Universidade do Minho
Escola de Engenharia

José Manuel da Silva Correia

Processamento Analítico de Dados em Contextos de Big Data com o Druid

Julho de 2018

Processamento Analítico de Dados em Contextos de Big Data com o Druid
José Manuel da Silva Correia
UMinho | 2018

Universidade do Minho
Escola de Engenharia

José Manuel da Silva Correia

Processamento Analítico de Dados em Contextos de *Big Data* com o Druid

Dissertação de Mestrado

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação da

Professora Doutora Maribel Yasmina Campos Alves Santos

Julho de 2018

DECLARAÇÃO

Nome: José Manuel da Silva Correia

Endereço eletrónico: a71863@alunos.uminho.pt

Telefone: 919844483

Número do Bilhete de Identidade: 14846001

Título da dissertação: Processamento Analítico de Dados em Contextos de *Big Data* com o Druid

Orientador: Professora Doutora Maribel Yasmina Santos

Ano de conclusão: 2018

Designação do Mestrado: Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Universidade do Minho, 26/07/2018

Assinatura: 

“It always seems impossible

until it’s done”

Nelson Mandela

AGRADECIMENTOS

A travessia que agora finda foi longa e curta ao mesmo tempo, muito dura, mas também muito proveitosa. Foi uma caminhada que me proporcionou vivenciar das melhores coisas que vivenciei na vida e, infelizmente, foi uma caminhada que também me obrigou a sentir aquilo que de pior há para se sentir, ao impor-me a despedida de uma das pessoas que me eram mais queridas e que mais de perto acompanhavam o meu percurso. Começo, portanto, por dedicar este trabalho aos meus avós paternos, ambos partidos, um dos quais durante esta minha caminhada, e que presentes fisicamente ou de uma outra forma qualquer que exista, sempre me guiaram.

Aos meus pais, por serem os meus ídolos desde sempre e para sempre, embora a sua humildade, por vezes excessiva, tenda a tentar afastar este papel. Às minhas irmãs pelo importante amparo, por todas as palavras, por todos os momentos e pelo prazer que foi e é ver uma crescer e desbravar caminho à minha frente, servindo-me de aprendizagem, e ver outra seguir as minhas pisadas e muito me orgulhar. Aproveito também para agradecer à restante família, em especial ao meu pequeno sobrinho por nunca me deixar trabalhar ao fim de semana!

À Coletividade de Indivíduos Individuais por todas as conversas profundas e por, em conjunto, sermos sempre capazes de encontrar as melhores saídas e de resolver os mais complicados enigmas, mesmo que diante de nós estejam as salas mais escuras e bem trancadas que alguém alguma vez viu. Na categoria de amigos, agradeço ainda ao Diogo, por ser dos poucos (se não o único) grande amigo que me acompanha quase desde que me conheço e que compartilhou comigo todos os momentos.

Um obrigado muito especial à professora Maribel que foi o motivo pelo qual abracei este desafio, em detrimento de outros. Obrigado por ser sempre uma incansável ouvinte e contribuidora para este trabalho e que sempre soube as palavras a aplicar em cada momento. Agradeço também a todos os meus colegas do LID4, essencialmente pelo enorme espírito de entreajuda e boa disposição que sempre demonstraram. Ao Carlos dedico também um grande obrigado por todas as discussões que partilhamos.

E se no início referi que esta longa caminhada me proporcionou das melhores coisas que vivi é, principalmente, por causa da minha companheira de todas as batalhas. Aquela pela qual sinto um amor inabalável e que é capaz de tornar todos os momentos de tristeza numa grande felicidade. Susana, um obrigado do tamanho do mundo meu amor!

Todos estas pessoas, direta ou indiretamente foram importantíssimas e me ajudaram e motivaram, fazendo de mim sempre capaz de desconstruir a frase citada na página anterior e de a referir vezes sem conta, até que a segunda parte da frase possa ser proferida: *"It's done"*!

RESUMO

Ao longo dos últimos anos tem-se assistido a um crescimento enorme na utilização de dispositivos como por exemplo *smartphones*, *tablets* e sensores. Esta tendência, aliada ao facto de se guardarem praticamente todas as interações com as aplicações e serviços disponíveis no mundo, origina a geração de vastas quantidades de dados diariamente, a grande velocidade e provenientes de diversas fontes. Simultaneamente, a pressão causada pelo ambiente intensamente competitivo, no qual as organizações estão inseridas também tem vindo a crescer, obrigando-as a fazer cada vez mais e melhor com cada vez menos recursos à sua disposição. Posto isto, urge que as organizações tirem o melhor partido possível das tecnologias à disposição, a fim de melhorar a sua produtividade, eficiência e tomada de decisão. Assim, importa-lhes perceber os contextos para os quais a utilização singular das tecnologias tradicionais já não é adequada, sendo necessário alicerçar os seus processos em tecnologias *Big Data* ou na combinação destas com as tecnologias tradicionais. O tradicional *Data Warehouse* (DW) é um exemplo da inadequação das tecnologias tradicionais para lidar com características como o volume, velocidade e variedade comummente associadas ao *Big Data*, o que obriga as organizações a optar por outras estratégias para ter uma visão central da sua informação. Essas estratégias passam por dotar os DW com as tecnologias *Big Data*, originando-se um *Big Data Warehouse* (BDW), que tem objetivos semelhantes ao DW e que é capaz de suprimir as suas limitações. Além disto, as organizações necessitam não só de analisar dados históricos, mas também dados em *Real-Time*, integrados num único repositório denominado por *Real-Time Big Data Warehouse* (RTBDW). Uma decisão em tempo útil, em contexto militar, por exemplo, pode salvar a vida de milhares de pessoas. Nesta dissertação, enquadrada num projeto de investigação em colaboração entre a Bosch e a Universidade do Minho, explora-se e avalia-se o Druid no processamento analítico de vastas quantidades de dados, aplicando-se diferentes estratégias de modelação, *segment granularity, query granularity* e *hashed partitions*, a fim de perceber se estas propriedades influenciam o desempenho e o espaço necessário de armazenamento. Para efetuar os diferentes testes de desempenho é utilizado o *Star Schema Benchmark*. Os resultados demonstram que a aplicação destas propriedades pode otimizar o desempenho, sendo que os cenários em que se utilizam partições, normalmente, alcançam menores tempos no processamento das *queries* avaliadas. É ainda proposta uma arquitetura de RTBDW funcional, em torno do Druid, validada recorrendo a um caso de demonstração.

Palavras-Chave: *Big Data*, *Big Data Warehouse*, Druid, OLAP, *Real-Time*

ABSTRACT

Over the last few years we have witnessed to a tremendous increase in the use of devices, such as smartphones, tablets and sensors. This trend coupled with the fact that we save most of the interactions with applications or services available in the world, leads to the generation of vast amounts of data on a daily basis, at a high velocity and from different sources. Simultaneously, the pressure caused by the intensively competitive environment, in which organization are operating has also being growing, forcing them to do more and better with fewer resources at their disposal. Thus, it's imperative that organizations make the best use of available technologies, in order to improve their productivity, efficiency and decision making. Therefore, they need to be aware of the contexts, in which the use of traditional technology is no longer appropriate and when it is necessary to support their processes in Big Data technologies or in a combination of these with traditional technologies. Traditional Data Warehouse (DW) is an example of the inadequacy of traditional technologies to handle characteristics such as volume, velocity and variety, commonly associated to Big Data, forcing organizations to choose other strategies to have a central view of their information. These strategies consist of equipping the DW with Big Data technologies, resulting in a Big Data Warehouse (BDW), which has similar goals to DW and that can suppress its limitations. In addition, organizations need to analyze not only historical data, but also real-time data, integrated into a single repository named Real-Time Big Data Warehouse (RTBDW). A timely decision, in a military context, can save thousands of people lives. In this master's thesis, which emerges in the context of a collaborative research project between Bosch and University of Minho, explores and evaluates the analytical processing of vast amounts of data in Druid, applying different strategies of data modelling, segment granularity, query granularity and hashed partitions, to verify the impact that these properties have in terms of performance and space required to store the data. To perform the several performance tests, the well-known Star Schema Benchmark is used. The obtained results show that the application of these properties can optimize the performance, and the scenarios in which partitions are used, usually achieve better processing times in the evaluated queries. It is also proposed a functional RTBDW architecture, around Druid, validated using a demonstration case.

