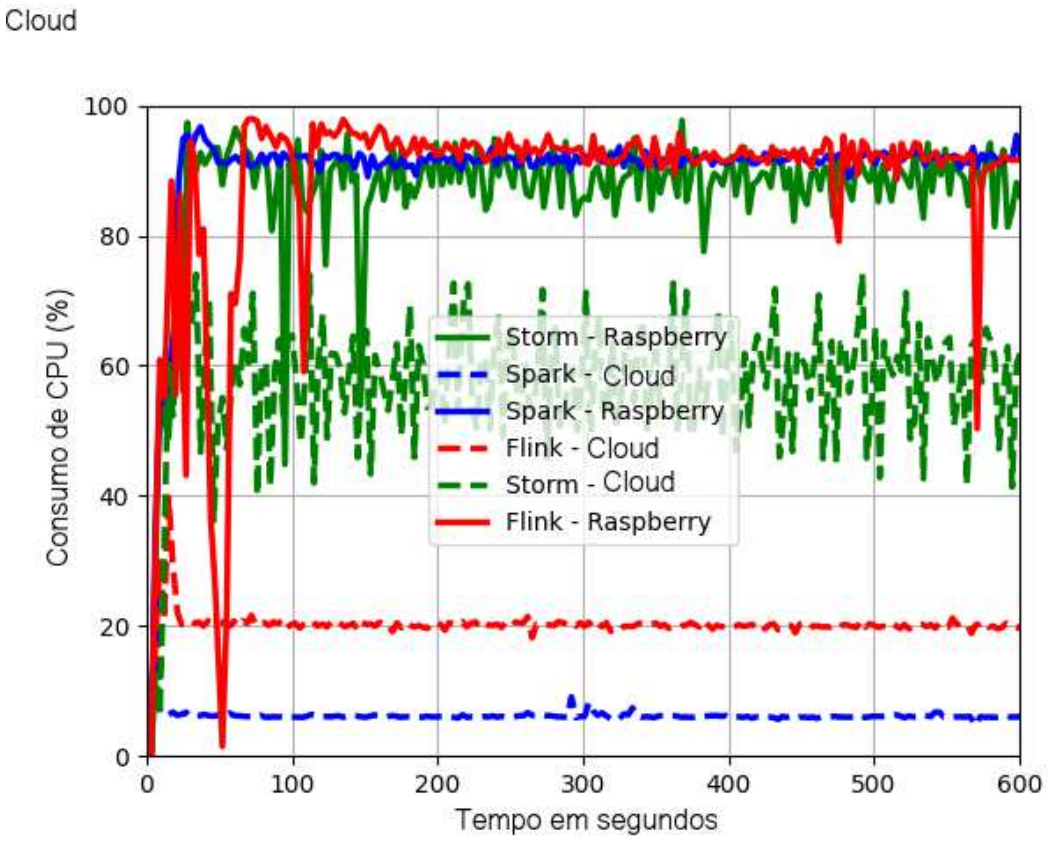

Figura 13 – Consumo de CPU em ambos os sistemas.

Assim como no experimento anterior, os dados são gerados e mantidos por um outro sistema com uma instância de Kafka e também um produtor Kafka. Neste experimento, o produtor foi configurado em um Ryzen 5 1400[12] com 4 núcleos e 8 threads e 16GB de memória RAM. O sistema possui uma rede Gigabyte, garantindo uma conexão rápida entre as máquinas.

A partir do gráfico apresentado na Figura 13 é possível verificar como é diferente o comportamento das ferramentas em um ambiente mais robusto. Observando o consumo de CPU (Métrica M4), observa-se um comportamento mais estável e com um consumo mais moderado quando comparado ao consumo no Raspberry, que estava trabalhando próximo ao seu limite. Deve-se lembrar que o resultado aqui é representado percentualmente e realizando uma análise sabendo que os hardwares são bastante diferentes. Essa discussão leva em conta que é uma análise preliminar e que o objetivo é levantar algumas hipóteses. Com isso em mente, é possível observar que o sistema do Storm continua exigindo grandes

[12] https://www.amd.com/pt/products/cpu/amd-ryzen-5-1400

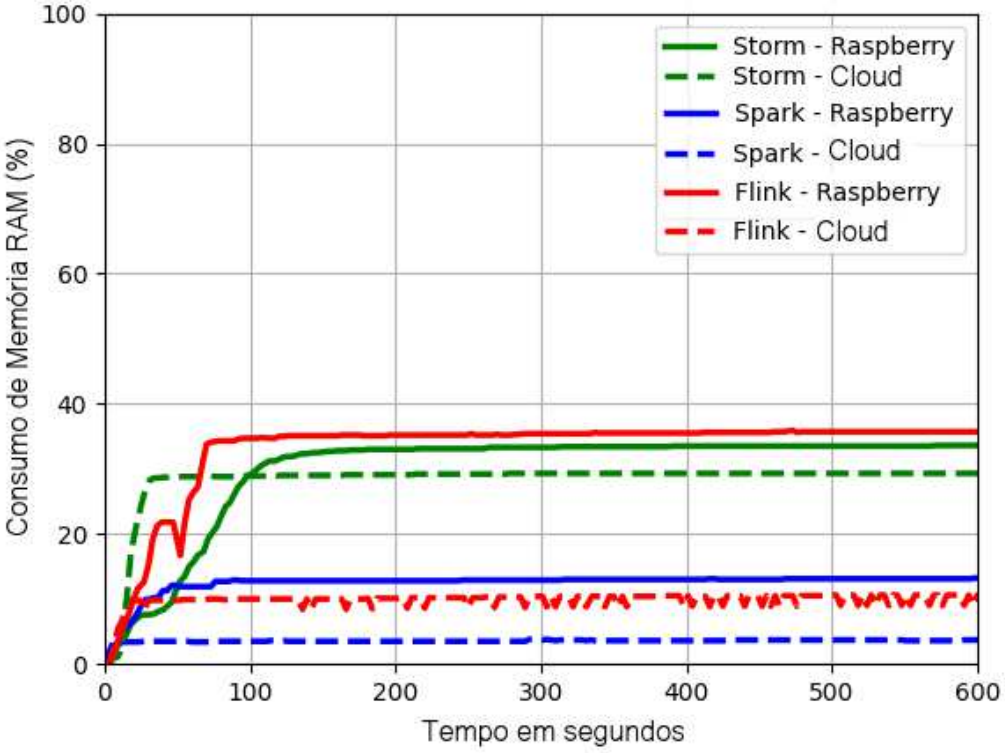

Figura 14 – Consumo de Memória RAM em ambos sistemas.

recursos de CPU, enquanto as outras duas ferramentas exigiram menos recurso de CPU, com o consumo do Flink próximo dos 20% e o Spark abaixo dos 10%.

Com relação ao consumo de memória (Métrica M5), observa-se na Figura 14 os comportamentos das diferentes ferramentas em ambos as camadas, *Fog* e *Cloud*. O primeiro ponto a se observar é que, ranqueando o consumo no Raspberry Pi (*Fog*) por ordem crescente, temos as ferramentas Spark, Storm e Flink, enquanto que no cluster (*Cloud*) temos Spark, Flink e Storm. Neste caso, deve ser observado o Flink próximo ao Spark, uma vez que os dois trabalham com estruturas montadas na memória, mas não foi exatamente o que aconteceu. Pode existir um overload por parte do Flink, pois é necessário subir uma aplicação que gerencia o sistema do Flink de disponível. É importante lembrar também que o sistema do Raspberry Pi possui 4GB de memória RAM, enquanto o Cluster possui 16GB.

No gráfico da Figura 15 verifica-e que os resultados apresentados para elementos processados (Métrica M6) do cluster são semelhantes aos resultados do Raspberry Pi, porém de forma escalonada considerando que é um sistema mais robusto, indicando que a mudança de plataforma gera pouco impacto no resultado. Na camada *Cloud*, o Flink foi a ferramenta com maior capacidade de processamento dos elementos, mas novamente com uma pequena margem de diferença para o Spark. Assim, é obeservado que o Flink

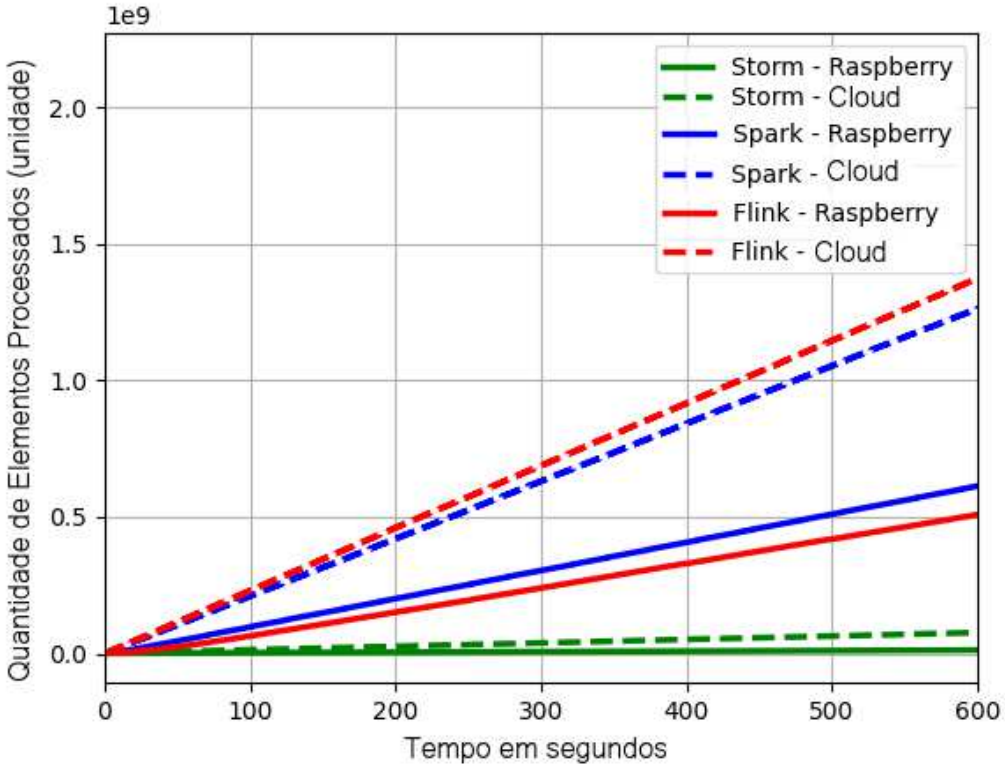

Figura 15 – Elementos processados em ambos sistemas.

e o Spark se destacam como ferramentas mais promissoras para serem utilizadas tanto em uma camada *Fog* quanto em uma camada *Cloud*. Porém, ambas as ferramentas se mostram competitivas, então é difícil definir qual é a melhor opção de uma forma objetiva.

5.1.6 Discussão sobre os Resultados

Este experimento apresenta o modelo computacional em *Fog* como uma alternativa para solucionar problemas apresentados pela computação em *Cloud*. A camada *Fog* tem propósitos principais de promover baixa latência, fornecendo uma resposta rápida a um usuário, e evitar sobrecarga na *Cloud* através da utilização dos recursos computacionais da infraestrutura da *Fog*.

O experimento apresenta um hardware com processamento limitado na camada *Fog* e um contraponto representado em uma camada *Cloud* com maior poder computacional. Neste caso, um elemento da rede produz dados para um sistema de publisher/subscriber para que o outros elementos consumam esses dados atestando a capacidade das FPSD dentro de cada ambiente. Essa configuração permite avaliar o funcionamento dessas ferramentas no hardware, variando a arquitetura computacional.

A avaliação foi realizada considerando ferramentas influentes na academia e indústria para um hardware conhecido, o Raspberry Pi. Os experimentos inicias demonstraram que as ferramentas, dadas as características determinadas, funcionam na camada *Fog*. Assim,

foi possível analisar como os recursos computacionais foram consumidos em cada caso, possibilitando que seja feito um planejamento baseado na utilização.

Para verificar se os objetivos foram alcançados, as questões de pesquisa propostas em 5.1.2 devem ser respondidas.

QPS1 - Como é o consumo dos recursos computacionais pelas FPSD na camada *Fog*?

O ambiente principal é um Raspberry Pi 4 model B, que conta com uma tecnologia bastante avançada, com um processador quad-core a 1.5 GHz, com 4GB de memória RAM. Apesar de ser um hardware robusto, uma FPSD necessita de uma quantidade expressiva de recursos computacionais para o processamento dos dados.

Durante os experimentos, foi observado, principalmente, os valores da Métrica M1, que corresponde ao consumo de processamento das ferramentas, uma vez que o algoritmo executado não possuía como característica grande consumo de memória. Nesse caso, foi concluído que o consumo de CPU eram semelhantes entre as ferramentas, não sendo possível apontar qual ferramenta seria a melhor opção para o cenário do experimento.

Com relação à Métrica M2, foi observado que e o Spark se sobressaiu com relação ao consumo de memória RAM, chegando a consumir, aproximadamente, metade da memória RAM quando em comparação com as outras duas ferramentas. Neste caso, no cenário do experimento, o Spark é uma melhor alternativa em ambientes com menos recurso de memória RAM.

Observando os valores da Métrica M3, é possível analisar a quantidade de elementos processados por cada ferramenta na camada *Fog*. Neste quesito, o Storm atinge uma quantidade muito baixa em comparação com as outras duas ferramentas, o que o exclui como uma boa FPSD no contexto do experimento conduzido. Por outro lado, o Spark e o Flink conseguem valores muito próximos de elementos processados com uma leve vantagem para o Spark. Levando em consideração o resultado de quantidade de elementos processados (Métrica M3) e consumo de memória RAM (Métrica M2), é razoável dizer que o Spark se saiu melhor durante os experimentos realizados neste estudo.

QPS2 - Qual tipo de comparação pode ser sugerida entre os resultados obtidos dentro de um sistema de computação *Fog* e um de computação *Cloud*?

As Métricas M4, M5 e M6 foram utilizadas com o propósito de responder a essa questão de pesquisa secundária. Para quantificar essas métricas, as ferramentas foram utilizadas em um Cluster com poder computacional muito superior ao utilizado na camada *Fog*. Com isso, a avaliação é que, dadas as devidas proporções dos ambientes computacionais, além da camada *Cloud* ter a capacidade de processar mais informação por segundo (Métrica M6), a grande diferença está ligada a estabilidade dos sistemas com

relação a consumo de CPU (Métricas M4), uma vez que os recursos computacionais são mais abundantes.

Com base nas respostas das duas Questões de Pesquisa Secundárias é possível responder a questão de pesquisa primária.

Questão de Pesquisa Primária - *O hardware disponível em uma camada Fog e capaz de suportar as operações de Ferramentas de Processamento de Streaming de Dados, considerando as limitações do hardware?*

O hardware, apesar de possuir uma tecnologia e arquitetura diferentes do ambiente computacional de uma *Cloud* convencional, conseguiu realizar as tarefas propostas e processar os dados através de uma FPSD. Isso está principalmente relacionado às ferramentas escolhidas e a evolução dos hardwares e softwares disponíveis na camada *Fog*. Além disso, o sistema operacional, apesar de ser desenvolvido para a plataforma ARM64, é baseado em linux. Então, diversas facilidades e tecnologias proporcionadas por esse tipo de sistema são incluídas, como Docker e acesso a uma Máquina Virtual Java, que são as tecnologias base para o projeto.

