UNIVERSIDADE FEDERAL DO ABC

DIVISÃO ACADÊMICA DO CECS

ENGENHARIA DE INFORMAÇÃO

Giselle Silva de Santana

Rodrigo Akiyama Abrantes

Computação de Borda e Bancos de Dados Distribuídos

Análise de métodos de computação de borda e sua aplicação para bancos de dados

Santo André - SP

2021

Giselle Silva de Santana
Rodrigo Akiyama Abrantes

Computação de Borda e Bancos de Dados Distribuídos

Análise de métodos de computação de borda e sua aplicação para bancos de dados

Trabalho de Graduação apresentado ao Curso de Graduação em Engenharia da Informação da Universidade Federal do ABC como requisito parcial para obtenção do título de Engenheiro da Informação.

Orientador: Prof. Dr. Ricardo Suyama

Santo André - SP
2021

RESUMO

Com o avanço da Internet das Coisas (IoT), o volume de dados gerados por dispositivos e sensores têm aumentado a cada dia. Por conta deste crescimento, novas soluções são propostas para o armazenamento e processamento desses dados, em busca de otimização de desempenho, diminuição de latência e redução de custos. Com o uso de bancos de dados distribuídos otimizados para ambientes na borda pode-se atingir uma performance dezenas de vezes maior em situações de carga de dados de séries temporais provenientes de aplicações de IoT. O presente trabalho tem como objetivo estudar as tecnologias de banco de dados aplicadas a contextos de computação de borda e comparar seu desempenho com um banco de dados relacional tradicional.

Palavras-chave: computação de borda; banco de dados; banco de dados distribuídos; internet das coisas; IoT.

ABSTRACT

Due to the advance of the Internet of Things, the data rate generated by devices and sensors has been increasing every day. Because of this phenomenon, new solutions are being proposed for storage and data processing, in search of a performance improvement, latency and cost reduction. With the use of distributed databases improved for edge environments, it is possible to reach a dozens of times higher performance when inserting data loads of time series generated by IoT applications. This paper aims to study the databases technologies applied in an edge computing scenario and compare it with traditional databases.

Keywords: edge computing; databases; distributed databases; internet of things; IoT.

SUMÁRIO

1. INTRODUÇÃO

Nos dias atuais, a utilização de provedores de computação em nuvem (Cloud Computing, em inglês) tem se tornado cada vez mais comum em detrimento da montagem e utilização do que se convencionou a chamar de infraestrutura on-premises, que nada mais é do que datacenters proprietários, devido aos custos associados à manutenção e a lentidão no provisionamento de recursos computacionais para a utilização em aplicações.

Porém, enquanto a nuvem pública obteve êxito em facilitar a obtenção de recursos computacionais sob-demanda, o tempo de latência de rede que é adicionado toda vez que é necessário realizar comunicação entre um dispositivo com o datacenter na nuvem via canais de internet apresenta um novo problema a ser solucionado. Além disso, o rápido desenvolvimento da Internet das Coisas (dispositivos capazes de enviar e receber dados via internet, de forma automatizada) tem possibilitado que uma quantidade cada vez maior de informação seja coletada e posteriormente processada para o uso. Sejam por necessidades de negócio ou por característica de aplicações (radares de velocidade, sensores de sinais vitais), estes processamentos precisam ocorrer cada vez mais rapidamente para que decisões sejam tomadas e a latência de rede associada a transferência destes dados passa a se tornar um problema insustentável.

A grande aposta para mitigar este problema é a utilização da computação de borda, onde se torna possível a realização de processamento completo ou parcial destes dados "on the spot", sendo estes posteriormente transferidos para as centrais de processamento na nuvem ou em infraestruturas on-premises. Os desafios apresentados por este nível de arquitetura de software distribuído, em especial no que tange o armazenamento de informação em bancos de dados e as novas tecnologias que surgiram como solução deste problema são os principais motivadores deste trabalho.

2. OBJETIVO

Este trabalho tem como objetivo a realização da revisão bibliográfica das tecnologias de banco de dados aplicadas a contextos de computação de borda,

assim como apresentar a comparação de resultados obtidos em simulações e experimentos práticos nos tempos de processamento de diferentes gerenciadores de banco de dados localizados na borda.

Para isto, serão abordados e explicitados os conceitos que compõem a computação de borda, seus casos de uso e a adoção atual da tecnologia. Em acréscimo, também serão exploradas as tecnologias dos bancos de dados não-relacionais que permitem a sua utilização em ambientes distribuídos.

Por fim, serão analisados os modelos de bancos de dados distribuídos já propostos na literatura. Serão apresentados simulações e aplicações de cenários de teste para estas tecnologias, exibindo a sua efetividade em situações reais.

3. REVISÃO BIBLIOGRÁFICA

3.1. Contextualização

A computação de borda é um paradigma relacionado à arquitetura de TI que propõe a movimentação do processamento de informação para a borda da rede [1]. Apesar de ser um conceito relativamente antigo, tendo sido citado pela primeira vez em 2005, suas propostas começaram a ser mais difundidas com a ascensão de dispositivos móveis capazes de coletar e transmitir dados por meio de redes de computadores, construindo a fundação do que se convencionou a chamar de Internet das Coisas ou IoT (do inglês, Internet of Things). Além desta interconectividade mencionada, outra característica preponderante é a restrição da capacidade computacional e de armazenamento de dados. Devido a estas duas características, é muito comum a necessidade de conexão destes dispositivos com a internet para que o processamento dos dados por eles coletados seja realizado por servidores remotos, na maioria das vezes sendo utilizados servidores virtuais hospedados em provedores de nuvem [1].

Embora estes provedores disponibilizam acesso imediato a capacidade de processamento sob-demanda e escalável quando há maiores quantidades de informação a ser trabalhada, a distância geográfica entre os dispositivos móveis e os datacenters acabam gerando um gargalo de performance devido às limitações de tráfego da rede e ao tempo de transmissão dos dados [3]. O fato destes dispositivos

utilizarem, em sua maioria, protocolos e tecnologias de comunicação móvel sem fio, como o 3G e/ou 4G, acaba por acrescentar ainda mais latência neste processamento. Um outro desafio é a disponibilidade de conexão estável com a internet para regiões geograficamente remotas ou que careçam de uma infraestrutura de telecomunicações ou de TI robusta o suficiente [2]. Desta forma, torna-se necessário o uso de uma solução de persistência de dados, na forma de um sistema gerenciador de banco de dados, na borda da rede tanto para que se evite a perda de dados em eventuais falhas de funcionalidade de rede, quanto para o processamento e consulta local das informações ali armazenadas durante estes períodos de indisponibilidade.

Desta forma, conforme a Internet das Coisas começa a se tornar o componente central de sistemas de tomada de decisão críticas, como o Smart Grid na área de energia e os sensores de monitoramento de sinais vitais na área da saúde [5], a latência de rede pode tornar inviável a utilização destes sistemas de forma confiável e eficaz, já que as informações processadas por estes sistemas visam a tomada de decisão em tempo real de forma automatizada [3].

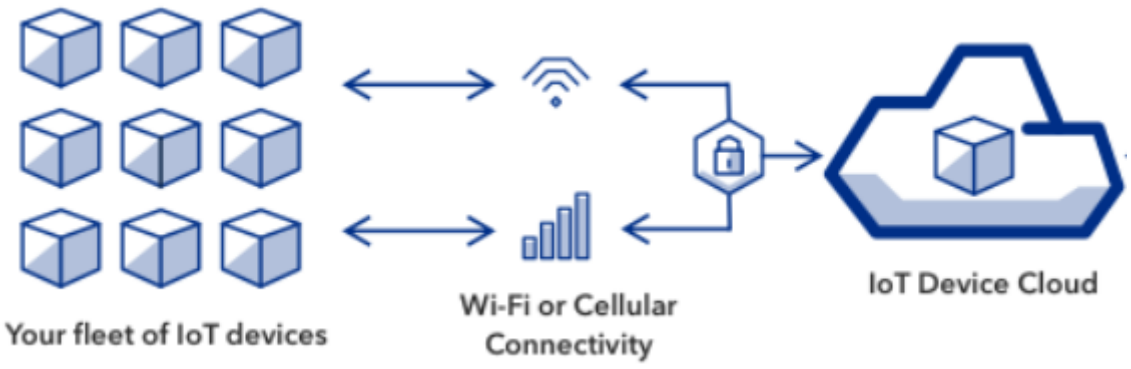

Figura 1: Conectividade entre dispositivos de internet das coisas e os servidores da Cloud [6].

Por este motivo, os autores em [2], ao analisar as previsões de que em 2019 a quantidade de dados produzidas por pessoas, máquinas e objetos atingiria 500 zettabytes, concluíram que o método mais eficiente de se processar os dados gerados na borda da rede seria levar as unidades de processamento até esta camada. Esta arquitetura de sistemas distribuídos, que possui como foco a realização do processamento total ou parcial da informação em regiões de borda da

rede, convencionou-se chamar de Computação de Borda (Edge Computing, em inglês).

Neste trabalho, a computação de borda será definida como todo o tipo de unidade computacional descentralizada que realiza algum tipo de processamento de dados que recebe via transferência em rede, seja ela de computadores ou de telecomunicação. A origem destes dados são dispositivos que possuam capacidade de captar informações do ambiente via sensores ou por meio de interação direta com o usuário, como smartphones, todos pertencentes à Internet das Coisas. O objetivo central de Edge Computing é a mitigação da latência de rede no processamento de dados gerados por estes dispositivos.

As seções seguintes serão dedicadas ao aprofundamento e apresentação das principais arquiteturas para Edge Computing e dos tipos de bancos de dados distribuídos mais indicados para serem aplicados na borda da rede.

3.2. Edge Computing

A Edge Computing (em tradução livre, Computação de Borda) tem como objetivo estender a capacidade de computação da nuvem (mas não limitada apenas a ela) para a borda da rede de forma que estas fiquem mais próximas as fontes dos dados e o tempo de transferência via rede seja reduzido [2], permitindo-se que o processamento de dados seja realizado na borda da rede próximo a sua fonte, funcionando tanto para downstream de dados para informações provenientes da cloud, como para upstream de dados provenientes dos dispositivos de usuários e IoT [18], uma vez que tais dispositivos não apenas produzem dados, como também os consomem.

Edge Computing (EC) possui uma topologia descentralizada, que mantém os dados de maneira local, o mais próximo possível de sua fonte [1] [2] [7], diferentemente de uma estrutura em Cloud Computing, que é inerentemente centralizada [19]. Sua arquitetura consiste em três camadas principais conforme apresentado na figura 2: a camada de dispositivos de borda (edge devices), uma segunda camada contendo os nós de borda (edge nodes) e a cloud compõe a terceira camada. Um nó de borda pode ser qualquer tipo de dispositivo que esteja

entre uma fonte de dados (dispositivo de borda) e um cloud datacenter, como por exemplo, um dispositivo móvel entre um conjunto de sensores e o ambiente cloud, ou uma cloudlet entre um smartphone e a cloud [18].

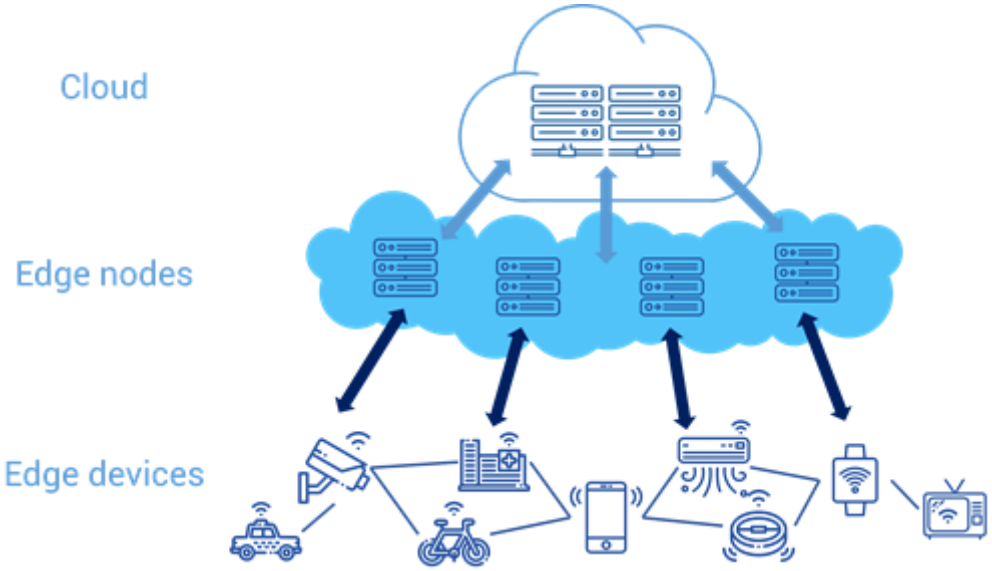

Figura 2: Arquitetura Edge Computing [17].

Além da redução de latência, o uso de Edge Computing pode aumentar a escalabilidade, também podendo causar diminuição de custos ao evitar armazenamento de dados redundantes. Em [4] os autores propõem o ReSIoT, um framework reconfigurável baseado em EC capaz de aumentar a segurança em sistemas IoT através da utilização de um dispositivo de borda, como por exemplo um roteador wireless, como agente de segurança (security agent), trazendo flexibilidade e alta compatibilidade para futuras implantações de novos protocolos de segurança em IoT.

Também podemos citar como exemplo de aplicação de EC o framework Paradrop [7][12], que possibilita a implantação de aplicações em roteadores Wi-Fi, sendo possível desenvolver soluções de Software as a Service próximas ao usuário final. Em [25] propõe-se o uso de Edge Computing para realizar caching e processamento de aplicações de Realidade Aumentada de dispositivos móveis, com o intuito de reduzir a latência.

O princípio de Edge Computing pode ser implementado de diferentes formas, e nas próximas seções deste trabalho, serão apresentados os conceitos de Mobile

Cloud Computing, Cloudlets e Mobile Edge Computing, que são as principais implementações de Computação de Borda presentes na literatura. Além disso, será apresentada uma breve exposição de uma arquitetura de computação para a borda que, como mencionado em [1], muitas vezes se confunde ao de Edge Computing, a Fog Computing.