Keywords: Big Data, Big Data Warehouse, Druid, OLAP, Real-Time

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SIGLAS E ACRÓNIMOS

Neste documento é utilizado um conjunto de siglas e acrónimos, listadas de seguida:

ACID - *Atomicity, Consistency, Isolation, Durability*

API - *Application programming interface*

BASE - *Basic Availability, Soft state, Eventual consistency*

BDW – *Big Data warehouse*

BI – *Business Intelligence*

BIPdiwa - *Business Intelligence Platform for Data Integration, Warehousing and Analysis*

CAP - *Consistency, Availability, Partition tolerance*

DSRM - *Design Science Research Methodology*

DW – *Data Warehouses*

ELT - *Extraction, Loading, Transformation*

ETL - *Extraction, Transformation and Loading*

HDFS - *Hadoop Distributed File System*

JSON - *JavaScript Object Notation*

KPI - *Key Performance Indicator*

NoSQL - *Not only SQL*

OLAP - *Online Analytical Processing*

OLTP - *Online Transaction Processing*

RTBDW – *Real-Time Big Data Warehouse*

RTDW - *Real-Time Data Warehouse*

SF – *Scale Factor*

SQL - *Structured Query Language*

XML - *Extensible Markup Language*

1. INTRODUÇÃO

Neste capítulo apresentam-se o enquadramento e a motivação para esta dissertação, assim como os principais objetivos associados, a abordagem metodológica adotada e o planeamento. Termina-se com a descrição da estruturação do documento.

1.1 Enquadramento e Motivação

Nos dias que correm, cada vez mais, tem havido uma enorme evolução na utilização de dispositivos digitais, de tal forma que, segundo um artigo do Economist (2011), o número de *smartphones* e *tablets* (cerca de 480 milhões) ultrapassaram o número de *PCs* (*laptops* e *desktops* – cerca de 380 milhões), pela primeira vez, em 2011. Embora o número de *PCs* em utilização tenha atingido mil milhões em 2008 (e continue a aumentar), o mesmo artigo projeta que em 2020 o número de *smartphones* e *tablets* em utilização atinja os 20 mil milhões.

Este crescimento leva a que se gerem grandes volumes de dados diariamente, a uma escala de *terabyte* ou até mesmo *exabyte*, provenientes de fontes distintas e que têm tendência a aumentar de volume e variedade (H. Chen, Chiang, & Storey, 2012). Simultaneamente, a pressão sobre as organizações também tem vindo a crescer. Por um lado, são obrigadas a fazerem cada vez mais e melhor (maior qualidade e inovação, melhores preços e prazos) se querem ser competitivas e ter sucesso ou pelo menos sobreviver. Por outro lado, é-lhes exigido que o façam com cada vez menos (menos tempo, pessoas e custos). É aqui que as tecnologias têm de auxiliar as organizações, a fim de encontrar espaços de inovação e melhoria para aumentar a sua produtividade e competitividade (Amaral, 2005). A área de *Big Data* permite auxiliar na análise destas vastas quantidades de dados capacitando as organizações de informação relevante, de *insights*, que as ajudem a perceber melhor o negócio e o mercado em que estão inseridas, as opiniões e necessidades dos seus clientes, a identificar oportunidades e a tomar decisões em consciência e em tempo útil (H. Chen et al., 2012).

De forma resumida, o termo *Big Data* aplica-se aos volumes de dados disponíveis em diferentes graus de complexidade e ambiguidade, gerados a diferentes velocidades e que não podem ser processados utilizando tecnologias ou outros métodos de processamento tradicionais (Krishnan, 2013; Zikopoulos, Eaton, DeRoos, Deutsch, & Lapis, 2011).

Para armazenar os dados de forma adequada e com propósitos analíticos, a implementação de *Big Data Warehouses* é o mais apropriado, de tal forma que é relevante estudar este conceito, as suas características, mudanças face aos *Data Warehouses* tradicionais, tecnologias de armazenamento, entre outros aspetos (C. Costa & Santos, 2017; M. Y. Santos & Costa, 2016a). Estes *Data Warehouses* têm de ser capazes de responder às constantes mudanças nos dados e permitir processar análises em tempo útil. Neste sentido, surgem *Big Data Warehouses* suportados em tecnologia *real-time* (Li & Mao, 2015). Associado à concretização de *Big Data Warehouses*, surge por exemplo o *Hadoop*[1], contudo, este sistema abriu um novo espaço de problemas, uma vez que, apesar de se destacar no armazenamento de grandes quantidades de dados, não é otimizado para os disponibilizar de imediato. Como solução para este problema, surge o Druid[2]. Trata-se de uma tecnologia *open-source* desenhada para fazer processamento e análise de vastas quantidades de dados (permite a concretização de *queries* OLAP - *Online Analytical Processing* -, como por exemplo, *roll-up, drill-down* e agregações), não só de dados históricos, mas também de dados em *real-time*. Um dos principais desafios desta tecnologia é permitir que os utilizadores consigam analisar dados e tomar decisões em *real-time* (Yang et al., 2014). Nesta dissertação, enquadrada num projeto de investigação em colaboração entre a Bosch e a Universidade do Minho, denominado por BIPdiwa (*Business Intelligence Platform for Data Integration, Warehousing and Analysis*), importa então estudar o desempenho desta tecnologia de forma a perceber em que cenários é que a sua utilização pode, de facto, constituir uma solução.

1.2 Objetivos e Resultados Esperados

Nesta dissertação a principal finalidade é estudar detalhadamente o Druid, por se tratar de uma tecnologia relativamente recente e que pode vir a assumir um papel de relevo na implementação de sistemas de BDW, dadas as suas capacidades de processamento analítico de dados, uma funcionalidade essencial em sistemas de *Business Intelligence & Analytics* para suporte à tomada de decisão. Para atingir esta finalidade, nos objetivos a concretizar importa perceber as suas características, as suas vantagens/desvantagens e realizar um conjunto alargado de testes capazes de avaliar o seu desempenho na análise e processamento de vastas quantidades de dados (tanto históricos como em *real-time*), de forma a identificar os cenários em que a sua utilização será recomendada.

[1] http://hadoop.apache.org/
[2] http://druid.io/

Atendendo à finalidade e aos objetivos definidos, os resultados esperados passam por:

- Sistematizar o estado da arte, no que diz respeito a abordagens de implementação de BDW e RTBDW;
- Contextualização da tecnologia e principais cenários de utilização;
- Avaliar o desempenho do Druid no processamento analítico de dados num contexto de BDW, utilizando um *benchmark* de referência;
- Estudar e testar a integração entre o Hive e o Druid, em comparação com outras tecnologias SQL-on-Hadoop, utilizando um *benchmark* de referência;
- Estudar o impacto que ambientes multiutilizador podem causar no desempenho do Druid e da integração entre o Hive e o Druid, utilizando um *bechmark* de referência;
- Identificar boas práticas para a otimização do desempenho do Druid no processamento analítico de dados, em contextos de BDW;
- Implementação e validação de uma arquitetura de RTBDW em torno do Druid, recorrendo a um caso de demonstração;

1.3 Abordagem Metodológica

O tema desta dissertação enquadra-se na área dos sistemas de informação. Para além disso, inerente a este trabalho está um processo de criação de conhecimento, resultante do desenvolvimento ou inovação de artefactos e das análises da utilização e desempenho dos mesmos em diferentes cenários. Posto isto, considera-se que a *Design Science Research Methodology (DSRM) for Information Systems* é apropriada para esta dissertação (Vaishnavi & Kuechler, 2015). Como método de investigação, enquadrado nesta metodologia, será utilizada uma experiência laboratorial (*benchmarking*).