5.2 EXPERIMENTO II

O segundo experimento busca resolver um problema do mundo real e a solução encontrada foi a aplicação de uma camada *Fog* ao sistema para solucionar problemas de atraso, armazenamento e Qualidade de Experiência (QoE). Em linhas gerais, existe um problema relacionado ao envio de tarefas por parte de alunos alocados em unidades acadêmicas (polos educacionais) fora do campus universitário durante o processo de educação a distância. Nesse caso, os alunos assistem e realizam atividades em um polo educacional ligado a um campus. O problema ocorre quando do envio das atividades necessárias para a avaliação dos alunos. Devido a problemas relacionados a largura de banda, estrutura precária de serviço de internet disponível nos polos, e a entrega das atividades pode não ser concluída a tempo dentro da janela de tempo máximo estipulada. Este problema é um fator crítico para os alunos, os professores e o sistema, uma vez que os alunos não conseguem enviar suas atividades e acabam tendo que contar com a boa vontade do avaliador para aceitar a entrega com atraso. O segundo experimento visa avaliar uma proposta na resolução deste problema e, com a devida proporção, tentar tornar esta solução escalável e acessível financeiramente para que seja possível expandir a solução para suportar novos polos.

5.2.1 Escopo da Avaliação

Os ambientes de ensino a distância são sistemas clássicos de computação em *Cloud*, na qual alunos em diferentes localizações geográficas têm acesso ao sistema pela Internet. Este sistema é uma plataforma online de ensino a distância disponibilizado por uma universidade para que os alunos possam interagir com professores e tutores. Todas as atividades avaliativas também são desenvolvidas nesta plataforma e os alunos submetem seus trabalhos, respeitando os prazos estabelecidos pelo professor. A arquitetura computacional desse tipo de plataforma é o modelo tradicional de aplicações cliente/servidor, onde o usuário permanece na unidade fora do campus, nos polos educacionais. Esses polos fornecem máquinas e conexão à Internet, e a universidade oferece um serviço Web com recursos de banco de dados. Os alunos realizam as tarefas e as submetem ao servidor através desta interface Web.

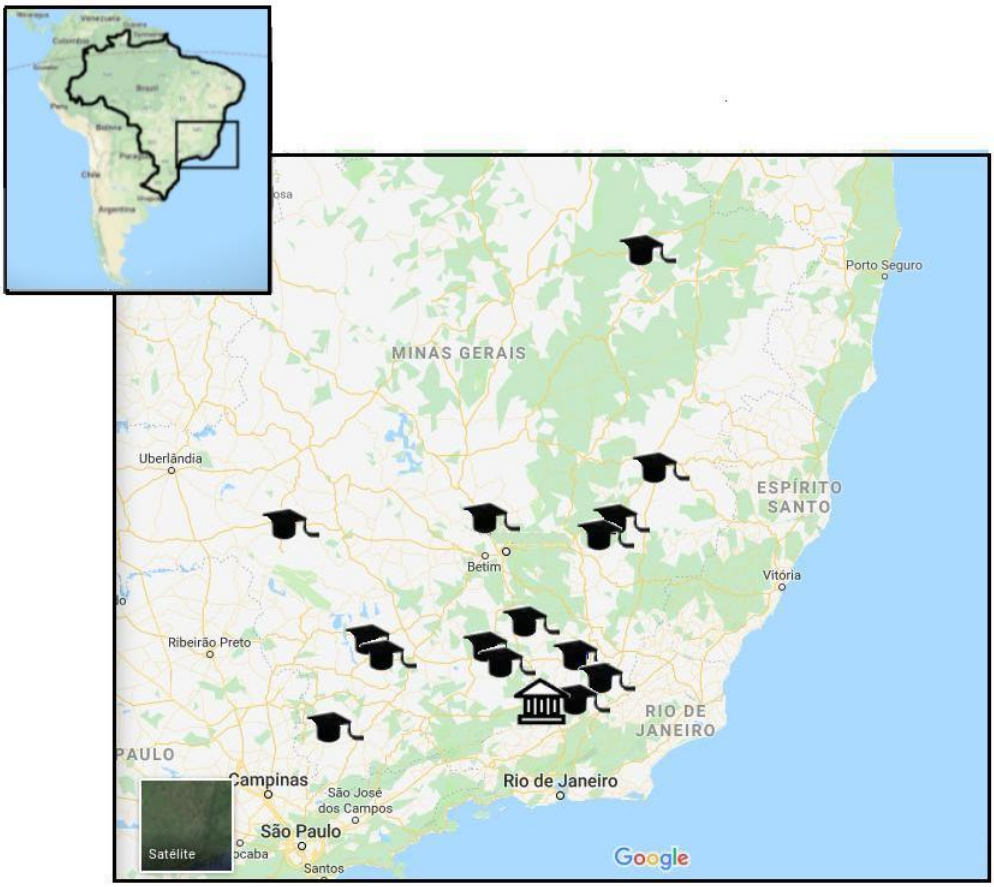

Figura 16 – Distribuição geográfica das Unidades Educacionais.

Esse cenário apresenta alguns problemas para os usuários da plataforma (alunos, professores e tutores) devido à infraestrutura precária nos polos educacionais: (i) falha de conexão; (ii) atraso na entrega dos trabalhos; (iii) dificuldades no download do material didático; e, principalmente, (iv) problemas no upload de documentos, causando perda de

conteúdo.

O mapa da Figura 16 refere-se à região sul do estado de Minas Gerais, no Brasil. Cada "chapéu de formatura" representa a posição geográfica dos polos educacionais, e a "construção" representa o campus universitário central (Universidade Federal de Juiz de Fora - UFJF). É possível visualizar como as unidades estão distribuídas dentro do estado. A maioria das unidades fora do campus estão localizadas em cidades pequenas, que não possuem uma boa conexão de internet.

É fundamental mostrar que, embora as cidades pertençam a um estado, Minas Gerais possui uma área de cerca de 586.528 km², o que a torna maior que países como Alemanha, Espanha e Itália. Além disso, a sua população tem pouco mais de 20 milhões de habitantes, o dobro da população de Portugal e da Grécia. A distância entre a cidade de Juiz de Fora e polo mais próximo é de 40km, mas a cidade mais distante fica a cerca de 900km de Juiz de Fora, distância equivalente a uma viagem de Paris a Berlim. Com essas informações, verificamos a extensão desse experimento, que poderia atender inclusive a países e até mesmo extrapolar para um continente, ao invés de atender apenas a cidades de um estado.

Nesse contexto, a solução proposta neste experimento busca diminuir o impacto dos problemas relacionados conectividade entre os polos e o servidor da instituição de ensino central, alocado na *Cloud*. O principal problema está relacionado aos alunos que enviam suas atividades concluídas aos servidores. Um segundo objetivo é resolver este problema de forma escalável e com o baixo custo, facilitando a implementação de novos polos e a aplicação da solução nos já existentes.

Dadas as especificidades de infraestrutura dos polos, o número de alunos e a complexidade do domínio educacional, observamos vantagens em uma cooperação *Fog-Cloud* para atuar próximo ao usuário final, realizando coleta e processamento de dados, incluindo a redução de saltos de rede, aproveitando o gestão de informação mais próximo do ponto de coleta de dados, melhorando o tempo de resposta geral, e reduzindo a carga da *Cloud* para realizar armazenamento e outras atividades como o processamentos de dados. Dessa forma, os dados dos alunos no polo podem ser geridos por um nó de computação *Fog*. Com base nos resultados obtidos no experimento I, assumimos então um nó *Fog* convencionaprovavelmente poder computacional suficiente para manipulação e captura dos dados, pré-processamento e enriquecimento desses dados, e, finalmente, envio do resultado para a *Cloud* para pós-processamento, análise e armazenamento.

O uso do framework proposto no Capítulo 4 tem o objetivo de solucionar os problemas de conectividade entre os polos e o servidor localizado no campus universitário. A Figura 17 representa a instanciação do framework no cenário descrito anteriormente.

Cada polo é um nó *Fog* com autonomia e poder de processamento. O maior problema

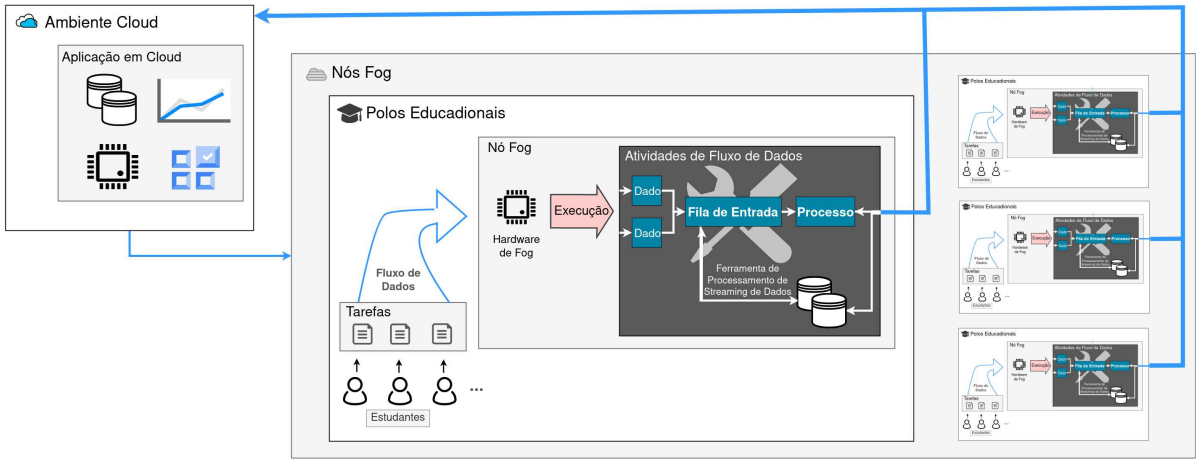

Figura 17 – Proposta de Framework com os componentes e o fluxo de dados para o experimento II.

aqui é a submissão das tarefas concluídas por parte dos alunos com destino ao servidor, sendo o foco deste estudo. Como segundo ponto, devemos lembrar que a solução deve ser viável e escalável para aplicação em larga escala. Por exemplo, criar uma infraestrutura para melhorar a conectividade seria uma solução eficiente, mas consideravelmente cara. A Figura 17 fornece uma visão geral dos nós *Fog* e os componentes processados em cada um deles.

De forma semelhante a execução do experimento anterior, os dados produzidos pelos alunos são coletados e enviados para o componente **Atividades de Fluxo de Dados**. Esta etapa é responsável por garantir as regras de Restrição de Tempo e a manutenção das atividades. Os dados recebidos pelo **Atividades de Fluxo de Dados** contêm informações sobre os alunos, atividade (data e hora de início e término) e o identificador do polo onde a ação foi realizada.

O **Atividades de Fluxo de Dados** é composto por duas subetapas: **Fila de Entrada** e **Processamento**. É necessário definir uma **Fila de Entrada** para receber, organizar e armazenar temporalmente os dados dos alunos. A **Fila de Entrada** é responsável por "ouvir" e armazenar os dados coletados em um banco de dados temporário até processá-los. Dessa forma, evitamos perdas de arquivos que podem prejudicar as avaliações dos alunos.

Posteriormente, os dados são processados, no componente **Processamento** , passando por uma etapa de filtragem e limpeza dos dados durante o processamento da fila. Ele identifica valores ausentes e inesperados, seja porque é de um tipo diferente ou porque é um valor discrepante. Além disso, enriquecemos os dados com informações como data/hora do início da atividade, identificador do tutor e professor, data/hora da atividade recebida pela fila, tempo de duração do processamento, data/hora do envio da atividade para a *Cloud*, etc. As saídas do **Processamento** são encaminhadas para a *Cloud* onde os arquivos são

armazenados no servidor da universidade, e os professores podem acessá-los e avaliá-los.

O sistema proposto é uma alteração no Moodle de forma que ele possa receber as tarefas dos alunos na *Fog*, e, posteriormente, quando a conexão com o servidor estiver ativa e os sistemas estiverem com maior disponibilidade, esses dados poderão ser enviados ao servidor. Dessa forma, é possível garantir a autenticidade, localização e tempo de envio da obra em ambiente autorizado.

Com esse framework, espera-se que os documentos dos alunos sejam processados rapidamente na *Fog*, sem a necessidade de enviá-los para a *Cloud*. Quando o aluno conclui a tarefa, o nó *Fog* armazena o documento localmente, salva o momento de conclusão das tarefas para respeitar as regras de restrição de tempo e envia as informações para a *Cloud* posteriormente. Este processo evita a sobrecarga da rede e aumenta o tempo de resposta.

Dada a complexidade da solução proposta, a avaliação foi realizada em duas etapas. Na primeira etapa, foi realizado um experimento simulando a rede em operação, verificando suas fragilidades a fim de descobrir o que atacar e qual o impacto esperado da solução. Os dados sobre a infraestrutura das unidades remotas foram então coletados para realizar o experimento, verificando o número de máquinas e o pacote de conexão à internet em cada unidade para que a simulação da solução reflita o ambiente real das unidades da melhor forma possível.

Na segunda etapa, o framework foi avaliado para verificar se a solução proposta poderia resolver os problemas da primeira etapa. Para isso, foi desenvolvido um protótipo utilizando como parâmetros os dados coletados nos polos. Esta simulação representa um dia escolar típico e como o sistema se comporta nesse sentido.

5.2.2 Questões de Pesquisa do Experimento II

O escopo desta avaliação foi baseado no método GQM (KITCHENHAM; CHARTERS, 2007) cujo o objetivo e escopo são descritos a seguir:

"**Analisar** a capacidade atual de infraestrutura dos polos educacionais e do framework proposto **com o objetivo de** fornecer uma solução de baixo custo capaz de resolver o problema de sincronização no recebimento de arquivos **em relação a** restrição de tempo no envio da resolução de atividades na modalidade EaD **sob o ponto de vista de** alunos, professores e tutores **no contexto de** polos educacionais no Brasil".

Com base na definição do escopo, foi definida uma Questão de Pesquisa Primária (QPP) e duas Questões de Pesquisa Secundárias (QPS) relacionadas ao segundo experimento:

QPP - ***Como contornar os problemas relacionados a horário de recebimento***

das atividades enviadas pelos alunos nos polos educacionais, considerando as regras de restrição de tempo na modalidade ensino a distância?

Essa questão vai de encontro ao objetivo desse experimento, que é resolver as falhas de envio geradas por conta da precariedade da infraestrutura disponível para ambientes em certos polos educacionais. A partir destes dados, desenvolver uma solução possível para resolver este problema para todo o meio que se encontra em condições semelhantes.

O experimento também valida a *Fog* como um ambiente sólido, que pode dar tanto suporte, como ser protagonista em sistemas computacionais, validando a relevância do poder computacional distribuído na *Fog* e toda a gama de aplicação que esse ambiente pode disponibilizar.

QPS1 - A cooperação *Fog-Cloud* é capaz de auxiliar os alunos no envio de suas atividades garantindo que o sistema reconheça o momento do envio?

