PM

Abordagem *Big Data* a dados de mobilidade em transportes públicos

PROJETO DE MESTRADO

Fábio Tolentino Henriques Pita
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

setembro | 2022

Abordagem *Big Data* a dados de mobilidade em transportes públicos

PROJETO DE MESTRADO

Fábio Tolentino Henriques Pita
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Leonel Domingos Telo Nóbrega

Resumo

A necessidade de armazenar, processar e analisar os dados é uma realidade cada vez presente nas empresas onde as decisões de negócio dependem muito das plataformas digitais. A introdução do conceito de *Data Warehouse* teve como finalidade facilitar e melhorar o processo de recolha de indicadores de negócio imprescindíveis.

O conceito de *Big Data* veio com o aumento da variedade e do volume de dados para fins de análise. Com esse conceito em mente, foram desenvolvidas tecnologias para fazerem face aos desafios impostos. A transformação digital no registo de entradas e saídas nos transportes público leva a grandes volumes de dados que podem ser usados para aplicar análises de negócio na área [1].

Este projeto visa a recolha de um conjunto de tecnologias na vertente do *Big Data* e a avaliação da capacidade de armazenamento, do método de elaboração dos métodos de *ETL* e do desempenho na obtenção de resposta a um conjunto de *queries*, consoante o aumento do volume de dados de mobilidade, referentes às entradas dos autocarros da companhia de transportes públicos Horários do Funchal.

É introduzida neste projeto uma revisão de literatura sobre os conceitos de *Data Warehouse*, de modelos dimensionais e de *Big Data*, bem como nas tecnologias existentes e trabalhos relacionados com manipulação de *Big Data.* Foi também objeto de análise do estado da arte a aplicação destas tecnologias nos transportes públicos.

Os resultados apresentados revelam que algumas plataformas conseguem adequar-se bem ao um aumento do volume de dados, com boas capacidades de desempenho, tanto na execução de processos de *ETL*, como na execução de *queries* de consulta, em comparação a outras tecnologias, cujo resultados são pouco práticos neste tipo de estudo.

Palavras-chave:

Data Warehouse, Big Data, Modelo Dimensional, Apache Hadoop, Apache Hive, Apache Spark, Presto, SQL Server, ETL, Docker, Transporte Público, AFC

Abstract

The need to store, process and analyse data is a increasingly present reality in companies where business decisions depend heavily on digital platforms. The purpose of introducing the Data Warehouse concept was to facilitate and improve the process of collecting essential business indicators.

The concept of Big Data came with the increase in the variety and the volume of data for analysis purposes. With the concept in mind, technologies were developed to face the imposed challenges. The digital transformation in the registration of entrances and exits in the public transport lead to large volumes of data that can be used to apply business analysis [1].

This project aims to collect a set of technologies in the field of Big Data and evaluate the storage capacity, the method of elaborating ETL methods and the performance in obtaining a response to a set of queries, referring to the entrances of the buses of public transport company Horários do Funchal.

This project introduces a literature review on the concepts of Data Warehouse, dimensional models and Big Data, as well as existing technologies and work related to Big Data manipulation. The application of these technologies in public transport was also subject to a state-of-the-art analysis.

The presented results reveal that some platforms are able to adapt well to an increase in the volume, with good performance capabilities, both in the execution of ETL processes and in the execution of queries, in comparison to other technologies, whose results are impractical in this type of study.

Keywords:

Data Warehouse, Big Data, Dimensional Model, Apache Hadoop, Apache Hive, Apache Spark, Presto, SQL Server, ETL, Docker, Public Transport, AFC

Agradecimentos

Em primeiro lugar, gostaria de agradecer à companhia Horários dos Funchal por disponibilizar os dados para este projeto.

Agradecer a todos os docentes que participaram no meu percurso académico desde a licenciatura até esta fase final. Agradeço em especial ao meu orientador Professor Leonel Domingos Telo Nóbrega, pela sua disponibilidade ao longo deste projeto, mas também pela sua vontade de me ensinar e introduzir-me ao mercado de trabalho.

Um agradecimento aos meus amigos e colegas de curso pelo apoio durante a minha vida académica.

Um agradecimento em especial aos meus pais e à minha irmã pelo apoio que me têm dado ao longo da minha vida, incentivando em acabar os meus estudos.

Por último agradecer à minha namorada, que, sem a força de vontade me incutiu, provavelmente não iria concluir este trabalho e a minha formação académica.

Índice

Índice de Figuras:

Índice de Tabelas

Lista de Abreviaturas

AQE: Adaptive Query Execution

AFC: Automatic Fare Collection

BI: Business Inteligence

CLI: Command-line interface

DW: Data Warehouse

ETL: Extract, Transform and Load

GTFS: General Transit Specification File

IoT: Internet of Things

JDBC: Java Database Connectivity

OBDC: Open Database Connectivity

OLAP: Online analytical processing

OLTP: Online Transaction Processing

RDD: Resilient Distributed Datasets

1 Introdução

Num mundo em que a tecnologia virtual está mais enraizada no dia a dia, desde o consumo das redes sociais até à utilização dos sistemas *Internet of Things* (*IoT*), o tratamento e a análise de milhões de dados constantemente gerados são dos grandes desafios que a maioria das empresas enfrenta.

Desde a década de 1990, com a introdução das bases de dados relacionais e a sua aplicação em sistemas transacionais, houve a necessidade de aplicar esses dados guardados para tomadas de decisão nas empresas. Com as representações de dados existentes, a obtenção de resultados tornou-se pouco prático e obsoleto. Devido a essas problemáticas desenvolveram conceitos de *Data* Warehouse (DW)[2].

No seu desenvolvimento, levou à criação de arquiteturas e de infraestruturas para fazerem face às necessidades de negócio das empresas e à introdução e aplicação dos modelos dimensionais (das tabelas de facto e de dimensão) e conceitos de *ETL*.

O aumento do volume e da variedade dos dados históricos recolhidos levou à introdução do termo *Big Data*. Os sistemas na área do *Big Data* deverão conseguir armazenar grandes volumes de dados, provenientes de várias fontes e formatos, mas também poder processar essas grandes quantidades para obter indicadores de negócio com melhor desempenho. Desde o seu aparecimento, foram desenvolvidos conjuntos de ferramentas e plataformas para corresponder aos desafios apresentados.

Na fase da digitalização na área dos transportes públicos, a aplicação de sistemas *Smart Card* e *AFC* (*Automatic Fare Collection*) tornou-se o ponto de partida na aplicação de análise preditiva, com a utilização de dados de mobilidade. Esses sistemas recolhem grandes volumes de dados todos os dias através da validação de entradas e saídas dos veículos por parte dos clientes. Esses dados ajudam a melhorar o serviço tornando-o benéfico para a empresa e para o cliente [3].

1.1 Identificação do problema

Como foi referido acima, o registo das entradas e saídas dos passageiros dos transportes públicos implica um grande volume dados para análise. Além da infraestrutura física, a escolha da tecnologia tem um papel determinante no desempenho nos processos de transformação e análise. Para o estudo de caso da companhia Horários do Funchal é necessário ser feita a recolha de um conjunto de tecnologias e avaliar qual é o que melhor se adequa.

1.2 Objetivos do estudo

Com a identificação do problema referido na secção anterior, foram propostos os seguintes objetivos este estudo:

- Recolha de informação sobre plataformas existentes no mercado na área do *Big Data*;
- Selecionar um conjunto de plataformas para análise;
- Desenvolver e simular ambiente de *Big Data*;
- Implementar processos de transformação de *ETL*;

- Conferir o desempenho do processo de transformação consoante o volume de dados;
- Comparação do desempenho no resultado das *queries* de negócio na área dos transportes públicos consoante o volume dos dados.

1.3 Estrutura do trabalho

Este trabalho é composto pela seguinte estrutura:

1. Revisão de Literatura: sobre os conceitos de *Data Warehouses*, modelos dimensionais e arquiteturas implementadas. Na parte do *Big Data* são referidas as plataformas existentes no mercado. Também se realizou uma revisão de trabalhos científicos no âmbito do *Big Data* e dos transportes públicos;
2. Análise ao problema: explicar o processo sequencial do trabalho desenvolvido, desde a recolha de dados para fins de teste até à escolha de indicadores de negócio;
3. Desenvolvimento: do projeto, do planeamento à criação do ambiente de testagem. Desenvolvimento dos processos *ETL* à criação das *queries* propostas para os indicadores de negócio;
4. Avaliação: comparativa das tecnologias em estudo, desde a sua capacidade de armazenamento, ao desempenho da execução dos processos *ETL*, até à avaliação do respetivo desempenho na obtenção dos indicadores de negócio;
5. Conclusão: obtida deste estudo, no cumprimento dos objetivos propostos e no que poderá ser realizado em trabalho futuro.

2 Revisão de Literatura

2.1 *Data Warehouse*

Desde o desenvolvimento de aplicações com fins transacionais e a introdução dos modelos relacionais para a gestão de dados na década de 70, muitas empresas adotaram a utilização dessas tecnologias para a facilitar a gestão e aquisição de dados [4]. A partir da década de 90, houve a necessidade de utilizar os dados históricos para realizar análises de negócio, desta forma, permitindo a criação de processos de apoio às decisões de negócio. Contudo, as bases de dados transacionais, as *OLTP*, eram desenhadas para operações do dia a dia, estando inaptas para *queries* complexas, que requeriam a utilização de várias tabelas e operações agregações de grandes volumes de dados. Essa necessidade levou à criação e desenvolvimento do conceito de *Data Warehouse* (DW) [2].

Uma *DW* visa o armazenamento de grandes volumes de dados e integrá-los com o fim de obter de informações uteis para o negócio. *Inmon* ([2], [5]), refere que uma *Data Warehouse* tem de corresponder à seguinte composição:

- Orientada ao sujeito: dedicada às necessidades de análise em diferentes áreas de negócio;
- Capacidade de Integração: poder agregar todos os dados, independente das origens;
- Não volátil: capacidade de os dados serem históricos e impedidos de serem removidos ou alterados;
- Variável ao tempo: ter vários valores para a mesma informação, devendo-se ao facto que longo do tempo haja mudanças na empresa que consequentemente leva a alterações de informação.

Sendo que uma das finalidades de uma *DW* é o armazenamento de grandes volumes de dados, muitas empresas enfrentam o desafio de manter as infraestruturas de suporte, assim como a possibilidade de atingir os limites de disponibilidade de recursos que essas infraestruturas, em geral, possuem. A utilização de *Data Marts* vem ajudar a contornar esses problemas através da separação do volume ([2], [6]). Num exemplo prático da sua utilização, uma empresa tem uma secção de vendas que analisa só os dados das vendas realizadas num certo período, ao passo que os Recursos Humanos necessitam dos dados dos empregados permanentemente [2].

As abordagens de implementação de uma *DW* são dependentes dos recursos, da velocidade, das necessidades e do volume de dados disponíveis às organizações. A abordagem *top-down* é uma visão mais complexa e requer melhor planeamento. O processo começa na recolha de todas as necessidades dos utilizadores, elaboração de um esquema único e na derivação em vários *Data Marts*, consoante as exigências. Sendo uma abordagem que necessita de muito tempo e planeamento, a sua adoção é pouco abordada pela maioria das organizações ([2], [6]). Um exemplo de aplicação de abordagem é a de *Inmon* [5].

A abordagem *bottom-up* é a alternativa mais adotada pelas organizações. Envolve uma perspetiva de criar esquemas diferentes para cada *Data Mart*, que no fim se unem. Esta

implementação tem a vantagem de obter respostas mais rápidas com recursos limitados, mas podem ocorrer inconsistências e redundância ([2], [6]). Arquitetura proposta por *Kimball* [8] é um exemplo de utilização da abordagem *bottom-up* [7].

2.1.1 Modelo Dimensional

Usar técnicas de desenho de uma base de dados transacional numa *DW* aumenta a sua complexidade, havendo muitas tabelas e relações entre elas. Esta complexidade, vem da necessidade de normalizar as tabelas para minimizar a redundância de dados, importante para ambientes transacionais onde ocorrem inúmeras transações, mas irrelevante num ambiente de *DW* [9].

A utilização de um modelo dimensional tem como fim reduzir a complexidade das bases de dados, diminuindo o número de tabelas e de relações, para que o utilizador final possa usufruir com facilidade e rapidez da obtenção de resultados ([2], [8], [9]).

O modelo dimensional é composto por ([2], [6], [8]):

- Tabela de Factos: onde é armazenado grande volume de dados que correspondem a dados de contexto e medidas para análise (exemplo: total de vendas);
- Tabelas de Dimensão: contêm informações contextuais para análise. Essas tabelas de dimensão contêm chaves primárias que se relacionam com as tabelas de Factos. Podem ser compostas por hierarquias de análise, exemplo: tabela de dimensão data pode ter a seguinte hierarquia: Ano > Mês> Dia.

Os desenhos dos esquemas mais usados no desenvolvimento do Modelo Dimensional são organizados em estrela ou em floco de neve. Os esquemas em estrela (Figura 1) são desenhados com uma tabela de factos central, interligada com várias tabelas de dimensão, sendo a sua apresentação semelhante a uma estrela. Isto tem vantagem na diminuição do número de relações entre tabelas, que, consequentemente, reduz na complexidade na obtenção de informação. Os esquemas em floco de neve (Figura 2) são semelhantes aos em estrela, mas contêm tabelas de dimensão normalizadas, evitando a replicação de dados. Têm, porém, a desvantagem de se tornarem mais complexos do que a representação em estrela ([6], [9], [10]).

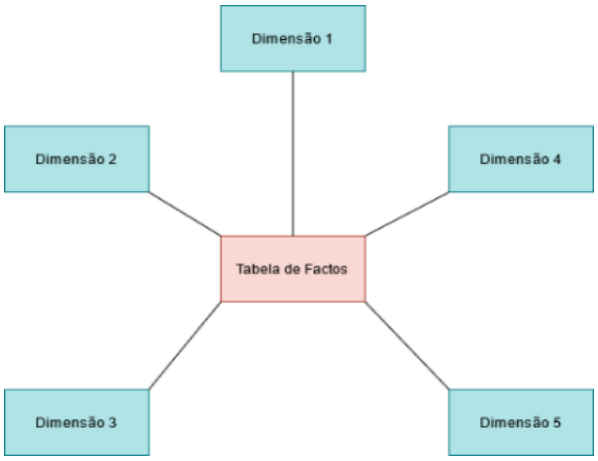


Figura 1. Exemplo de um esquema em estrela

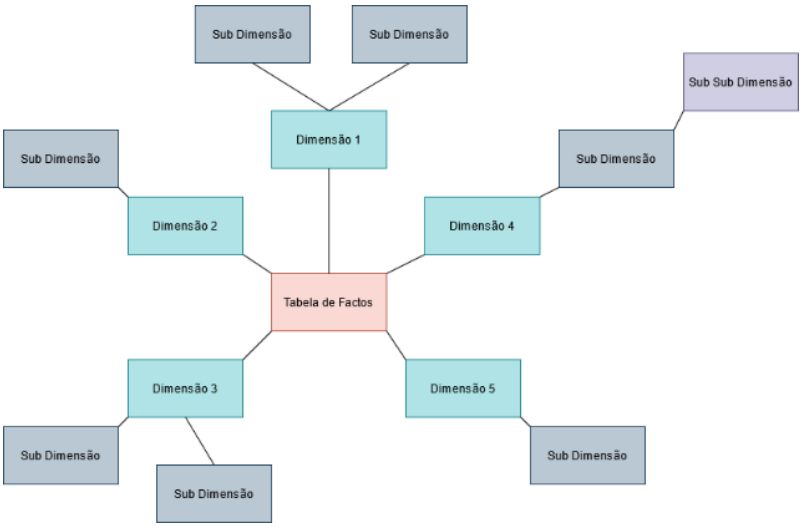


Figura 2. Exemplo do modelo representado floco de neve.

2.1.2 Arquiteturas de *Data Warehouse*

No desenvolvimento de uma arquitetura para uma *DW*, *Kimball* [8] sugere que tem de conter os seguintes elementos: sistemas de origem operacional, ferramentas de *ETL* (Extract, *Transform and Load*), área de apresentação *Data Warehouse* e aplicações de *BI* (*Business Intelligence*).

Nos sistemas de origem operacional, são registadas transações de negócio que ficam fora das *DW*. São sistemas focados em velocidade e disponibilidade, mantendo o seu registo histórico mais baixo [8].

Ferramentas de *ETL* são a porta de entrada entre os sistemas de origem operacional com a área de apresentação. Esses sistemas são formados por estas seguintes etapas [8], [10]:

- Extração (*Extract*): é a primeira etapa, nela retiram-se os dados de várias bases de dados e de vários ficheiros;
- Transformação (*Transform*): após a extração, os dados passam por um processo de transformações para a integração. Podem ser a limpeza de erros ou inconsistências, substituição por dados já predefinidos, integração de dados de várias fontes e de outras funções, que melhoram a qualidade dos dados;
- Carregamento (*Load*): nesta última etapa todos os dados transformados são carregados para a *DW*, na área de apresentação para o seu respetivo modelo dimensional;

Na área de apresentação é onde se situam-se os dados históricos, com grande nível de detalhe, organizados pelos seus modelos dimensionais, e disponíveis para leitura e análise utilizando aplicações de *BI* [8]. Esta área contém um conjunto de várias *Data Marts*, cujos modelos dimensionais têm tabelas de dimensão capazes de partilhar com várias tabelas de factos. Isto possibilita a geração de novos modelos dimensionais e reduzindo a replicação de tabelas de dimensão, graças à existência de comunicação entre as *Data Marts* ([8], [11]).

As aplicações *BI* são aplicações onde o utilizador consegue recolher os dados da área de apresentação por meio de *querys* ou através de aplicações mais sofisticadas, em que é possível gerar gráficos e outras funções para complementar o raciocínio analítico [8].

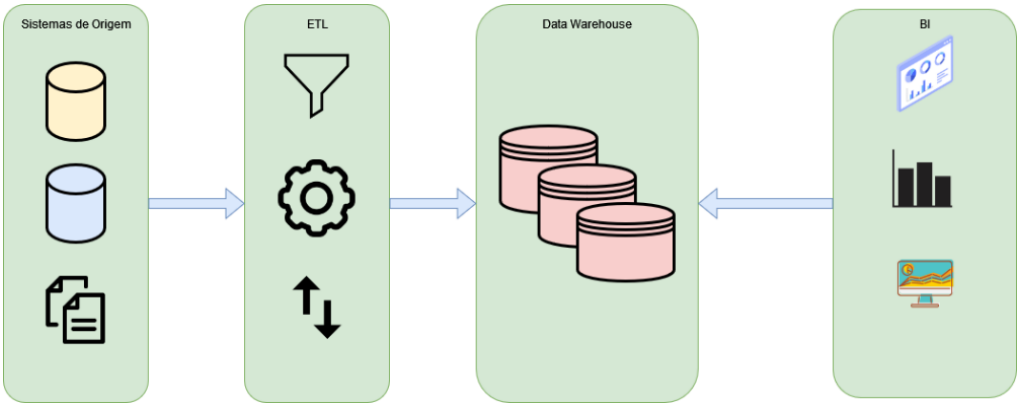


Figura 3. Exemplo de proposta de arquitetura por Kimball [8].

2.1.2.1 Arquiteturas alternativas

Desde a adaptação dos conceitos de *Data Warehouse* na indústria tem havido exemplos de arquiteturas alternativas em comparação à proposta de *Kimball*. A escolha de alternativas pesa-se no facto de serem mais adequadas às necessidades de cada negócio.

2.1.2.1.1 *Data Marts* independentes

A utilização desta arquitetura vem com a finalidade que os dados sejam exportados, integrados e carregados para *Data Marts* independentes, organizadas por um dado grupo ou departamento sem conhecimento de outras adjacentes. É vantajoso pela simplicidade de implementação, pouca utilização de recursos e maior velocidade em obtenção de respostas para análise de negócio. Sem a comunicação com os outros *Data Marts*, a longo termo, tornará ineficiente com a redundância de dados vindos das mesmas fontes ([6], [8]).

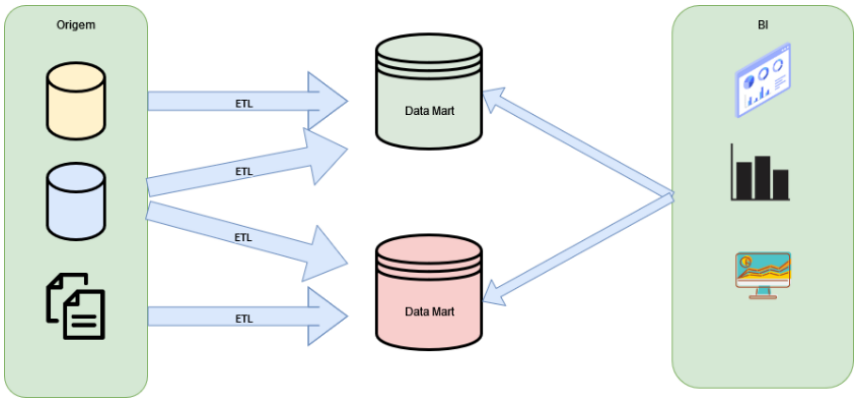


Figura 4. Exemplo de uma arquitetura de Data Marts independentes. [8]

2.1.2.1.2 Arquitetura *Inmon*

A alternativa proposta por *Inmon* difere da de *Kimball* em vários de aspetos. Todas as fontes de dados são extraídas, tratadas e carregadas numa *Data Warehouse* geral por

sistemas *ETL*. Após o carregamento, os dados são derivados consoante a necessidade das secções das empresas e carregados nas *Data Marts* exclusivas de cada secção ([5], [8], [11]). *Inmon* considera que a *DW* é fisicamente separada com as *Data Mart*, porque está orientada ao armazenamento e adaptada a novos requisitos, em comparação com as *Data Marts*, que estão mais orientadas aos requisitos do utilizador [11].

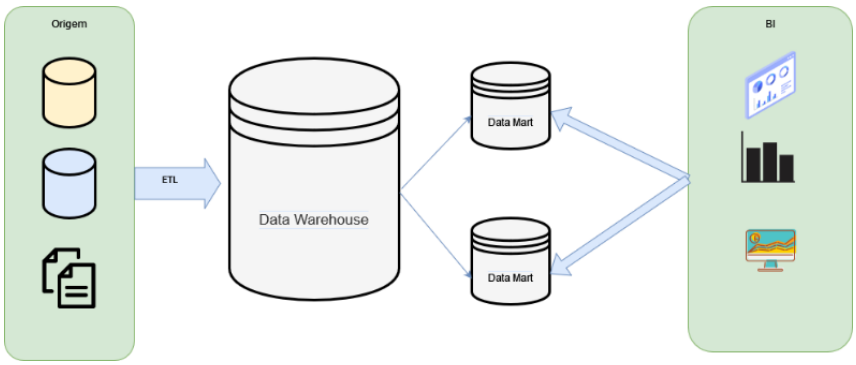


Figura 5. Exemplo de uma arquitetura proposta por Inmon [8].

2.1.3 Conclusão

Com o aparecimento das bases de dados transacionais, a geração de volumes de dados e a necessidade de analisá-los levou a criação dos conceitos de *Data Warehouse*.

As abordagens *top-down* e *bottom-up* são designadas na implementação das *Data Warehouse*s. A implementação por *top-down* é mais complexa do que a *bottom-up*, que resulta numa fraca adoção pelas organizações. Por sua vez, a *bottom-up* é uma implementação mais rápida e traz vantagens ao nível de desempenho na obtenção de resultados de análise, mas com um planeamento descuidado, a longo prazo, pode ocorrer redundância de dados. Exemplos de utilização das abordagens são arquiteturas propostas por *Kimball*, *bottom-up*, e *Inmon*, *top-down*.

Pela visão de *Kimball*, as *Data Warehouse*s conseguem armazenar os dados providentes de várias fontes e de tipos. Esses dados têm de ser tratados e integrados através das ferramentas *ETL* e, por fim, carregados para o seu modelo dimensional. Esses modelos são compostos por tabelas de factos e por tabelas de dimensão e são representados através esquemas em estrela ou em floco de neve. Os modelos dimensionais podem estar armazenados em várias *Data Marts* que se comunicam entre si, capazes de partilhar informação e gerar novos esquemas. A visão de *Inmon*, é uma alternativa face à de *Kimball*. *Inmon* refere que todos os dados devem ser tratados e carregados para uma *Data Warehouse* única e derivar-se utilizando os *Data Mart*s.

A utilização dessas arquiteturas tem a sua relevância na implementação por meio de ambientes onde são utilizadas bases de dados tradicionais. Devido ao aumento do volume e variedade de dados produzidos diariamente, originou-se a necessidade de fazer o processamento paralelo dos dados e a introdução de arquiteturas *Big Data* [12].

2.2 *Big Data*

A comunidade científica não tem uma clara definição sobre o conceito *Big Data*. Numa revisão de literatura, Andrea de Mauro et al.[13] concluiu que este conceito se traduz-se na informação armazenada em grande volume, na velocidade de acesso e na variedade dos mesmos. De tal modo, necessita de um conjunto de tecnologias e metodologias analíticas para fazer face aos desafios [13]. O aparecimento do termo *Big Data* vem da existência das redes sociais e da *Internet of Things* (*IoT*), que revertem na vasta geração de dados não estruturados[14]. *Big Data* é caracterizado essencialmente pelos 3 *V's*, Volume, Velocidade e Variedade ([14], [15], [16]), mas estendido para 7 *V's*[17].

O Volume refere-se à geração excessiva de dados, não só textuais, como de outros formatos (vídeos, música, etc.), sendo necessário a criação de tecnologias próprias que acompanhem o seu crescimento. A Velocidade traduz-se na necessidade do armazenamento e processamento em tempo real. A Variedade, capaz de aceitar qualquer tipo de dados([14], [16], [17]). O quarto V é a Variabilidade, corresponde à mudança constate dos dados. O quinto V é a Veracidade, traduz-se na qualidade dos dados, necessidade de remover anormalidades que compromete a sua qualidade. O sexto V é a Visualização, que relaciona com a apresentação dos dados ao utilizador por gráficos ou esquemas. O sétimo V é o Valor, isto é, as conclusões que são possíveis de extrair em prol da empresa [17].

A utilização das ferramentas do *Big Data* tem um papel fundamental em várias áreas da nossa sociedade [16]:

- Saúde Pública: tornar os serviços mais personalizados para cada tipo de população, melhorar a qualidade e o custo;
- *IoT*: otimizar a logística das empresas através da criação de rotas alternativas pela recolha de dados do GPS;
- Transporte público: através da quantidade de validações por parte do passageiro é possível as empresas criar rotas alternativas e aumentar a frequências em alturas mais apropriadas.

Com a necessidade de armazenar, processar e analisar dados de vários formatos e de grandes dimensões, foram desenvolvidas ferramentas *open-source* como o *Hadoop* e *Spark*.

2.3 Tecnologias

2.3.1 Bases Dados Relacionais e Não Relacionais

Nos tempos iniciais do desenvolvimento desta área, os armazenamentos dos dados eram feitos por meio de documentos. O aumento do seu volume levou a que este método se tornasse pouco eficiente e provocou o aumento o nível de manutenção dos mesmos. A introdução das bases de dados relacionais impulsionou ao armazenamento dos dados através da aplicação de tabelas relacionais. Cada registo é guardado em cada linha e pode ser relacionado com outras tabelas [18]. Exemplos de bases de dados relacionais existentes no mercado são o *SQL Server* e *MySQL*.

