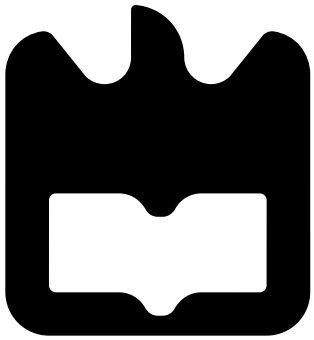
Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
2018

Mafalda
da Mota Rodrigues

Análise de performance de redes de telecomunicações, utilizando bases de dados orientadas às séries temporais

Performance analysis of telecommunications networks, using time series databases



Mafalda
da Mota Rodrigues

Análise de performance de redes de telecomunicações, utilizando bases de dados orientadas às séries temporais

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor José Manuel Matos Moreira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor Joaquim Manuel Henriques de Sousa Pinto
Professor Auxiliar, Universidade de Aveiro

vogais / examiners committee

Professor Doutor João Pedro Carvalho Leal Mendes Moreira
Professor Auxiliar, Universidade do Porto - Faculdade de Engenharia

Professor Doutor José Manuel Matos Moreira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (Orientador)

agradecimentos / acknowledgements

Gostaria de agradecer ao meu orientador José Moreira pela orientação e visão assim como a motivação que disponibilizou para o desenvolvimento deste trabalho.
Ao centro de investigação Nokia de Aveiro pelo acolhimento e entreajuda dada. Ao engenheiro Paulo Costa e particularmente ao engenheiro Ricardo Marques pela oportunidade dada, pelas palavras de incentivo, pela confiança que depositaram em mim e por toda a paciência que tiveram neste processo.
A toda a minha família que estiveram comigo incondicionalmente e permitiram-me concluir esta fase tão importante para a minha vida.
Finalmente, aos meus amigos de todas as horas que sem eles não teria chegado até aqui como o Leandro Ricardo, Marco Silva, Ana Malta, Eduardo Sousa, Mariana Rodrigues e André Garcia. E a todos que fizeram parte de uma maneira ou de outra neste percurso!

Palavras-Chave Base de dados orientadas às séries temporais, Nokia Performance Manager, performance de redes de telecomunicações

Resumo

A evolução e o aparecimento de novas tecnologias na área das telecomunicações levou as operadoras de rede a interessarem-se por novas formas de extrair conhecimento a partir dos dados que recolhem continuadamente. A Nokia utiliza um sistema de gestão de performance de redes (NPM) para monitorizar e analisar o desempenho das redes das operadoras de telecomunicações. O NPM é um sistema que permite processar, normalizar e integrar dados provenientes de numerosos equipamentos de rede que são disponibilizados sob a forma de relatórios. Sabendo que o volume de dados a tratar é grande e que tende a aumentar rapidamente, surge a necessidade de procurar novas soluções de escalabilidade que permitam adaptar a capacidade de processamento do NPM aos volumes de dados recolhidos. Nesse sentido, esta dissertação proposta em parceria com a Nokia, tem como objetivo explorar e comparar diversas tecnologias de bases de dados orientadas às séries temporais (TSDB), para selecionar uma solução que possa ser integrada no NPM. A seleção foi feita com base nas características das tecnologias e nos requisitos definidos pela Nokia, tendo sido selecionado o Apache Kudu. Foi implementado um ambiente de testes e foram realizados testes de desempenho para definir a arquitetura do sistema, e selecionar os métodos de particionamento e de compressão de dados a usar na Nokia. A seguir, foram gerados dados para simular o ambiente típico de funcionamento do NPM e avaliar o desempenho da solução proposta nesta dissertação. Os testes finais englobam os processos de geração de relatórios de rede, inserção de dados e cálculo de agregações executados concorrentemente. Por fim, foram realizados testes para comparar o desempenho do Apache Presto e do Apache Impala na comunicação com o Apache Kudu. Estas ferramentas permitem aceder ao Apache Kudu usando a linguagem SQL e conclui-se que o primeiro tem melhores tempos de resposta, porém o desempenho diminui quando as consultas têm 3 ou mais *joins*. Globalmente, os resultados mostram que o desempenho do Apache Kudu é uma solução viável em termos de desempenho, mas ainda existem problemas ao nível da documentação e da configuração do sistema que permitam garantir a fiabilidade desejada pela Nokia.

Keywords Time series database, Nokia Performance Manager, performance of telecommunications networks

Abstract The evolution and the emergence of new technologies in the area of telecommunication allowed network operators to become interested in new ways of extracting knowledge from the large volumes of data that they are able to collect. Nokia uses the network performance and telecommunication management system (NPM) to analyze network performance as well as the help of its monitoring. NPM allows you to generate network reports according to time periods. This dissertation is a joint proposal with the Nokia and aims to explore, analyze, compare and select a solution directed to the efficient storage and management of time series data, using time series databases (TSDB) so that it is integrated into the NPM. For this reason, a test environment was implemented to simulate the NPM workload and evaluate the selected solution, Apache Kudu. The results obtained allow us to conclude that Apache Kudu demonstrates that the best partitioning method is Combined (Hash plus Range), and Snappy is the best method of data compression. The main test encompasses the generation of network reports, data insertion, and the computation of aggregations simultaneously. Finally, equating communication to Apache Kudu between Apache Presto and Apache Impala, it is concluded that the first one has better response times, but performance decreases when queries include 3 or more *joins*.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

ACID Atomicidade, Consistência, Isolamento, Durabilidade.

ASCII American Standard Code for Information Interchange.

BASE Basically Available, Soft state, Eventual consistency.

CAP Consistência, Disponibilidade e Tolerância ao Particionamento de rede.

CPU Central Processing Unit.

CRUD Create, Read, Update, Delete.

CSV Comma-separated values.

DFS Sistema de ficheiros distribuído.

GB Gigabytes.

HDFS Hadoop Dristributed File System.

IDC International Data Corporation.

JDBC Java Database Connectivity.

JSON JavaScript Object Notation.

KPI Key Performance Indicators.

LTE Long Term Evolution.

MPP Processamento paralelo massivo.

NoSQL Not only SQL.

NPM Nokia Performance Network.

OMeS Open Measurement Standard.

PLMN Public land mobile network.

PM Performance Manager.

RDBMS Sistema de gestão de bases de dados relacionais.

SQL Structured Query Language.

TSDB Base de dados orientadas às séries temporais.

WAL Write Ahead Log.

XML Extensible Markup Language.

ZB Zettabytes.

Capítulo 1

Introdução

1.1 Enquadramento

Nos últimos anos, a evolução tecnológica iniciou um processo de mudança na sociedade, representando uma nova era em que diariamente se observa um crescimento do volume de dados proveniente de diferentes fontes, tais como telemóveis, computadores, centrais elétricas, entre outras. Segundo a Data Age 2025 da IDC (International Data Corporation), é previsto que até 2025 o volume de dados gerados globalmente cresça para 163 Zettabytes (ZB), sendo dez vezes mais do que os 16.1 ZB de dados gerados em 2016 [1]. Por conseguinte, surgiram novos mecanismos e sistemas com o intuito de extrair, compreender e analisar dados, transformando-os em informação relevante, desafiando as empresas e as organizações a mudar a forma como tomam decisões, confiando mais nos dados do que na experiência.

Refletindo sobre o impacto que estes mecanismos e sistemas podem ter no quotidiano, a Nokia criou um sistema para a gestão de performance de redes e telecomunicações, designado por Network Performance Manager (NPM). A Nokia é uma multinacional de performance de redes de telecomunicações, sediada na Finlândia, na cidade de Espoo, com presença em cerca de 150 países, que tem como objetivo a inovação dos equipamentos de telecomunicações, assim como dos serviços de comunicação móvel, tais como 2G, 3G, LTE (Long Term Evolution) e recentemente 5G. Além disso, acolhe dois centros tecnológicos em Portugal, localizados em Lisboa e em Aveiro com cerca de 2000 colaboradores, dedicados a atividades de investigação e cooperação com a indústria e as universidades [2].

A principal finalidade do NPM é recolher dados provenientes da rede e das telecomunicações, de forma a conseguir extrair conhecimento dos mesmos. Este conhecimento ajuda na análise do desempenho da rede, na monitorização da sua qualidade e acessibilidade, resultando na capacidade de deteção e correção de falhas permanentes, previsão de tendências futuras, assim como auxiliar na tomada de decisões dos clientes conforme os seus objetivos. Adicionalmente, o NPM permite um conjunto de funcionalidades como a visualização de relatórios em tabelas, gráficos e mapas; gerar relatórios através de fórmulas de indicadores-chave de desempenho (KPI), exportar dados para Excel e CSV (Comma-separated values) para apresentação sob a forma de tabelas e gráficos; relatórios de navegação; agendamento de relatórios; filtragem e pesquisas complexas, entre outras. Para este efeito, os dados podem ser agregados por dimensões de rede e períodos temporais (ano, mês, dia e hora) possibilitando análises com diferentes níveis de granularidade [3].

1.2 Motivação

A principal motivação desta dissertação proposta em parceria com a Nokia, consiste em perceber se a utilização de bases de dados orientadas às séries temporais pode vir a ser uma mais valia para o sistema de gestão de performance de redes e de telecomunicações (NPM). O NPM é direcionado para operadores de rede de forma a permitir obter numerosos indicadores de performance, que são utilizados para analisar a performance da rede ou dos serviços que esta proporciona. Nesta perspetiva, os relatórios de rede gerados pelo NPM incluem dados inerentemente temporais. A análise de uma rede usando períodos temporais permite obter uma melhor perceção da rede ou do serviço, por exemplo, para analisar as ocorrências de um determinado evento ao longo do tempo e perceber se isso pode representar ou não uma situação anómala. Neste trabalho pretende-se avaliar se é possível encontrar uma solução mais eficiente do que as atualmente em uso na Nokia, alinhada com os requisitos de desempenho atuais e com o aumento previsível do volume de dados a processar no futuro próximo.

1.3 Objetivos

Esta dissertação tem como objetivo explorar, analisar, comparar e selecionar uma solução direcionada para armazenamento e processamento eficiente de séries de dados, recorrendo especificamente a bases de dados orientadas às séries temporais (TSDB). Pretende-se integrar a TSDB selecionada com o NPM e avaliar a sua performance na geração dos relatórios de rede. Para concretizar este objetivo, o trabalho foi dividido em 4 etapas:

- Pesquisa, análise e comparação de TSDB.
- Seleção de uma tecnologia TSDB.
- Definição da arquitetura e implementação do sistema com a tecnologia selecionada.
- Implementação de um ambiente de testes e discussão dos resultados.

A primeira etapa consiste numa pesquisa e avaliação das tecnologias TSDB existentes atualmente no mercado (Apache Kudu, TimescaleDB, entre outras). É feita uma análise e comparação das características genéricas destas tecnologias, tendo ainda em atenção duas características importantes para este projeto: escalabilidade horizontal e alta disponibilidade.

A informação recolhida foi organizada por tópicos de comparação, considerando as características das tecnologias e os pré-requisitos estabelecidos pela Nokia. Estes pré-requisitos conduziram à exclusão de várias tecnologias e a seleção final foi feita por comparação das outras considerando a informação geral, sobre o suporte comercial e o licenciamento, características como o modelo de dados, a linguagem e o armazenamento e, por fim, a escalabilidade horizontal e alta disponibilidade.

A terceira etapa consiste em integrar a tecnologia selecionada na etapa anterior, analisando os componentes de armazenamento que poderiam ser substituídos por uma solução baseada em TSDB. Deve ser feita uma implementação num ambiente de testes, dividida em duas fases: estudar como implementar o processo de pré-processamento e inserção de dados na TSDB; implementar os processos de consulta de dados, considerando o fluxo de dados desde o pedido de um cliente até à TSDB, e o retorno dos resultados.

Por fim, a quarta etapa consiste em estabelecer um ambiente de testes, adaptando os procedimentos de geração de dados, relatórios de rede e cargas de trabalho usados no âmbito do NPM, de forma a poderem ser integrados com a nova tecnologia testada neste trabalho. O ambiente de testes fornece dados sobre a infraestrutura da rede bem como as métricas a usar para a avaliação dos resultados obtidos usando a TSDB. Foram consideradas diversas estratégias de avaliação, para analisar diferentes características da TSDB. Os testes finalizam com a geração de relatórios de rede com diferentes cargas e a comparação da comunicação com duas tecnologias que permitem consultas à TSDB usando SQL.

1.4 Estrutura da dissertação

Este documento encontra-se organizado em 7 capítulos. O primeiro capítulo faz um enquadramento e explicita a motivação e os objetivos do trabalho. Também faz uma apresentação das principais etapas do trabalho.

O segundo capítulo apresenta conceitos principais, descrevendo o aparecimento do conceito de Big Data, as limitações das bases de dados relacionais (RDBMS) neste contexto de utilização, o papel das bases de dados NoSQL (Not only SQL) e o surgimento das TSDB.

No terceiro capítulo são analisadas as principais tecnologias direcionadas para a gestão de dados em séries temporais. Apresentam-se as tecnologias do ecossistema Hadoop, Apache Kudu e Timescale, acrescentando formatos para o armazenamento e serialização de dados, tecnologias para o processamento de dados e, por fim, tecnologias para a consulta dos dados.

O quarto capítulo descreve o trabalho desenvolvido para a utilização de TSDB no NPM. Começa pela apresentação do problema e do sistema atual (NPM); apresenta os requisitos propostos pela Nokia e os critérios que estiveram por base da escolha da tecnologia adotada neste trabalho.

O quinto capítulo descreve a arquitetura proposta usando a TSDB selecionada e as propriedades essenciais a considerar na sua implementação. Explica-se o trabalho de implementação realizado, as configurações necessárias ao bom funcionamento do sistema, assim como os processos de inserção e consulta de dados.

O sexto capítulo apresenta o ambiente de testes, a sua infraestrutura, as métricas de avaliação utilizadas e as estratégias delineadas para avaliação do sistema implementado. Apresentam-se os resultados e o discute-se o comportamento da TSDB mediante as diferentes estratégias de avaliação.

O documento termina apresentando as principais conclusões deste trabalho, as lições aprendidas e algumas ideias para trabalho futuro.

Capítulo 2

Bases de dados orientadas às séries temporais

Este capítulo apresenta os principais conceitos que foram relevantes para este trabalho, apresentando definições que servem de base para a sua compreensão. A secção 2.1 começa por apresentar o conceito de **Big Data**, as suas principais características e o impacto no desenvolvimento de novas soluções de armazenamento e gestão de dados. Apresenta-se o contexto que levou ao aparecimento de novas alternativas face às limitações das **bases de dados relacionais**, nomeadamente as **bases de dados NoSQL**, dando ênfase às suas principais características e propriedades. De seguida, salienta-se o aparecimento das **bases de dados orientadas às séries temporais**, apresentando o modelo de dados subjacente e as suas principais características. No final, apresentam-se alguns trabalhos sobre as TSDB relevantes para esta dissertação.

2.1 Big Data

2.1.1 Conceito

O conceito de Big Data está interligado com a evolução tecnológica e com o aumento do volume de dados gerados por diversos dispositivos tecnológicos, assim como com a variedade e a complexidade dos seus formatos. Consequentemente, surgiram novos desafios tendo em vista encontrar novas tecnologias capazes de recolher, processar, armazenar e analisar grandes volumes de dados e ter tempos de resposta baixos [4, 5].

O Big Data é normalmente descrito em função das suas principais características, nomeadamente: Volume, Velocidade, Variedade, Veracidade e Valor, ou por outras palavras, pelos "5V's"[6, 7].

- **Volume -** Representa a quantidade de dados gerados e que devem ser processados.
- **Velocidade -** Refere-se à velocidade com que os dados são produzidos e à necessidade de serem processados e analisados em quase tempo-real.
- **Variedade -** Define os diferentes formatos dos dados existentes, que podem ser estruturados, semi-estruturados ou não estruturados.

- **Veracidade -** Refere-se ao grau de confiança nos dados consoante a qualidade dos mesmos.

- **Valor -** Define a capacidade de transformar informação em conhecimento a partir da análise de dados.

Estas características refletem o estado atual das tecnologias desenvolvidas nos últimos anos, nas quais novas plataformas permitem gerar cada vez mais dados em diversos formatos, por exemplo, texto, áudio ou vídeo. Atualmente são vistos no Youtube mais de 4 biliões de horas de vídeos por mês [8]. Além disso, existe uma aposta cada vez maior na diversidade e na quantidade de dispositivos móveis sempre conectados, como telemóveis, tablets e sensores, que aumentam a produção e o consumo médio diário de dados. Por exemplo, os carros modernos incluem aproximadamente 100 sensores para fazer a sua monitorização, como controlar o nível de gasóleo e da pressão dos pneus [8].

2.1.2 Desafios

Face ao aumento da quantidade e complexidade dos dados gerados diariamente, começaram a surgir novos paradigmas e desafios para as RDBMS.

As RDBMS são essencialmente base de dados centralizadas desenhadas para a gestão de dados normalizados e estruturados, utilizando uma linguagem declarativa para as consultas designada por SQL (Structured Query Language). As RDBMS usam transações com propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade), para garantir a atomicidade de cada transação, ou seja, cada tarefa incluída numa transação tem que ser executada como um todo, não permitindo falhas parciais; a consistência dos dados no principio e no fim da transação; o isolamento, assegurando que nenhuma transação tem acesso a dados modificados por outra transação que estejam num estado intermediário ou inacabado, pelo que cada transação é independente e isolada; e durável, isto é, após a conclusão de uma transação, o seu estado será sempre imutável [9, 10].

Porém, sabendo que hoje em dia 80% dos dados gerados pelas empresas são semi-estruturados ou não estruturados [11], a representação desses dados em RDBMS implicaria implementar sistemas complexos para transformá-los e colocá-los num formato estruturado. Além disso, a arquitetura das RDBMS é orientada para a escalabilidade vertical, ou seja, com o aumento do volume de dados é preciso adquirir *hardware* mais caro, com mais capacidade de processamento, memória e armazenamento [9]. Estas limitações das RDBMS impulsionaram o desenvolvimento de novas abordagens de processamento e armazenamento de dados.

2.2 Base de dados NoSQL

As base de dados NoSQL surgiram da necessidade de analisar um grande volume de dados, possibilitando a distribuição dos dados por vários servidores de forma a garantir respostas rápidas [12]. As principais características destas bases de dados são: processamento de uma grande quantidade de dados de forma eficiente e rápida, modelo de dados versátil, aceitando dados estruturados e não estruturados e, por fim, a inexistência da obrigatoriedade de usar SQL como linguagem [13].

As bases de dados NoSQL seguem uma filosofia de desenho designada por BASE (Basically Available, Soft State, Eventual Consistency) não suportando transações ACID tal como as RDBMS. O atributo *"Basically Available"*, significa que um sistema garante a disponibilidade dos dados, mesmo que existam inconsistências. *"Soft State"* significa que o sistema pode alterar-se ao longo do tempo, permitindo a inconsistência dos dados e, *"Eventual Consistency"*, significa que o sistema se vai tornar consistente em algum momento no futuro [10].

O teorema de CAP (Consistência, Disponibilidade e Tolerância ao Particionamento), surge associado aos sistemas distribuídos e afirma que um sistema só pode conter duas das três propriedades seguintes em simultâneo: [10, 13, 14]

- **Consistência -** Refere-se à consistência de cada transação, ou seja, os dados ficam disponíveis para serem utilizados após a sua inserção ou atualização.
- **Disponibilidade -** Assegura que um sistema permanece ativo durante um determinado período de tempo.
- **Tolerância ao Particionamento -** Representa a capacidade de um sistema continuar disponível após uma falha, ou seja, garante que as operações vão ser completadas, mesmo que existam falhas em alguns dos componentes.

Atendendo à diversidade de problemas e aplicações que têm emergido nos últimos anos, surgiram diversas propostas de classificação associadas às bases de dados NoSQL, por exemplo:

- **Base de dados chave-valor -** Armazenam os dados em tabelas distribuídas usando o método de *hashing*, onde existe uma chave única correspondente a um valor. Por esta razão, são caracterizadas por serem simples, escaláveis e permitirem consultas rápidas, com pouca latência [6, 13, 15]. Dois exemplos bem conhecidos são o Riak [1] e o DynamoDB [2].
- **Base de dados orientada a documentos -** São caracterizadas por armazenarem os dados em documentos, que podem ter formatos complexos, por exemplo, XML, JSON, entre outros. O acesso e o esquema de armazenamento dos dados são flexíveis [6, 13, 15]. Um dos exemplos mais conhecidos é o MongoDB [3].
- **Base de dados orientada a colunas -** Caracterizam-se por armazenarem os dados em colunas e famílias de colunas. Por esta razão, facilitam a compressão dos dados e o particionamento funcional por coluna, sendo eficientes no cálculo de agregações, entre outras operações [6, 13, 15]. Cassandra [4] e Hbase [5] são dois exemplos de bases de dados orientadas a colunas.
- **Base de dados orientada a grafos -** Utilizam grafos para o armazenamento dos dados, incluindo nós, relacionamento entre os nós e propriedades dos nós. Por esta razão, são usadas quando existe um maior interesse nos relacionamentos entre os dados

[1] http://basho.com/products/
[2] https://aws.amazon.com/pt/dynamodb/
[3] https://www.mongodb.com/
[4] http://cassandra.apache.org/
[5] https://hbase.apache.org/

do que nos dados em si [6, 13, 15]. Um exemplo de uma base de dados orientada a grafos é o Neo4j [6].

2.3 Base de dados orientada às séries temporais

Muitos dos dados gerados pelos dispositivos mais recentes têm uma componente temporal que importa analisar, por exemplo um carro conectado pode recolher 25GB de dados por hora [16]. Por este motivo surgiu a necessidade de criar métodos e formas de processamento especializados para séries de dados temporais. Face a esta nova necessidade, as TSDB têm procurado desenvolver modelos de dados de dados e métodos de processamento otimizados baseando-se nas características das bases de dados NoSQL, nomeadamente a escalabilidade horizontal.

2.3.1 Séries Temporais

O conceito de séries temporais não é novo e é geralmente associado a uma representação de uma sequência ordenada de dados medidos ao longo de um período de tempo. Por outras palavras, os dados simbolizam eventos representados por um valor numérico e um valor temporal [17–19]. As principais características de uma série temporal são: ter sempre um atributo temporal (*timestamp*), os valores uma vez inseridos não devem ser atualizados e, por fim, os novos dados são tipicamente mais recentes do que os anteriores. As séries temporais estão presentes em muitas aplicações, por exemplo em sequências de áudio, na monitorização financeira, ambiental e em sistemas computacionais [20, 21]. Uma série temporal pode ser estudada em vários contextos, tais como: [18]

- **Análise de séries temporais -** A análise de séries temporais é utilizada para determinar a causa de o valor de uma variável mudar ao longo do tempo ou para interpretar as mudanças de uma variável comparando-a com outras variáveis no mesmo período temporal.

- **Previsão -** A previsão utiliza a informação sobre os dados históricos e os padrões associados, de forma a prever atividades futuras, considerando fatores como a sazonalidade ou tendências de longo prazo.

2.3.2 Modelo de dados

Atualmente, existem várias abordagens para modelar e armazenar os dados numa TSDB, porém só irão ser descritos os modelos mais relevantes, tais como os modelos de dados relacionais e os modelos de dados orientados por colunas.

Modelo de dados relacional

Uma TSDB que utiliza o modelo de dados relacional tem como vantagens usar a linguagem SQL e as propriedades ACID das RDBMS. Contudo, a chave primária tem a componente temporal contrariamente, à RDBMS que não inclui esta particularidade. Para descrever as

[6] https://neo4j.com/

TSDB baseadas no modelo dados relacional será tido como exemplo o TimescaleDB, que é uma extensão do PostgreSQL [7].

Os dados são representados em linhas, em que cada linha é constituída por um valor temporal seguido por outros tipos de campos (por exemplo: *float*, *int*, entre outros). Além disso, permite a criação de índices e chaves secundárias, onde estas podem armazenar metadados adicionais. Contudo, neste modelo é necessário obrigatoriamente escolher previamente um esquema e decidir explicitamente se existem ou não índices [22, 23].

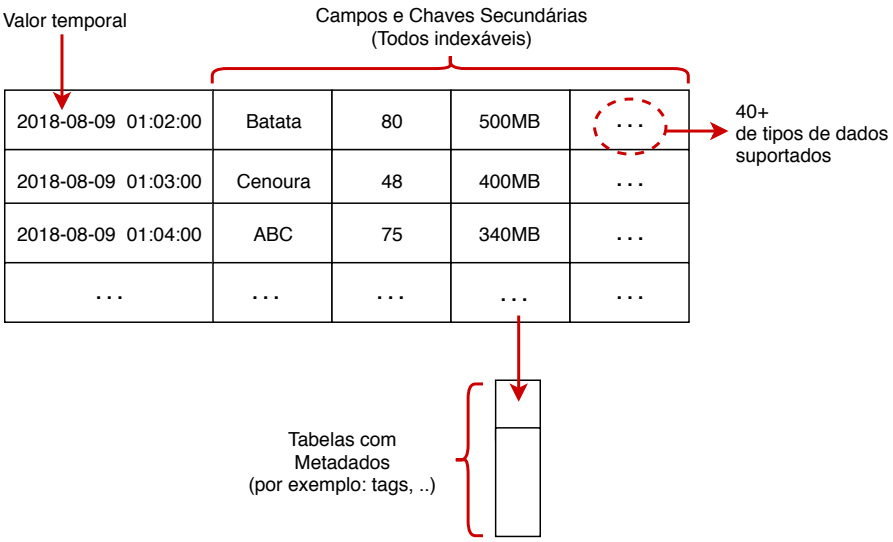


Figura 2.1: Modelo de dados relacional, adaptado de [22, 23]

Modelo de dados orientado por colunas

Os dados numa TSDB orientada por colunas são representados por uma chave designada por chave primária ou chave de partição, e valores, tal como ilustrado na Figura 2.2 [20]. A chave é composta pelo identificador de uma série temporal e por um *timestamp* (data e hora) representando o inicio de um intervalo de tempo. Cada série temporal está associada a um objeto, por exemplo, um equipamento de rede. Os valores são medidas associadas ao objeto capturadas ao longo do tempo. Cada coluna representada na Figura 2.2 corresponde a um deslocamento temporal relativamente ao *timestamp* da chave. Fisicamente os dados são ordenados pela chave, pelo que os dados de uma série temporal ficam armazenados em zonas próximas no disco. Isto significa que uma consulta sobre uma determinada série temporal num intervalo de tempo envolverá operações em disco maioritariamente sequenciais e, portanto, será mais rápida do que seria caso os dados estivessem dispersos.