Existem várias propostas de modelos de DSRM, sendo que, no âmbito desta dissertação, será utilizada a proposta de (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007), que se apresenta na Figura 1.

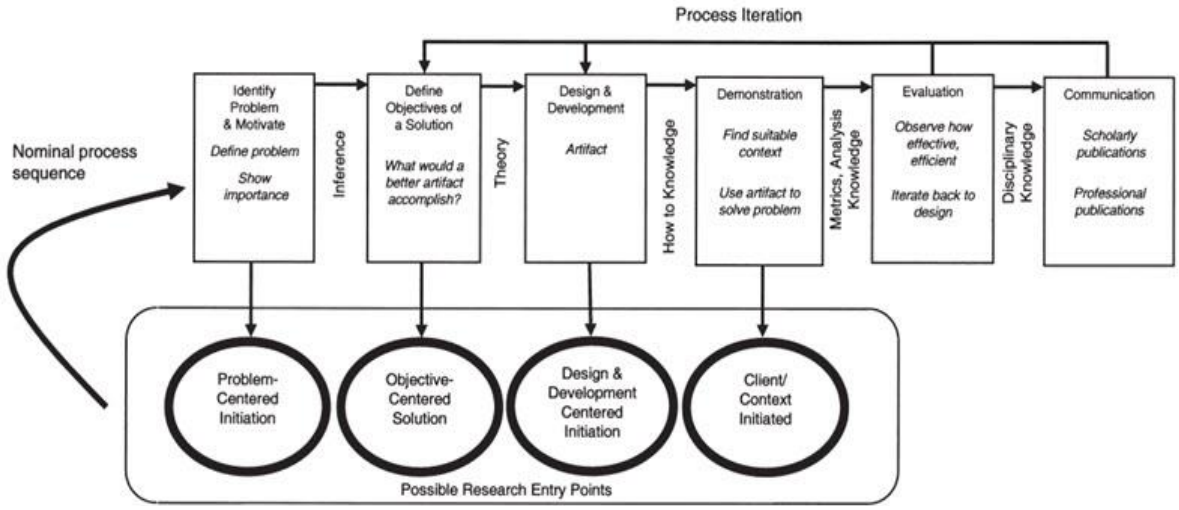


Figura 1 - DSRM for Information Systems. Retirado de (Peffers et al., 2007).

Este modelo procura sintetizar a literatura acerca da DSRM e distingue-se dos outros existentes, principalmente pelo facto de ser flexível ao ponto de permitir que a investigação se inicie a partir de vários contextos/pontos de partida (estes diferentes pontos de partida possíveis podem ser consultados na Figura 1, nas 4 elipses que surgem como "*Possible Research Entry Points*") (Vaishnavi & Kuechler, 2015). No caso desta dissertação, a investigação será iniciada centrada no problema, percorrendo-se, de seguida, a sequência de 6 fases ilustradas na Figura 1.

Outro dos motivos que levou a adotar este modelo é a qualidade e o nível de detalhe com que os autores definem e descrevem todas as suas fases, resultados esperados e a sequência do processo. Os autores incluem ainda 4 casos de estudo onde aplicam a metodologia a casos reais de projetos de investigação em sistemas de informação, anteriormente publicados, e que tiveram diferentes pontos de partida.

1.3.1 Descrição das Tarefas

A decomposição das várias tarefas/atividades nas quais se decompõem os trabalhos desta dissertação, resultam do cruzamento da metodologia adotada (DSRM de Peffers et al., (2007), com outras tarefas consideradas relevantes para o resultado final. Como consequência deste cruzamento, resultaram as seguintes tarefas:

1. Definição do plano de trabalho – elaborar um documento com a identificação do tema da dissertação, o enquadramento e motivação, os principais objetivos e resultados esperados, assim como a abordagem metodológica que irá ser seguida, a identificação e descrição das tarefas e, por fim, a calendarização das mesmas.
2. Identificação do problema e motivação – definir o problema específico da investigação e justificar o valor da solução. Visto que a definição do problema será utilizada para desenvolver um artefacto que possa, efetivamente, constituir uma solução será importante detalhar concetualmente o problema, de tal forma que a solução consiga capturar a sua complexidade. Justificar o valor da solução, não se trata apenas de motivar o investigador e a audiência interessada a procurar uma solução, mas também a ajudar na perceção do raciocínio que conduz o investigador à sua compreensão do problema. O conhecimento do estado do problema e a importância de se encontrar solução para o mesmo são condições relevantes para esta tarefa.
3. Definição dos objetivos da solução – inferir os objetivos da solução a partir da definição do problema e do conhecimento do que é possível e viável. Os objetivos tanto podem ser quantitativos, quantificando certos termos que constituem a solução ideal, como qualitativos, descrevendo, por exemplo, como é que é expectável que o novo artefacto represente solução para os problemas ainda por adereçar. Para a realização sucedida desta atividade espera-se como requisito o conhecimento do estado do problema, das soluções atuais e a eficácia das mesmas, caso existam.
4. Revisão do estado da arte – identificar, analisar e compreender a literatura científica e técnica relevante definindo e relacionando os conceitos, os problemas e os avanços relevantes que têm sido alcançados na área em que se insere esta dissertação. Nesta tarefa serão apresentados os principais conceitos e trabalhos na área de *Big Data Warehouses*, sendo que depois se dará uma maior atenção aos estudos que se preocupem com a otimização do processamento e análise de vastas quantidades de dados (históricos e em *real-time*).
5. Enquadramento tecnológico – explorar e apresentar as tecnologias enquadradas no âmbito da dissertação. Trata-se de uma fase importante porque resulta numa familiaridade para com as tecnologias que serão utilizadas e até numa perceção das tecnologias concorrentes que poderão existir para responder aos mesmos problemas. Nesta dissertação, a tecnologia alvo de estudo é o Druid, contudo, esta tecnologia terá de ser enquadrada numa arquitetura, juntamente com outras tecnologias, pelo que é importante conhecê-las e enquadrá-las. Paralelamente, poderá ser

importante perceber que tecnologias é que possuem características capazes de responder aos mesmo desafios que o Druid, representando um seu concorrente.

6. Conceção e desenvolvimento – esta atividade inclui a determinação das funcionalidades desejadas, a definição da arquitetura e, por fim, a criação do artefacto. Estes artefactos podem ser construções, modelos, métodos ou instanciações. O bom desempenho desta atividade está dependente do conhecimento teórico que pode ser utilizado para fazer a transição dos objetivos para a conceção e desenvolvimento da solução.
7. Demonstração – demonstrar a utilização do artefacto para resolver o problema. Pode ser feita através de experiências, simulações ou outras formas consideradas apropriadas. Esta atividade requer um conhecimento prévio efetivo de como utilizar o artefacto para resolver o problema.
8. Avaliação – observar e medir quão bem o artefacto suporta a solução para o problema, através da comparação dos objetivos definidos com os resultados obtidos na demonstração. Requer o conhecimento de métricas e de técnicas de análise relevantes. Terminada esta atividade, o investigador pode decidir se deve iterar, voltando para a atividade 6 (Conceção e desenvolvimento), a fim de tentar otimizar o artefacto, ou se prossegue para a atividade de comunicação, deixando as melhorias adicionais para projetos futuros.
9. Comunicação – comunicar todo o processo aos investigadores e a toda a audiência interessada, como profissionais da área, desde o problema e sua importância, até ao artefacto, destacando a sua utilidade e novidade, assim como o rigor da sua conceção e a sua eficácia para resolver o problema identificado. O sucesso desta atividade requer o conhecimento para saber produzir documentos com rigor científico e identificar oportunidades que ampliem o alcance do trabalho e o valorizem junto da comunidade científica. Posto isto, esta fase inclui a escrita e apresentação da dissertação, assim como a publicação de artigos relacionados em jornais, revistas ou conferências relevantes na área.