3.2.1. Mobile Cloud Computing

Este paradigma considera que, devido às limitações de storage e de poder de processamento dos dispositivos móveis, se faz necessário que estas ações sejam realizadas por um agente remoto. De início, considerava-se que apenas os provedores de cloud computing estariam aptos para isto porém, com o desenvolvimento dos dispositivos móveis e em especial os smartphones, estas necessidades computacionais poderiam também ser realizadas pelos próprios dispositivos que se encontram na borda da rede. Com esta evolução, torna-se possível o oferecimento de serviços ou aplicações que não dependem totalmente do acesso à internet para a sua utilização, agindo como aplicações on demand.

Em Cloud Computing, plataformas e aplicações são consideradas como serviços (as a service) e não como produtos, sendo comumente dividida em três principais categorias: Software as a Service (SaaS), Platform as a Service (PaaS) e Infrastructure as a Service (IaaS). Através de SaaS, o usuário é capaz de acessar aplicações como e-mail, redes sociais, editores de texto online, entre outros. Com PaaS, temos por exemplo, o provisionamento de ambientes para desenvolvimento de aplicações e disponibilização de APIs. E com IaaS, podemos citar como exemplo a disponibilização de ambientes para armazenamento e visualização de dados [10].

O conceito de Mobile Cloud Computing (MCC) surgiu da ideia de que dispositivos móveis não deveriam ser responsáveis pelo armazenamento e processamento de seus dados, delegando estes a entidades remotas na cloud. Com o avanço das pesquisas sobre o assunto, o conceito de MCC foi expandido, sendo que nesta nova visão o processamento de dados também pode ser realizado por dispositivos presentes na borda da rede [8]. Uma das implementações mais comuns de MCC é Cloudlet Computing [1], que será descrita mais à frente neste trabalho.

Aplicativos de celular que realizam tradução de linguagens, reconhecimento de voz ou de navegação via GPS são exemplos de utilização de Mobile Cloud Computing [8].

3.2.1.1. Arquitetura Mobile Cloud Computing

Na arquitetura MCC, conforme apresentado na figura 3, temos uma primeira camada que pode ser considerada como camada de usuário ou camada front-end, onde as requisições realizadas pelos usuários são processadas e armazenadas em provedores externos ao dispositivo [15]. Estes provedores podem oferecer serviços como autorização e autenticação. Na camada intermediária, tem-se acesso ao poder computacional que pode estar hospedado em cloudlets, servidores ou redes wireless. Na terceira camada, temos a camada de cloud, que proporciona a infraestrutura de backend, fazendo com que todas as requisições do usuário possam ser respondidas [16].

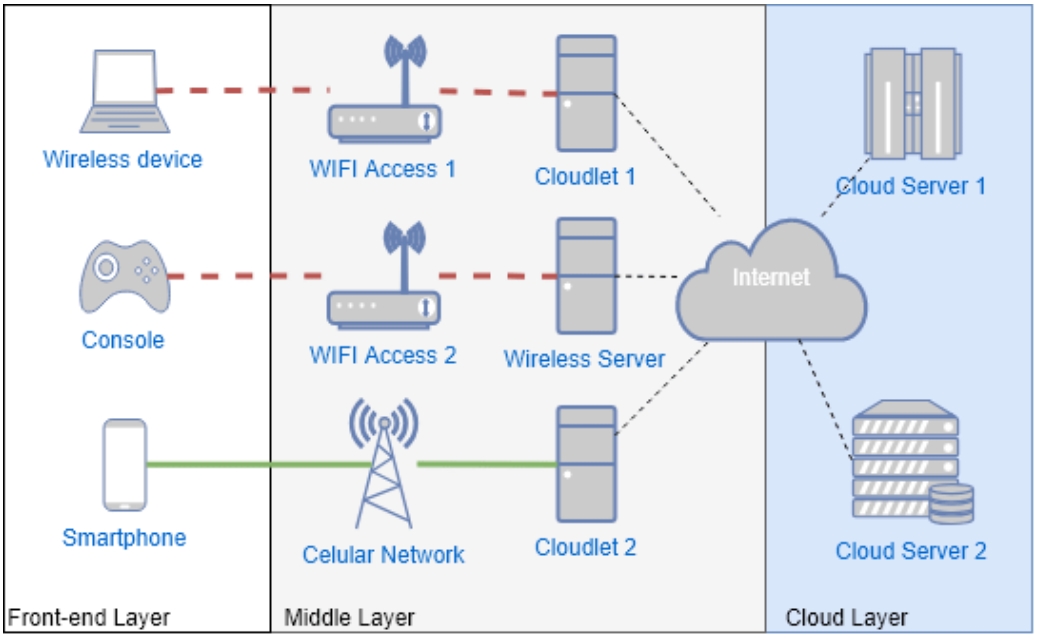

Figura 3: Arquitetura Mobile Cloud Computing [16].

Um exemplo comum de aplicação que se utiliza deste modelo de arquitetura são os assistentes digitais, como o Google Assistant e a Amazon Alexa. Estes softwares utilizam dispositivos como smartphones ou alto-falantes como a camada de front-end para a interação com o usuário, recebendo requisições diversas por voz ou texto livre. Esta entrada então é enviada para servidores remotos para ser

processada por algoritmos de aprendizado de máquina, para que ela então seja compreendida pelo computador e que a ação desejada seja realizada [37] [38].

3.2.1.2. Cloudlets

O termo Cloudlet foi utilizado pela primeira vez em um artigo de 2009 publicado no IEEE [14], em um momento da história onde a capacidade computacional dos dispositivos móveis não era robusta o suficiente para sua utilização em aplicações que exigem alta carga de processamento, como inteligência artificial e realidade aumentada. Além disso, a infraestrutura de telecomunicação para internet móvel ainda não havia atingido o nível de maturidade dos dias atuais e a quarta geração das redes para celular (popularmente chamada de 4G) ainda não havia chegado ao mercado consumidor. A soma da baixa capacidade computacional e a latência das redes móveis tornava próximo de inviável a utilização de dispositivos móveis nas sofisticadas aplicações cognitivas discutidas no artigo.

Para solucionar estes problemas listados, os autores de [14] propuseram a utilização de unidades de processamento descentralizadas que seriam dispostas na borda da rede, mais próximas do usuário final. Estas unidades, então denominadas Cloudlets, agiriam como uma camada de caching e de processamento de dados para dispositivos que não possuem capacidade computacional robusta o suficiente e dependem de agentes externos para processar os dados que eles coletam.

3.2.1.3. Arquitetura de um Cloudlet

Um Cloudlet é composto por uma unidade computacional que é colocada na borda da rede, podendo ela ser desde uma simples placa dedicada para processamento, como um Raspberry Pi, ou um servidor de pequeno porte. Seu objetivo é reduzir a latência da rede para aplicações e dispositivos que se encontram na borda da rede.

Entre suas principais atribuições estão o caching de dados e a realização de cargas de processamentos para dispositivos que não contam com o poder computacional para realizá-los localmente. Esta última característica também é a

responsável por diferenciar Cloudlets das CDN, ou Content Delivery Network (em tradução livre, Rede de Entrega de Conteúdo), que são constituídas por servidores que também são dispostos na borda da rede e são responsáveis exclusivamente por fornecer acesso mais rapidamente a conteúdo estático como vídeos e páginas web.

Por se tratar de uma camada de caching e pelo fato de um Cloudlet estar exposto a condições ambientais adversas, já que estes são dispostos na borda da rede e nem sempre contam com uma infraestrutura que garanta a integridade do seu hardware, não é realizada a persistência ou armazenamento de informação no espaço de armazenamento (storage) embutido nestas unidades. Os dados são armazenados temporariamente apenas para fins de processamento, ficando então a persistência do dado como atribuição dos servidores na nuvem ou dos dispositivos móveis requisitantes.

Os Cloudlets utilizam-se de soluções de virtualização para a montagem dinâmica dos ambientes computacionais de processamento. Há também a possibilidade de se contar com unidades de orquestração de cloudlets em uma área, os cloudlets agents, de forma que possa ser realizado um balanceamento de carga entre o processamento de cada nó que compõe a malha de cloudlets de uma região e também identificar falhas com políticas de segurança e compliance [11].

Os Cloudlets também são capazes de trabalhar em conjunto com os grandes datacenters dos provedores de nuvem para a realização de processamento intensivo ou quando a sua capacidade local é esgotada [1] [8]. Outra vantagem percebida na utilização dos Cloudlets está na redução do consumo de energia dos dispositivos móveis, uma vez que o processamento é realizado de forma remota, e no tempo de latência, já que os Cloudlets estão dispostos na borda da rede e mais próximos destes dispositivos [8].

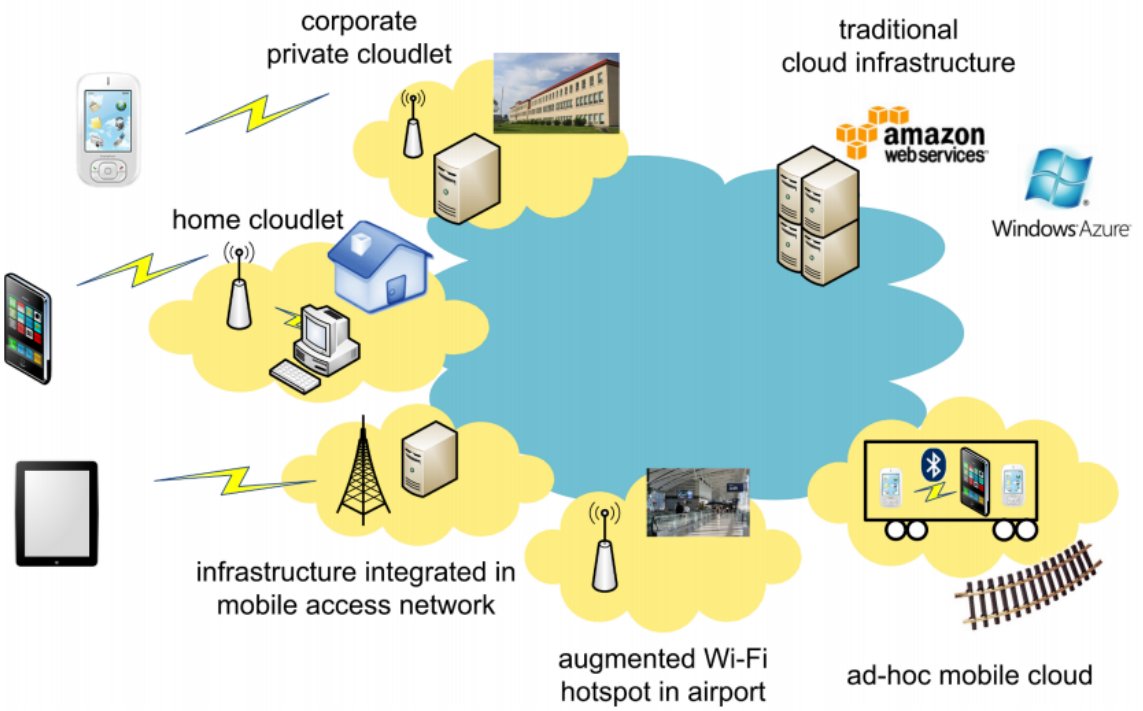

Figura 4: Representação do fluxo de processamento utilizando cloudlets [9].

3.2.2. Mobile Edge Computing

Concebido inicialmente em 2013, em uma parceria da IBM com a Nokia-Siemens que resultou em uma plataforma que permitia execuções e aplicações em uma infraestrutura de telecomunicação móvel, o conceito de MEC (Mobile Edge Computing ou Multi-access Edge Computing) adquiriu sua definição atual em 2014 quando o grupo ETSI (European Telecommunication Standards Institute, instituto europeu cujo objetivo é definir padrões que permitam ao mercado europeu funcionar como um todo ao nível das telecomunicações), desenvolveu um grupo de especificações padrões para Mobile Edge Computing. Seu objetivo central é prover serviços de TI e de computação em nuvem na borda da rede. Além disso, o padrão também prevê a distribuição de aplicações multi-vendor entre as diferentes plataformas MEC disponíveis [13][23].

Como os servidores MEC fazem parte da mesma infraestrutura de telecomunicação móvel, há um ganho de performance e usabilidade na utilização de

serviços que necessitam ou dependem de informações de geolocalização e de menores tempos de resposta. Além disso, há também uma redução da carga de informação que é trafegada pelo core da rede. Este fato é extremamente relevante considerando o aumento progressivo observado anualmente no volume de informação que é gerado na borda a partir de dispositivos IoT.

Por utilizarem a infraestrutura local de telecomunicações, o MEC oferece também uma vantagem econômica pois é possível realizar a coleta de dados locais. Estas podem ser utilizadas em modelos de Big Data e aprendizado de máquina para que possam ser inferidas estatísticas e informações da população daquela localização. Estes dados podem então ser utilizados como auxílio para a tomada de decisões estratégicas e de marketing para comerciantes e empresários [20].

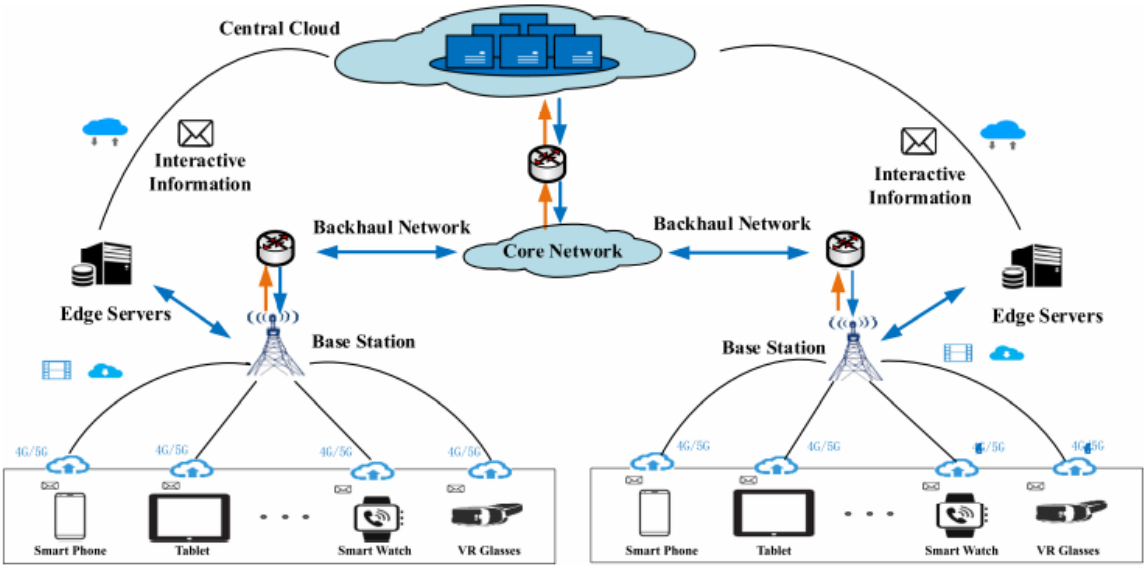

Figura 5: Modelagem do fluxo de informação em uma região com Multi Access Edge Computing. Neste exemplo, os dispositivos IoT enviam dados para as torres de telecomunicação via rede móvel. Estas torres podem utilizar realizar o processamento localmente na infraestrutura MEC acoplada, e esta pode enviar os dados para as centrais de cloud caso necessário [13].