As bases de dados não relacionais começaram a ser uma alternativa às bases de dados relacionais, por ser mais simples de desenhar e de escalar para vários nós em comparação

às relacionais. Também há um aumento da sua utilização em aplicações de *Big Data*. De momento, existe um conjunto de vários tipos [18]:

- *Document* (ex. MongoDB);
- *Key value* (ex. Cassandra);
- *Graph* (ex. Neo4j);
- *Wide Column* (ex. HBase).

Ambos os sistemas de base de dados contém a suas vantagens e desvantagens [18]:

Base de Dados Relacional	Base de Dados Não Relacional
Vantagens: • Fácil de executar e manter; • Integridade dos Dados. Desvantagens: • Pode conter custos monetários; • Limitações de estrutura; • Pode tornar-se lento em certas aplicações.	Vantagens: • Rápido e flexível; Desvantagens: • Pouco suporte; • Pouca integridade nos dados.

2.3.2 Hadoop

Hadoop é uma *framework* desenvolvida para lidar com os desafios do *Big Data*. Consegue transpor os limites de recursos do armazenamento através do seu sistema distribuído de ficheiros, *HDFS*, e o processamento de dados em paralelo através do seu paradigma de programação *MapReduce* ([14], [19], [20]). *Hadoop* é muitas vezes considerado *Schema-on-Read*, referindo que todos os dados não processados podem ser guardados sem restrições, ao contrário dos sistemas tradicionais de base de dados, que necessitam ser definidos antes de ser submetidos. Esses sistemas são referidos como *Schema-on-Write* [21].

A utilização do *Hadoop* traz a vantagem em permitir o armazenamento de dados provenientes de diferentes fontes e de diferentes formatos, poupando tempo aos utilizadores. Dispõe um acesso rápido aos dados armazenados através da divisão em pequenos blocos e a capacidade de replicar os dados armazenados pelos nós do agrupamento, evitando perdas em caso de falhas pontuais[22].

Com o vasto uso deste instrumento e com aparecimento de novas necessidades das empresas, além dos avanços tecnológicos, desenvolveram-se novas ferramentas que se complementam com o *Hadoop*, fazendo parte do seu ecossistema, sendo as mais referenciadas o *Pig*, *Hive*, *HBase*, *Sqoop*, *Flume* e *Zookeeper* ([14], [20]).

2.3.2.1.1 Hadoop Distributed File System (HDFS)

O sistema de ficheiros do *Hadoop* (Figura 6) segue um padrão arquitetural mestre-escravo para de modo a facilitar o armazenamento de dados distributivamente pelo agrupamento, através da divisão em pequenos blocos. A arquitetura *HDFS* é composta pelo nó mestre, *Namenode*, e pelos nós escravos, *Datanodes*([14], [16], [23]).

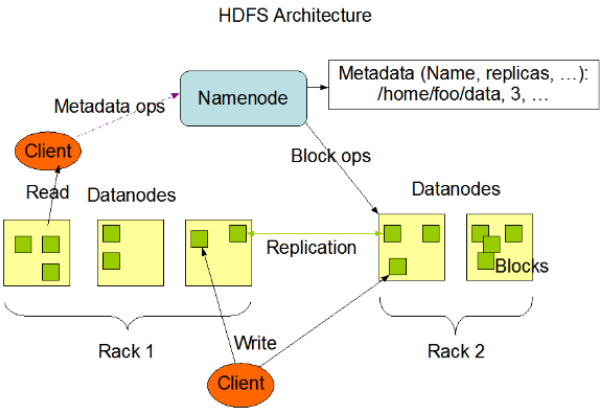


Figura 6. Esquema de arquitetura HDFS [24].

O *Namenode*, o nó mestre da arquitetura, é onde são armazenados os metadados do sistema. Da informação dos blocos que compõem os ficheiros, bem como a sua localização no seu agrupamento. Este dá acesso aos clientes para as operações de leitura, de escrita e de execução de programas de *MapReduce* [25].

No *Datanode*, o nó escravo da arquitetura, é feito o armazenamento por pequenos blocos. Mantém-se em constante comunicação com *Namenode*, dando informações sobre o seu estado e o dos blocos armazenados [25]. Quando o cliente faz um pedido de acesso a um ficheiro, o *Namenode* responde ao cliente quais os *Datanodes* onde contêm os blocos referentes [14].

Em situações de falhas, o *HDFS* consegue replicar os blocos existentes para outras *racks* disponíveis, gerindo de forma mais rápida e viável. A existência de um *Namenode* secundário têm com a função de, periodicamente, fazer backup dos metadados que, em caso de falha, podem ser restaurados [25].

2.3.2.2 MapReduce

É um modelo de processamento de dados em paralelo, desenvolvido pela *Google* [26], e tornou-se uma componente importante no tratamento de grande volume de dados, essencial na tecnologia *Hadoop*. O objetivo resume-se na capacidade de receber uma quantidade de dados e processá-los paralelamente em pequenos blocos, resultando numa maior eficiência de obtenção de resultados [14], [16], [23]. A estrutura do *MapReduce* é composta por ([16], [21], [27]):

- Fase de *Map*: os dados são recebidos pelos *Mappers* e aplicam tarefas de transformação e filtragem, resultando em pares de *Key/Value*;
- Fase de *Sort* e *Shuffle*: nesta fase, recebem-se os pares *Key/Value* que são organizados e baralhados, como base do valor da sua *Key*;
- Fase de *Reduce*: os pares que passaram pela fase anterior, são recebidos pelos *reducers*, que operam várias funções de agregação resultando de novos pares *Key/Value* mais reduzido.

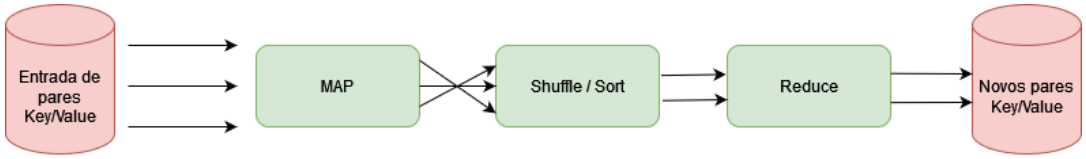


Figura 7. Exemplo do processo MapReduce [27].

MapReduce é um complemento para as limitações dos sistemas de Base de Dados Relacionais no armazenamento e na análise de grandes volumes. Aplicação em operações de atualização e inserção são mais indicadas para as Bases de Dados relacionais. Já *MapReduce* é indicado em aplicações onde os dados são armazenados uma só vez, mas acedidos várias vezes. Na tabela abaixo indica as suas diferenças [28], [29]:

Tabela 1. Diferenças entre MapReduce e Bases de dados Relacionais

	Bases de Dados Relacionais	*MapReduce*
Escalabilidade	Reduzida, para um número limitado de nós do agrupamento	Alta, capaz de criar grandes agrupamentos
Capacidade dos Dados	*GigaBytes*	*Petabytes*
Falha de tolerância	Baixa	Alta
Suporte de esquema	Sim	Não
Integridade	Alta	Baixa

MapReduce é uma ferramenta com muito potencial para processamento de grandes volumes de dados em paralelo, mas tem as suas desvantagens. Durante a sua implementação e manutenção é necessário ter os conceitos de *Map* e *Reduce* bem presentes, resultando num desenvolvimento de soluções mais demoradas e sujeitas a falhas. Na aplicação do *MapReduce* no *Hadoop* é corrente, entre as operações, escrever resultados intermédios no disco, de modo a ser recuperável em situações onde ocorrem falhas de tolerâncias, resultando na degradação de desempenho [21].

O *MapReduce* no *Hadoop* (Figura 8) executa em paralelo com os nós do *HDFS*. Quando o utilizador executa um programa *MapReduce*, o *Namenode* inicializa o *JobTracker*, criando tarefas e distribuindo pelos nós escravos, os *TaskTrackers*. Além da gestão de tarefas, o *JobTracker* tem a função de gerir na atribuição de recursos necessários que cada *TaskTracker* pode utilizar. Cada *TaskTracker*, executa um conjunto de várias tarefas (*map* ou *reduce*) propostas em paralelo e faz o seu papel de monitorização dos estados em que se encontram e comunicando-os ao *JobTracker* [16], [30].

Esta utilização causa um conjunto de limitações, tais como [30]:

- Número reduzido de nós para cada agrupamento, obrigando a criação e manutenção de mais agrupamentos;
- Existindo um único *JobTracker*, em caso falha é necessário reiniciar o processo todo;
- O número de o *map* e *reduce* para cada *TaskTracker*, levando a uma má gestão dos recursos;
- Exclusividade para aplicações de *MapReduce*;

Tendo em conta com estas limitações, foram necessárias novas soluções. Com o desenvolvimento da nova versão do *Hadoop*, foi introduzido uma nova componente, o *YARN*.

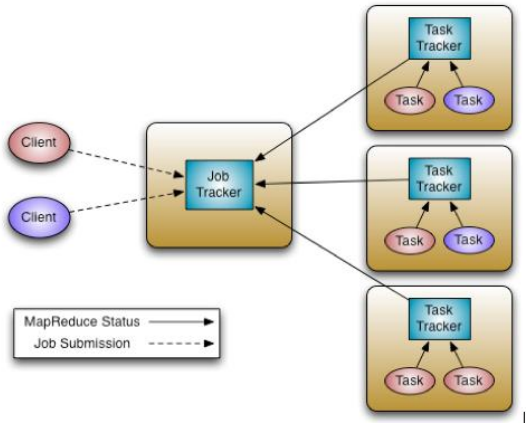


Figura 8. Arquitetura do MapReduce no Hadoop[30], [31]

2.3.2.3 YARN

YARN, acrónimo para *Yet Another Resource Negotiator*, introduzido na versão 2 do *Hadoop*, veio com o fim de melhorar a implementação do *MapReduce* e a computação distribuída [28]. Nas versões anteriores, a componente *JobTracker* tinha a função de gestão dos recursos e de tarefas para o processamento dos dados. YARN, isola essas funcionalidades para duas componentes: *ResourceManager* (gestão dos recursos disponíveis) e *ApplicationManager* (processamento de dados). Graças ao isolamento dessas funções houve integração de novas ferramentas de processamento de dados, evitando a dependência única de *MapReduce* ([28], [30]).

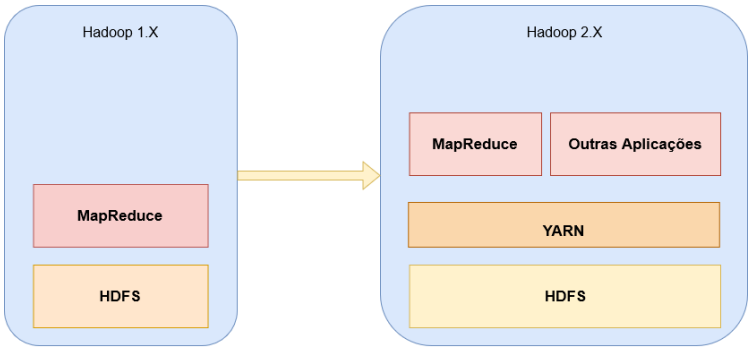


Figura 9. Transição da versão 1.X para a 2.X do Hadoop

YARN (Figura 10) é composto por:

- *Resource Manager*: orienta os recursos necessários para a execução das aplicações submetidas pelo utilizador. Contém um *Scheduler*, para o agendamento das aplicações com base nos recursos disponíveis, e o *Application Manager*, na submissão de trabalhos e a criação de containers para o *Aplication Master*[28];
- *Node Manager*: responsável pelos *containers* e monitorização dos recursos a ser utilizados [28];
- *Application Master*: componente que coordena a execução da aplicação em que ficou encarregue [30];

- *Containers*: criados pelo *Resource Manager*, onde são alocados os recursos e executam das tarefas [30].

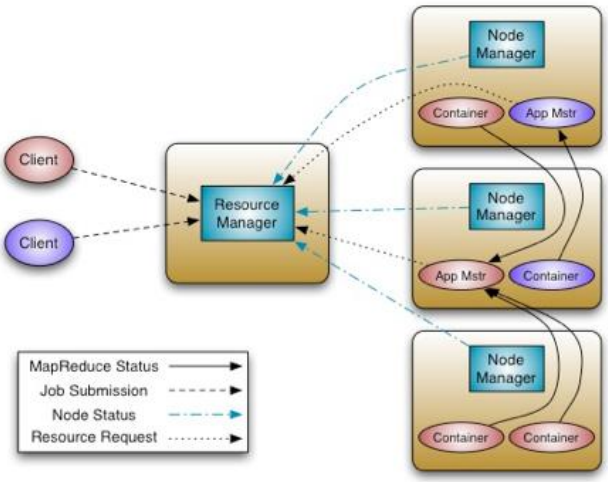


Figura 10. Arquitetura em YARN [32]

A execução de uma aplicação em *YARN* é composta pelos seguintes passos [33]:

1. Cliente submete a aplicação, com as especificações próprias, ao *Resource Manager*;
2. O *Resource Manager* cria um *Application Master*;
3. *Application Master* regista-se ao *Resource Manager;*
4. *Application Master* negoceia os recursos necessários para os *containers*;
5. *Application Master* inicia um container fornecendo as especificações do *Node Manager* e executa o código da aplicação;
6. Durante a execução o cliente comunica diretamente ao *Application Master* para obter o seu estado;
7. Quando aplicação é finalizada, *Application Master* tira o seu registo ao *Resource Manager* e desliga-se, dando oportunidade a novas submissões.

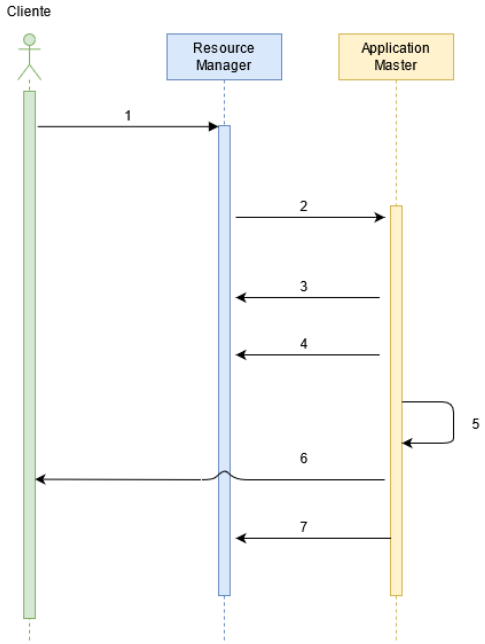


Figura 11. Diagrama de sequência na submissão de uma aplicação pelo YARN.

YARN ajuda a superar um conjunto de limitações que ocorriam com o tradicional *MapReduce* na primeira versão do *Hadoop*. Na Tabela 2, é realizada uma análise comparativa entre os dois sistemas ([28], [30]):

Tabela 2. Comparação entre o tradicional MapReduce com o YARN ([28], [30])

	***MapReduce* (*Hadoop* 1.X)**	***YARN* (*Hadoop* 2.X)**
Escalabilidade	Limitado para 4000 nós do agrupamento e para 40000 tarefas.	Expansível para 10000 nós e para 100000 tarefas.
Disponibilidade	Limitado pela disponibilidade do *Job Tracker*.	Mais disponível com a separação funções do *Job Tracker* pelo *Resource Manager* e *Application Master*.
Utilização	Fica limitado a uma quantidade estática para cada executor.	Melhor gestão de recursos, evitando limitações.
Aplicações	Exclusivo para aplicações *MapReduce*	Maior suporte em vários tipos de aplicações e ferramentas sem ser *MapReduce*.

2.3.3 Hive

Para acesso e transformação dos dados através do *Hadoop* é necessário que os utilizadores estejam familiarizados com a programação e dos conceitos do *MapReduce*, levando a um grande consumo de tempo em formação e à elaboração desse tipo de aplicações. O *Hive* foi implementado para superar os problemas de aprendizagem [34], sendo um dos

primeiros sistemas em contexto *SQL-on-Hadoop*, O *Hive* gera e executa um conjunto de tarefas *MapReduce* através de uma linguagem semelhante ao *SQL* (*HiveQL*) [35].

O *Hive* é usado para aplicações de *Data Warehouse*, mas não é uma base de dados completa, sendo impróprio para fins de transações, mas adequado para as operações de análise. Como a maioria das aplicações de *Data Warehouse* tradicionais são implementadas por meio de bases de dados relacionais, o *Hive* ajuda na aplicação desses conhecimentos para o ecossistema do *Hadoop* [36]. Tem a vantagem de facilitar o acesso e transformação dos dados, em comparação com a criação manual de aplicações *MapReduce*. Tem a desvantagem de não ser adequado em dados não estruturados e ineficiente em tarefas muito complexas [37].

Além de ter a capacidade de interpretar ficheiros estruturados (exemplo: *CSV* e *TXT*), mas também disponibiliza a leitura de ficheiros de formatação de coluna, que inclui algoritmos de compactação, para melhorar o desempenho em operações *I/O*, mais sustentável no armazenamento e desempenho em transferência. Os ficheiros utilizados são *ORC* (*Optimized Row Columnar*) e ficheiros *Parquet* ([19], [35], [38]).

A arquitetura do *Hive* é estruturada por ([34], [39]):

- Uma *Metastore*: onde é armazenada a informação sobre as estruturas das tabelas, colunas e partições, e a localização dos seus ficheiros;
- *Driver*: gere o tempo de vida do programa submetido *HiveQL*. Executa, coleta dados intermédios e finais, e finaliza a sua execução;
- Compilador *Query*: compila o *HiveQL*. Gera um plano de execução que inclui conjunto de fases e tarefas;
- *Optimizer:* procura o melhor plano para a execução da *query*;
- *Execution Engine*: executa as operações propostas pelo compilador;
- *HiveServer*: providencia um serviço de *JDBC*/*OBDC* para fazer a integração do *Hive* com outras aplicações;
- Componentes para o cliente: linha de comandos (*CLI*), interface *Web* e *JDBC*/*ODBC*.

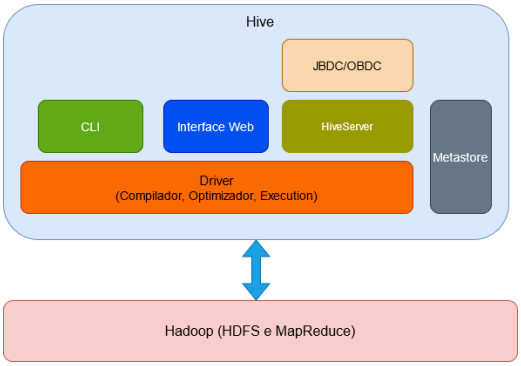


Figura 12. Exemplo da arquitetura Hive[34]

A execução de uma *query* através do *Hive* tem o seguinte procedimento [34]:

1. A *query HiveQL* é submetida pelo cliente;

2. Driver recebe a *query* e envia para o compilador;
3. O compilador verifica se a sintaxe corresponde aos dados da *Metastore*;
4. Após a verificação, o compilador gera um plano lógico e otimizado para a execução;
5. Do plano é gerado tarefas de *MapReduce* e de acesso ao *HDFS* e envia ao *Execution Engine*;
6. O *Execution Engine* executa essas tarefas usando a *framework Hadoop*;
7. Após finalizar a sua execução, os resultados são enviados ao cliente.

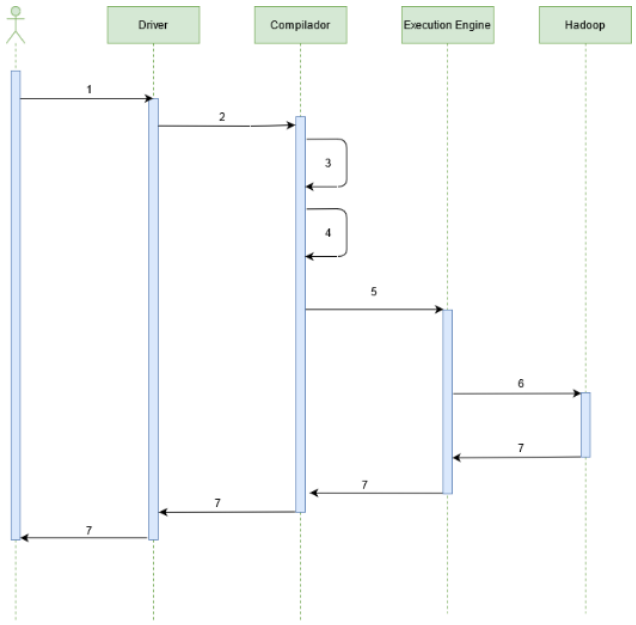


Figura 13. Diagrama de sequência quando o cliente executa uma query

No início, o *Hive* utilizava operações de *MapReduce* na obtenção de resultados através das suas *querys*, mas como já foi referido, o *MapReduce* necessita de escrever no disco resultados das operações intermédias, resultando em tempos de processamento grandes, face ao aumento dos dados. Com aparecimento do *Apache Tez* esses problemas foram amenizados. A utilização do *Hive* com o *Tez* veio reduzir a latência, tornando as *querys* mais interativas, através da redução de operações de leitura e de escrita no *HDFS*. Na figura abaixo corresponde a um exemplo utilizando o *MapReduce* tradicional, comparando com a execução pelo *Apache Tez* através do *Hive* [40], [41].

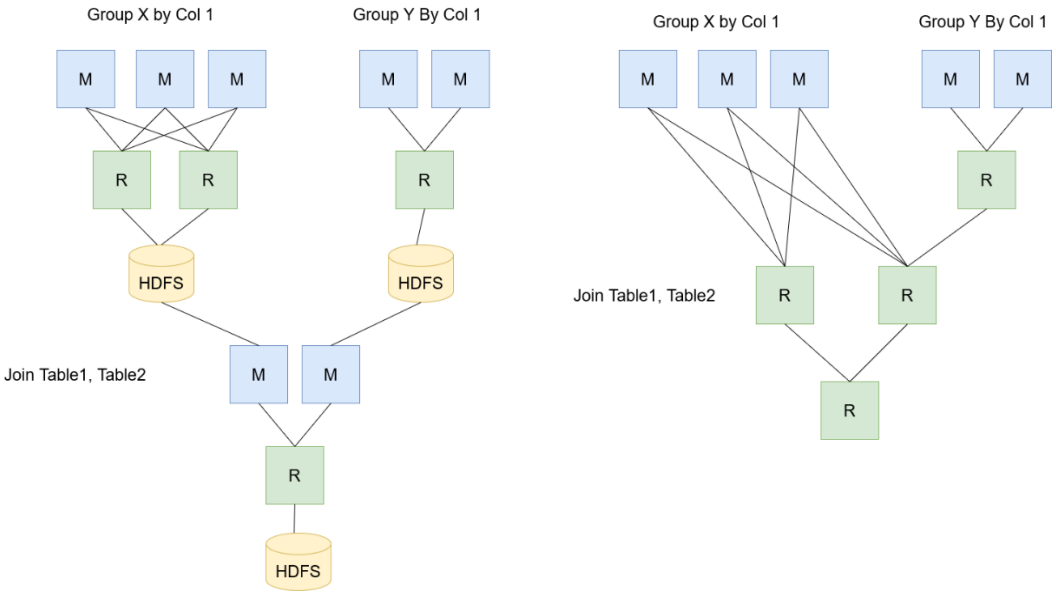


Figura 14. Comparação de execução de query por MapReduce (esquerda) e Tez (direita) [41].

2.3.4 Presto

Desenvolvido pelo *Facebook*, o *Presto* tem o objetivo de executar *queries* analíticas em grandes volumes de dados de forma distribuída, sendo os dados provenientes de várias fontes, tais como o *Hive* e base de dados relacionais. *Presto* é uma alternativa às ferramentas que utilizam *MapReduce*, como *Hive* e *Apache Pig* ([39], [42]). Tem o poder de executar centenas de *queries* em memória e com uso intensivo de *CPU*. Reduz a complexidade de integração com vários sistemas através do processamento de dados de várias fontes numa só *query*. É flexível para situações limitadas [43].

Um agrupamento do *Presto* é composto por um *coordinator* e por vários *workers*. O *coordinator* é responsável por receber as *queries* em *SQL* pelo utilizador, interpretar, planear e gerir os *workers*. Após o *coordinator* receber uma *query*, ele analisa e gere um plano. Esse plano é composto por fases, com estrutura hierárquica, que contém um conjunto de tarefas para serem tratadas pelos *workers* ([39], [42]).

Cada *worker* é responsável por executar as tarefas propostas pelo *coordinator* e fazer o seu processamento em paralelo. É responsável na recolha dos dados em várias fontes através dos conectores, e partilha os resultados intermédios com outros *workers*. Após acabar o processamento, o *coordinator* faz a recolha dos dados e apresenta ao utilizador ([39], [42]).

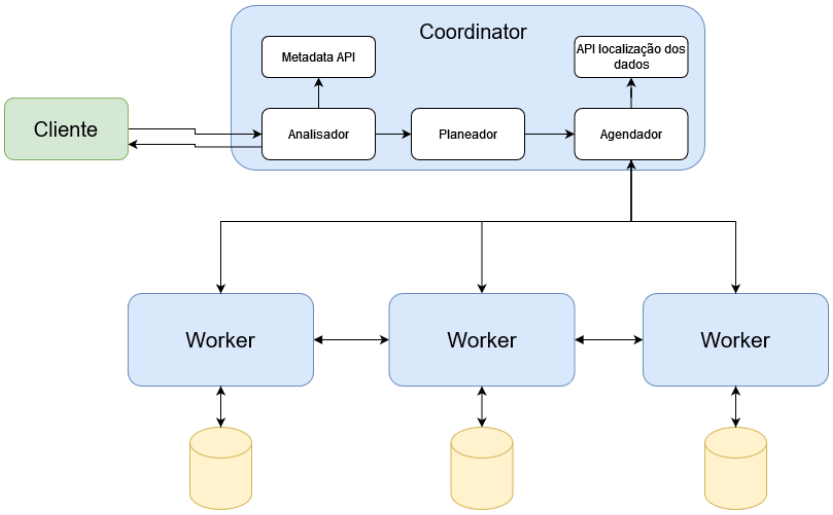


Figura 15. Exemplo de arquitetura do Presto [44].

2.3.5 Apache Spark

Desenvolvido em 2009, o *Apache Spark* [45], vem com o fim de aplicar o processamento em grande volume de dados históricos, mas também com a capacidade de processamento de dados em tempo real (*streaming*). *Spark* é capaz de armazenar e processar os dados em memória distribuídos em todo o seu agrupamento. Com essa aplicação evita a necessidade de aceder ao disco, melhorando o seu desempenho. Tem a interoperabilidade no acesso a várias fontes de dados, tais como *HDFS*, bases de dados (relacionais e não relacionais) e em vários tipos de ficheiros ([46], [47]).

Spark em comparação com o *MapReduce* do *Hadoop*[46]:

- Mais rápido, 100 vezes mais rápido quando usa memória, 10 vezes quando usa o disco, através da redução de operações de leitura e de escrita e armazenamento dos dados em memória;
- Sendo o processo feito em memória, *Spark* acaba por ser mais dispendioso em comparação com *MapReduce* que utiliza disco;
- Oferece operações de alto nível, facilitando o desenvolvimento de aplicações;
- Fornece processamento em tempo real e em grandes volumes.