[7] https://docs.timescale.com/v0.8/introduction/data-model

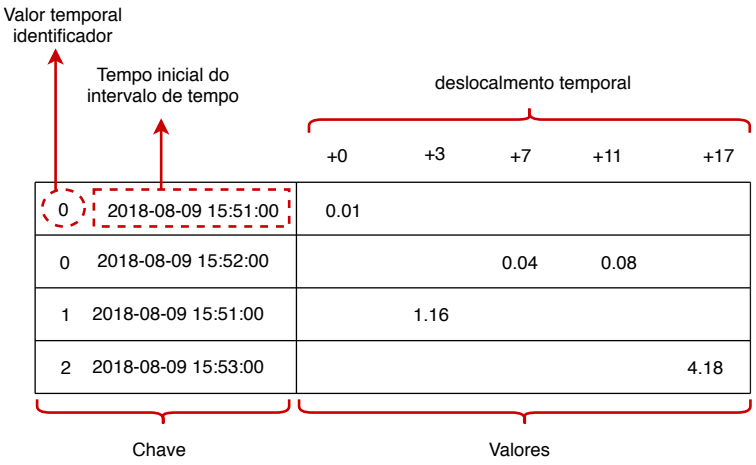


Figura 2.2: Modelo de dados orientado por colunas de uma TSDB, adaptado de [20]

2.3.3 Características

Ao optar por uma TSDB, existem algumas considerações importantes que se devem ter em atenção, nomeadamente: [24]

- **Localização dos dados:** Os dados relacionados, ou seja, dados dentro do mesmo intervalo de tempo, são armazenados na mesma parte do *cluster* da base de dados, o que permite o acesso rápido para uma análise mais rápida e eficiente.
- **Consultas rápidas e de complexidade reduzida:** Como as TSDB mantêm os dados correlacionados juntos, garantem que as consultas sejam rápidas. Por conseguinte, deve ser utilizada uma linguagem que facilite a análise dos dados de séries temporais.
- **Alto desempenho de escrita:** Devem garantir alta disponibilidade e alto desempenho durante a execução de um grande volume de operações de consulta e escrita de dados. Muitas vezes, os dados da série temporal são escritos a cada segundo ou até menos do que isso, portanto, as operações de escrita precisam de ser o mais eficientes possível.
- **Compressão dos dados:** À medida que os dados de séries temporais ficam mais antigos, a granularidade dos dados geralmente torna-se menos relevante para o propósito de análise útil. Para poder armazenar e recuperar os dados de forma eficiente, as organizações devem ter a capacidade de acumular e comprimir os dados de acordo com as necessidades do utilizador.

Por fim, uma base de dados otimizada para manipular dados de séries temporais deve fornecer as seguintes vantagens: [24]

- **Escalabilidade e desempenho:** As TSDB permitem ter escalabilidade horizontal e introduzem funcionalidades para otimização, resultando em melhoria de desempenho, incluindo inserção e consultas mais rápidas.
- **Tempo de inatividade reduzido:** As TSDB devem evitar e minimizar o tempo de indisponibilidade dos dados, por exemplo, no caso de falhas de partições ou falhas de *hardware*.

- **Baixo custo:** As TSDB podem usar *hardware* comum para a sua escalabilidade, reduzindo, por este motivo, custos operacionais.
- **Melhores decisões de negócios:** As TSDB permitem a análise de dados em quase tempo real, permitindo às organizações fazer ajustes ao seu funcionamento mais rápidos e precisos. Por exemplo, a deteção atempada de situações anómalas pode levar a tomar medidas para mitigar as suas consequências mais cedo.

2.4 Trabalhos relacionados

Esta secção apresenta alguns dos trabalhos relacionados com a implementação de TSDB, nomeadamente o LittleTable, o Gorilla e o BTrDB.

O LittleTable foi criado pela Cisco Meraki para uso interno e é uma base de dados relacional otimizada para os dados de séries temporais. Este sistema tem como objetivo armazenar estatísticas de uso, registo de eventos, assim como outros dados de séries temporais [25]. Complementarmente, os dados são agrupados em duas dimensões através da combinação do atributo data e hora (*timestamp*), e uma chave de identificação dos objetos. As chaves têm uma ordem e podem ter uma organização hierárquica. Assim, o LittleTable permite representar os dados como uma tabela bidimensional, como se os dados estivessem organizados em retângulos, em que um dos eixos representa um conjunto de chaves (identificadores de objetos) contíguas e o outro eixo representa um intervalo de tempo (Figura 2.3). Considera-se que os dados contíguos têm maior probabilidade de serem acedidos em conjunto e devem ser armazenados na mesma zona do disco para o acesso mais rápido [25].

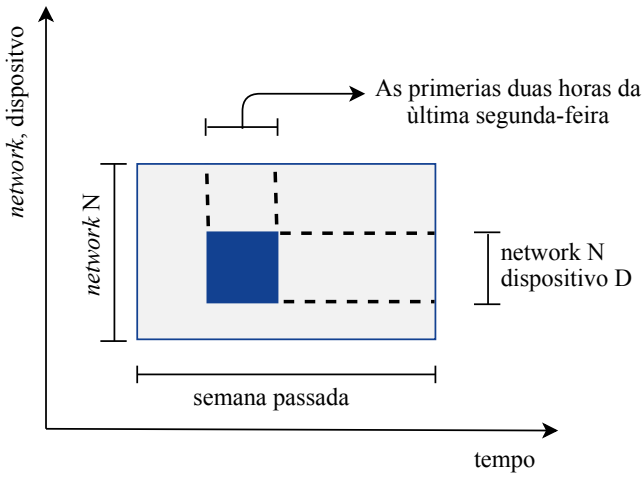


Figura 2.3: Agregação bidimensional, adaptado de [25]

O Gorilla é um sistema criado pelo Facebook para uso interno que foi implementada como uma TSDB em memória distribuída *open-source* [26], denominada por Beringei. Funciona como uma cache para monitorização dos dados da infraestrutura do Facebook, armazenando os dados históricos no Apache HBase. Adicionalmente, o Facebook tinha como finalidade inserir mais de 700 milhões de pontos de dados (*timestamp* e valor) por minuto assim como também melhorar o tempo de resposta das consultas para os dados armazenados nas últimas

26 horas. Por esta razão utilizam um sistema de compressão dos dados para reduzir os custos de armazenamento.

A arquitetura do Gorilla utiliza um sistema de compressão de *streaming* sem perdas para comprimir os dados de uma série temporal, que pode ser visualizado Figura 2.4; Os dados são armazenados num tuplo com uma chave do tipo string (caracteres), um *timestamp* de 64 bits e um valor de vírgula flutuante. Contudo, o *timestamp* e o seu valor correspondente são comprimidos separadamente, com dois métodos de compressão diferentes.

O *timestamp* utiliza o método de compressão *delta-of-delta*, no qual em primeiro lugar é calculada a diferença entre o *timestamp* atual e o anterior, armazenando o resultado do valor delta final, representado menos bits que o valor do *timestamp* original. No entanto, como as séries temporais utilizadas na monitorização no Facebook acontecem num intervalo de tempo regular, permite que o resultado do valor delta final em 96% dos casos possa ser armazenado em um único bit [26].

Os valores associados a cada *timestamp* na série temporal utilizam o método de compressão XOR, porque este é frequentemente armazenado como um valor inteiro assim como também o seu valor não muda significativamente quando comparado aos seus valores vizinhos. Mais precisamente, este método calcula um XOR do valor atual e anterior, armazenando a sua diferença.

O Gorilla fornece um novo sistema de compressão que permite armazenar de forma eficiente os dados de séries temporais, reduzindo a latência nas consultas e tendo alta disponibilidade.

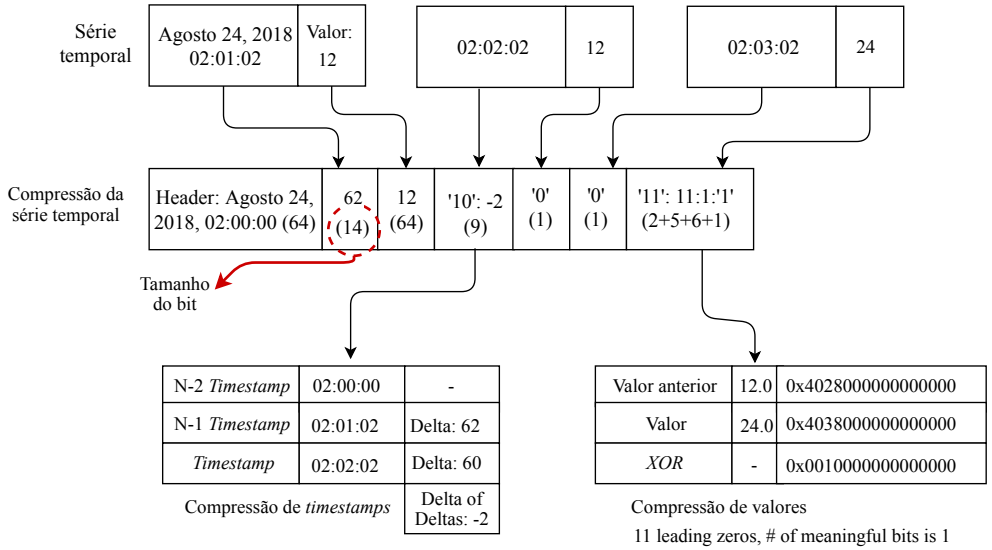


Figura 2.4: Algoritmo de compressão do Gorilla, adaptado de [26]

O BTrDB (Berkeley Tree Database) é uma alternativa às TSDB existentes, com o intuito de suportar sensores de rede de alta precisão que geram dados de telemetria, em que estes podem chegar fora de ordem, atrasados e duplicados. Este sistema tem como principal objetivo suportar 1000 dispositivos por servidor, mais de 1,4 milhões de dados inseridos por segundo e consultas e inserções para análise de dados [27]. Complementarmente, o BTrDB contém uma estrutura de dados que consiste numa árvore *k-ary* com controlo de versões, particionamento

de tempo e *copy-on-write*, possibilitando a rápida localização dos dados num determinado período temporal. Esta estrutura suporta algumas operações básicas tais como: *InsertValues* que cria uma nova versão de *stream* com os pares (tempo, valor) fornecidos e *GetRange* que consulta os dados entre dois períodos temporais numa determinada versão de *streams*. Para armazenar os dados o BTrDB usa o sistema de ficheiros do computador ou o Ceph, que é um repositório distribuído que fornece replicação e recuperação dos dados.

A figura 2.5 ilustra um exemplo de uma árvore com particionamento em que os intervalos de tempo têm a duração de 16 nanossegundos (ns); inclui uma lista de pares de dados tempo(T) e valor(V) para cada 4 ns em cada nó folha e o número da versão em que foi introduzido. Os dados históricos são armazenados na árvore utilizando *copy-on-write*, porque cada inserção forma uma sobreposição na árvore anterior acessível através de um novo nó. Por fim, cada nó armazena estatísticas associadas aos dados na sua sub-árvore, nomeadamente o valor mínimo, o valor médio, o valor máximo e o número de pontos de dados.

O BTrDB fornece um novo mecanismo de armazenamento e consulta de dados de séries temporais de telemetria, onde existem funções estatísticas de agregação e indexação por tempo. Além disso, consegue fazer a inserção de 53 milhões de dados por segundo e 119 milhões de dados consultados por segundo com um *cluster* de quatro nós.

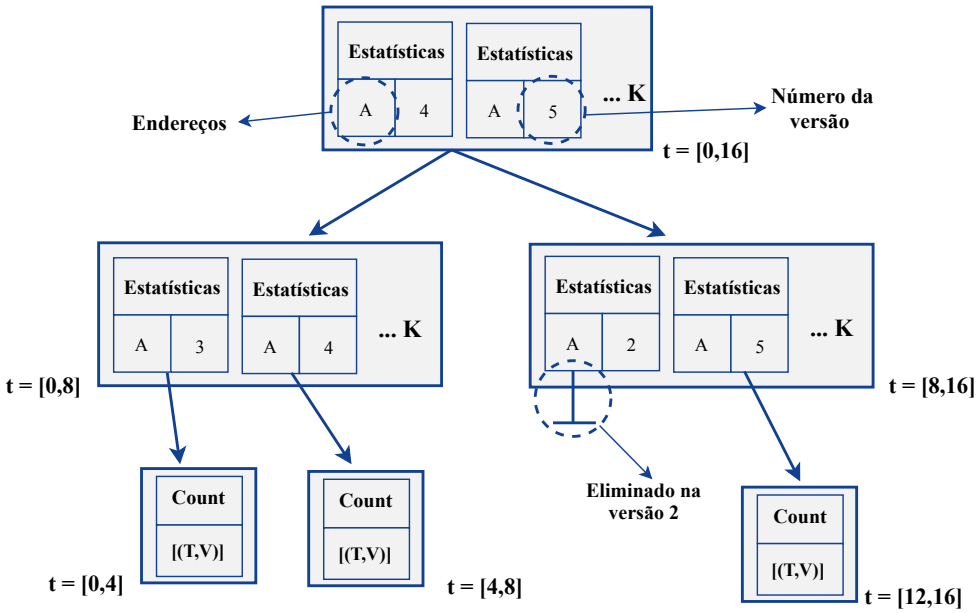


Figura 2.5: Um exemplo de uma árvore com particionamento de tempo e controle de versão. O tamanho dos nós correspondem a uma implementação de K=64, adaptado de [27]

A análise de alguns trabalhos desenvolvidos por outros autores representou um início adequado para o entendimento do modelo de dados, métodos de compressão, modelos de armazenamento utilizados por uma TSDB. Estes trabalhos expõem a necessidade que os levou a criar uma TSDB, destacando os principais problemas e as soluções encontradas para resolvê-los.

2.5 Síntese

Este capítulo apresenta um conjunto de conceitos, de forma a contextualizar o surgimento das TSDB. Começa por apresentar o conceito de Big Data, as suas principais características e a sua influência no aparecimento de novos sistemas de armazenamento. Faz uma introdução às base de dados NoSQL e ao aparecimento das TSDB. Realça a importância da análise de dados em séries temporais, o seu modelo de dados, as suas características, finalizando com a apresentação de alguns trabalhos relacionados que se destacaram nesta área.

Os conceitos estudados neste capítulo são importantes para fundamentar a escolha da TSDB mais adequada a usar no NPM. A seguir, deverá que ser feita uma análise das tecnologias existentes no mercado com potencial para serem utilizadas neste projeto.

Capítulo 3

Tecnologias relacionadas com as base de dados orientadas às séries temporais

Este capítulo faz uma análise de algumas tecnologias usadas no processamento e armazenamento dos dados otimizadas para as séries temporais, que poderiam ser integradas no NPM. Apresenta as tecnologias incluídas no Ecossistema Hadoop, o Timescale e o Apache Kudu. Em seguida apresentam-se alguns formatos para o armazenamento e serialização de dados e, posteriormente, tecnologias para o processamento de grandes volumes de dados. Por fim, analisaram-se as tecnologias que permitem a consulta dos dados através da linguagem SQL e que poderiam ser integradas nas tecnologias de armazenamento de dados.

3.1 Ecossistema Hadoop

O Apache Hadoop é uma tecnologia *open-source* que está preparada para processar e armazenar grandes quantidades de dados, sendo uma possível solução para o Big Data. Está preparado para ser mais flexível na recolha, processamento e análise de dados, comparativamente às base de dados tradicionais, assim como a processar dados com formatos heterogéneos [28].

O Hadoop permite o processamento distribuído dos dados, dividindo-os em partes e desta forma cria conjuntos de tarefas mais pequenas de modo a ser mais fácil o seu processamento. Além disso, permite um armazenamento distribuído e tolerante a falhas, permitindo adicionar facilmente novas máquinas (escalabilidade horizontal) e assegurar a alta disponibilidade dos sistemas implementados sobre Hadoop [29, 30].

O Hadoop recorre a um sistema de ficheiros distribuídos designado por Hadoop Dristributed File System (HDFS), que permite armazenar grandes quantidades de dados de forma redundante e particionada. O HDFS é tolerante a falhas e foi desenvolvido para ser utilizado em *hardware* comum [31]. Este tem uma arquitetura (master/worker), como pode ser observado na Figura 3.1, constituída por um Namenode (master) e vários Datanodes (workers). O Namenode tem como principal função fazer a gestão dos metadados do sistema de gestão de armazenamento para que os Datanodes façam o armazenamento dos dados [28, 30, 32].

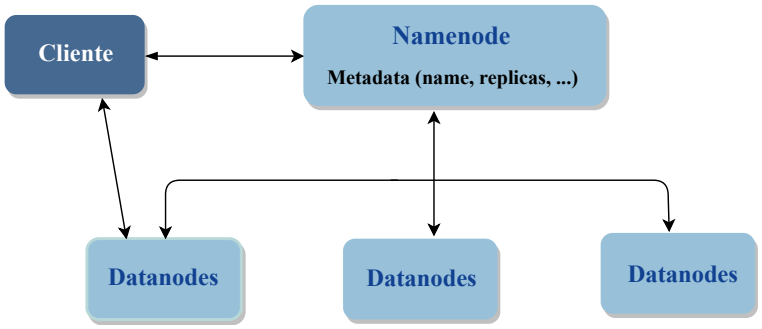


Figura 3.1: Arquitetura do HDFS, adaptado de [28]

O Hadoop usa o Yarn para a gestão das tarefas e dos recursos dos clusters para que seja possível utilizar o modelo de programação de processamento distribuído designado por MapReduce. Este modelo tem duas fases de processamento: o Map e o Reduce. A primeira fase consiste em processar os dados de entrada dividindo-os em tarefas mais pequenas pelas várias máquinas do cluster, enviando o resultado para a fase seguinte. A segunda fase é responsável por associar o resultado das várias máquinas do cluster [29, 33].

As soluções do ecossistema Hadoop que mais se destacam são:

- **Ambari -** Fornece uma interface Web para a gestão e monitorização do Hadoop, que suporta HDFS, Hive, Hbase, Zookeeper, entre outros [28, 34].
- **Cassandra -** Base de dados NoSQL escalável, sem pontos de falha [28, 35].
- **Hbase -** Base de dados NoSQL orientada às colunas, escalável, que fornece acesso rápido a grandes conjuntos de dados [28, 36].
- **Hive -** É uma tecnologia de *data warehouse*, em que se destaca a capacidade para a gestão de grandes conjuntos de dados por causa do seu armazenamento distribuído e a utilização da linguagem SQL [28, 37].
- **Zookeeper -** Serviço *open-source* centralizado para a coordenação de sistemas distribuídos [28, 38].

3.2 Timescale

O Timescale é uma base de dados orientada às séries temporais, *open-source*, otimizada para a inserção rápida e para consultas complexas, que utiliza o PostgreSQL como sistema de armazenamento. Por esta razão, o Timescale disponibiliza todas as funcionalidades do PostgreSQL, adicionando algumas funcionalidades novas para o armazenamento de dados e processamento de consultas [22, 39].

Os dados são armazenados em estruturas de dados designadas *hypertables*. Cada *hypertable* é divida em *chunks*, organizados usando um valor temporal e uma chave de partição, como pode ser visualizado na Figura 3.2. Desta forma, as inserções são otimizadas, porque os dados podem ser adicionados diretamente no *chunk* correspondente, assim como as consultas,

porque as pesquisas são feitas apenas no *chunk* respetivo. Além disso, o Timescale pode ser integrado com tecnologias de visualização de dados, tais como o Grafana [1] e o Tableau [2].

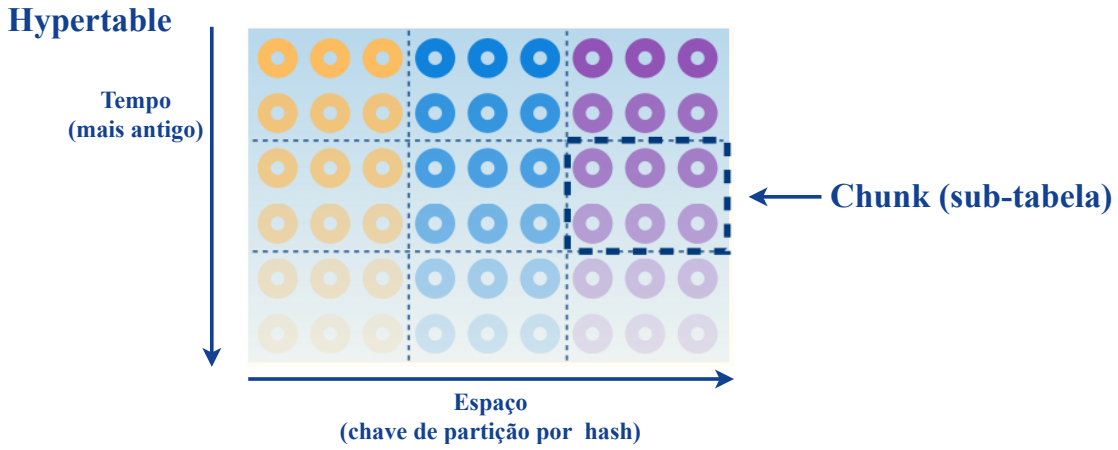


Figura 3.2: Arquitetura do Timescale, adaptado de [40]

3.3 Apache Kudu

O Apache Kudu é uma base de dados NoSQL orientada às colunas e *open source*, que tem como intuito fornecer baixa latência em operações CRUD (inserção, consulta, atualização e eliminação) assim como a otimização de consultas aleatórias com o uso de *joins* e agregações. Esta base de dados pode ser integrada no ecossistema Hadoop; no entanto, não precisa das tecnologias como HDFS e Zookeeper. O modelo de dados é similar às RDBMS, porque exige que os dados para cada tabela sejam estruturados e identificados por uma chave primária única, porém não contém índices secundários. A chave primária pode ser um atributo ou a união de dois atributos. O Kudu garante tolerância a falhas, replicando os dados pelo menos por 3 máquinas. Os dados são divididos em unidades menores denominadas de *tablets* [41, 42].

3.3.1 Arquitetura e Replicação

O Apache Kudu tem uma arquitetura (master/worker), tipicamente composta por pelo menos 3 ou 5 Kudu Master Server (master) e por 3 ou mais Kudu Tablets Server (workers), como pode ser visualizado na Figura 3.3. O Apache Kudu precisa que existam pelo menos 3 máquinas para cada função, devido ao modo de como este faz o armazenamento, a gestão e a replicação dos dados, assim como a gestão de máquinas. O Kudu Master Server (Kudu Master) tem como principal função direcionar as operações no cluster, armazenando os metadados das tabelas tais como: nome, tipo de colunas, a sua localização e a sua informação de estado. O Kudu Tablet Server é responsável por todas as operações relacionadas com a gestão de dados: armazenamento, acesso, codificação, compressão e replicação [43].

[1] https://grafana.com/
[2] https://www.tableau.com/

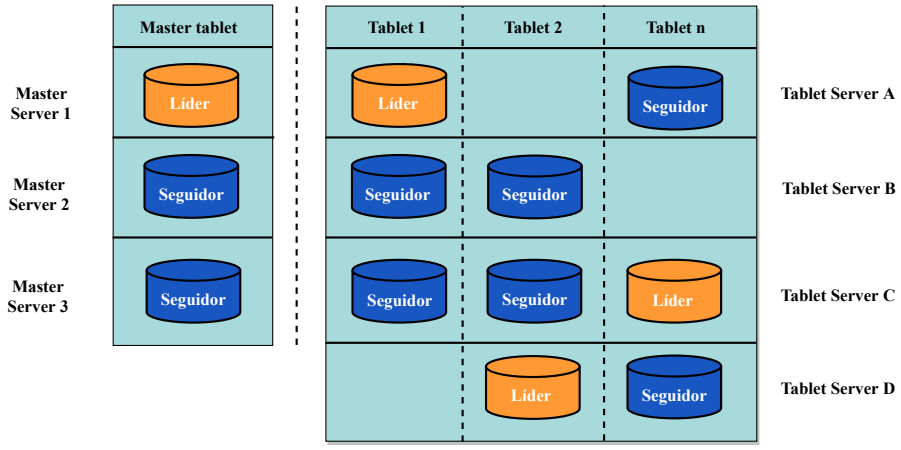


Figura 3.3: Arquitetura geral do Apache Kudu, adaptado de [43]

O Apache Kudu utiliza o algoritmo Raft [44] para implementar os mecanismos de armazenamento, gestão e replicação de dados. Por exemplo, a eleição do líder do Kudu Master pode ser dividida em 4 fases (Figura 3.4). Na primeira fase, se um seguidor dentro de um período de 500 a 1500 milissegundos não receber informação de um líder, este torna-se candidato a líder. Em seguida, envia pedidos para as outras máquinas a questionar se pode ser o próximo líder. Por fim, se obtiver a maioria dos votos, por exemplo 2 dos 3 Kudu Masters, este é eleito líder e todas as máquinas guardam essa informação num o Write Ahead Log (WAL) local [42, 43].