1.3.2 Processo de Revisão de Literatura

A criação de um processo para a recolha e seleção da literatura relevante é essencial para se seguir um método consistente e que facilite a pesquisa e leitura de conteúdos associados aos temas cobertos por esta dissertação, assim como a identificação das áreas onde a investigação é prioritária

(Webster & Watson, 2002). Neste sentido, numa fase inicial foi concebido um fluxograma, ilustrado na Figura 2, e que representa o plano seguido para a obtenção e seleção da documentação científica.

Antes de explicar o fluxograma elaborado, é importante salientar que a literatura passível de ser estudada corresponde ao subconjunto da literatura disponível para a comunidade científica da Universidade do Minho, de entre toda a literatura existente. Além disso, foi definida uma restrição temporal, limitando a pesquisa a documentação com data igual ou superior a 2001, salvo raras exceções referentes a autores ou publicações científicas relevantes na área.

No que diz respeito aos serviços de indexação, foram utilizados os seguintes: Google Scholar, Scopus, Web of Science, IEEE Xplore, RepositoriUM, AIS Electronic Library, Semantic Scholar, ACM Digital Library. Por vezes, foi também necessária a utilização do Google para identificar literatura de índole técnica e/ou relacionada com conferências não indexadas nos serviços anteriormente referidos.

A identificação e seleção da literatura ocorreu entre outubro de 2017 e janeiro de 2018, recorrendo às seguintes palavras chave, de forma individual ou combinada: *Big Data*, *Data Warehouse*, *Big Data Warehouse*, *NoSQL*, *Druid*, *real-time.* Quanto à forma como a literatura obtida nas pesquisas foi filtrada até se chegar ao conjunto de literatura utilizada, esta está ilustrada na Figura 2.

De uma forma resumida, no fluxograma apresentado, a literatura é classificada em 3 categorias:

- **Leitura muito relevante** – fazer uma análise e leitura cuidada;
- **Leitura relevante** –fazer uma leitura com baixo grau de profundidade, a fim de perceber se a documentação se comprova importante, adequada e uma mais valia à investigação;
- **Leitura irrelevante** – a documentação não se revela útil, pelo que deve ser descartada.

Considera-se ainda importante clarificar que a decisão inicial quanto à potencial relevância da documentação inclui uma ponderação entre o número de citações e a data de publicação, isto é, podem existir casos em que um documento com data inferior a 2001 é considerado mais relevante que um documento com data mais recente. Para terminar, destaca-se também que no caso de se tratar de documentação classificada como relevante ou muito relevante são analisadas as referências da

documentação em causa, assim como os artigos que o citam, com o objetivo de identificar literatura potencialmente útil.

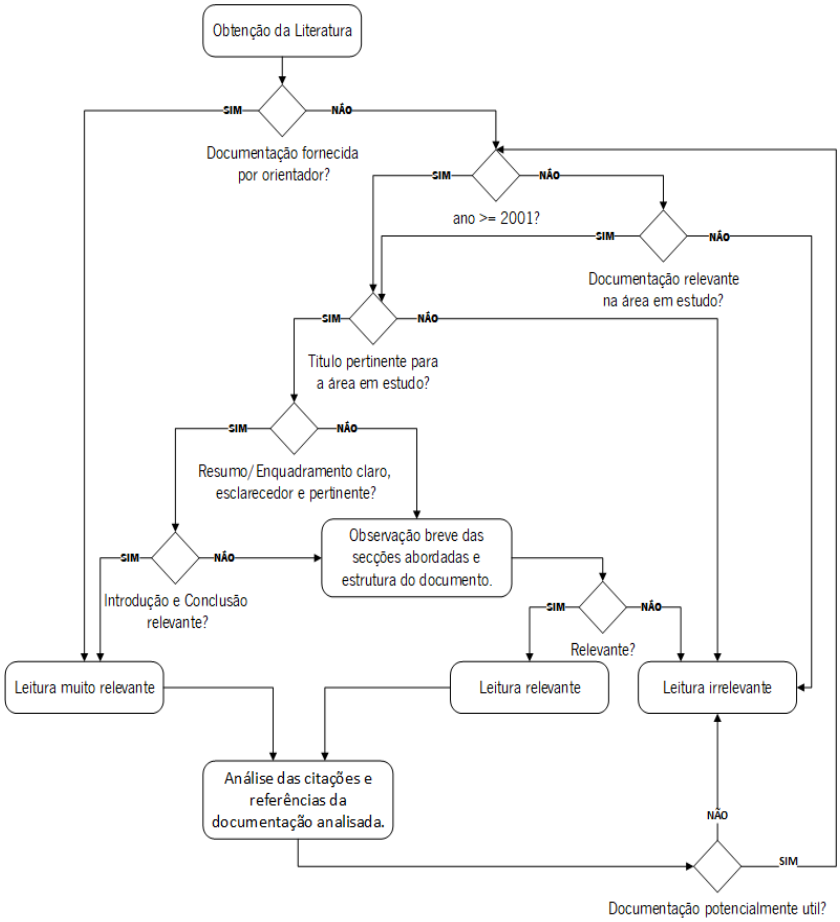


Figura 2 - Processo de Obtenção e Seleção da Literatura.

1.4 Projeto de Investigação

Esta dissertação, tal como já foi referido anteriormente, surge enquadrada no âmbito de um projeto de investigação, numa colaboração entre a Bosch e a Universidade do Minho, denominado por *Business Intelligence Platform for Data Integration, Warehousing and Analysis* (BIPdiwa), no qual o autor deste documento é bolseiro desde julho de 2017. O objetivo do projeto é desenvolver um sistema integrado de dados que permita, através de um processo interativo, o desenvolvimento do *Data Warehouse* Organizacional, contribuindo para o aumento da qualidade das operações de fábrica, em termos da eficiência do acesso e qualidade da informação crítica necessária para a tomada de decisão e envolvimento dos atores, a montante e jusante da cadeira de valor.

Desta forma, espera-se que o trabalho desta dissertação possa contribuir para o projeto de investigação que decorre em simultâneo.

1.5 Organização do Documento

Esta dissertação encontra-se organizada em seis capítulos. Inicialmente, no capítulo 1, é possível encontrar a introdução ao projeto, na qual é enquadrado o âmbito do estudo, é identificado o problema de investigação, os objetivos e resultados esperados e apresentada a metodologia de investigação adotada. No capítulo 2, é apresentado um enquadramento concetual, explorando alguns conceitos considerados muito pertinentes para esta dissertação, nomeadamente o conceito de *Big Data*, *Data Warehouse* e *Big Data Warehouse*. É ainda estudada a importância de implementar *Big Data Warehouses* que suportem análises a dados históricos e a dados em *real-time* e são apresentados os trabalhos relacionados aos objetivos desta dissertação. Finaliza-se este capítulo, com a apresentação de um mapa de conceitos para sintetizar o relacionamento entre as várias temáticas e tecer algumas considerações. No capítulo 3 apresenta-se o enquadramento tecnológico focado, essencialmente no Druid, por ser a tecnologia principal em estudo nesta dissertação. Contudo, abordam-se outras tecnologias relevantes e que poderão integrar com o Druid, construindo um RTBDW que suporte análises a dados históricos e a dados em *real-time*. O capítulo 4 apresenta os vários cenários de teste e respetivos resultados, com o objetivo de avaliar o Druid no processamento analítico de dados em contextos de BDW. Seguidamente, no capítulo 5 é apresentada uma arquitetura de RTBDW e um caso de demonstração da aplicação da mesma. O último capítulo apresenta as principais conclusões, sistematizando o trabalho que foi realizado, os objetivos alcançados, as dificuldades e limitações encontradas e o trabalho futuro.

2. ENQUADRAMENTO CONCETUAL

O conteúdo presente neste capítulo visa contribuir para a sistematização e compreensão dos conceitos mais relevantes associados ao tema desta dissertação. Com este enquadramento concetual pretende-se dissipar eventuais dúvidas ou dificuldades que possam surgir durante a dissertação. Assim, espera-se que o enquadramento, alicerçado em literatura relevante proveniente da comunidade científica e técnica, possa formar uma base firme de conhecimento em volta das temáticas endereçadas, ajudando a perceber cada um dos conceitos, os desafios encontrados, as áreas que carecem mais de investigação e a enquadrar melhor esta dissertação.