Um sistema *Spark* segue o padrão arquitetural mestre-escravo. Consiste num coordenador, *driver*, e um conjunto de *workers* distribuídos pelo agrupamento. O *driver* é o ponto de entrada entre o utilizador e as funcionalidades do *Spark* (através do *SparkContext*), converte as aplicações em pequenas tarefas (*tasks*), agenda *jobs* e aloca recursos para a execução. Cada *worker* cria um executador que executa as tarefas propostas e o processamento de dados. O *Cluster Manager* é um serviço externo com a responsabilidade da aquisição de recursos para as aplicações. *Spark* tem o seu próprio Cluster Manager, mas é possível utilizar o *YARN* como alternativa [46].

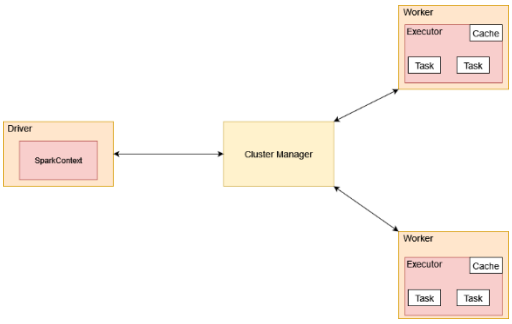


Figura 16. Exemplo de arquitetura do Spark.[46], [48]

Quando um programa *Spark* é submetido para o seu agrupamento, é gerado um diagrama acíclico. Esse é composto por um conjunto de fases com o fim de obter resultados. Cada fase é composta por um conjunto de tarefas sequenciais e otimizadas para uma maior performance. Esse ciclo é definido por *DAG* (*Directed Acyclic Graph*) [43]. *Spark* disponibiliza ao utilizador uma visualização gráfica o *DAG* gerado.

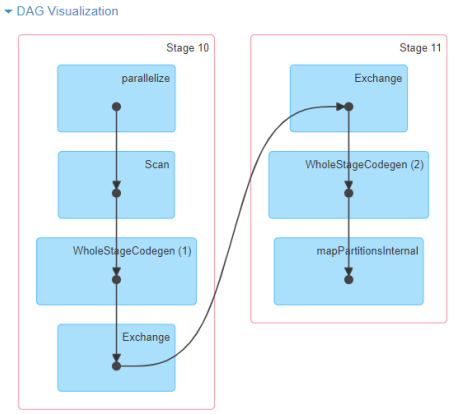


Figura 17. Exemplo de um esquema gráfico do DAG [49].

Existe um conjunto de componentes que fazem parte do ecossistema do *Spark* [46], [50]:

- *Spark Core*: é o pilar do *Spark* fornecendo *API* para processamento de dados em grande escala por meio de um conjunto de objetos distribuídos pelos nós do agrupamento, os *Resilient Distributed Datasets* (*RDD*). Ainda fornece operações de gestão de memória, agendamento de tarefas e recuperação de falhas;
- *Spark SQL*: é a execução de *querys SQL* pelo *Spark*;
- *Spark Streaming*: é o componente para o processamento em tempo real, através de pequenos lotes de dados;
- *MLlib*: fornecer algoritmos de *Machine Learning* em grande escala, quando há necessidade de paralelismo de dados ou do modelo;
- *SparkR*: trata-se do pacote para a integração de *Spark* com a linguagem R;
- *GraphX*: está dedicado para computação de grafo;

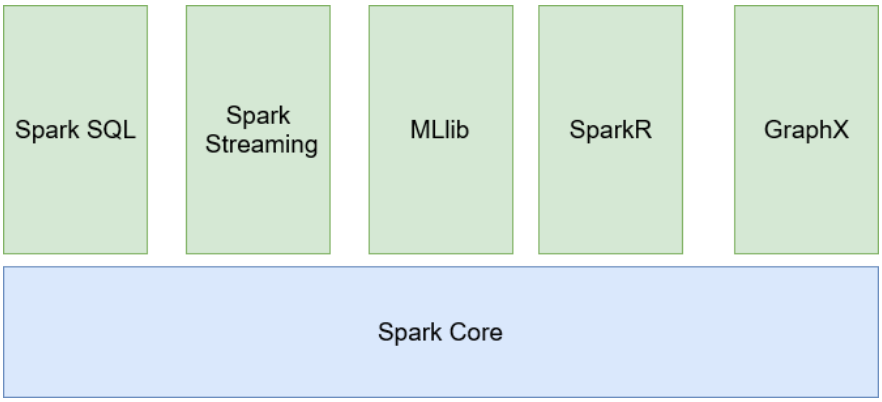


Figura 18. Componentes do Spark [46], [50].

2.3.5.1 Spark SQL

Spark SQL é uma componente de alto nível do *Spark* que permite aos utilizadores integrar e aproveitar os benefícios do processamento relacional e utilizar conjuntamente com outras bibliotecas mais avançadas disponibilizadas pelo *Spark*.

Spark SQL dá aos utilizadores acesso à interface *SQL* por conectores *JDBC/ODBC* e por consola, habilitando-se a ser um motor de *SQL* distribuído capaz de conectar a um conjunto de ferramentas de *BI*. Além disso, disponibiliza o acesso por meio de *DataFrames* para as aplicações em *Java*, *Python* e *Scala*. Permite o acesso fácil às tabelas do *Hive* e a execução das *querys* em ambiente do *Spark*, mas também acesso a outras fontes, tais como as Bases de dados relacionais, ficheiros estruturados e semiestruturados, e bases de dados *NoSQL* (ex. *Cassandra*, *HBase*) [47], [51].

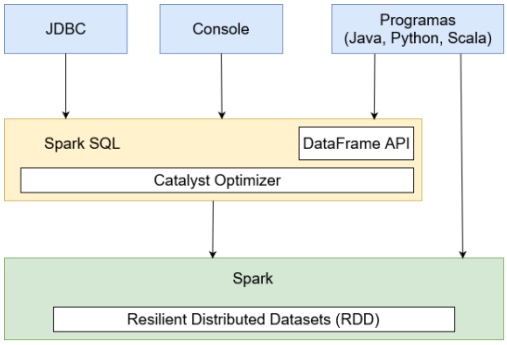


Figura 19. Interação com Spark SQL [51].

Para a obtenção de resultados com maior desempenho foi desenhado o *Catalyst Optimizer* (Figura 20). Tem a finalidade de integrar novas técnicas de otimização para o *Spark SQL*. Este permite estender e adicionar novas regras de otimização dependendo da situação do utilizador. Quando o utilizador submete uma *query SQL*, ou por um *DataFrame*, é recebido pelo *Catalyst Optimizer* que passará por um conjunto de várias fases antes de ser executado.

Na primeira fase de análise, é recebido um *query SQL* ou um *DataFrame*. Existindo um conjunto de atributos e relações por resolver, é utilizado o catálogo para verificar e aprovar se está apto para a próxima fase.

Na fase de otimização lógica, é recebido o plano lógico e sofre um processo de transformações através de um conjunto de regras de otimização.

No planeamento físico, é recebido um plano lógico otimizado sendo gerado vários planos físicos baseado em regras de otimização estabelecidas. Esses planos são a seguir selecionados e, através do modelo de custo, é selecionado tendo o melhor custo. Por fim, o plano selecionado segue-se pela fase de geração de código, gerando-se em *Java Bytecode* para executar pelos nós do agrupamento [51].

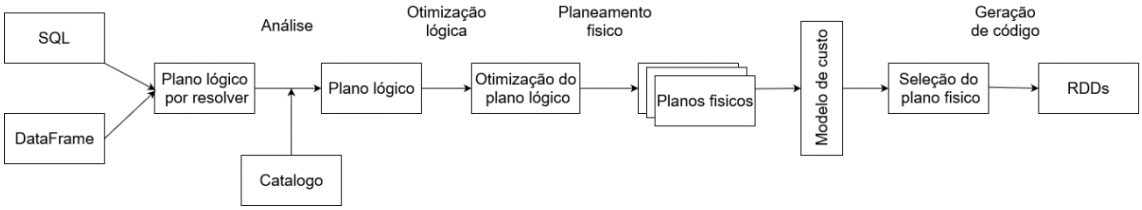


Figura 20. Processo do Catalyst Optimizer [51].

As otimizações do *Catalyst Optimizer* são baseadas em regras selecionadas por meio de estatísticas e estimativas antes de executar o programa e ao longo da execução poderão ocorrer problemas, além da redução de desempenho. A partir da versão 3.0 do *Spark*, foi adicionado uma nova componente de otimização para fazer face a esse problema, o *Adaptive Query Execution* (*AQE*). Este permite que o *Spark* recolha as estatísticas adquiridas durante as fases de execução, otimização e escolha do novo plano de *query* ([52], [53]).

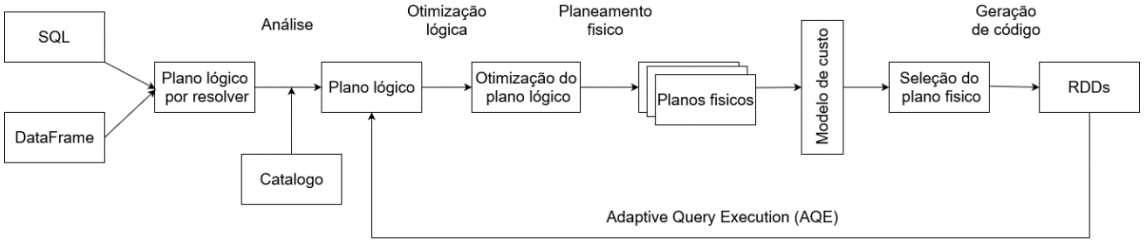


Figura 21. Processo do Catalyst Optimizer com AQE [52].

2.3.6 Conclusão

Neste capítulo foi feita a revisão de literatura de um conjunto de tecnologias. As Bases de Dados Relacionais e Não Relacionais são usadas em muitas aplicações. Ambos os tipos de bases de dados contêm vantagens e desvantagens, sendo que a utilização das Não Relacionais tem maior uso em aplicações de *Big Data* face às Relacionais.

O desenvolvimento da ferramenta *Hadoop* trouxe vantagens em responder aos desafios do *Big Data*. Com o seu sistema distribuído, *HDFS*, tornou-se capaz de armazenar conteúdos, independentemente do tipo de formato, de forma distribuída entre vários agrupamentos. Foi introduzido o algoritmo *MapReduce* para o processamento paralelo em pequenos blocos e fases. Com o desenvolvido o YARN, tornou o sistema *Hadoop* mais escalável e com maior suporte para outras aplicações sem ser o *MapReduce*.

O *Hive* tem o intuito em aplicar conceitos de *Data Warehouse* no ecossistema do *Hadoop* (*SQL-on-Hadoop*), com a representação dos dados através de tabelas e facilitar o acesso

via linguagem semelhante ao *SQL* (*HiveQL)*. As *queries* criadas são convertidas num conjunto de processos de *MapReduce*.

O *Presto* e o *Spark* foram desenvolvidos para dar complemento aos desafios do *Big Data*, não substituindo *Hadoop* ou o *Hive*. O *Presto* tem a capacidade de processar e receber grandes quantidades de *queries* provenientes de várias fontes, incluindo do *Hive*, através do uso intensivo de CPU e de memória disponível pelo agrupamento. O *Spark* tem o propósito de fazer o processamento dos dados em memória, evitando a necessidade do disco. É composto por ecossistema rico, desde aquisição e processamento de dados em tempo real (*Spark Streaming*), até a um conjunto de bibliotecas disponíveis para *Machine Learning*.

Uma das funcionalidades do seu ecossistema, o *Spark SQL*, proporciona uma interface *SQL* para o acesso às tabelas do *Hive*, mas também permite ao acesso de vários tipos de fontes. Dispõe um *Catalyst Optimizer* que recebe o *SQL* pelo utilizador e por meio de processos de análise, otimização e de planeamento, é gerado código de execução para os nós do agrupamento.

A elaboração deste capítulo tornou-se importante na escolha de tecnologias existentes para este trabalho na área do *Big Data*. Na próxima secção, são reportados os trabalhos relacionados à prática destas tecnologias referidas em ambientes de *Big Data*, por meio de comparações entre o desempenho de cada tecnologia, mas também da aplicação de outros tipos de metodologia no armazenamento e na representação dos dados.

2.4 Trabalhos relacionados em *Big Data*

Com aquisição dos conceitos de *Data Warehouse*, de *Big Data* e a introdução de novos tipos de dados gerados através das redes sociais e dos *IoT*, ocorrem exemplos de necessidade de migrar ou implementar alternativas às *Data Warehouses* usando as tecnologias de *Big Data*. Para este trabalho, os estudos mencionados neste capítulo foram um ponto de partida na introdução de métodos e arquiteturas utilizadas referentes ao *Big Data*.

Sebaa et al. [54], elaboraram uma *Data Warehouse* através do ecossistema *Hadoop*, com o objetivo de transpor das dificuldades encontradas em *Data Warehouses* na área da medicina, de modo a melhorar a qualidade dos serviços de saúde nas tomadas de decisão. Na arquitetura proposta, aplicaram processos de *ETL* e submeteram para o sistema de ficheiros *Hadoop* (*HDFS*). Os dados foram representados por tabelas de *Hive* e acedidos por ferramentas de análise de *Business Intelligence*. O modelo de dados foi composto por tabelas de facto e tabelas de dimensão, tendo sido aplicados técnicas de partição e de *bucket* nas tabelas de facto, com o objetivo de aumentar o desempenho das *queries*.

Chang et al. [55] propuseram uma arquitetura de *Big Data Warehouse* para monitorizar e visualizar, em tempo real, o consumo de eletricidade do *campus*. Nessa arquitetura, foram utilizados o *Hadoop* (como armazenamento distribuído), o *Hive* (na representação em tabelas), *Sqoop* [56] (na recolha de dados históricos de origem *MySQL*), o *Spark* (para recolha de dados energéticos e tratamento dos mesmos) e o *Impala* (para análise de dados).

Na avaliação, concluíram que a execução de *queries* no *Impala* teve melhor desempenho em comparação ao *Spark SQL* e o *HiveQL*. Graças ao facto de o *Impala* criar ficheiros menores durante a execução das *queries* através de agrupamento. As aplicações *ETL* através do *Spark* foram mais rápidas em comparação ao *Hive* [55].

Na área da agricultura, *Ngo et al*. [57] propuseram a criação de uma plataforma de análise. Foi utilizando o *Hive*, o *MongoDB* [58] e o *Cassandra* [59]. O *MongoDB* tinha a facilidade de receber dados em tempo real pelas aplicações móveis e pela *Web*, além de dar reposta aos resultados necessários em tempo real. O *Hive* tinha a função de importar os dados do *MongoDB* e retornar dados processados. Em situações de cálculos mais complexos, *Hive* intervém na fase de responder. Todos os dados não processados de diferentes bases de dados e ficheiros eram armazenados no *Cassandra* e, por meio do *ETL,* eram tratados e enviados para o *Hive*. Na representação dos dados da *Data Warehouse,* foi utilizado um conjunto de tabelas de facto e tabelas de dimensão, assemelhando-se ao esquema constelação. Avaliaram a *Data Warehouse*, aplicando um conjunto de *queries*, incluindo vários comandos de *SQL* e compararam o seu desempenho com *MySQL*. Após a comparação, concluíram que a sua proposta tem maior desempenho.

Viera et al.[60], combinou um conjunto de abordagens para aquisição de requisitos para o desenvolvimento de uma *Data Warehouse* em contexto *Big Data*. A proposta apresentada permitiu incluir dos dados importantes de análise e desenvolver modelo dimensional utilizando o *Hive*. Martinho e Santos[61], propuseram que o as tabelas de facto e de dimensão existentes da *Data Warehouse* tradicional sejam exportadas para o *HDFS*, representandas através de tabelas do *Hive* e transformar de um modelo dimensional para o modelo tabular.

Santos et al.[62] avaliaram o desempenho de tecnologias *SQL-on-Hadoop* (*Hive*, *Spark*, *Presto* e *Drill* [63]) em ambientes onde os recursos disponíveis eram mais limitados. Para esse estudo foi utilizado o *dataset TPC-H* [64] desnormalizado para uma tabela única. A desnormalização teve o objetivo de reduzir a quantidade dos recursos utilizados e do tempo de execução das *queries*. Com os resultados obtidos concluíram que, para volumes menores o *Hive* não é o ideal. Em situações de maior aumento de volume começou a haver degradação de desempenho por parte do *Spark* e do *Drill*, havendo situações que o *Hive* era o melhor. *Presto* manteve-se com um bom desempenho consoante ao aumento do volume em comparação ao *Hive*.

Nadeem Ahmed et al. [65] realizaram uma análise comparativa no desempenho das *queries* entre *SQL Server* e *Hive*. Os dados utilizados para esse estudo consistiam em registos de crimes ocorridos em Inglaterra divididos em ficheiros de 89 MB e 1 GB. Concluíram que, durante a execução de um conjunto de *queries*, o *SQL Server* teve melhor desempenho em comparação ao *Hive*, mas com o aumento do volume de dados notou-se uma maior degradação do seu tempo de execução.

Rafael Almeida *et al*. [66] avaliaram o comportamento do desempenho das tecnologias *SQL Server* (versão 2012), com *MySQL* e *InnoDB* em ambientes de apoio à decisão consoante o aumento do volume de dados. Nessa avaliação, utilizaram o modelo dimensional em estrela, tendo uma tabela de factos com capacidade inicial 1GB até 24GB. Após executar um conjunto de *queries* e consoante ao aumento do volume, concluíram

que *SQL Server* obteve bons resultados em grandes *datasets* em comparação às outras tecnologias, sendo o mais adequado para ambientes de apoio à decisão.

Aluko et al. [39], aplicaram um estudo intensivo em três diferentes conjuntos de dados, avaliando os sistemas *Hive*, *Spark SQL*, *Presto* e o *Impala*. Além da conclusão de vários factos referenciados no seu estudo, ao nível de utilização de recursos, *Presto* e Impala utilizaram grandes quantidades de memória. O consumo de memória por parte de *Spark SQL* foi variável dependendo do número de executadores e de *queries* em paralelo. O *Hive* utilizou, em abundância, o disco e o Impala é mais balanceado na utilização de recursos.

2.5 *Big Data* na área dos transportes públicos

Para uma grande parte da população, a utilização dos transportes públicos ocorre no seu dia a dia. A introdução de sistemas *Smart Card* e *AFC* (*Automatic Fare Collection*), veio como meio de facilitar o acesso e gestão de novas tarifas para os seus utilizadores. Para as empresas que utilizam esses sistemas, providencia maior qualidade de dados recolhidos para efeitos de análise para melhorar os seus serviços [1].

Sistemas *AFC* são compostos por bases de dados transacionais, onde as transações ocorridas correspondem às validações, por parte dos clientes, quando inserem o seu *Smart Card* nos leitores disponíveis nos veículos. Com a inclusão dos sistemas de *GPS* nos veículos, é possível saber qual é a paragem que o cliente entrou ou saiu após a sua validação [67].

A adoção destes sistemas retorna grandes volumes de dados, mas muitos deles podem não ser úteis para efeitos de análise. Essas situações ocorrem quando [68]:

- Existem problemas de sistema, tais como falta de requerimentos para a recolha de dados, de data e horas do sistema não estarem sincronizados, etc.;
- Problemas de hardware, nos sistemas de validação, comunicação, etc.;
- Falhas do utilizador, por má utilização do cartão ou utilização de cartão inválido.

Vários investigadores utilizaram os dados gerados pelos sistemas *AFC* para classificarem as motivações, por parte do utilizador, na utilização do serviço. As motivações dependiam do tipo de utilizador, do tempo da próxima validação e de outros critérios que podem ser classificados como: trabalho, escola ou outros ([69], [70]). Além disso, os autores de [70], [71] e [72] determinaram períodos onde os utilizadores transferiam de um veículo para o outro com o fim de detetar situações onde era necessário reformular o plano de viagem. *Song et al*. [73] utilizaram cubo *OLAP* para abordar uma melhor visualização das transações ocorridas no mês de outubro em *Naijing* para encontrar padrões por parte dos utilizadores. Também foram utilizados algoritmos de aprendizagem e de agrupamento (ex. *k-means*) na identificação de padrões de viagem [67].

Os requisitos de validação dos sistemas *AFC* dependem das decisões pelas empresas. Existe sistemas *AFC* que obrigam os clientes a validarem o seu cartão nas entradas e saídas das viaturas, mas existe outros sistemas obrigam os clientes validarem, unicamente, à entrada. Nessas situações, levam a dúvida do destino final. Esse problema, levaram à criação de vários métodos para a estimar a origem-destino de cada cliente [74].

Horta *et al*. [75] e Nunes *et al*. [76] abordaram um conjunto de regras para estimar o destino de cada cliente.

3 Análise do Problema

Após ter sido feito a revisão da literatura sobre os conceitos de *Data Warehouse*, modelos dimensionais, a definição de *Big Data*, tecnologias, aplicações e análises de dados na área dos transportes públicos, concluímos que obtivemos informações suficientes para passar a seguinte fase de análise ao problema que nos foi apresentado. Na figura abaixo, são apresentados os passos que foram efetuados para a análise ao problema:

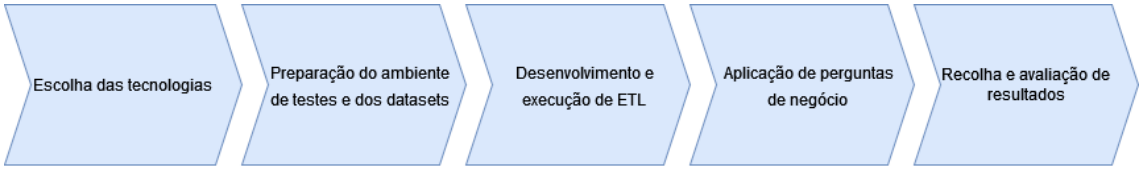


Figura 22. Passos para o desenvolvimento.

3.1 Recolha dos dados

Os dados de mobilidade que foram utilizados para análise do problema pertenciam à companhia de transporte público do Funchal, a Horários do Funchal. A companhia foi criada em 1986 e inicializou o seu serviço publico urbano em 1987 pelo concelho do Funchal. Em 1997, iniciou-se o seu serviço interurbano em freguesias adjacentes (ex. Curral das Freiras, Camacha, Faial, etc.). Em 2007, adotaram o sistema elétrico de bilheteiras, o GIRO, disponibilizando maior controlo na gestão de bilheteira. No momento deste trabalho, disponibilizam à população um serviço de transporte público urbano pelo concelho do Funchal, com uma frota de 158 autocarros disponíveis. Também dispõe uma frota de serviço interurbano de 62 autocarros [77]–[80].

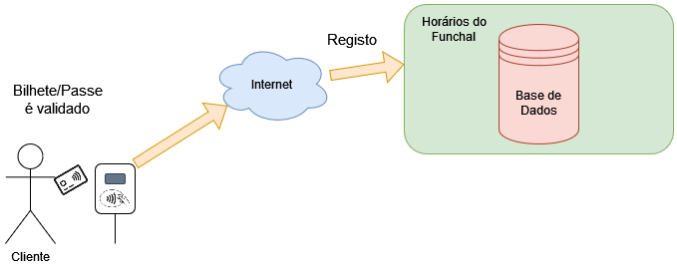


Figura 23. Arquitetura do sistema GIRO.

Os dados históricos que foram disponibilizados pela companhia correspondam às validações, à entrada dos veículos, pelos clientes através do sistema de bilheteira da companhia. Os dados compreendiam entre os anos 2015 e 2019 referentes aos serviços suburbanos. Devido às políticas de privacidade da empresa, não foi possível descrever a estrutura da base de dados e a composição dos dados disponibilizados para este trabalho.

Além dos dados que foram fornecidos pela empresa, foi incluído um conjunto de ficheiros *GTFS* (*General Transit Feed Specification*) [81]. Esses ficheiros foram disponibilizados para dar informação sobre os transportes públicos disponíveis utilizando a aplicação G*oogle Maps.* Os ficheiros utilizados para este estudo foram [82]:

- *Routes.txt*: lista de rotas disponíveis para o público;
- *Stops.txt:* lista de paragens onde os veículos recolhem ou desembarcam os passageiros;

Nessa fase, numa primeira instância, foi necessário fazer uma pré-seleção dos atributos nas tabelas e nos ficheiros que foram necessários para este projeto. Após essa escolha, todos os dados foram carregados para o ambiente desenvolvido.

3.2 Preparação do ambiente de testes

Infelizmente não foi possível disponibilizar, para este projeto, infraestruturas dedicadas para estes ambientes de testes em *Big Data*. A alternativa para este problema foi escolher entre implementação por meio de máquinas virtuais ou a criação de *containers* pela plataforma *Docker*.

Em comparação às máquinas virtuais, *Docker* é menos dispendioso ao nível de gestão de recursos e acessível em elaboração de sistemas distribuídos por *containers*[83]. *Potdar et al.*[83], avaliaram o desempenho entre as máquinas virtuais e o *Docker*. Concluíram que o *Docker* obteve melhor desempenho em termos de CPU, memória, *I/O* do disco, etc. Com essas conclusões, decidiu-se utilizar o *Docker* para o desenvolvimento deste projeto.

3.2.1 Tecnologias escolhidas

Com base das tecnologias estudadas na revisão de literatura, foram escolhidas:

- *Apache Hadoop*: para o armazenamento dos dados pelo seu sistema de ficheiros;
- *Apache Hive*: para a representação dos dados em tabelas e na execução de *queries*;
- *Apache Spark*: para o processamento de *ETL* e de execução de *queries;*
- *Presto*: para a execução de *queries;*
- *SQL Server*: armazenamento e execução de *queries*;

Para o seu desenvolvimento, foi necessário desenvolver um ficheiro *docker-compose* para gerar, de forma automática, todos dos *containers*. Para a criação dos *containers* Hadoop, Hive e Spark foram baseados nos projetos de *Big Data Europe* [84]. Já para o Presto foi baseado no projeto *Yasunari S.* [85]. Ambos os projetos, no momento que este documento está a ser desenvolvido, estão disponíveis através da plataforma *GitHub.* No *SQL Server* foi utilizado a imagem do *Docker* oficial disponibilizada pela *Microsoft.*

Durante o processo de escolha de tecnologias, averiguou-se a inclusão o *MongoDB* [58] para este estudo, por ser uma base de dados Não Relacional e estar mais adaptado para tecnologias em *Big Data* em comparação de uma base de dados Relacional [18]. Infelizmente, durante o processo de desenvolvimento, ocorreram um conjunto de erros, desde falhas de importação de dados até a constante reinicialização dos *containers*. Com esses problemas, que acabou por ser excluído deste estudo. O mesmo aplicou-se na abordagem de cubos *OLAP* e nas ferramentas de *Bussiness Inteligence* (exemplo: *PowerBI*). Devido a ocorrência de falhas de conexão, não foi possível incluir neste estudo.

O *SQL Server*, sendo um base de dados Relacional, não é muito utilizado em aplicações de *Big Data* em comparação à outras bases de dados Não Relacionais[18]. Mas, atendendo a que obteve bons resultados de desempenho ao *Hive* em grandes volumes de dados ([65], [66]) considerou-se que poderia, no fim, obter resultados uteis para este estudo.