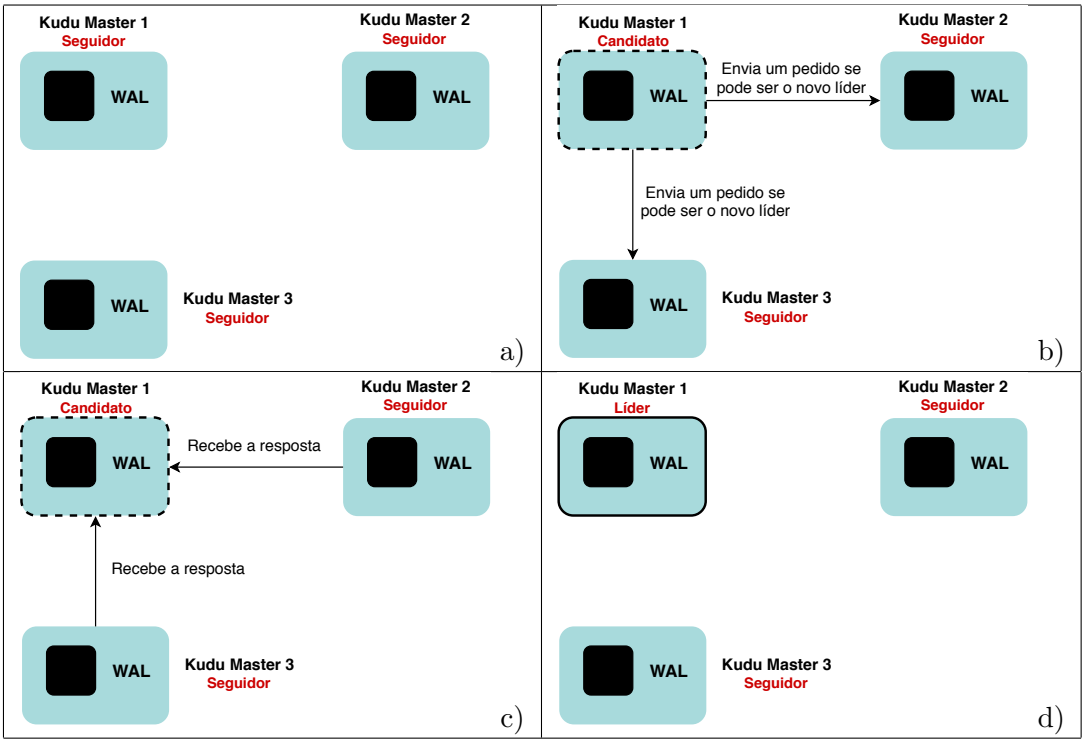


Figura 3.4: Exemplo da eleição do Kudu Master líder com o algoritmo Raft

Uma das questões importantes que se levanta é determinar o número ótimo de *tablets*. Esta questão não tem uma resposta exata porque depende da memória do cluster, do volume de dados e de outras características. No entanto, a fórmula apresentada abaixo permite calcular um valor aproximado, onde d representa o tamanho do conjunto de dados a ser inserido, k o espaço máximo de um Tablet Server, p a percentagem de *overhead* que se pretende acrescentar e r o fator de replicação [43].

$$t = \left(\frac{d}{(k * (1 - p))} \right) * r$$

3.3.2 Armazenamento

Como foi dito anteriormente, o Apache Kudu tem um armazenamento orientado às colunas, onde estas são armazenadas e comprimidas separadamente em *tablets*. Adicionalmente, é possível especificar os métodos de compressão e codificação para cada coluna da tabela assim como o método de particionamento da tabela.

Nas operações de inserção, o Apache Kudu começa por armazenar os dados em memória em *MemRowSets*. Posteriormente, quando o *MemRowSet* está cheio, este faz um "flush"dos dados para o disco, passando a ser designados por *DiskRowSet*. Por exemplo, ao inserir os valores "segredo"na coluna "pass"e o valor "mikas"na coluna "login"na tabela utilizadores, estes serão armazenados numa primeira fase em memória. Mais tarde, quando for executado o "flush"dos dados para o disco, o Kudu irá criar um *DiskRowSet*, separando as colunas, como ilustrado na Figura 3.5. Não obstante, numa próxima inserção na mesma tabela, em primeiro lugar o Kudu analisa todos os *DiskRowSet* e *MemRowSet*, de forma a certificar-se que a chave primária não existe. Se não existir, o processo repete-se e esta é inserida em primeiro lugar num *MemRowSet* para depois ser feito um "flush"para o *DiskRowSet* [43].

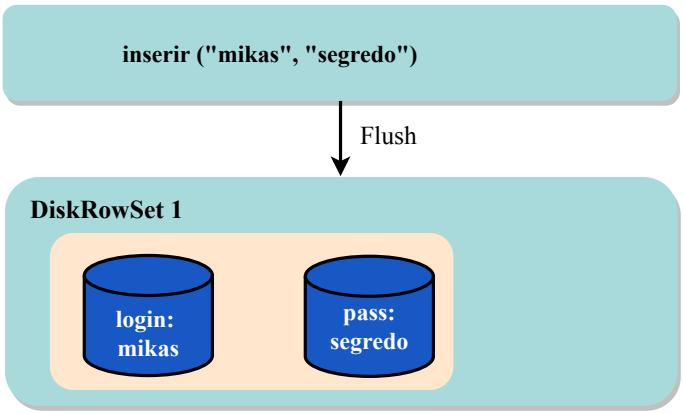


Figura 3.5: Armazenamento no Kudu, adaptado de [43]

Em suma, no armazenamento dos dados no Kudu, cada operação de escrita utiliza as seguintes etapas principais :

- Todas as operações são escritas no WAL de um tablet.
- Primeiro armazena os dados em memória, designado por MemRowSet.

- Quando o *MemRowSet* está cheio, executa um "flush"para o disco tornando-se *DiskRowSet*.

3.3.3 Partições

O Apache Kudu utiliza um método de particionamento para dividir os dados em conjuntos mais pequenos e independentes, possibilitando desta forma alto desempenho, disponibilidade dos dados, escalabilidade e balanceamento da carga. O Kudu tem três esquemas de particionamento: particionamento por Hash, particionamento por intervalo (Range) e particionamento combinado incluindo simultaneamente os dois esquemas anteriores. Contudo, não existe nenhum esquema ideal ou pré-definido para todos os casos e deve-se ter em atenção ao *hotspotting* que poderá existir. O *hotspotting* refere-se ao desbalanceamento da quantidade de consultas ou de inserções, em que estas são executadas maioritariamente no mesmo *tablet* [41, 43].

O particionamento por Range consiste em distribuir os dados por intervalos previamente definidos consoante um ou mais atributos referentes à chave primária atribuída. A Figura 3.6 apresenta um exemplo de uma tabela de sensores com os atributos ID (identificador) e data de inserção. Portanto, se a chave primária for considerada unicamente o atributo data de inserção não poderiam existir registos de sensores diferentes para a mesma data. Por este motivo, a chave primária deve ser composta pelos atributos ID e data de inserção. Neste caso, o particionamento dos *tablets* pode ser feito por identificador do sensor ou por intervalo de datas. Optando pelo particionamento por intervalo de datas, os dados mais recentes vão ser sempre inseridos no mesmo *tablet*, existindo o risco de *hotspotting* [43].

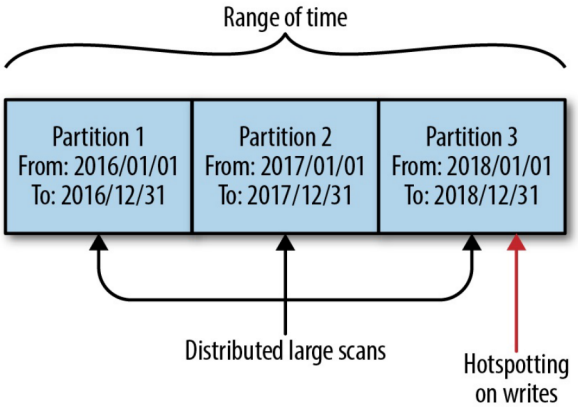


Figura 3.6: Particionamento por Range, retirado de [43]

No particionamento por Hash, os dados são inseridos e divididos uniformemente pelo número de *tablets*. Por exemplo, como se demonstra na Figura 3.7, os dados foram particionados por 3 *tablets* utilizando o atributo ID do sensor. Por esta razão, as inserções iriam ser distribuídas uniformemente pelos *tablets* existentes porque a cada ID do sensor é atribuído um Hash diferente. Contudo, quando é executada uma consulta sobre todos os valores de um sensor, os dados estarão no mesmo *tablet* existindo também o risco de *hotspotting* [43].

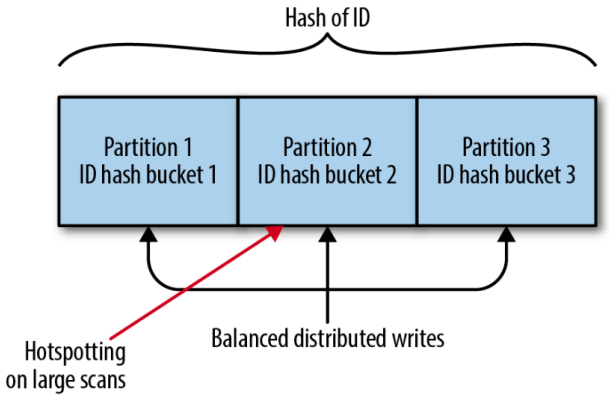


Figura 3.7: Particionamento em Hash, retirado de [43]

Com a combinação das duas estratégias de particionamento pode-se melhorar a performance tanto na inserção de dados como na sua consulta. Por exemplo, a Figura 3.8, mostra como é possível combinar o particionamento em Hash pelo ID do sensor e o particionamento por Range pelo atributo data de inserção. Desta forma, assumindo uma distribuição uniforme dos valores do atributo de ID do sensor e um número de eventos balanceado ao longo das duas estratégias, teríamos os dados balanceados pelos *tablets* [43].

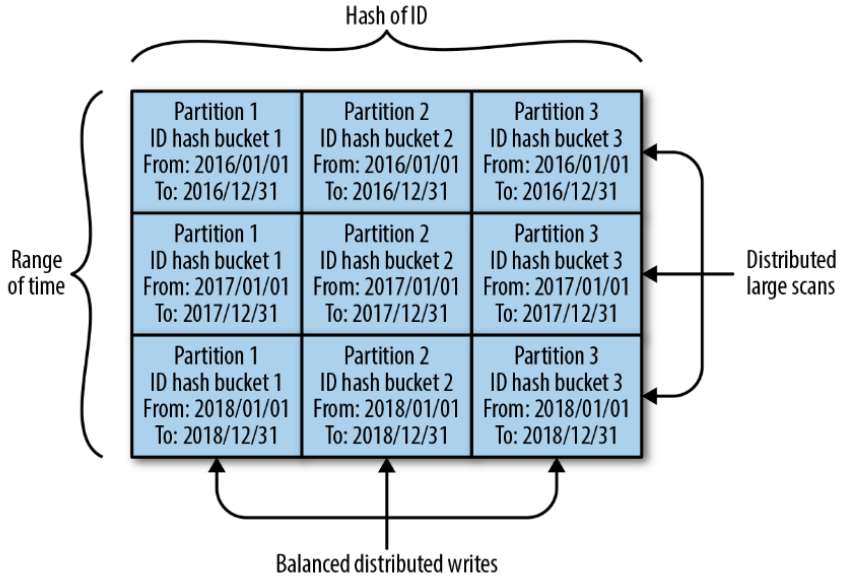


Figura 3.8: Particionamento Combinado, retirado de [43]

3.4 Estrutura de Dados

Para implementar uma TSDB com dados distribuídos em múltiplos sistemas computacionais, é necessário usar protocolos que forneçam um formato de serialização para tornar os

dados persistentes e um formato para a comunicação entre nós e com os programas clientes. Existem diversas soluções para este efeito, nomeadamente o Apache Avro e o Apache Parquet.

- **Apache Avro -** É formato *open-source* do ecossistema Hadoop para serialização de dados num formato binário compacto. A estrutura utilizada para os dados é o formato JSON (JavaScript Object Notation), que facilita a leitura e a interpretação dos dados e permite a sua representação em binário, tornando os dados mais compactos [45].
- **Apache Parquet -** É um formato de armazenamento de dados por colunas *open-source* do ecossistema Hadoop, que permite usar diferentes métodos de codificação e compressão de dados de forma eficiente [46].

3.5 Processamento dos dados

Existem várias tecnologias que permitem o processamento de grandes quantidades de dados de forma eficiente no contexto Big Data. Neste sentido, serão apresentadas o Apache Kafka e o Apache Spark.

- **Apache Kafka -** O Apache Kafka é um sistema distribuído para publicar e subscrever (*publish-subscribe*) mensagens em tempo real, desenvolvido inicialmente pelo LinkedIn. A arquitetura do Kafka, que pode ser visualizada na Figura 3.9, inclui produtores (*producers*) que têm como finalidade publicar mensagens em tópicos para um nó do Kafka, que vão ser processadas por consumidores (*consumers*). Um tópico representa uma categoria de mensagens que compartilham características semelhantes. As categorias podem ter várias partições ordenadas sequencialmente. Esta tecnologia é uma boa solução para a monitorização e processamento de fluxos de dados, fornecendo baixa latência e tolerância a falhas [47–49].

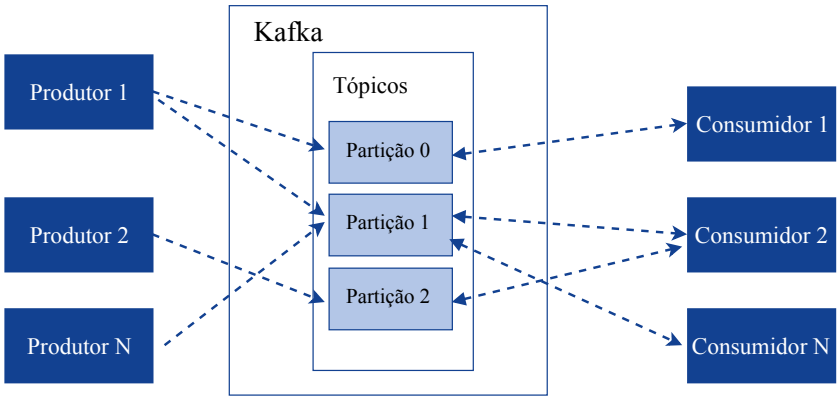


Figura 3.9: Arquitetura do Apache Kafka, adaptada de [49]

- **Apache Spark -** O Apache Spark é uma tecnologia de processamento paralelo, escalável e tolerante a falhas idealizada para processar de forma eficaz uma grande quantidade de dados. Inclui um *cluster manager* associado a vários nós que executam tarefas em paralelo. O Apache Spark tem 4 componentes, ilustradas na Figura 3.10: o Spark SQL para o processamento e consulta de dados estruturados, o Spark Streaming para

o processamento de fluxos de dados, o MLib para uso de *machine learning* e GraphX para o processamento de gráficos [48, 50].

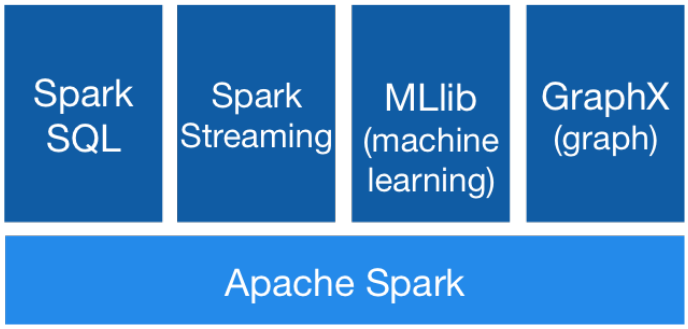


Figura 3.10: Componentes do Apache Spark, retirado de [50]

3.6 Consulta dos dados

O artigo [51] inspirou o aparecimento de várias tecnologias para consultar e manipular grandes volumes de dados, tais como o Apache Drill, o Apache Impala e o Apache Presto. Estas tecnologias têm a capacidade de consultar grandes conjuntos de dados armazenados em diferentes fontes (HDFS, Apache Kudu, entre outros) utilizando a linguagem SQL. A consulta dos dados pode ser efetuada usando:

- **Apache Hue -** É uma interface web desenvolvida pelo Cloudera [3] para integrar e interagir com o ecossistema Hadoop. Além disso, também fornece a possibilidade de execução de consultas utilizando a linguagem SQL, recorrendo ao armazenamento HDFS, e permite a análise dos dados através de gráficos [52].

- **Apache Impala -** É uma tecnologia distribuída *open source* que fornece um processamento paralelo massivo (MPP) tornando as consultas SQL simples e rápidas. Esta tecnologia foi desenvolvida para alto desempenho de consultas SQL distribuídas e baixa latência com dados armazenados em HDFS no ecossistema Hadoop. A arquitetura, que pode ser visualizada na Figura 3.11, inclui 3 serviços principais, o impala daemon (Impalad), o daemon Statestore (Statestored) e o daemon catatalog (Catalogd). O Impalad atua como o um coordenador das consultas, tendo como responsabilidade aceitar e gerir a execução das consultas no *cluster*. O Statestored é um serviço para publicar e subscrever os metadados do Impala, distribuindo a informação de todo o processo pelo *cluster*. O Catalogd armazena e agrega as informações dos metadados recolhidos de outros repositórios de metadados, como o Hive Metastore e o HDFS Namenode [53, 54].

[3] https://www.cloudera.com/

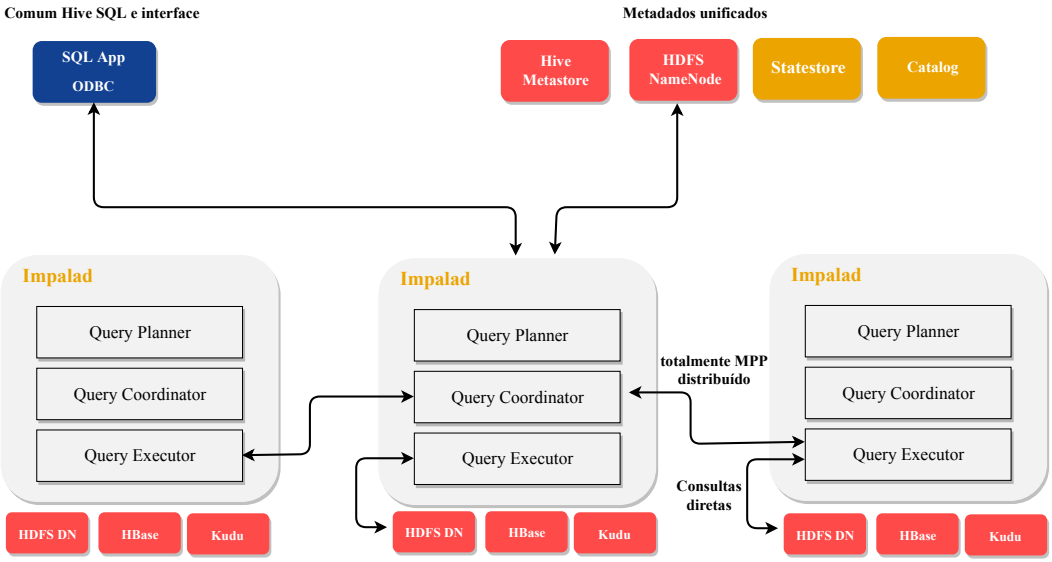


Figura 3.11: Arquitetura do Apache Impala, adaptado de [53]

- **Apache Presto -** É uma tecnologia distribuída *open source* que permite consultas analíticas interativas através do uso de SQL. O Apache Presto permite trabalhar com diversas fontes de dados tais como o HDFS, o Apache Kudu, o MongoDB, entre outras. A arquitetura inclui um Presto Coordinator e múltiplos Presto Workers. O Presto Coordinator tem como finalidade fazer a gestão dos diversos Presto Workers e distribuir as consultas pelos Presto Workers, isto é, começa por receber e processar a consulta, de seguida planeia a consulta e, por fim, distribui-a pelos Presto Workers disponíveis. Também armazena os seus metadados no Hive Metastore. O Presto Worker tem como função a execução das consultas. Para este efeito, deve comunicar com a fonte de dados respetiva, processar os dados e retornar o seu resultado ao cliente [55].

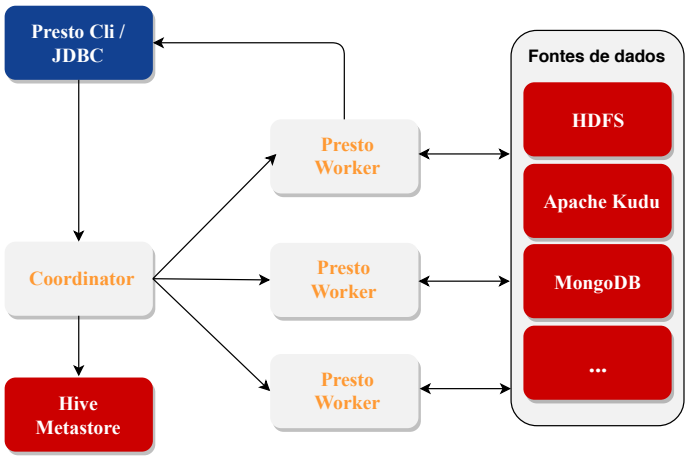


Figura 3.12: Arquitetura geral do Apache Presto, adaptado de [55]

- **Apache Drill -** O Apache Drill é uma tecnologia *open source* que permite a paralelização na execução das consultas SQL, tendo em vista conseguir baixa latência e suporte à consulta de grandes quantidades de dados com diferentes formatos. A arquitetura do Apache Drill, ilustrada na Figura 3.13, mostra vários nós, designados por *drillbits*, e o Zookeeper que é usado para manter a coordenação do cluster. O *drillbit* tem como finalidade aceitar os pedidos dos clientes e processar as consultas retornando o seu resultado. Porém, o *drillbit*, que recebe a consulta do cliente torna-se o Foreman da consulta e responsabiliza-se por distribuí-la para ser processada pelos outros nós. Permite o acesso a várias fontes de dados, tais como base de dados não relacionais (MongoDB e Hbase), sistemas de ficheiros (HDFS), entre outras fontes de dados [56].

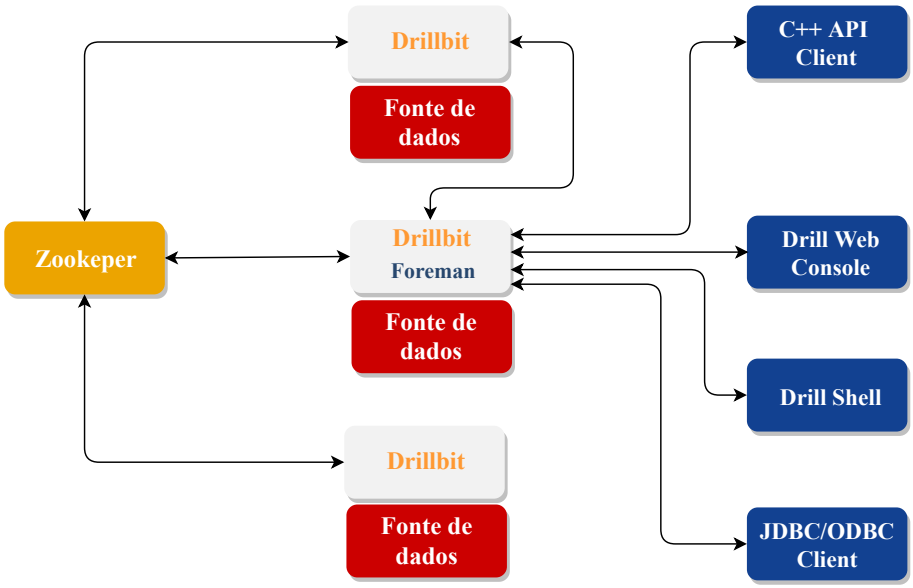


Figura 3.13: Arquitetura do Apache Drill, retirado de [56]

3.7 Síntese

Este capítulo apresenta uma análise de um conjunto de tecnologias que podem vir a integrar o NPM, em que cada uma delas pode ter um papel para completar o objetivo desta dissertação. Como o NPM recebe uma grande quantidade de dados provenientes de toda a rede e telecomunicações, que podem estar em diferentes formatos, revelou-se essencial compreender que ferramentas podem ser usadas para integrar e processar dados em diferentes formatos. Logo após, é feita uma apresentação sobre alguns sistemas de armazenamento orientados para os dados de séries temporais, concluindo com uma pesquisa sobre as tecnologias que executam consultas em linguagem SQL e usam diferentes fontes de dados.

Capítulo 4

Proposta de uma solução para o sistema atual

Este capítulo faz uma contextualização do problema, explicando os principais objetivos e desafios desta dissertação. Seguidamente, apresenta o sistema atual da Nokia, destacando o modelo de dados utilizado pelo NPM, a sua arquitetura implementada recorrendo à base de dados ORACLE e o seu mais recente projeto piloto para o armazenamento de dados de séries temporais. De seguida, apresenta uma comparação de funcionalidades de diversas TSDB e termina descrevendo o procedimento adotado para a escolha da solução implementada e testada neste trabalho.

4.1 Contextualização do Problema

O aumento exponencial da quantidade de dados provenientes das redes e de telecomunicações criou a necessidade de estudar novos mecanismos para processamento, armazenamento e visualização desses dados. A Nokia propôs, em conjunto com a Universidade de Aveiro, fazer um estudo sobre as TSDB existentes e emergentes com o intuito de substituir a tecnologia de armazenamento atual do NPM, de forma a otimizar a inserção e consultas de dados, bem como melhorar a performance da geração dos relatórios de rede. Não obstante, como esta dissertação se enquadra num estágio com a Nokia, algumas decisões foram tomadas consoante os requisitos funcionais e não funcionais fornecidos pela empresa.