Essa questão avalia a competência da camada *Fog* em cooperação com a *Cloud* computacional e a capacidade de ambos trabalharem em conjunto para resolver problemas provocados por uma falta de qualidade de infraestrutura. Com esta questão buscamos avaliar se o sistema reconhece o momento de envio das atividades pelos alunos mesmo que ocorram problemas de conexão com o servidor na *Cloud*.

QPS2 - O framework proposto é capaz de gerenciar o envio das atividades para a *Cloud* sem necessidade de grandes investimentos de recursos financeiros?

Esta questão tem o objetivo de verificar se o framework proposto é capaz de solucionar o problema de envio das atividades sem a necessidade de investimentos em infraestrutura. Assim, além de verificar a viabilidade técnica da solução, buscamos discutir custos, considerando um cenário real onde não é possível grandes investimentos econômicos.

Assim como na avaliação anterior, as métricas foram definidas para responder as questões de pesquisa (KITCHENHAM; CHARTERS, 2007):

M1: Tempo necessário para o translado do arquivo partindo do usuário até ser completamente entregue ao servidor que mantém os dados.

M2: Volume de dados armazenados no ambiente do servidor de dados considerando a soma dos arquivos acumulados durante uma execução.

5.2.3 Descrição do Cenário de Experimentação

Conforme relatado por professores, alunos e tutores, próximo ao horário previsto para o início das atividades online há uma grande quantidade de acessos, pois os alunos

precisam acessar a plataforma para se informarem sobre as atividades e arquivos de conteúdo relacionados. Nesse intervalo de tempo, muitas solicitações são direcionadas para um servidor centralizado na universidade.

Figura 18 – Comportamento do uso de dados na rede em relação ao tempo.

As ações dos alunos no Moodle se resumem, de maneira geral, em uploads e downloads de pacotes de um determinado polo com concentração no início e no final do tempo disponível. Essa restrição de tempo é um desafio para a infraestrutura existente, pois exige que todos os arquivos sejam armazenados na *Cloud* dentro do prazo estabelecido pelo professor. Durante o período de atividades dos alunos, há tráfego de dados, mas que não é volumoso o bastante para ser considerado, ou seja, não há uma concentração grande o suficiente de transferência de dados nesse período. Perto do prazo final de envio da tarefa, há outra curva acentuada de transferência de pacotes, que é quando os alunos enviam suas tarefas concluídas para o servidor da plataforma de ensino a distância. O gráfico da Figura 18 exibe como esse comportamento acontece considerando o uso da rede ao longo do tempo. Esta distribuição baseia-se na experiência dos professores e na análise dos dados relativos às atividades realizadas e armazenadas na plataforma Moodle.

Tabela 2 – Configuração das unidades remotas

Unidades Remotas	**Número de Máquinas**	**Fluxo (Mb/s)**
1 - Vermelho	30	20
2 - Azul	20	4
3 - Verde	47	4
4 - Amarelo	50	4
5 - Laranja	30	10

Este cenário foi simulado para entender as reais limitações da infraestrutura de cada polo. Os dados foram coletados nos polos da Universidade Federal de Juiz de Fora,

na cidade de Juiz de Fora, Brasil, com o objetivo de realizar a avaliação. Foram utilizados dados de cinco polos: Barroso, Boa Esperança, UAB de Juiz de Fora, Santa Rita de Caldas e Sete Alagoas e os valores são apresentados, não nominalmente, por questão de sigilo, conforme Tabela 2. O servidor da UFJF é considerado com uma conexão de 100Mb/s, que é o valor utilizado na avaliação.

De forma geral, os alunos podem levar até 120 minutos (2 horas) para concluir a atividade. O professor define as restrições de tempo e, após o término do intervalo os alunos não podem mais enviar o trabalho pela plataforma Moodle. Alunos com problemas na entrega dos trabalhos têm de contar com a disponibilidade dos tutores e dos professores responsáveis pelas atividades e enviar os arquivos de forma não convencional, como, por exemplo, e-mail.

Durante a avaliação, um ponto a se considerar é que parte dos provedores de internet no Brasil oferecem planos com limite de taxa de upload inferior ao limite de taxa de download. A primeira indicação desse fato são as informações fornecidas pelo SPEEDTEST[13], site que realiza testes de velocidade em todo o mundo e compila os dados. Segundo esse site, a taxa de upload do Brasil representa uma média de 46% da velocidade de download .

Outra taxa de upload amplamente adotada no Brasil é 10% da taxa de download. Assim, durante o processo de avaliação, três fatores foram considerados: 10%, 46% e 100% do valor inicial da taxa de download.

5.2.4 Resultados

Nesta primeira etapa, o objetivo foi verificar a solução quanto ao uso do paradigma *Fog-Cloud*, considerando o contexto dos polos educacionais no Brasil. Para isso, com base em dados reais, são analisados os impactos da infraestrutura das unidades durante o envio das atividades dos alunos, buscando responder à primeira questão de pesquisa secundária.

Para o presente experimento, as aulas dos alunos nos polos foram “realizadas simultaneamente” para obter a maior quantidade de acesso possível no servidor de ensino a distância da universidade. A intenção era verificar se a origem do gargalo estaria no servidor de ensino a distância. No entanto, esta hipótese foi descartada através de uma análise inicial dos dados após a simulação, evidenciando que o problema era proveniente das unidades remotas.

Embora a ideia fosse chegar o mais próximo possível de um ambiente real, outros tipos de fluxo de rede não foram considerados. Foi simulado apenas o fluxo de conexão entre os polos e o servidor de ensino a distância, o que exclui problemas externos como número de saltos, congestionamento de rede ou queda em algum dos serviços, aproximando

[13] https://www.speedtest.net/global-index/brazilfixed

esse cenário do melhor caso possível de acordo com o que o sistema oferecia.

A ferramenta utilizada para simular o envio de atividades e o fluxo da rede fora do campus é o *Network Simulator 2 (ns-2)*[14]. Além disso, com essa ferramenta é possível modelar um ambiente de rede de computadores, definindo os elementos e conexões da rede e as operações que ocorrem nele, como tráfego de pacotes e fluxo de dados.

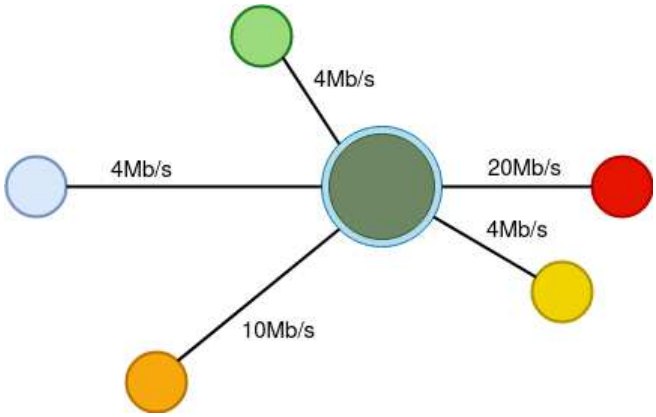

Figura 19 – Grafo da Rede.

A modelagem dessa ferramenta funciona como um grafo em que as arestas se referem aos valores de fluxo e largura de banda, e os vértices são os elementos da rede, como hubs, roteadores e computadores. A Figura 19 ilustra a rede configurada no *Network Simulator*. O nó central representa a universidade, enquanto os outros nós representam os polos. As cores estão relacionadas aos elementos apresentados no Gráfico 2.

Com a rede gerada, é necessário enviar arquivos para verificar o tráfego de dados e o tempo necessário para enviar todos os arquivos para a *Cloud*. A intenção é se aproximar de um modelo semelhante ao da Figura 18, que é o comportamento dos alunos durante as avaliações. O simulador tem como funcionalidade a especificação do tamanho de arquivos, protocolos e tempo de envio inicial e final, o que nos permite analisar o volume de arquivos que precisa ser armazenado temporariamente na *Fog* até que sejam finalmente transferidos para a *Cloud*.

A simulação gera um log com todos os pacotes que cruzaram a rede, assim como acontece no Wireshark[15]. A partir desses dados, é possível verificar como ocorreu o fluxo de dados na rede, erros, pacotes perdidos, tempo total de envio, etc. Os dados são processados através de uma ferramenta desenvolvida durante o projeto que usa a linguagem Python, e os gráficos foram plotados usando a biblioteca Matplotlib[16].

Um programa Python foi desenvolvido e executado por um scrip na linguagem shell script utilizando o Network Simulator 2 (ns-2) para a análise do fluxo da rede das

[14] https://www.isi.edu/nsnam/ns/
[15] https://www.wireshark.org/
[16] https://matplotlib.org/

unidades fora do campus. A saída do simulador serviu como entrada por um algoritmo em outro Python que processa dados e se utiliza da biblioteca matplotlib com o objetivo de gerar gráficos para melhor visualização e análise dos resultados obtidos. Os códigos-fonte deste framework estão disponibilizados no GitHub[17] para auxiliar outros pesquisadores. Os resultados são descritos a seguir.

5.2.4.1 Fator 100%

Quando as taxas de upload e download são iguais, a simulação revela que apenas algumas atividades são enviadas após o tempo limite. Embora o atraso na entrega seja apenas até 2 minutos, é suficiente para que alunos tenham problemas no envio de arquivos. Isso ocorre porque alguns alunos enviam os arquivos próximo ao horário limite.

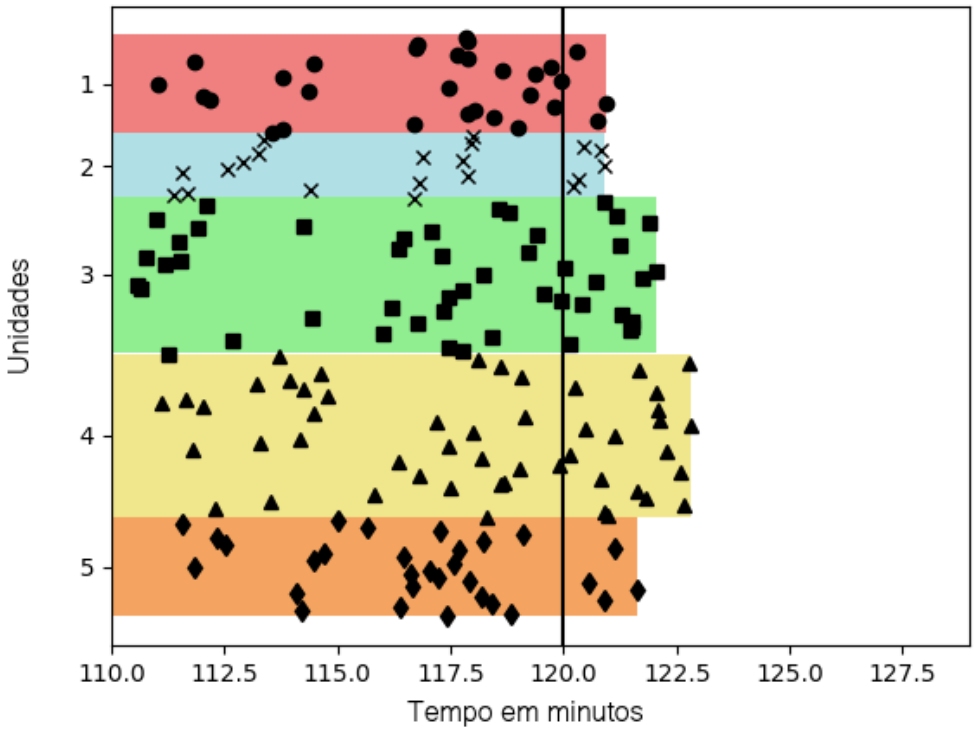

Figura 20 – Arquivos enviados em relação ao tempo de conclusão da tarefa (fator de 100%).

A Figura 20 refere-se ao momento final que cada arquivo finaliza o envio, representado pelas figuras geométricas. O início não está descrito no gráfico, mas está dentro do intervalo final de 10 minutos antes do fina das atividades. Os polos educacionais estão divididos em diferentes cores, como apresentado na Tabela 2 e o tamanho horizontal está relacionado ao número de máquinas do respectivo polo. A figura indica o momento da entrega dos arquivos para o servidor em *Cloud*. Neste caso, onde a taxa de upload é a

[17] https://github.com/lucaslarcher/Framework_Analise_Saida_NS2

mesma do download, o tempo de upload do arquivo mais demorado é de aproximadamente 3 minutos.

5.2.4.2 Fator 46%

No caso em que a taxa de upload é 46% da taxa de download a entrega se torna mais demorada, refletindo a situação mencionada por professores, alunos e tutores. Nesse caso, o gargalo da rede começa a se tornar mais evidente.

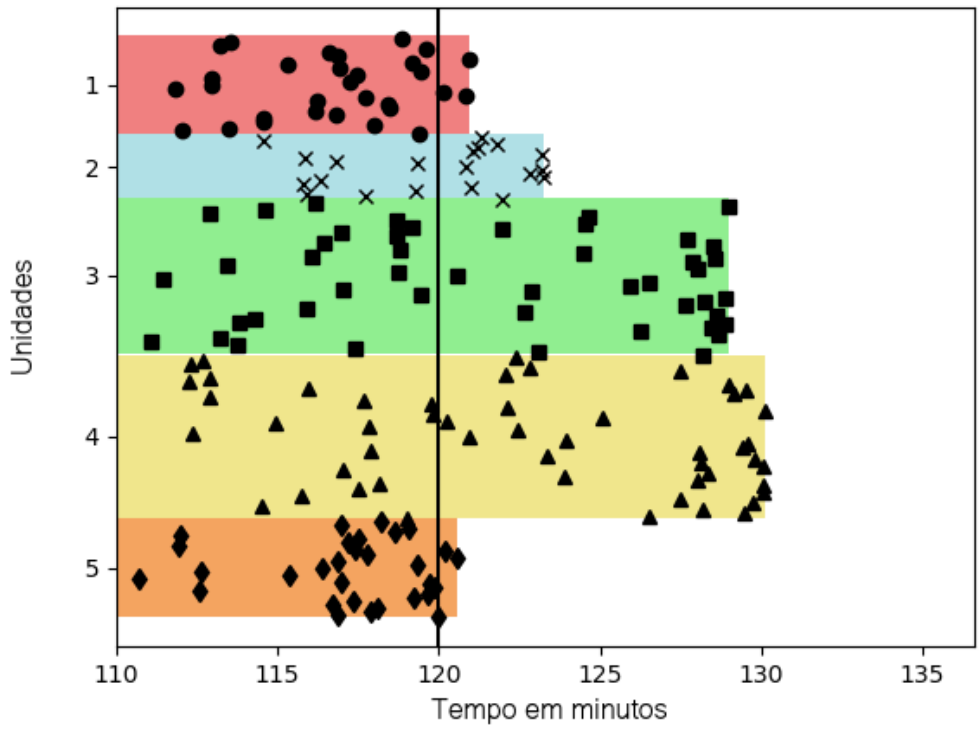

Figura 21 – Arquivos enviados em relação ao tempo de conclusão da tarefa (fator de 46%).

É visível no gráfico da Figura 21 que os polos apresentam piora no tempo de entrega. Os polos representados nas cores verde e amarelo levam aproximadamente 8 minutos para enviar um arquivo. Há congestionamento gerado por conta da criação de uma fila de pacotes que não possui uma transmissão rápida o suficiente para atende-la. Nesse caso, os alunos precisam começar a enviar as tarefas com ainda mais antecedência, ou seja, os alunos possuem menos tempo para resolver as tarefas nesses polos do que os alunos de outras unidades. Nesse cenário os arquivos que mais demoraram para serem enviados chegam a demorar aproximadamente 10 minutos.