3.3 Criação de *ETL*

Antes de desenvolver os processos de *ETL* foi necessário idealizar a representação dos dados no modelo final. A abordagem dos conceitos dos modelos dimensionais, das tabelas de facto e de dimensão, esteve muito presente ([54], [57], [60], [62]). Por isso, idealizou-se um modelo dimensional onde os dados poderiam ser carregados após o processo de transformação.

Para o desenvolvimento e avaliação dos desempenhos de *ETL*, as tecnologias escolhidas foram:

- *Spark*, com a biblioteca *PySpark;*
- *SQL Server Integration Services* (*SSIS*).

Para a abordagem do *PySpark*, foi necessário desenvolver um *script* em linguagem *Python*. Nele foi incluído a biblioteca *PySpark* para dar o máximo aproveito do processamento em paralelo da tecnologia *Spark*, também integrado com as tabelas do *Hive*. O *SSIS* é uma ferramenta disponibilizada pela *Microsoft* que permite o desenvolvimento de soluções *ETL* através de uma *interface* gráfica. Essa abordagem facilita à utilizadores menos experientes na programação a desenvolver conjuntos de soluções sem utilizar nenhuma linha de código. Esta ferramenta foi utilizada para a criação de *ETL* para o *SQL Server*.

3.4 Aplicação às perguntas de negócio

Posteriormente, todos os dados foram integrados e carregados seguindo o modelo dimensional escolhido. Na fase seguinte foi dedicada à criação de perguntas de negócio para análise de desempenho. Tendo por base das perguntas recolhidas na revisão de literatura, na Tabela 3, foi utilizado essas perguntas:

Tabela 3. Perguntas de negócio.

Q1	Em cada quadrimestre, quantidade de validações feitas por cada rota.
Q2	Quantidade de transferências e a média de tempo ao longo do dia.
Q3	Percentagem de validações ao longo do dia, pelo tipo de cliente, durante o primeiro e terceiro trimestre.
Q4	Com base no tipo de tarifa e da diferença de tempo entre viagens, total de clientes que utilizaram transporte público pelos seguintes motivos: • Estudar; • Trabalho; • Outro.

Para cada pergunta, foi necessário escrever *queries* análogas, em linguagem *SQL*, para ser avaliado para cada um dos sistemas usados. Os critérios utilizados na recolha dos motivos para pergunta de negócio Q4 foram baseados por *Devillaine* et al.[86].

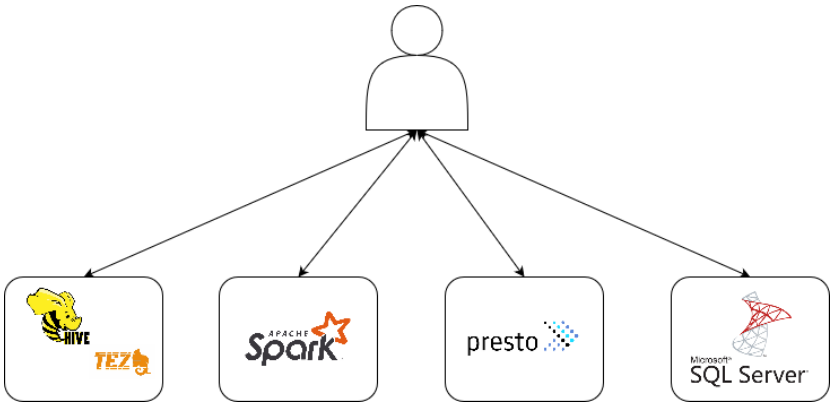


Figura 24. Aplicação das queries nos sistemas em estudo

3.5 Recolha e avaliação dos dados

Com os ambientes desenvolvidos, os dados integrados para o modelo dimensional e com as *queries* desenvolvidas, passámos à fase de recolha de valores relacionados com o desempenho na obtenção dos resultados das perguntas de negócio e com a execução dos processos *ETL*, consoante ao aumento do volume de dados. Para cada *dataset*, foi executado um conjunto de *queries*. Em cada *query* foi recolhido valores temporais de execução. Com a finalização do processo de recolha dos tempos de execução, todos os dados foram analisados e as tecnologias em estudo foram avaliadas com base nas suas prestações.

3.6 Conclusão

Neste capítulo foi exposta a análise ao problema e relatou-se o que iria ser implementado neste trabalho. Foram mencionados os dados usados, a preparação do ambiente de testes, as escolhas das tecnologias sujeitas à avaliação, o desenvolvimento do processo de recolha, a transformação e o carregamento dos dados ao modelo dimensional escolhido, a criação de perguntas de negócio da área dos transportes públicos e recolha e avaliação de resultados de desempenho.

4 Desenvolvimento

Neste capítulo é apresentado o processo de desenvolvimento para obtenção de resultados comparativos das tecnologias escolhidas do estudo. Ao contrário dos vários trabalhos relacionados feitos, não foi possível desenvolver infraestruturas com vários computadores físicos a correr várias instâncias de *Hadoop* e de outros sistemas. Devido às limitações de recursos e de infraestruturas, foi utilizado o ambiente de virtualização do *Docker* para o desenvolvimento.

Os recursos alocados e a versão da plataforma *Docker* para o desenvolvimento foram:

- *Docker Desktop* versão 4.10;
- *CPU*: 6 (de 8) núcleos do Intel Core i7-10700F 2.90 *GHz*;
- Memória: 12GB de RAM;
- Armazenamento: *HDD Seagate* 1 TB de 7200 *RPM*.

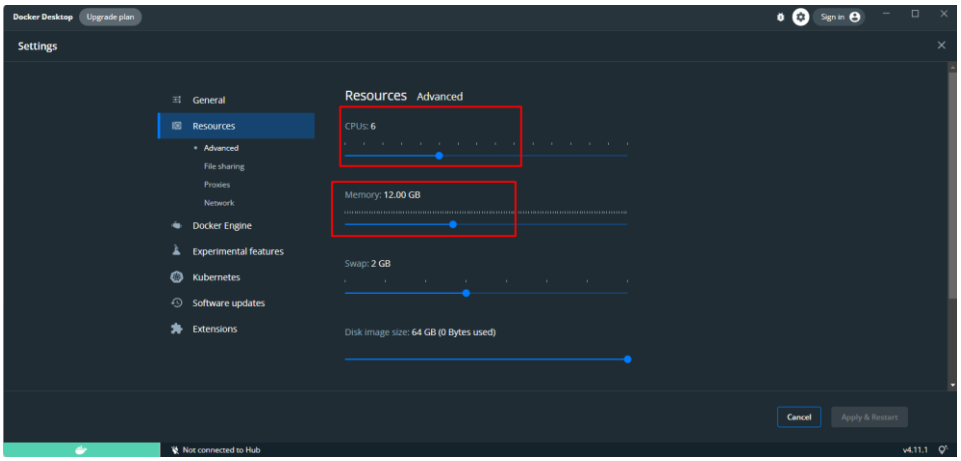


Figura 25. Recursos alocados para o Docker

4.1 Elaboração da Infraestrutura

A figura abaixo corresponde a proposta de arquitetura em ambiente para o ambiente de testes. Para o seu desenvolvimento através do *Docker*:

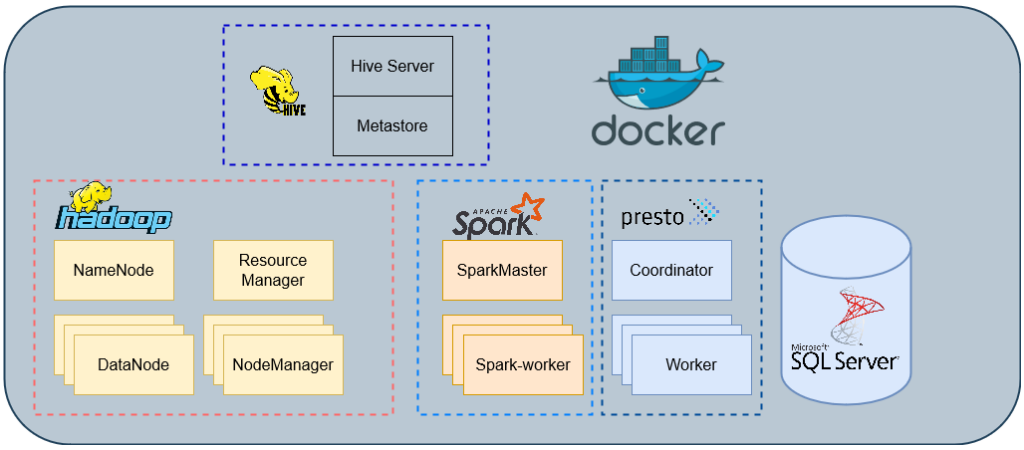


Figura 26. Infraestrutura em Docker criada.

Baseado nos repositórios desenvolvidos e pela documentação disponibilizada, elaborou-se um ficheiro *docker-compose.yml*, ver anexo 8.1, para a geração automática dos containers para a nossa arquitetura *Big Data*, a saber:

- *Hadoop*, anexo 8.1.1:
  - Para o sistema de ficheiros distribuído (*HDFS*), gerou-se um container *NameNode* e três *DataNodes*;
  - Para implementação do *YARN*, criou-se um container como *Resource Manager* e três *Node Managers*.
- *Hive*, anexo 8.1.3:
  - *Container* como servidor do *Hive*, incluindo bibliotecas do *Apache Tez*;
  - *Container* como base de dados para armazenamento dos metadados das tabelas.
- *Spark*, anexo 8.1.2 :
  - Gerar um agrupamento básico *Standalone*, composto por um *container* como mestre (*SparkMaster*);
  - Três *containers* como nós trabalhadores (*SparkWorker*).
- *Presto*, anexo 8.1.4:
  - Um *container* como *Coordinator*;
  - Três *containers* como *Workers*.
- *SQL Server*, anexo 8.1.5:
  - Um container.

Atendendo às limitações de infraestrutura, houve necessidade de configurar a distribuição de recursos para os nós trabalhadores do *Hadoop*, *Spark* e *Presto*. Para cada nó *Worker* (*Spark* e *Presto*) e *Node Manager* (*Hadoop*), foram alocados 2 núcleos de CPU e 3 GB de RAM. Em SQL Server foi configurado com 6 núcleos de CPU e 9 *GB* (9216 *MB*) de *RAM*.

Para estabelecer as configurações necessárias para o *Hadoop*, *Hive*, *Spark* e *Presto* foi preciso editar um conjunto de ficheiros de configuração. Já *SQL Server* foi nos apresentado uma configuração mais simples.

4.1.1.1 Configuração do *Hadoop*

O *Hadoop* disponibiliza ao utilizador 4 ficheiros de formato *XML* ([20] [87]) para configurar todo o agrupamento:

- *core-site.xml*: configurações relativas ao núcleo do *Hadoop*;
- *hdfs-site.xml*: configurações relativas ao sistema distribuído de ficheiros, *HDFS*. Configurações do *Namenode* e dos *Data*nodes;
- *mapred-site.xml*: configurações relativas ao *MapReduce*;
- *yarn-site.xml*: configuração dos *Resource Manager* e *Node Manager*.

Ambos os ficheiros de configuração são compostos por um vasto número de variáveis para montar um sistema *Hadoop*. Muitas delas contêm valores já pré-definidos. A configuração desses ficheiros foi um processo demorado e de tentativa e erro.

Em *Docker*, foi necessário elaborar um ficheiro *hadoop.env* (ver anexo 8.2) e referenciar como variáveis de ambiente do container, com a seguinte *template* [88]:

- *CORE_CONF*_[propriedade]: corresponde configuração de uma propriedade para o *core-site.xml*;
- *HDFS*_CONF_[propriedade]: corresponde configuração de uma propriedade para o *hdfs-site.xml*;
- *YARN* _[propriedade]: corresponde configuração de uma propriedade para o *yarn-site.xml*;
- *MAPRED_CONF*_[propriedade]: corresponde configuração de uma propriedade para o *mapred-site.xml*.

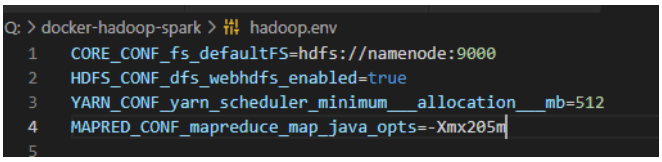


Figura 27. Extrato do ficheiro hadoop.env

A configuração dos recursos no ficheiro *mapred-site.xml* e de *yarn-site.xml* podem ser consultados nos anexos 8.2.1 e 8.2.2 correspondentemente.

Com a configuração do ecossistema *Hadoop* concluída, foi verificado se o sistema era capaz de detetar todos os *Datanodes* e os *NodeManager*s. Ao aceder pelo browser pelo endereço *localhost:9870*, obteve-se informação sobre o estado nos *Datanodes* disponíveis no *HDFS*. Na Figura 28 confirmou-se que foram detetados os 3 *Datanodes* (*Datanode1*, *Datanode2* e *Datanode3*) para o *HDFS*. Além de se verificar se estão em serviço, também inclui informações adicionais, como a capacidade, quantidade de blocos existentes, etc.

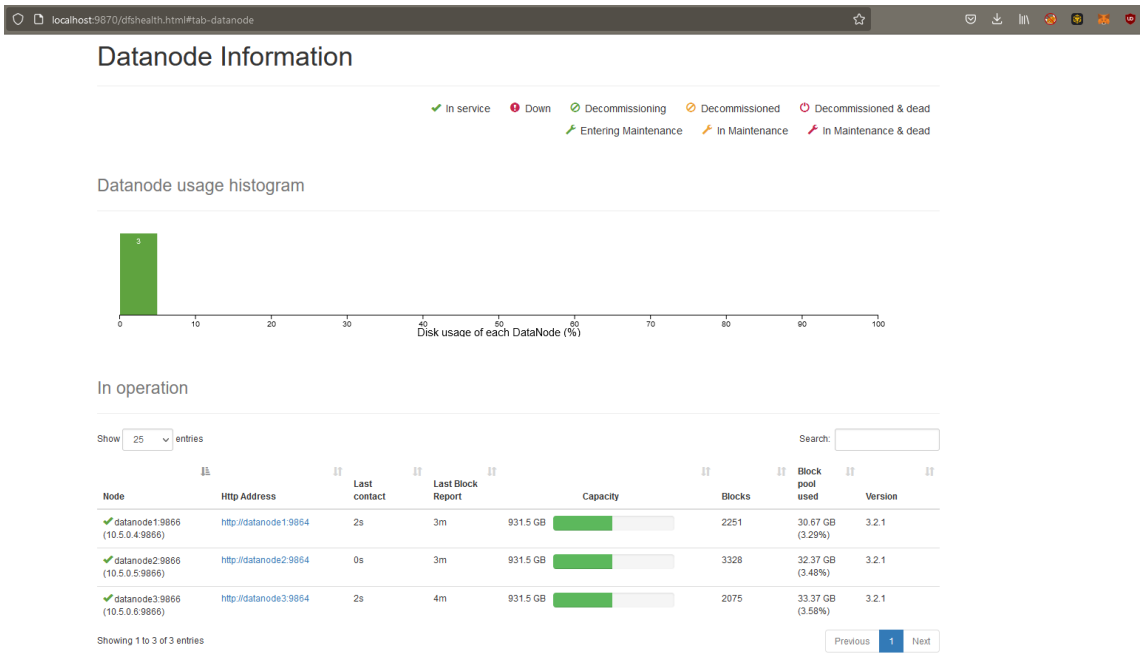


Figura 28. Dashboard do Sistema de Ficheiros do Hadoop (HDFS)

Na Figura 29, podemos confirmar que todos os *Node Manager* do YARN foram criados com sucesso, e também foi possível afirmar que cada um inclui 3GB de memória e 2 cores de CPU (4 vCores).

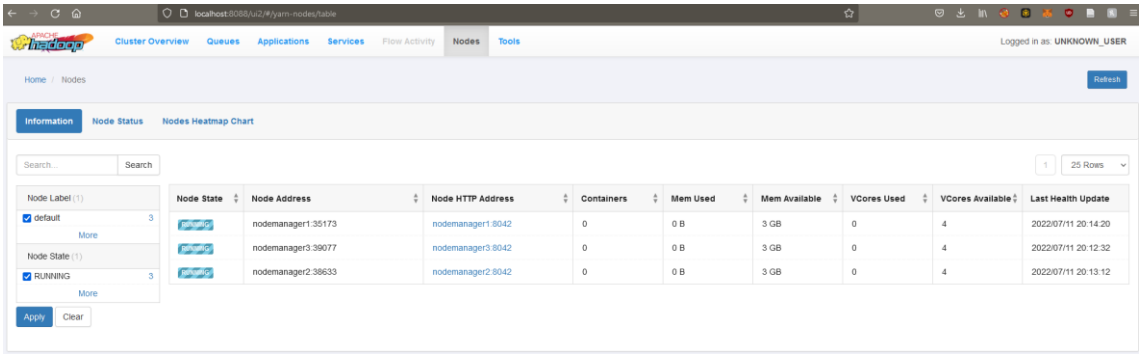


Figura 29. Dashboard do YARN

4.1.1.2 Configuração do *Hive* e *Tez*

À semelhança do *Hadoop*, o *Hive* necessitou de ser configurado pelo ficheiro *Hive-site.xml*. Como o *Hadoop*, para configurar foi preciso criar um ficheiro *hadoop-hive*.env, com o seguinte *template* [89]: *HIVE_SITE_CONF_*[propriedade].

Para o *Hive* utilizar o *Tez* como *engine* de execução foi necessário [90]:

1. Descarregar o ficheiro *Tez*.x.y.z.tar.gz;
2. Copiar o ficheiro para o *HDFS*;
3. Criar ficheiro de configuração *Tez-site.xml*;
4. Referenciar o ficheiro em *hadoop*-env.sh: *TEZ_CONF_DIR=[diretorio]/tez-site.xml*;
5. Em *Hive* aplicar este comando [91]: *set hive.execution.engine=tez.*

Após a configuração foi criado uma *query* simples para testar se *Hive* utilizou o *Tez*. Na figura seguinte, através da *Dashboard* do YARN, confirmou-se que *Tez* estava a funcionar.

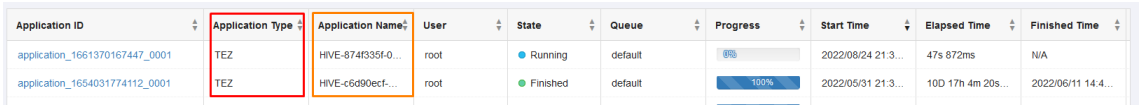


Figura 30. Confirmação da execução do Hive pelo Tez

4.1.1.3 Configuração do *Spark*

Com a implementação de um agrupamento em *Standalone*, o processo de configuração de Spark foi mais simples face ao Hadoop. Para a sua configuração, foi necessário editar um conjunto de variáveis no ficheiro *spark-env.sh* [92]. As propriedades configuradas podem ser vistas no anexo 8.4.

Abrindo a interface *Web* do *Spark* (Figura 31), podemos confirmar que foi criado com sucesso o nó mestre e os 3 nós trabalhadores do nosso agrupamento. Pela figura,

confirmou-se que cada nó trabalhador dispõe de 2 cores de CPU e 3GB de memória (ao todo 9GB de memória).

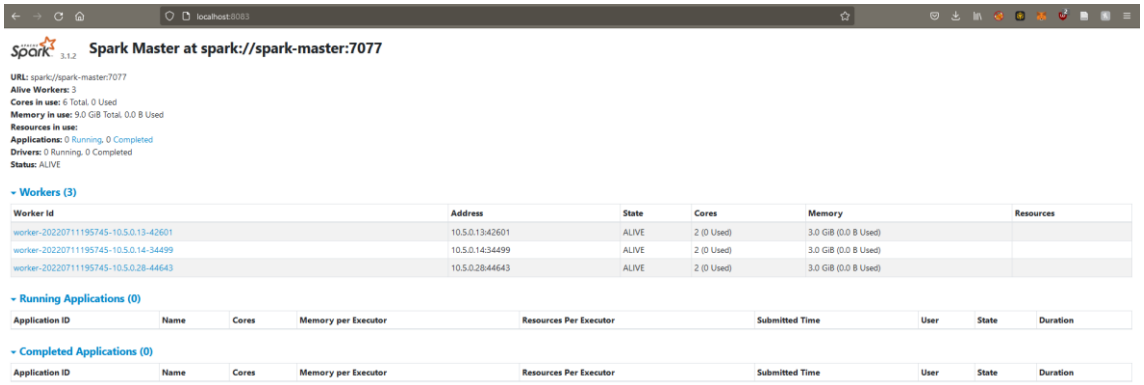


Figura 31. Dashboard Spark

Para o *Spark* obter o acesso às tabelas do *Hive* e ao *HDFS*, foi preciso importar os seguintes ficheiros de configuração [93]:

- *hive-site.xml;*
- *core-site.xml;*
- *hdfs-site.xml;*

4.1.1.4 Configuração do *Presto*

À semelhança da configuração anterior, configuração do *Presto* também foi simples. Para tal, foi necessário fazer alterações ao ficheiro *config.properties* [94]. Essas configurações podem ser consultadas no anexo 8.5. Na imagem abaixo demostra que foram detetados 3 *workers* na arquitetura do *Presto.*

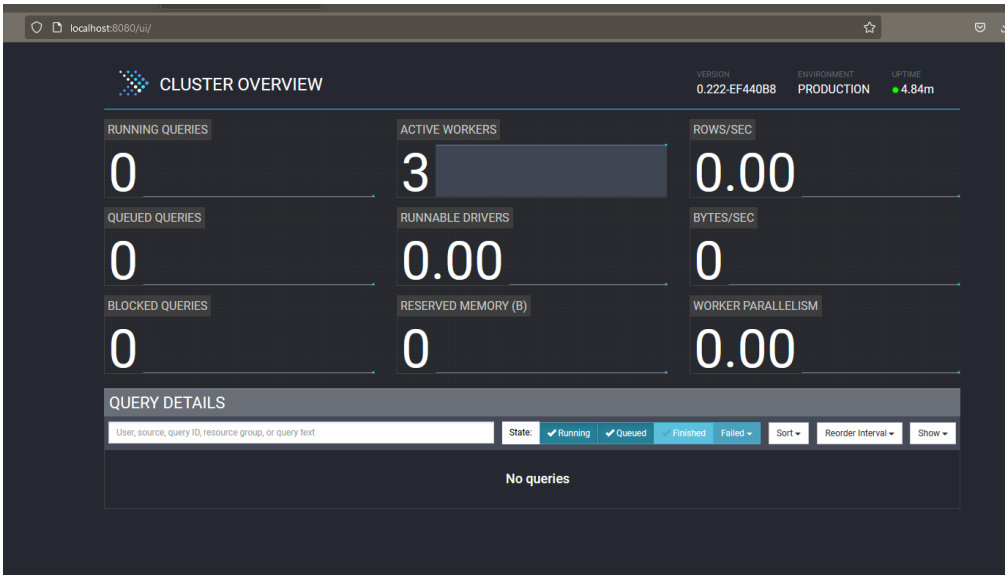


Figura 32. Dashboard Presto

Para o acesso às tabelas do *Hive*, foi criado o ficheiro *hive.properties* e copiado para dentro do *container* as seguintes configurações [95]:

Tabela 4. Variáveis de configuração do Presto

Variaveis:	Valor:	Definição
Conector.name	*Hive-Hadoop2*	Nome do conector
Hive.metastore.uri	*Thrift://hive-metastore:9083*	*URI* onde estão armazenados os metadados do *Hive*

Para testar se a configuração do *Hive* para o *Presto* como esperado foi descarregado um programa em *Java* (*.jar*), que disponibiliza ao utilizador uma interface de linha de comandos (*CLI*) para executar as *queries* pelo *Presto*.

Para aceder à interface, é necessário ter o Java instalado no dispositivo, abrir o terminal e inserir esta linha de comando [96]:

- *java -jar ./Presto.jar --Server localhost:8080 --catalog Hive;*

Após o acesso, verificou-se que se detetam os esquemas existentes no *Hive* utilizando o comando *Show Schemas.* A figura abaixo aborda exemplo de sucesso na deteção dos esquemas existentes em *Hive*:

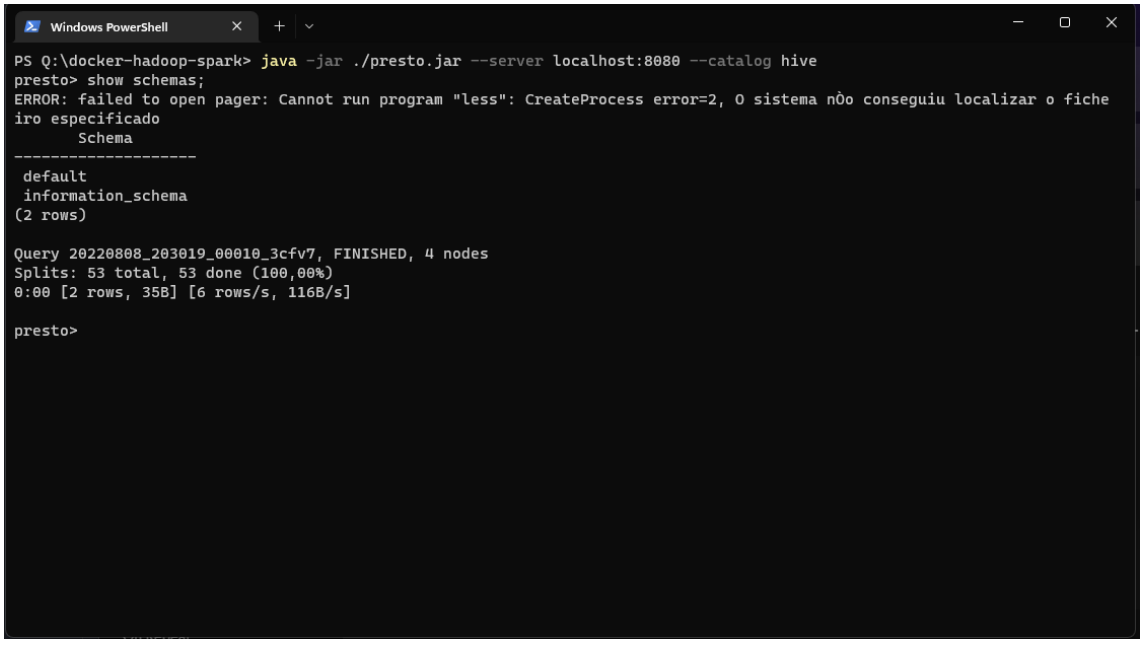


Figura 33. Verificação com sucesso do Hive e Presto.