A Figura 4.1 apresenta uma visão geral do fluxo de dados que se espera implementar neste trabalho. O fluxo começa com vários dispositivos de rede a enviar dados para serem processados e armazenados numa TSDB, finalizando com a execução de consultas à base de dados, de modo a gerar relatórios de rede e permitir fazer outras operações de análise dos dados.

Os principais desafios para o enquadramento de uma tecnologia TSDB no NPM, são:

- Adaptar-se ao seu modelo de dados e como estes são processados, assim como suportar consultas em SQL.
- Deve permitir a escalabilidade horizontal.
- Deve ser possível realizar consultas recorrendo ao JDBC.

- Otimizar as consultas e a geração dos relatórios de rede de forma a ser possível ter respostas em quase tempo real.

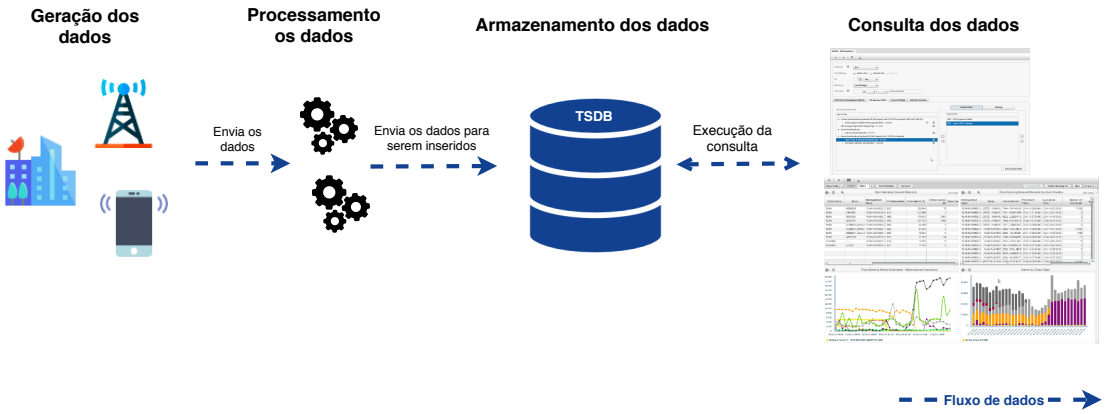


Figura 4.1: Visão geral do fluxo de dados com uma TSDB

4.2 Sistema atual da Nokia

Esta secção apresenta o modelo de dados utilizado atualmente pelo NPM, a sua arquitetura utilizando como base de dados o ORACLE, assim como o um projeto piloto que utiliza uma base de dados implementada sobre HDFS.

4.2.1 Modelo de dados

A Nokia utiliza o NPM para processar, normalizar e gerar relatórios com os dados provenientes de vários equipamentos e tecnologias de rede. A Figura 4.2 evidencia que existe uma estrutura definida para cada tecnologia, nomeadamente 2G, 3G, LTE, entre outras.

O NPM tem um modelo de dados genérico que segue uma orientação standard *3rd Generation Partnership Project* (3GPP), onde é definido o modelo de dados da topologia, o modelo de Performance Manager (PM) e o modelo de Reporting. No caso da tecnologia LTE, que foi usada no contexto desta dissertação, o modelo de dados é representado pela Figura 4.3.

O modelo de topologia envolve a definição de classes associadas a cada elemento de rede, bem como alguns atributos básicos, como por exemplo a designação da classe (PLMN (Public land mobile network), eNodeB, CELL) e a relação hierárquica entre esses elementos de rede. No caso da tecnologia LTE, e de acordo com a Figura 4.3, a topologia começa pela classe PLMN; cada classe é uma entidade lógica, que não mapeia para nenhum elemento de rede, mas está sempre presente em qualquer tecnologia para representar a rede do operador. As classes seguintes representam o elemento de rede LTE eNodeB, bem como as células associadas (CELL). Em seguida, o modelo de PM define os contadores básicos de rede e a forma como os mesmos se agrupam nas chamadas medidas de contadores. Podem existir medidas associadas a qualquer classe da tecnologia, como por exemplo, ao nível da classe CELL a Cell Load, Power e Quality, e ao nível da classe BTS o GTP protocol interface. Este modelo define os atributos associados às medidas e contadores, de forma a permitir que os mesmos sejam usados pelo modelo de Reporting, que deve incluir funções de agregação dos contadores por

tempo e por objeto, unidades do contador, níveis de agregação temporal e objeto da medida, entre outras. Por fim, o modelo de Reporting possibilita a definição de KPI, que são fórmulas implementadas sobre os contadores básicos providenciados pelo modelo de PM, bem como a definição de relatórios que apresentam esses KPI em diversos formatos (tabelas, gráficos e mapas).

Em síntese, para cada domínio de rede, o NPM, deve incluir dados não tratados (raw), onde estão todos os contadores enviados dos elementos de rede que estão disponíveis para os relatórios, e dados agregados (agg) que podem ter granularidades por tempo (hora, dia, mês, ano) e granularidades de objeto (PLMN, eNodeB e CELL).

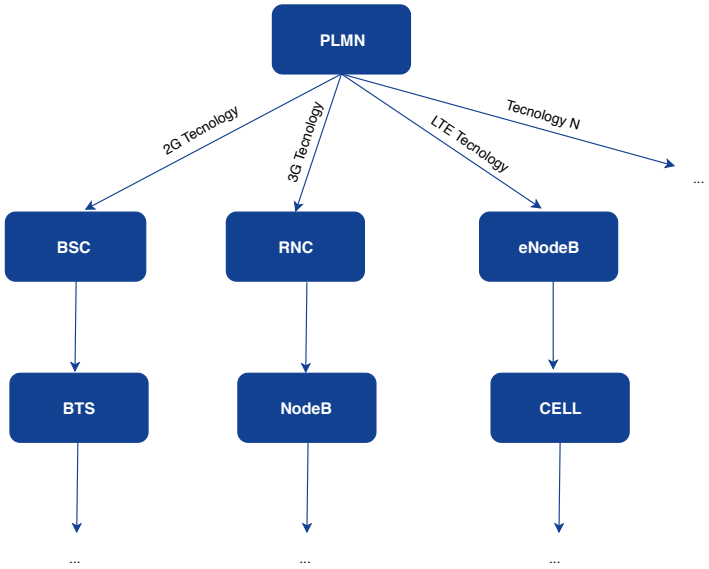


Figura 4.2: Organização hierárquica dis equipamentos de rede

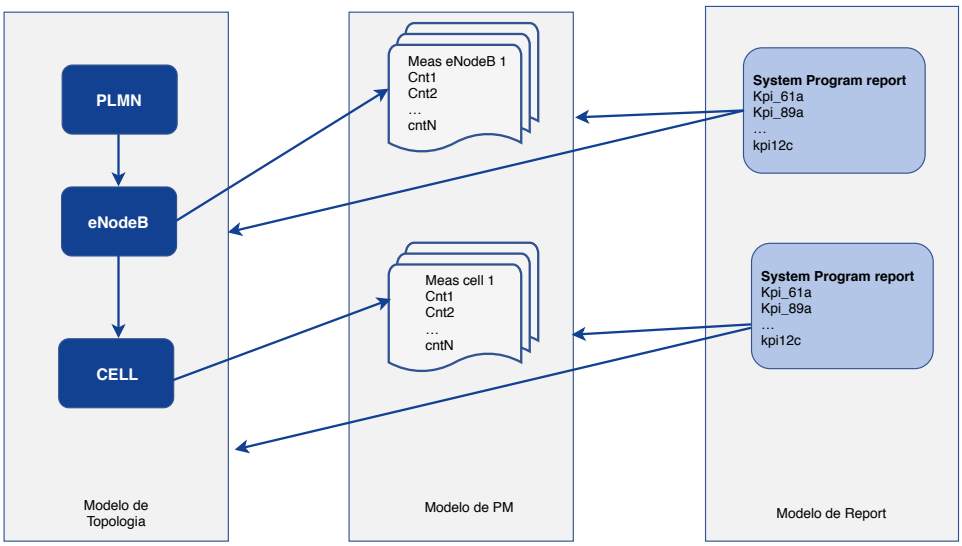


Figura 4.3: Modelo de dados genérico do NPM

4.2.2 Arquitetura ORACLE

O NPM é um sistema centralizado de gestão de performance de redes e telecomunicações, que tem como principal funcionalidade lidar com os dados provenientes dos diversos equipamentos usados em telecomunicações, de forma a conseguir extrair informação, tal como estatísticas de falhas e dados sobre as configurações dos equipamentos. Outro foco do NPM é fornecer dados históricos de desempenho de rede agregados por um período de tempo escolhido, nomeadamente hora, dia, mês ou ano, que também estão preparados para serem utilizados para o cálculo de KPI, agilizando a produção de relatórios de rede. Finalizando, este sistema permite visualizar a performance de rede em quase tempo real, atualizando os dados em intervalos de aproximadamente de 15 minutos.

A arquitetura do NPM está ilustrada na Figura 4.4 e consiste em quatro quatro fases principais, designadas por: Aquisição de dados, Mediador, Processamento e Visualização.

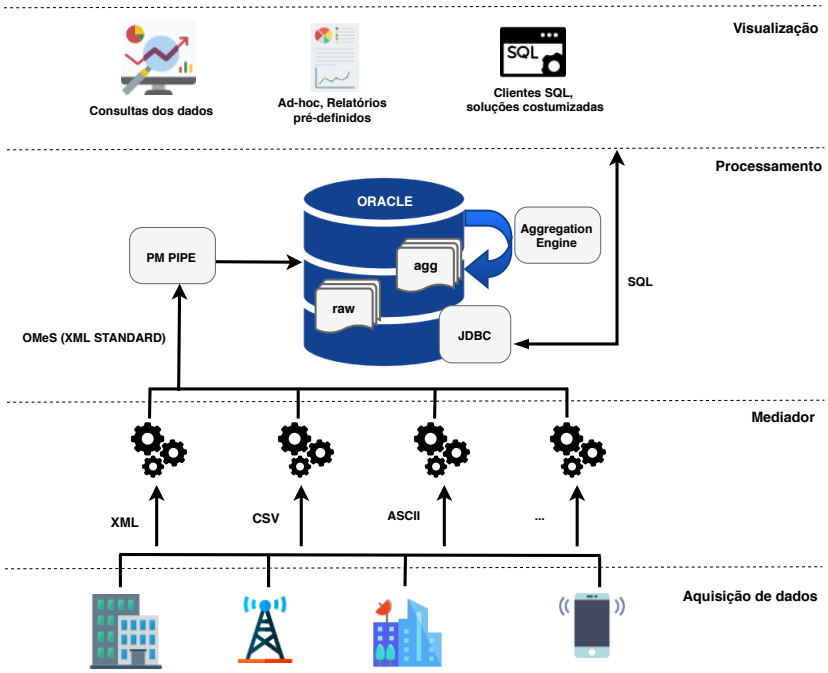


Figura 4.4: Arquitetura do NPM ORACLE

Na primeira fase, Aquisição de dados, é feita uma recolha dos dados nos múltiplos sistemas de rede, que são colocados num *data warehouse*, ou seja, um repositório de dados centralizado, podendo os primeiros estar em formatos diferentes, nomeadamente CSV, ASCII, XML, entre outras opções. Na segunda fase designada por Mediador, é criado um sistema ETL (Extract, transform, load), que tem como principal funcionalidade processar os ficheiros recolhidos em múltiplos formatos e colocá-los num ficheiro XML standard, denominado por Open Measurement Standard (OMeS). Na terceira fase, designada por Processamento, o PM PIPE tem como função carregar os ficheiros XML na base de dados ORACLE [1] diretamente para tabelas raw. Não obstante, é nesta fase que é realizado um processo de agregação de forma a que os dados sejam pré-calculados para um determinado período temporal associado a um elemento

[1] https://www.oracle.com/index.html

de rede, como por exemplo agregar por hora, dia, mês ou ano. Os elementos de rede podem ser RNC, BTS, entre outros. Por fim, na quarta fase o cliente final faz pedidos via JDBC à base de dados, resultando na geração e visualização dos relatórios de rede através de uma interface. Além disso, também é possível gerar relatórios com base em fórmulas KPI, exportar dados para Excel ou para CSV, agendar relatórios, filtrar os dados e executar pesquisas complexas, entre outras opções.

4.2.3 Arquitetura para séries temporais

A Nokia criou um projeto piloto para a alterar a arquitetura do NPM, substituindo o sistema de base de dados relacional (ORACLE), por um sistema de armazenamento HDFS recorrendo ao uso de ficheiros Parquet do ecossistema Hadoop. A razão para esta mudança prende-se com a possibilidade de utilizar *hardware* comum, fornecer escalabilidade horizontal para alto desempenho e otimizar a inserção e processamento de grandes volumes de dados. Complementarmente, foi utilizada a plataforma Cloudera [2], de modo a facilitar a instalação, manipulação e gestão das soluções do ecossistema Hadoop e projetos relacionados.

A Figura 4.5 mostra a arquitetura do NPM com a utilização do sistema de armazenamento HDFS que, tal como a arquitetura da secção 4.2.2, tem quatro fases: Aquisição de dados, Mediador, Processamento e Visualização. A única fase que difere é a fase de Processamento.

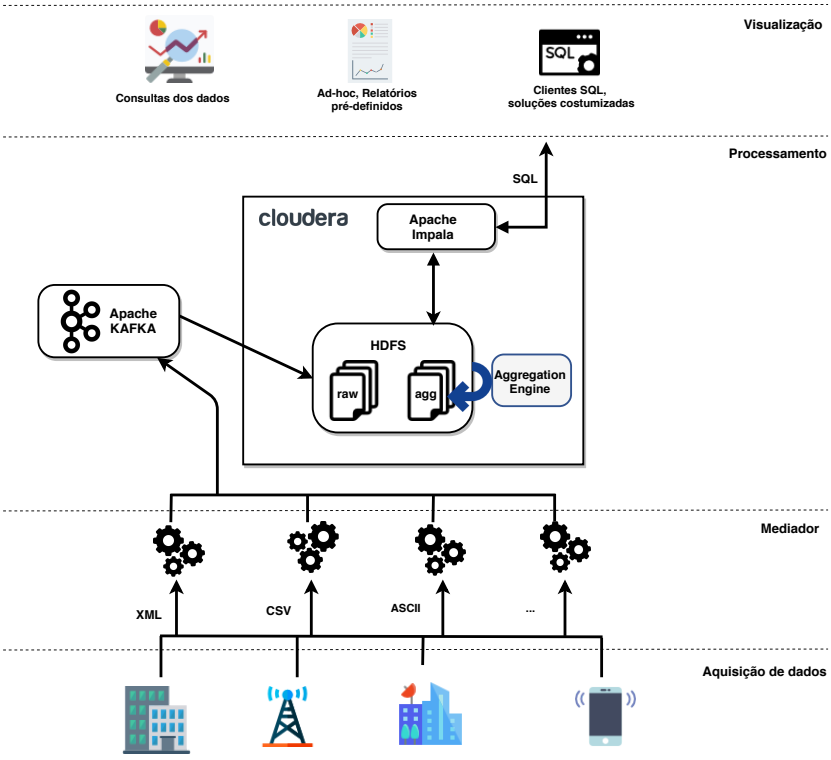


Figura 4.5: Arquitetura do NPM HDFS

A fase de processamento utiliza o Kafka para normalizar a entrada dos dados no sistema, retirando desta forma o formato OMeS apresentada na arquitetura da secção 4.2.2. Além

[2]https://www.cloudera.com/

disso, a utilização do Kafka trouxe a vantagem de permitir cenários de uso em tempo real, nomeadamente no cálculo de KPI. O Kafka trata de enviar os dados para o armazenamento HDFS. Adicionalmente, para a execução das consultas SQL e geração de relatórios de rede era obrigatória a utilização de JDBC e, por esta razão, foi utilizada a tecnologia Apache Impala para esse efeito. As principais razões para a utilização desta tecnologia foram a sua capacidade de integração com o Cloudera e com o ecossistema Hadoop, nomeadamente capaz de se integrar com o sistema de armazenamento HDFS bem como fornecer MPP das consultas em SQL.

Esta arquitetura será tomada como a arquitetura atual da Nokia, pois em contexto do Big Data, trouxe melhor desempenho, escalabilidade horizontal e menor latência.

4.3 Requisitos

Alinhados com a natureza dos dados processados e com o objetivo de avaliar a utilização de uma TSDB no NPM, foram definidos os seguintes requisitos para a auxiliar na seleção da tecnologia a usar neste trabalho:

- Incluir suporte à comunidade e preferencialmente suporte comercial.
- Ser capaz de interagir com o sistema atual da Nokia.
- Ser capaz de processar os dados recebidos através do Kafka.
- Escalabilidade horizontal.
- Alta disponibilidade, de forma a ser tolerante a falhas, permitindo a replicação dos dados pelo menos em 3 máquinas.
- Permitir o particionamento dos dados.
- Suportar JDBC e consultas SQL ou SQL-Like.
- Ser capaz de executar operações CRUD.
- O sistema deve conseguir fazer consultas utilizando uma média de 15 junções (*joins*).
- Deve ser capaz de otimizar as inserções e consultas dos dados de séries temporais.

A tecnologia proposta nesta dissertação deveria incluir suporte à comunidade, tendo como preferência esta ser incluída na sua solução suporte comercial. Também deveria ser capaz de interagir e integrar com todos os componentes da arquitetura atual da Nokia. Por este motivo, deveria estar apta para a integração com o Kafka assim como também de processar os dados recebidos de forma eficiente. Adicionalmente, esta tecnologia deve ser escalável horizontalmente de forma simples, para que consiga lidar com grandes quantidade de dados e ter a capacidade de agilizar a carga de trabalho no processamento e armazenamento dos dados. A TSDB deveria incluir mecanismos de tolerância a falhas, ou seja, estar preparada se um componente ficar inativo. Por esta razão, considera-se que a informação deve ser replicada no mínimo em 3 máquinas. Os dados devem ser particionados, de modo a obter um melhor desempenho na execução das consultas.

Além disso, a tecnologia selecionada deve permitir a otimização da inserção e consulta de dados, de forma a melhorar os tempos de resposta para a geração dos relatórios de rede. Conectividade à base de dados via JDBC e suportar a linguagem SQL ou SQL-like são requisitos obrigatórios. O sistema selecionado deve ser capaz de suportar consultas com 15 *joins*. Deve suportar operações CRUD , ou seja, ser capaz de inserir, consultar, atualizar e eliminar dados. Por outras palavras, deve permitir a inserção e consulta dos dados de rede, contudo quando se observam anomalias na rede deve ser possível atualizar os dados correspondentes ou removê-los se necessário.

4.4 Comparação das Tecnologias

A indústria empresarial tecnológica foi uma das grandes responsáveis por impulsionar o aparecimento das TSDB, devido à necessidade de obter performance na monitorização de sistemas através de sequências de dados temporais. As tecnologias consideradas neste estudo estão listadas nas tabelas 4.1 a 4.3, que descrevem as suas principais características e destacam a sua capacidade de escalabilidade e alta disponibilidade. Toda a informação inserida nestas tabelas resultou da análise de documentação online, artigos e testes [57, 58].

A tabela 4.1 permite observar que as TSDB são recentes e que as primeiras apareceram por volta do ano 2006. Após a "explosão"do Big Data, denotou-se um maior foco para a sua aparição . Adicionalmente, as tecnologias têm maioritariamente uma versão *open-source*.

A tabela 4.2 mostra para cada tecnologia qual o modelo de dados utilizado, o tipo de armazenamento, se suporta SQL ou SQL-like e se suporta JDBC. O modelo de dados pode ser chave/valor, orientado a colunas ou relacional. As base de dados podem utilizar um armazenamento interno e integrado na mesma base de dados sem nenhuma dependência de outra base de dados, um sistema de ficheiros distribuído (DFS) ou serem dependentes de uma base de dados.

Na tabela 4.3 é possível observar que a maior parte das TSDB são escaláveis horizontalmente. Contudo, o InfluxDB só o permite na versão comercial, o KairosDB só tem escalabilidade horizontal quando se utiliza o Cassandra e o OpenTSDB utiliza a configuração de multi-node do HBase.

Tabela 4.1: Características das TSDB: suporte e licenciamento

Tecnologia	Versão	OpenSource	Suporte Comercial	Licença	Lançamento
TimescaleDB	0.8.0	Sim	Não	Apache 2.0	2017
SiriDB	2.0.26	Sim	Não	MIT	2017
Apache Kudu	1.7.0	Sim	Sim[a]	Apache 2.0	2016
Riak TS	1.5.2	Sim	Não	Não	2015
Prometheus	2.1.0	Sim	Não	Apache 2.0	2015
InfluxDB	1.5.0	Sim	Sim	MIT	2013
KairosDB	1.2.0	Sim	Não	Apache 2.0	2013
Druid	0.11.0	Sim	Não	Apache 2.0	2012
OpenTSDB	2.3.0	Sim	Não	LGPLv2.1+	2011
Cassandra	3.11.1	Sim	Não	Apache 2.0	2008
Graphite	1.1.1	Sim	Não	Apache 2.0	2006

[a] Com a utilização do Cloudera

Tabela 4.2: Características das TSDB: modelo de dados, armazenamento e linguagem

Tecnologia	Modelo de Dados	Armazenamento	Suporte SQL	JDBC
TimescaleDB	Relacional	PostgreSQL	Sim	Sim
SiriDB	Orientado às colunas	Cassandra	Não	Não
Kudu	Orientado às colunas	custom[a]	Sim [b]	Sim [b]
Riak TS	Chave/valor	leveldb	Sim	Não
Prometheus	Chave/valor	custom[a]	Não	Não
InfluxDB	Orientado a colunas	custom[a]	Sim	Não
KairosDB	Orientado a colunas	Cassandra	Não	Não
Druid	Orientado às colunas	DFS	Experimental	Experimental
OpenTSDB	Orientado às colunas	Hbase	Não	Não
Cassandra	Orientado às colunas	custom[a]	Sim	Não
Graphite	?	Whisper	Não	Não

[a] Não têm nenhuma dependência com outra base de dados.
[b] Com a integração de sistemas de consultas SQL, tais como o Impala, o Drill e o Presto

Tabela 4.3: Características das TSDB: escalabilidade e alta disponibilidade

Tecnologia	Alta disponibilidade	Escalabilidade Horizontal
TimescaleDB	Manual	Não
SiriDB	Clustering	Sim
Kudu	Clustering	Sim
Riak TS	Clustering	Sim
Prometheus	Clustering	Sim
InfluxDB	Premium	Premium
KairosDB	Clustering	Sim
Druid	Clustering	Sim
OpenTSDB	Clustering	Hbase
Cassandra	Clustering	Sim
Graphite	Manual	Não

4.5 Seleção da Tecnologia

Após uma análise de toda a informação descrita tanto no estado de arte como na secção 4.4 sobre a comparação das tecnologias encontradas, iniciou-se o processo de seleção da tecnologia que viria a ser proposta para integrar a arquitetura do NPM, de modo a estudar a viabilidade de substituir o sistema de armazenamento atual. Além disso, para o efeito foram instalados alguns sistemas listados na secção anterior de modo a perceber o seu funcionamento e a veracidade da documentação, pois devido às base de dados serem relativamente recentes a sua documentação pode estar desatualizada ou o projeto estar abandonado.

O método de seleção considerou quatro requisitos principais que a Nokia estabeleceu como obrigatórios, que consequentemente foram definidos como os critérios de exclusão. Estes critérios são: suportar SQL, ter um JDBC driver para executar as consultas SQL, analisar o seu suporte comercial e de comunidade e, finalmente, permitir escalabilidade horizontal e

alta disponibilidade.

4.5.1 Suporte SQL/SQL-like

O primeiro requisito imposto é que a tecnologia deve suportar consultas através da linguagem SQL ou SQL-Like. Tendo como referência a Tabela 4.2, foram selecionadas as tecnologias TimescaleDB, RiakTS, InfluxDB, Druid, Apache Kudu, Cassandra.

Em seguida, foi escrito código de teste para determinar a capacidade da linguagem SQL suportar operações CRUD, junções e agregações, como pode ser visualizado na tabela 4.4. Tendo em conta estes requisitos, as tecnologias candidatas a seguirem para a próxima fase foram o TimescaleDB, o Apache Kudu, o InfluxDB e o RiakTS. Porém, foi detetado que o RiakTS não permite eliminar as tabelas após a sua criação e, por esta razão, esta tecnologia não seguiu para a próxima fase.

Tabela 4.4: Características das TSDB: suporte à linguagem SQL

Produto	Select	Insert	Update	Delete	Join	Aggregações
TimescaleDB	Sim	Sim	Sim	Sim	Sim	Sim
Riak TS	Sim	Sim	Sim	Sim	Sim	Sim
InfluxDB	Sim	Sim	Não	Sim	Não	Sim
Druid	Sim	Sim	Sim	Sim	Não	Sim
Apache Kudu	Sim	Sim	Sim	Sim	Sim	Sim
Cassandra	Sim	Sim	Sim	Sim	Não	Sim

4.5.2 JDBC driver

O segundo requisito era que a tecnologia suportasse JDBC, de forma a possibilitar o uso de código SQL em JAVA. Foram analisados o InfluxDB, o TimescaleDB e o Apache Kudu e, de acordo com os resultados apresentados na tabela 4.2, foi excluído o InfluxDB. As tecnologias que seguiram para a fase final foram o TimescaleDB e o Apache Kudu.

4.5.3 Suporte Comercial e de Comunidade

A seguir, analisou-se o suporte comercial e da comunidade das tecnologias TimescaleDB e Apache Kudu, tendo também em atenção a sua maturidade, ou seja, há quanto tempo a tecnologia existe no mercado (Tabela 4.1). Como estas são tecnologias *open-source* recentes, é necessário avaliar a capacidade de as empresas que as desenvolveram dar suporte a problemas ocorridos na sua implementação ou outro tipo de erros, assim como quais as plataformas que estas incluem para esse efeito. Neste caso, ambas utilizam uma plataforma Slack [3] para dar suporte à comunidade.