5.2.4.3 Fator 10%

Considerando que a taxa de upload é 10% da taxa de download, este é o estado com o tempo de entrega mais crítico. Neste caso, é possível verificar que a fila se torna ainda mais difícil de ser resolvida, o que torna esse caso um caráter emergencial.

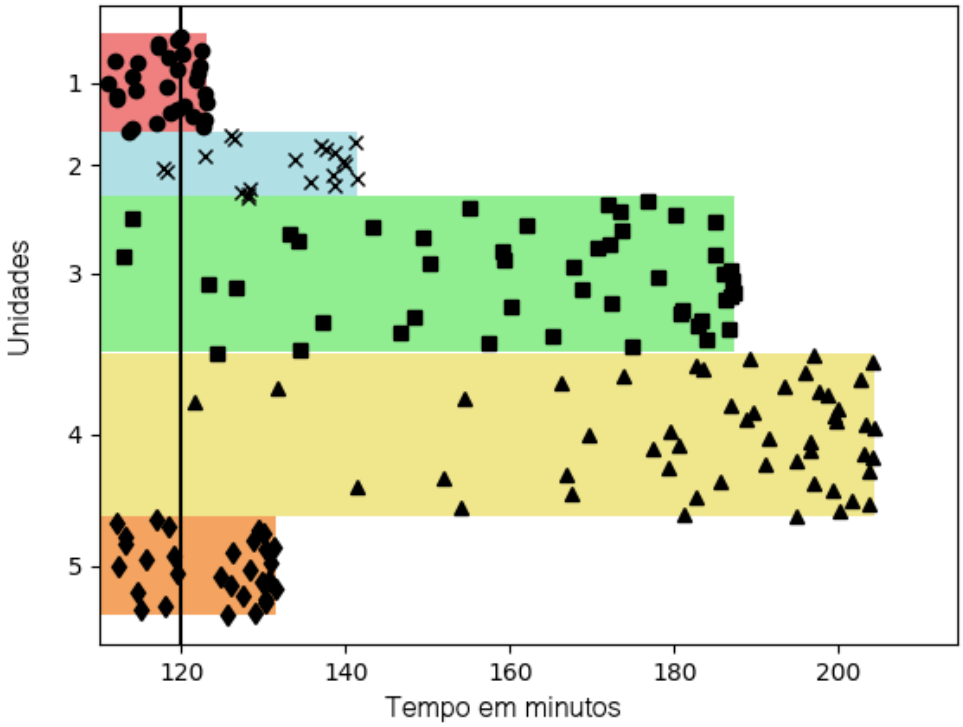

Figura 22 – Arquivos enviados em relação ao tempo de conclusão da tarefa (fator de 10%).

A Figura 22 exibe o tempo que todas as unidades remotas levam para enviar os arquivos até o final da transmissão. O prazo para envio dos arquivos é de 120 minutos (duas horas para realizar as atividades online). Há um atraso perceptível de acima 80 minutos para entregar arquivos nos piores casos. A Figura apresenta problemas que já eram esperados seguindo os dados dos experimentos anteriores, pois mais uma vez o fluxo não é suficiente e a estrutura acaba ficando sobrecarregada. Esta situação extrapola o caso anterior e mostra a urgência sobre o uso de outro tipo de projeto de infraestrutura nesses polos que possuem infraestrutura mais precária.

5.2.4.4 Conclusão Parcial

Com base nos gráficos, é possível concluir que o sistema está saturado nas pontas e mesmo no cenário mais amigável, os arquivos não foram transferidos em sua totalidade, mesmo que todas as tarefas sejam enviadas antes do prazo final da atividade. Como esperado, por meio de experimentos, foi verificado que o problema ocorre por conta da

qualidade da conexão de internet disponível em regiões dos polos e assim excluindo a teoria de sobrecarga do sistema na *Cloud*.

Com base nas análises realizadas nesta etapa da avaliação, é possível afirmar que a arquitetura padrão em *Cloud* não é indicada considerando a infraestrutura dos polos em que a situação de conexão é precária. Como o gargalo está presente no o envio dos arquivos para a *Cloud*, há evidências de que uma proposta que leve em consideração o armazenamento local das atividades para que posteriormente haja o envio das mesmas pode ser uma solução viável, respondendo parcialmente a segunda questão de pesquisa do experimento. Na próxima seção, esta cooperação *Fog-Cloud* é avaliada através do desenvolvimento e execução do framework proposto.

5.2.5 Análise da cooperação Fog-Cloud

Nesta etapa de avaliação, foi verificada a necessidade de se passar por uma série de tarefas, desde a geração de scripts até uma análise gráfica dos resultados. Para agilizar as tarefas o framework foi desenvolvido para dar suporte a este processo, utilizando as tecnologias e etapas apresentadas na Figura 23.

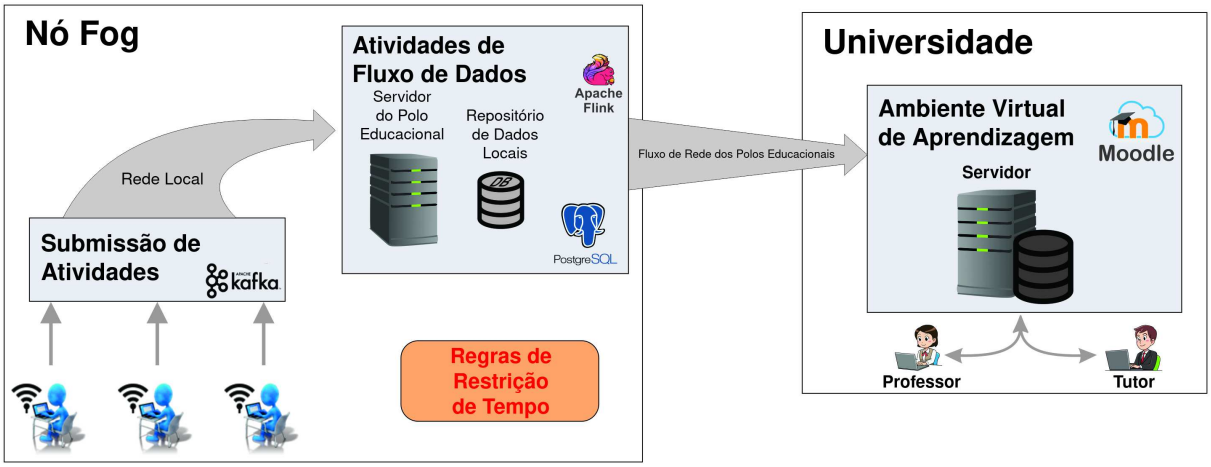

Figura 23 – Tecnologias utilizadas no desenvolvimento do framework e seu fluxo de execução. Começa com os alunos enviando suas atividades e termina com o servidor da universidade armazenando os arquivos, permitindo que professores e tutores acessem.

A plataforma Moodle é o Ambiente Virtual de Aprendizagem adotado pela Universidade Federal de Juiz de Fora em cursos de educação a distância. Esta plataforma trata o tráfego de dados de forma simplificada, deixando os aspectos relacionados à segurança e manutenção da informação a cargo do File Transport Protocol (FTP) (TANENBAUM; WETHERALL et al., 2010). Este tratamento não garante a transferência de informações importantes que não podem ser extraviadas. Nesse contexto, observa-se a oportunidade de usar uma FPSD para dar suporte ao volume de dados que trafegam e reduzir o risco de perda de informações. Assim, o desenvolvimento da solução se baseia na utilização de

um servidor local em cada unidade fora do campus (nós *Fog*) e no envio controlado das atividades dos alunos para o Moodle.

Este servidor local pode ser alocado em hardware simples, como um Raspberry pi, devido ao baixo custo de recursos computacionais para manter a modificação do sistema em *Fog*. O sistema precisa manter os dados por um curto período enquanto não consegue enviá-los para o servidor na *Cloud*. Os arquivos que serão enviados posteriormente para a *Cloud* ficam pendentes até que a conexão com a internet seja habilitada para que não atrapalhe nenhuma tarefa realizada pelos alunos.

O Framework foi desenvolvido como uma solução assíncrona em que a o servidor na universidade é responsável por requisitar a atualização periodicamente. Nos polos, as atividades do aluno são enviadas para um fila de entrada usando o Apache Kafka como um gerenciador de dados distribuído que é interessante por funcionar no modelo Publicador/Consumidor. Implementamos o nó *Fog* usando Apache Flink, que é a ferramenta selecionada aqui para processamento de streaming de dados. Para garantir o processamento dos dados ingeridos, o Flink possui um estado na memória ou no disco, dependendo da capacidade de memória do nó. Ele também é tolerante a falhas, garantindo a consistência dos dados. Por isso, decidimos implementar o nó *Fog* utilizando o Apache Flink.

Desenvolvemos os serviços de fluxo de atividades no Flink, então apenas a rede local é necessária para que o framework funcione. Este serviço é responsável pela implementação da fluxo de dados, que pré-processa os dados e os armazena temporariamente. Com base na janela deslizante, os arquivos enviados pelos alunos são mantidos pelo Flink para serem enviados posteriormente para a *Cloud*. Este componente usa uma janela de tamanho de 30 minutos e uma janela deslizante de 15 minutos. Com isso, a etapa de Processo é executada a cada 15 minutos, considerando 30 minutos de dados. As janelas de tempo de 30 minutos e a janela deslizante de 15 minutos foram obtidas empiricamente com base na experiência dos professores com o processo de ensino a distância. A etapa Processo tenta enviar as atividades dos alunos, uma a uma, e manter aquelas que não são postadas na fila. O framework é responsável por verificar a disponibilidade de acesso e enviar o arquivo para a *Cloud*.

Foi utilizado o banco de dados PostgreSQL[18] para armazenar os dados das atividades dos alunos. O armazenamento pode ser permanente para se ter um histórico das atividades realizadas nos polos ou temporário, após o envio final para a *Cloud*, o arquivo é excluído do nó *Fog*. Foi necessário configurar esta etapa com armazenamento temporário para reduzir o risco de melhorias na infraestrutura.

O Processo do Activity Streaming é responsável por controlar o envio de arquivos

[18] https://www.postgresql.org/

para a *Cloud*. Como esta proposta visa reduzir o custo de implementação da solução e as unidades fora do campus possuem infraestruturas privadas, cada unidade fora do campus pode ser melhor adaptada para armazenar arquivos no nó *Fog*, seja em memória ou em disco local. Caso ocorram muitas falhas no envio dos arquivos, exigindo que sejam mantidos na fila para envio posterior, o servidor do nó *Fog* precisa ter infraestrutura suficiente para armazenamento, caso contrário, não há garantia que as atividades não serão perdidas.

Para esse experimento, um nó *Fog* foi desenvolvido usando um Intel i7-6500U @ 3.100GHz, 4 núcleos de CPU, 8GB de RAM. Ubuntu 16.04.6 LTS x86_64, Flink 1.11-SNAPSHOT, Kafka 2.4 e PostgreSQL 9.6.17.

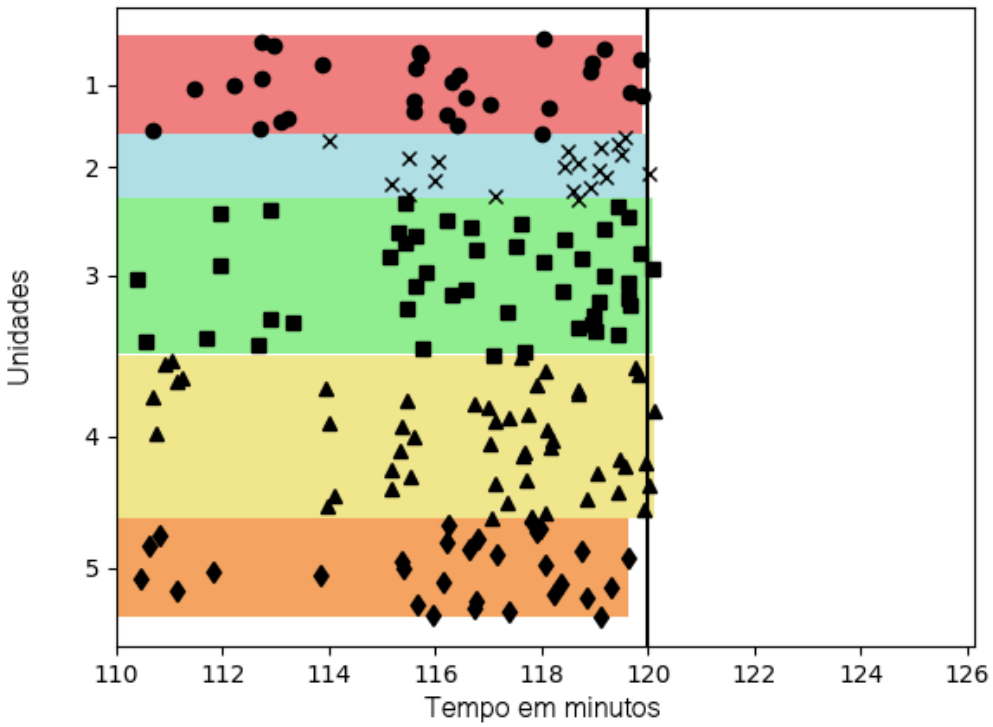

Figura 24 – Arquivos enviados em relação ao tempo de conclusão do trabalho, considerando apenas o Activity Streaming da unidade fora do campus (nó *Fog*).

Naturalmente, se as atribuições forem enviadas para o servidor local, que é um nó *Fog*, o tempo será muito menor. A Figura 24 apenas reforça esta afirmação com o gráfico para conclusão de submissão de arquivos. Neste caso, os tempos são curtos e estão abaixo de um minuto. A cooperação *Fog-Cloud* é uma alternativa viável para resolver o problema de envio nos polos em regiões em que a conectividade é precária. Com esta estratégia, o problema de restrição de tempo é resolvido, tornando o sistema escalável, respondendo a primeira questão de pesquisa secundaria.

Mesmo com o framework apresentando bons resultados considerando a implementação do nó *Fog*, é fundamental lembrar que o processo deve reenviar os arquivos armazenados localmente devido a problemas de envio até que o Moodle confirme o upload desses arquivos. Para analisar o reenvio desses arquivos e seus metadados, as ferramentas de fluxo de dados se fazem importantes, nesse caso o Flink.