4.1.1.5 *SQL Server*

Por fim, elaborou-se um container com *SQL Server*. Para configurar a quantidade de núcleos de CPU e de memória bastou ir ao software *Microsoft SQL Server Management Studio*, ir às propriedades da base de dados e configurar como está indicado na imagem abaixo:

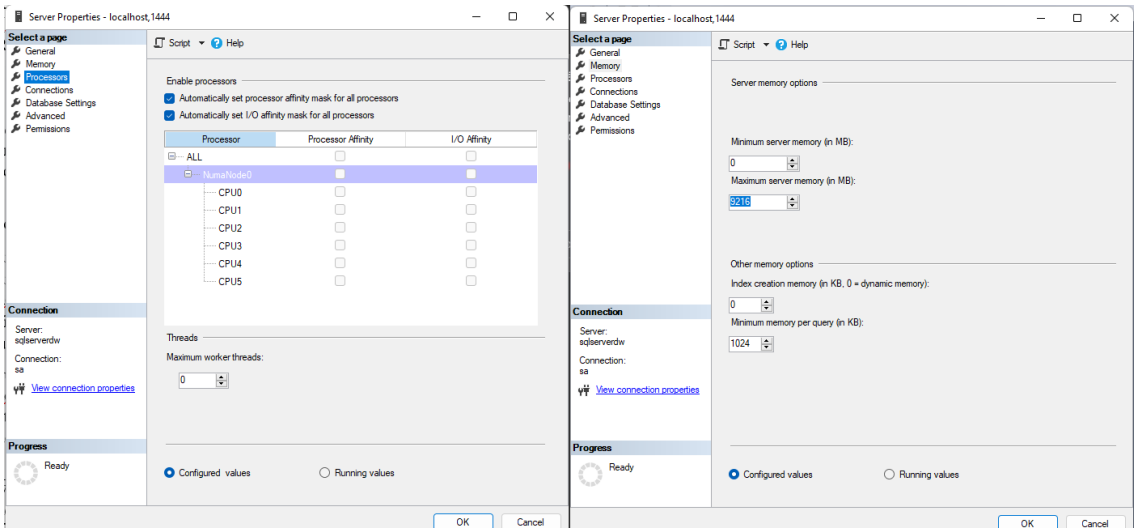


Figura 34. CPU e memória para SQL Server

4.1.2 Elaboração de processos de integração dos dados (*ETL*)

Como foi referido, na fase de *ETL* foi necessário corrigir incoerências e providenciar integrações dos dados, melhorando qualidade para fins analíticos. Antes da fase do *ETL*, os *Datasets* foram preparados de seguinte modo:

- Selecionado, na origem, as colunas úteis para este estudo;
- Agrupar o *Dataset* pelo ano civil (2015 até 2019);
- Importado os *Datasets* para o *SQL Server* e para o *HDFS* em CSV;
- Importado os ficheiros disponibilizados pelo *Google Transit* para o *SQL Server* e *HDFS*;
- Importado dos dados referentes a valores temporais (datas, tempo);
- Importado dos dados relativas às tarifas praticadas pela companhia;
- Em *Hadoop*, foi registado os ficheiros *CSV* em tabelas do *Hive* e convertidos em ficheiros de formato *ORC*.

Na tabela abaixo apresenta um conjunto de *Datasets* agrupados pelo período de ano civil.

Tabela 5. Datasets criados

Nome do *Dataset*	Anos Civis	N.º de linhas
A	2015	18 190 219
B	2015 até 2016	35 912 880
C	2015 até 2017	53 293 745
D	2015 até 2018	70 319 028
E	2015 até 2019	85 381 030

A conversão dos dados de *CSV* para *ORC* foi necessário, graças à capacidade de compressão no armazenamento. Torna-se benéfico na aplicação de grandes quantidades dados em economizar espaço disponível [39].

Os registos obtidos nos sistemas de bilheteira dos autocarros da Horários do Funchal apresentavam problemas que podem ter sido ocorridos por falhas humanas, ou por falhas

do sistema. Após uma análise da existência de registos incompletos, foi imposto um conjunto de diretrizes:

- Remoção de registos caso houvesse valores em falta sobre as paragens e carreiras;
- Substituir os valores em falta das tarifas pelo título do bilhete ou passe;
- Remoção de validações duplicadas;

Além da aplicação dessas diretrizes, também foi criado um conjunto de transformações que ajudam a responder às perguntas de negócio propostas. Para indicar que existe transferência de um autocarro para outro, se verificou se a diferença de tempo entre duas validações é menos de 45 minutos. Caso se verifique, é guardado a carreira e a paragem que foi feito. Em ambas as situações, é guardada a diferença de tempo (em minutos e em horas) entre duas validações pelo mesmo cliente. Na Figura 35 é apresentado um fluxograma do processo de *ETL* feito:

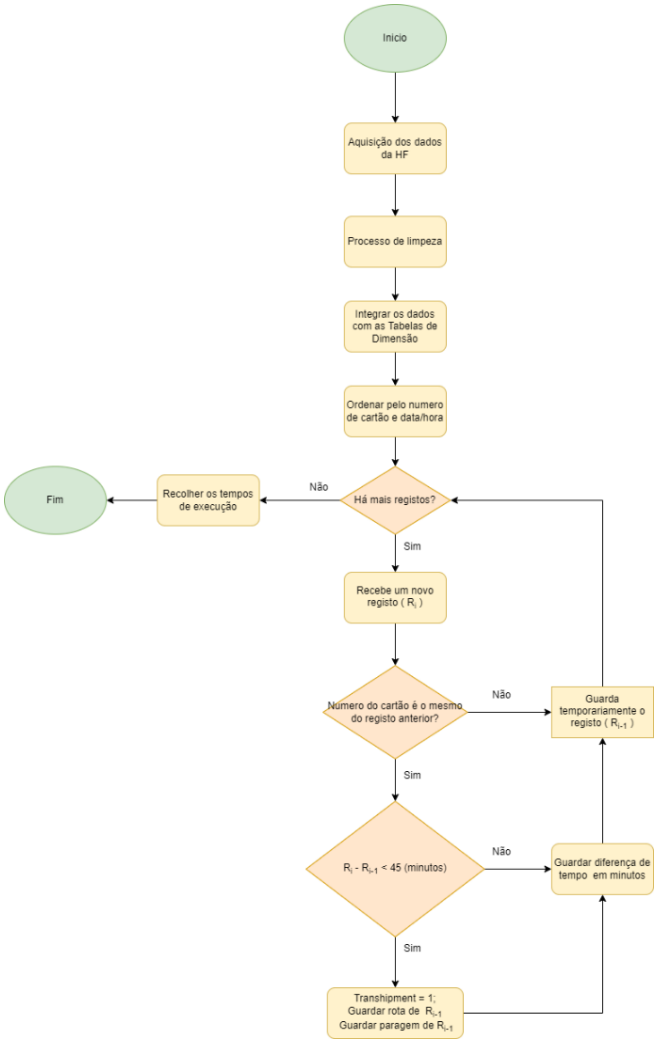


Figura 35. Diagrama de fluxo do processo de ETL.

Após a fase de tratamento, estes foram integrados para o modelo dimensional proposto. Esse modelo foi representado por um esquema em estrela. Esse esquema foi composto por uma tabela central de factos e por cinco tabelas de dimensão.

As tabelas de dimensão foram:

- *DimRoute*: rotas disponíveis pela companhia. Contém atributos como o número da Carreira e o nome do percurso;
- *DimTicket*: corresponde aos bilhetes que a companhia dispõe à venda ao público. Contém informação como o tipo do título e ao tipo de cliente se destina;
- *DimDate* e *DimTime*: referem-se às dimensões de data e de tempo;
- *DimStop*: informação das paragens disponíveis pela cidade do Funchal e noutras localidades interurbanas.

A tabela de factos do modelo em estrela chama-se *FactValidations*. Além de conter atributos referentes às tabelas de dimensão, também continha alguns atributos para dar resposta às perguntas de negócio:

- *Transhipment*: valor 1 se a validação foi feita em menos de 45 minutos da anterior, 0 caso contrário;
- *RouteKeyTranshipment*: o valor caso o atributo *Transhipment* é 1, é inserido a chave da carreira anterior;
- *StopKeyTranshipment*: caso o atributo *Transhipment* é 1, é inserido a chave da paragem anterior.
- *DiffMin*: correspondia à diferença de minutos entre duas validações adjacentes feitas pelo utilizador;
- *DiffHour*: correspondia à diferença de minutos entre duas validações adjacentes feitas pelo utilizador;
- *SentidoPerc*: corresponde ao sentido em que o autocarro percorria.

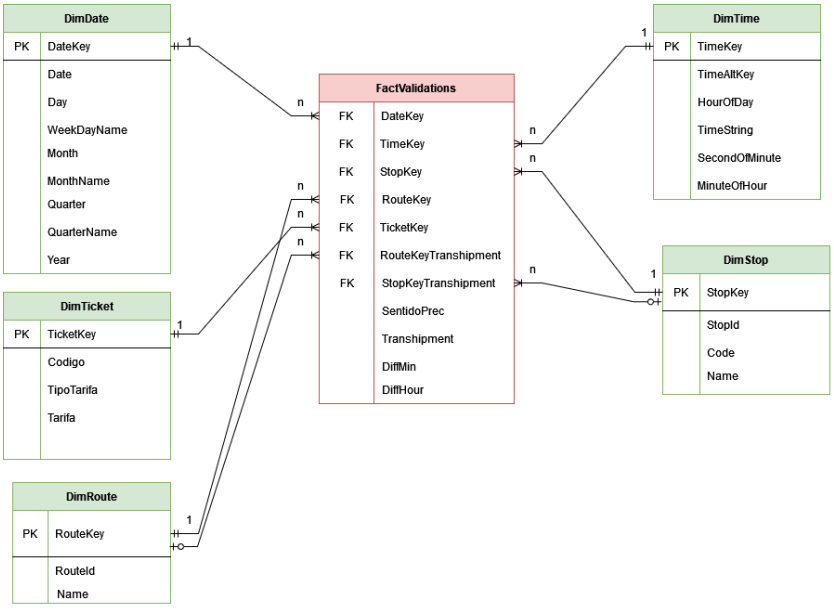


Figura 36. Modelo de dados final.

As secções seguintes correspondem à aplicação desses processos de extração, transformação/integração e carregamento dos dados (*ETL*) para o modelo dimensional acima.

4.1.3 *ETL pelo PySpark*

Para aplicação do processo de *ETL* foi criado um *script* em *Python*. Esse *script* inclui a biblioteca *PySpark*, sendo capaz de elaborar operações *ETL* pelo *Spark SQL* em tabelas do *Hive* e no *HDFS* e processar nos nós do agrupamento do *Spark*. A cada nó escravo foi atribuído 2 núcleos de *CPU* e 3 *GB* de memória. Fazendo num total de 6 núcleos de *CPU* e de 9 *GB* de memória pelo agrupamento. O *script* e o comando de execução do estão disponíveis a partir do anexo 8.6.

O comando de execução do *script* foi composto pelas seguintes opções:

- *--master:* o *URL* do nó mestre do agrupamento;
  - *spark://spark-master:7077*
- *--executor-cores*: número de núcleos de CPU para cada executador;
  - 2
- *--executor-memory*: memoria para cada executador;
  - 3G (3 *GB*)
- *--conf:* inclusão de configurações adicionais, para fins de performance;
  - *spark.sql.adaptive.enabled=true*: habilitar o *Adaptive Query Execution* (*AQE*);
  - *spark.sql.adaptive.skewJoin.enabled=true*: habilitar a redução da distorção dos dados durante as suas agregações.
- *pyspark_etl.py*: *script ETL*.

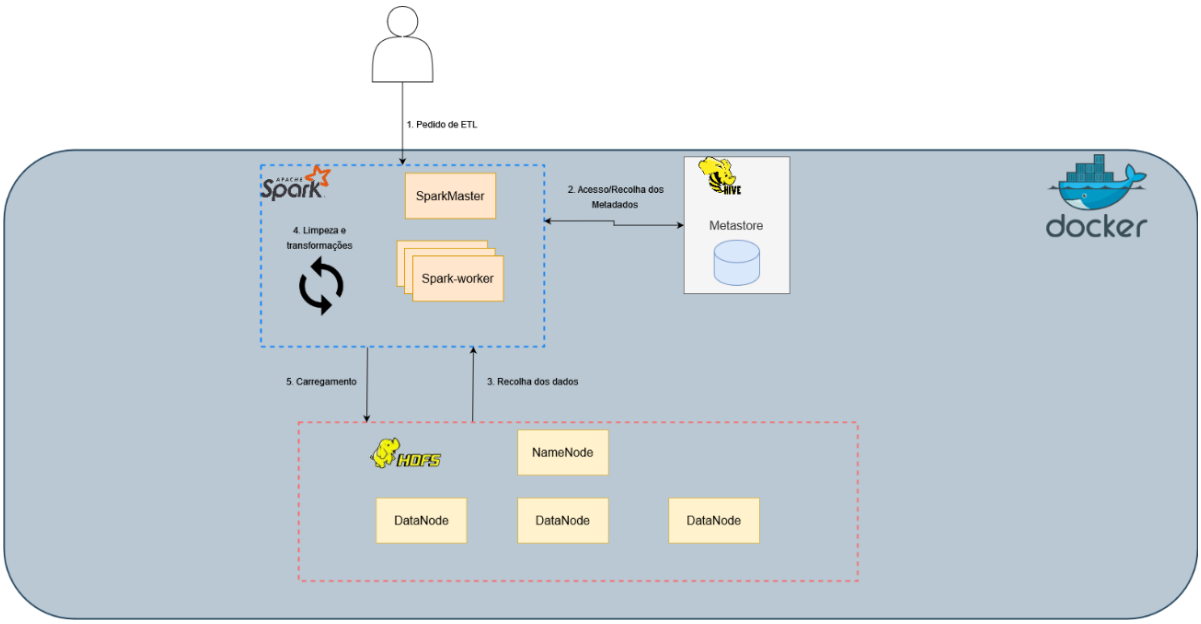


Figura 37. ETL pelo Spark, esquema baseado por [97]

A utilização da *interface Web* tornou-se fundamental no processo de desenvolvimento do *script*, graças à monitorização de tarefas e a representação gráfica do *DAG* na geração do fluxo de dados. Na figura abaixo, encontra-se uma visualização gráfica do *DAG* gerado pelo *Spark*, numa das fases da execução:

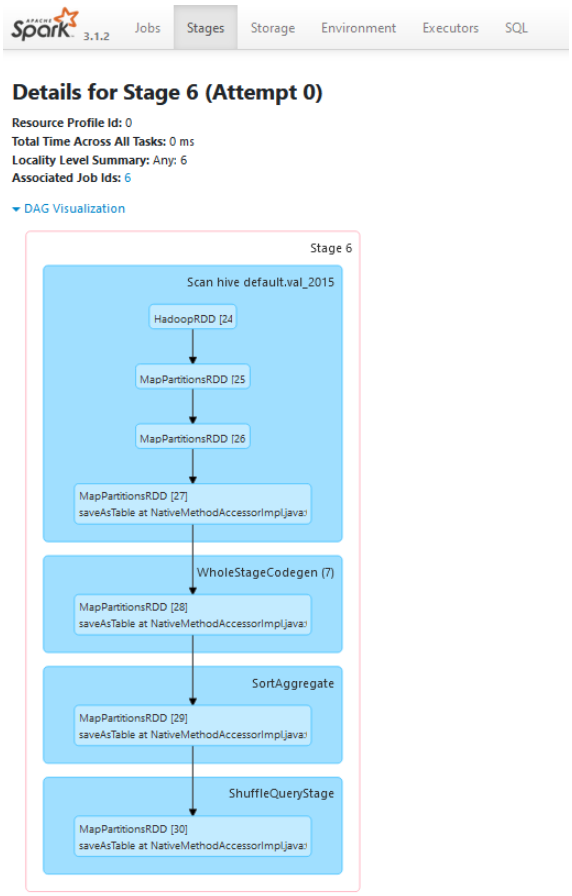


Figura 38.Exemplo do DAG gerado para o ETL.

4.1.4 *ETL* em *SSIS*.

Para desenvolver esta solução, foi necessário instalar o *Visual Studio* e foi incluído o pacote dedicado ao tratamento de dados. A utilização de uma interface gráfica disponibilizada pelo *SSIS* facilitou no desenvolvimento do fluxo de dados nas operações de ordenação (*Sort*) e na integração com as tabelas de dimensão, por *Merge Join*. Todavia, foi necessário desenvolver um *script* em C# para fazer a verificação da existência de transferências pelo cliente. Na figura seguinte representa, graficamente, a composição do fluxo de dados final:

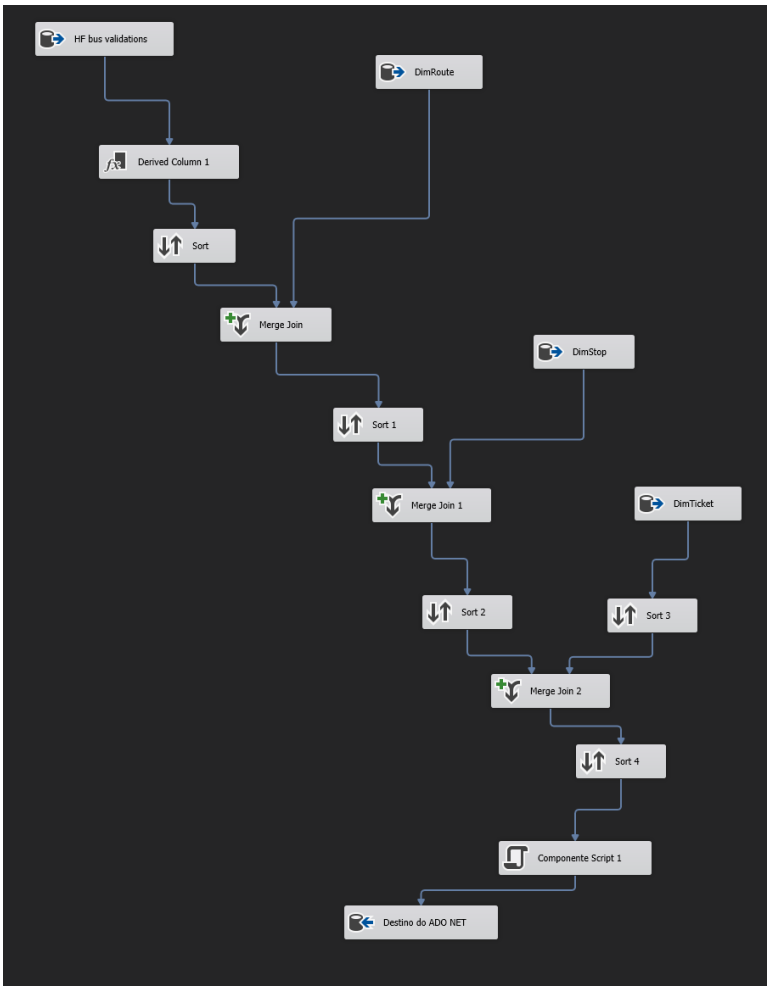


Figura 39. Esquema ETL gráfico pelo SSIS

Para o *deploy* desta solução, foi requerido que o *SQL Server* esteja configurado e com serviços instalados para guardar e processar pacote *SSIS*. Infelizmente, a imagem oficial disponibilizada para o *Docker* utilizada neste trabalho não inclui esses serviços. Após várias tentativas falhadas, a alternativa encontrada para este problema foi utilizar um computador com o *SQL Server* instalado e com *SSIS* configurado. O computador utilizado para o *deploy* dos pacotes *SSIS* foi um portátil com as seguintes características:

- Processador: Intel I7-6500U 2.5GHz;
- Memória: 16GB;
- Comunicação entre os dois computadores é por Ethernet 1GB;

O computador recebia os pacotes de *ETL* criados, executava-os, importava de volta ao container *SQL Server* hospedado noutra máquina. Aplicava as transformações e carregava de volta para o container, para o modelo dimensional.

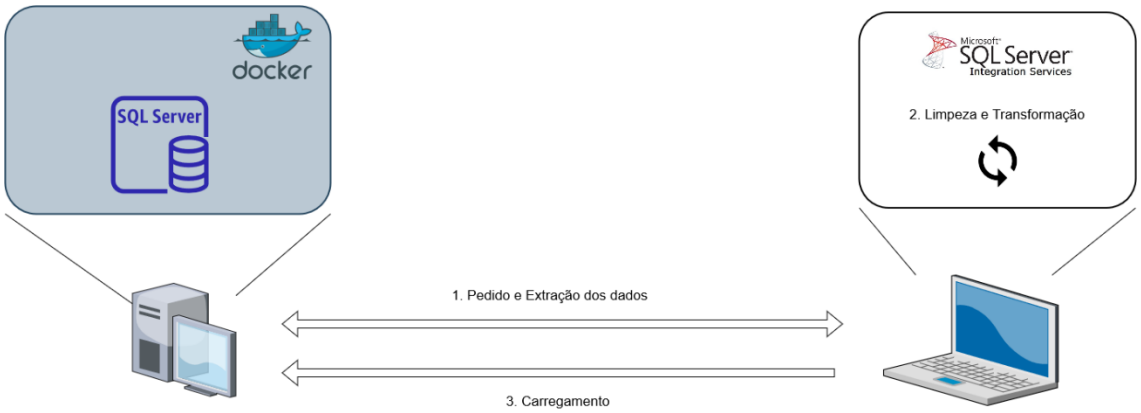


Figura 40. Processo de ETL usando SSIS.

4.1.5 Criação e execução dos indicadores de negócio

Nesta fase foi aplicado um conjunto de *queries* que correspondem às perguntas de negócios anteriormente referidas. Foi desativado, durante a execução e recolha dos resultados temporais, qualquer tipo de armazenamento de cache que influencia diretamente no desempenho da resposta. As tecnologias usadas para execução dessas *queries* foram:

- *Hive* (*Tez* configurado);
- *Spark* (*Spark SQL*, com *Adaptive Query Execution* ativado);
- *Presto;*
- *SQL Server.*

Foram aplicadas conjunto de 4 *queries* em *SQL* para responder, correspondentemente, às perguntas de negócio. A elaboração das *queries* foram feitas de seguinte forma:

4.1.5.1.1 Indicador de negócio 1(Q1):

Foi adquirido a contagem de todos os registos da tabela de factos, o nome da rota e os quadrimestres no *SELECT*. Foi aplicado *INNER JOIN* às tabelas de dimensão referenciadas às Datas e às Rotas.

```
SELECT COUNT(1), rota.route_name, d.quarter, d.quartername
FROM fact_table AS fact
    INNER JOIN dimroute_ORC rota ON fact.routekey = routa.routekey
    INNER JOIN dimdate_ORC d ON fact.datekey = d.datekey
GROUP BY d.quarter, d.quartername, routa.route_name;
```

4.1.5.1.2 Indicador de negócio 2 (Q2):

No *SELECT* foi usado o atributo referente à hora da tabela de dimensão do tempo como medida. Foi usado a função *SUM()* para somar todos os *transhipments* (transferências). Além disso, incluiu a operação *AVG()* para obter a média de tempos entre as transferências. O *ROUND()* correspondeu ao arredondamento das médias às unidades. Todos os registos da tabela de factos foram filtrados quando o valor *transhipment* foi de valor 1, onde ocorreu de transferência.

```
SELECT tempo.hourofday       AS   hora,
    SUM(fact.transhipment) AS   total_de_transferencias,
    ROUND(avg(fact.diffmin), 0) media_de_tempo_de_transferencia
FROM fact_table AS   fact
     INNER JOIN dimtime_ORC tempo on fact.timekey = tempo.timekey
WHERE fact.transhipment = 1
GROUP BY tempo.hourofday;
```

4.1.5.1.3 Indicador de negócio 3 (Q3):

Para obter a resposta à pergunta de negócio foi necessário recorrer a duas *subqueries*:

- Na primeira *subquery* tinha a finalidade de obter o total das validações todas por cada hora do dia, no primeiro e terceiro quadrimestre;
- Na segunda *subquery*, além de conter o resultado de validações totais de cada hora (adquirido na primeira *subquery*), também obteve a quantidade de validações feitas na mesma hora para cada tipo de tarifa. É aplicado operações de *INNER JOIN* em 3 tabelas de dimensão e na primeira *subquery*.

Com os resultados obtidos nas *subqueries*, na operação *SELECT* foi selecionada a tarifa, a hora e a divisão entre as validações por tarifa para obter a percentagem.

```
SELECT final.tarifa, final.hora_tempo, final.contagem_per_tarifa / final.contagem_total
FROM (SELECT count(1) AS contagem_per_tarifa, do.tarifa, tempo_1.hourofday AS hora_tempo, total.contagem_total
    FROM fact_table AS fact
        INNER JOIN dimtime_ORC tempo_1 ON fact.timekey = tempo_1.timekey
        INNER JOIN dimdate_ORC d ON fact.datekey = d.datekey and (d.quarter = 1 OR d.quarter = 3)
        INNER JOIN dimticket_ORC do ON fact.ticketkey = do.ticketkey
        INNER JOIN (SELECT count(1) AS contagem_total, tempo_1.hourofday hora
                FROM fact_table AS fact_1
                    INNER JOIN dimdate_ORC ON dimdate_ORC.datekey = fact_1.datekey AND
                                (dimdate_ORC.quarter = 1 OR dimdate_ORC.quarter = 3)
                    INNER JOIN dimtime_ORC tempo_1 ON fact_1.timekey = tempo_1.timekey
                GROUP BY tempo_1.hourofday) as total ON total.hora = tempo_1.hourofday
    GROUP BY do.tarifa, tempo_1.hourofday, total.contagem_total) final;
```

4.1.5.1.4 Indicador de negócio 4 (Q4):

Na declaração *SELECT*, além de conter a tarifa, também incluiu um conjunto de operações de soma. Dentro dessas operações contém operação *CASE WHEN* resultando 1 ou 0 se corresponde aos parâmetros propostos. Os motivos utilizados para esta *query* são Estudar, Trabalho ou Outro.

- Os critérios para o motivo for “Estudar”:

    - A tarifa tem de ser de estudante ou de criança;
    - A diferença de tempo entre os dois pontos tem de ser entre 2 horas (120 minutos) e 6 horas (360 minutos);
- Critérios para o motivo “Trabalho”:
    - A tarifa tem de ser tipo Normal;
    - A diferença de tempo entre os dois pontos tem de ser entre 2 horas (120 minutos) e 7 horas (420 minutos);
- Critérios para motivo “Outro”:
    - Quando o registo não existe diferença de tempo;
    - Quando a diferença de tempo é inferior a 2 horas e superior a 7 horas;
    - Quando a tarifa é tipo Idoso ou Reformado.

```
SELECT dimticket_ORC.tarifa,
    SUM (CASE
      WHEN (fact.diffmin >= 120 AND fact.diffmin <= 360) AND
        dimticket_ORC.tarifa = 'Estudante' THEN 1
      WHEN (fact.diffmin >= 120 AND fact.diffmin <= 360) AND
        dimticket_ORC.tarifa = 'Crianca' THEN 1
      ELSE 0
      END) AS Estudar,
      SUM (CASE
      WHEN (fact.diffmin >= 120 AND fact.diffmin < 420) AND
        dimticket_ORC.tarifa = 'Normal' THEN 1
      ELSE 0
      END) AS trabalho,
        SUM (CASE
          WHEN (fact.diffmin <= 120 or fact.diffmin >= 420) THEN 1
          WHEN dimticket_ORC.tarifa = 'Idoso' or dimticket_ORC.tarifa = 'Reformado'   THEN 1
          WHEN fact.diffmin is null THEN 1
      ELSE 0
      END) AS outro_motivo
from fact_table AS fact,
   dimticket_ORC
where fact.ticketkey = dimticket_ORC.ticketkey
group by dimticket_ORC.tarifa;
```

5 Avaliação

Com o desenvolvimento do projeto feito e com a recolha dos valores de desempenho, neste capítulo é efetuada uma avaliação comparativa dos resultados obtidos, desde as capacidades de armazenamento, o desempenho na introdução dos processos *ETL* para a limpeza e integração dos dados para o modelo dimensional final, até ao desempenho na execução das *queries* relativas aos indicadores de negócio na área dos transportes públicos.