O TimescaleDB foi oficialmente lançado no ano de 2017, não inclui suporte comercial, no entanto a nível de suporte da comunidade e de atualizações existe uma atividade moderada [4], assim como também pode usufruir da comunidade do PostgreSQL. O Apache Kudu teve a sua primeira versão através da incubadora da Apache Software Foundation no ano de 2016, inclui

[3] https://slack.com/

[4] https://www.openhub.net/p/timescaledb

suporte comercial devido à utilização do Cloudera e relativamente ao suporte à comunidade e atualizações da tecnologia existe uma atividade alta [5]. Adicionalmente, o TimescaleDB ainda está na versão 0.80 sendo uma tecnologia em desenvolvimento, comparativamente ao Apache Kudu que está na versão 1.7.0 e que inclui uma versão de produção mais desenvolvida, pois está no mercado há mais um ano que o TimescaleDB.

Portanto, ambas tecnologias têm um suporte ativo na comunidade, tendo o Apache Kudu o benefício de incluir suporte comercial.

4.5.4 Escalabilidade Horizontal e Alta Disponibilidade

Finalmente, no quarto requisito, procedeu-se à análise das tecnologias TimescaleDB e Apache Kudu ao nível da escalabilidade horizontal e alta disponibilidade com base na Tabela 4.3.

Pretende-se alta disponibilidade, porque em algum momento pode ocorrer uma falha, tanto a nível de *hardware*, por exemplo, por erros no disco, ou por outro motivo e, é necessário garantir que o sistema está sempre disponível de forma a não perder dados. Enquanto que com o TimescaleDB é preciso proceder à sua implementação e configuração manualmente, o Apache Kudu utiliza o algoritmo Raft para garantir que metade do número de réplicas estejam disponíveis, tanto no caso do Master como no Tablet Server. Em segundo lugar, as tecnologias deveriam fornecer escalabilidade horizontal, porém só o TimescaleDB é que não suporta este requisito, pois só permite consultas em single-node.

Por fim, em relação à análise destes quatro requisitos, a única tecnologia que os respeita integralmente é o Apache Kudu. Esta tecnologia foi proposta como a solução a adotar nesta dissertação e foi implementada num ambiente de testes de forma a compará-la com o sistema atual da Nokia.

4.6 Síntese

Este capítulo descreve como foi feita a ligação entre os conceitos e tecnologias analisadas na escolha de uma TSDB, consoante o contexto da Nokia. Apresenta o sistema atual da Nokia, explicando o seu modelo de dados e a sua arquitetura. De seguida, de forma a possibilitar uma comparação uniforme das tecnologias, foi realizado um levantamento de requisitos que a tecnologia que iria ser proposta deveria incluir. Finalmente, é detalhado o processo da seleção da tecnologia, tendo-se concluído que a tecnologia que melhor se adapta ao NPM é o Apache Kudu.

Após a conclusão deste capítulo, segue-se a apresentação da arquitetura e implementação da TSDB selecionada no contexto da NPM.

[5] https://www.openhub.net/p/apache-kudu

Capítulo 5

Arquitetura e implementação

A seguir à seleção da tecnologia procedeu-se à definição de uma proposta para a arquitetura do sistema e à sua implementação.

Este capítulo começa por apresentar a proposta de arquitetura do novo sistema baseado em Apache Kudu e explica os passos realizados para a sua implementação, dando ênfase ao procedimento de escolha da chave primária, ao particionamento das tabelas e às configurações mais importantes. Para terminar, apresenta as configurações necessárias para a comunicação com as aplicações externas que fazem as consultas SQL e o fluxo de dados para os processos de inserção e de consulta de dados.

5.1 Arquitetura do sistema

A proposta da nova arquitetura tem por base a arquitetura atual, descrita na secção 4.2.3, e inclui os seguintes componentes: Aquisição de dados, Mediador, Processamento e Visualização. Porém a única fase que sofreu alterações foi a do Processamento.

A fase de Processamento continua a utilizar o Apache Kafka para normalizar os dados recebidos do Mediador e para calcular os KPI em tempo real, finalizando com o envio dos dados para o sistema de armazenamento selecionado, o Apache Kudu. Posteriormente, foi tomada a decisão de implementar o Apache Kudu recorrendo à plataforma do Cloudera, porque esta já era utilizada pela Nokia e porque esta facilitaria a implementação do ambiente de testes e realização das configurações necessárias. Complementarmente, para o Apache Kudu cumprir dois dos requisitos obrigatórios, nomeadamente permitir consultas SQL e fornecer JDBC, foi necessário incluir na arquitetura proposta um sistema de consultas SQL, por exemplo, o Apache Impala, o Apache Presto ou o Apache Drill. Selecionou-se o Apache Impala, porque este está incluído nas ofertas do Cloudera, e tem a vantagem de ter uma implementação acessível, comunicação com o Apache Kudu e fornecer consultas MPP.

5.2 Propriedades do Apache Kudu

O Apache Kudu fornece um conjunto de propriedades que são necessárias definir para a sua implementação, incluindo o procedimento de escolha de uma chave primária, o particionamento das tabelas e, por fim, as principais configurações que precisaram de ser feitas.

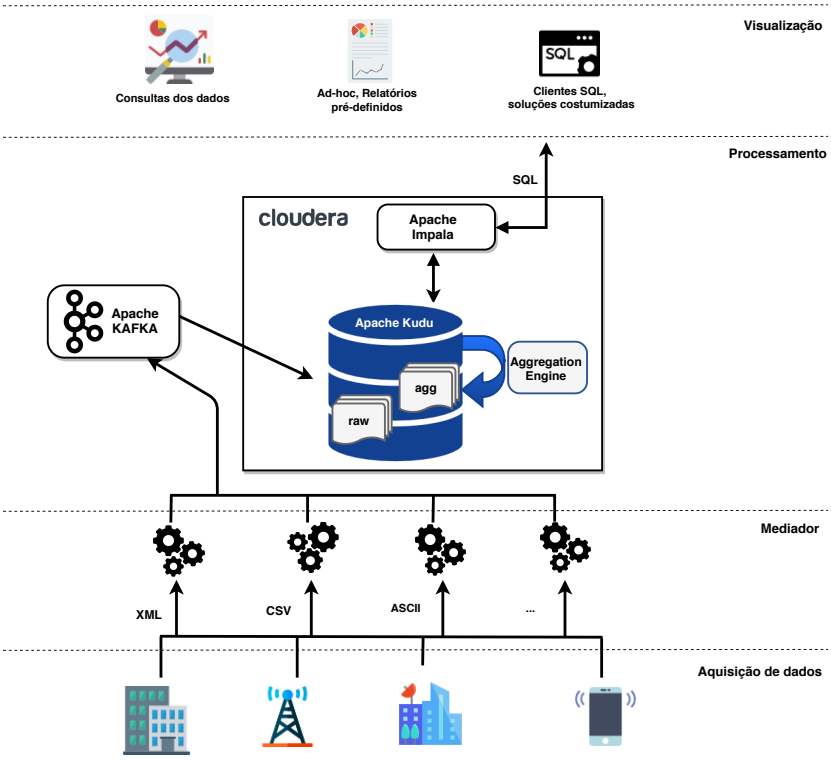


Figura 5.1: Arquitetura proposta com o Apache Kudu adaptada ao NPM

5.2.1 Chave Primária

O Apache Kudu obriga a definir uma chave primária que é usada para o particionamento de cada tabela. Atendendo ao modelo de dados descrito na secção 4.2.1, foi decidido usar como chave primária o par de atributos *timestamp* e o identificador do objeto de rede. Um exemplo de uma chave primária usando objetos de rede LTE seria o par de atributos "PLMN/eNodeB"e um valor temporal "2018-08-09 22:00:00". Esta escolha está de acordo com as recomendações encontradas na literatura, tal como descrito na secção 2.3.2.

5.2.2 Particionamento das tabelas

O Apache Kudu permite o particionamento das tabelas em *tablets*, que podem ser distribuídas em vários *Tablets Servers*. Os dados são sempre replicados em pelo menos 3 *Tablets Servers*. Em conformidade com a Nokia, a escolha da aplicação de uma estratégia de particionamento utilizou como método o tamanho de cada *tablet*, o total número de linhas inseridas e o número de colunas de cada tabela. Para este método foi realizado um teste usando dados recolhidos durante um período de 49 horas, sem qualquer tipo de particionamento das tabelas, de forma a compreender o relacionamento entre o tamanho de cada *tablet*, o número de linhas inseridas e o número de colunas. A tabela 5.1 mostra os resultados obtidos considerando 3 grupos: tabelas pequenas, tabelas grandes e tabelas médias, ou seja, os outros casos possíveis.

Inicialmente, as tabelas são divididas em dois grupos, onde o número de colunas é superior ou igual a 100 e o número de colunas é menor que 100. Uma das restrições desta tecnologia é apenas permitir um máximo de 300 colunas. Posteriormente, analisou-se o número total

de linhas inseridas, observando-se que os valores aproximados mais comuns foram 15000 linhas e 35000 linhas. Além disso, considerou-se também o tamanho de cada *tablet* usando como referência o valor de 2GB, que é o tamanho máximo recomendado para um *tablet* sem particionamento, de acordo com a documentação do Apache Kudu e em consenso com a Nokia.

A seguir, as tabelas médias são analisadas de forma mais detalhada, tal como demonstrado na tabela 5.2. Refira-se que os casos não contemplados correspondem a situações que não se verificaram no conjunto de dados usados para teste. A tabela mostra que as tabelas intermédias têm aproximadamente 35000 linhas, mas diferem no número de colunas, identificando-se dois grupos: médias_1 e médias_2. As tabelas médias_1 têm 20 colunas ou menos e têm menos de 2 GB e as tabelas consideradas médias_2 têm entre 20 a 100 colunas resultando em um tamanho superior a 2 GB.

Concluindo, as tabelas consideradas pequenas e médias_1 têm um tamanho inferior a 2GB e, por isso, vão ser mantidas numa única partição. Para as tabelas consideradas médias_2 e grandes, vão ser testadas várias estratégias de partição de modo a analisar qual a estratégia de particionamento mais adequada ao *workload* proposto.

Tabela 5.1: Definição do particionamento das tabelas

Tabela	Número de colunas	Total linhas inseridas/min	Tamanho do *tablet*
pequena	< 100	≈15 000	$< 2GB$
grande	≥ 100	≈35 000	$\geq 2GB$
Os outros casos possíveis			

Tabela 5.2: Definição do particionamento das tabelas médias

Tabela	Número de colunas	Total linhas inseridas/min	Tamanho do *tablet*
médias_1	≤ 20	≈35 000	$< 2GB$
médias_2	> 20 a < 100	≈35 000	$\geq 2GB$

5.2.3 Configurações Necessárias

Existem algumas configurações relevantes a considerar de forma a obter um melhor desempenho. As configurações mais importantes para o Kudu Master e Kudu Tablet Server são demonstradas na tabela 5.4 e na tabela 5.3.

As configurações do Kudu Master devem incluir uma referência à *flag* master_address que tem como objetivo que o Kudu Master reconheça todos os Kudu Master existentes, através do seu endereço IP ou do *hostname*. Também é necessário definir os diretórios onde os dados são armazenados (fs_data_dirs), os ficheiros de WAL (fs_wal_dir) e os ficheiros de log (log_dir). A *flag* maintenance_manager_numb_threads define o número de threads para a execução de tarefas em paralelo. Esta *flag* pode melhorar o desempenho das consultas e a rapidez de execução das operações de *flush* e compressão de dados.

As configurações mais relevantes do Kudu Tablet Server são a *flag* tserver_master_addrs, que define o endereço IP ou do *hostname* dos Kudu Master com os quais deve comunicar. Também é preciso definir os diretórios onde os dados são armazenados (fs_data_dirs), os ficheiros de WAL (fs_wal_dir) e os ficheiros de log (log_dir). A *flag* memory_limit_hard_bytes define

o tamanho máximo de memória a alocar para cada Tablet Server e block_cache_capacity_mb define o tamanho máximo da cache. A *flag* mainteance_manager_numb_threads define o número de threads para a execução de tarefas em paralelo.

Tabela 5.3: Propriedades de configuração Kudu Tablet Server

Flags	Definição
tserver_master_addrs	IP ou *hostname* de todos os Kudu Masters, aos quais o Kudu Tablet Server deve conetar-se
fs_data_dirs	Diretório onde são armazenados os dados
fs_wal_dir	Diretório onde são armazenados os ficheiros de WAL
log_dir	Diretório para os ficheiros de logs
memory_limit_hard_bytes	A quantidade máxima de memória alocada utilizada
block_cache_capacity_mb	A quantidade de memória alocada para cache
mainteance_manager_numb_threads	Número de threads dedicadas para as operações de *flush* e compressão

Tabela 5.4: Propriedades de configuração Kudu Master

Flags	Definição
master_addresses	IP ou *hostname* de todos os Kudu Masters
fs_data_dirs	Diretório onde são armazenados os dados
fs_wal_dir	Diretório onde são armazenados os ficheiros de WAL
log_dir	Diretório para os ficheiros de logs
mainteance_manager_numb_threads	Número de threads dedicadas para as operações de *flush* e compressão

5.3 Implementação do Apache Kudu

Uma vez definida a arquitetura do sistema e as configurações principais do Apache Kudu, o passo seguinte é proceder-se à implementação da tecnologia de forma a permitir a sua integração num ambiente de testes.

Neste sentido, esta secção apresenta os passos principais para implementar o processo de inserção, agregação e consulta dos dados, bem como as configurações do Apache Impala e do Apache Presto. Como o processo de agregação dos dados consiste na execução de uma consulta por um período temporal pretendido (por exemplo por hora, diário, entre outros), seguido de uma inserção de dados recorrendo à utilização de um JDBC, a sua explicação será integrada no processo de consultas.

5.3.1 Processo de inserção de dados

A primeira fase para a integração do ambiente de testes é perceber como proceder à inserção dos dados, e simular o fluxo de dados desde a sua geração até à inserção na base de dados. Por esta razão, a implementação deste processo é dividida em duas etapas, o

processamento de dados e o armazenamento de dados, como pode ser visualizado na Figura 5.2.

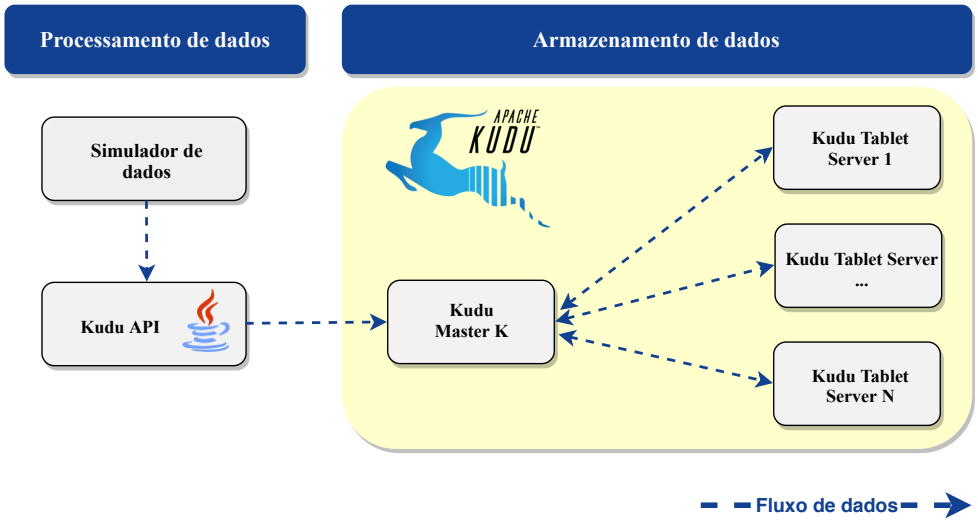


Figura 5.2: Fluxo de dados para o processo de inserção

O processamento dos dados inclui dois componentes: o simulador de dados e o Kudu API. Estes têm o objetivo de gerar, tratar e preparar os dados, finalizando com o seu envio para a base de dados. A segunda etapa corresponde ao armazenamento dos dados na tecnologia Apache Kudu. Isto envolve a comunicação entre o Kudu Master K, em que K identifica uma de 3 a 5 máquinas, e os N Kudu Tablets Servers. É necessário que existam no mínimo 3 máquinas. A comunicação e distribuição dos dados é descrita na secção 3.3.2.

A utilização do componente do **simulador de dados** deve-se a questões de privacidade, pois não é possível usar os dados dos clientes da Nokia. O simulador de dados usa a tecnologia LTE como referência e permite controlar o *workload*, ou seja, a quantidade de contadores de rede gerados. Cada contador tem uma data de inserção, o nome do objeto de rede, entre outras descrições e métricas. Os dados simulados são organizados em diferentes diretórios. Cada diretório representa uma tabela da base de dados e tem um ou mais ficheiros no formato Apache Parquet [1]. Os dados foram armazenados em ficheiros Parquet porque este formato permite a compressão e codificação dos dados de forma eficiente.

Finalmente, o componente do **Kudu API** é um programa escrito em JAVA que tem como principal função ler os diretórios e analisar os ficheiros em Parquet de forma a tratar os dados para serem inseridos no Apache Kudu através da sua API. A Figura 5.3 apresenta o *workflow* deste componente, começando pela leitura de todos os diretórios e dos ficheiros nesses diretórios. Em seguida, é extraída e tratada a informação sobre os contadores de rede, onde esta é colocada em *batch* temporariamente antes de ser inserida no Apache Kudu. Contudo, quando são atingidos 500 registos de contadores de rede, estes são inseridos diretamente no Apache Kudu.

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

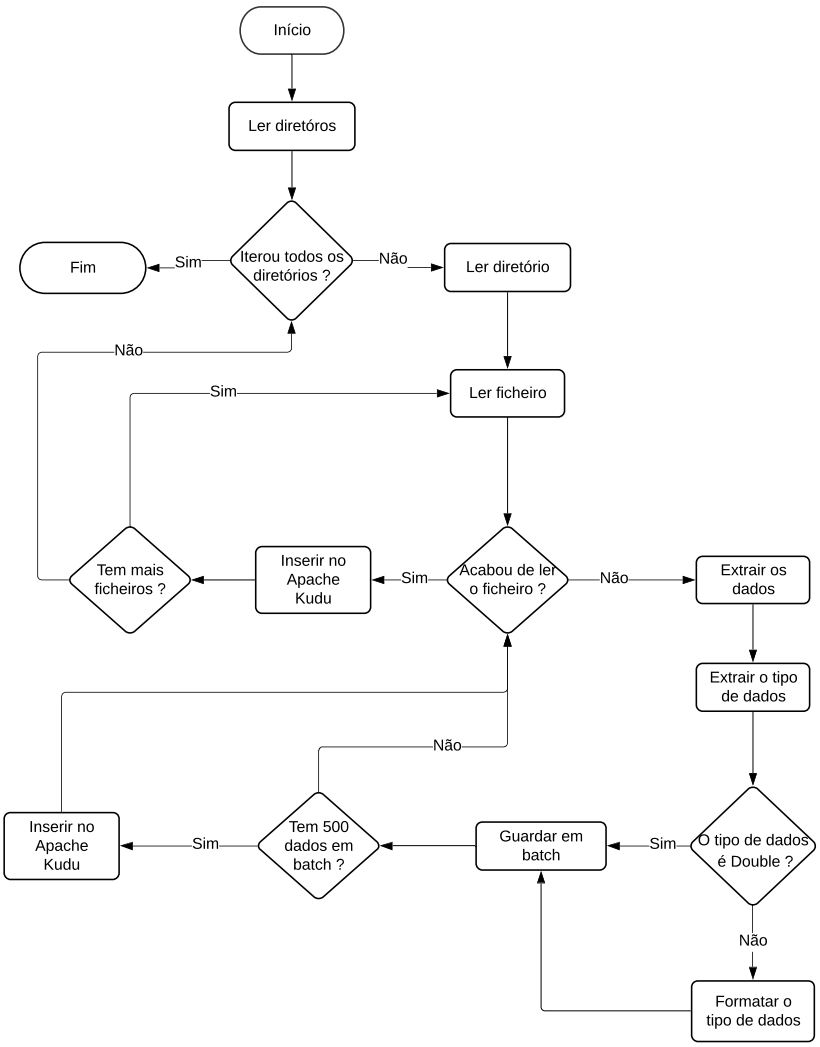


Figura 5.3: *Workflow* do componente do Kudu API

5.3.2 Processo de consulta dos dados

Este processo consiste em simular as consultas realizadas por JDBC entre o Apache Kudu e o Apache Impala. A implementação deste processo é dividida em duas etapas, o processamento de dados e armazenamento de dados, como pode ser observado na Figura 5.4.

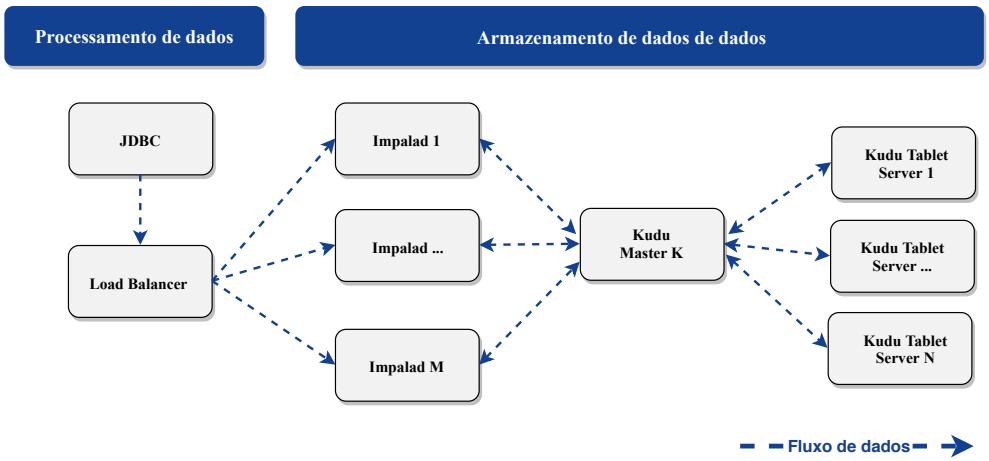


Figura 5.4: Fluxo de dados para o processo de consulta

O processamento de dados tem um componente designado JDBC e o componente Load Balancer. Estes têm como função enviar os pedidos de consulta para a base de dados e fazer o balanceamento da carga usando M máquinas Impalad, em que M deve incluir o mínimo de 3 máquinas. A utilização das máquinas Impalad justifica-se pela razão de ser onde se aplica o balanceamento da carga e alta disponibilidade. As máquinas statestored e catalogd não são replicadas porque não são suscetíveis de resultar em perda de dados. Caso essas máquinas fiquem indisponíveis devido a uma interrupção numa determinada máquina, é possível interromper o serviço Impala, excluir as funções Impala StateStore e Impala Catalog Server, adicionar as funções numa máquina diferente e reiniciar o serviço Impala [59]. A segunda etapa consiste na comunicação entre as máquinas Impalad e o Kudu Master, que por sua vez comunica com os Kudu Tablet Server.

O componente **JDBC** controla a execução das consultas dos dados. Este componente tem como objetivo simular consultas, recorrendo ao JDBC do Apache Impala, passando pelo componente Load Balancer. Para a execução de testes mais complexos, como o de agregação dos dados e a simulação de relatórios, foi utilizado o projeto em uso atualmente pela Nokia porque já tem os scripts SQL preparados para a comunicação com o Apache Impala.

Por fim, foi implementado o componente **Load Balancer** seguindo as recomendações que o Cloudera disponibiliza na sua documentação para o balanceamento da carga no Apache Impala. É necessário instalar o HAproxy e configurar um ficheiro para especificar as máquinas Impalad existentes [2], tal como exemplificado na Figura 5.5. Este componente tem como benefício a difusão dos pedidos de consulta para um Impalad disponível, evitando o sobrecarregamento ou indisponibilidade do Impalad e de utilizar um Impalad em específico. Além disso, para cada consulta, existe uma máquina Impalad que fica coordenadora, que potencialmente requer mais memória e CPU (Central Processing Unit) do que as outras máquinas que processam a consulta. O servidor proxy distribui as consultas para que cada conexão use uma máquina coordenadora diferente. O componente Load Balancer permite que as máquinas Impalad compartilhem esse trabalho adicional, em vez de concentrá-lo numa única máquina.

[2]https://www.cloudera.com/documentation/enterprise/5-14-x/topics/impala_proxy.html

```
global
    log         127.0.0.1 local0
    log         127.0.0.1 local1 notice
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon

#---------------------------------------------------------------------
# Os valores de tempo limite devem ser configurados dependendo do
# cluster e do tempo previsto para a execução das consultas.
#---------------------------------------------------------------------

defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    maxconn                 3000
    timeout connect 5000
    timeout client 3600s
    timeout server 3600s

# Configuração para Impala. O Impala client connecta-se ao load_balancer_host:25003.
# HAProxy balanceia as connções entre a lista definada em baixo

listen impala :25003
    mode tcp
    option tcplog
    balance leastconn

    server symbolic_name_1 impala-host-1.example.com:21000 check
    server symbolic_name_2 impala-host-2.example.com:21000 check
    server symbolic_name_3 impala-host-3.example.com:21000 check
    server symbolic_name_4 impala-host-4.example.com:21000 check

 # Setup para Hue ou para outras aplicações que utilizem JDBC.
 # A aplicação conecta-se ao load_balancer_host: 21051 e HAProxy balanceia #as conceções pelos Impalad

listen impalajdbc :21051
    mode tcp
    option tcplog
    balance source
    server symbolic_name_5 impala-host-1.example.com:21050 check
    server symbolic_name_6 impala-host-2.example.com:21050 check
    server symbolic_name_7 impala-host-3.example.com:21050 check
    server symbolic_name_8 impala-host-4.example.com:21050 check
```

Figura 5.5: Exemplo do ficheiro de configuração do componente Load Balancer, retirado de [59]

5.3.3 Configurações com o Apache Impala

A configuração do Apache Impala foi feita através do uso da plataforma Cloudera e, por este motivo, não foi preciso fazer muitas alterações nas suas configurações para comunicar e integrar com o Apache Kudu.