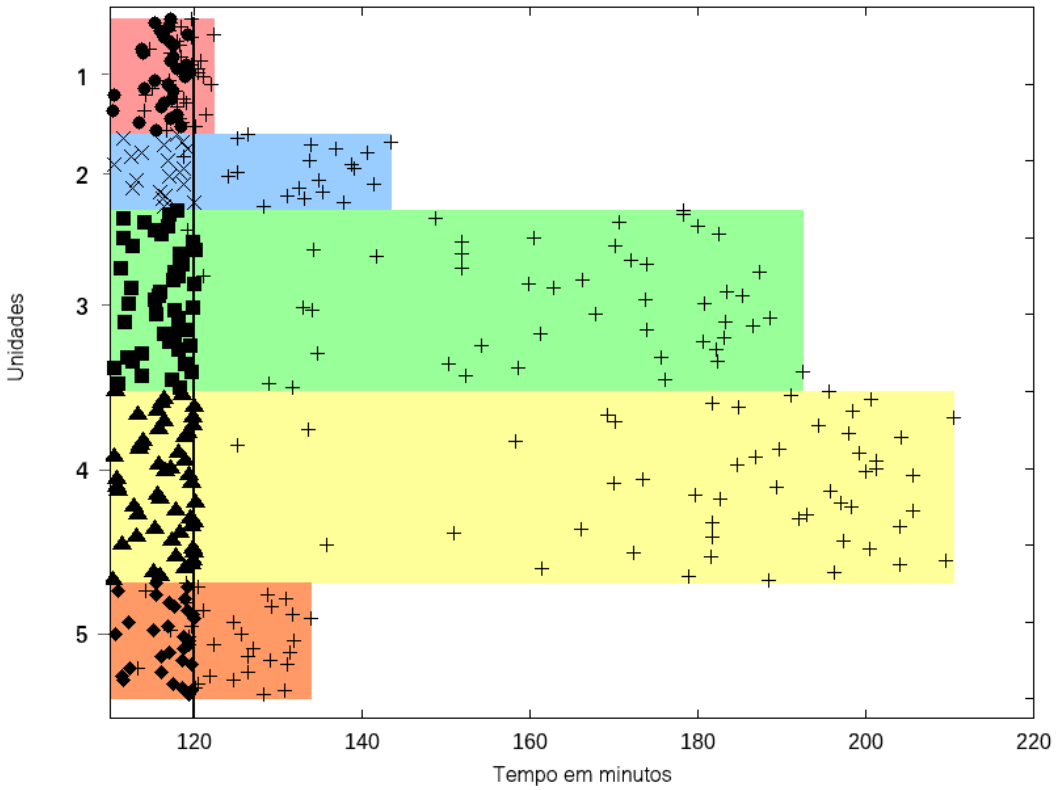

Figura 25 – Tempo de envio de arquivos da *Fog* e para a *Cloud* com um fator de 10%. (Cooperação *Fog-Cloud*).

Os gráficos nas Figuras 25, 26 e 27 exibem o tempo gasto para o armazenando dos arquivos na *Fog* e o tempo gasto armazenando na *Cloud*, considerando os fatores de 10%, 46% e 100%, respectivamente. Lembrando que esses fatores são a proporção do upload em relação ao valor do download. Para facilitar o entendimento, adotamos a mesma notação e coloração utilizada nos gráficos anteriores para representar o armazenamento de arquivos no nó *Fog* (representado por círculo, x, quadrado, triângulo e losango). Utilizamos o símbolo '+' para representar o momento de confirmação do upload completo para a *Cloud*.

É possível observar que independente do fator das unidades fora do campus, a maior parte arquivos foram armazenados no nó *Fog* respeitando a regra de restrição de tempo (120 minutos). Existe a exceção de algum arquivo que foi mandado muito próximo ao tempo limite e por isso não foi enviado a tempo, mas um pequeno ajuste com tolerância de atraso solucionaria essa questão.

Como esperado, o tempo de confirmação de upload da *Cloud* varia de acordo com

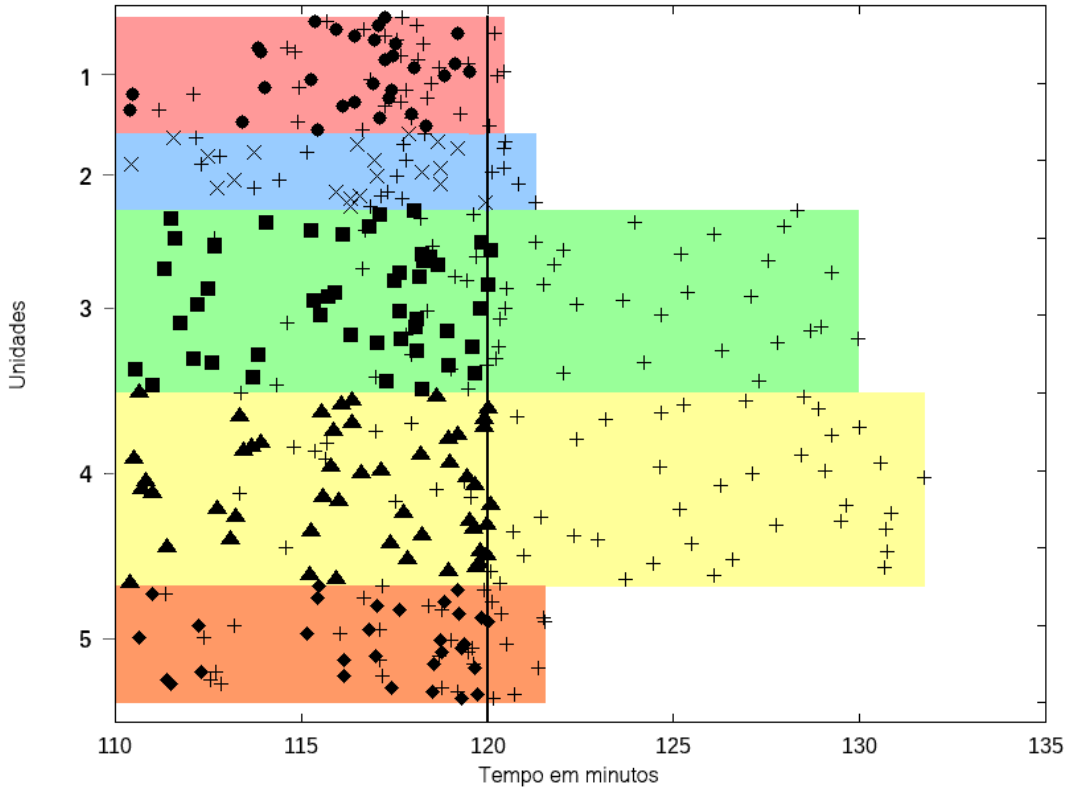

Figura 26 – Tempo de envio de arquivos da *Fog* e para a *Cloud* com um fator de 46%. (Cooperação *Fog-Cloud*).

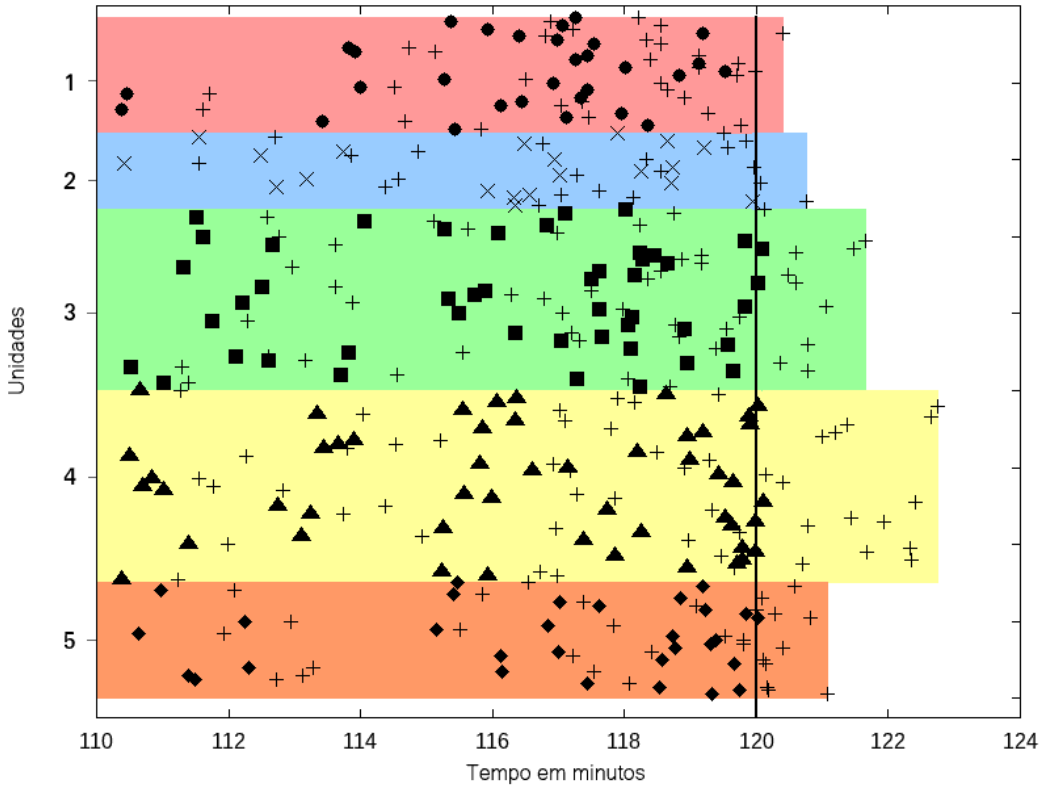

Figura 27 – Tempo de envio de arquivos da *Fog* e para a *Cloud* com um fator de 100%. (Cooperação *Fog-Cloud*).

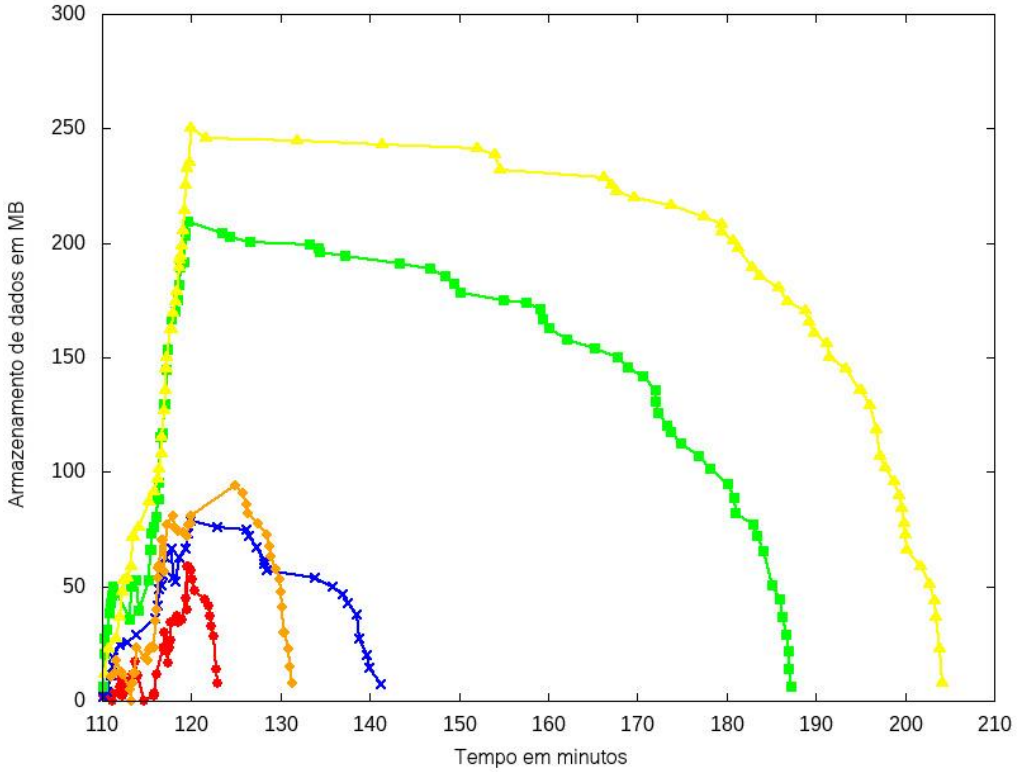

Figura 28 – Volume de dados gerados em unidades fora do campus com um fator de 10%. (Cooperação *Fog-Cloud*).

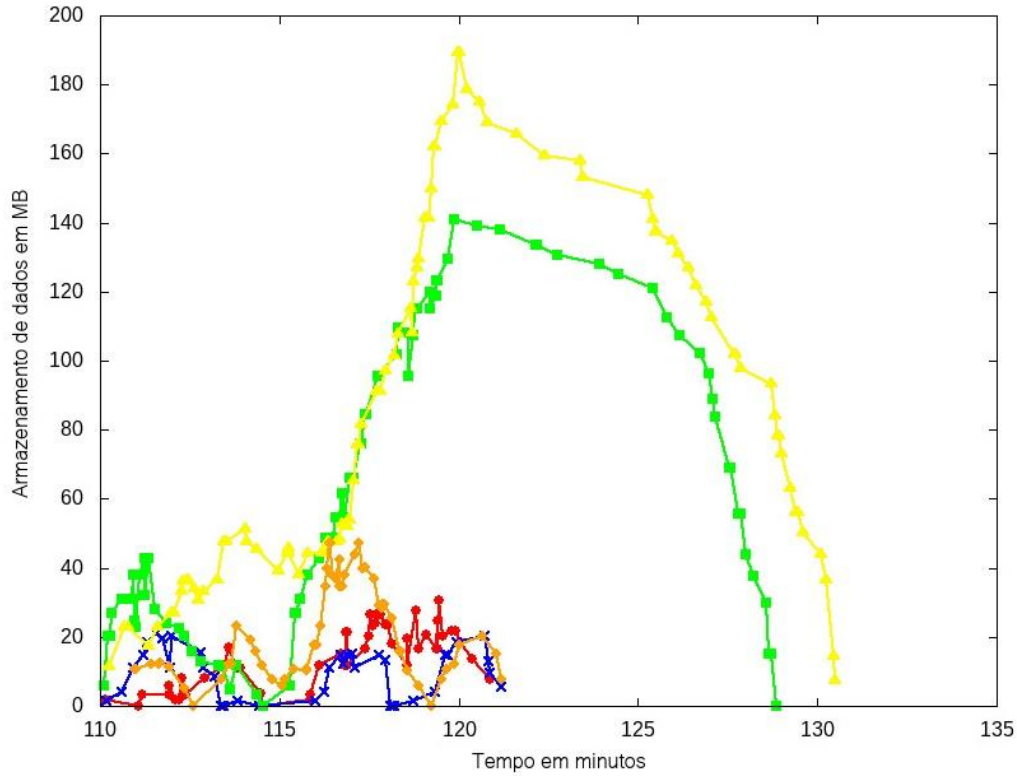

Figura 29 – Volume de dados gerados em unidades fora do campus com um fator de 46%. (Cooperação *Fog-Cloud*).