5.1 Capacidade de armazenamento.

Como já foi referido, os ficheiros *ORC* ajudam, em maior parte dos casos, na obtenção de melhores resultados de desempenho, em comparação com a utilização de ficheiros tipo texto, facilitando a análise de dados [39]. O gráfico abaixo refere-se à comparação da capacidade de armazenamento utilizado, por *ORC* e *SQL Server*, consoante ao aumento progressivo do volume de dados antes de aplicar *ETL*:

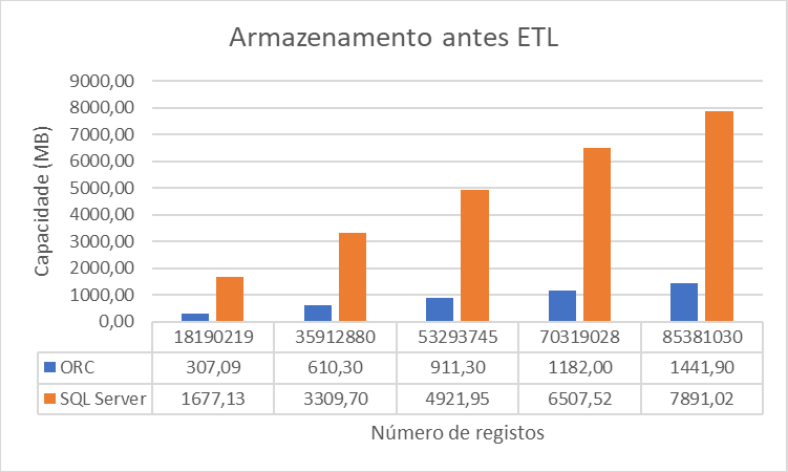


Figura 41. Comparação gráfica da capacidade de armazenamento antes do ETL

No gráfico verificou-se que a utilização dos ficheiros *ORC* reverteu para uma poupança de armazenamento perto dos 82%, face ao *SQL Server*. Com isso, reverteu para uma maior economia de espaço, levando ao utilizador a não se preocupar com o volume de dados a ser consumido.

Tabela 6. Diferença de capacidade antes do ETL entre ORC e SQL Server

Dataset	*ORC* (MB)	*SQL Server* (MB)	Diferença (%)
A	307,09	1677,13	81,69
B	610,30	3309,70	81,56
C	911,30	4921,95	81,48
D	1182,00	6507,52	81,84
E	1441,90	7891,02	81,73
		Média	81,66

Após a aplicação dos processos *ETL* e da integração dos dados ao modelo dimensional proposto, verificou-se que existiu uma maior redução de armazenamento pelo *SQL Server*, em média, nos 41% face ao *ORC*, perto de 17%.

Tabela 7. Diferença de capacidade entre antes e após ETL

Dataset	Diferença em *ORC* (%)	Diferença em *SQL Server* (%)
A	17,72	42,01
B	18,08	42,09
C	16,90	41,31
D	14,78	41,03
E	16,19	40,28
Média	16,73	41,34

Isso revelou que, a utilização de modelos dimensionais para as bases de dados Relacionais, como o *SQL Server*, em termos de armazenamento, tem um impacto maior em comparação ao *ORC*. É notada pouca diferença em *ORC* devido à utilização de algoritmos de compressão.

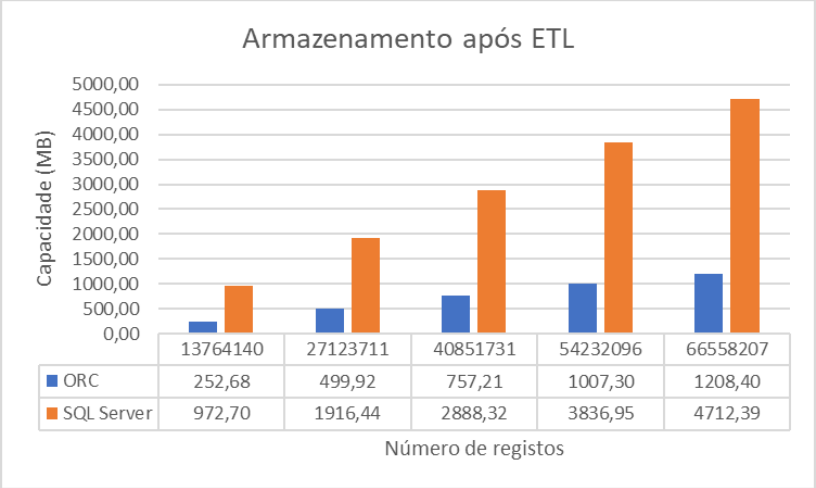


Figura 42. Comparação gráfica da capacidade de armazenamento após o ETL.

5.2 Comparação temporal do processo *ETL*

Aplicando o processo de *ETL* para integrar os dados e carregar os dados no modelo dimensional, obtiveram-se diferenças notórias do tempo de execução entre as duas tecnologias.

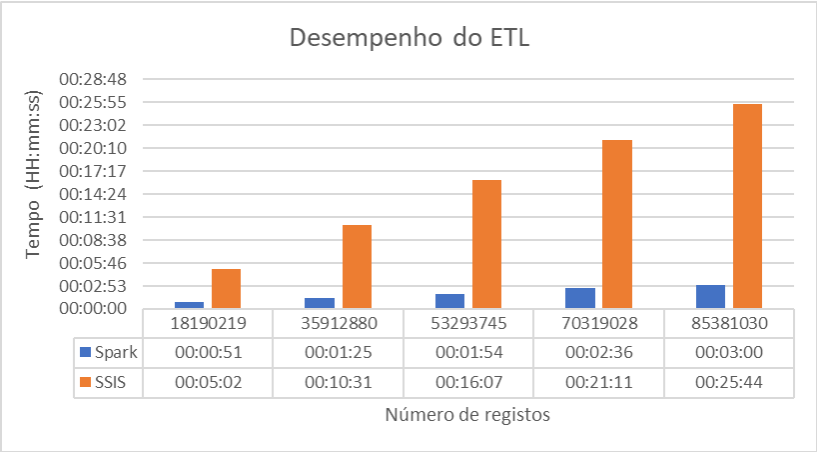


Figura 43. Comparação gráfica entre os tempos de desempenho entre Spark e SSIS.

A capacidade de computação em paralelo entre os nós do agrupamento pelo *Spark* e juntamente com a representação em tabelas através do *Hive* em ficheiros *ORC* ajudaram a processar grandes volumes de dados em tempos mais reduzidos face à utilização do *SSIS*. Como base nos resultados apresentados na Tabela 8 a percentagem média de diferença entre os tempos de desempenho de ambas tecnologias foi de 86,80%.

Tabela 8. Diferença temporal entre Spark e SSIS

Dataset	*Spark* (HH:mm:ss)	*SQL Server* (HH:mm:ss)	Diferença (%)
A	00:00:51	00:05:02	83,11
B	00:01:25	00:10:31	86,52
C	00:01:54	00:16:07	88,21
D	00:02:36	00:21:11	87,72
E	00:03:00	00:25:44	88,34
		Média	86,80

Uma das razões que está a contribuiu para o maior tempo de processamento por parte do *SSIS* face ao *Spark*, foi a necessidade de, em cada operação de *Merge Join* entre dois fluxos de dados, a obrigatoriedade de adicionar operação de *Sort* previamente.

Ao incluir, o fluxo de dados ficará bloqueado até ambas as ordenações fiquem completas [98]. Mesmo que o processo *ETL* esteja hospedado num computador com maior quantidade de memória em comparação ao agrupamento do *Spark*, não ajuda na redução do aumento do tempo do processo por parte do *SSIS*.

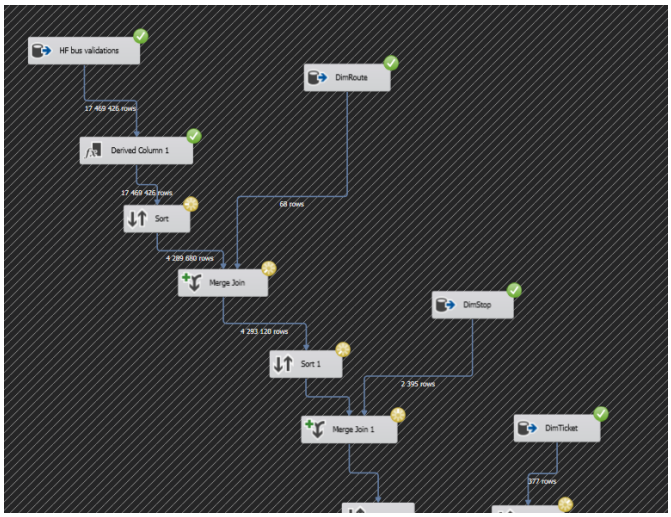


Figura 44. Exemplo durante a execução operação Join

Com os processos executados, na Tabela 9 mostra a diferença da quantidade de dados antes e após o *ETL*, resultando numa diferença media de -23,39% do volume inicial.

Tabela 9. Diferença do número de dados após ETL

*Data*set	N.º linhas antes *ETL*	N.º linhas após *ETL*	Diferença (%)
A	18190219	13764140	-24,33
B	35912880	27123711	-24,47
C	53293745	40851731	-23,34
D	70319028	54232096	-22,87
E	85381030	66558207	-22,04
		Média	- 23,39

5.3 Comparação dos dados temporais de desempenho.

Nesta secção, pretende-se uma avaliação comparativa dos resultados temporais recolhidos durante o processo de execução das *queries* ao longo do aumento do volume de dados. Com a execução das *queries* referentes aos indicadores de negócio na área dos transportes públicos propostos obtiveram-se resultados interessantes em todos os sistemas testados.

Numa primeira vista, o *Hive* foi mais lento em comparação às outras tecnologias em estudo. Em Q1 e Q3, o *Hive* obteve grandes diferenças de tempo de execução em comparação aos outros, chegando a tempos superiores de 1 minuto e 2 minutos de execução, respetivamente. Um dos motivos deveu-se ao facto de que Q1 e Q3, contêm várias operações de *INNER JOIN*. Com base no estudo de Carlos Costa *et al.* [99], o desempenho do *Hive* degradou-se consoante a quantidade de operações de *JOINS.*.

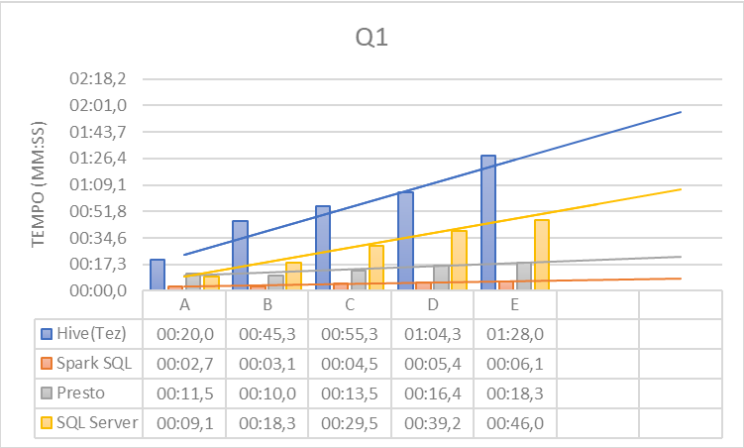


Figura 45. Resultados temporais em Q1

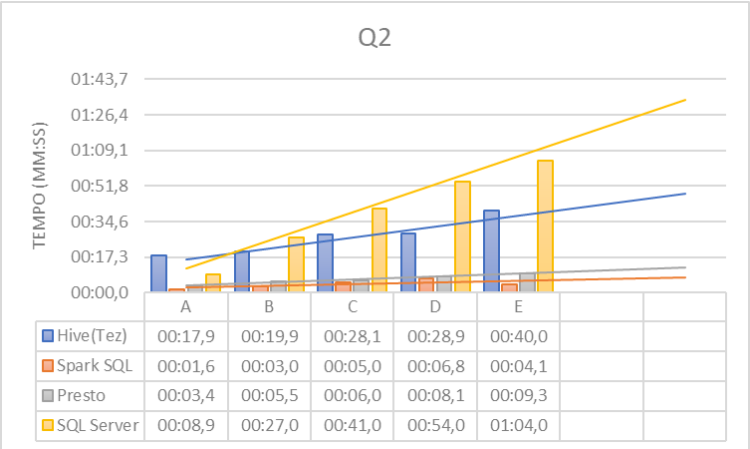


Figura 46. Resultados temporais em Q2

O SQL Server foi o próximo a ter pouco desempenho com o aumento do volume, mas em lotes iniciais obteve resultados muito melhores. Até em situações em que o seu desempenho foi superior ao *Presto* (em Q1) e ao *Spark* (em Q3), que utilizam intensivamente CPU e memória.

A partir do segundo lote de volume de dados, começou a notar-se uma degradação de desempenho em todas as *queries* de negócio, retornando no final, indicadores temporais pouco práticos para uso normal do utilizador. Sendo que, a maior de desempenho foi em Q2 e Q4, acabando por ter valores temporais superiores e semelhantes ao *Hive*. As operações utilizadas em Q2 e Q4 (*SUM()*, *ROUND()*, *AVG()* e *CASE()*), poderá ter impacto direto de uma maior degradação de desempenho para *SQL Server*, em comparação a Q1 e Q3.

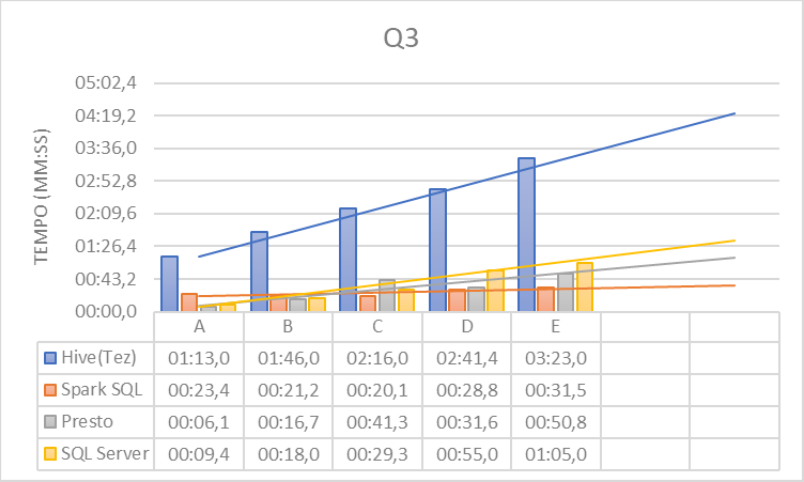


Figura 47. Resultados temporais em Q3

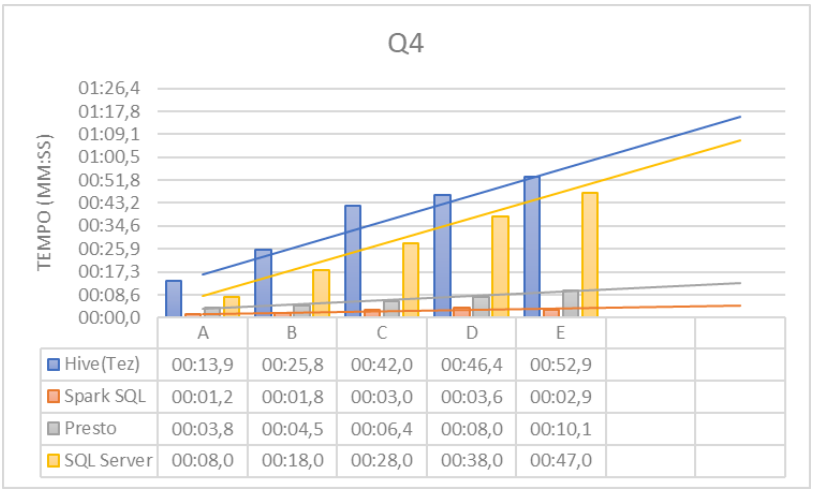


Figura 48. Resultados temporais em Q4

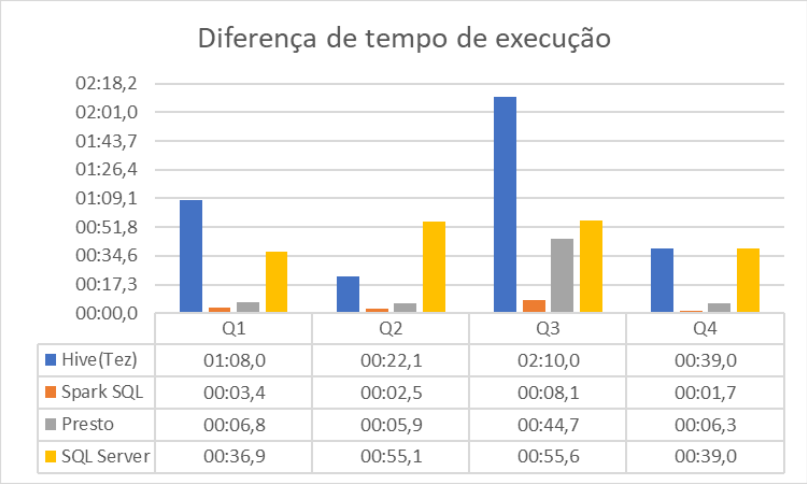
O Presto e o *Spark* provaram ser boas escolhas ao nível de desempenho, consoante ao aumento do volume de dados, sendo o *Spark* o mais vantajoso. Em ambos os casos, as suas variações de tempo de execução entre *Datasets* não foram grandes comparando ao *SQL Server* e o *Hive*, com a exceção no *Presto* no *Dataset* C em Q3. Durante a execução do Q3, o *Presto* teve dificuldades em obter resultados a partir do *Dataset* C, sendo necessário reiniciar todos os nós que pertencem ao seu agrupamento e repetir a execução da *query*. Algumas falhas na configuração dos recursos disponibilizados podem ser a causa das constantes falhas.

Utilizando regressão linear, para aplicar uma previsão do comportamento das tecnologias consoante o aumento dos dados em todas as *queries* de negócio, o *SQL Server* e o *Hive* continuaram com grandes aumentos temporais em comparação com as outras tecnologias, tendo em conta que em Q4 *SQL Server* acabaria por ultrapassar o *Hive*, o *Spark* e o *Presto* acabarão por manter a sua diferença de desempenho entre *Data*sets reduzida.

Aplicando uma diferença de tempos de execução (ver Figura 49) entre o primeiro *Dataset* (13 milhões de validações), com o último *Dataset* (de 66 milhões), o *Hive*, na maioria dos

casos, obteve uma grande diferença do tempo, com a exceção em Q2 e Q4 que obteve, correspondentemente, uma diferença menor e igual ao *SQL Server*.

O *SQL Server* ficou a seguir do *Hive* com maior diferença de tempo entre os *Datasets*, frisando que as utilizações de sistemas de base de dados relacionais poderão tronarem-se obsoletas face ao aumento do volume de dados. O *Spark* e o *Presto* foram as tecnologias em que a sua diferença de tempo é reduzida, sendo que o *Spark* que obteve a menor diferença entre todas as tecnologias testadas. *Presto*, em Q3 teve a maior diferença em comparação às outras *queries*, levando à conclusão de que a utilização de *subqueries* tem um impacto significativo de desempenho.



	Q1	Q2	Q3	Q4
Hive(Tez)	01:08,0	00:22,1	02:10,0	00:39,0
Spark SQL	00:03,4	00:02,5	00:08,1	00:01,7
Presto	00:06,8	00:05,9	00:44,7	00:06,3
SQL Server	00:36,9	00:55,1	00:55,6	00:39,0

Figura 49. Resultados diferenças temporais.

5.4 Conclusão

Com base nos resultados obtidos durante a fase de desenvolvimento, nesta secção foi retratada a avaliação da capacidade de armazenamento e do desempenho das tecnologias em estudo.

A utilização de ficheiros de formato *ORC* tornam-se uma mais-valia para empresas que recolhem grandes quantidades de dados e sem se preocupar com a falta de capacidade de armazenamento graças a utilização de algoritmos de compressão de ficheiros. Ao nível da capacidade de volume dos dados armazenados, *SQL Server* não conseguiu acompanhar ao *ORC*, obtendo valores acima dos 7GB em comparação a 1GB de *ORC*.

A aplicação de processos de *ETL* é importante para a remoção e integração dos dados ao modelo dimensional final. Em comparação aos tempos de desempenho de execução desses processos, a utilização do processamento em paralelo da plataforma *Spark* teve melhores desempenhos de execução, por uma grande margem, em comparação à utilização do *SSIS*. A utilização de operações *MERGE JOIN* e *SORT* em *SSIS* pode ter comprometido, de forma direta, no seu desempenho dos processos de transformação e integração dos dados. Na vertente no armazenamento dos dados, integração dos dados para o modelo dimensional teve um contributo significativo na qualidade de armazenamento por parte do *SQL Server*, face ao *ORC*.

Em comparação aos valores de desempenho na execução das *queries* de negócio, o *Spark* e o *Presto* foram as tecnologias que tiveram melhores resultados de desempenho, em comparação ao *Hive* e ao *SQL Server*, sendo que, a execução dos *Spark* foi a melhor.

O Hive e *SQL Server* foram as tecnologias que tiveram a maior degradação de desempenho consoante ao volume de dados. Em volumes iniciais, *SQL Server* foi capaz de corresponder a valores temporais de desempenho semelhantes ao *Spark* e ao *Presto*, mas ao longo do aumento gradual do volume, houve uma degradação do seu desempenho. Em ambas as tecnologias, em *Datasets* finais, retornaram valores temporais pouco práticos para um uso regular por parte dos utilizadores.

A aplicação de uma regressão linear é um bom complemento para fazer uma previsão de que como as tecnologias avaliadas se irão se comportar futuramente, mas não atendendo em consideração à capacidade dos recursos disponíveis, poder-se-á deduzir em erro esses resultados. Tecnologias como o *Spark* e *Presto* podem ocorrer falhas de execução de *queries* devido à limitação de recursos existentes.

6 Conclusão

Este projeto teve como objetivo em avaliar o comportamento de um conjunto de tecnologias em ambiente de *Big Data*, relativamente aos dados de mobilidade na área dos transportes públicos. Após a recolha de informações que foram apresentadas na revisão de literatura, desenvolveu-se, utilizando o *Docker*, vários sistemas de forma a simular um ambiente real. As tecnologias utilizadas para este estudo foram *SQL Server*, *Hadoop*, *Hive*, *Spark* e *Presto*. Os dados utilizados para este estudo pertencem à companhia de transporte público do Funchal, referentes ao registo de estradas dos autocarros por parte dos clientes.

Para além da recolha dos dados de mobilidade e do desenvolvimento do ambiente de teste em *Big Data*, foi necessário desenvolver dois processos de extração, tratamento e carregamento dos dados (*ETL*), em *PySpark* e *SSIS*, para fazer as transformações e as integrações necessárias para serem carregados no modelo dimensional final.

Durante o processo de desenvolvimento, houve um processo demorado nas configurações das plataformas *Hadoop, Spark e Presto*, mais em concreto na disponibilização dos recursos que os nós do agrupamento que poderiam utilizar, face ao *SQL Server* que apresenta uma *interface* gráfica mais intuitiva.

A avaliação foi dividida em três fases: capacidade de armazenamento, o processo *ETL* e o tempo de execução dos indicadores de negócio. No que se refere à capacidade de armazenamento, a utilização de ficheiro *ORC* para o armazenamento dos dados iniciais acarreta vantagens em comparação ao armazenamento pelo *SQL Server*. Isso deve-se ao resultado na aplicação de algoritmos de compressão dos dados em *ORC*. A aplicação de processos *ETL* trouxe grande impacto no armazenamento no *SQL Server*, mas não conseguiu equiparar aos *ORC*.

Na fase de comparação do *ETL*, a utilização de uma interface gráfica para o desenvolvimento de *pipelines* para o tratamento dos dados foi uma grande vantagem em *SSIS*, com a exceção da elaboração de um *script* em C#, aplicação dos blocos tornou-se muito intuitivo e fácil. Em comparação ao *Spark*, foi necessário elaborar, de raiz, um *script* em *Python* utilizando a biblioteca *PySpark*. A nível de desempenho, aplicação de um sistema distribuído e o processamento em paralelo pelo *Spark* obtiveram grande impacto em desempenho do tratamento e integração dos dados, em comparação ao *SSIS*. Em situações em que o volume se torna cada vez maior e a necessidade da velocidade de processamento é cada vez maior, *Spark* torna-se uma boa alternativa na aplicação de *ETL*.

Relativamente à comparação dos tempos de execução dos indicadores de negócio criados, *Hive* e *SQL Server* foram os que obtiveram valores temporais pouco práticos. A utilização de operações de *Join* tem um impacto de degradação de desempenho para o *Hive*, já a utilização de operações *SUM()*, *AVG()*, *ROUND()* e *CASE()* tem impactos para o desempenho de *SQL Server*. De todos, o sistema *Spark* provou que consegue manter os tempos de execução em valores mais aceitáveis. A aplicação de uma regressão linear tornou-se útil para a previsão do comportamento das tecnologias em estudo, face ao aumento da quantidade de dados. Infelizmente, não é possível dar por garantido, devido à limitação de recursos disponíveis para dados futuros.

Com a elaboração das fases de comparação, pode-se concluir que, a melhor abordagem para este estudo de caso com dados de mobilidade pela companhia Horários do Funchal, consiste em armazenar os dados no sistema de ficheiros do *Hadoop* (*HDFS*) em formato *ORC* com a representação dos mesmos em tabelas do *Hive* e por último utilizar *Apache Spark* para desenvolver processos de *ETL* e executar *queries* analíticas.

6.1 Limitações e trabalho futuro:

Durante o desenvolvimento do trabalho, ocorreram um conjunto de limitações:

- Houve um grande consumo de tempo para a preparação durante a configuração e alocação dos recursos, quer no *Docker*, quer no *Hadoop*, *Spark* e *Presto*;
- Na tentativa de elaboração em MongoDB, houve um conjunto de falhas desde conexão, as falhas dos containers durante a importação levaram à desistência;
- Várias tentativas falhadas na utilização de sistemas *OLAP* e de *Bussiness Inteligence*;
- Na pergunta de negócio Q4 não foi incluído, em motivo de Estudo, verificar se as validações ocorrem nos dias uteis;
- Muitas das vezes, durante o processo de execução do *ETL*, dos *Data*sets de maior volume, através do *Spark*, o *Docker* deixava de responder o que resultava em reiniciar a máquina hospedeira várias vezes;
- Durante a execução do indicador de negócio Q3, o *Presto* deixava de responder, resultando na necessidade de reiniciar os *containers* relacionados;
- A base de dados disponibilizada pela companhia Horários do Funchal é bastante complexa, resultando de um grande consumo de tempo em analisar e recolher os dados necessários para o trabalho.

Para trabalho futuro, seria interessante aplicar esta análise comparativa em ambientes físicos de *Big Data*, com várias máquinas distribuídas por vários agrupamentos em maior escala. Seria também de experimentar, nesses ambientes, outros indicadores de negócio na área dos transportes públicos, *Data*sets de grandes capacidades e utilização de outros sistemas na área do *Big Data* não utilizados neste trabalho.

7 Referencias

[1] M.-P. Pelletier, M. Trépanier, e C. Morency, «Smart card data use in public transit: A literature review», *Transportation Research Part C: Emerging Technologies*, vol. 19, n.º 4, pp. 557–568, ago. 2011, doi: 10.1016/j.trc.2010.12.003.