Começar-se-á por enquadrar o conceito de *Big Data*, abordando não só como é que pode ser definido, mas também as suas características, motivações, áreas de intervenção, desafios inerentes e diferenças face às tecnologias tradicionais. De seguida, estudam-se vários conceitos sobre a temática do armazenamento de dados, nomeadamente: as bases de dados SQL, NoSQL e NewSQL, os *Data Warehouses* e *Big Data Warehouses*, assim como os modelos de dados e as diferenças entre os sistemas analíticos e os sistemas operacionais. Por fim, aborda-se a importância do *real-time*, apresentam-se os trabalhos relacionados com esta dissertação e termina-se tecendo algumas considerações sobre o mapa concetual que sistematiza os principais conceitos desta dissertação, assim como as relações entre si.

2.1 *Big Data*

Nos dias que correm, é praticamente inconcebível viver num mundo no qual vastos volumes de dados não estejam a ser gerados através de diversas fontes. Contudo, e apesar da sua importância já ser amplamente reconhecida, o mesmo não acontece quando o assunto é a sua definição, geralmente envolta em ambiguidade e opiniões distintas. Posto isto, é importante referir que o conceito de *Big Data* deve ser visto como uma abstração de diferentes componentes, e não apenas como um vasto volume de dados (M. Chen, Mao, & Liu, 2014; C. Costa & Santos, 2017).

Gandomi e Haider (2015) corroboram que o conceito de *Big Data* tem sido envolto em ambiguidade, gerando um conjunto de opiniões que formam um discurso fragmentado e acrescentam que o surgimento do *Big Data* foi repentino, constituindo um tópico de atenção e, em certa medida, de receio. No sentido de consolidar o discurso que os próprios identificaram como fragmentado, Gandomi e Haider (2015) contribuíram com um estudo dos contributos que consideram relevantes começando por referir que o volume de dados é a primeira e, por vezes, a única dimensão que é abordada quando

se refere *Big Data*, contudo, no entendimento dos autores, esta é apenas uma de muitas dimensões que o constituem, visto que, outras características como a frequência com que os dados são gerados são igualmente importantes na definição de *Big Data*.

No seu trabalho, Ward e Barker (2013) alertam para a associação pouco rigorosa da definição de *Big Data* ao armazenamento e análise de dados, assim como a falta de quantificação para o definir. Os autores prosseguem apresentando várias definições, umas alicerçadas nas características do *Big Data*, outras no aumento das fontes de dados tradicionais com a adição de dados não estruturados, outras tentando quantificar o *Big Data* e algumas que referem a inadequação das tecnologias tradicionais para lidar com estes novos tipos de dados. Por fim, os autores concluem que as definições de *Big Data* incluem pelo menos um dos seguintes aspetos: volume, complexidade, tecnologias. Ou seja, apesar de críticos, Ward e Barker (2013) incluem na sua própria definição de *Big Data*, o armazenamento e análise de vastos e complexos conjuntos de dados utilizando um conjunto de novas técnicas.

Zikopoulos et al. (2011), para definir *Big Data*, começam por dar uma pista aos leitores, argumentando que nós fazemos parte deste fenómeno todos os dias. Os autores defendem que *Big Data* se aplica à informação que não pode ser processada ou analisada utilizando processos ou ferramentas tradicionais.

A visão de Krishnan (2013), em certa medida, complementa a anterior, dizendo que *Big Data* pode ser definido como volumes de dados disponíveis em vários graus de complexidade, gerados a diferentes velocidades e com vários graus de ambiguidade, que não podem ser processados utilizando tecnologias, métodos, algoritmos tradicionais ou quaisquer outras tecnologias pré-configuradas.

O trabalho de Dumbill (2013) contribui com uma definição, que possui pontos comuns com as visões de Zikopoulos et al. (2011) e Krishnan (2013), afirmando que *Big Data* são dados que excedem a capacidade de processamento dos sistemas convencionais de bases de dados. O volume de dados é muito vasto, move-se muito rapidamente ou não cumpre as restrições das arquiteturas de bases de dados convencionais. Posto isto, é imperativo escolher formas alternativas para processar estes dados e extrair valor dos mesmos. M. Chen et al. (2014) concordam com esta definição e acrescentam o facto do software/hardware tradicional não conseguir, em tempo razoável, recolher, gerir e processar este volume e tipo de dados.

De facto, e apesar de existirem várias definições distintas, algumas delas tendem a convergir em alguns pontos. Mais recentemente, De Mauro, Greco e Grimaldi (2016) levaram a cabo um trabalho com o objetivo de propor uma definição formal para o *Big Data*, baseada nas suas características essenciais e coerente com as definições habitualmente utilizadas. Como resultado deste estudo, os autores sugerem

que *Big Data* é caracterizado por um vasto volume, velocidade e variedade de dados, ao ponto de requerer tecnologias e métodos analíticos específicos para a sua transformação em valor.

2.1.1 Principais Características

Considerando os contributos anteriormente referidos, verifica-se que grande parte das definições se baseiam em três características principais: volume, variedade e velocidade. O primeiro a identificar estas três características foi Laney (2001), apresentando o modelo dos 3Vs, que emergiu como uma *framework* para descrever *Big Data* (Gandomi & Haider, 2015). Este modelo encontra-se representado na Figura 3.

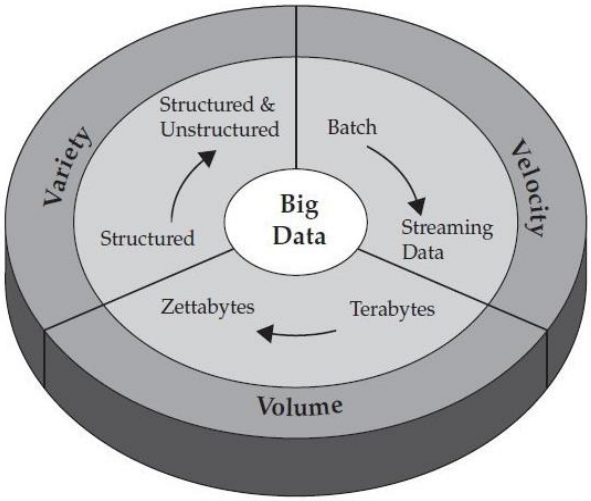


Figura 3 - Representação do Modelo dos 3Vs. Retirado de (Zikopoulos et al., 2011).

Seguidamente, são descritas cada uma das características apresentadas na Figura 3, o volume, a variedade e a velocidade.

Começando pelo **volume**, de acordo com Gandomi e Haider (2015), este diz respeito à magnitude dos dados, descrita geralmente em múltiplos *terabytes* e *petabytes*. Contudo, os mesmos autores argumentam que as definições de volume são relativas e variam de acordo com o tempo e o tipo de dados. É impossível definir um limite específico, a partir do qual se passa a falar de volumes de *Big Data*, visto que, no futuro a capacidade de armazenamento irá aumentar, permitindo capturar ainda maiores volumes. Além disso, duas fontes de dados com o mesmo volume podem necessitar de diferentes tecnologias, tendo em conta o seu tipo de dados (exemplo: dados tabulares versus dados de vídeo). Para Krishnan (2013), o volume é caracterizado pela quantidade de dados que são gerados continuamente e, segundo C. Costa e Santos (2017) e Zikopoulos et al. (2011), a principal causa para o

crescimento do volume é o facto de se guardarem todas as interações com a maior parte dos serviços e/ou aplicações disponíveis no mundo.