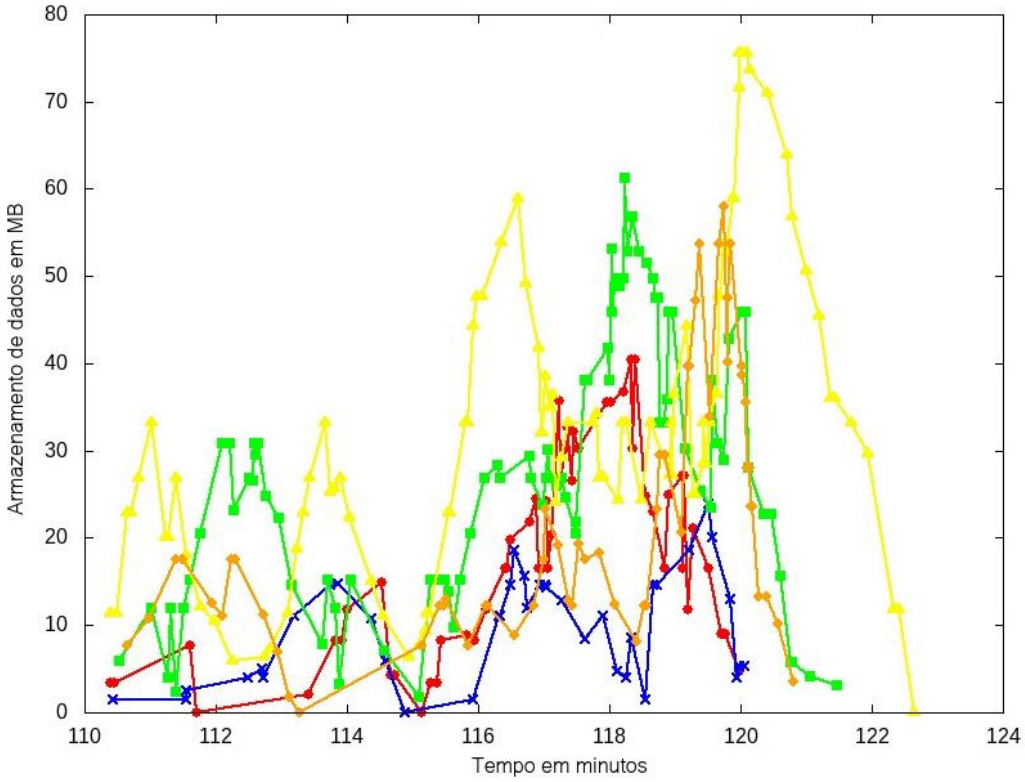

Figura 30 – Volume de dados gerados em unidades fora do campus com um fator de 10%. (Cooperação *Fog-Cloud*).

os fatores de download e upload contratados pelas unidades fora do campus. Assim, ao acessar o ambiente Moodle, o professor tem acesso aos arquivos enviados por parte dos alunos contendo informações sobre o momento do envio para o nó *Fog* e o momento do recebimento na *Cloud*.

Vale ressaltar que os alunos que enviaram suas atividades violando a restrição de tempo não tiveram seus arquivos enviados para a *Cloud*, conforme a restrição determinada pelo professor. Por esse motivo, os dados referentes às atividades enviadas com atraso pelos alunos não foram enviados ao Moodle e não foram apresentados nos gráficos deste estudo.

Para mais detalhes, os dados relacionados ao envio de arquivos estão listados na Tabela 3 no final deste trabalho. A tabela 3 contém o registro de 177 máquinas em três casos iniciais que representam os cenários (100%, 46% e 10% taxa de upload), em que os arquivos são enviados diretamente para a *Cloud* e a proposta em que os arquivos são mantidos em *Fog* e depois são direcionados para a *Cloud*. O termo "início" refere-se a quando o arquivo é enviado e o termo "fim" quando chega ao seu destino final. A linha do tempo desta tabela considera os 120 minutos referentes às atividades dos alunos até que o arquivo chegue ao seu destino final. As cores representando cada um dos polos, seguindo o mesmo padrão de cores definido na Tabela 2 e o tempo é apresentado em minutos. Por

fim, em negrito e vermelho, são destacados os casos que ultrapassaram 120 minutos.

Como buscamos uma solução com baixo custo de implantação, foi analisado o consumo da memória para verificar se é viável configurar o Flink para armazenamento em memória sem investimento em equipamentos. Os gráficos das Figuras 28, 29 e 30 mostram o consumo de memória (disco), considerando os fatores de 10%, 46% e 100% , respectivamente. Observamos que o maior volume de armazenamento foi de aproximadamente 250 MB quando o fator de download e upload para unidades fora do campus foi de 10% (pior caso).

A maioria das unidades fora do campus tem picos de baixo consumo de memória independente do fator de download e upload, indicando que não é necessário investir grandes recursos em infraestrutura para implantar o framework como uma solução definitiva.

Assim, verificamos que o framework pode garantir que todos os alunos que cumpriram o prazo para envio das atividades tenham seus arquivos enviados ao Moodle. Além disso, foi verificado para que todos os alunos tenham o mesmo prazo para desenvolver a atividade sem perder arquivos com a cooperação *Fog-Cloud*. Não seria necessário investir grandes quantias em infraestrutura para realizar este estudo, pois os hardwares da unidade fora do campus possuem poder computacional suficiente para serem usados como servidor local. Com isso, podemos dizer que o framework atingiu os objetivos deste trabalho, respondendo a segunda questão de pesquisa secundaria.

5.2.6 Conclusão do Experimento

Este experimento revela limitações do modelo computacional em *Cloud*, considerando a abordagem do ambiente de ensino a distância em um estudo de caso real no Brasil. Nele a camada *Fog* foi apresentada como uma extensão que aborda as lacunas encontradas no paradigma *Cloud*. A *Fog* foi empregada para dois propósitos principais: fornecer uma resposta rápida a um usuário comum e desafogar os recursos computacionais da infraestrutura primária do sistema.

O framework aborda a cooperação *Fog-Cloud*. Assim, o sistema original continua funcionando com o servidor em *Cloud* (localizado na universidade), mas a grande inovação foi a adoção de nós *Fog* (localizados nos polos), próximos à fonte de dados, onde poderiam ser instalados. Essa configuração permite um baixo número de saltos na rede e, em princípio, um hardware com maior disponibilidade para processar os dados dos alunos.

Este framework foi planejado para validar a proposta e resolver um problema real no contexto educacional. A avaliação foi realizada considerando uma conexão de internet fraca como fatores debilitantes para a adoção de uma arquitetura de *Cloud* clássica. Primeiro, considerando parâmetros como questões econômicas, estruturais e geográficas, verificou-se que alguns polos não possuíam uma conexão razoável o suficiente para realizar as tarefas como deveria.

Os resultados obtidos na avaliação revelam que nossa proposta é uma abordagem útil para auxiliar alunos e professores quanto às regras de restrição de tempo da modalidade ensino a distância. A proposta resolve o problema de Qualidade de Experiência (QoE), evitando a perda de arquivos na submissão de tarefas dos alunos. Para determinar se nossos objetivos foram alcançados, colocamos uma questão central de pesquisa e a dividimos em duas questões secundárias. Cada uma delas é respondida a seguir.

QPS1 - *A cooperação Fog-Cloud é capaz de auxiliar os alunos no envio de suas atividades, garantindo que o sistema reconheça o momento do envio?*

Em relação à solução *Fog-Cloud*, o framework foi eficiente no processamento das atividades localmente com base no tempo de processamento mostrado na seção anterior. Como a confirmação do armazenamento na *Cloud* não impacta no processo de envio de atividades do aluno, todos os dados foram processados rapidamente, garantindo que todos os alunos pudessem enviar suas atividades dentro do prazo do professor. Isso nos permite afirmar que a estrutura poderia processar todos os arquivos enviados para essas cinco unidades fora do campus.

QPS2 - *O framework proposto é capaz de gerenciar o envio das atividades para a Cloud sem necessidade de grandes investimentos de recursos financeiros?*

A análise dos resultados obtidos durante o envio das atividades permite afirmar que o framework conseguiu receber os arquivos, processá-los e armazená-los localmente, sem perda de informações e de forma transparente para os alunos e o professor. O Activity Streaming garantiu que todas as atividades fossem entregues e armazenadas permanentemente no Moodle, mesmo aquelas submetidas quase no prazo estabelecido pelo professor.

Por outro lado, observou-se que é possível flexibilizar a configuração dos nós *Fog*, no que diz respeito ao armazenamento em memória ou em disco, já que o uso de memória é baixo, mesmo nos piores casos.

Nossos experimentos destacaram problemas no ambiente real, ou seja, forneceram evidências e confirmaram a hipótese original relacionada às limitações da configuração da *Cloud*. O sistema original estava próximo ao limite, principalmente no que diz respeito à expansão da estrutura e ao número de polos. A cooperação *Fog-Cloud* é apresentada como uma solução valiosa uma vez que tem como premissa evitar a sobrecarga.

A estrutura proposta confirma claramente que é uma solução funcional viável e escalável para resolver o problema e que pode até prolongar a vida útil da estrutura de um polo. O experimento prova que o paradigma *Fog* pode ajudar a infraestrutura *Cloud* e expõem como essas camadas podem cooperar.

QPP - ***Como contornar os problemas relacionados a horário de recebimento das atividades enviadas pelos alunos nos polos educacionais, consi-***

derando as regras de restrição de tempo na modalidade ensino a distância?

A Tabela 3 apresenta os resultados numéricos apresentados nos gráficos das Figuras 20, 21, 22 e 24