São diversas as sugestões de caso de uso para MEC, sendo que em [24] temos como sugestão o seu uso para análise de dados provenientes de sistemas de vigilância, uma vez que grandes quantidades de dados provenientes de câmeras e sensores de segurança seriam analisados localmente. Outra proposta seria sua aplicação na comunicação entre veículos autônomos e infraestrutura de tráfego,

utilizando-se da rede móvel de telecomunicação para envio de alertas com baixa latência entre os veículos de uma mesma área.

3.2.2.1. Arquitetura Mobile Edge Computing

O MEC utiliza-se de uma série de outras tecnologias, como a virtualização e a computação em nuvem, para executar o seu modelo de provisionamento, gerenciamento e escalabilidade de ambientes computacionais na borda da rede [20].

Neste contexto, a nuvem disponibilizada dentro de um MEC é entendida como uma plataforma capaz de fornecer três tipos de serviços: [28] [29] [30]

- Infraestrutura, tanto virtualizada quanto física, com capacidade de escalabilidade de seus recursos. Estes serviços oferecem a possibilidade de configuração e preparação do ambiente provisionado pelo próprio usuário, podendo ele ser customizado para diferentes usos.
- Plataforma, oferecendo ambientes pré-configurados e especializados para serem utilizados no desenvolvimento e execução de aplicações.
- Software, disponibilizando o acesso via internet ou rede local a um software que está hospedado no provedor de nuvem.

Nos últimos anos, a ETSI tem realizado esforços no estabelecimento de um modelo de implementação da plataforma MEC. Esta seção o descreve resumidamente, apresentando os dois módulos que o compõem: o framework e a arquitetura MEC.

O Framework MEC estabelece as aplicações utilizadas nesta plataforma como entidades de software que são executadas sobre uma infraestrutura virtualizada ou conteinerizada localizada na borda da rede. Estas entidades são agrupadas em três níveis: [21]

- MEC Host Level: é a camada fundamental do framework, nela ocorre a orquestração e provisionamento dos ambientes virtuais responsáveis pela execução de aplicações MEC.
- Networks: realiza a conexão entre a plataforma MEC, o core da rede e as redes locais que utilizam esta infraestrutura.
- MEC System Level: Camada de abstração entre o MEC Host Level e os usuários das aplicações ali hospedadas. Consiste de APIs (Interface de

Programação de Aplicações, em português) utilizadas para requisição e transferência de informações entre os dispositivos móveis e a plataforma.

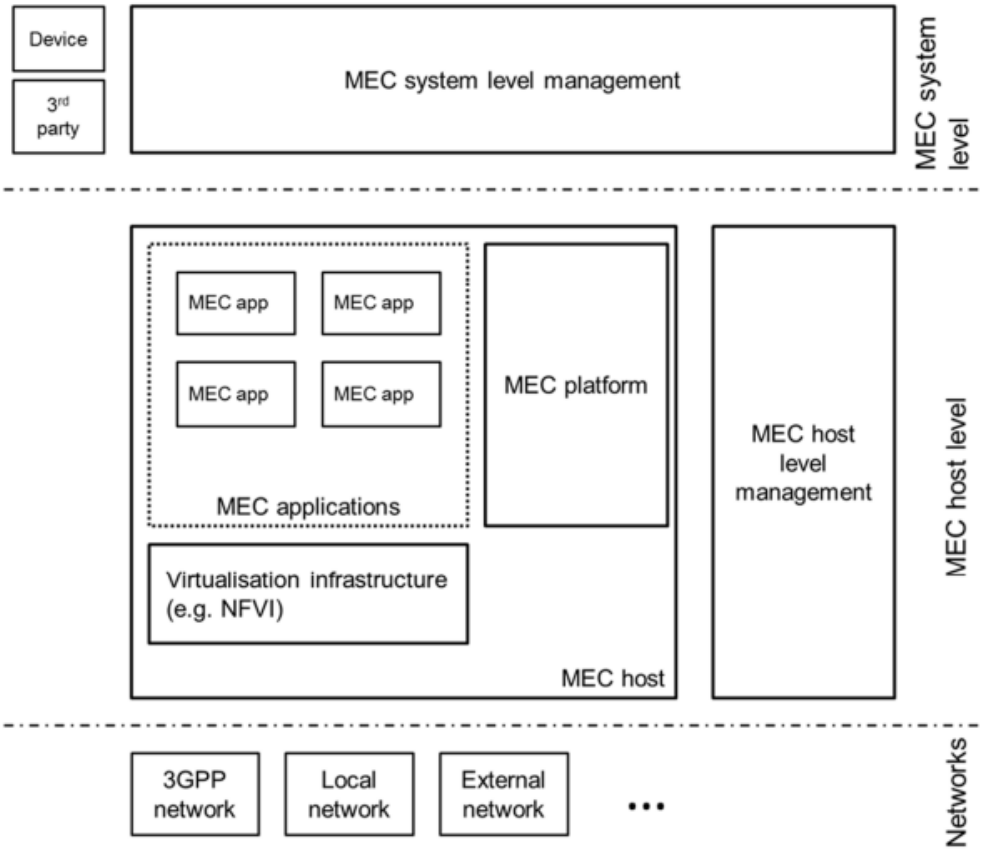

Figura 6: Framework de arquitetura MEC, como definido pela ETSI [21].

O guia de referência para arquitetura MEC concentra-se em detalhar as especificidades das camadas MEC Host Level e MEC System Level do Framework, uma vez que estas são as que compõem o núcleo funcional da plataforma.

Como mencionado no Framework MEC, o Host Level (mais especificamente, no MEC Host) é o responsável pelo provisionamento e orquestração dos ambientes virtuais que contam com storage, capacidade computacional, recursos de rede e as funcionalidades fundamentais para a execução de aplicações em um ambiente MEC. Estas funcionalidades permitem que as aplicações possam realizar ou que sejam descobertos por outros serviços que se utilizam de infraestrutura MEC por meio da interface Mp3, apresentada na figura 7, indicando a comunicação entre diferentes Hosts e plataformas MEC. Para isso, são configurados DNS (Domain

Name Services) para facilitar a chegada de tráfego para a aplicação MEC desejada e permitir a comunicação entre plataformas [20] [22].

O gerenciamento do Host Level é realizado pelo MEC Plataform Manager e pelo Virtualization Infrastructure Manager (VIM, em abreviação). O primeiro é responsável principalmente por gerenciar o ciclo de vida das aplicações virtualizadas no host e resolução de tráfego de rede com a rede externa por meio de verificação de regras de roteamento, autorização de acesso, configuração de DNS, etc. É neste componente onde também são centralizados os relatórios de falhas e de performance dos ambientes virtualizados para posterior análise. Já o VIM realiza a gestão dos ambientes virtualizados dentro de um MEC. Ele é encarregado de configurar e provisionar a infraestrutura virtual com uma imagem da aplicação já carregada para rápida utilização do ambiente, assim como também a sua desativação quando este já não se faz mais necessário [20] [21].

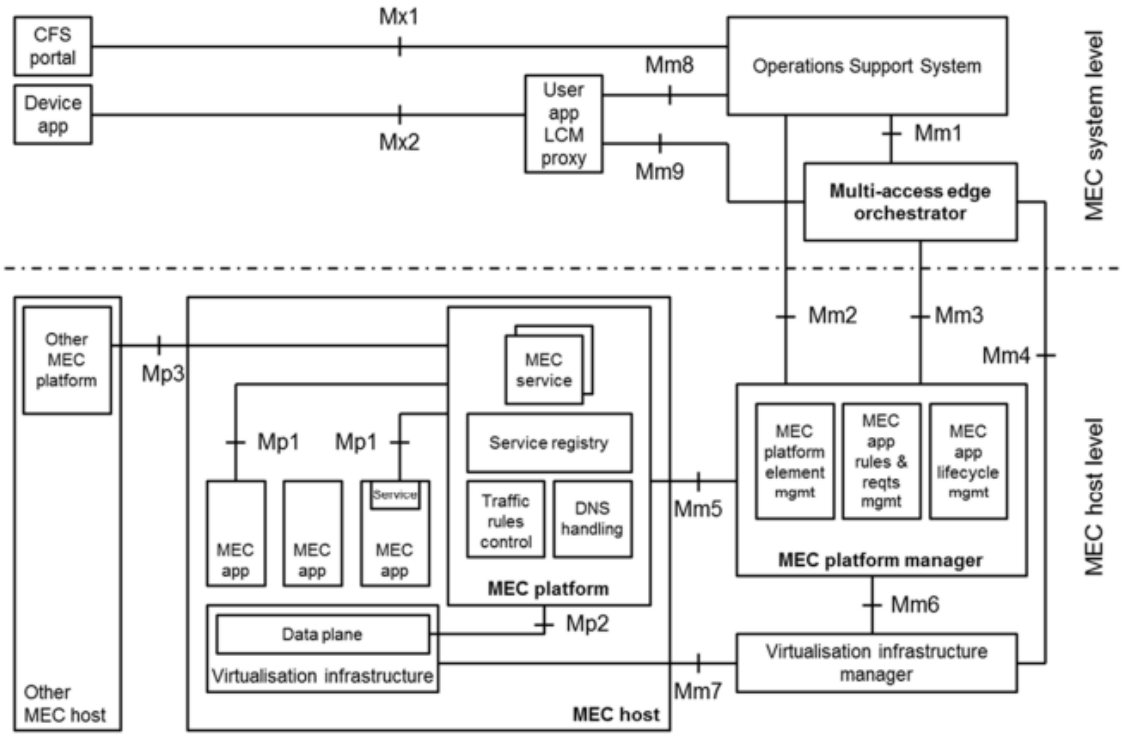

Figura 7: Arquitetura de referência para MEC, detalhando as camadas MEC System Level e MEC Host Level descritas no Framework. Na imagem, são configurados três grupos de interface entre entidades que estabelecem o vínculo entre elas. São estes: Mp para conexões relacionadas a funcionalidade da plataforma; Mm para gerenciamento; Mx para conexão com entidades externas à plataforma, como os usuários finais [21].

Na System Level, a camada responsável pela abstração na comunicação entre os recursos contidos no Host Level e o usuário final, o Multi-Access Edge Orchestrator é a entidade central para o seu funcionamento. Ela possui uma visão geral dos recursos e da capacidade de toda a infraestrutura de rede disponibilizada na borda, incluindo um catálogo completo de todas as aplicações que estão disponíveis para uso. É o Orchestrator o responsável por capturar as condições de gatilho que aciona os mecanismos de provisionamento ou desligamento de aplicações, assim como a sua realocação baseado em necessidades de recursos computacionais ou menor tempo de latência de rede. Essas condições ou requisições são informadas pela associação das entidades Operations Support System (OSS) e Customer Facing Service (CFS). O CFS oferece a capacidade de interação com os serviços MEC, via portal ou por aplicações móveis. Suas requisições são passadas ao OSS, que é responsável por receber todos os pedidos relacionados a aplicações MEC e repassá-las para o Multi-Access Edge Orchestrator [20][21][24].

3.2.3 Fog Computing

Definido pela primeira vez em 2012 pela CISCO como “uma plataforma altamente virtualizada provida de capacidade computacional, storage e serviços de rede entre os dispositivos de usuários finais, não estando estes presentes exclusivamente na borda, e a infraestrutura de Cloud Computing” [74], é possível perceber certo grau de semelhança entre as definições entre Fog Computing e Edge Computing.

Desde então, outras definições foram realizadas. Em [13], Fog Computing é definida como uma camada intermediária entre os chamados dispositivos terminais, responsáveis por receberem a entrada de dados, localizados na borda da rede e a camada de Cloud Computing. Esta camada seria responsável por fornecer os mesmos serviços definidos já em 2021 por [74]. Porém é acrescentado que esta camada intermediária, denominada de Camada Fog, é constituída por uma ampla gama de dispositivos computacionais que vão desde unidades dedicadas como roteadores e servidores de pequeno porte a unidades computacionais temporárias

como smartphones. Esta definição, que enfatiza o processamento descentralizado, é compartilhada por outros artigos como [75], [1] e [76].

Porém, tal característica tida como determinante para diferenciar a Fog Computing está presente também nas arquiteturas de Edge Computing já listadas nas seções anteriores. Tanto os Cloudlets do Mobile Cloud Computing quanto às unidades processuais presentes na Mobile Edge Computing são capazes de realizar o processamento distribuído na camada intermediária entre os dispositivos terminais e a Cloud. Desta forma, não foi considerado que há diferenças suficientes para que estes dois termos sejam referidos a tecnologias diferentes e foi preferível realizar o estudo direcionado apenas de Edge Computing.

3.3. Bancos de Dados

Em computação, um banco de dados é um software que possui a responsabilidade de realizar o armazenamento persistente de informações em forma de registros. A informação armazenada em um banco de dados representa sempre algum aspecto do mundo real, sendo ele definido como Universo de Discurso. Desta forma, estas informações precisam estar disponíveis para consulta e as alterações ocorridas no Universo de Discurso precisam ser refletidas no banco de dados. Estas operações são realizadas a partir da manipulação dos dados armazenados através de comandos ou scripts que executam a seleção, inserção, atualização e exclusão.

O acesso às informações armazenadas em um banco de dados é realizado por um Sistema Gerenciador de Banco de Dados (SGBD), que é quem faz o gerenciamento das requisições de acesso e garante que as alterações das informações armazenadas sejam realizadas de forma consistente, a partir da interpretação dos comandos computacionais enviados por aplicações. Estes comandos precisam estar escritos em uma linguagem computacional do qual o SGBD seja capaz de compreender [26] [27] [39].