Com a utilização de sensores, dispositivos inteligentes e também com as tecnologias de colaboração social, têm-se gerado vastos volumes de dados, sem que seja possível existir um controlo sobre o formato dos mesmos, surgindo assim a característica da **variedade** (Krishnan, 2013; Zikopoulos et al., 2011). Os dados têm-se tornado cada vez mais complexos, podendo chegar de forma estruturada (exemplo: bases de dados relacionais), semiestruturada (exemplo: *logs* de servidores *web*, *Extensible Markup Language* - XML) ou não estruturada (exemplo: *posts* nas redes sociais, vídeos, imagens) (Chandarana & Vijayalakshmi, 2014; Gandomi & Haider, 2015; Zikopoulos et al., 2011). As tecnologias tradicionais apresentam muitas dificuldades para lidar com a variedade, tornando necessária uma mudança de paradigma para que as organizações acomodem dados estruturados, semiestruturados e não estruturados, a fim de maximizarem as vantagens que retiram do valor dos mesmos (C. Costa & Santos, 2017; Zikopoulos et al., 2011). Gandomi e Haider (2015) clarificam a definição de variedade, com o exemplo do conhecimento que se pode retirar através das tecnologias de reconhecimento facial, numa loja de comércio aberta ao publico, na qual detetam a idade e o género dos clientes, assim como os seus padrões de movimento. Estas tecnologias de reconhecimento facial geram dados em formatos que causam problemas às tecnologias de armazenamento e processamento tradicionais, no entanto, os dados em questão possuem um valor inquestionável.

Finalmente, no que à **velocidade** diz respeito, para Gandomi e Haider (2015), esta refere-se tanto à taxa a que os dados são gerados, como à velocidade a que estes devem ser analisados e dar suporte à tomada de decisão. Esta velocidade na geração de dados de várias fontes, desde *batch* até *real-time* é a principal causa do *Big Data* (Chandarana & Vijayalakshmi, 2014). É relevante aplicar a definição de velocidade a dados em movimento, ao invés de aplicá-la à taxa a que os dados são recolhidos, armazenados e recuperados do sistema de armazenamento. Os fluxos de dados contínuos podem criar vantagens competitivas em contextos onde a identificação de tendências, problemas ou oportunidades deve ser feita em pequenos períodos de tempo, antes de qualquer outra pessoa (C. Costa & Santos, 2017; Krishnan, 2013; Zikopoulos et al., 2011).

Através desta contextualização em torno dos três Vs atribuídos ao *Big Data*, percebe-se a intensa complexidade associada. Daí se depreende que as três características enunciadas no modelo dos três Vs não devem ser vistas como independentes. Quem o defende é Krishnan (2013) que identifica três novas características que emergem do cruzamento entre volume, variedade e velocidade, como ilustra a Figura 4.

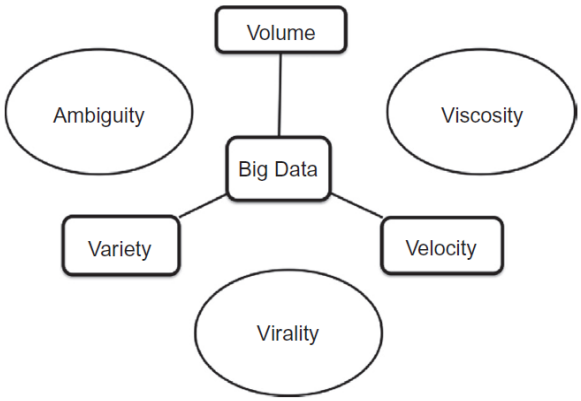


Figura 4 - Modelo dos 3Vs com Características Adicionais. Retirado de (Krishnan, 2013).

Estas três novas características apresentadas na Figura 4, são descritas por Krishnan (2013), da seguinte forma:

- **Ambiguidade:** manifesta-se entre o volume e a variedade e é fruto da falta de metadados. Por exemplo, o "M" e o "F" podem significar género (Masculino ou Feminino) ou podem ser interpretados como Segunda-feira (em inglês *Monday*) e Sexta-feira (em inglês *Friday*);
- **Viscosidade:** é a resistência presente no fluxo de dados e que provoca atrasos. Manifesta-se entre o volume e a velocidade. Esta resistência pode ter origem, por exemplo, nas regras de negócio ou em limitações tecnológicas;
- **Viralidade:** manifesta-se entre a velocidade e a variedade e caracteriza o quão rápido os dados são partilhados entre os diversos *nodes* de uma rede.

Com o passar do tempo e o crescente interesse da comunidade científica e técnica, no conceito de *Big Data*, foram existindo contributos de outros autores sugerindo a inserção de novas características no modelo dos três Vs. Uma das características sugeridas na literatura é constantemente mencionada, de forma implícita e destaca o problema mais crítico em *Big Data,* que é como extrair **valor** de vastos conjuntos de dados, de diferentes tipos e gerados muito rapidamente (M. Chen et al., 2014). As restantes características identificadas na literatura são a veracidade, a validade, a volatilidade, a variabilidade e a

complexidade, sendo todas estas descritas de seguida. De referir que a variabilidade e a complexidade, tal como referem C. Costa e Santos (2017), não são tão mencionadas na literatura.

O **valor** é visto por Khan, Uddin, e Gupta (2014) como um V especial, porque se refere ao resultado desejado do processamento de *Big Data,* que usualmente tem pouco valor (quando os dados estão na forma original, sem tratamento) relativamente ao seu volume, exigindo análises adequadas (Gandomi & Haider, 2015). De acordo com Chandarana e Vijayalakshmi (2014), o valor é o conhecimento escondido que se consegue extrair dos dados e que se traduz numa vantagem competitiva e na melhoria do negócio.

A **veracidade**, segundo Gandomi e Haider (2015), diz respeito à falta de fiabilidade inerente a algumas fontes de dados. Os autores exemplificam com os sentimentos dos clientes, extraídos das redes sociais, que apesar de incertos por natureza, podem ser valiosos. Tal como referem Chandarana e Vijayalakshmi (2014), quando se lida com dados que fluem a grande velocidade, por vezes, não se pode despender tempo a fazer a limpeza dos mesmos, antes do seu consumo, como tal, são necessários mecanismos para lidar com uma combinação de dados precisos, imprecisos, corretos ou incorretos. Khan et al (2014) concordam com esta visão, destacando que, no seu ponto de vista, este é o V mais relevante para o processamento e análise de *Big Data*, porque não se podem tomar decisões que influenciam o dia a dia das organizações e o seu futuro, tendo por base dados que não são confiáveis.

A **validade** é similar ao conceito de veracidade e representa a exatidão e a precisão dos dados, de acordo com a utilização pretendida. Os dados podem não ter questões de veracidade mas não serem válidos, por exemplo, no caso de serem incompreensíveis (Khan et al., 2014).

A **volatilidade** está relacionada com a política de retenção dos dados de uma organização, referindo-se ao tempo no qual os dados devem ser mantidos e a partir do qual podem ser destruídos, porque deixam de ser relevantes (Khan et al., 2014).

A **variabilidade** está relacionada com a variação das taxas a que os dados fluem, de acordo com diferentes picos e depressões periódicas de velocidade (Gandomi & Haider, 2015).

A **complexidade** refere-se ao facto do *Big Data* ser gerado através de diversas fontes, o que impõe o desafio de integrar, associar, limpar e transformar os dados recebidos de todas essas fontes (Gandomi & Haider, 2015).

A Figura 5 resume as várias características identificadas na literatura e caracterizadas ao longo desta subsecção.