As configurações utilizadas podem ser visualizadas na Tabela 5.5. A primeira é a *flag* Kudu Service que tem como objetivo ativar a comunicação entre o Apache Impala e o Apache Kudu. Também é necessário definir os diretórios para o armazenamento dos dados (scratch_dirs) e o diretório para o ficheiro de logs (log_dir), finalizando com a definição da quantidade máxima de memória que é possível utilizar (mem_limit).

Tabela 5.5: Propriedades de configuração do Apache Impala

Flags	Definição
Kudu Service	Nome do serviço Kudu que o Apache Impala depende
scratch_dirs	Diretórios em que os Impalad armazenam os dados
log_dir	Diretório para os ficheiros de logs
mem_limit	A quantidade máxima de memória alocada utilizada

5.3.4 Configurações com o Apache Presto

O Apache Presto requer a instalação dos componentes Presto Coordinator e Presto Workers. Para configurar a comunicação com o Apache Kudu é necessário criar um ficheiro de configuração tal como exemplificado na Figura 5.6, indicando o hostname dos Kudu Masters [60].

```
1  ## Nome do conector
2  connector.name=kudu
3
4  ## Lista dos IP ou hostname de todos os Kudu Masters
5  kudu.client.master-addresses=slabnode1619,slabnode1620,slabnode1621
6
7  ## Kudu does not support schemas, but the connector can emulate them optionally.
8  ## By default, this feature is disabled, and all tables belong to the default schema.
9  ## For more details see connector documentation.
10 #kudu.schema-emulation.enabled=false
11
12 ## Prefix to use for schema emulation (only relevant if `kudu.schema-emulation.enabled=true`)
13 ## The standard prefix is `presto::`. Empty prefix is also supported.
14 ## For more details see connector documentation.
15 #kudu.schema-emulation.prefix=
16
17 ########################
18 ### Advanced Kudu Java client configuration
19 ########################
20
21 ## Default timeout used for administrative operations (e.g. createTable, deleteTable, etc.)
22 # kudu.client.defaultAdminOperationTimeout = 30s
23
24 ## Default timeout used for user operations
25 # kudu.client.defaultOperationTimeout = 30s
26
27 ## Default timeout to use when waiting on data from a socket
28 #kudu.client.defaultSocketReadTimeout = 10s
29
30 ## Disable Kudu client's collection of statistics.
31 #kudu.client.disableStatistics = false
```

Figura 5.6: Exemplo do ficheiro do Conetor do Kudu, adaptado de [60]

5.4 Síntese

Este capítulo apresenta os detalhes da arquitetura e implementação dos processos para utilização do Apache Kudu no NPM. Foi escolhida a tecnologia Apache Impala para a implementação de consultas em SQL. Descrevem-se as propriedades do Apache Kudu mais relevantes, tais como a definição da chave primária, a explicação da melhor estratégia de particionamento das tabelas, e por fim, as configurações necessárias. A seguir apresentam-se os processos de inserção e consulta de dados, e as ferramentas que foram usadas para acesso à base de dados usando SQL.

O Apache Kudu permite definir várias opções de configuração que importa avaliar no

sentido de definir quais as melhores estratégias a adotar no NPM. Para isso, é necessário implementar um ambiente de testes e uma metodologia de avaliação tal como se descreve no capítulo a seguir.

Capítulo 6

Testes e Resultados

Este capítulo começa por apresentar o ambiente de testes, descrevendo a infraestrutura utilizada ao nível do *hardware* e ao nível do *software*, e as métricas de avaliação usadas. De seguida, descreve as estratégias de avaliação utilizadas neste trabalho, de forma a planear como irá proceder-se à avaliação global do sistema. Termina com a apresentação e a discussão dos resultados dos testes. Mediante os resultados obtidos será possível validar este trabalho consoante os objetivos delineados inicialmente pela Nokia.

6.1 Ambiente de testes

6.1.1 Infraestrutura

A infraestrutura utilizada para o desenvolvimento do ambiente de testes é constituída por 8 máquinas, sendo que uma delas é unicamente uma máquina cliente utilizada para o lançamento dos programas. Foi utilizada a plataforma Cloudera para instalação e gestão do *software* a usar em cada máquina. Primeiro foram instalados os componentes do ecossistema Hadoop, nomeadamente 6 DataNodes e 2 NameNodes. Adicionalmente, foram instalados 3 JournalNodes e 2 Failover Controller para ter um sistema tolerante a falhas. De seguida, foram instaladas 3 máquinas com o Zookeeper, uma máquina com a tecnologia Hue e, por fim, o gestor de recursos do Yarn em simultâneo com o gestor de tarefas. Além disso, o Apache Impala está instalado em 6 máquinas, em que 5 incluem o Impalad e na outra o Catalogd e Statestored. Finalmente, o Apache Kudu foi instalado em 3 máquinas com Kudu Master e 7 máquinas com o Kudu Tablet Server.

Esta infraestrutura a nível de *hardware* continha as seguintes características:

- 192 GB de RAM (Random Access Memory).
- CPU da Intel Xeon E5540, 2.53GHz.
- 2 discos de 900 GB.
- Sistema operativo Red Hat Enterprise Linux Server release 7.2. (Maipo).

6.1.2 Métricas de avaliação

As métricas que vão ser utilizadas com o intuito de avaliar o desempenho da tecnologia selecionada, são:

- Tempo total de inserção nas tabelas *raw*.
- Tempo total de inserção nas tabelas agregadas por intervalos temporais com diferentes granularidades.
- Tempo total das consultas.

A primeira métrica tem como objetivo analisar o tempo de inserção nas tabelas *raw* conforme o processo descrito na secção 5.3.1, ou seja, o tempo que o Apache Kudu API demora a inserir os dados nas tabelas *raw*, de forma a que estes fiquem disponíveis para serem consultados. A segunda métrica analisa o tempo necessário para calcular e inserir os dados nas tabelas agregadas por um período temporal, por exemplo hora ou dia. Os dados agregados são calculados a partir dos dados *raw*. O acesso à base de dados é feito via JDBC.

A última métrica analisa o tempo de execução das consultas pré-definidas submetidas via JDBC. Foram definidas duas listas com tipos diferentes de consultas, como se pode visualizar na Tabela 6.1 e na Tabela 6.2. A primeira lista inclui mais duas consultas do que a segunda, que correspondem a dois exemplos de elaboração de relatórios de rede: a consulta C5 inclui 4 *joins* e a C6 incluí 25 *joins*. As duas listas contêm uma consulta que não inclui nenhum *join*, uma consulta com a inclusão de um *join*, consultas com dois *joins* e, por fim, consultas que incluem três *joins*. Para as consultas com um *join*, faz-se a diferenciação entre FULL OUTER JOIN e INNER JOIN. Uma consulta que engloba dois grupos e inclui um FULL OUTER JOIN retorna todos os valores dos dois grupos, enquanto que para o INNER JOIN só retorna os valores que são comuns aos dois grupos.

Tabela 6.1: Consultas das tabelas raw

Consulta	**Designação**
C1	Consulta sem nenhum join
C2	Consulta com um join (FULL OUTER JOIN)
C3	Consulta com dois join
C4	Consulta com três join
C5	Consulta de um exemplo de relatório de rede
C6	Consulta de um exemplo de relatório de rede
C7	Consulta com um join (INNER JOIN)

Tabela 6.2: Consultas das tabelas agregadas

Consulta	**Designação**
C1	Consulta sem nenhum join
C2	Consulta com um join (FULL OUTER JOIN)
C3	Consulta com dois join
C4	Consulta com três join
C5	Consulta com um join (INNER JOIN)

6.2 Estratégias de avaliação

As estratégias de avaliação organizam-se em quatro fases:

- **Primeira fase -** Comparação dos tipos de particionamento das tabelas.
- **Segunda fase -** Comparação dos tipos de compressão dos dados.
- **Terceira fase -** Geração de relatórios de rede.
- **Quarta fase -** Comparação da comunicação entre o Apache Kudu e o Apache Impala e do Apache Kudu com o Apache Presto.

Para incorporar o Apache Kudu no ambiente de testes foi necessário perceber qual o tipo de particionamento das tabelas mais adequado para o NPM. O Apache Kudu disponibiliza 3 tipos de particionamento: por Range, Hash e Combinado. Assim, na primeira fase analisa-se o tempo para a inserção de dados nas tabelas *raw* e para a inserção de dados nas tabelas agregadas por hora. De seguida, irá proceder-se à analise dos tempos das consultas pré-definidas. Foi decidido, em consenso com a Nokia, que o tipo de particionamento com melhores resultados iria ser aplicado na fase seguinte e os outros dois tipos de particionamento seriam descartados.

A segunda fase tem em vista selecionar o tipo de compressão de dados a adotar. As hipóteses são Snappy, LZ4, Zlib e Sem Compressão. Primeiro analisam-se os tempos de inserção dos dados nas tabelas *raw* e nas tabelas agregadas (dia e hora) e, depois os tempos das consultas pré-definidas.

Na terceira fase avalia-se se a solução proposta é adequada ao contexto da Nokia. Por este motivo, esta fase vai integrar as decisões tomadas anteriormente, acrescentado a geração de relatórios de rede. A geração de relatórios consiste em gerar aproximadamente 440 relatórios de rede usando um programa em ciclo que após concluir a execução de 440 relatórios, volta a gerá-los novamente. Não obstante, enquanto são gerados os relatórios de rede também são inseridos dados nas tabelas *raw* bem como nas tabelas agregadas. Por fim, como objetivo essencial para esta fase, foi definida a geração de relatórios de rede durante 9 horas e incluir dois testes com diferentes cargas para o simulador de dados (aproximadamente 108 milhões de contadores de rede e aproximadamente 245 milhões de contadores de rede).

Finalmente, a quarta fase terá como finalidade comparar a comunicação de duas tecnologias Apache Impala e Apache Presto com o Apache Kudu, para executar consultas usando a linguagem SQL. Assim, procedeu-se à implementação e configuração das tecnologias, em que foram utilizadas as configurações apresentadas nas secções 5.3.4 e 5.3.3. Foi usado um Presto Coordinator e 6 Presto Workers.

6.3 Resultados

Esta secção apresenta os resultados obtidos usando as métricas definidas no ambiente de testes e as estratégias de avaliação delineadas no capítulo anterior. Complementarmente, será utilizando o T-Test com o nível de significância de 5%, para avaliar se existe uma diferença estatisticamente significativa entre os conjuntos de resultados [61].

6.3.1 Primeira Fase

A primeira fase de testes aplica-se à análise dos tipos de particionamento que o Apache Kudu fornece, ou seja, o particionamento por Range, Hash e Combinado. Uma questão natural é perceber quantas linhas foram inseridas na base de dados e qual a duração total do teste:

- Horas executadas: 49 horas.
- Número de elementos de rede processados nas tabelas *raw*: 367 824 240.
- Número de elementos de rede processados nas tabelas agregadas por hora: 152 162 346.

O teste inclui a simulação de mais de 108 milhões de contadores de rede e teve uma duração de 49 horas, resultando em mais de 367 milhões de elementos de rede inseridos nas tabelas *raw*. Estes foram inseridos nas tabelas *raw* de 15 em 15 minutos. Deste modo, foram agregados por hora mais de 152 milhões de elementos de rede.

Foi necessário definir a forma como foi feito o particionamento dos dados e, desta forma, o número de *tablets* por tabela. A decisão sobre as tabelas que devem ser particionadas foi apresentada na secção 5.2.2, e o número de *tablets* resultante por tabela é apresentado na Tabela 6.3. Para o particionamento por Hash, o número de *tablets* por tabela é 7, por ser o número total de máquinas que a infraestrutura proporciona para os Kudu Tablet Servers. Para o particionamento por Range, o número de *tablets* por tabela é 3, porque as 49h de execução da primeira fase foram intercaladas em 3 dias, e os dados foram divididos por dia. Para o particionamento Combinado, foi tido em conta o conhecimento retirado das decisões anteriores e foram usados 3 *tablets* para o particionamento por Range, cada um dividido em 4 *tablets* particionados por Hash, obtendo-se 12 *tablets* por tabela.

Tabela 6.3: Quantidade de *tablets* por tabela

Tipo de particionamento	**Número de *tablets* por tabela**
Hash	7
Range	3
Combinado	4*3

Inserção dos dados

O primeiro teste corresponde a uma análise do tempo total de inserção nas tabelas *raw* e o resultado final pode ser visualizado na Figura 6.1.

Esta figura mostra que o tempo médio de inserção usando o particionamento por Range e Hash varia entre os 6 e os 6.25 segundos, enquanto que usando particionamento Combinado a média é aproximadamente 5 segundos. Verifica-se que no particionamento Combinado existe, em média uma diferença de 25%, isto é superior a 1 segundo comparativamente aos outros dois tipos de particionamento.

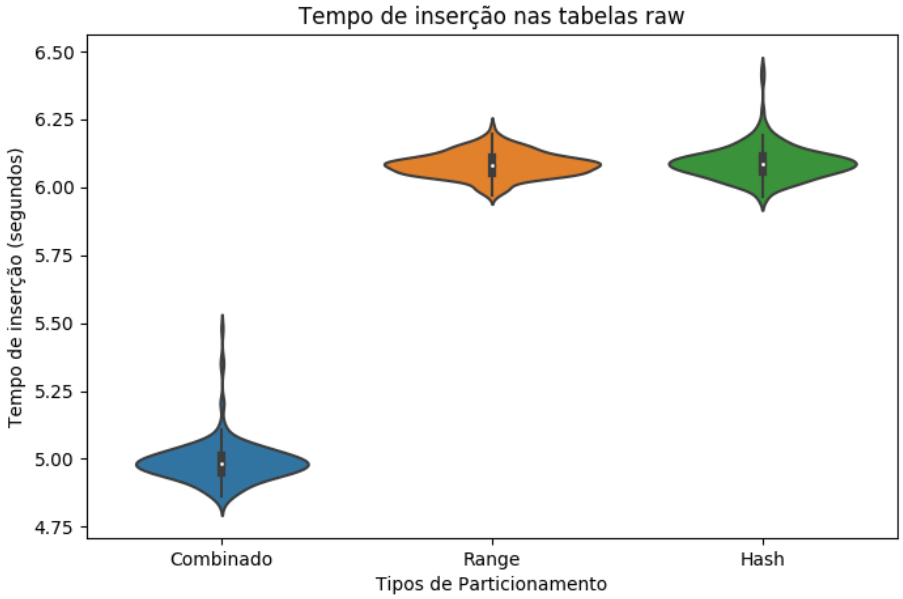


Figura 6.1: Comparação dos diferentes tipos de particionamento nas tabelas *raw*

De seguida, foi executado o T-Test, cujo resultado pode ser visualizado na Tabela 6.4, que mostra que o nível de significância da diferença de resultados do método Combinado com os métodos Range e Hash é elevado. Por outro lado, conclui-se também que não existe uma diferença significativa entre os métodos Range e Hash.

Tabela 6.4: T-Test - Comparação dos diferentes tipos de particionamento nas tabelas *raw*

Conjuntos	**T-statistic**	**P-value**
Combinado vs Range	178.727	0 %
Combinado vs Hash	162.225	0 %
Range vs Hash	1.274	20 %

O segundo teste corresponde à análise do tempo total de inserção nas tabelas agregadas, considerando a agregação dos dados por hora, e o resultado pode ser visualizado na Figura 6.2.

O particionamento por Hash apresenta o pior resultado, sendo que o tempo médio de inserção dos dados nas tabelas agregadas oscila entre os 30 e 35 segundos. O tempo médio de inserção dos dados nas tabelas agregadas usando particionamento por Range varia entre 24 e 30 segundos e o melhor resultado corresponde ao particionamento Combinado onde o tempo médio de inserção dos dados é de cerca de 24 segundos (35% e 13% mais rápido que Hash e Range, respetivamente) e o desvio padrão é inferior ao obtido para os outros métodos de particionamento. Os resultados do T-Test podem ser observados na Tabela 6.5 e confirmam a significância dos resultados apresentados na Figura 6.2.

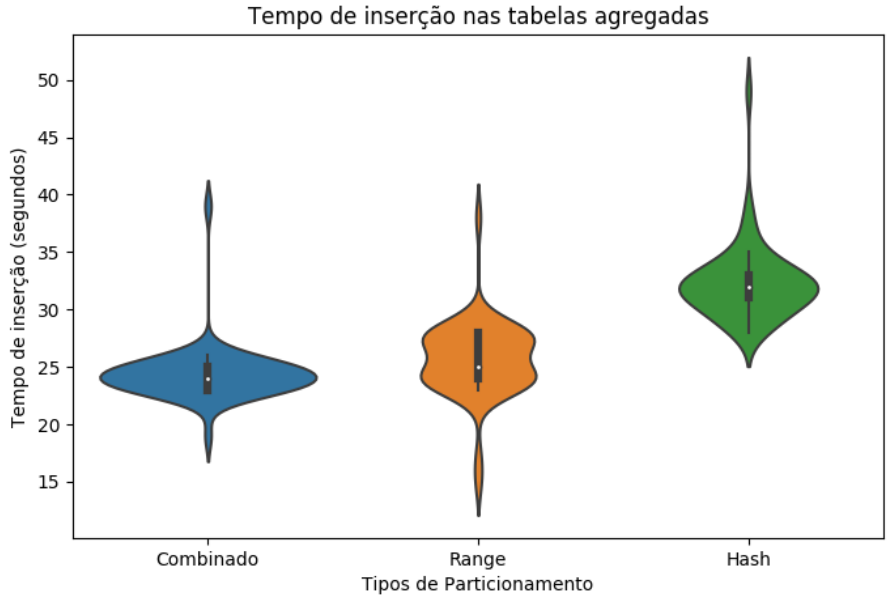


Figura 6.2: Comparação dos diferentes tipos de particionamento nas tabelas agregadas por hora

Tabela 6.5: T-Test - Comparação dos diferentes tipos de particionamento nas tabelas agregadas por hora

Conjuntos	**T-statistic**	**P-value**
Combinado vs Range	2.209	3 %
Combinado vs Hash	13.957	0 %
Range vs Hash	10.483	0 %

Consultas de dados

O último teste analisa o tempo total das consultas pré-definidas às tabelas *raw* e agregadas por hora, e os resultados podem ser visualizados na Figura 6.3 e na Figura 6.4.

A Figura 6.3 mostra que os resultados do particionamento por Hash são os piores e que os melhores obtêm-se para o particionamento Combinado. Uma análise mais detalhada das consultas C2 e C7, que correspondem a dois tipos de *join* diferentes, mostra que os tempos de resposta para a C7 são mais rápidos e que o tipo de particionamento com melhores resultados é o Hash. De seguida, apresentam-se outras consultas, nomeadamente C1, que verifica o tempo da consulta sem nenhum *join*, a C3, que verifica o tempo de execução com 2 *joins* e a C4, que verifica o tempo de execução com 3 *joins*. Nestas três consultas, os tempos de execução obtêm melhores resultados no particionamento Combinado.

Para a inclusão da terceira fase, é essencial incluir dois exemplos de relatórios de rede, diferenciado-os pela quantidade de *joins* que cada um inclui (C5 e C6). Consequentemente, mesmo que o particionamento por Range e Combinado apresentem resultados próximos, o particionamento Combinado tem tempos de execução mais baixos em cerca de 2 segundos.

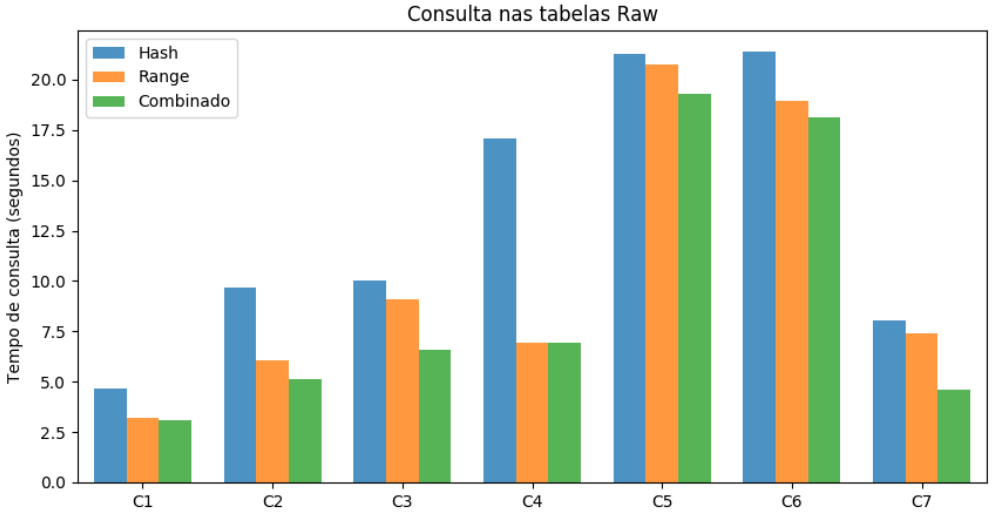


Figura 6.3: Consultas das tabelas raw, com diferentes tipos de particionamento

Os resultados da avaliação das consultas pré-definidas agregadas por hora, podem ser visualizados na Figura 6.4. O particionamento por Hash apresenta os piores resultados comparativamente aos outros tipos de particionamento, e o particionamento Combinado é o que tem melhores resultados nas consultas. A comparação da C2 e da C5, mostra que o tempo de execução para as consultas com INNER JOIN é superior às consultas com FULL OUTER JOIN. A diferença dos tempos de execução em C5 usando os diferentes tipos de particionamento é aproximadamente 1 segundo. Para as outras consultas, nomeadamente C1, C3 e C4, verifica-se o aumento do tempo de execução quando o número de *joins* é maior, porém o particionamento Combinado continua a apresentar o melhor resultado.

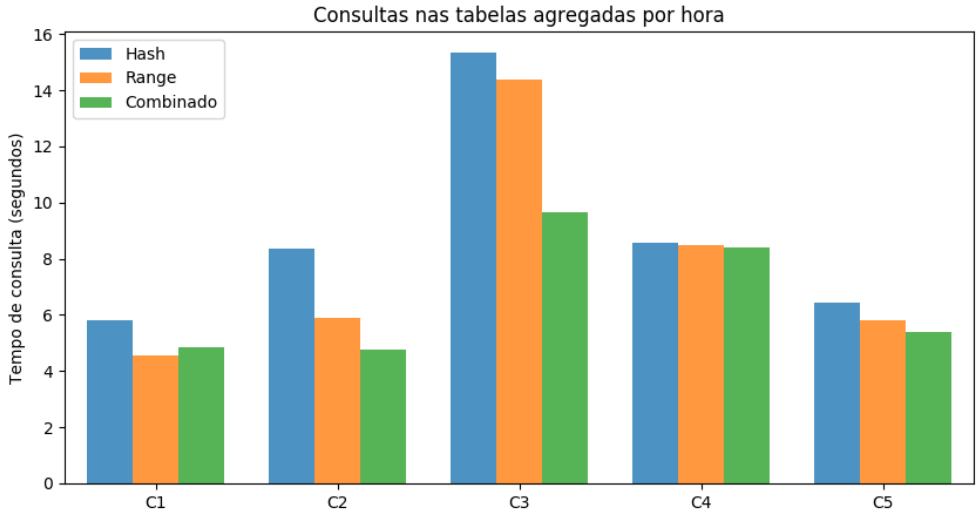


Figura 6.4: Consultas das tabelas agregadas por hora, com diferentes tipos de particionamento

A análise global destes resultados permite constatar qual o melhor tipo de particionamento para o ambiente de testes definido. O tipo de particionamento com os melhores resultados, tanto a nível de inserção como de consultas, é o Combinado. Desta forma, é este o tipo de particionamento selecionado para a próxima fase de testes.

6.3.2 Segunda Fase

A segunda fase de testes foca-se na análise dos tipos de compressão dos dados que o Apache Kudu proporciona, nomeadamente Snappy, LZ4, Zlib e sem compressão. Os testes nesta fase baseiam-se na simulação de mais 108 milhões de contadores de rede, o uso do particionamento Combinado e agregação dos dados por dia. As principais estatísticas relacionadas com esta fase de testes são:

- Horas executadas: 49 horas.
- Número de elementos de rede processados nas tabelas raw: 367 824 240.
- Número de elementos de rede processados nas tabelas agregadas por hora: 152 162 346.
- Número de elementos de rede processados nas tabelas agregadas por dia: 6 210 708.

Este teste teve uma duração total de 49 horas, das quais resultaram mais de 367 milhões de elementos de rede processados e inseridos nas tabelas *raw* de 15 em 15 minutos. Agregando estes elementos de rede por hora obtemos mais de 152 milhões e por dia obtemos mais de 6 milhões a inserir nas tabelas de dados agregados. No entanto, para as tabelas com valores agregados por dia, só será avaliada a terceira métrica, porque os dados não são suficientes para a análise das outras métricas.

Inserção dos dados

A primeira métrica de avaliação corresponde à análise do tempo total de inserção nas tabelas *raw* e os tempos de inserção estão representados na Figura 6.5.