	CLOUD						FOG-CLOUD			CLOUD						FOG-CLOUD	
	10%		46%		100%		100%			10%		46%		100%		100%	
	Begin	End	Begin	End	Begin	End	Begin	End		Begin	End	Begin	End	Begin	End	Begin	End
	117,4	119,8	118,0	118,8	113,5	114,7	113,5	113,6		116,9	**186,7**	115,8	117,1	115,8	116,1	115,8	115,8
	117,6	119,5	112,6	113,5	115,2	115,5	115,2	115,2		115,4	**146,7**	119,9	**121,1**	115,6	116,9	115,6	115,7
	115,7	117,1	112,9	113,2	115,2	116,7	115,2	115,4		117,2	**165,3**	118,2	119,1	115,9	116,2	115,9	115,9
	116,0	117,1	119,1	119,6	118,3	119,6	118,3	118,4		111,2	**184,0**	112,9	113,8	116,1	117,0	116,1	116,2
	119,3	**122,4**	115,6	116,6	118,1	118,4	118,1	118,1		118,8	**157,5**	113,2	113,4	112,5	113,8	112,5	112,6
	116,8	120,0	115,6	116,9	119,9	**121,4**	119,9	**120,1**		116,0	**174,8**	119,3	119,8	111,1	112,0	111,1	111,2
	117,3	118,3	119,7	**120,9**	118,8	119,8	118,8	118,8		116,5	**134,4**	115,8	116,8	119,8	**120,9**	119,8	119,9
	111,1	111,8	118,0	118,9	117,7	119,2	117,7	117,8		110,9	**124,4**	115,8	117,1	116,9	117,9	116,9	117,0
	113,7	114,5	112,7	113,6	119,9	**121,4**	119,9	120,0		112,2	197,1	120,0	**121,2**	119,2	**120,4**	119,2	119,3
	118,8	**122,1**	113,0	113,2	116,5	118,2	116,5	116,7		116,7	**189,2**	118,2	119,1	115,5	115,7	115,5	115,5
	118,4	119,5	119,1	119,6	110,4	111,1	110,4	110,5		119,9	**204,1**	112,9	113,8	112,2	112,9	112,2	112,3
	118,5	**121,9**	115,6	116,6	120,0	**121,4**	120,0	**120,1**		117,1	**182,7**	113,2	113,4	116,2	118,0	116,2	116,2
	113,4	113,9	115,6	116,9	117,3	118,1	117,3	117,3		118,7	**183,5**	119,4	119,8	115,6	116,9	115,6	115,7
	119,3	**121,8**	119,8	**121,0**	117,5	118,2	117,5	117,5		113,2	**196,0**	115,8	116,8	115,9	117,2	115,9	116,0
	110,1	111,0	118,0	118,9	119,9	**121,0**	119,9	120,0		117,9	173,8	115,9	117,1	117,2	119,0	117,2	117,3
	116,0	118,3	112,7	113,6	115,7	116,2	115,7	115,7		112,1	202,7	120,0	121,2	119,4	120,6	119,4	119,5
	112,8	114,5	113,0	113,2	117,6	117,8	117,6	117,6		112,4	166,2	118,2	119,1	113,8	114,7	113,8	113,8
	117,7	**122,8**	119,2	119,6	118,3	119,5	118,3	118,4		118,9	**193,4**	112,9	113,8	113,8	115,4	113,8	113,9
	111,1	112,1	115,6	116,6	116,4	116,8	116,4	116,4		114,0	**131,8**	113,2	113,5	118,9	**120,8**	118,9	119,0
	111,8	112,2	115,7	116,9	119,6	**121,2**	119,6	119,7		118,8	**197,6**	119,4	119,8	111,5	111,8	111,5	111,6
	119,3	**123,1**	119,8	**121,0**	115,7	116,9	115,7	115,8		113,3	**198,6**	115,9	116,8	118,3	**120,4**	118,3	118,5
	118,7	**120,2**	118,0	118,9	116,3	117,7	116,3	116,4		110,2	**154,5**	115,9	117,1	115,5	116,9	115,5	115,6
	116,8	119,4	112,7	113,6	118,5	119,9	118,5	118,6		110,2	**121,6**	120,0	**121,2**	110,8	111,5	110,8	110,8
	116,8	118,6	113,0	113,3	117,0	118,4	117,0	117,1		110,6	**186,9**	118,2	119,1	116,6	117,0	116,6	116,6
	117,7	**121,4**	119,2	119,6	117,0	118,6	117,0	117,2		119,0	**200,0**	112,9	113,8	116,4	118,4	116,4	116,5
	119,4	**122,8**	115,7	116,6	117,0	118,4	117,0	117,2		113,4	**189,7**	113,2	113,5	115,8	117,2	115,8	115,9
	115,8	117,0	115,7	116,9	117,6	118,0	117,6	117,6		117,3	**199,6**	119,4	119,9	119,2	**120,8**	119,2	119,3
	117,5	**122,7**	119,8	**121,0**	119,5	120,0	119,5	119,6		119,9	**188,8**	115,9	116,8	116,5	118,5	116,5	116,6
	112,1	113,9	118,0	118,9	115,4	116,0	115,4	115,4		115,7	**199,8**	115,9	117,2	119,7	**120,6**	119,7	119,7
	112,3	113,7	112,7	113,6	117,1	117,9	117,1	117,2		120,0	**203,4**	120,0	**121,2**	112,0	113,8	112,0	112,2
	117,6	**126,1**	113,0	113,3	115,9	116,7	115,9	115,9		118,1	**204,3**	118,3	119,1	117,6	119,6	117,6	117,8
	112,8	**126,4**	119,2	119,7	118,9	**120,4**	118,9	119,0		116,9	**179,5**	112,9	113,8	115,0	116,4	115,0	115,1
	116,3	**141,1**	115,7	116,6	116,2	118,1	116,2	116,4		119,3	**169,6**	113,2	113,5	116,0	118,0	116,0	116,1
	119,6	**137,0**	115,7	117,0	117,2	118,6	117,2	117,3		112,8	**191,5**	119,4	119,9	112,1	112,4	112,1	112,1
	119,9	**137,6**	119,8	**121,0**	119,6	**120,5**	119,6	119,7		111,9	**196,7**	115,9	116,9	119,3	**120,1**	119,3	119,3
	111,9	**138,7**	118,1	118,9	117,1	118,4	117,1	117,2		116,5	**180,6**	115,9	117,2	113,0	114,6	113,0	113,1
	116,6	**122,9**	112,7	113,6	115,4	116,3	115,4	115,4		110,6	**177,4**	120,0	**121,2**	115,8	116,7	115,8	115,9
	117,0	**133,8**	113,0	113,3	117,1	118,5	117,1	117,2		119,7	**196,6**	118,3	119,1	116,4	118,1	116,4	116,5
	111,2	**139,6**	119,2	119,7	119,2	**120,7**	119,2	119,4		117,7	**203,2**	112,9	113,8	111,5	112,8	111,5	111,6
	119,6	**139,9**	115,7	116,7	116,6	118,2	116,6	116,7		116,3	**204,1**	113,2	113,5	115,0	115,5	115,0	115,1
	110,9	117,9	115,7	117,0	119,4	**120,5**	119,4	119,4		115,2	**195,0**	119,4	119,9	117,4	119,2	117,4	117,5
	111,2	118,3	119,8	**121,0**	119,7	**120,8**	119,7	119,8		118,6	**191,1**	115,9	116,9	119,4	**121,2**	119,4	119,5
	116,8	**138,5**	118,1	118,9	117,2	118,1	117,2	117,3		117,6	**179,3**	115,9	117,2	113,2	114,0	113,2	113,2
	118,6	141,3	112,7	113,6	119,5	119,8	119,5	119,6		119,3	**203,8**	120,0	**121,2**	112,0	112,3	112,0	112,0
	115,9	**135,7**	113,0	113,3	113,1	114,6	113,1	113,2		116,1	**167,0**	118,3	119,2	113,6	115,1	113,6	113,7
	119,2	**138,6**	119,2	119,7	119,5	**121,2**	119,5	119,6		119,2	**151,9**	113,0	113,9	113,0	113,8	113,0	113,1
	113,8	**128,4**	115,7	116,7	119,6	120,0	119,6	119,7		117,1	**185,7**	113,3	113,5	116,1	116,5	116,1	116,1
	110,5	**127,2**	115,7	117,0	112,5	113,9	112,5	112,6		117,0	**197,0**	119,4	119,9	115,2	116,1	115,2	115,3
	110,1	**128,1**	119,8	**121,0**	116,3	117,8	116,3	116,4		117,0	**141,4**	115,9	116,9	119,6	**121,1**	119,6	119,7
	116,2	**128,1**	118,1	119,0	115,9	116,1	115,9	115,9		118,5	**199,3**	115,9	117,2	110,2	112,0	110,2	110,3
	116,5	**176,8**	112,8	113,7	119,2	**120,6**	119,2	119,3		115,2	**167,6**	120,0	**121,3**	116,3	117,2	116,3	116,3
	110,8	**171,9**	113,1	113,3	110,3	111,1	110,3	110,4		119,0	**182,7**	118,3	119,2	119,2	119,8	119,2	119,3
	117,7	**155,0**	119,2	119,7	116,6	117,7	116,6	116,7		111,9	**201,7**	113,0	113,9	111,8	112,5	111,8	111,8
	116,7	**173,3**	115,7	116,7	110,4	110,7	110,4	110,4		119,5	**203,8**	113,3	113,5	116,6	117,4	116,6	116,6
	116,9	**180,1**	115,7	117,0	118,8	**120,7**	118,8	118,9		111,6	**153,9**	119,5	119,9	113,9	114,9	113,9	113,9
	110,1	114,1	119,9	**121,1**	111,3	112,9	111,3	111,4		119,1	**200,1**	115,9	116,9	117,9	118,4	117,9	117,9
	118,6	**185,0**	118,1	119,0	117,4	118,1	117,4	117,4		118,3	**181,2**	116,0	117,2	118,8	119,6	118,8	118,8
	113,2	**162,1**	112,8	113,7	110,3	111,2	110,3	110,4		116,9	**195,0**	120,0	**121,3**	117,5	119,3	117,5	117,6
	111,2	**143,3**	113,1	113,3	111,3	113,1	111,3	111,5		113,3	116,9	118,3	119,2	116,4	117,6	116,4	116,5
	116,2	**173,7**	119,3	119,7	115,1	115,5	115,1	115,2		110,9	112,1	113,0	113,9	118,5	118,8	118,5	118,5
	119,1	**133,2**	115,7	116,7	118,3	118,9	118,3	118,3		115,3	118,5	113,3	113,6	116,2	117,0	116,2	116,3
	115,7	**149,4**	115,8	117,0	118,3	119,9	118,3	118,4		116,1	**129,4**	119,5	119,9	117,8	119,0	117,8	117,9
	111,1	**134,3**	119,9	**121,1**	112,0	112,7	112,0	112,0		117,8	130,0	116,0	116,9	116,9	117,2	116,9	116,9
	118,2	**172,1**	118,1	119,0	117,4	118,7	117,4	117,5		110,9	113,1	116,0	117,2	117,6	118,8	117,6	117,7
	115,4	**170,6**	112,8	113,7	118,5	119,4	118,5	118,6		117,1	**128,7**	120,0	**121,3**	117,1	118,0	117,1	117,2
	118,5	**159,1**	113,1	113,4	110,4	111,0	110,4	110,4		111,4	113,2	118,3	119,2	119,7	**120,8**	119,7	119,8
	115,4	**185,0**	119,3	119,7	119,0	**120,9**	119,0	119,2		119,5	**131,3**	113,0	113,9	117,8	119,5	117,8	118,0
	110,1	113,0	115,8	116,7	110,7	112,0	110,7	110,8		116,3	**130,2**	113,3	113,6	116,1	117,2	116,1	116,2
	110,5	**159,3**	115,8	117,0	119,3	**120,2**	119,3	119,4		113,8	**126,3**	119,5	120,0	118,9	**120,5**	118,9	119,0
	110,3	**150,2**	119,9	**121,1**	115,1	116,7	115,1	115,2		116,2	119,0	116,0	116,9	119,4	**120,9**	119,4	119,6
	117,3	**167,7**	118,1	119,0	117,2	119,1	117,2	117,3		113,6	115,6	116,0	117,3	119,2	**120,8**	119,2	119,4
	119,2	**186,8**	112,8	113,7	117,1	117,7	117,1	117,1		115,9	**130,9**	120,0	**121,3**	119,7	**121,3**	119,7	119,9
	116,4	**186,0**	113,1	113,4	117,7	118,5	117,7	117,8		111,6	112,3	118,4	119,2	118,3	119,6	118,3	118,4
	119,6	**178,0**	119,3	119,8	118,2	119,8	118,2	118,3		115,9	**128,4**	113,0	113,9	118,1	119,5	118,1	118,2
	118,7	**187,2**	115,8	116,7	115,7	117,0	115,7	115,8		116,4	119,4	113,3	113,6	117,4	118,5	117,4	117,5
	110,2	**123,4**	115,8	117,1	118,2	119,0	118,2	118,2		116,6	**124,8**	119,5	120,0	114,0	114,9	114,0	114,0
	110,2	**126,6**	119,9	**121,1**	112,9	113,3	112,9	112,9		116,3	130,7	116,0	117,0	117,0	118,0	117,0	117,1
	116,5	**168,9**	118,2	119,0	111,8	112,5	111,8	111,8		119,6	**129,8**	116,0	117,3	116,5	116,8	116,5	116,5
	116,9	**187,4**	112,8	113,7	115,1	116,6	115,1	115,2		115,6	**126,1**	120,0	**121,3**	110,2	111,8	110,2	110,3
	115,4	**187,0**	113,1	113,4	116,6	117,7	116,6	116,7		120,0	**131,4**	118,4	119,2	115,5	117,1	115,5	115,6
	117,7	**186,2**	119,3	119,8	113,8	115,2	113,8	114,0		113,7	114,7	113,0	113,9	112,2	113,8	112,2	112,3
	117,3	**172,3**	115,8	116,8	119,6	**121,5**	119,6	119,7		117,2	**127,4**	113,3	113,6	115,1	116,7	115,1	115,2
	115,2	**160,1**	115,8	117,1	116,7	118,2	116,7	116,9		119,7	**130,2**	119,5	120,0	111,0	111,7	111,0	111,1
	115,5	**180,9**	119,9	**121,1**	118,6	**120,4**	118,6	118,7		117,1	**130,3**	116,0	117,0	116,0	116,3	116,0	116,0
	117,2	**180,9**	118,2	119,0	118,4	118,9	118,4	118,5		115,1	118,1	116,0	117,3	112,6	113,7	112,6	112,7
	113,7	**148,4**	112,8	113,7	111,1	112,5	111,1	111,2		113,5	115,1	120,0	**121,3**	116,7	118,4	116,7	116,9
	119,3	**183,3**	113,1	113,4	113,8	114,6	113,8	113,9		117,9	**128,9**	118,4	119,3	117,2	118,9	117,2	117,3
	116,2	**137,2**	119,3	119,8	119,1	**120,5**	119,1	119,2		116,6	**125,6**	113,1	114,0	116,1	117,4	116,1	116,2
	119,4	**183,0**	115,8	116,8	117,9	118,3	117,9	117,9									

Tabela 3 – Inicio e final de tempo de *upload* para cada um dos polos educacionais.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

A computação em *Cloud* é um paradigma que se tornou tão presente que atualmente é difícil pensar em soluções que não podem ser encontradas na *Cloud*, e aquelas que ainda não são encontradas, ou já estão sendo desenvolvidas ou não podem se apoiar nessa tecnologia por algum motivo. A *Cloud*, de forma geral, é uma estrutura com sistemas grandes e centralizados para lidar com uma grande carga de trabalho gerado pela demanda, com uma enorme variedade de serviços com alta disponibilidade. Entretanto, como resultado negativo, é observada menor disponibilidade de recursos com o aumento do fluxo de solicitações e, principalmente, uma distância média alta para os clientes do serviço, o que no final das contas gera atraso.

A camada *Fog* vem de encontro às problemáticas da *Cloud*, como um aliado, aproximando o usuário do sistema de serviços, diminuindo atrasos, aumentando disponibilidade e até mesmo diminuindo a carga de trabalho que vai pra *Cloud*. É aí que esse trabalho se apoia, fomentando o desenvolvimento para a abordagem *Fog-Cloud*, em especial tratando do processamento e gerenciamento do fluxo de dados que passam por esse ambiente.

Para o desenvolvimento dessa solução foi necessário verificar na literatura trabalhos que apontassem caminhos que poderiam ser seguidos, mas foi observada uma lacuna em pesquisa a respeito da camada *Fog* como solução. Dessa forma, as contribuições apresentadas neste trabalho tem o intuito de abrir novas portas no campo da pesquisa de sistemas na borda uma vez que apresentam resultados experimentais importantes.

A proposta deste trabalho foi apresentar uma soluções experimentais através de uma abordagem *Fog-Cloud* com o uso de um framework para processamento de dados na *Fog*. Este framework apresenta um fluxo que guia o funcionamento geral de processamento de dados no ambiente. Ele foi aplicado em dois experimentos distintos, um em ambiente controlado, teórico, com o objetivo de descobrir a capacidade computacional do ambiente, e outro prático, resolvendo um problema que atinge usuários reais.

No primeiro experimento, foram executados testes em um hardware típico da camada *Fog*, com objetivo de verificar consumo dos recursos computacionais e poder de processamento nessa camada. Entre as ferramentas de processamento de dados escolhidas (Storm, Spark e Flink) observamos uma pequena vantagem com relação a consumo de recursos do Spark, com o Storm e o Flink muito próximos. Quanto ao volume de dados processados, o Spark se destaca, seguido de perto pelo Flink e bastante atrás o Storm. O comparativo das ferramentas nessa aplicação destaca o Spark, seguido do Flink, com o Storm bem atrás.

Ainda no primeiro experimento, comparamos o sistema do Raspberry Pi 4b com um processador octa-core, com 16 threads, que nesse caso é equiparado com um cluster na *Cloud*. Foi observado comportamento semelhantes entre os ambientes e, apesar do

consumo de recursos se apresentar de forma mais estável no cluster, os resultados são semelhantes. Uma diferença aqui é que o Flink se destaca, um pouco a frente do Spark quanto a quantidade de elementos processados, demonstrando que o ambiente pode fazer diferença nesse resultado.

Ainda falando de elementos processados, o cluster apresenta resultados melhores (pouco mais de duas vezes o resultado da mesma ferramenta na camada *Fog*), mas que não é diretamente proporcional a diferença de recursos computacionais. Dessa forma, há indícios de que um ambiente com múltiplos hardwares na *Fog* pode atingir resultados muito interessantes.

Em relação ao segundo experimento, foi verificada a aplicação do paradigma *Fog* em um problema real. Neste caso, as condições de infraestrutura de conexão tornavam a aplicação de EAD muito penosa para os usuários. Como resposta a *Fog* apresentaria um sistema que poderia se manter enquanto houvesse as oscilações da rede. Foi observado, através do experimento, que as mudanças propostas podem melhorar a qualidade de experiência dos usuários, e a disponibilidade do sistema, que são as principais motivações de desenvolver uma solução para o problema apresentado.

Como trabalhos futuros é interessante realizar, no primeiro experimento, um estudo sob a óptica do consumo elétrico das abordagens e realizar testes em um sistema ainda mais heterogêneo, reforçando a *Fog* como um ambiente bastante aberto a possibilidades de hardware. Para o segundo trabalho seria interessante colocar o sistema em produção e observar os avanços em confiabilidade do sistema e qualidade de experiência.

6.1 CONTRIBUIÇÕES

Durante o desenvolvimento deste trabalho, parte dos resultados foram publicados em artigos científicos. Estes resultados são indicadores que o trabalho tem apelo na comunidade científica.

O artigo Larcher, Ströele e Dantas (2020) apresenta uma solução para um problema encontrado no sistema de educação a distância devidas as condições precárias de infraestrutura relacionadas à conexão. Alunos de polos educacionais ligados a universidade por vezes não conseguem submeter suas tarefas através do ambiente disponibilizado pela universidade em que eles são afiliados. Isso gera problemas como atraso, necessidade de enviar as tarefas por outros meios não oficiais, reduz a qualidade da experiência do usuário, adiciona carga de trabalho aos professores e tutores que lidam com as turmas do polo.

A resposta encontrada para resolver o problema é primorando ambiente remoto com a aplicação do paradigma *Fog-Cloud* com a transformação do polo educacional em uma camada *Fog*. Adicionar um sistema inteligente nessa estrutura, é possível resolver problemas de confiabilidade, disponibilidade, manutenção dos dados, qualidade de experiência, entre

outros. A simulação com dados reais dá indícios que a solução em produção resolve não só o problema local, como de diversos que também pode ser aplicada.

Ambient Assisted Living (AAL) são sistemas técnicos para apoiar os idosos e doentes na sua rotina diária para permitir e promover um estilo de vida independente e seguro. Esse paradigma utiliza objetos e tecnologias inteligentes para facilitar os processos de monitoramento. Este processo também envolve sistemas capazes de processar dados relevantes nas bordas (*Fog*) e estabelecer canais de comunicação rápido entre usuários, seu ambiente e cuidadores (DOHR et al., 2010).

Em Larcher et al. (2020) apresenta a cooperação *Fog-Cloud* como solução de sistema de processamento para o AAL, mantendo o hardware próximo ao usuário, garantindo maior disponibilidade, menor tempo de resposta e menor consumo de recursos na *Cloud*. Durante o experimento é apresentado um framework de contendo o fluxo dos dados, passando na obtenção dos dados, processamento em uma FPSD na *Fog* com uma aplicação de aprendizado de máquina, até o envio dos dados enriquecidos para *Cloud*. Neste processo, análises de tempo de resposta e resultados do enriquecimento são apresentados, demonstrando a competência do ambiente *Fog*, e como essa cooperação *Fog-Cloud* pode ser importante dentro de diversos cenários e aplicações.