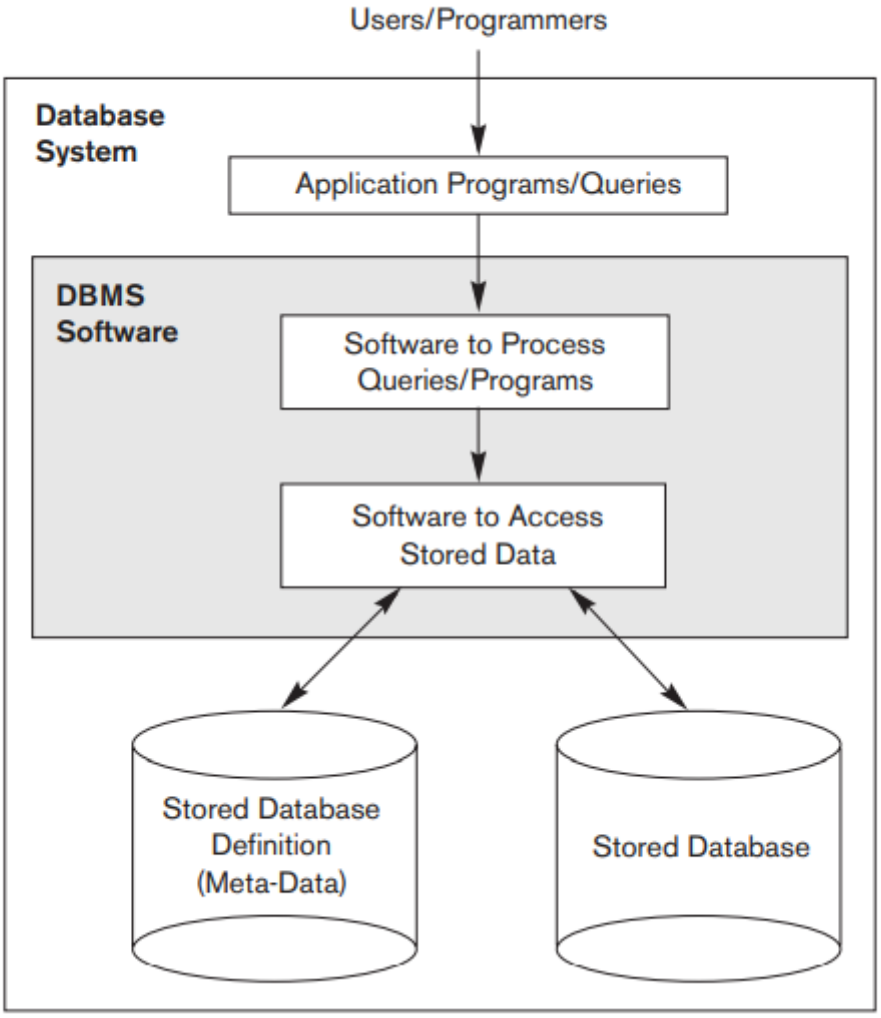

Figura 8: Visualização simplificada do fluxo de informações em um sistema de banco de dados [26].

Tradicionalmente, o modelo relacional é o mais comumente associado aos SGBDs. Nele, as informações são visualizadas no formato de linhas em uma tabela. Cada tabela pode representar os atributos de uma entidade ou o relacionamento entre duas entidades. As informações entre tabelas podem ser relacionadas através de mecanismos de chave primária e estrangeira, permitindo a realização de consultas complexas que abrangem múltiplas tabelas. Uma chave primária (em inglês, primary key ou PK) é um atributo (ou conjunto de atributos) que identificam de forma única uma instância em uma entidade, não podendo receber o valor nulo. Já a chave estrangeira (em inglês, foreign key ou FK) é um atributo (ou conjunto de

atributos) cujos valores correspondem a chave primária em uma outra tabela [55] [56] [57].

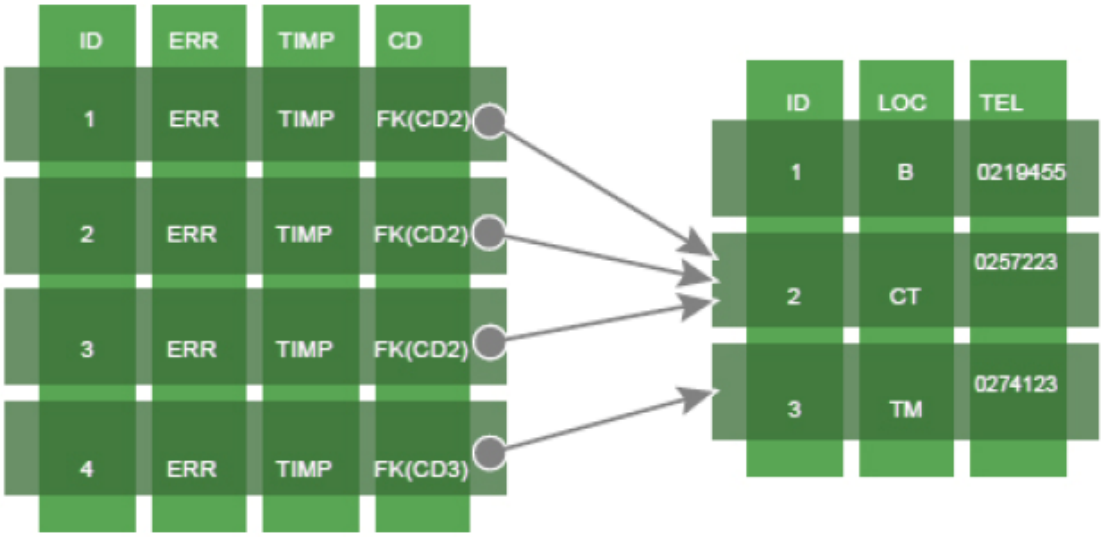

Figura 9: Representação de um Modelo Relacional contendo uma tabela de logs de erro e uma tabela com informações de datacenters [52].

A figura 9 apresenta um exemplo de modelo relacional onde estão sendo armazenadas logs de erro. Na Tabela de Erros (à esquerda), cada registro contém um identificador único (ID) e é composto por um código de erro (ERR), um timestamp (TIMP) e o datacenter (CD) onde o erro ocorreu. Para que as informações de localização e telefone dos datacenters não fossem repetidas na Tabela de Erros, cada registro aponta para uma linha da Tabela de Datacenters (à direita), que contém a localização e o telefone de cada um [52], onde o campo CD na Tabela de Erros possui o valor do campo ID na Tabela de DataCenters. Sendo assim, o campo CD é uma chave estrangeira na Tabela de Erros.

Uma vantagem deste tipo de modelagem é a baixa quantidade de dados duplicados presentes no banco de dados mas, como existe uma grande dependência entre as entidades, a mudança em um registro pode acarretar no bloqueio de múltiplas tabelas até que se garanta a consistência dos dados.

Porém, com o crescente desenvolvimento de aplicações IoT gerando grandes massas de dados a todo momento, juntamente com o maior interesse em se analisar dados históricos para previsões de comportamentos em diferentes áreas, a

necessidade de se buscar SGDBs capazes de lidar com dados que, em sua grande maioria, são não estruturados e difíceis de se lidar, se tornou cada vez mais relevante. É por este motivo que os banco de dados não relacionais, comumente chamados de bancos de dados NoSQL (do inglês, Not Only SQL), “Não Apenas SQL” em tradução livre, têm ganhado cada vez mais importância no mundo da computação.

A modelagem NoSQL surgiu com o intuito de proporcionar uma arquitetura flexível que fosse capaz de lidar com grandes quantidades de dados não estruturados (dados cuja estrutura interna não é passível de indexação) ou semi estruturados (dados que possuem chaves para definição de uma hierarquia, apesar de não estarem em conformidade com a estrutura de um banco relacional), proporcionando alta escalabilidade e rápido desempenho.

No modelo não relacional, as informações não são armazenadas no formato de linhas em uma tabela como no modelo relacional e nem são realizadas operações de join entre as entidades, mas sim, são armazenadas de forma otimizada para o tipo de dado que está sendo trabalhado, além de utilizarem outras linguagens além de SQL para realização de queries. Tais bancos permitem a replicação de maneira nativa, diminuindo o tempo de recuperação de dados, além de serem facilmente horizontalmente escalados, apenas aumentando-se o número de máquinas disponíveis no sistema, o que em um banco relacional é inviável por poder causar concorrência. Porém, por conta de toda sua flexibilidade, os bancos NoSQL muitas vezes não conseguem manter a consistência dos dados [58] [61] [63].

Para que se consiga atingir flexibilidade no armazenamento de diferentes tipos de dados, os bancos NoSQL possuem alguns diferentes modelos de armazenamento, que se adequam ao tipo de dado que se irá armazenar. Os modelos mais utilizados são: [60] [63] [65] [66] [67]

- Chave-valor (Key-value Store Databases): Cada valor é associado a uma chave única, o que resulta em uma boa performance de velocidade do SGBD por proporcionar a busca por chave, se assemelhando a uma hash table. Podemos citar como exemplo o banco Redis ou o Amazon DynamoDB.

CustomerID	Column Family: Identity
001	**First name:** Mu Bae **Last name:** Min
002	**First name:** Francisco **Last name:** Vila Nova **Suffix:** Jr.
003	**First name:** Lena **Last name:** Adamcyz **Title:** Dr.

CustomerID	Column Family: Contact Info
001	**Phone number:** 555-0100 **Email:** someone@example.com
002	**Email:** vilanova@contoso.com
003	**Phone number:** 555-0120

Figura 10: Key-value Store Databases [63].

- Orientado a Colunas (Column-Oriented Databases): Conceitualmente se assemelhando ao modelo de dados relacional, este tipo utiliza linha e coluna como chave. Basicamente, os dados de uma coluna são indexados por um conjunto composto por linha, coluna e um timestamp, sendo o timestamp responsável pela identificação de cada uma de maneira única. Dessa forma, proporciona boa eficiência nas execuções de leitura e escrita de dados. Exemplos de bancos orientados a colunas são Cassandra ou Big Table.

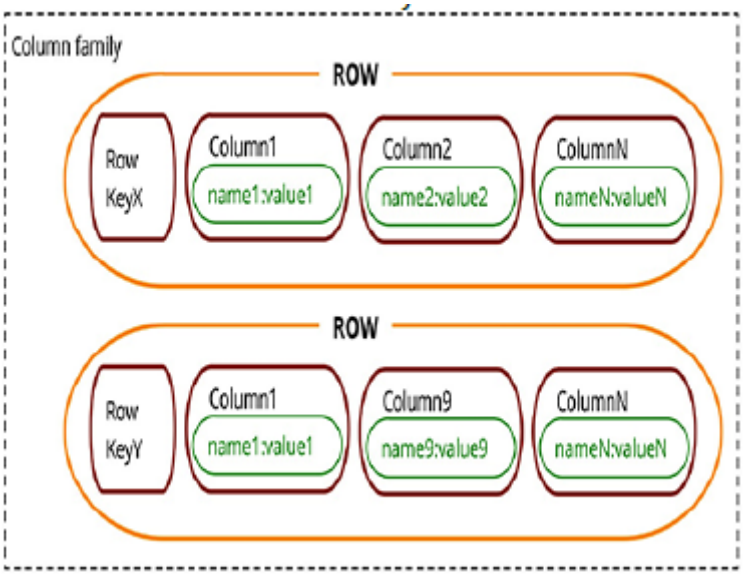

Figura 11: Column-Oriented Databases [67].

- Orientado a Documentos (Document Store Databases): Neste modelo são armazenadas coleções de documentos, que são compostos por campos de strings e valores, bem como outros documentos aninhados. Se assemelha

com o modelo de chave-valor, pois cada documento pode ser considerado como um conjunto de pares chave-valor. Normalmente tais documentos possuem o formato JSON ou XML. Na literatura os exemplos mais citados são os bancos MongoDB e o CouchDB.

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Figura 12: Document Store Databases [63].

- Grafos (Graph Databases): Neste tipo de banco os dados são armazenados em forma de grafos, formando uma coleção de nós e arestas. Os nós representam as entidades, enquanto que as arestas especificam as relações entre essas entidades. Também possuem propriedades, que são as informações de cada relação. Este modelo permite que sejam realizadas consultas que atravessam toda a rede de nós e arestas de forma a se analisar os relacionamentos de maneira eficiente. Como exemplo temos o Neo4j ou o GraphDB.

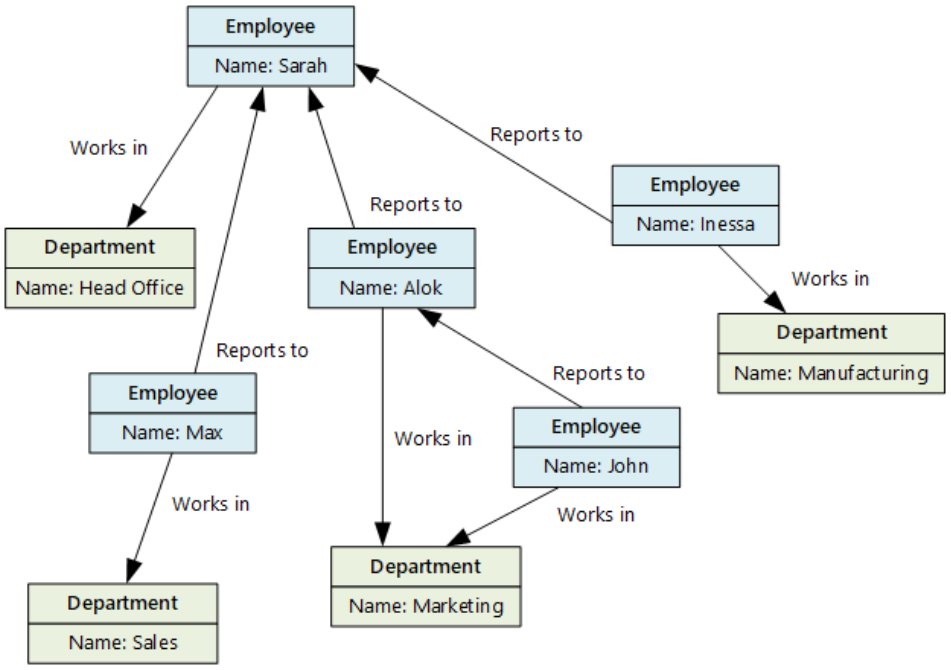

Figura 13: Graph Databases [63].