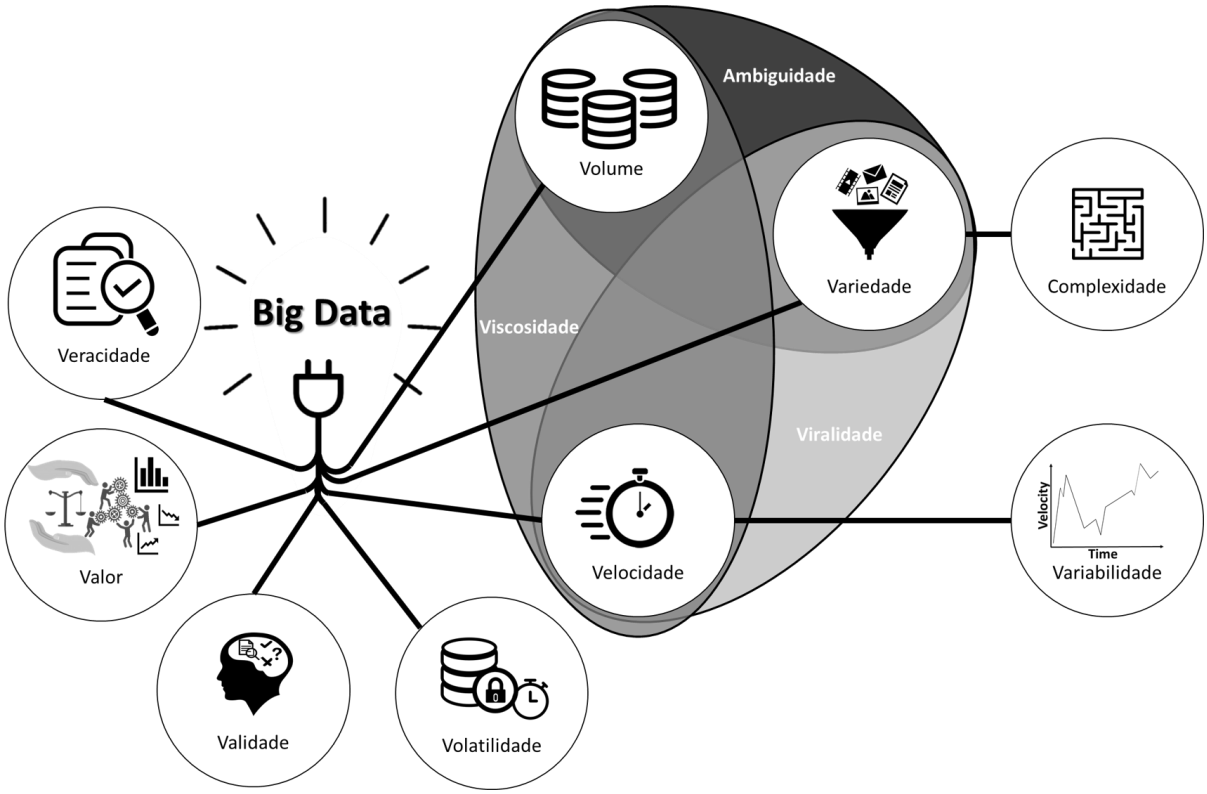


Figura 5 - Principais Características do Big Data Identificadas na Literatura.

2.1.2 Motivação e Importância do *Big Data*

O número de dispositivos inteligentes utilizados, assim como os sensores e as tecnologias de colaboração social têm vindo a aumentar consideravelmente (Economist, 2011; Zikopoulos et al., 2011). Este aumento, aliado ao facto de atualmente se guardarem todas as interações, com a maior parte dos serviços disponíveis no mundo, implica que se estejam gerar dados, no universo digital, a taxas cada vez maiores (Chandarana & Vijayalakshmi, 2014; C. Costa & Santos, 2017; Zikopoulos et al., 2011). Para se compreender melhor a amplitude deste fenómeno, apresentam-se na Figura 6, um conjunto de estatísticas elucidativas da evolução dos dados gerados.

Todos estes dados gerados e armazenados podem produzir informação muito valiosa e, consecutivamente, vantagens competitivas, quando as organizações têm capacidade para os processar. As organizações querem o máximo de informação disponível, porque, tal como refere a Gartner, na Figura 6, "a informação é o petróleo do século XXI". Assim, quanto mais informação as organizações tiverem, mais fundamentadas poderão ser as suas decisões e maior poderá ser o proveito a retirar das vantagens competitivas sobre as organizações congéneres (Chandarana & Vijayalakshmi, 2014).

Com a preocupação das organizações em guardar o máximo de dados disponíveis, dados esses muito vastos, gerados a altas taxas e que possuem diferentes graus de complexidade, ao ponto de não poderem ser armazenados e processados por tecnologias tradicionais, surge o *Big Data,* tal como referido anteriormente (M. Chen et al., 2014; De Mauro et al., 2016; Dumbill, 2013; Krishnan, 2013; Zikopoulos et al., 2011).

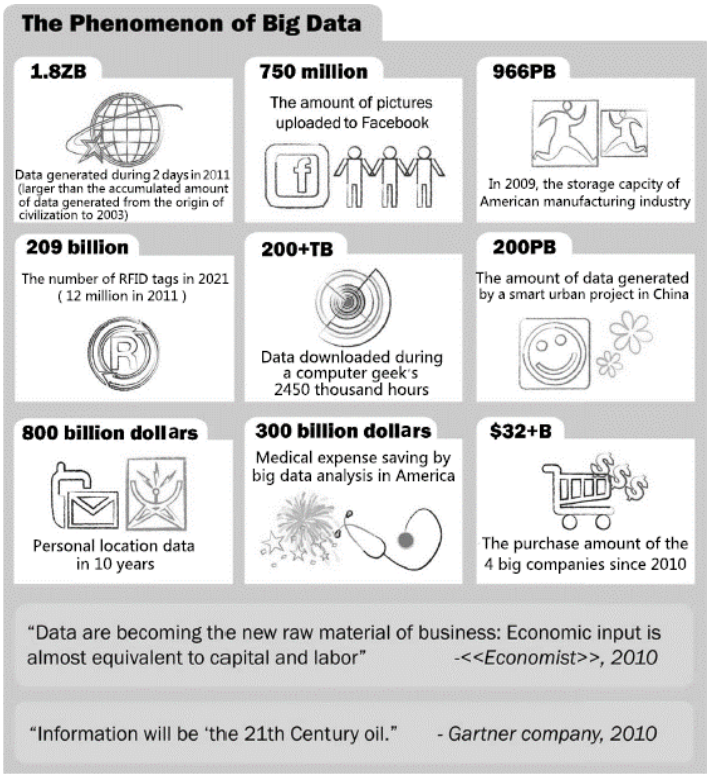


Figura 6 - O fenómeno do Big Data. Retirado de (M. Chen et al., 2014).

Apesar do enquadramento efetuado a este tema na subsecção anterior, continuam a ser levantadas questões associadas ao porquê desta área de intervenção e do momento em que surge. Segundo Krishnan (2013), a reposta à primeira questão está na promessa do *Big Data* em aceder a vastos volumes de dados que podem ser úteis para retirar informações relevantes dos mesmos, sendo que, este processo pode ser executado com muito pouca intervenção humana, tornando as análises mais simples e livres de erros. A resposta à segunda questão, de acordo com este autor, está na disponibilidade de infraestruturas que integram *frameworks* de processamento de dados com plataformas, como por exemplo Hadoop e NoSQL, que resultam em custos significativamente menores e escalabilidades mais altas, comparando com as plataformas tradicionais.

Krishnan (2013) termina referindo que o aspeto chave a ser percebido é que o *Big Data* sempre existiu, porém, era utilizado de forma muito mais manual e com muita intervenção humana. Aquilo que

foi alterado e provocou que *Big Data* se tornasse uma *buzzword* foi o aparecimento da capacidade de processamento automático, extremamente rápido, escalável e flexível.

De facto, já eram geradas grandes quantidades de dados (embora nos dias que correm sejam gerados cada vez mais), contudo, o processamento dos mesmos necessitava de muita intervenção humana. Além disso, segundo Zikopoulos et al. (2011), a percentagem de dados que as organizações conseguiam processar estava muito abaixo dos dados disponíveis, por isso era obrigatório pensar em alternativas mais capazes. Assim, a partir de 2011, a IBM e outras organizações tecnológicas apostaram no setor, promovendo novas tecnologias capazes de lidar com o fenómeno e de reduzir a intervenção humana. A partir daqui o interesse foi crescendo cada vez mais por parte da comunidade científica e técnica (Ekbia et al., 2015; Gandomi & Haider, 2015).