[2] A. Vaisman e E. Zimányi, *Data Warehouse Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. doi: 10.1007/978-3-642-54655-6.

[3] «Big-Data-Policy-Brief.pdf». Acedido: 3 de setembro de 2022. [Em linha]. Disponível em: https://www.apta.com/wp-content/uploads/Big-Data-Policy-Brief.pdf

[4] K. Krishnan, «Data Warehousing in the Age of Big Data», p. 583.

[5] W. H. Inmon, *Building the data warehouse*, 4th ed. Indianapolis, Ind: Wiley, 2005.

[6] C. Ballard, D. Herreman, D. Schau, R. Bell, E. Kim, e A. Valencic, «Data Modeling Techniques for Data Warehousing», p. 216.

[7] P. Makele e S. Doss, «A Survey on Data Warehouse Approaches for Higher Education Institution», *International Journal of Innovative Research in Applied Sciences and Engineering*, vol. 1, n.º 11, p. 223, mai. 2018, doi: 10.29027/IJIRASE.v1.i11.2018.223-227.

[8] R. Kimball e M. Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, 2011.

[9] D. L. Moody, «From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design», p. 12.

[10] A. Vaisman e E. Zimányi, *Data Warehouse Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. doi: 10.1007/978-3-642-54655-6.

[11] L. Yessad e A. Labiod, «Comparative study of data warehouses modeling approaches: Inmon, Kimball and Data Vault», em *2016 International Conference on System Reliability and Science (ICSRS)*, Paris, France: IEEE, nov. 2016, pp. 95–99. doi: 10.1109/ICSRS.2016.7815845.

[12] M. Golfarelli e S. Rizzi, «From Star Schemas to Big Data: 20 $$+$$ Years of Data Warehouse Research», em *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, S. Flesca, S. Greco, E. Masciari, e D. Saccà, Eds., em Studies in Big Data, vol. 31. Cham: Springer International Publishing, 2018, pp. 93–107. doi: 10.1007/978-3-319-61893-7_6.

[13] A. De Mauro, M. Greco, e M. Grimaldi, «What is big data? A consensual definition and a review of key research topics», apresentado na INTERNATIONAL CONFERENCE ON INTEGRATED INFORMATION (IC-ININFO 2014): Proceedings of the 4th International Conference on Integrated Information, Madrid, Spain, Madrid, Spain, 2015, pp. 97–104. doi: 10.1063/1.4907823.

[14] M. Sogodekar, S. Pandey, I. Tupkari, e A. Manekar, «Big data analytics: hadoop and tools», em *2016 IEEE Bombay Section Symposium (IBSS)*, Baramati, India: IEEE, dez. 2016, pp. 1–6. doi: 10.1109/IBSS.2016.7940204.

[15] X. L. Dong e D. Srivastava, «Big data integration», em *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, Brisbane, QLD: IEEE, abr. 2013, pp. 1245–1248. doi: 10.1109/ICDE.2013.6544914.

[16] A. Oussous, F.-Z. Benjelloun, A. Ait Lahcen, e S. Belfkih, «Big Data technologies: A survey», *Journal of King Saud University - Computer and Information Sciences*, vol. 30, n.º 4, pp. 431–448, out. 2018, doi: 10.1016/j.jksuci.2017.06.001.

[17] H. M. S. Ali, M. J. Arshad, e I. A. Sumra, «7, Vs of Big Data: A Survey», p. 6.

[18] W. Ali, M. U. Shafique, M. A. Majeed, e A. Raza, «Comparison between SQL and NoSQL Databases and Their Relationship with Big Data Analytics», *AJRCoS*, pp. 1–10, out. 2019, doi: 10.9734/ajrcos/2019/v4i230108.
[19] «BIG DATA ANALYTICS», *IIS*, 2015, doi: 10.48009/2_iis_2015_81-90.
[20] «Survey on Hadoop and Introduction to YARN», *IJETAE*.
[21] M. Grover, T. Malaska, J. Seidman, e G. Shapira, «Hadoop Application Architectures», p. 563.
[22] R. Jagadale e P. Adkar, «A Review Paper on Big data & Hadoop», *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 6, n.º 5, p. 6.
[23] J. Singh, «Big Data: Tools and Technologies in Big Data», *International Journal of Computer Applications*, vol. 112, n.º 15, p. 5.
[24] «Apache Hadoop 3.3.1 – HDFS Architecture». https://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (acedido 10 de fevereiro de 2022).
[25] P. S. Honnutagi, «The Hadoop distributed file system», vol. 5, p. 6, 2014.
[26] J. Dean e S. Ghemawat, «MapReduce: simplified data processing on large clusters», *Commun. ACM*, vol. 51, n.º 1, pp. 107–113, jan. 2008, doi: 10.1145/1327452.1327492.
[27] B. Bengfort, «Data Analytics with Hadoop», p. 288.
[28] T. White, «Hadoop: The Definitive Guide», p. 805.
[29] X. Qin *et al.*, «Beyond Simple Integration of RDBMS and MapReduce -- Paving the Way toward a Unified System for Big Data Analytics: Vision and Progress», em *2012 Second International Conference on Cloud and Green Computing*, Xiangtan, Hunan, China: IEEE, nov. 2012, pp. 716–725. doi: 10.1109/CGC.2012.39.
[30] S. Shaikh e D. Vora, «YARN versus MapReduce - A Comparative Study», em *2014 International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi: IEEE, mar. 2014.
[31] «Apache Hadoop YARN – Background and an Overview», *Cloudera Blog*, 7 de agosto de 2012. https://blog.cloudera.com/apache-hadoop-yarn-background-and-an-overview/ (acedido 21 de junho de 2021).
[32] «Apache Hadoop 3.3.1 – Apache Hadoop YARN». https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html (acedido 20 de junho de 2021).
[33] Asso.Prof. Ashish Sharma e Snehlata Vyas, «Hadoop2 Yarn», *IPASJ INTERNATIONAL JOURNAL OF COMPUTER SCIENCE(IIJCS)*, pp. 018–026, setembro de 2015.
[34] A. Thusoo *et al.*, «Hive - a petabyte scale data warehouse using Hadoop», em *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, Long Beach, CA, USA: IEEE, 2010, pp. 996–1005. doi: 10.1109/ICDE.2010.5447738.
[35] A. Floratou, U. F. Minhas, e F. Özcan, «SQL-on-Hadoop: full circle back to shared-nothing database architectures», *Proc. VLDB Endow.*, vol. 7, n.º 12, pp. 1295–1306, ago. 2014, doi: 10.14778/2732977.2733002.
[36] E. Capriolo, D. Wampler, e J. Rutherglen, *Programming Hive*, 1st ed. Sebastopol, CA: O'Reilly & Associates, 2012.
[37] M. I. Baig, L. Shuib, e E. Yadegaridehkordi, «Big Data Tools: Advantages and Disadvantages», p. 7, 2019.
[38] «LanguageManual ORC - Apache Hive - Apache Software Foundation». https://cwiki.apache.org/confluence/display/hive/languagemanual+orc#LanguageManualORC-HiveQLSyntax (acedido 22 de junho de 2021).

[39] V. Aluko e S. Sakr, «Big SQL systems: an experimental evaluation», *Cluster Comput*, vol. 22, n.º 4, pp. 1347–1377, dez. 2019, doi: 10.1007/s10586-019-02914-4.
[40] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, e C. Curino, «Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications», em *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, Melbourne Victoria Australia: ACM, mai. 2015, pp. 1357–1369. doi: 10.1145/2723372.2742790.
[41] S. Pal, *SQL on Big Data*. Berkeley, CA: Apress, 2016. doi: 10.1007/978-1-4842-2247-8.
[42] M. Fuller, M. Moser, e M. Traverso, «Presto: The Definitive Guide», p. 311.
[43] R. Sethi *et al.*, «Presto: SQL on Everything», em *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, Macao, Macao: IEEE, abr. 2019, pp. 1802–1813. doi: 10.1109/ICDE.2019.00196.
[44] «Presto | Overview». https://prestodb.io/overview.html (acedido 7 de fevereiro de 2022).
[45] «Apache Spark™ - Unified Engine for large-scale data analytics». https://spark.apache.org/ (acedido 18 de janeiro de 2022).
[46] S. Chellappan e D. Ganesan, *Practical Apache Spark: Using the Scala API*. Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3652-9.
[47] V. Ankam, *Big data analytics: a handy reference guide for data analysts and data scientists to help obtain value from big data analytics using Spark on Hadoop clusters*. Birmingham Mumbai: Packt, 2016.
[48] «Cluster Mode Overview - Spark 3.2.1 Documentation». https://spark.apache.org/docs/latest/cluster-overview.html (acedido 10 de fevereiro de 2022).
[49] «Web UI - Spark 3.0.0 Documentation». https://spark.apache.org/docs/3.0.0/web-ui.html (acedido 28 de julho de 2022).
[50] S. Salloum, R. Dautov, X. Chen, P. X. Peng, e J. Z. Huang, «Big data analytics on Apache Spark», *Int J Data Sci Anal*, vol. 1, n.º 3–4, pp. 145–164, nov. 2016, doi: 10.1007/s41060-016-0027-9.
[51] M. Armbrust *et al.*, «Spark SQL: Relational Data Processing in Spark», em *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, Melbourne Victoria Australia: ACM, mai. 2015, pp. 1383–1394. doi: 10.1145/2723372.2742797.
[52] P. L, «Adaptive Query Execution in Spark 3.0», *Dive Into DataScience (DIDS)*, 10 de julho de 2020. https://medium.com/self-training-data-science-enthusiast/whats-new-in-spark-3-0-4a37287da90f (acedido 8 de fevereiro de 2022).
[53] «Performance Tuning - Spark 3.2.1 Documentation». https://spark.apache.org/docs/latest/sql-performance-tuning.html (acedido 8 de fevereiro de 2022).
[54] A. Sebaa, F. Chikh, A. Nouicer, e A. Tari, «Medical Big Data Warehouse: Architecture and System Design, a Case Study: Improving Healthcare Resources Distribution», *J Med Syst*, vol. 42, n.º 4, p. 59, abr. 2018, doi: 10.1007/s10916-018-0894-9.
[55] C.-H. Chang, F.-C. Jiang, C.-T. Yang, e S.-C. Chou, «On construction of a big data warehouse accessing platform for campus power usages», *Journal of Parallel and Distributed Computing*, vol. 133, pp. 40–50, nov. 2019, doi: 10.1016/j.jpdc.2019.05.011.
[56] «Sqoop -». https://sqoop.apache.org/ (acedido 22 de fevereiro de 2022).

[57] V. M. Ngo, N.-A. Le-Khac, e M.-T. Kechadi, «Designing and Implementing Data Warehouse for Agricultural Big Data», em *Big Data – BigData 2019*, K. Chen, S. Seshadri, e L.-J. Zhang, Eds., em Lecture Notes in Computer Science, vol. 11514. Cham: Springer International Publishing, 2019, pp. 1–17. doi: 10.1007/978-3-030-23551-2_1.

[58] «MongoDB: The Application Data Platform», *MongoDB*. https://www.mongodb.com (acedido 22 de fevereiro de 2022).

[59] «Apache Cassandra | Apache Cassandra Documentation». https://cassandra.apache.org/_/cassandra-basics.html (acedido 22 de fevereiro de 2022).

[60] A. A. C. Vieira, L. Pedro, M. Y. Santos, J. M. Fernandes, e L. S. Dias, «Data Requirements Elicitation in Big Data Warehousing», em *Information Systems*, M. Themistocleous e P. Rupino da Cunha, Eds., em Lecture Notes in Business Information Processing, vol. 341. Cham: Springer International Publishing, 2019, pp. 106–113. doi: 10.1007/978-3-030-11395-7_10.

[61] B. Martinho e M. Y. Santos, «An Architecture for Data Warehousing in Big Data Environments», em *Research and Practical Issues of Enterprise Information Systems*, A. M. Tjoa, L. D. Xu, M. Raffai, e N. M. Novak, Eds., em Lecture Notes in Business Information Processing, vol. 268. Cham: Springer International Publishing, 2016, pp. 237–250. doi: 10.1007/978-3-319-49944-4_18.

[62] M. Y. Santos *et al.*, «Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware», em *Proceedings of the 21st International Database Engineering & Applications Symposium on - IDEAS 2017*, Bristol, United Kingdom: ACM Press, 2017, pp. 242–252. doi: 10.1145/3105831.3105842.

[63] M. Hausenblas e J. Nadeau, «Apache Drill: Interactive Ad-Hoc Analysis at Scale», *Big Data*, vol. 1, n.º 2, pp. 100–104, jun. 2013, doi: 10.1089/big.2013.0011.

[64] «TPC-H Homepage». http://www.tpc.org/tpch/ (acedido 22 de fevereiro de 2022).

[65] N. Ahmed, S. Ahamed, J. I. Rafiq, e S. Rahim, «Data processing in Hive vs. SQL server: A comparative analysis in the query performance», em *2017 IEEE 3rd International Conference on Engineering Technologies and Social Sciences (ICETSS)*, Bangkok: IEEE, ago. 2017, pp. 1–5. doi: 10.1109/ICETSS.2017.8324202.

[66] R. Almeida, P. Furtado, e J. Bernardino, «Performance Evaluation MySQL InnoDB and Microsoft SQL Server 2012 for Decision Support Environments», em *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering - C3S2E '15*, Yokohama, Japan: ACM Press, 2008, pp. 56–62. doi: 10.1145/2790798.2790808.

[67] H. Faroqi, M. Mesbah, e J. Kim, «Applications of transit smart cards beyond a fare collection tool: a literature review», p. 17, 2018.

[68] S. Robinson, B. Narayanan, N. Toh, e F. Pereira, «Methods for pre-processing smartcard data to improve data quality», *Transportation Research Part C: Emerging Technologies*, vol. 49, pp. 43–58, dez. 2014, doi: 10.1016/j.trc.2014.10.006.

[69] S. G. Lee e M. Hickman, «Trip purpose inference using automated fare collection data», *Public Transp*, vol. 6, n.º 1–2, pp. 1–20, abr. 2014, doi: 10.1007/s12469-013-0077-5.

[70] A. Ali, J. Kim, e S. Lee, «Travel behavior analysis using smart card data», *KSCE J Civ Eng*, vol. 20, n.º 4, pp. 1532–1539, mai. 2016, doi: 10.1007/s12205-015-1694-0.

[71] L. Wu, J. E. Kang, Y. Chung, e A. Nikolaev, «Monitoring Multimodal Travel Environment Using Automated Fare Collection Data: Data Processing and Reliability

Analysis», *J. Big Data Anal. Transp.*, vol. 1, n.º 2–3, pp. 123–146, dez. 2019, doi: 10.1007/s42421-019-00012-w.
[72] W. Jang, «Travel Time and Transfer Analysis Using Transit Smart Card Data», *Transportation Research Record*, vol. 2144, n.º 1, pp. 142–149, jan. 2010, doi: 10.3141/2144-16.
[73] Y. Song, Y. Fan, X. Li, e Y. Ji, «Multidimensional visualization of transit smartcard data using space–time plots and data cubes», *Transportation*, vol. 45, n.º 2, pp. 311–333, mar. 2018, doi: 10.1007/s11116-017-9790-2.
[74] T. Li, D. Sun, P. Jing, e K. Yang, «Smart Card Data Mining of Public Transport Destination: A Literature Review», *Information*, vol. 9, n.º 1, p. 18, jan. 2018, doi: 10.3390/info9010018.
[75] J. Hora, T. G. Dias, A. Camanho, e T. Sobral, «Estimation of Origin-Destination matrices under Automatic Fare Collection: the case study of Porto transportation system», *Transportation Research Procedia*, vol. 27, pp. 664–671, 2017, doi: 10.1016/j.trpro.2017.12.103.
[76] A. A. Nunes, T. Galvao Dias, e J. Falcao e Cunha, «Passenger Journey Destination Estimation From Automated Fare Collection System Data Using Spatial Validation», *IEEE Trans. Intell. Transport. Syst.*, vol. 17, n.º 1, pp. 133–142, jan. 2016, doi: 10.1109/TITS.2015.2464335.
[77] «Horarios do Funchal - Quem Somos». http://www.horariosdofunchal.pt/index.php?option=com_content&task=view&id=63&Itemid=211 (acedido 22 de agosto de 2022).
[78] «Horarios do Funchal - Quem Somos». http://www.horariosdofunchal.pt/index.php?option=com_content&task=view&id=91&Itemid=318 (acedido 22 de agosto de 2022).
[79] «Horarios do Funchal - Quem Somos». http://www.horariosdofunchal.pt/index.php?option=com_content&task=view&id=37&Itemid=54 (acedido 22 de agosto de 2022).
[80] «Horarios do Funchal - Quem Somos». http://www.horariosdofunchal.pt/index.php?option=com_content&task=view&id=96&Itemid=323 (acedido 25 de junho de 2021).
[81] «Sobre o Google Transit - Ajuda do Transit Partners». https://support.google.com/transitpartners/answer/1111471 (acedido 29 de junho de 2021).
[82] «Referência | Google Transit estático | Google Developers». https://developers.google.com/transit/gtfs/reference (acedido 3 de março de 2022).
[83] A. M. Potdar, N. D G, S. Kengond, e M. M. Mulla, «Performance Evaluation of Docker Container and Virtual Machine», *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020, doi: 10.1016/j.procs.2020.04.152.
[84] «Big Data Europe», *GitHub*. https://github.com/big-data-europe (acedido 26 de junho de 2021).
[85] Y. S, «docker-presto». 4 de agosto de 2021. Acedido: 21 de março de 2022. [Em linha]. Disponível em: https://github.com/smizy/docker-presto
[86] F. Devillaine, M. Munizaga, e M. Trépanier, «Detection of Activities of Public Transport Users by Analyzing Smart Card Data», *Transportation Research Record*, vol. 2276, n.º 1, pp. 48–55, jan. 2012, doi: 10.3141/2276-06.
[87] «Apache Hadoop 3.3.3 – Hadoop Cluster Setup». https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html (acedido 7 de julho de 2022).

[88] «Hadoop Docker». Big Data Europe, 24 de agosto de 2022. Acedido: 24 de agosto de 2022. [Em linha]. Disponível em: https://github.com/big-data-europe/docker-hadoop
[89] «docker-hive». Big Data Europe, 22 de agosto de 2022. Acedido: 24 de agosto de 2022. [Em linha]. Disponível em: https://github.com/big-data-europe/docker-hive
[90] «Apache Tez – Install and Deployment Instructions». https://tez.apache.org/install.html (acedido 24 de agosto de 2022).
[91] «4. Enable Tez for Hive Queries - Hortonworks Data Platform». https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.1.3/bk_installing_manually_book/content/rpm-chap-tez-enable_tez_for_hive_queries.html (acedido 24 de agosto de 2022).
[92] «Spark Standalone Mode - Spark 3.3.0 Documentation». https://spark.apache.org/docs/latest/spark-standalone.html (acedido 11 de julho de 2022).
[93] «Distributed SQL Engine - Spark 3.3.0 Documentation». https://spark.apache.org/docs/latest/sql-distributed-sql-engine.html (acedido 24 de agosto de 2022).
[94] «Deploying Presto — Presto 0.273.3 Documentation». https://prestodb.io/docs/current/installation/deployment.html#config-properties (acedido 12 de julho de 2022).
[95] «Hive Connector — Presto 0.273.3 Documentation». https://prestodb.io/docs/current/connector/hive.html?highlight=hive#hive-configuration-properties (acedido 12 de julho de 2022).
[96] «Command Line Interface — Presto 0.273.3 Documentation». https://prestodb.io/docs/current/installation/cli.html (acedido 8 de agosto de 2022).
[97] «Using Spark SQL». https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.0.0/developing-spark-applications/content/using_spark_sql.html (acedido 3 de maio de 2022).
[98] chugugrace, «Sort Data for the Merge and Merge Join Transformations - SQL Server Integration Services (SSIS)». https://docs.microsoft.com/en-us/sql/integration-services/data-flow/transformations/sort-data-for-the-merge-and-merge-join-transformations (acedido 16 de maio de 2022).
[99] C. Costa e M. Y. Santos, «Evaluating Several Design Patterns and Trends in Big Data Warehousing Systems», em *Advanced Information Systems Engineering*, J. Krogstie e H. A. Reijers, Eds., em Lecture Notes in Computer Science, vol. 10816. Cham: Springer International Publishing, 2018, pp. 459–473. doi: 10.1007/978-3-319-91563-0_28.
[100] «https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml». https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml (acedido 7 de julho de 2022).
[101] «11. Determine YARN and MapReduce Memory Configuration Settings - Hortonworks Data Platform». https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html (acedido 7 de julho de 2022).
[102] «https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-common/yarn-default.xml». https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-common/yarn-default.xml (acedido 7 de julho de 2022).

8 Anexos

8.1 Ficheiro *docker-compose.yml*

8.1.1 Containers do *Hadoop*

8.1.1.1 HDFS

```
namenode:
   image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
   container_name: namenode
   hostname: 'namenode'
   restart: always
   ports:
     - 9870:9870
     - 9000:9000
   volumes:
     - Q:\Hadoop\namenode:/hadoop/dfs/name
     - Q:\Hadoop\ficheiros:/opt/ficheiros
     - Q:\docker-hadoop-spark\base\entrypoint.sh:/entrypoint.sh
     - Q:\docker-hadoop-spark\hosts:/etc/hosts
   environment:
     - CLUSTER_NAME=test
     - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
   env_file:
     - ./hadoop.env
   networks:
     Hadoop:
       ipv4_address: 10.5.0.3

 datanode1:
   image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
   container_name: datanode1
   hostname: 'datanode1'
   restart: always
   volumes:
     - Q:\Hadoop\datanode1:/hadoop/dfs/data
     - Q:\docker-hadoop-spark\hosts:/etc/hosts
   environment:
     SERVICE_PRECONDITION: "namenode:9870"
     CORE_CONF_fs_defaultFS: hdfs://namenode:9000
   ports:
     - "9864:9864"
   networks:
     Hadoop:
       ipv4_address: 10.5.0.4

 datanode2:
   image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
   container_name: datanode2
```

```
    hostname: 'datanode2'
    restart: always
    volumes:
      - Q:\Hadoop\datanode2:/hadoop/dfs/data
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
    environment:
      SERVICE_PRECONDITION: "namenode:9870"
      CORE_CONF_fs_defaultFS: hdfs://namenode:9000
    ports:
      - "9865:9864"
    env_file:
      - ./hadoop.env
    networks:
      Hadoop:
        ipv4_address: 10.5.0.5

  datanode3:
    image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
    container_name: datanode3
    hostname: 'datanode3'
    restart: always
    volumes:
      - Q:\Hadoop\datanode3:/hadoop/dfs/data
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
    environment:
      SERVICE_PRECONDITION: "namenode:9870"
      CORE_CONF_fs_defaultFS: hdfs://namenode:9000
    ports:
      - "9866:9864"
    env_file:
      - ./hadoop.env
    networks:
      Hadoop:
        ipv4_address: 10.5.0.6
```

8.1.1.2 YARN

```
resourcemanager:
    image: resourcemanager
    container_name: resourcemanager
    hostname: 'resourcemanager'
    restart: always
    volumes:
      - Q:\docker-hadoop-spark\apache-tez-0.9.2-bin:/opt/tez
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
      - Q:\docker-hadoop-spark\spark:/opt/spark
      - Q:\docker-hadoop-spark\sqoop-1.4.4.bin__hadoop-2.0.4-
alpha:/opt/sqoop
      - Q:\docker-hadoop-spark\pig-0.17.0:/opt/pig
```

```
      - Q:\docker-hadoop-spark\tez-site.xml:/opt/hadoop-
3.2.1/etc/hadoop/tez-site.xml
      - Q:\docker-hadoop-spark\hadoop-env2.sh:/opt/hadoop-
3.2.1/etc/hadoop/hadoop-env.sh
    environment:
      SERVICE_PRECONDITION: "namenode:9000 namenode:9870 datanode1:9864
datanode2:9864"
    env_file:
      - ./hadoop.env
    ports:
      - "8088:8088"
      - "10010:10000"
      - "4044:4040"
    networks:
      Hadoop:
        ipv4_address: 10.5.0.7

  nodemanager1:
    image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
    container_name: nodemanager1
    hostname: 'nodemanager1'
    restart: always
    volumes:
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
    environment:
      SERVICE_PRECONDITION: "namenode:9000 namenode:9870 datanode1:9864
resourcemanager:8088"
    env_file:
      - ./hadoop.env
    ports:
      - "8042:8042"
    networks:
      Hadoop:
        ipv4_address: 10.5.0.8

  nodemanager2:
    image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
    container_name: nodemanager2
    hostname: 'nodemanager2'
    restart: always
    volumes:
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
    environment:
      SERVICE_PRECONDITION: "namenode:9000 namenode:9870 datanode2:9864
resourcemanager:8088"
    env_file:
      - ./hadoop.env
    ports:
      - "8043:8042"
```

```
    networks:
      Hadoop:
        ipv4_address: 10.5.0.9

  nodemanager3:
    image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
    container_name: nodemanager3
    hostname: 'nodemanager3'
    restart: always
    volumes:
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
    environment:
      SERVICE_PRECONDITION: "namenode:9000 namenode:9870 datanode3:9864
resourcemanager:8088"
    env_file:
      - ./hadoop.env
    ports:
      - "8044:8042"
    networks:
      Hadoop:
        ipv4_address: 10.5.0.10
```