- Séries Temporais (Time Series Databases): Séries Temporais é um conjunto de valores ordenados por tempo, sendo este modelo otimizado para este tipo de dado com timestamp, como por exemplo, eventos de monitoramento, medições, dados de sensores, dentre outros. Neste tipo de banco de dados, as informações raramente sofrem atualização ou são deletadas. Mais à frente neste trabalho entraremos em mais detalhes sobre séries temporais. Podemos citar o Apache IoTDB ou o BTrDB como exemplos de banco de séries temporais.

timestamp	deviceid	value
2017-01-05T08:00:00.123	1	90.0
2017-01-05T08:00:01.225	2	75.0
2017-01-05T08:01:01.525	2	78.0

Figura 14: Time Series Databases [63].

3.3.1. Desafios na Utilização de Banco de Dados Distribuídos em Ambientes na Borda

Um banco de dados distribuído (BDD) pode ser definido como uma coleção de múltiplos bancos de dados logicamente inter relacionados distribuídos por uma rede de computadores, ou seja, apesar de os dados logicamente pertencerem ao mesmo sistema, estes podem estar distribuídos por diversas máquinas localizadas no mesmo local ou por uma rede de computadores interconectados que estejam na mesma localidade. Estes múltiplos computadores interconectados podem ser chamados de nós. Os sistemas de banco de dados distribuído (SBDD) podem ser homogêneos, quando todos os nós rodam o mesmo SGBD, ou heterogêneos, quando cada nó pode estar rodando um diferente SGBD (que pode ser relacional ou não relacional) [26] [27] [39] [57].

Para um banco relacional, a escalabilidade acontece de forma vertical, sendo necessário que se realize um upgrade do hardware da máquina para se obter um melhor desempenho, desta forma, sempre haverá um limite ao nível de escalabilidade que se pode atingir pois existe dependência com o hardware, o que gera altos custos e aumento na dificuldade de manutenção. Este tipo de banco também não costuma ser capaz de lidar bem com grandes volumes de dados, além de ser limitado em relação à variedade dos tipos de dados aceitos, também possuindo limitações para realização de operações de agregações, uma vez que costumam ser realizadas queries multi-tabela, que não são eficientes quando se trata de um grande volume de dados [31] [32].

Como na borda há geração de grandes quantidades de dados a todo instante, sendo que estes muitas vezes são amostras sequenciais de dados que são registrados em um intervalo definido de tempo, provenientes de aplicações IoT. Neste cenário, um BDD relacional não seria adequado e por este motivo, para se obter melhores resultados de performance, bancos não relacionais se tornam muito relevantes.

Os NoSQL possuem escalabilidade horizontal, conseguindo atingir altos níveis de escalabilidade apenas aumentando a quantidade de nós do sistema distribuído. Desta forma, não se é tão dependente de hardware como acontece com

o relacional, pois máquinas mais simples e baratas podem ser combinadas, além de em alguns cenários ser possível se utilizar até mesmo de máquinas virtuais, reduzindo ainda mais o custo de escalabilidade. Por conta da facilidade de escalabilidade, com este tipo de banco é possível realizar a técnica conhecida como Sharding, que consiste em dividir os dados entre diferentes máquinas, sendo que este processo ajuda a reduzir a carga em cada nó [61] [59].

Outra vantagem no uso de bancos não relacionais em ambiente na borda é a sua capacidade de lidar com uma grande variedade e quantidade de dados, além de possuir bons frameworks otimizados para se trabalhar com agregações de grandes massas, característica muito relevante quando se trata de informações que em algum momento serão utilizadas em análises e predições com base nos dados históricos ali armazenados [31] [32].

3.3.2. O Teorema CAP

Em bancos de dados relacionais, comumente temos associado o padrão ACID (Atomicidade, Consistência, Isolamento e Durabilidade) que tem como propósito manter a consistência dos dados armazenados. Porém, quando falamos em bancos NoSQL, por serem mais flexíveis, não são capazes de garantir as propriedades ACID, uma vez que para atingir um rápido desempenho e proporcionar escalabilidade horizontal, a consistência de seus dados não deve ser um ponto de obrigatoriedade. Por este motivo, os bancos NoSQL são então associados ao Teorema CAP.

O Teorema CAP (Consistency, Availability, Partition Tolerance) [69], que também é conhecido como Teorema Brewer, foi proposto pelo professor Eric Brewer no ano 2000 e pouco tempo depois, no ano de 2002, Seth Gilbert e Nancy Lynch o comprovaram. Basicamente, mostrou-se que em um sistema computacional distribuído real apenas duas das três condições do teorema podem ser satisfeitas simultaneamente [70].

Os três pontos do teorema são [26] [64] [70] [34]:

- Consistência (Consistency): A consistência é a propriedade que garante que após a escrita ou alteração de um registro no banco de dados, este estará

disponível imediatamente e que todos os usuários terão exatamente a mesma visão mais recente daquela informação. Para isso, é necessário que todos os nós do cluster do sistema sejam sincronizados, o que muitas vezes acarreta em perda de desempenho.

- Disponibilidade (Availability): Diz respeito ao período de tempo que um sistema se mantém operacional, ou seja, qualquer solicitação de dados realizada deve obter uma resposta, mesmo que um ou mais nós estejam desativados.
- Tolerância ao Particionamento (Partition Tolerance): Tolerância ao particionamento diz respeito à capacidade do sistema continuar em funcionamento caso ocorra algum tipo de interrupção, mesmo que parcial, de algum componente ou nó do sistema, impossibilitando a comunicação entre os nós de clusters distintos.

Figura 15: Representação Teorema CAP [34].

Dado que apenas duas das três propriedades citadas acima podem ser garantidas, podemos ter três diferentes tipos de sistemas [69] [62]:

- Sistemas CA (Consistência e Disponibilidade): São sistemas que possuem forte consistência e alta disponibilidade, não sendo capazes de lidar com falhas de particionamento, o que provoca o risco de todo o sistema ficar indisponível, pois a alta disponibilidade é referente a cada nó e não ao cluster como um todo.

- Sistemas CP (Consistência e Tolerância ao Particionamento): Nestes sistemas, abre-se mão da disponibilidade para se focar em forte consistência e tolerância ao particionamento. Neste caso, pode ocorrer a rejeição da escrita de um dado caso haja particionamento mas o sistema não seja capaz de sincronizar por completo naquele momento.
- Sistemas AP (Disponibilidade e Tolerância ao Particionamento): Por fim, neste caso os sistemas focam em possuir alta disponibilidade e tolerância ao particionamento, mesmo que para isso não se atinja forte consistência. Este cenário ocorre em sistemas que não podem ficar fora do ar, em sua grande maioria por serem críticos. Neste caso, costuma-se aceitar todas as novas escritas de dados e a sincronização para obtenção de consistência é realizada em um momento posterior, sendo possível existir um período de inconsistência de dados.

Grande parte dos bancos de dados de séries temporais se encaixam na categoria de sistemas CP, pois a consistência de dados para aplicações IoT é de extrema importância, já que dados dessincronizados podem gerar alertas incorretos ou análises e predições imprecisas. Além disso, o sistema é capaz de se manter operante em caso de partição, sendo capaz de responder às requisições mesmo que demore um pouco mais para isso.

3.3.3. Banco de Dados para Aplicações IoT

Bancos de dados relacionais são capazes de lidar com grandes volumes de dados, porém apresentam grande dificuldade quando se trata de escalabilidade o que é uma grande preocupação quando para aplicações e plataformas de IoT [77] [60], além do tempo de inserção, tempo de execução de funções de agregação e o tempo de consulta em dados IoT. Bancos NoSQL se tornam populares para essas aplicações pois apresentam coleções sem schema horizontal, característica muito útil para dados provenientes de diferentes fontes com diferentes estruturas [78].

Bancos de dados relacionais como PostgreSQL e MySQL já apresentaram bons resultados quando se trabalha com pequenas quantidades de registros de IoT, porém para volumes grandes de informação, os bancos NoSQL, como por exemplo

o MongoDB atingem melhores níveis de performance, conforme apresentado em [78]. O uso de bancos de dados NoSQL para dados de IoT visa suprir diversas necessidades características deste tipo de dado, como por exemplo, questões de segurança e privacidade, tratamento de dados espaciais (geolocalização), interoperabilidade semântica, heterogeneidade e agregação de dados.

Os autores em [60] afirmam que os bancos de dados não relacionais Couchbase, MongoDB e Neo4j são excelentes opções para se tratar de dados de aplicações IoT. O banco Couchbase se adequa a cenários onde o processamento de dados espaciais em tempo real e facilidade em escalabilidade são a prioridade. Os bancos MongoDB e Neo4j são muito bons para tratamento de dados de geolocalização, porém o MongoDB possui a vantagem de possuir melhor performance ao realizar funções de agregação. Outros bancos NoSQL como Cassandra e Redis também podem ser citados como exemplos de bancos de dados que possuem um bom desempenho com dados de sistemas IoT.

3.3.4. Banco de Dados de Séries Temporais

Uma série temporal é uma coleção de amostras de uma variável realizadas sequencialmente em um intervalo definido de tempo. Todo o dado coletado por uma série temporal possui um valor de tempo associado a ela, este tipo de dado é definido como um timestamped data. Exemplos para estes são: informações coletadas por sensores e logs de eventos ou execução. Dados deste tipo são comumente gerados em grande volume por dispositivos IoT localizados na borda da rede [40] [41] [42].

Entre as formas de utilização da informação gerada, pode-se citar:

- Análise de Série Temporal, onde é realizado o estudo e monitoramento da variação de uma variável ao longo do tempo.
- Análise de Regressão, onde é verificado como as mudanças em uma variável podem afetar outras variáveis ao longo do tempo.
- Predição, utilizando-se das informações históricas e seus padrões para prever eventos futuros [40].

Dada a grande volumetria de dados desta natureza que é produzida na borda da rede por estes dispositivos, julga-se necessário a utilização de bancos de dados que apresentam boa compatibilidade com as características específicas de um fluxo de dados de série temporal. São estas:

- Alta performance em operações de escrita, uma vez que diversos dispositivos irão requisitar a escrita da informação por eles coletados em intervalos definidos de tempo e estas operações podem apresentar picos de volume a ser escrito.
- Compressão de dados e escalabilidade, dada a natureza contínua da inserção de novos registros. O SGBD necessita de funcionalidades que realizem a compactação de dados de forma eficiente e também apresentar possibilidade de escalar horizontalmente sua capacidade, oferecendo múltiplos nós de acesso para operações tanto de escrita quanto de leitura.
- Possibilidade de utilização de funções de agrupamento para a consulta de dados armazenados. Exemplos deste tipo de função são o cálculo de média, somatório e contagem de valores que são encontradas em SGBDs relacionais [40] [42] [47].

A grande volumetria de dados a serem processados continuamente, a performance necessária para estas operações serem realizadas e a necessidade de escalabilidade, são requisitos particularmente desafiadores de serem aplicados a bancos de dados relacionais, embora existam aplicações que os utilizam para armazenamento de dados de série temporal, como o VividCortex ([44],[45]) e o Druid ([43],[46]). Devido a estas características, os bancos de dados NoSQL, em especial os denominados Time Series Databases (bancos de dados de série temporal), são preferíveis para a realização desta tarefa. Estes tipos de banco de dados são altamente especializados para o atendimento às características citadas anteriormente, oferecendo ainda, em alguns casos, soluções de alta disponibilidade e de tratamento da granularidade das informações que são armazenadas, possibilitando que informações mais antigas possam ser agregadas em médias, por exemplo, ou por outras funções agregadoras, fornecendo uma utilização de recursos mais eficiente.

Este tipo de funcionalidade é especialmente útil no contexto da computação de borda, uma vez que os recursos computacionais para a realização tanto de processamento quanto de armazenamento são limitados. Neste contexto, o SGBD utilizado necessita que de uma arquitetura leve quando aplicado em dispositivos IoT, além de possibilitar a replicação dos dados de múltiplos nós dispostos na borda da rede para clusters primários localizados em um datacenter privado ou na nuvem. Dois exemplos de bancos de dados de série temporal otimizados para aplicações IoT são o Apache IoTDB e o BTrDB (Berkeley Tree Database).

3.3.4.1. Apache IoTDB

Iniciado em 2016 na Universidade de Tsinghua, na China, o IoTDB realiza o armazenamento das informações em formato de coluna com o intuito de facilitar o gerenciamento de grandes volumes de dados de série temporal. Contando com suporte a uma linguagem SQL-Like e a biblioteca JDBC para conexão com o banco de dados, o IoTDB se destaca por sua facilidade de implementação e utilização [48] [47].

Todas as requisições realizadas ao IoTDB passam pelo QueryEngine para a tradução dos comandos SQL e requisições via API enviadas, construção e otimização de planos de execução para o resgate das informações armazenadas, assim como o seu retorno para o cliente. O plano de execução das requisições é então executado pelo QueryExecutor, que realiza o controle de sessões em execução [53].

A escrita e leitura das informações é coordenada pelo StorageEngine. Toda requisição de escrita é primeiramente armazenada em memória para posteriormente ser escrita em disco, em um processo que se assimila aos mecanismos de escrita em log de transações nos bancos de dados relacionais. Tanto os dados coletados por dispositivos quanto os metadados do IoTDB são escritos em arquivos dedicados denominados TsFile que ficam armazenados no File System do servidor [54].

No IoTDB, a modelagem de dados é construída a partir de uma árvore hierárquica. No exemplo de modelagem de uma usina elétrica, exibido na figura 16, temos que no topo da hierarquia reside um nó Root obrigatoriamente. Abaixo desse

nó existem as camadas de Power Group e Power Plant, as últimas estão contidas em um nó da primeira. Nas duas últimas camadas temos os Devices e os Sensors. Os devices podem ser entendidos como qualquer coisa que produza métricas a serem medidas. Os sensores são as métricas em si que são coletadas de um device [50][35].