O trabalho Larcher, Ströele e Dantas (2021) é uma evolução de Larcher, Ströele e Dantas (2020), trazendo as ferramentas de processamento de streaming de dados para a abordagem, pensando em enriquecimento de dados e tolerância a falhas. Além disso, foi adicionado aos experimentos a visão de consumo de espaço em disco na camada *Fog*. Aqui também tem uma discussão sobre a manutenção dos dados na *Fog*, o consumo no experimento e a discussão sobre os resultados. Aqui verificamos que o volume não é grande o suficiente para sobrecarregar hardwares mais simples, como o computador de placa única.

REFERÊNCIAS

AHMED, Arif; PIERRE, Guillaume. Docker container deployment in fog computing infrastructures. In: IEEE COMPUTER SOCIETY. **2018 IEEE International Conference on Edge Computing (EDGE)**. [S.l.], 2018. p. 1–8.

ALAM, Muhammad; RUFINO, Joao; FERREIRA, Joaquim; AHMED, Syed Hassan; SHAH, Nadir; CHEN, Yuanfang. Orchestration of microservices for iot using docker and edge computing. **IEEE Communications Magazine**, IEEE, v. 56, n. 9, p. 118–123, 2018.

ALEXANDROV, Alexander; BERGMANN, Rico; EWEN, Stephan; FREYTAG, Johann-Christoph; HUESKE, Fabian; HEISE, Arvid; KAO, Odej; LEICH, Marcus; LESER, Ulf; MARKL, Volker et al. The stratosphere platform for big data analytics. **The VLDB Journal**, Springer, v. 23, n. 6, p. 939–964, 2014.

ATLAM, Hany F; WALTERS, Robert J; WILLS, Gary B. Fog computing and the internet of things: A review. **big data and cognitive computing**, Multidisciplinary Digital Publishing Institute, v. 2, n. 2, p. 10, 2018.

BAKER, Mark. Cluster computing white paper. **arXiv preprint cs/0004014**, 2000.

BATTULGA, Davaadorj; MIORANDI, Daniele; TEDESCHI, Cédric. Fogguru: a fog computing platform based on apache flink. In: IEEE. **2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)**. [S.l.], 2020. p. 156–158.

BLAKE, Geoffrey; DRESLINSKI, Ronald G; MUDGE, Trevor. A survey of multicore processors. **IEEE Signal Processing Magazine**, IEEE, v. 26, n. 6, p. 26–37, 2009.

BRIAN, Hayes; BRUNSCHWILER, Thomas; DILL, Heinz; CHRIST, Hanspeter; FALSAFI, Babak; FISCHER, Markus; GRIVAS, Stella Gatziu; GIOVANOLI, Claudio; GISI, Roger Eric; GUTMANN, Reto et al. Cloud computing. **Communications of the ACM**, v. 51, n. 7, p. 9–11, 2008.

CHINTAPALLI, Sanket; DAGIT, Derek; EVANS, Bobby; FARIVAR, Reza; GRAVES, Thomas; HOLDERBAUGH, Mark; LIU, Zhuo; NUSBAUM, Kyle; PATIL, Kishorkumar; PENG, Boyang Jerry et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In: IEEE. **2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)**. [S.l.], 2016. p. 1789–1792.

COSTA, Felipe S; NASSAR, Silvia M; DANTAS, Mario AR. Focuser: A fog online context-aware up-to-date sensor ranking method. **Journal of Sensor and Actuator Networks**, MDPI, v. 11, n. 2, p. 25, 2022.

DEAN, Jeffrey; GHEMAWAT, Sanjay. Mapreduce: simplified data processing on large clusters. **Communications of the ACM**, ACM New York, NY, USA, v. 51, n. 1, p. 107–113, 2008.

DING, Shifei; WU, Fulin; QIAN, Jun; JIA, Hongjie; JIN, Fengxiang. Research on data stream clustering algorithms. **Artificial Intelligence Review**, Springer, v. 43, n. 4, p. 593–600, 2015.

DOHR, Angelika; MODRE-OPSRIAN, Robert; DROBICS, Mario; HAYN, Dieter; SCHREIER, Günter. The internet of things for ambient assisted living. In: IEEE. **2010 seventh international conference on information technology: new generations**. [S.l.], 2010. p. 804–809.

GAMA, João; GABER, Mohamed Medhat. **Learning from data streams: processing techniques in sensor networks**. [S.l.]: Springer, 2007.

GARCÍA-GIL, Diego; RAMÍREZ-GALLEGO, Sergio; GARCÍA, Salvador; HERRERA, Francisco. A comparison on scalability for batch big data processing on apache spark and apache flink. **Big Data Analytics**, BioMed Central, v. 2, n. 1, p. 1–11, 2017.

GEHRKE, Johannes; KORN, Flip; SRIVASTAVA, Divesh. On computing correlated aggregates over continual data streams. **ACM SIGMOD Record**, ACM New York, NY, USA, v. 30, n. 2, p. 13–24, 2001.

GIL, Antonio Carlos. Como elaborar projetos de pesquisa. 12 reimpr, atlas. **São Paulo**, 1991.

GOSLING, James; MCGILTON, Henry. The java language environment. **Sun Microsystems Computer Company**, v. 2550, p. 38, 1995.

GROUP, OpenFog Consortium Architecture Working et al. Openfog reference architecture for fog computing. **OPFRA001**, OpenFog Consortium Fremont, CA, USA, v. 20817, p. 162, 2017.

HAN, Jiawei; PEI, Jian; KAMBER, Micheline. **Data mining: concepts and techniques**. [S.l.]: Elsevier, 2011.

HIPPEL, Eric Von. Learning from open-source software. **MIT Sloan management review**, v. 42, n. 4, p. 82–86, 2001.

HONG, Cheol-Ho; VARGHESE, Blesson. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 52, n. 5, p. 1–37, 2019.

IJTIHADIE, Royyana M; HIDAYANTO, Bekti C; AFFANDI, Achmad; CHISAKI, Yoshifumi; USAGAWA, Tsuyoshi. Dynamic content synchronization between learning management systems over limited bandwidth network. **Human-Centric Computing and Information Sciences**, Springer, v. 2, n. 1, p. 1–16, 2012.

IORGA, Michaela; FELDMAN, Larry; BARTON, Robert; MARTIN, Michael J; GOREN, Nedim S; MAHMOUDI, Charif et al. Fog computing conceptual model. National Institute of Standards and Technology, 2018.

IQBAL, Muhammad Hussain; SOOMRO, Tariq Rahim et al. Big data analysis: Apache storm perspective. **International journal of computer trends and technology**, Citeseer, v. 19, n. 1, p. 9–14, 2015.

ISAH, Haruna; ABUGHOFA, Tariq; MAHFUZ, Sazia; AJERLA, Dharmitha; ZULKERNINE, Farhana; KHAN, Shahzad. A survey of distributed data stream processing frameworks. **IEEE Access**, IEEE, v. 7, p. 154300–154316, 2019.

ISIKDAG, Umit. Internet of things: Single-board computers. In: **Enhanced Building Information Models**. [S.l.]: Springer, 2015. p. 43–53.

JAGGAR, Dave. Arm architecture and systems. **IEEE micro**, IEEE Computer Society, v. 17, n. 04, p. 9–11, 1997.

JOHNSTON, Steven J; BASFORD, Philip J; PERKINS, Colin S; HERRY, Herry; TSO, Fung Po; PEZAROS, Dimitrios; MULLINS, Robert D; YONEKI, Eiko; COX, Simon J; SINGER, Jeremy. Commodity single board computer clusters and their applications. **Future Generation Computer Systems**, Elsevier, v. 89, p. 201–212, 2018.

KAUP, Fabian; HACKER, Stefan; MENTZENDORFF, Eike; MEURISCH, Christian; HAUSHEER, David. The progress of the energy-efficiency of single-board computers. **Tech. Rep. NetSys-TR-2018-01**, 2018.

KITCHENHAM, Barbara; CHARTERS, Stuart. Guidelines for performing systematic literature reviews in software engineering. Citeseer, 2007.

KRAWCZYK, Bartosz; MINKU, Leandro L; GAMA, João; STEFANOWSKI, Jerzy; WOŹNIAK, Michał. Ensemble learning for data stream analysis: A survey. **Information Fusion**, Elsevier, v. 37, p. 132–156, 2017.

LAGHARI, Asif Ali; JUMANI, Awais Khan; LAGHARI, Rashid Ali. Review and state of art of fog computing. **Archives of Computational Methods in Engineering**, Springer, v. 28, n. 5, p. 3631–3643, 2021.

LARCHER, Lucas; STRÖELE, Victor; DANTAS, Mario. A fog-cloud approach to enhance communication in a distance learning cloud based system. In: SPRINGER. **Advances on P2P, Parallel, Grid, Cloud and Internet Computing: Proceedings of the 14th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2019) 14**. [S.l.], 2020. p. 275–286.

LARCHER, Lucas; STRÖELE, Victor; DANTAS, Mário. A cloud-based system for distance learning supported by fog-cloud cooperation. **International Journal of Grid and Utility Computing**, Inderscience Publishers (IEL), v. 12, n. 5-6, p. 618–634, 2021.

LARCHER, Lucas; STRÖELE, Victor; DANTAS, Mário; BAUER, Michael. Event-driven framework for detecting unusual patterns in aal environments. In: IEEE. **2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS)**. [S.l.], 2020. p. 309–314.

LASI, Heiner; FETTKE, Peter; KEMPER, Hans-Georg; FELD, Thomas; HOFFMANN, Michael. Industry 4.0. **Business & information systems engineering**, Springer, v. 6, n. 4, p. 239–242, 2014.

LEE, Hwejoo; OH, Junghyun; KIM, Kyungrae; YEON, Hunje. A data streaming performance evaluation using resource constrained edge device. In: IEEE. **2017 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.], 2017. p. 628–633.

LUU, Hien. **Beginning Apache Spark 2: with resilient distributed datasets, Spark SQL, structured streaming and Spark machine learning library**. [S.l.]: Apress, 2018.

MELL, Peter; GRANCE, Tim et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

MORALES, Gianmarco De Francisci; BIFET, Albert; KHAN, Latifur; GAMA, Joao; FAN, Wei. Iot big data stream mining. In: **Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining**. [S.l.: s.n.], 2016. p. 2119–2120.

NASIR, Mansoor; MUHAMMAD, Khan; LLORET, Jaime; SANGAIAH, Arun Kumar; SAJJAD, Muhammad. Fog computing enabled cost-effective distributed summarization of surveillance videos for smart cities. **Journal of Parallel and Distributed Computing**, Elsevier, v. 126, p. 161–170, 2019.

NGOM, Babacar; GUILLERMET, Hervé; NIANG, Ibrahima. Enhancing moodle for offline learning in a degraded connectivity environment. In: IEEE. **2012 international conference on multimedia computing and systems**. [S.l.], 2012. p. 858–863.

RABL, Tilmann; TRAUB, Jonas; KATSIFODIMOS, Asterios; MARKL, Volker. Apache flink in current research. **it-Information Technology**, De Gruyter Oldenbourg, v. 58, n. 4, p. 157–165, 2016.

RADOJEVIĆ, Branko; ŽAGAR, Mario. Analysis of issues with load balancing algorithms in hosted (cloud) environments. In: IEEE. **2011 Proceedings of the 34th international convention MIPRO**. [S.l.], 2011. p. 416–420.

RICHARDSON, Matt; WALLACE, Shawn. **Getting started with raspberry PI**. [S.l.]: "O'Reilly Media, Inc.", 2012.

SCOTT, Michael Lee. **Programming language pragmatics**. [S.l.]: Morgan Kaufmann, 2000.

SEVERANCE, Charles. The apache software foundation: Brian behlendorf. **Computer**, IEEE, v. 45, n. 10, p. 8–9, 2012.

SHAHVERDI, Elkhan; AWAD, Ahmed; SAKR, Sherif. Big stream processing systems: an experimental evaluation. In: IEEE. **2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)**. [S.l.], 2019. p. 53–60.

SILVA, Edna Lucia Da; MENEZES, Estera Muszkat. Metodologia da pesquisa e elaboração de dissertação. **UFSC, Florianópolis, 4a. edição**, v. 123, 2005.

SILVA, Jonathan A; FARIA, Elaine R; BARROS, Rodrigo C; HRUSCHKA, Eduardo R; CARVALHO, André CPLF de; GAMA, João. Data stream clustering: A survey. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 46, n. 1, p. 1–31, 2013.

SINGH, Jagdeep; SINGH, Parminder; GILL, Sukhpal Singh. Fog computing: A taxonomy, systematic review, current trends and research challenges. **Journal of Parallel and Distributed Computing**, Elsevier, v. 157, p. 56–85, 2021.

STONEBRAKER, Michael; ÇETINTEMEL, Uğur; ZDONIK, Stan. The 8 requirements of real-time stream processing. **ACM Sigmod Record**, ACM New York, NY, USA, v. 34, n. 4, p. 42–47, 2005.

SUNYAEV, Ali. Cloud computing. In: **Internet computing**. [S.l.]: Springer, 2020. p. 195–236.

TA, Van-Dai; LIU, Chuan-Ming; NKABINDE, Goodwill Wandile. Big data stream computing in healthcare real-time analytics. In: IEEE. **2016 IEEE international conference on cloud computing and big data analysis (ICCCBDA)**. [S.l.], 2016. p. 37–42.

TANENBAUM, Andrew S; WETHERALL, David J et al. **Computer networks. 5th**. [S.l.: s.n.], 2010.

THUSOO, Ashish; SARMA, Joydeep Sen; JAIN, Namit; SHAO, Zheng; CHAKKA, Prasad; ANTHONY, Suresh; LIU, Hao; WYCKOFF, Pete; MURTHY, Raghotham. Hive: a warehousing solution over a map-reduce framework. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 2, n. 2, p. 1626–1629, 2009.

TOOR, Affan Ahmed; USMAN, Muhammad; YOUNAS, Farah; FONG, Alvis Cheuk M; KHAN, Sajid Ali; FONG, Simon. Mining massive e-health data streams for iomt enabled healthcare systems. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 20, n. 7, p. 2131, 2020.

TORVALDS, Linus. The linux edge. **Communications of the ACM**, ACM New York, NY, USA, v. 42, n. 4, p. 38–39, 1999.

VISHWANATH, Kashi Venkatesh; NAGAPPAN, Nachiappan. Characterizing cloud computing hardware reliability. In: **Proceedings of the 1st ACM symposium on Cloud computing**. [S.l.: s.n.], 2010. p. 193–204.

WEN, Zhenyu; YANG, Renyu; GARRAGHAN, Peter; LIN, Tao; XU, Jie; ROVATSOS, Michael. Fog orchestration for internet of things services. **IEEE Internet Computing**, IEEE, v. 21, n. 2, p. 16–24, 2017.

XIA, Feng; YANG, Laurence T; WANG, Lizhe; VINEL, Alexey. Internet of things. **International journal of communication systems**, v. 25, n. 9, p. 1101, 2012.

YUN, Unil; KIM, Donggyu; YOON, Eunchul; FUJITA, Hamido. Damped window based high average utility pattern mining over data streams. **Knowledge-Based Systems**, Elsevier, v. 144, p. 188–205, 2018.