8.1.2 Spark

```
  spark-master:
    image: sparkmaster
    container_name: spark-master
    hostname: 'spark-master'
    depends_on:
      - namenode
      - datanode1
      - datanode2
    ports:
      - "8083:8080"
      - "7077:7077"
      - 6066:6066
      - 4040:4040
      - 10001:10000
    environment:
      - INIT_DAEMON_STEP=setup_spark
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
      - PYSPARK_PYTHON=python3
    env_file:
      - ./hadoop-hive.env
    volumes:
    - Q:\docker-hadoop-spark\hosts:/etc/hosts
    - Q:\Hadoop\jsons:/jsons
    - Q:\docker-hadoop-spark\hive-site.xml:/spark/conf/hive-site.xml
    - Q:\docker-hadoop-spark\core-site.xml:/spark/conf/core-site.xml
    - Q:\docker-hadoop-spark\hdfs-site.xml:/spark/conf/hdfs-site.xml
```

```
    - Q:\docker-hadoop-spark\spark-cassandra-connector_2.12-
3.1.0.jar:/spark/jars/spark-cassandra-connector_2.12-3.1.0.jar
    - Q:\docker-hadoop-spark\spark-measure_2.12-
0.17.jar:/spark/jars/spark-measure_2.12-0.17.jar

    networks:
      Hadoop:
        ipv4_address: 10.5.0.12

  spark-worker-1:
    image: bde2020/spark-worker:3.1.2-hadoop3.2
    container_name: spark-worker-1
    hostname: 'spark-worker-1'
    volumes:
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
      - Q:\docker-hadoop-spark\hive-site.xml:/spark/conf/hive-site.xml
      - Q:\docker-hadoop-spark\core-site.xml:/spark/conf/core-site.xml
      - Q:\docker-hadoop-spark\hdfs-site.xml:/spark/conf/hdfs-site.xml
      - Q:\docker-hadoop-spark\spark-cassandra-connector_2.12-
3.1.0.jar:/spark/jars/spark-cassandra-connector_2.12-3.1.0.jar
      - Q:\docker-hadoop-spark\spark-measure_2.12-
0.17.jar:/spark/jars/spark-measure_2.12-0.17.jar
    depends_on:
      - spark-master
    ports:
      - "8081:8081"
    env_file:
      - ./hadoop-hive.env
    environment:
      - "SPARK_MODE=worker"
      - "SPARK_WORKER_MEMORY=3G"
      - "SPARK_WORKER_CORES=2"
      - "SPARK_MASTER=spark://spark-master:7077"
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
      -
HIVE_CORE_CONF_javax_jdo_option_ConnectionURL=jdbc:postgresql://hive-
metastore/metastore
    networks:
      Hadoop:
        ipv4_address: 10.5.0.13

  spark-worker-2:
    image: bde2020/spark-worker:3.1.2-hadoop3.2
    container_name: spark-worker-2
    hostname: 'spark-worker-2'
    volumes:
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
      - Q:\docker-hadoop-spark\hive-site.xml:/spark/conf/hive-site.xml
      - Q:\docker-hadoop-spark\core-site.xml:/spark/conf/core-site.xml
```

```
      - Q:\docker-hadoop-spark\hdfs-site.xml:/spark/conf/hdfs-site.xml
      - Q:\docker-hadoop-spark\spark-cassandra-connector_2.12-
3.1.0.jar:/spark/jars/spark-cassandra-connector-assembly_2.12-3.1.0.jar
      - Q:\docker-hadoop-spark\spark-measure_2.12-
0.17.jar:/spark/jars/spark-measure_2.12-0.17.jar
    depends_on:
      - spark-master
    ports:
      - "8082:8081"
    env_file:
      - ./hadoop-hive.env
    environment:
      - "SPARK_MODE=worker"
      - "SPARK_WORKER_MEMORY=3G"
      - "SPARK_WORKER_CORES=2"
      - "SPARK_MASTER=spark://spark-master:7077"
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
      -
HIVE_CORE_CONF_javax_jdo_option_ConnectionURL=jdbc:postgresql://hive-
metastore/metastore
    networks:
      Hadoop:
        ipv4_address: 10.5.0.14
  spark-worker-3:
    image: bde2020/spark-worker:3.1.2-hadoop3.2
    container_name: spark-worker-3
    hostname: 'spark-worker-3'
    volumes:
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
      - Q:\docker-hadoop-spark\hive-site.xml:/spark/conf/hive-site.xml
      - Q:\docker-hadoop-spark\core-site.xml:/spark/conf/core-site.xml
      - Q:\docker-hadoop-spark\hdfs-site.xml:/spark/conf/hdfs-site.xml
      - Q:\docker-hadoop-spark\spark-cassandra-connector_2.12-
3.1.0.jar:/spark/jars/spark-cassandra-connector_2.12-3.1.0.jar
      - Q:\docker-hadoop-spark\spark-measure_2.12-
0.17.jar:/spark/jars/spark-measure_2.12-0.17.jar
    depends_on:
      - spark-master
    ports:
      - "8084:8081"
    env_file:
      - ./hadoop-hive.env
    environment:
      - "SPARK_MODE=worker"
      - "SPARK_WORKER_MEMORY=3G"
      - "SPARK_WORKER_CORES=2"
      - "SPARK_MASTER=spark://spark-master:7077"
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
```

```
      -
HIVE_CORE_CONF_javax_jdo_option_ConnectionURL=jdbc:postgresql://hive-
metastore/metastore
    networks:
      Hadoop:
        ipv4_address: 10.5.0.28
```

8.1.3 Hive

```
  hive-server:
    image: bde2020/hive:2.3.2-postgresql-metastore
    container_name: hive-server
    hostname: 'hive-server'
    depends_on:
      - namenode
      - datanode1
      - datanode2
    env_file:
      - ./hadoop-hive.env
    volumes:
      - Q:\Hadoop\ficheiros:/opt/ficheiros
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
      - Q:\docker-hadoop-spark\apache-tez-0.9.2-bin:/opt/tez
      - Q:\docker-hadoop-spark\sqoop-1.4.4.bin__hadoop-2.0.4-
alpha:/opt/sqoop
      - Q:\docker-hadoop-spark\pig-0.17.0:/opt/pig
      - Q:\docker-hadoop-spark\hadoop-env.sh:/opt/hadoop-
2.7.4/etc/hadoop/hadoop-env.sh
      - Q:\docker-hadoop-spark\tez-site.xml:/opt/hadoop-
2.7.4/etc/hadoop/tez-site.xml
    environment:
      HIVE_CORE_CONF_javax_jdo_option_ConnectionURL:
"jdbc:postgresql://hive-metastore/metastore"
      SERVICE_PRECONDITION: "hive-metastore:9083"
    ports:
      - "10000:10000"
      - "10002:10002"
    networks:
      Hadoop:
        ipv4_address: 10.5.0.15

  hive-metastore:
    image: bde2020/hive:2.3.2-postgresql-metastore
    container_name: hive-metastore
    hostname: 'hive-metastore'
    volumes:
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
    env_file:
      - ./hadoop-hive.env
```

```
    command: /opt/hive/bin/hive --service metastore
    environment:
      SERVICE_PRECONDITION: "namenode:9870 datanode1:9864 datanode2:9864
hive-metastore-postgresql:5432"
    ports:
      - "9083:9083"
    networks:
      Hadoop:
        ipv4_address: 10.5.0.16

  hive-metastore-postgresql:
    image: bde2020/hive-metastore-postgresql:2.3.0
    container_name: hive-metastore-postgresql
    hostname: 'hive-metastore-postgresql'
    volumes:
      - postgredata:/var/lib/postgresql/data
      - Q:\docker-hadoop-spark\hosts:/etc/hosts
    restart: always
    networks:
      Hadoop:
        ipv4_address: 10.5.0.17
```

8.1.4 Presto

```
  coordinator-1:
    container_name: coordinator-1
    networks:
      Hadoop:
        ipv4_address: 10.5.0.51
    hostname: coordinator-1
    restart: always
    image: smizy/presto:0.222-alpine
    ports:
      - 8080:8080
    environment:
      - PRESTO_DISCOVERY_URI=http://coordinator-1:8080
      - PRESTO_JVM_MAX_HEAP=4608M
      - PRESTO_QUERY_MAX_MEMORY=9GB
      - PRESTO_QUERY_MAX_MEMORY_PER_NODE=2304MB
      - PRESTO_QUERY_MAX_TOTAL_MEMORY_PER_NODE=3456MB
      - PRESTO_MEMORY_HEAP_HEADROOM_PER_NODE=922MB
      - TERM=xterm
    volumes:
      - Q:\docker-hadoop-spark\hive.properties:/usr/local/presto-server-
0.222/etc/catalog/hive.properties
      - Q:\docker-hadoop-spark\cassandra.properties:/usr/local/presto-
server-0.222/etc/catalog/cassandra.properties
    command: coordinator

  worker-1:
```

```
    container_name: worker-1
    networks:
      Hadoop:
        ipv4_address: 10.5.0.52
    hostname: worker-1
    restart: always
    image: smizy/presto:0.222-alpine
    depends_on: ["coordinator-1"]
    environment:
      - PRESTO_DISCOVERY_URI=http://coordinator-1:8080
      - PRESTO_JVM_MAX_HEAP=4608M
      - PRESTO_QUERY_MAX_MEMORY=9GB
      - PRESTO_QUERY_MAX_MEMORY_PER_NODE=2304MB
      - PRESTO_QUERY_MAX_TOTAL_MEMORY_PER_NODE=3456MB
      - PRESTO_MEMORY_HEAP_HEADROOM_PER_NODE=922MB
      - TERM=xterm
    volumes:
      - Q:\docker-hadoop-spark\hive.properties:/usr/local/presto-server-
0.222/etc/catalog/hive.properties
      - Q:\docker-hadoop-spark\cassandra.properties:/usr/local/presto-
server-0.222/etc/catalog/cassandra.properties
    command: worker

  worker-2:
    container_name: worker-2
    networks:
      Hadoop:
        ipv4_address: 10.5.0.53
    hostname: worker-2
    restart: always
    image: smizy/presto:0.222-alpine
    depends_on: ["coordinator-1"]
    environment:
      - PRESTO_DISCOVERY_URI=http://coordinator-1:8080
      - PRESTO_JVM_MAX_HEAP=4608M
      - PRESTO_QUERY_MAX_MEMORY=9GB
      - PRESTO_QUERY_MAX_MEMORY_PER_NODE=2304MB
      - PRESTO_QUERY_MAX_TOTAL_MEMORY_PER_NODE=3456MB
      - PRESTO_MEMORY_HEAP_HEADROOM_PER_NODE=922MB
      - TERM=xterm
    volumes:
      - Q:\docker-hadoop-spark\hive.properties:/usr/local/presto-server-
0.222/etc/catalog/hive.properties
      - Q:\docker-hadoop-spark\cassandra.properties:/usr/local/presto-
server-0.222/etc/catalog/cassandra.properties
    command: worker

  worker-3:
    container_name: worker-3
```

```
    networks:
      Hadoop:
        ipv4_address: 10.5.0.54
    hostname: worker-3
    restart: always
    image: smizy/presto:0.222-alpine
    depends_on: ["coordinator-1"]
    environment:
      - PRESTO_DISCOVERY_URI=http://coordinator-1:8080
      - PRESTO_JVM_MAX_HEAP=4608M
      - PRESTO_QUERY_MAX_MEMORY=9GB
      - PRESTO_QUERY_MAX_MEMORY_PER_NODE=2304MB
      - PRESTO_QUERY_MAX_TOTAL_MEMORY_PER_NODE=3456MB
      - PRESTO_MEMORY_HEAP_HEADROOM_PER_NODE=922MB
      - TERM=xterm
    volumes:
      - Q:\docker-hadoop-spark\hive.properties:/usr/local/presto-server-
0.222/etc/catalog/hive.properties
      - Q:\docker-hadoop-spark\cassandra.properties:/usr/local/presto-
server-0.222/etc/catalog/cassandra.properties
    command: worker
```

8.1.5 SQL Server

```
  sqlserverdw:
    container_name: sqlserverdw
    image: mcr.microsoft.com/mssql/server:2019-latest
    hostname: sqlserverdw
    ports:
      - 1444:1433
    environment:
      - "ACCEPT_EULA=Y"
      - "SA_PASSWORD=Password123"
      - "MSSQL_PID=Enterprise"
    volumes:
      - Q:\SQL:/var/opt/mssql
    networks:
      Hadoop:
        ipv4_address: 10.5.0.25
```

8.1.6 Docker networks and volumes

```
networks:
    Hadoop:
      driver: bridge
      ipam:
```

```
      config:
        - subnet: 10.5.0.0/16
          gateway: 10.5.0.1

volumes:
  postgredata:
  pg_data:
```

8.2 Hadoop.env

```
CORE_CONF_fs_defaultFS=hdfs://namenode:9000
HDFS_CONF_dfs_webhdfs_enabled=true
YARN_CONF_yarn_scheduler_minimum___allocation___mb=512
MAPRED_CONF_mapreduce_map_java_opts=-Xmx205m

CORE_CONF_hadoop_http_staticuser_user=root
CORE_CONF_hadoop_proxyuser_hue_hosts=*
CORE_CONF_hadoop_proxyuser_hue_groups=*
CORE_CONF_io_compression_codecs=org.apache.hadoop.io.compress.SnappyCodec
CORE_CONF_hadoop_http_filter_initializers=org.apache.hadoop.security.Http
CrossOriginFilterInitializer
CORE_CONF_hadoop_http_cross___origin_enabled=true
CORE_CONF_hadoop_http_cross___origin_allowed___origins=*
CORE_CONF_hadoop_http_cross___origin_allowed___methods=GET,POST,HEAD,DELE
TE,OPTIONS
CORE_CONF_hadoop_http_cross___origin_allowed___headers=X-Requested-
With,Content-Type,Accept,Origin
CORE_CONF_hadoop_http_cross___origin_maxvage=1800
CORE_CONF_org_apache_hadoop_security_HttpCrossOriginFilterInitializer=had
oop.http.filter.initializers


HDFS_CONF_dfs_webhdfs_enabled=true
HDFS_CONF_dfs_replication=2
HDFS_CONF_dfs_permissions_enabled=false
HDFS_CONF_dfs_namenode_datanode_registration_ip___hostname___check=false
HDFS_CONF_dfs_datanode_socket_write_timeout=3000000
HDFS_CONF_dfs_socket_timeout=3000000

YARN_CONF_yarn_log___aggregation___enable=true
YARN_CONF_yarn_log_server_url=http://historyserver:8188/applicationhistor
y/logs/
YARN_CONF_yarn_resourcemanager_recovery_enabled=true
YARN_CONF_yarn_resourcemanager_store_class=org.apache.hadoop.yarn.server.
resourcemanager.recovery.FileSystemRMStateStore
YARN_CONF_yarn_resourcemanager_scheduler_class=org.apache.hadoop.yarn.YAR
N_CONF_yarn_scheduler_capacity_root_default_maximum___allocation___mbserv
er.resourcemanager.scheduler.capacity.CapacityScheduler
=3072
```

```
YARN_CONF_yarn_scheduler_minimum___allocation___mb=512
YARN_CONF_yarn_scheduler_maximum___allocation___mb=3072
YARN_CONF_yarn_scheduler_capacity_root_default_maximum___allocation___vco
res=4
YARN_CONF_yarn_resourcemanager_fs_state___store_uri=/rmstate
YARN_CONF_yarn_resourcemanager_system___metrics___publisher_enabled=true
YARN_CONF_yarn_resourcemanager_hostname=resourcemanager
YARN_CONF_yarn_resourcemanager_address=resourcemanager:8032
YARN_CONF_yarn_resourcemanager_scheduler_address=resourcemanager:8030
YARN_CONF_yarn_resourcemanager_resource__tracker_address=resourcemanager:
8031
YARN_CONF_yarn_timeline___service_enabled=true
YARN_CONF_yarn_timeline___service_generic___application___history_enabled
=true
YARN_CONF_yarn_timeline___service_hostname=historyserver
YARN_CONF_mapreduce_map_output_compress=true
YARN_CONF_mapred_map_output_compress_codec=org.apache.hadoop.io.compress.
SnappyCodec
YARN_CONF_yarn_nodemanager_resource_memory___mb=3072
YARN_CONF_yarn_nodemanager_resource_cpu___vcores=4
YARN_CONF_yarn_nodemanager_disk___health___checker_max___disk___utilizati
on___per___disk___percentage=98.5
YARN_CONF_yarn_nodemanager_remote___app___log___dir=/app-logs
YARN_CONF_yarn_nodemanager_aux___services=mapreduce_shuffle
YARN_CONF_yarn_webapp_ui2_enable=true

MAPRED_CONF_mapreduce_framework_name=yarn-tez
MAPRED_CONF_mapred_child_java_opts=-Xmx3072m
MAPRED_CONF_mapreduce_map_memory_mb=256
MAPRED_CONF_mapreduce_reduce_memory_mb=512
MAPRED_CONF_mapreduce_map_java_opts=-Xmx205m
MAPRED_CONF_mapreduce_reduce_java_opts=-Xmx410m
MAPRED_CONF_yarn_app_mapreduce_am_resource_mb=512
MAPRED_CONF_yarn_app_mapreduce_am_command___opts=-Xmx410m
MAPRED_CONF_yarn_app_mapreduce_am_env=HADOOP_MAPRED_HOME=/opt/hadoop-
3.2.1/
MAPRED_CONF_mapreduce_map_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1/
MAPRED_CONF_mapreduce_reduce_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1/
```

8.2.1 Configuração de recursos no *mapred-site.xml*

Variável	Valor pré-definido [100]	Valor inserido	Calculo [101]
Mapreduce.map.memory.mb	-1	256	RAM por container gerado pelo *Node Manager*
Mapreduce.reduce.memory.mb	-1	512	2 x (RAM por container gerado *Node Manager*)
Mapreduce.map.java.opts	--	-Xmx205m	0.8 x (RAM por container gerado *Node Manager*)
Mapreduce.reduce.java.opts	--	-Xmx410m	0.8 x 2 x (RAM por container gerado *Node Manager*)

8.2.2 Configuração de recursos no *yarn-site.xml*

Variável	Valor pré-definido [78]	Valor inserido	Calculo [101]
Yarn.scheduler.minimum-allocation-memory-mb	1024	512	RAM por container gerado pelo *Node Manager*
Yarn.scheduler.maximum-allocation-memory-mb	8192	3072	(n.º de containers gerados pelo *Node Manager*) x (RAM por container)
Yarn.NodeManager.resource.memory-mb	-1	3072	(n.º de containers gerados pelo *Node Manager*) x (RAM por container)
Yarn.NodeManager.resource.cpu-vcores	-1	4	----

8.3 *Hadoop-hive.env*

```
HIVE_SITE_CONF_javax_jdo_option_ConnectionURL=jdbc:postgresql://hive-metastore-postgresql/metastore
HIVE_SITE_CONF_javax_jdo_option_ConnectionDriverName=org.postgresql.Driver
HIVE_SITE_CONF_javax_jdo_option_ConnectionUserName=hive
HIVE_SITE_CONF_javax_jdo_option_ConnectionPassword=hive
HIVE_SITE_CONF_datanucleus_autoCreateSchema=false
HIVE_SITE_CONF_hive_metastore_uris=thrift://hive-metastore:9083
HIVE_SITE_CONF_hive_mapred_mode=nonstrict
HIVE_SITE_CONF_hive_strict_checks_cartesian_product=false
HIVE_SITE_CONF_hive_exec_parallel=true
HIVE_SITE_CONF_hive_vectorized_execution_enabled=true
HIVE_SITE_CONF_hive_execution_engine=tez
HIVE_SITE_CONF_hive_tez_container_size=512
HIVE_SITE_CONF_hive_tez_java_opts=-Xmx410m
HIVE_SITE_CONF_hive_tez_exec_print_summary=true
HIVE_SITE_CONF_tez_am_log_level=debug
HIVE_SITE_CONF_mapreduce_map_log_level=debug
```

```
HIVE_SITE_CONF_hive_tez_log_level=debug
HIVE_SITE_CONF_hive_server2_tez_initialize_default_sessions=true
HIVE_SITE_CONF_hive_users_in_admin_role=root
HIVE_SITE_CONF_hive_log_explain_output=true
HIVE_SITE_CONF_hive_llap_execution_mode=auto
```

8.4 Configuração do *Spark*

Variável:	Valor:	Definição:
SPARK_MASTER_HOST	*Spark-master*	IP ou *hostname* do mestre
SPARK_MASTER_PORT	7077	Porta do mestre
SPARK_MASTER_WEBUI_PORT	8080	Porta para o acesso à interface *Web*
SPARK_WORKER_CORES	2	Número de núcleos disponível para o nó trabalhador
SPARK_WORKER_MEMORY	3G (3 GB)	Quantidade de memória para o nó trabalhador

8.5 Configuração do *Presto*

Variáveis:	Valor:	Definição:
query.max-memory	9GB	Máximo de memoria que pode ser dedicado a uma *query*.
query.max-memory-per-node	2304MB	Máximo de memoria da *query* para cada nó pelo utilizador
query.max-total-memory-per-node	3456MB	Máximo de memória da *query* de cada nó pelo utilizador e pelo sistema.
Discovery.uri	coordinator-1	*URI* referente ao nó coordenador.

8.6 ETL pelo *PySpark*

8.6.1 *Script*

```
#bibliotecas necessarias

from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
from pyspark.sql import Row
from pyspark.sql import HiveContext
from pyspark.sql.functions import *
from pyspark.sql.window import Window
```

```
sc = SparkContext.getOrCreate()
#connectar as tabelas de hive
hive_context = HiveContext(sc)

#CONNECTAR AS tabelas de necessarias
tabela_val = hive_context.sql("select * from val_a")

#limpar valores vazios

tabela_val = tabela_val.na.drop(subset=["paragementradaord",
"paragfimord" ])
tabela_val = tabela_val.withColumn('carreira', trim(tabela_val.carreira))

tabela_paragens = hive_context.sql("select * from codigo_paragem_orc")


# obter paragens dos autocarros
cond = [tabela_val.carreira == tabela_paragens.numeroautocarro,
tabela_val.sentidoperc == tabela_paragens.direcao,
tabela_paragens.stopsequence == tabela_val.paragementradaord]
tabela_val = tabela_val.join(tabela_paragens, cond, 'inner')
tabela_val = tabela_val.na.drop(subset=['code'])
tabela_val = tabela_val.select(tabela_val.datahora, tabela_val.veiculo,
tabela_val.carreira, \
    tabela_val.sentidoperc, tabela_val.nscartaohi, tabela_val.nscartaolo,
\
    tabela_val.code, tabela_val.titulo)
tabela_val = tabela_val.withColumnRenamed('code', 'codigoparagementrada')
tabela_val = tabela_val.dropDuplicates(['nscartaolo', 'datahora']) #Faz
remover os duplicados


#tabelas de dimensao
tabela_ticket = hive_context.sql('select * from dimticket_orc')
tabela_stops = hive_context.sql('select * from dimstop_orc')
tabela_routes = hive_context.sql('select * from dimroute_orc')
tabela_tempo = hive_context.sql('select * from dimtime_orc')
tabela_data = hive_context.sql('select * from dimdate_orc')

#junta-las
tabela_joined = tabela_val.join(tabela_ticket, tabela_val.titulo ==
tabela_ticket.codigo, "inner").\
    join(tabela_routes, tabela_val.carreira ==
tabela_routes.route_short_name, "inner").\
    join(tabela_stops, tabela_val.codigoparagementrada ==
tabela_stops.stop_code, "inner").\
    join(tabela_data, to_date(tabela_val.datahora) == tabela_data.date,
"inner").\
```

```
    join(tabela_tempo, date_format(tabela_val.datahora, 'HH:mm:ss') ==
tabela_tempo.timealtkey, "inner")
tabela_val.unpersist()
tabela_ticket.unpersist()
tabela_stops.unpersist()
tabela_routes.unpersist()
tabela_tempo.unpersist()
tabela_data.unpersist()
tabela_paragens.unpersist()
tabela_joined=tabela_joined.select(tabela_joined.datahora,
tabela_joined.datekey, tabela_joined.timekey, tabela_joined.veiculo,
tabela_joined.routekey, tabela_joined.sentidoperc,
tabela_joined.nscartaohi, tabela_joined.nscartaolo,
tabela_joined.stopkey, tabela_joined.ticketkey)

window = Window.partitionBy("nscartaolo").orderBy("nscartaolo",
"datahora")
tabela_joined = tabela_joined.withColumn("diffprev",
lag(tabela_joined.datahora).over(window))


tabela_joined = tabela_joined.withColumn('diffmin',
(unix_timestamp(tabela_joined.datahora.cast("timestamp")) -
unix_timestamp(tabela_joined.diffprev.cast("timestamp")))/60)
tabela_joined = tabela_joined.withColumn('diffhour',
(unix_timestamp(tabela_joined.datahora.cast("timestamp")) -
unix_timestamp(tabela_joined.diffprev.cast("timestamp")))/3600)
#filtrar se tem menos de 2 min para remover

tabela_joined = tabela_joined.withColumn('transhipment',
when(((tabela_joined.diffmin < 45)  & (tabela_joined.diffmin > 2) &
(tabela_joined.nscartaohi != 0)), 1).otherwise(0))
tabela_joined = tabela_joined.withColumn('transhipment_routekey',
lag(tabela_joined.routekey).over(window))
tabela_joined = tabela_joined.withColumn('transhipment_routekey_1', when(
tabela_joined.transhipment == 1, tabela_joined.transhipment_routekey))
tabela_joined = tabela_joined.withColumn('transhipment_stopkey',
lag(tabela_joined.stopkey).over(window))
tabela_joined = tabela_joined.withColumn('transhipment_stopkey_1', when(
tabela_joined.transhipment == 1, tabela_joined.transhipment_stopkey))



# tabela final antes de enviar
tabela_final=tabela_joined.select( \
    tabela_joined.datekey, tabela_joined.timekey, tabela_joined.veiculo,
\
    tabela_joined.routekey, tabela_joined.sentidoperc,
tabela_joined.nscartaohi, \
```

```
    tabela_joined.nscartaolo, tabela_joined.stopkey,
tabela_joined.ticketkey, tabela_joined.diffmin, tabela_joined.diffhour, \
    tabela_joined.transhipment, \
    tabela_joined.transhipment_routekey_1,
tabela_joined.transhipment_stopkey_1\
    )

tabela_final = tabela_final.withColumnRenamed('transhipment_routekey_1',
'transhipment_routekey')
tabela_final = tabela_final.withColumnRenamed('transhipment_stopkey_1',
'transhipment_stopkey')
tabela_joined.unpersist()

tabela_final.write.format("orc").mode("overwrite").saveAsTable("default.f
actval_a")
tabela_final.unpersist()
```

8.6.2 Comando de execução

/spark/bin/spark-submit --master spark://spark-master:7077 --executor-cores 2 --executor-memory 3G --conf spark.sql.adaptive.enabled=true --conf spark.sql.adaptive.coalescePartitions.enabled=true --conf spark.sql.adaptive.skewJoin.enabled=true pyspark_etl.py

8.7 Tabelas de resultados do desempenho *ETL*

8.7.1 SQL Server e *SSIS*

SSIS	Nº linhas Total	Tempo (min:sec)
A	18190219	00:05:02
B	35912880	00:10:31
C	53293745	00:16:07
D	70319028	00:21:11
E	85381030	00:25:44

8.7.2 Spark e *PySpark*

Spark	Nº linhas Total	Tempo (min:sec)
A	18190219	00:00:51
B	35912880	00:01:25
C	53293745	00:01:54
D	70319028	00:02:36
E	85381030	00:03:00

8.8 Tabela de resultados do desempenho nas *queries* de negócio

8.8.1 Hive

Hive				
A	B	C	D	E

Q1	00:20,0	00:45,3	00:55,3	01:04,3	01:28,0
Q2	00:17,9	00:19,9	00:28,1	00:28,9	00:40,0
Q3	01:13,0	01:46,0	02:16,0	02:41,4	03:23,0
Q4	00:13,9	00:25,8	00:42,0	00:46,4	00:52,9

8.8.2 SQL Server

	SQL Server				
	A	B	C	D	E
Q1	00:09,1	00:18,3	00:29,5	00:39,2	00:46,0
Q2	00:08,9	00:27,0	00:41,0	00:54,0	01:04,0
Q3	00:09,4	00:18,0	00:29,3	00:55,0	01:05,0
Q4	00:08,0	00:18,0	00:28,0	00:38,0	00:47,0

8.8.3 Presto

	Presto				
	A	B	C	D	E
Q1	00:11,5	00:10,0	00:13,5	00:16,4	00:18,3
Q2	00:03,4	00:05,5	00:06,0	00:08,1	00:09,3
Q3	00:06,1	00:16,7	00:41,3	00:31,6	00:50,8
Q4	00:03,8	00:04,5	00:06,4	00:08,0	00:10,1

8.8.4 *Spark SQL*

	Spark SQL				
	A	B	C	D	E
Q1	00:02,7	00:03,1	00:04,5	00:05,4	00:06,1
Q2	00:01,6	00:03,0	00:05,0	00:06,8	00:04,1
Q3	00:23,4	00:21,2	00:20,1	00:28,8	00:31,5
Q4	00:01,2	00:01,8	00:03,0	00:03,6	00:02,9