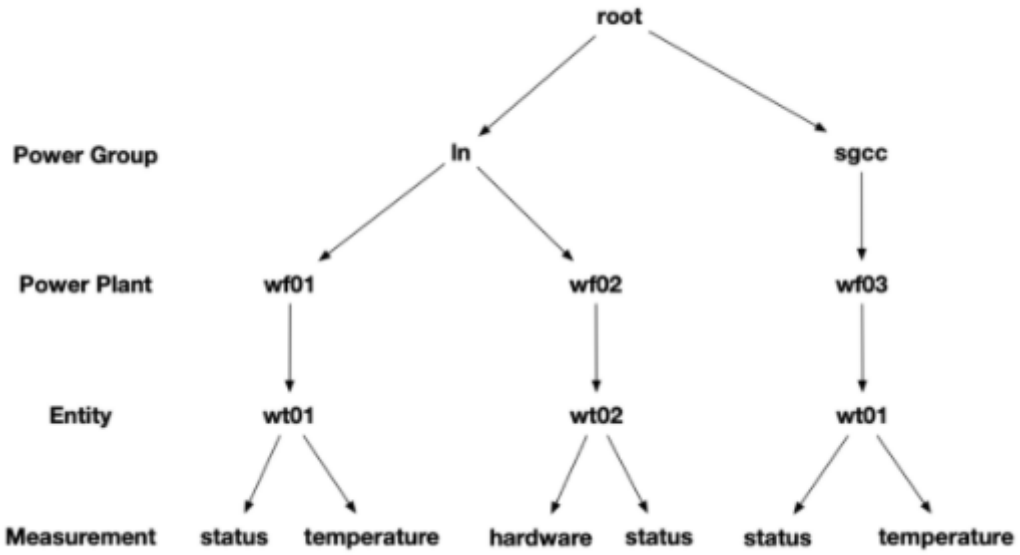

Figura 16: Exemplo de modelagem de dados para o IoTDB [35].

As séries temporais são criadas utilizando-se os nomes de cada nó que esteja presente no caminho que liga o Root até a medida desejada, sendo cada nome separado por um “.”. Um exemplo de criação de uma série temporal para o status da entidade wt02 da figura 16 seria:

create timeseries root.ln.wf02.wt02.status with datatype = BOOLEAN.

Atualmente, os tipos de dados suportados pelo IoTDB para a criação de séries temporais são:

- Boolean
- INT32
- INT64
- FLOAT
- DOUBLE

- TEXT

Além disto, há suporte também para diferentes configurações do timestamp associado a cada dado que é inserido na série temporal [36].

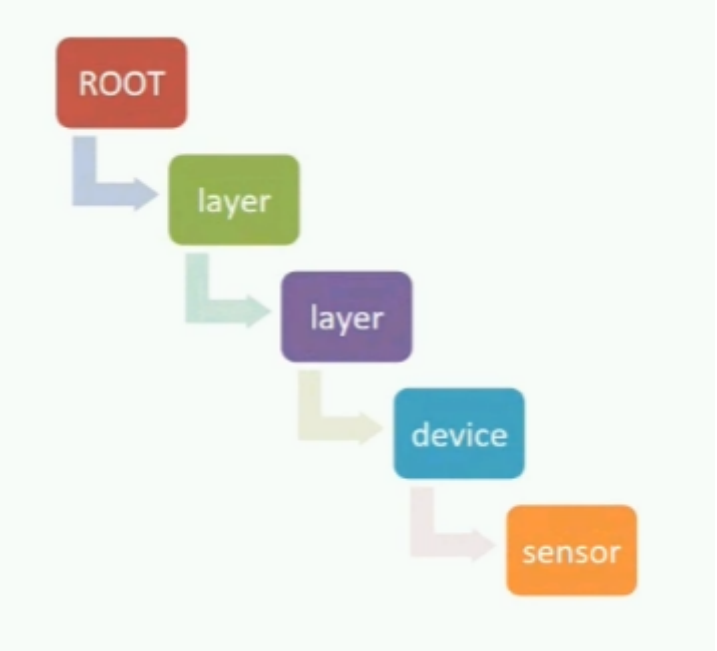

Figura 17: Hierarquia utilizada na modelagem de dados para o IoTDB, onde podem haver N camadas “Layer” entre o Root e o Device, dependendo do escopo da modelagem de dados escolhida [50].

O IoTDB conta com suporte em deploys tanto em datacenters como localmente em dispositivos de borda como um Raspberry PI. Para este último caso, o IoTDB conta com uma versão embarcada com requisitos de hardware mais baixos, sendo necessário no mínimo 32MB de memória para a sua execução e oferecendo suporte a processadores ARM7. É obrigatória também a presença de storage local para evitar a perda de dados em caso de falhas na camada de rede. Nestes casos, os dados são persistidos localmente no formato TsFiles e estes podem ser posteriormente sincronizados com o auxílio do módulo File Sync com uma instância IoTDB que esteja instalada em um servidor centralizado [47].

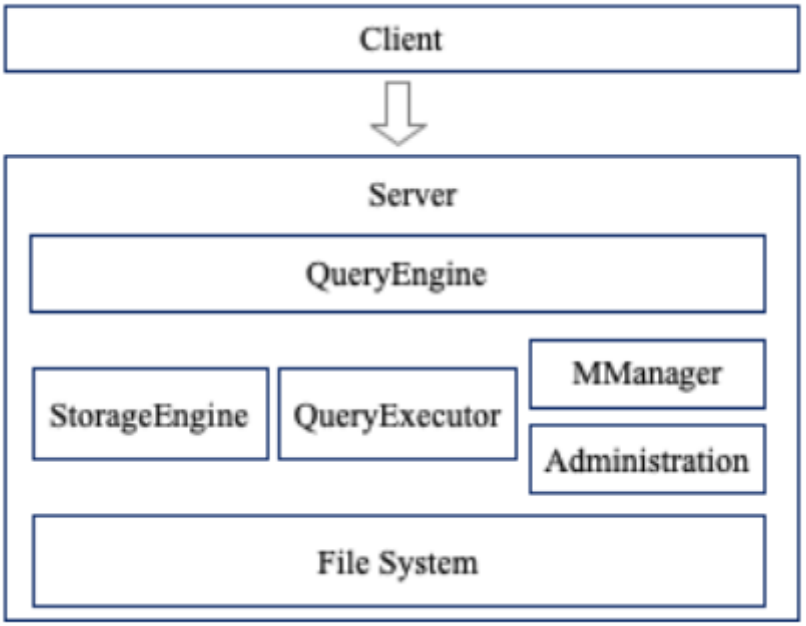

Figura 18: Arquitetura Cliente-Servidor empregada pelo IoTDB, onde o servidor hospeda uma instância de banco de dados. O Query Engine é o responsável por realizar a tradução dos comandos enviados ao IoTDB, acionar os motores necessários para o atendimento da requisição e retornar o resultado obtido para o Cliente [48].

3.3.4.2. BTrDB

Desenvolvido na Universidade da Califórnia e já implementado em plataformas de analytics dedicadas a manipulação de dados de sensores como o Predictive Grid, o BTrDB (Berkeley Tree Database) é um banco de dados de série temporal desenvolvido com foco no armazenamento dos dados coletados por micro synchrophasor, que são sensores de alta precisão para a captura de métricas relacionadas a eletricidade em sistemas de transmissão, como voltagem, corrente e frequência. Como esta nova classe de sensores possui um custo relativamente baixo e a capacidade de coleta de dados é precisa e elevada, o BTrDB foi desenvolvido de modo a suportar taxas de escrita e leitura extremas. Em [49], os autores citam que este produto é capaz de chegar a 53 milhões de registros inseridos por segundo e de ler 119 milhões de valores por segundo em um cluster de quatro nós.

Para chegar a estes valores elevados, foi necessária a construção de uma nova abstração para dados de séries temporais e uma estrutura que as implementasse. Utilizando-se de uma “time-partitioning, multi-resolution,

version-annotated, copy-on-write k-ary tree", o BTrDB armazena os valores recebidos e o seus respectivos timestamps nas folhas da sua estrutura de dados. Cada nó interno da árvore armazena as estatísticas já calculadas dos dados armazenados nos nós externos a ela. Com isto, o BTrDB consegue acelerar a execução de funções de agregação de dados como mínimo, máximo e média [49] [68].

O BTrDB realiza também um versionamento dos dados armazenados, não permitindo que os dados já inseridos sejam alterados. Para que novas versões daqueles dados sejam inseridos (devido a uma recalibração do equipamento, delay de rede, etc) é necessário que um novo bloco de dados seja inserido no banco de dados, com um versionamento diferente. Isto permite que as consultas sejam reproduzíveis independentemente da chegada de novos valores a serem inseridos [71] [72].

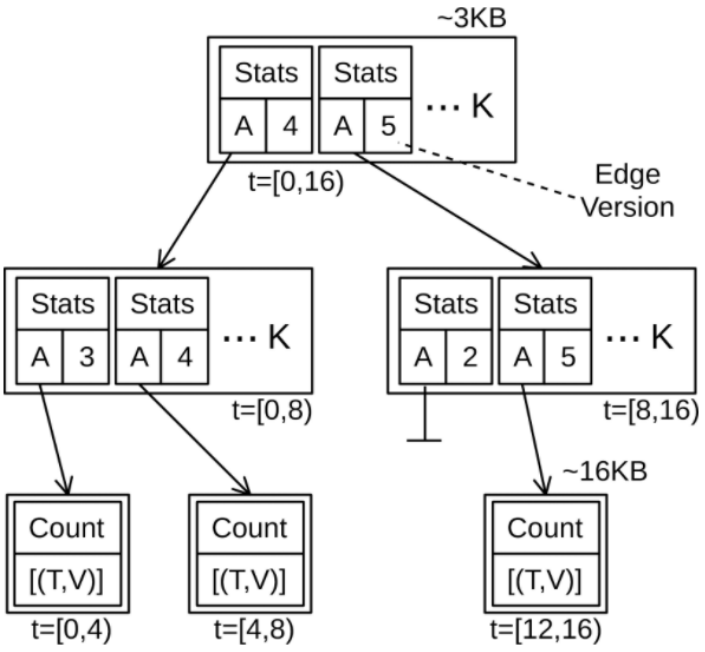

Figura 19: Estrutura da árvore "k-ary" utilizada pelo BTrDB. Nela, as folhas contém os pares de valores que são enviados pelos sensores e cada nó interno da árvore possui já calculados as estatísticas da sub-árvore a ela associada. Desta forma, cada nó interno pode realizar o cálculo das suas estatísticas utilizando como base os valores armazenados em seus nós filhos e o recálculo destes é facilmente realizado ao se percorrer os nós até se chegar à raiz da árvore na ocasião da inserção ou alteração de valores nas folhas [68].

4. SIMULAÇÃO

4.1. Proposta

A proposta de aplicação prática para este trabalho é a realização de um teste comparativo entre um banco de dados relacional tradicional e um banco de dados não relacional de séries temporais.

O banco de dados relacional escolhido foi o Microsoft SQL Server 2019, por ser um SGBD amplamente conhecido comercialmente, além de os integrantes do projeto possuírem familiaridade com seu ecossistema. O banco de dados de séries temporais escolhido para análise comparativa é o já anteriormente citado, Apache IoTDB na versão 0.12.2, por prometer uma fácil implantação e alta velocidade em operações de leitura e escrita para dados complexos [51].

Foi utilizada a ferramenta IoTDB-Benchmark que é capaz de realizar benchmarking entre o IoTDB e outros bancos de dados e bancos de séries temporais. Esta ferramenta, apresentada em [33], foi especialmente desenvolvida para séries temporais. Com ela, é possível se especificar quais são as métricas que precisam ser coletadas do sistema que está sendo analisado.

O IoTDB-Benchmark possui um gerador de dados que é capaz de simular diversas distribuições de dados, como ondas quadradas, senóides, dentre outras, além de fornecer configurações para uso de operações mais simples, como ingestão de dados e execução de queries. Também fornece opções de combinações de operações complexas para simulações mais próximas a casos reais de cargas de dados complexos.

4.2. Configuração do Ambiente

Para a realização do Benchmark, foi construída uma máquina virtual dedicada para este procedimento com o intuito de haver o mínimo de interferência possível de outros softwares sendo executados no sistema operacional. Foi utilizado o VMWare Workstation Player 16 para a montagem do ambiente virtual. Tanto as suas configurações quanto as do computador hospedeiro estão descritas abaixo.

Sistema Operacional	Windows 10 Education (10.0.19043 Build 19043)
Memória	16 GB
CPU	Intel Core i7-10750H CPU @ 2.60GHz, 2592 Mh, 6 Núcleos, 12 Procesadores Lógicos
Storage	512 GB SSDNvme M.2 PCIe

Tabela 1: Configurações de Hardware do computador utilizado como host da máquina virtual utilizada no Benchmark.

Sistema Operacional	Windows 10 Education (10.0.19043 Build 19043)
Memória	8 GB
CPU	2 CPUs (2.60MHz, 2592Mhz)
Storage	60 GB (SSD Nvme M.2 PCIe)

Tabela 2: Configurações de Hardware da máquina virtual Windows executada no VMWare Player Workstation 16 para a execução dos testes de performance de banco de dados.

Foi utilizado o Windows como sistema operacional devido a maior facilidade apresentada por ele para a instalação e configuração do SQL Server quando comparado ao sistema Linux, onde ele não conta com interfaces gráficas para a configuração do SGBD e das opções de conexão via TCP/IP. Além disso, o IoTDB-Benchmark exige que o usuário utilizado para se conectar ao SQL Server possua a permissão de Bulkadmin, que permite que o usuário realize operações de inserção em massa (bulk operations*)*. Esta permissão não existe na versão para Linux do SQL Server. Abaixo, estão listadas as configurações do SQL Server utilizado:

SGBD	SQL Server 2019 Developer Edition - 15.0.2080.9
Maximum Server Memory	5940 MB
Maximum Degree of Paralelism	4
Cost Threshold for Paralelism	5

Tabela 3: Configurações da instância SQL Server 2019 utilizada no benchmark.

Para a execução de uma instância do IoTDB, é pré-requisito a instalação do JDK (Java Development Kit) e do JRE (Java Runtime Enviroment). A conexão com a instância é realizada por meio da execução de um arquivo .bat. A partir da interface exibida na figura 20, é possível realizar a criação de séries temporais e as operações de inserção e consulta de dados, entre outras.