Os resultados mostram que a média dos tempos de inserção dos dados a cada 15 minutos usando os tipos de compressão Snappy, Sem compressão, LZ4 e Zlib é aproximadamente 5 segundos. De facto, verifica-se que não existe uma diferença significativa ao nível da inserção dos dados nas tabelas *raw* para os diversos tipos de compressão, podendo apenas constatar-se que a dispersão dos resultados para o LZ4 e o Zlib é ligeiramente inferior aos outros dois métodos. Para comprovar a diferença ou a similaridade entre os quatro tipos de compressão de dados, foi realizado o T-Test, e os resultados podem ser visualizados na Tabela 6.6. O valor T-Statistic para os tipos de compressão de dados LZ4 e Zlib, assim como Snappy e Sem compressão, são similares, e o valor do P-value correspondente é próximo de 50 %. Isto comprova a similaridade dos resultados obtidos para estes pares de métodos. O valor do P-value para os outros conjuntos é 0 % e 3 %, logo é possível rejeitar os casos nulos.

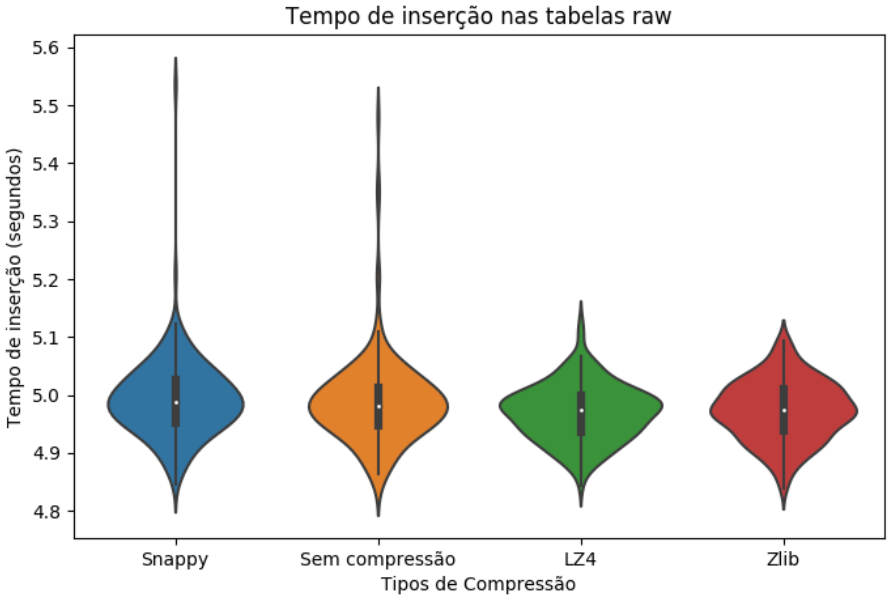


Figura 6.5: Comparação dos diferentes tipos de compressão nas tabelas raw

Tabela 6.6: T-Test - Comparação dos diferentes tipos de compressão nas tabelas raw

Conjuntos	**T-Statistic**	**P-value**
LZ4 vs Snappy	3.806	0 %
LZ4 vs Sem compressão	2.778	0 %
LZ4 vs Zlib	0.664	50 %
Snappy vs Sem compressão	0.694	46 %
Snappy vs Zlib	3.169	0 %
Sem compressão vs Zlib	2.205	3 %

A Figura 6.6 apresenta o tempo total de inserção dos dados nas tabelas agregadas por hora. Os tempos de inserção são mais elevados para os tipos de compressão LZ4 e Zlib, sendo a média aproximadamente 30 segundos. Para o Zlib, o tempo de inserção dos dados varia entre os 20 e os 30 segundos. Comparativamente, o Snappy e Sem compressão de dados tiveram resultados mais consistentes, e os tempos médios de inserção são aproximadamente 24 e 26 segundos. Na Figura 6.6 pode visualizar-se que os conjuntos não têm diferenças superiores a 20% entre eles, ou seja, superior a 5 segundos.

A Tabela 6.7 mostra o resultado do T-Test confirmando a significância dos resultados apresentados na Figura 6.6.

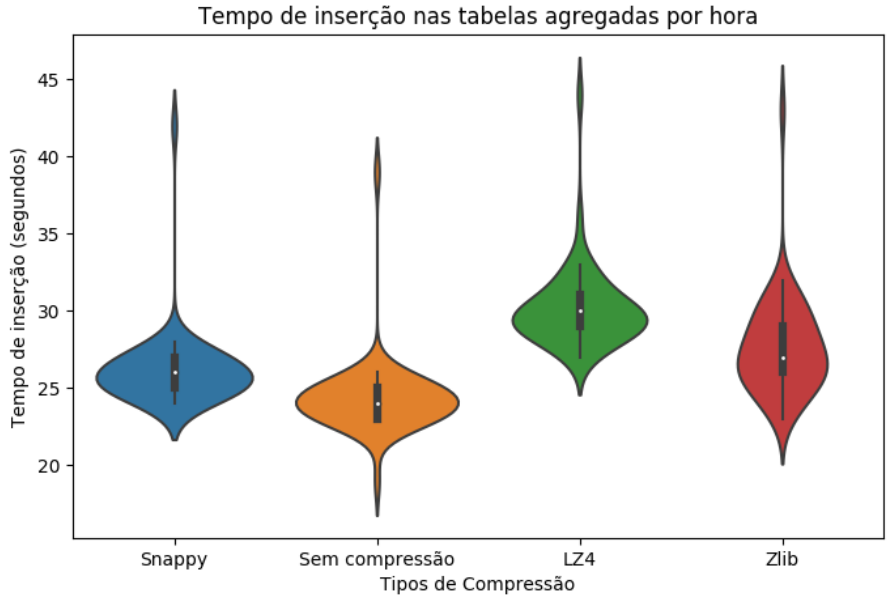


Figura 6.6: Comparação dos diferentes tipos de compressão nas tabelas agregadas por hora

Tabela 6.7: T-Test - Comparação dos diferentes tipos de compressão nas tabelas agregadas por hora

Conjuntos	**T-Statistic**	**P-value**
LZ4 vs Snappy	7.723	0 %
LZ4 vs Sem compressão	11.355	0 %
LZ4 vs Zlib	4.330	0 %
Snappy vs Sem compressão	3.568	0 %
Snappy vs Zlib	2.569	1 %
Sem compressão vs Zlib	5.748	0 %

Consultas de dados

A terceira métrica de avaliação corresponde ao tempo total para a execução de consultas de dados nas tabelas *raw* e nas tabelas com os dados agregados por hora e por dia. Os resultados são apresentados nas Figuras 6.7, 6.8 e 6.9.

Os resultados apresentados na Figura 6.7 mostram que o tipo de compressão LZ4 tem os piores resultados, seguindo-se os tipos de compressão Zlib e Snappy. Os melhores resultados são obtidos quando não existe compressão dos dados, porque quando é executado o processo de consulta este não precisa de descomprimir os valores. Comparando as consultas C2 e C7, de forma a perceber como é que a compressão dos dados pode influenciar as consultas com diferentes tipo de *joins*, verifica-se que a C7 obteve tempos de execução mais reduzidos. Além disso, através da análise das consultas C5 e C6, verifica-se que os melhores resultados são obtidos quando não existe compressão de dados, mas a diferença relativamente aos métodos Snappt e Zlib é pequena. Comparando as consultas C1, C3 e C4 quanto ao número de *joins*, verifica-se que os melhores resultados são obtidos quando não existe compressão de dados,

vindo logo a seguir os tipos de compressão Snappy e Zlib, finalizando com o LZ4.

A Figura 6.8 apresenta os resultados das consultas às tabelas agregadas por hora. Os tempos são mais elevados para o tipo de compressão LZ4. Além disso, é possível verificar que, se os dados forem comprimidos através do Snappy, as consultas são mais rápidas (C1, C3 e C4) em relação ao dados sem qualquer tipo de compressão e aos dados comprimidos com o Zlib. Por outro lado, os resultados para as consultas C2 e C5 são melhores quando não existe compressão de dados.

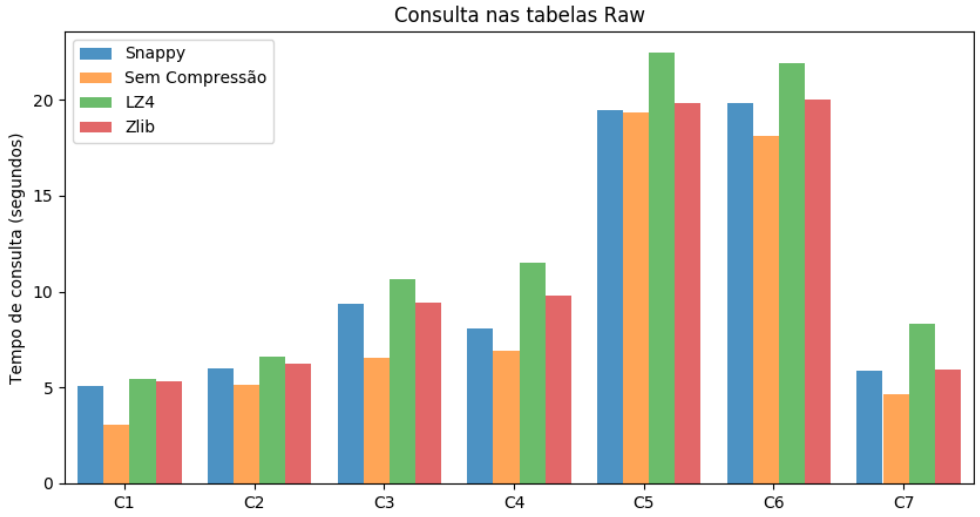


Figura 6.7: Consultas das tabelas raw, com diferentes tipos de compressão

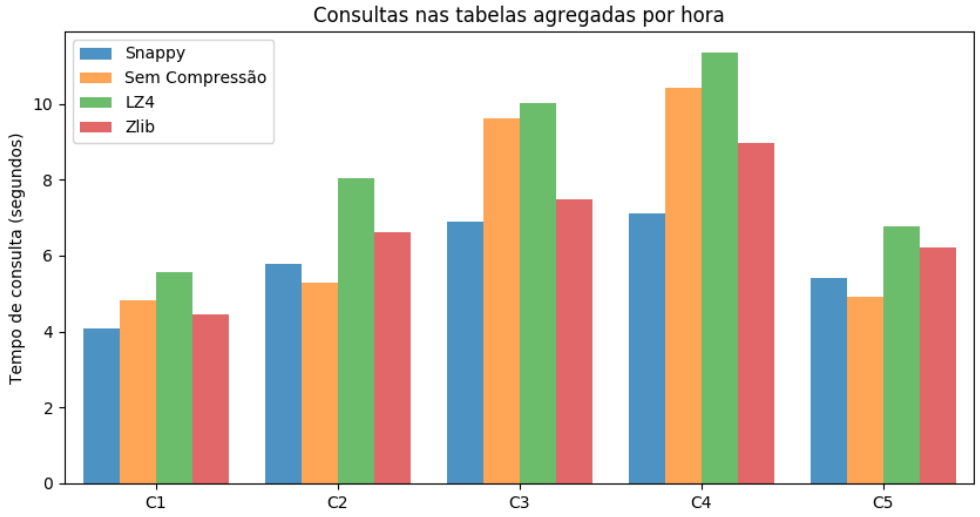


Figura 6.8: Consultas das tabelas agregadas por hora, com diferentes tipos de compressão

Os resultados apresentados na Figura 6.9 mostram que o desempenho das consultas usando o tipo de compressão de dados LZ4 também é pior comparativamente aos outros tipos de compressão. Além disso, é possível verificar que os melhores resultados são obtidos através do tipo de compressão de dados Snappy. Com mais detalhe, C1 e C4 são as consultas em que não usar compressão de dados tem melhores resultados em relação à compressão de dados Snappy.

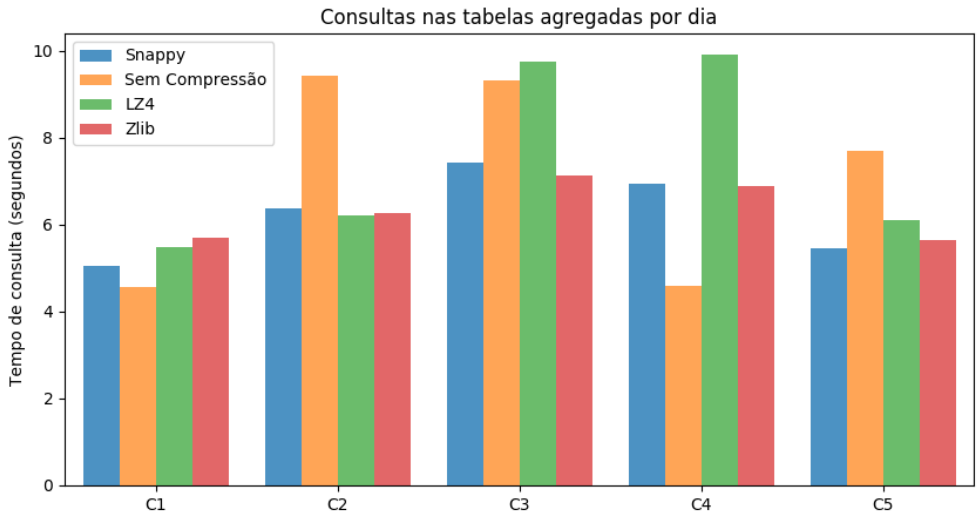


Figura 6.9: Consultas das tabelas agregadas por dia, com diferentes tipos de compressão

A seguir foi feita uma análise do desempenho obtido na inserção dos dados e nas consultas dos dados. Os resultados confirmam que, para este ambiente de testes, os dados comprimidos com o tipo LZ4 apresentam o pior desempenho. Para completar a avaliação, a Tabela 6.8 apresenta o espaço ocupado em disco para cada um dos tipos de compressão de dados. Os resultados mostram que o LZ4, embora tenha o pior desempenho em termos de tempos de execução, ocupa 2 vezes menos espaço em disco do que os outros tipos de compressão de dados. Adicionalmente, os dados comprimidos através do Snappy ocupam menos espaço em disco do que quando se usa o Zlib ou quando não se usa compressão de dados.

Finalizando, através da análise das três métricas de avaliação bem como o espaço em disco ocupado, foi decidido escolher para a próxima fase o tipo de compressão de dados Snappy.

Tabela 6.8: Espaço em disco (média)

Tipos de compressão	**Valor médio**
LZ4	25.1 GB
Zlib	45.51 GB
Snappy	44.87 GB
Sem Compressão	45.4 GB

6.3.3 Terceira Fase

A terceira fase de testes tinha como objetivo gerar relatórios de rede durante um período de 9h. Foram realizados dois testes que se diferenciavam pelo número de contadores de rede. O primeiro teste simulava mais de 108 milhões de contadores de rede, enquanto o segundo teste simulava mais de 245 milhões de contadores de rede.

Primeiro teste

O primeiro teste consistiu na simulação de mais 108 milhões de contadores de rede, o uso do particionamento Combinado e a compressão dos dados por Snappy. Os dados sobre estes testes são os seguintes:

- Tempo executado: Desde o dia 09 de outubro às 9h até o dia 16 de outubro até às 19h.
- Número de elementos de rede processados nas tabelas raw: 1 283 778 720.
- Número de elementos de rede processados nas tabelas agregadas por hora: 552 753 012.
- Número de elementos de rede processados nas tabelas agregadas por dia: 21 737 478.

Este teste decorreu entre o dia 9 de Outubro às 10 horas e o dia 16 de Outubro até às 19 horas, resultando em mais mil milhões de elementos de rede processados e inseridos nas tabelas *raw* de 15 em 15 minutos. Consequentemente, foram gerados mais de 21 milhões de elementos de rede agregados por dia bem como mais de 552 milhões de elementos de rede agregados por hora.

Porém, com as configurações até agora implementadas, ainda não era possível atingir o objetivo de executar 9 horas consecutivas na geração de relatórios de rede. Após uma discussão no slack do Kudu, observou-se a necessidade de utilizar a flag "SCHEDULE RANDOM REPLICA"e a flag "is_executor"nas configurações no Apache Impala. A primeira flag permite otimizar o algoritmo utilizado para decidir a melhor máquina Impalad que executa cada processo que não sejam HDFS, de forma a reduzir *hostspotting*. A segunda flag foi necessária para não sobrecarregar um Impalad em específico, fazendo com que esse fosse apenas coordenador das consultas e nunca o seu executor. A Figura 6.10 mostra os resultados obtidos após fazer estas alterações, representando a simulação de geração de relatórios em simultâneo com a inserção dos dados nas tabelas raw e agregadas.

O tempo máximo conseguido na geração de relatórios de rede teve uma duração de 6 horas e 30 minutos e os resultados são apresentados na Figura 6.10. Existem picos recorrentes em todas horas, e por este motivo decidiu-se criar um gráfico com a aproximação de 1 hora para analisar um período mais detalhadamente. A Figura 6.11 mostra a aproximação efetuada durante o período entre as 11h e as 12h, na qual se destaca um pico que acontece um pouco depois das 11:10h. A razão desse pico deve-se à agregação dos dados por hora ser feita sempre aos 10 minutos de cada hora, explicando os picos em cada período de uma hora na Figura 6.10. O impacto da inserção de dados de 15 em 15 minutos nas *raw* não é tão evidente.

Contudo, o resultado obtido nesta fase continuou a não cumprir com o objetivo anteriormente delineado, mais precisamente de possibilitar a geração de relatórios de rede durante um período de 9 horas consecutivas.

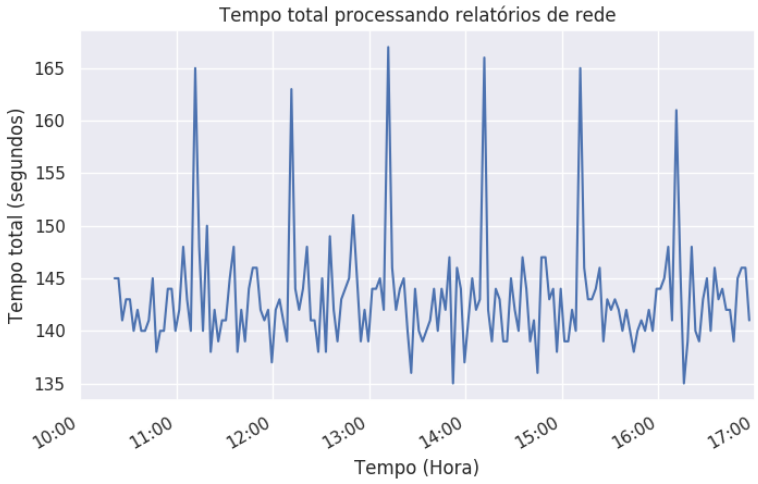


Figura 6.10: Geração de relatórios de rede, iniciou às 10:21h e terminou as 16:51h

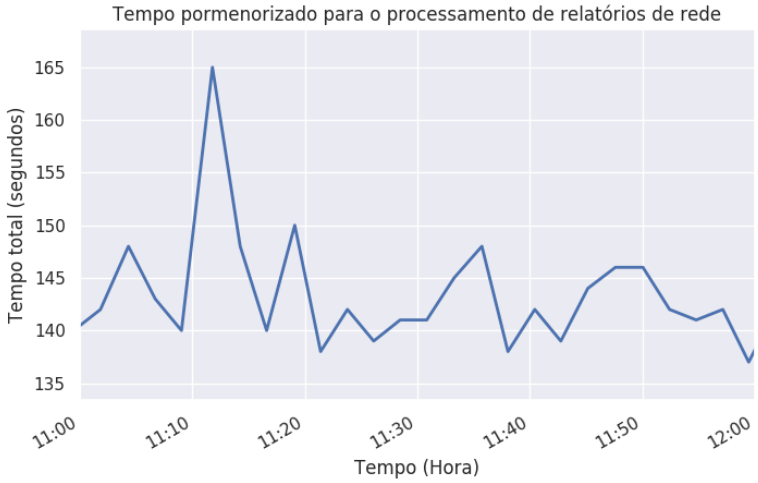


Figura 6.11: Consulta pormenorizada entre as 11h e as 12h

Segundo teste

O segundo teste incluiu a simulação de mais 245 milhões de contadores de rede, o uso do particionamento Combinado, e, por fim, a compressão dos dados por Snappy. Os dados sobre estes testes são os seguintes:

- Tempo executado: Desde o dia 19 às 21h:00 até o dia 22 até as 19h16.
- Número de elementos de de rede processados nas tabelas *raw*: 1 139 896 800.
- Número de elementos de de rede processados nas tabelas agregadas por hora: 497 806 134.

- Número de elementos de rede processados nas tabelas agregadas por dia: 7 011 354.

A execução deste teste decorreu entre o dia 19 de Outubro às 21 horas e o dia 22 de Outubro até às 19:16 horas, resultando em mais de mil milhões de elementos de rede inseridos nas tabelas *raw* de 15 em 15 minutos. Daqui resultaram mais de 7 milhões de dados, agregados por dia bem como mais de 497 milhões de dados, agregados por hora.

Além disso, acrescentou-se a opção "/* +NOSHUFFLE,NOCLUSTERED */"na construção das consultas, para desativar o particionamento dos dados e a sua ordenação [1]. A Figura 6.12 mostra os resultados obtidos ao fazer estas alterações, representando a simulação da geração de relatórios em simultâneo com a inserção dos dados nas tabelas raw e agregadas. O tempo máximo obtido na geração de relatórios de rede teve uma duração de 7 horas (das 11:48h até às 18:49h) e os resultados são apresentados na Figura 6.12. Também é detetada a existência de picos recorrentes em todas horas, por este motivo decidiu-se criar um gráfico com a aproximação de 1 hora com o intuito de analisar um dos períodos de uma hora com mais detalhe.

A Figura 6.13 demonstra a aproximação efetuada durante o período entre as 12h e as 13h, na qual se observa que esse pico é um pouco depois das 12:10h. A razão desse pico deve-se à agregação dos dados por hora ser feita sempre aos 10 minutos de cada hora, explicando os picos de todas as horas vistos anteriormente. O impacto da inserção dos dados *raw* não é detetado. Por fim, o resultado obtido nesta fase continuou a não cumprir com o objetivo anteriormente delineado, mais precisamente de possibilitar a geração de relatórios de rede durante um período de 9 horas, porém conseguiu aumentar meia hora comparativamente ao primeiro teste, obtendo um valor de 7 horas.

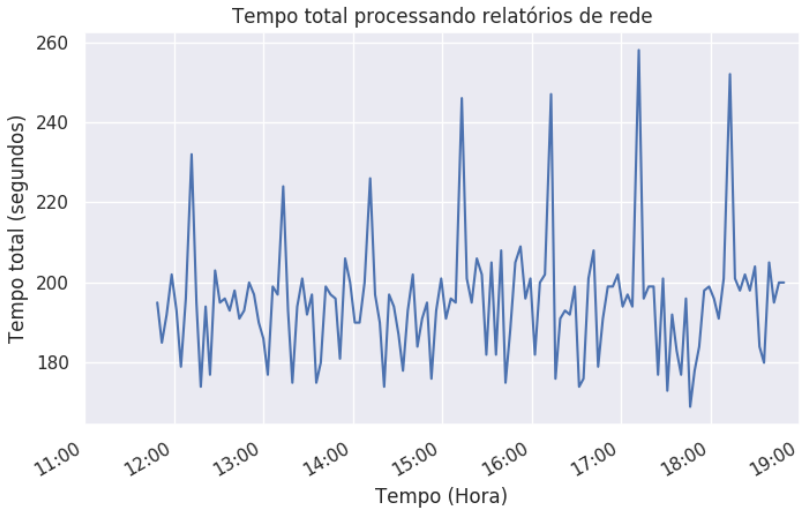


Figura 6.12: Geração de relatórios de rede, iniciou às 11:48h e terminou as 18:49h

[1] https://www.cloudera.com/documentation/enterprise/5-14-x/topics/impala_hints.html

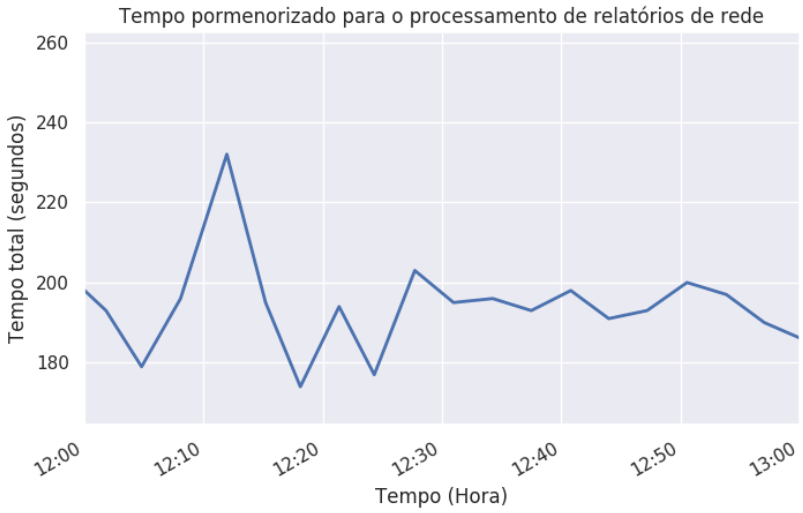


Figura 6.13: Consulta pormenorizada entre as 12h e as 13h

6.3.4 Quarta Fase

A quarta fase de testes tinha como objetivo comparar a comunicação de duas tecnologias (Apache Impala e Apache Presto) com o Apache Kudu. Tal como na fase anterior, foram simulados mais de 245 milhões de contadores de rede, foi usado o particionamento Combinado e o tipo de compressão dos dados por Snappy. Os dados deste teste são:

- Número de elementos de rede processados nas tabelas *raw*: 2 515 915 080.
- Número de elementos de rede processados nas tabelas agregadas por hora: 1 100 782 494.
- Número de elementos de rede processados nas tabelas agregadas por dia: 49 079 478.