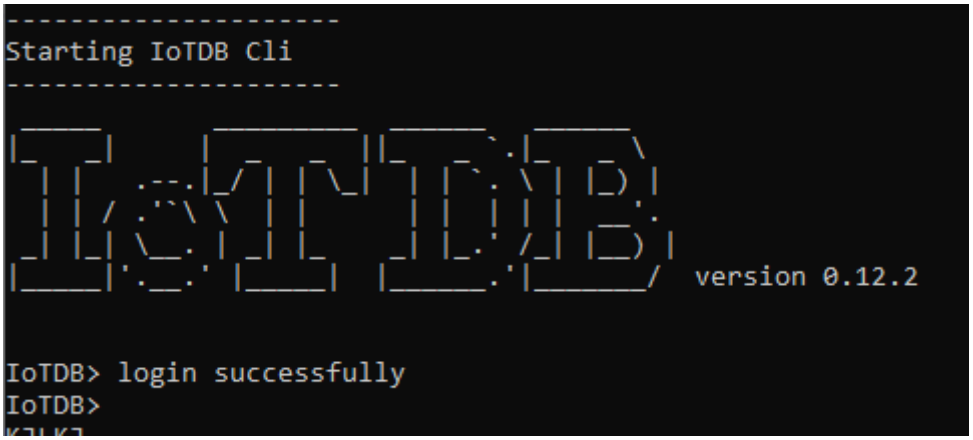

Figura 20: Interface de linha de comando do IoTDB, executado a partir do terminal do Windows. Toda a interação direta com este com uma instância deste SGBD é realizada a partir desta interface mediante a execução de um arquivo .bat, fornecendo o nome do usuário e senha.

Os scripts de execução dos benchmarks de performance do IoTDB-Benchmark estão no formato bash, portanto foi necessário a instalação e configuração de um ambiente Linux dentro do Windows instalando o WSL 2 (Windows Subsystem for Linux), que é um módulo de sistema operacional do Windows 10 que visa a disponibilização de um ambiente Linux compatível com o sistema operacional da Microsoft. A partir dele é possível realizar a execução de programas nativos do Linux dentro do Windows. A distribuição do Linux utilizada foi o Ubuntu 20.04.

4.3. Metodologia

O IoTDB-Benchmark fornece em seu repositório oficial no github [73] diversos cenários pré-definidos de teste de performance para dados de séries temporais, além de oferecer a possibilidade de construção de testes personalizados a partir da alteração dos parâmetros contidos no arquivo de configuração config.properties

deste framework. Este trabalho irá utilizar o cenário de inserção de dados de série temporal.

Segue uma breve descrição da funcionalidade de cada campo alterado no arquivo config.properties para configuração das cargas de trabalho do IoTDB-Benchmark

Parâmetro	Descrição
HOST	IP da Instância de Banco de dados alvo
PORT	Porta de conexão com a instância de banco de dados alvo
DB_SWITCH	Define qual será o SGBD que será conectado,sendo IoTDB-012-SESSION_BY_TABLET para o IoTDB e MSSQLSERVER para SQL Server
BENCHMARK_WORK_MODE	Define o tipo de operação que será realizada no teste (escrita.leitura,etc)
OPERATION_PROPORTION	Define a proporção de operações que são realizadas contra o banco de dados, dentre 10 disponiveis. Exemplos: Operações de seleção com filtro de igualdade, desigualdade, utilização de funções de agregação, etc.
GROUP_NUMBER	Número de storage groups, que são unidades de organização de séries temporais. Uma ou mais séries temporáis estão armazenadas em um storage group.
DEVICE_NUMBER	Número de Devices, que no IoTDB são definidos como dispositivos que possui métricas associadas.
SENSOR_NUMBER	Número de métricas distintas que são associadas a um device.

CLIENT_NUMBER	Número de Clients que se conectam ao banco de dados
BATCH_SIZE_PER_WRITE	Número de métricas que são inseridas no banco de dados por batch. O Número de métricas descritas em uma linha no IoTDB para um device é dada pela fórmula: número de data points = SENSOR_NUMBER * BATCH_SIZE_PER_WRITE
POINT_STEP	Intervalo entre timestamps inseridos.

Tabela 4: Funcionalidade dos campos alterados do arquivo config.properties.

Os valores utilizados para a execução do cenário de inserção são os definidos na seção 6.1 em [73].

```
HOST=127.0.0.1
PORT=6667
DB_SWITCH=IoTDB-012-SESSION_BY_TABLET
BENCHMARK_WORK_MODE=testWithDefaultPath
OPERATION_PROPORTION=1:0:0:0:0:0:0:0:0:0:0
GROUP_NUMBER=20
DEVICE_NUMBER=20
SENSOR_NUMBER=300
CLIENT_NUMBER=20
BATCH_SIZE_PER_WRITE=1
POINT_STEP=5000
```

Serão utilizados como métricas de performance o relatório final emitido pelo IoTDB-Benchmark ao final da execução de seus cenários de teste e os contadores de performance configurados no Performance Monitor do Windows. No Performance Monitor as métricas, coletadas em intervalos de 1 segundo, foram:

- %Processor Time: Porcentagem do tempo total em que o processador está ocupado executando algum processo. Este contador fornece uma visão geral do quão ocupado está o processador em um dado momento.

- %User Time: Porcentagem do tempo em que o processador está no modo usuário. Simplificadamente, este contador exibe a quantidade de tempo em que o processador está executando aplicações de usuário.
- Disk Writes/sec: Taxa de operações de escrita por segundo no disco.

A partir dos dados coletados pelo Performance Monitor durante a execução do teste foram construídos gráficos de linha para uma melhor visualização da variação destas métricas ao longo do tempo.

Para o IoTDB-Benchmark, as métricas de referência para performance são:

- Test Elapsed Time: Tempo de execução no teste, dado em segundos.
- Throughput: Razão entre as operações de leitura ou escrita executadas com sucesso e o Test Elapsed Time.

4.4. Discussão de Resultados

Os resultados obtidos na execução do teste de inserção de dados do IoTDB-Benchmark confirmaram aquilo que havia sido apresentado por este trabalho: os bancos de dados de série temporal são mais eficientes na compressão dos dados armazenados e apresentam maior capacidade de ingestão de dados quando comparados aos bancos de dados relacionais. Estas conclusões foram obtidas ao realizar a análise de 5 execuções de cada cenário de teste, todos com resultados muito parecidos com uma variância de cerca de 5%. Os números apresentados a seguir são referentes a quinta execução.

Os dados obtidos pelo relatório emitido pelo IoTDB-Benchmark, listados na tabela 4, indicam que o IoTDB obteve uma performance 20 vezes maior quando comparado ao SQL Server na realização da carga de trabalho aplicada pelo IoTDB-Benchmark. O armazenamento de dados também foi mais eficiente no IoTDB, armazenando a mesma quantidade de dados ocupando 3,3 vezes menos espaço em disco, conforme indicado na tabela 6.

SGBD	Elapsed Time (s)	Inserções realizadas com sucesso (Ok Points)	Throughput (point/s)
SQL Server	214,40	600.000.000	27.985,11

IoTDB	10,55	600.000.000	568.960,84

Tabela 5: Comparação das métricas obtidas no IoTBenchmark para teste de escrita

SGBD	Espaço Ocupado em Disco (MB)
SQL Server	350,75
IoTDB	104

Tabela 6: Espaço ocupado em disco pelos bancos de dados após processo de inserção de dados.

Os dados obtidos pelo Performance Monitor corroboram com a conclusão apresentada acima sobre a maior capacidade de inserção de dados por parte do IoTDB. Ao se realizar a comparação da informação contida nas tabelas 7 e 8, percebe-se que o IoTDB realizou aproximadamente 18 vezes mais escritas em disco por segundo em comparação com o SQL Server, estando este valor condizente com o número de throughput apresentado pelo IoTDB-Benchmark. Os valores de pico e média de consumo de CPU em modo usuário também foram menores quando comparados ao SQL Server e ambos os SGBDs conseguiram fazer com que a utilização de CPU da máquina virtual chegasse a 100% de uso durante os testes.

Como mencionado em [47], o IoTDB utiliza uma tabela temporária em memória para a realização da ordenação dos dados antes de realizar a escrita em disco. Uma vez que o benchmark foi configurado para que os dados fossem enviados ordenadamente, esta funcionalidade que faz uso intenso do CPU não foi necessária e ela pode ser uma das justificativas para o menor uso médio deste recurso quando comparado a execução no SQL Server. A inserção de dados em batch fornecida pelo IoTDB foi mais eficiente também do que a realizada pelo SQL Server, onde eram observados inserts de linhas individuais sendo realizadas, embora seja importante informar que o SQL Server fornece suporte a inserção massiva de dados a partir de arquivos por meio da operação de bulk insert ou por meio do utilitário BCP (Bulk Copy Program*)*. Esta funcionalidade, porém, não é recomendada para a execução de operações online como as relacionadas a IoT uma vez que o fluxo de informações entre os dispositivos e os receptores é feito constantemente.

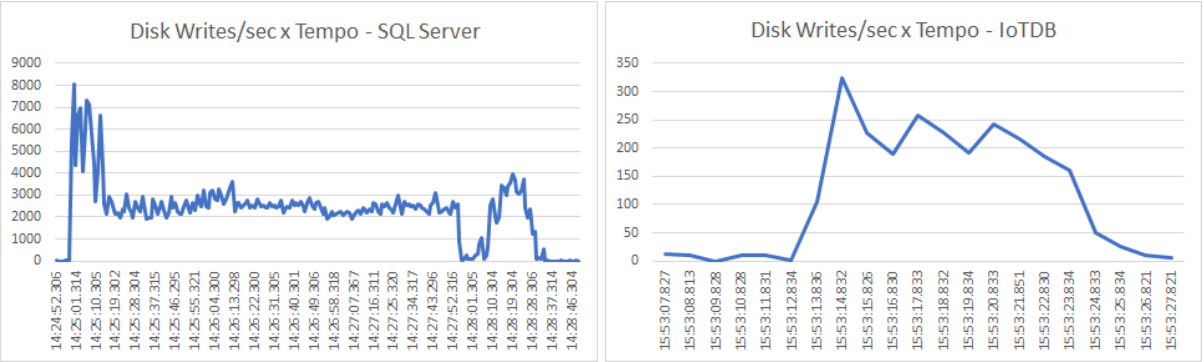

Gráfico 1: Gráfico de escrita em disco ao longo do tempo, construído a partir das métricas coletadas pelo Performance Monitor durante o teste de performance para escrita no SQL Server e IoTDB.

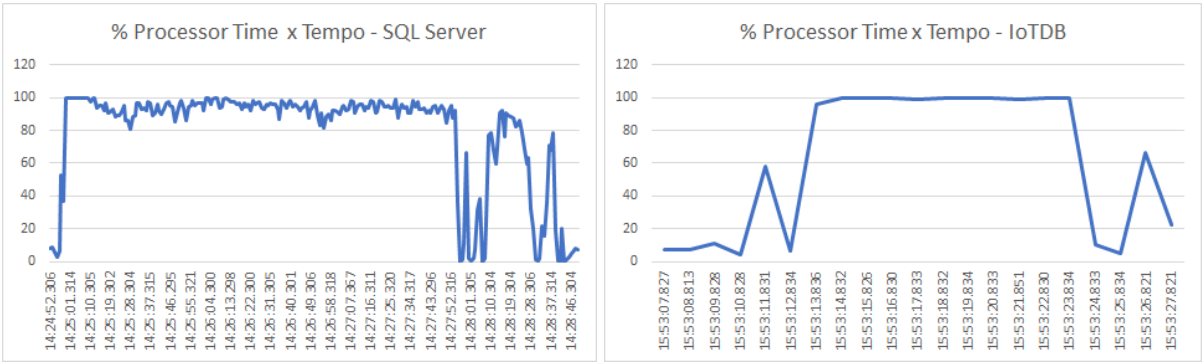

Gráfico 2: Gráfico de Porcentagem do tempo total em que o processador está ocupado ao longo do tempo, construído a partir das métricas coletadas pelo Performance Monitor durante o teste de performance para escrita no SQL Server e IoTDB.

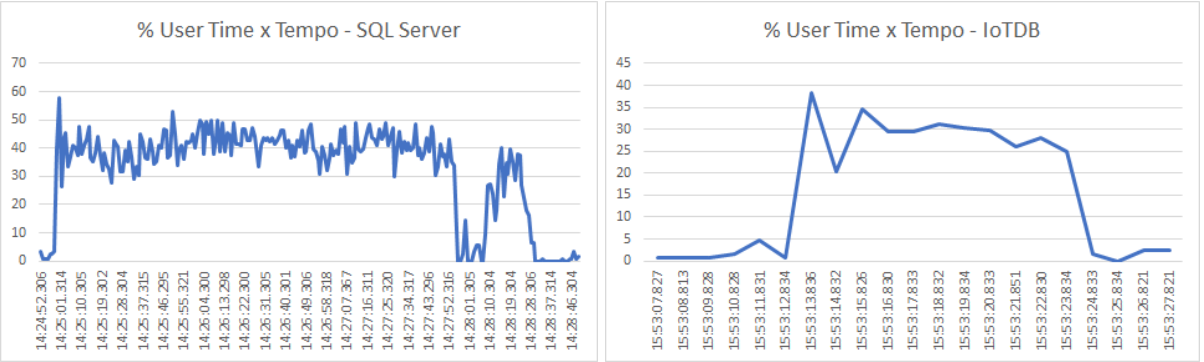

Gráfico 3: Gráfico de Porcentagem do tempo em que o processador está no modo usuário, construído a partir das métricas coletadas pelo Performance Monitor durante o teste de performance para escrita no SQL Server e IoTDB.

Métricas	Mínimo	Média	Máximo
% Processor Time	0,000	69,317	98,.457

% User Time	0,000	27,012	50,000
Disk Writes/sec	0,000	1.882,074	3.941,274

Tabela 7: Métricas coletadas pelo Performance Monitor para o teste de escrita no SQL Server.

Métricas	Mínimo	Média	Máximo
% Processor Time	22,666	61,521	100,000
% User Time	0,000	16,114	38,187
Disk Writes/sec	0,000	117.191,000	325.435,000

Tabela 8: Métricas coletadas pelo Performance Monitor para o teste de escrita no IoTDB.

5. CONCLUSÃO

A rápida evolução na área da tecnologia ocorrida na virada do século 20 e intensificada a partir da década de 2010 com o desenvolvimento da Internet das Coisas, onde toda forma de dispositivo possui capacidade de se conectar a uma rede de computadores e transmitir informações coletadas por ele, impulsionou a busca por formas mais eficientes de processamento de informação.

Para enfrentar este desafio, a utilização da computação de borda, que consiste na disponibilização de recursos computacionais para processamento remoto, semelhante ao cloud computing, tem sido a principal aposta do setor tecnológico para evitar os tempos de latência de rede quando se é necessário realizar a comunicação com servidores que estejam a grandes distâncias. Em acréscimo a isto, é necessário também que as unidades computacionais dispostas na borda da rede para atender esta necessidade processional contem com uma tecnologia de banco de dados que comporte altas taxas de inserção e leitura de informação.

Com o intuito de atender esta necessidade, os bancos de dados de série temporal e em especial aqueles adaptados para atender a necessidades específicas da borda, como o IoTDB com sua alta capacidade de ingestão, compactação e replicação de dados em servidores centrais remotos, são os sistemas gerenciadores de banco de dados mais indicados para esta função.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] DE DONNO, Michele; TANGE, Koen; DRAGONI, Nicola. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. Ieee Access, v. 7, p. 150936-150948, 2019.

[2] SHI, Weisong et al. Edge computing: Vision and challenges. IEEE internet of things journal, v. 3, n. 5, p. 637-646, 2016.

[3] YANG, Yang; CAO, Qiang; JIANG, Hong. EdgeDB: An efficient time-series database for edge computing. IEEE Access, v. 7, p. 142295-142307, 2019.

[4] HSU, Ruei-Hau et al. Reconfigurable security: Edge-computing-based framework for IoT. IEEE Network, v. 32, n. 5, p. 92-99, 2018.

[5] RUMAN, Md Raseduzzaman et al. IoT based emergency health monitoring system. In: 2020 International Conference on Industry 4.0 Technology (I4Tech). IEEE, 2020. p. 159-162.

[6] LEE, J. How to choose the right IoT cloud platform. Disponível em: <https://dzone.com/articles/how-to-choose-the-right-iot-cloud-platform>. Acesso em: 11 jul. 2021. Adaptado.

[7] BANERJEE, Suman et al. ParaDrop: An Edge Computing Platform in Home Gateways. Fog for 5G and IoT; Wiley: Hoboken, NJ, USA, p. 13, 2017.

[8] ROMAN, Rodrigo; LOPEZ, Javier; MAMBO, Masahiro. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems, v. 78, p. 680-698, 2018.

[9] VERBELEN, Tim et al. Cloudlets: Bringing the cloud to the mobile user. In: Proceedings of the third ACM workshop on Mobile cloud computing and services. 2012. p. 29-36.

[10] RAHIMI, M. Reza et al. Mobile cloud computing: A survey, state of art and future directions. Mobile Networks and Applications, v. 19, n. 2, p. 133-143, 2014.

[11] DOLUI, Koustabh; DATTA, Soumya Kanti. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In: 2017 Global Internet of Things Summit (GIoTS). IEEE, 2017. p. 1-6.

[12] WILLIS, Dale; DASGUPTA, Arkodeb; BANERJEE, Suman. Paradrop: a multi-tenant platform to dynamically install third party services on wireless gateways. In: Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture. 2014. p. 43-48.

[13] REN, Ju et al. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. ACM Computing Surveys (CSUR), v. 52, n. 6, p. 1-36, 2019.

[14] SATYANARAYANAN, Mahadev et al. The case for vm-based cloudlets in mobile computing. IEEE pervasive Computing, v. 8, n. 4, p. 14-23, 2009.

[15] DINH, Hoang T. et al. A survey of mobile cloud computing: architecture, applications, and approaches. Wireless communications and mobile computing, v. 13, n. 18, p. 1587-1611, 2013.

[16] DOS REIS BEZERRA, Wesley; KOCH, Fernando Luiz; BECKER WESTPHALL, Carlos. Models of Computing as a Service and IoT: an analysis of the current scenario with applications using LPWAN. arXiv e-prints, p. arXiv: 2105.06050, 2021.

[17] ARVINDPDMN, P. Edge Computing. Disponível em: <https://devopedia.org/edge-computing>. Acesso em: 13 out. 2021.

[18] SHI, Weisong; DUSTDAR, Schahram. The promise of edge computing. Computer, v. 49, n. 5, p. 78-81, 2016.

[19] Real-life use cases for edge computing. Disponível em: <https://innovationatwork.ieee.org/real-life-edge-computing-use-cases/>. Acesso em: 13 out. 2021.

[20] TALEB, Tarik et al. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. IEEE Communications Surveys & Tutorials, v. 19, n. 3, p. 1657-1681, 2017.

[21] Multi-access Edge Computing (MEC); Framework and Reference Architecture. ETSI GS MEC 003 V2.2.1 (2020-12). Disponível em: <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.02.01_60/gs_MEC003v020201p.pdf>. Acesso em: 20 set. 2021.

[22] SABELLA, Dario et al. Mobile-edge computing architecture: The role of MEC in the Internet of Things. IEEE Consumer Electronics Magazine, v. 5, n. 4, p. 84-91, 2016.

[23] Multi-Access Edge Computing (MEC); Terminology. ETSI GS MEC 001 V2.1.1 (2019-01). Disponível em: <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/001/02.01.01_60/gs_MEC001v020101p.pdf>. Acesso em: 20 set. 2021

[24] SABELLA, D. et al. Mobile-edge computing architecture: The role of MEC in the internet of things. IEEE consumer electronics magazine, v. 5, n. 4, p. 84–91, 2016.

[25] EROL-KANTARCI, Melike; SUKHMANI, Sukhmani. Caching and computing at the edge for mobile augmented reality and virtual reality (AR/VR) in 5G. Ad Hoc Networks, p. 169-177, 2018.

[26] ELMASRI, R.; NAVATHE, S. B. Fundamentals of database systems. 7. ed. Upper Saddle River, NJ, USA: Pearson, 2015.

[27] DATE, C. J. An introduction to database systems: International edition. 8. ed. Upper Saddle River, NJ, USA: Pearson, 2004.

[28] ZHANG, Qi; CHENG, Lu; BOUTABA, Raouf. Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, v. 1, n. 1, p. 7-18, 2010.

[29] ARMBRUST, Michael et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[30] VAQUERO, Luis M. et al. A break in the clouds: towards a cloud definition. 2009. SIGCOMM Comput. Commun. Rev. 39, 1 (January 2009), 50–55.

[31] KUNDA, Douglas; PHIRI, Hazael. A comparative study of nosql and relational database. Zambia ICT Journal, v. 1, n. 1, p. 1-4, 2017.

[32] TAURO, Clarence JM; PATIL, Baswanth Rao; PRASHANTH, K. R. A comparative analysis of different nosql databases on data model, query model and replication model. In: Proceedings of the International Conference on ERCICA. 2013.

[33] LIU, Rui; YUAN, Jun. Benchmarking time series databases with IoTDB-benchmark for IoT scenarios. arXiv preprint arXiv:1901.08304, 2019.

[34] RODRIGUES, Wagner Braz et al. NoSQL-A análise da modelagem e consistência dos dados na era do Big Data. 2017.

[35] Data Concept. Disponível em: <https://iotdb.apache.org/UserGuide/Master/Data-Concept/Data-Model-and-Terminology.html>. Acesso em: 31 out. 2021.

[36] Data Type. Disponível em: <https://iotdb.apache.org/UserGuide/Master/Data-Concept/Data-Type.html#data-type>. Acesso em: 31 out. 2021.

[37] Device Actions Overview. Google Assistant. Disponível em: <https://developers.google.com/assistant/sdk/device-actions-overview>. Acesso em: 02 nov. 2021.

[38] Functional Requirements for AVS Products. Amazon Alexa. Disponível em: <https://developer.amazon.com/en-US/docs/alexa/alexa-voice-service/functional-requirements.html>. Acesso em: 02 nov. 2021.

[39] RAMAKRISHNAN, R.; GEHRKE, J. Database Management Systems. 3. ed. Nova Iorque, NY, USA: McGraw-Hill Professional, 2003.

[40] NAQVI, Syeda Noor Zehra; YFANTIDOU, Sofia; ZIMÁNYI, Esteban. Time series databases and influxdb. Studienarbeit, Université Libre de Bruxelles, v. 12, 2017.

[41] NAMIOT, Dmitry. Time Series Databases. DAMDID/RCDL, v. 1536, p. 132-137, 2015.

[42] BADER, Andreas; KOPP, Oliver; FALKENTHAL, Michael. Survey and comparison of open source time series databases. Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband, 2017.

[43] YANG, Fangjin et al. Druid: A real-time analytical data store. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. 2014. p. 157-168.

[44] How VividCortex's MySQL-Based Time Series Database Works. Disponível em: <https://www.youtube.com/watch?v=7WRwB7yNmv8>. Acesso em: 05 nov. 2021.

[45] Baron Schwartz - building A time series database on MySQL - SCALE 13x. Disponível em: <https://www.youtube.com/watch?v=grROVHvMFrQ>. Acesso em: 05 nov. 2021.

[46] BADER, Andreas. Comparison of time series databases. 2016. Dissertação de Mestrado.

[47] WANG, Chen et al. Apache IoTDB: time-series database for internet of things. Proceedings of the VLDB Endowment, v. 13, n. 12, p. 2901-2904, 2020.

[48] Application Overview. Disponível em: <https://iotdb.apache.org/SystemDesign/Architecture/Architecture.html>. Acesso em: 06 nov. 2021.

[49] ANDERSEN, Michael P.; CULLER, David E. Btrdb: Optimizing storage system design for timeseries processing. In: 14th {USENIX} Conference on File and Storage Technologies ({FAST} 16). 2016. p. 39-52.

[50] IoTDB - moving from relational to timeseries - Trevor Hart. Disponível em: <https://www.youtube.com/watch?v=-eZB0qMqcqs>. Acesso em: 06 nov. 2021.

[51] Features. Disponível em: <https://iotdb.apache.org/UserGuide/Master/IoTDB-Introduction/Features.html>. Acesso em: 06 nov. 2021.

[52] BĂZĂR, Cristina et al. The transition from rdbms to nosql; A comparative analysis of three popular non-relational solutions: Cassandra, mongodb and couchbase. Database Systems Journal, v. 5, n. 2, p. 49-59, 2014.

[53] QueryEngine. Disponível em: <https://iotdb.apache.org/SystemDesign/QueryEngine/QueryEngine.html>. Acesso em: 06 nov. 2021.

[54] Storage engine. Disponível em: <https://iotdb.apache.org/SystemDesign/StorageEngine/StorageEngine.html>. Acesso em: 06 nov. 2021.

[55] HEUSER, Carlos Alberto. Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS. Bookman Editora, 2009.

[56] ARAÚJO, M. A. P. Modelagem de Dados–Teoria e Prática. Saber Digital: Revista Eletrônica do CESVA, Valença, v. 1, n. 1, p. 33-69, 2008.

[57] ELLER, Nery Artur et al. Estudo e implementação de um sistema de banco de dados distribuído. 1997.

[58] ABDULLAH, Ahmad; ZHUGE, Qingfeng. From relational databases to NoSQL databases: Performance evaluation. Research Journal of Applied Sciences, Engineering and Technology, v. 11, n. 4, p. 434-439, 2015.

[59] AQEL, Musbah J.; AL-SAKRAN, Aya; HUNAITY, Mohammad. A Comparative Study of NoSQL Databases. Biosci. Biotech. Res. Comm. Special Issue Vol 12 No (1) January 2019.

[60] AMGHAR, Souad; CHERDAL, Safae; MOULINE, Salma. Which NoSQL database for IoT applications?. In: 2018 international conference on selected topics in mobile and wireless networking (mownet). IEEE, 2018. p. 131-137.

[61] Bagga, Simmi; Sharma, Anil. A comparative study of NoSQL databases. The International Conference on Recent Innovations in Computing. Springer, 2020, pp. 51–61.

[62] STEPPAT, N. NoSQL - Do teorema CAP para P?(A|C):(C|L). Disponível em: <https://blog.caelum.com.br/nosql-do-teorema-cap-para-paccl/amp/>. Acesso em: 03 nov. 2021.

[63] EDPRICE-MSFT. Non-relational data and NoSQL. Disponível em: <https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data>. Acesso em: 04 nov. 2021.

[64] ROBVET. Relational vs. NoSQL data. Disponível em: <https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data>. Acesso em: 04 nov. 2021.

[65] NAYAK, Ameya; PORIYA, Anil; POOJARY, Dikshay. Type of NOSQL databases and its comparison with relational databases. International Journal of Applied Information Systems, v. 5, n. 4, p. 16-19, 2013.

[66] SAHATQIJA, Kosovare et al. Comparison between relational and NOSQL databases. In: 2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, 2018. p. 0216-0221.

[67] PATEL, Tejal; ELTAIEB, Tarik. Relational database vs NoSQL. Journal of Multidisciplinary Engineering Science and Technology (JMEST), v. 2, n. 4, p. 691-695, 2015.

[68] BTrDB. Disponível em: <http://btrdb.io/about.html>. Acesso em: 05 nov. 2021.

[69] BREWER, Eric A. Towards robust distributed systems. Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC. ACM Press, 2000.

[70] GILBERT, Seth; LYNCH, Nancy. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. Acm Sigact News, v. 33, n. 2, p. 51-59, 2002.

[71] BTrDB Explained — btrdb v5.11.6 documentation. Disponível em: <https://btrdb.readthedocs.io/en/latest/explained.html>. Acesso em: 07 nov. 2021.

[72] Concepts — btrdb v5.11.6 documentation. Disponível em: <https://btrdb.readthedocs.io/en/latest/concepts.html>. Acesso em: 07 nov. 2021.

[73] iotdb-benchmark: IoTDB-benchmark is a tool for benchmarking IoTDB with other databases and time series solutions. Repositório Thulab no GitHub. Disponível em: <https://github.com/thulab/iotdb-benchmark>. Acesso em: 05 nov. 2021.

[74] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, ‘‘Fog computing and its role in the Internet of Things,’’ in Proc. 1st Ed. MCC Workshop Mobile Cloud Comput. (MCC), New York, NY, USA, 2012, pp. 13–16.

[75] M. Goudarzi, H. Wu, M. Palaniswami and R. Buyya. An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments. IEEE Transactions on Mobile Computing, vol. 20, no. 4, pp. 1298-1311, 2021.

[76] K. H. Abdulkareem et al.. A Review of Fog Computing and Machine Learning: Concepts, Applications, Challenges, and Open Issues. IEEE Access, vol. 7, pp. 153123-153140, 2019.

[77] Zyrianoff, Ivan, et al. Scalability of an Internet of Things platform for smart water management for agriculture. 2018 23rd Conference of Open Innovations Association (FRUCT). IEEE, 2018.

[78] ASIMINIDIS, Christodoulos; KOKKONIS, George; KONTOGIANNIS, Sotirios. Database systems performance evaluation for IoT applications. International Journal of Database Management Systems (IJDMS) Vol, v. 10, 2018.