A Figura 6.14 apresenta os resultados para as tabelas *raw*. Globalmente os tempos de execução das consultas usando o Apache Presto são mais baixos do que usando o Apache Impala. Porém, quando as consultas envolvem 3 *joins* (C4), o Apache Presto tem um agravamento de 10 segundos perante o Apache Impala. A razão deste agravamento pode ser uma má configuração da tecnologia ou por o Apache Presto ser menos eficiente a executar operações *join*. Comparando as consultas C2 e a C3, verifica-se que no primeiro caso o Apache Presto teve uma diferença aproximadamente de 10 segundos relativamente ao Apache Impala, porém com a inclusão de 2 *joins* os tempos de execução passam a ser semelhantes. Relativamente às consultas nas tabelas agregadas por hora (Figura 6.15), o Presto tem melhores resultados quando as consultas têm zero ou um *join* mas, a diferença tende a diminuir aumentado o número de *joins*. Por fim, para as tabelas agregadas por dia (Figura 6.16) verifica-se uma maior discrepância nos tempos das consultas entre as duas tecnologias.

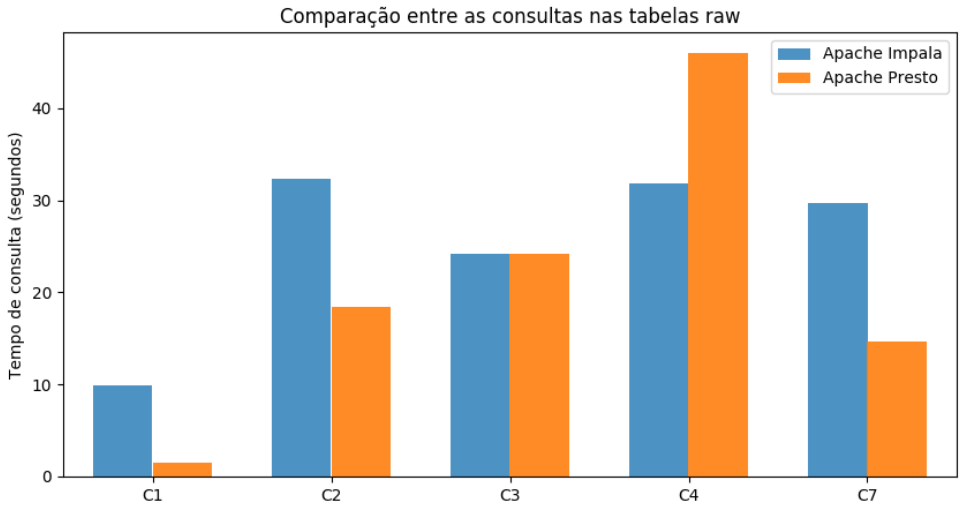


Figura 6.14: Consulta das tabelas raw usando Apache Impala e Apache Presto

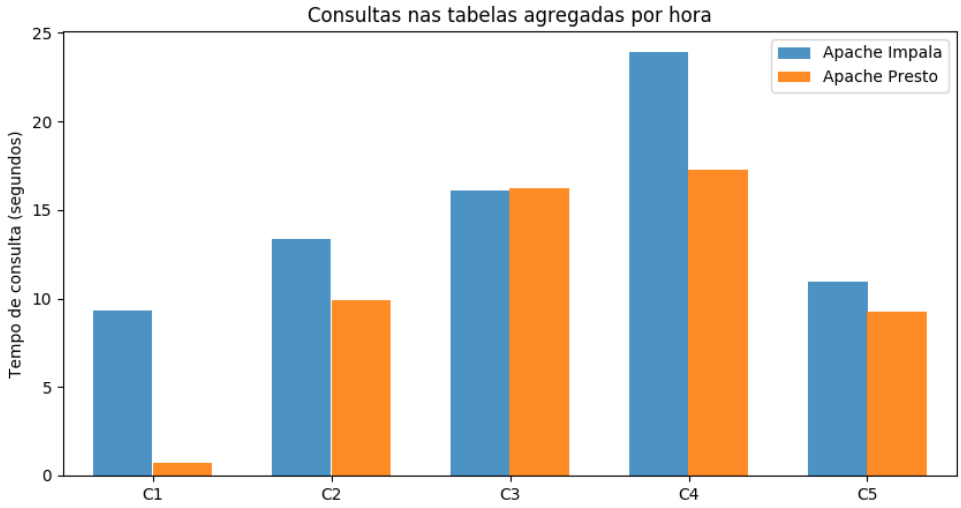


Figura 6.15: Consultas das tabelas agregadas por hora usando Apache Impala e Apache Presto

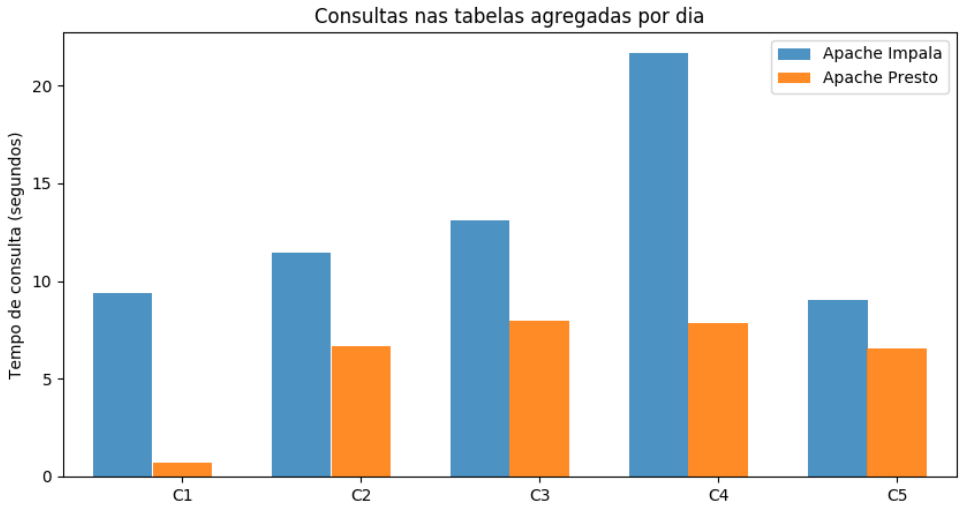


Figura 6.16: Consultas das tabelas agregadas por dia usando Apache Impala e Apache Presto

6.4 Síntese

Este capítulo apresenta os resultados da avaliação do Apache Kudu tendo em conta os objetivos traçados pela Nokia. Primeiro, avaliou-se qual o tipo de particionamento mais adequado e os resultados apontam para o particionamento Combinado. No entanto, importa referir que o número *tablets* usado durante os testes pode ter influenciado os resultados, porque o particionamento combinado foi implementado usando 12 *tablets* enquanto que para o particionamento por Hash foi 7 *tablets* e por Range foi 3 *tablets*.

Na análise por tipo de compressão dos dados analisou-se a relação entre o espaço em disco e os tempos de inserção e consulta de dados. Os tempos de execução para o tipo de compressão LZ4 são mais elevados do que quando se usam outros tipos de compressão de dados ou quando não se usa compressão, porém o espaço de armazenamento necessário é cerca de metade. Optou-se pelo tipo de compressão de dados Snappy, porque apresentava a melhor relação entre os tempos de inserção e consulta de dados, e o espaço ocupado em disco.

Após tomar estas duas decisões, foram realizados os testes de geração de relatórios de rede que simulam as cargas de trabalho típicas do NPM. O primeiro teste consistiu na simulação de mais 108 milhões de contadores de rede e teve uma duração máxima de 6:30 horas. O segundo teste consistiu na simulação de mais de 245 milhões de contadores rede e teve uma duração máxima de 7 horas. Porém, estes testes não cumpriram o objetivo delineado para esta fase por nenhum ter atingido a duração de 9 horas. A razão pode estar relacionada com a concorrência dos componentes de inserção dos dados e geração de relatórios de rede ou por problemas de configuração do Apache Kudu.

Finalmente, fez-se uma comparação entre o Apache Impala e o Apache Presto, que são duas tecnologias que permitem fazer consultas SQL sobre dados armazenados usando o Apache Kudu. Globalmente o Apache Presto tem melhores resultados do que o Apache Impala. No entanto, quando se começa a aumentar o número de *joins* a performance do Apache Presto agrava mais do que quando se usa o Apache Impala.

Capítulo 7

Conclusão e Trabalho Futuro

7.1 Conclusão

Esta dissertação, realizada em parceria com a Nokia, apresenta um estudo para integrar uma base de dados orientada às séries temporais (TSDB) no sistema de gestão de performance de redes e telecomunicações (NPM), desenvolvido por esta empresa. O estudo inclui a pesquisa e a comparação das características de diversas TSDB e a realização de testes de desempenho para avaliar a solução selecionada e determinar as configurações mais eficientes no contexto do NPM.

A tecnologia selecionada foi o Apache Kudu porque permite a escalabilidade horizontal e alta disponibilidade, tem suporte SQL e inclui um JDBC driver. Procedeu-se à implementação de um ambiente de testes adaptado ao NPM, de modo a avaliar o desempenho do Apache Kudu consoante o particionamento das tabelas, a compressão dos dados, a integração com a geração dos relatórios de rede e, finalmente, uma comparação entre o Apache Presto e Apache Impala para permitir a consulta de dados usando a linguagem SQL.

Os testes de desempenho realizados mostram que o Apache Kudu tem melhores resultados usando particionamento Combinado (Hash e Range) e compressão de dados Snappy. O comportamento do sistema cumpre a larga maioria dos requisitos definidos pela Nokia, mas quando se realizam testes de longa duração, executando de forma concorrente os processos de inserção nas tabelas raw, inserção nas tabelas de dados agregados e geração de relatórios de rede, os testes falham ao fim de 7 horas. O objetivo traçado pela Nokia era atingir as 9 horas de execução contínua sem a ocorrência de falhas. Como o Apache Kudu não permite a utilização da linguagem SQL e de JDBC, foi necessário recorrer ao Apache Presto e Apache Impala para acrescentar esta funcionalidade. Conclui-se que o primeiro tem melhores tempos de resposta, porém o desempenho diminui quando as consultas incluem 3 ou mais *joins*.

O trabalho realizado leva a concluir que a solução proposta baseada em Apache Kudu ainda tem algumas limitações que fazem com que a sua utilização como tecnologia de base para o armazenamento e a consulta de dados no NPM seria prematura. No entanto, esta é uma área em grande desenvolvimento e a utilização de TSDB no âmbito dos sistemas de gestão de performance de redes e telecomunicações deve continuar a ser considerada no futuro.

7.2 Lições aprendidas

O desenvolvimento desta dissertação levou a algumas lições aprendidas que serão descritas a seguir de acordo com as seguintes fases:

1. Análise do problema.
2. Arquitetura e Implementação.
3. Comunicação entre o Apache Kudu e Apache Impala.
4. Comunicação entre o Apache Kudu e Apache Presto.

A **Análise do problema** tinha como objetivo interligar os conceitos teóricos com as características das tecnologias existentes. A questão crítica nesta fase foi conseguir perceber se as tecnologias encontradas conseguiriam dar resposta aos requisitos enumerados pela Nokia e validar a sua documentação, pois por vezes a documentação estava desatualizada por se tratar de uma tecnologia abandonada ou por ser uma tecnologia muito recente. Os 3 requisitos principais considerados foram: escalabilidade Horizontal, e possibilidade de usar linguagem SQL e JDBC. A escalabilidade horizontal foi um dos requisitos principais da Nokia, porque a solução a adotar deveria ser capaz de lidar com um grande volume de dados. A versão 0.8.0 do TimescaleDB não cumpria este requisito, porém, atualmente a versão 1.0 permite a paralelização de consultas. Por outro lado, por se tratarem maioritariamente de bases de dados NoSQL, são poucas as que incluem JDBC. Por exemplo, o InfluxDB (na versão 1.6) e o Graphite (na versão 1.14) continuam a não incluir. O terceiro requisito foi permitir consultas SQL. A documentação do RiakTS era incompleta e detetaram-se limitações, nomeadamente não permitir eliminar tabelas. Ao analisar o seu Github concluiu-se tratar-se de um projeto abandonado pois já não existiam atualizações há mais de um ano. Na sequência deste trabalho de pesquisa e avaliação de soluções, a primeira lição aprendida foi que não existe uma solução perfeita ou melhor em todos os aspetos pretendidos pela Nokia.

Na fase **Arquitetura e Implementação**, foi proposta uma arquitetura baseada em Apache Kudu para suportar o armazenamento e a consulta de dados no NPM. Estudou-se o procedimento para o particionamento das tabelas tendo em atenção o modelo de dados do NPM e o *workload* esperado. A segunda lição aprendida é que não existe um particionamento de tabelas melhor para todos os casos e que este tem que ser adaptável a cada situação.

Em seguida, a fase sobre a **Comunicação entre o Apache Impala e o Apache Kudu** baseou-se nas decisões sobre a configuração do sistema tomadas na fase de arquitetura e implementação, e centrou-se nas estratégias de avaliação global do desempenho da solução proposta. Os objetivos propostos foram atingidos até ao momento em que todos os processos foram lançados concorrentemente: inserção de dados nas tabelas *raw* e nas tabelas agregadas por hora e dia, e geração de relatórios de rede. Quando a geração de relatórios de rede é lançada em simultâneo com a inserção de dados nas tabelas *raw* e agregadas, existem falhas que só se manifestam ao fim de algumas horas de execução em contínuo. A pesquisa de soluções e a reconfiguração do sistema permitiu prolongar o tempo de execução até 7 horas consecutivas. A terceira lição aprendida é que a a combinação de *flags* permite melhorar o desempenho do sistema. Eventualmente seria possível atingir o objetivo de 9 horas de execução contínua proposto pela Nokia.

A implementação de um exemplo funcional na fase **Comunicação entre o Apache Presto e o Apache Kudu** foi relativamente simples, A parte difícil foi perceber quais as *flags* que devem ser usadas para melhorar o desempenho do sistema e a quarta lição aprendida é que as operações de *join* têm um grande impacto no desempenho do Apache Presto. Podem ser problemas de configuração do sistema, por exemplo, quais as *flags* que devem ser usadas, ou outros problemas que ainda não foram descobertos.

7.3 Trabalho Futuro

Primeiro, e tendo em atenção os objetivos propostos pela Nokia, seria importante continuar a tentar descobrir as causas que levam à falha do sistema ao fim de algumas horas de operação e tentar atingir pelo menos 9 horas de execução contínua. Para este efeito, poderia ser feita uma análise mais detalhada de inúmeras combinações de *flags* possíveis entre as tecnologias Apache Kudu e o Apache Impala.

Na realização deste trabalho considerou-se que todos os atributos de todas as tabelas usariam o mesmo tipo de compressão de dados. No futuro, seria interessante considerar a possibilidade de usar tipos de compressão de dados ajustados às características de cada atributo. Por exemplo, atributos com muitos valores repetidos (cardinalidade baixa) podem ser mais adequados para o uso de compressão de dados.

Por fim, considerar a utilização do Apache Presto como alternativa ao Apache Impala, no processamento de consultas SQL. Para este efeito deveria ser otimizada a performance das consultas quando são utilizadas operações *join*.

Referências

[1] David Reinsel, John Gantz and John Rydning. *Data Age 2025: The Evolution of Data to Life-Critical*. 2017. URL: https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf (acedido em 24/01/2018).

[2] *Nokia in Portugal | Nokia Networks*. URL: https://networks.nokia.com/pt (acedido em 24/01/2018).

[3] Nokia Solutions and Networks. *Product Description Nokia Solutions and Networks Performance Manager. Technical Report*. 2014.

[4] Gartner. *Gartner IT Glossary - Big Data Defined*. 2013. URL: https://www.gartner.com/it-glossary/big-data.

[5] Ahmed Oussous et al. «Big Data technologies: A survey». Em: *Journal of King Saud University - Computer and Information Sciences* 30.4 (2018), pp. 431 –448. ISSN: 319-1578.

[6] Veda C. Storey and Il Yeol Song. «Big data technologies and Management: What conceptual modeling can do». Em: *Data & Knowledge Engineering* 108 (2017), pp. 50 –67. ISSN: 0169-023X.

[7] *Big data: conheça os 5 V's e sua aplicação prática para PMEs*. URL: https://pt.semrush.com/blog/big-data-conheca-os-5-vs-e-sua-aplicacao-pratica-para-pmes/ (acedido em 02/09/2018).

[8] Infographic: The Four V's of Big Data. *IBM Big Data & Analytics Hub*. URL: http://www.ibmbigdatahub.com/infographic/four-vs-big-data (acedido em 06/09/2018).

[9] Min Chen, Shiwen Mao and Yunhao Liu. «Big Data: A Survey». Em: *Mobile Networks and Applications* 19.2 (2014), pp. 171–209. ISSN: 1572-8153.

[10] *ACID vs. BASE: O deslocamento do pH do processamento de transações do banco de dados - DATAVERSITY*. URL: http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/ (acedido em 25/09/2018).

[11] *In the Machine Learning Era, Unstructured Data Management is More Important Than Ever*. URL: https://www.igneous.io/blog/structured-data-vs-unstructured-data (acedido em 25/09/2018).

[12] Ameya Nayak, Anil Poriya and Dikshay Poojary. «Type of NOSQL Databases and its Comparison with Relational Databases». Em: *International Journal of Applied Information Systems* 5.4 (2013). URL: http://research.ijais.org/volume5/number4/ijais12-450888.pdf.

[13] Alejandro Corbellini et al. ≪Persisting big-data: The NoSQL landscape≫. Em: *Information Systems* 63 (2017), pp. 1–23. ISSN: 0306-4379.

[14] E. Brewer. ≪CAP twelve years later: How the "rules" have changed≫. Em: *Computer* 45 (2012), pp. 23–29.

[15] Felix Gessert et al. ≪NoSQL database systems: a survey and decision guidance≫. Em: *Computer Science - Research and Development* 32.3 (2017), pp. 353–365. ISSN: 1865-2042.

[16] Quartz. *Connected cars will send 25 gigabytes of data to the cloud every hour*. 2015. URL: https://qz.com/344466/connected-cars-will-send-25-gigabytes-of-data-to-the-cloud-every-hour/ (acedido em 30/07/2018).

[17] G Shurkhovetskyy et al. ≪Data Abstraction for Visualizing Large Time Series≫. Em: 37.1 (2018), pp. 125–144.

[18] INVESTOPEDIA. *Time series*. URL: https://www.investopedia.com/terms/t/timeseries.asp (acedido em 06/09/2018).

[19] Dmitry Namiot. ≪Time series databases≫. Em: *Proceedings of the XVII International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL)*. Vol. 1536. 2015, pp. 132–137.

[20] Ted Dunning and Ellen Friedman. *Time Series Databases New Ways to Store and Access Data*. O'Reilly Media, 2014, p. 72. ISBN: 9781491917022.

[21] *TimescaleDB Docs — What Is Time-series Data?* URL: https://docs.timescale.com/v1.0/introduction/time-series-data (acedido em 25/09/2018).

[22] *Timescale*. 2017. URL: https://www.timescale.com/ (acedido em 09/12/2017).

[23] *TimescaleDB vs. InfluxDB: purpose built differently for time-series data*. URL: https://blog.timescale.com/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877 (acedido em 27/09/2018).

[24] Basho. *Time Series Databases*. URL: http://basho.com/resources/time-series-databases/ (acedido em 06/09/2018).

[25] Sean Rhea et al. ≪LittleTable: A Time-Series Database and Its Uses≫. Em: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD '17. Chicago, Illinois, USA: ACM, 2017, pp. 125–138. ISBN: 978-1-4503-4197-4. URL: http://doi.acm.org/10.1145/3035918.3056102.

[26] Tuomas Pelkonen et al. ≪Gorilla: A Fast, Scalable, In-memory Time Series Database≫. Em: *Proc. VLDB Endow.* 8.12 (ago. de 2015), pp. 1816–1827. ISSN: 2150-8097. URL: http://dx.doi.org/10.14778/2824032.2824078.

[27] Michael P. Andersen and David E. Culler. ≪BTrDB: Optimizing Storage System Design for Timeseries Processing≫. Em: *Proceedings of the 14th Usenix Conference on File and Storage Technologies*. FAST'16. Santa Clara, CA: USENIX Association, 2016, pp. 39–52. ISBN: 978-1-931971-28-7. URL: http://dl.acm.org/citation.cfm?id=2930583.2930587.

[28] *Hadoop*. URL: http://hadoop.apache.org/ (acedido em 24/05/2018).

[29] Akshit Dhar. ≪Big Data Technologies for Batch and Real-Time Data Processing: A Review Google File System (GFS), Hadoop Distributed File System (HDFS), Relational Database Management Systems (RDBMS), Resilient Distributed Datasets (RDDs)≫. Em: *International Journal of Engineering Science and Computing* (2017).

[30] Himani Saraswat et al. ≪Enhancing the Traditional File System to HDFS: A Big Data Solution≫. Em: *International Journal of Computer Applications* 167.9 (2017), pp. 975–8887.

[31] Wei Wei et al. ≪A Study of SQL-on-Hadoop Systems≫. Em: *Big Data Benchmarks, Performance Optimization, and Emerging Hardware* 8807 (2014), pp. 209–220. ISSN: 16113349.

[32] Konstantin Shvachko et al. *The Hadoop distributed file system*. Rel. téc. 2010, pp. 1–10.

[33] Ren Li et al. ≪MapReduce Parallel Programming Model: A State-of-the-Art Survey≫. Em: *Int. J. Parallel Program.* 44.4 (ago. de 2016), pp. 832–866. ISSN: 0885-7458. URL: http://dx.doi.org/10.1007/s10766-015-0395-0.

[34] *Ambari*. URL: https://ambari.apache.org/ (acedido em 16/07/2018).

[35] *Apache Cassandra*. URL: http://cassandra.apache.org/ (acedido em 06/09/2018).

[36] *Apache HBase*. URL: https://hbase.apache.org/ (acedido em 06/09/2018).

[37] *Apache Hive*. URL: https://hive.apache.org/ (acedido em 16/07/2018).

[38] *Apache ZooKeeper*. URL: https://zookeeper.apache.org/ (acedido em 06/09/2018).

[39] TimescaleDB. *TimescaleDB: SQL made scalable for time-series data*. 2017.

[40] Michael J Freedman. *Re-engineering PostgreSQL as a time-series database*. 2018. URL: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=ZSP04803USEN& (acedido em 30/10/2018).

[41] *Apache Kudu*. URL: https://kudu.apache.org/ (acedido em 13/07/2018).

[42] Todd Lipcon et al. *Kudu : Storage for Fast Analytics on Fast Data*. Rel. téc. 2015. URL: https://kudu.apache.org/kudu.pdf.

[43] Ryan Bosshart et al. *Getting Started with Kudu*. O'Reilly Media, 2018, p. 156. ISBN: 978-1-491-98025-5.

[44] Diego Ongaro and John Ousterhout. ≪In Search of an Understandable Consensus Algorithm≫. Em: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX ATC'14. Philadelphia, PA: USENIX Association, 2014, pp. 305–320. ISBN: 978-1-931971-10-2. URL: http://dl.acm.org/citation.cfm?id=2643634.2643666.

[45] *Apache Avro*. URL: https://avro.apache.org/docs/1.8.2/ (acedido em 13/07/2018).

[46] *Apache Parquet*. URL: https://parquet.apache.org/documentation/latest/ (acedido em 13/07/2018).

[47] Jay Kreps, Neha Narkhede and Jun Rao. ≪Kafka: a Distributed Messaging System for Log Processing≫. Em: *ACM SIGMOD Workshop on Networking Meets Databases* (2011), p. 6.

[48] Xiufeng Liu, Nadeem Iftikhar and Xike Xie. ≪Survey of Real-time Processing Systems for Big Data≫. Em: *Proceedings of the 18th International Database Engineering & Applications Symposium*. IDEAS '14. Porto, Portugal: ACM, 2014, pp. 356–361. ISBN: 978-1-4503-2627-8. URL: http://doi.acm.org/10.1145/2628194.2628251.

[49] *Apache Kafka*. URL: https://kafka.apache.org/ (acedido em 13/07/2018).

[50] *Apache Spark™ - Unified Analytics Engine for Big Data*. URL: https://spark.apache.org/ (acedido em 09/08/2018).

[51] Sergey Melnik et al. ≪Dremel: Interactive Analysis of Web-scale Datasets≫. Em: *Proc. VLDB Endow.* 3.1-2 (set. de 2010), pp. 330–339. ISSN: 2150-8097. URL: http://dx.doi.org/10.14778/1920841.1920886.

[52] *Apache Hue*. URL: http://gethue.com/ (acedido em 13/07/2018).

[53] *Apache Impala*. URL: https://impala.apache.org/ (acedido em 13/07/2018).

[54] Marcel Kornacker et al. ≪Impala: A Modern, Open-Source SQL Engine for Hadoop≫. Em: *In Proc. CIDR'15*. 2015.

[55] *Apache Presto*. URL: https://prestodb.io/ (acedido em 13/07/2018).

[56] *Apache Drill*. URL: https://drill.apache.org/ (acedido em 13/07/2018).

[57] Andreas Bader, Oliver Kopp and Michael Falkenthal. ≪Survey and Comparison of Open Source Time Series Databases≫. Em: *Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband*. Ed. por Bernhard Mitschang et al. Gesellschaft für Informatik e.V., 2017, pp. 249–268.

[58] Soren Kejser Jensen, Torben Bach Pedersen and Christian Thomsen. ≪Time Series Management Systems: A Survey≫. Em: *IEEE Transactions on Knowledge and Data Engineering* (2017), p. 1. ISSN: 1041-4347. eprint: 1710.01077.

[59] *Using Impala through a Proxy for High Availability | 5.14.x | Cloudera Documentation*. URL: https://www.cloudera.com/documentation/enterprise/5-14-x/topics/impala_proxy.html (acedido em 09/10/2018).

[60] *PrestoDB- Kudu Connector*. URL: https://prestodb.io/docs/current/connector/kudu.html (acedido em 02/10/2018).

[61] *T-Test*. URL: https://www.investopedia.com/terms/t/t-test.asp (acedido em 02/10/2018).