A fim de perceber se, efetivamente, se verifica um aumento do interesse por parte da comunidade científica e técnica, em *Big Data*, ilustra-se, na Figura 7, a evolução do número de publicações contendo o termo "Big Data", entre os anos de 2001 e 2017, em quatro bases de dados académicas comummente utilizadas e que são também utilizadas nesta dissertação (Scopus, IEEE Xplore, Web Of Science e Semantic Scholar). Por análise da mesma figura, verifica-se que, efetivamente, depois do ano de 2011 começou a existir um crescendo de interesse pelo termo.

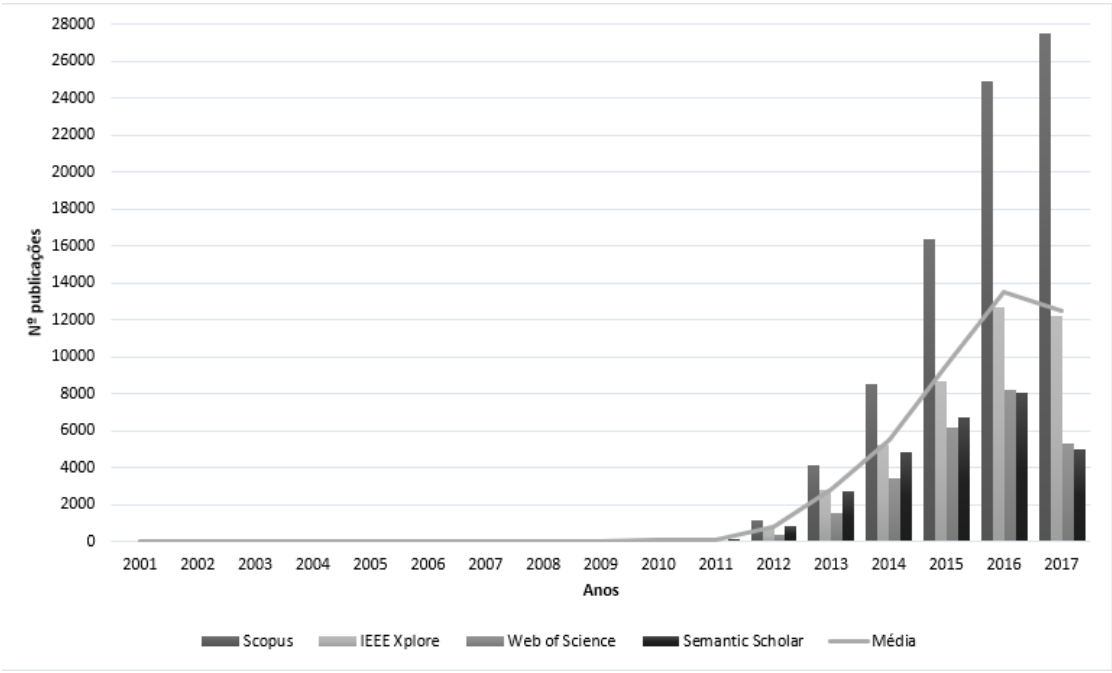


Figura 7 - Evolução do número de publicações contendo o termo "Big Data" entre os anos de 2001 e 2017.

Para se conseguir perceber melhor o número de publicações para os anos que obtiveram números mais reduzidos, apresenta-se, na Figura 8, um gráfico semelhante ao anterior, mas com escala logarítmica, na qual o "x" corresponde ao número de publicações.

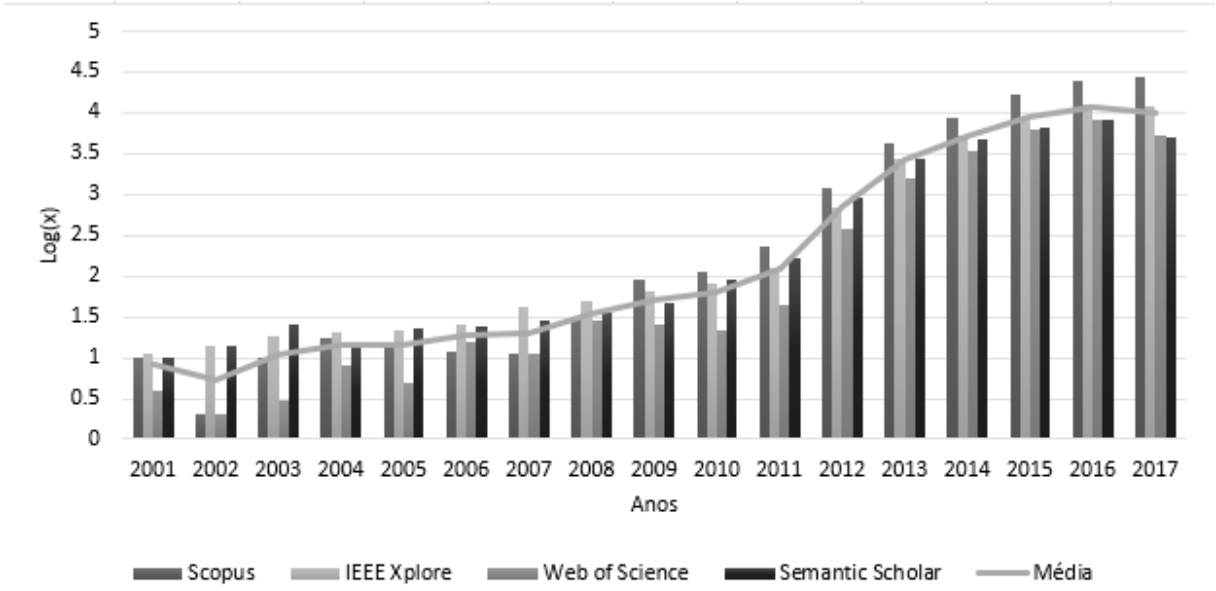


Figura 8 - Evolução do número de publicações contendo o termo "Big Data" entre os anos de 2001 e 2017, com escala logarítmica.

Efetuadas estas considerações, é importante acrescentar mais uma questão a esta subsecção: "Quando é que se deve utilizar uma solução *Big Data*?".

Zikopoulos et al. (2011) afirmam que as soluções de *Big Data* não podem ser vistas como um substituto para as tecnologias relacionais existentes. Aliás, as tecnologias relacionais podem ter um papel muito relevante quando conjugadas com uma plataforma de *Big Data*. Os autores fazem até uma analogia entre esta questão e as mãos direita e esquerda. Cada uma das mãos tem pontos fortes. Por exemplo, no basebol, uma das mãos é melhor a arremessar e a outra a capturar. É claro que se pode trocar e colocar uma mão a fazer o trabalho da outra, contudo, os resultados provavelmente não serão os mesmos. O mesmo acontece com o *Big Data*, que tem muito potencial, mas há tarefas para as quais é melhor e outras em que as bases de dados tradicionais são mais indicadas.

Neste contexto, Zikopoulos et al. (2011) estabelecem alguns princípios chave para a utilização de soluções de *Big Data*. De acordo com os autores, deve utilizar-se este tipo de soluções quando:

- Se analisam dados não só estruturados, mas também semiestruturados e não estruturados provenientes de várias fontes distintas;
- Se pretende analisar vastos volumes de dados, visto que, a análise de uma determinada amostra não se revela tão eficaz;
- Se pretende realizar análises iterativas e exploratórias.

A motivação para se utilizar *Big Data* advém também das imensas oportunidades associadas, sendo que algumas delas já foram sendo enunciadas anteriormente, como a criação de valor e o seu aproveitamento para suportar o processo de tomada de decisão e alcançar vantagens competitivas (Gandomi & Haider, 2015).

O interesse pelo *Big Data* conduziu a que, em 2012, o regime do anterior Presidente Obama anunciasse um investimento de mais de 200 milhões de dólares, distribuído por várias agências governamentais, com o fim de dar um passo em frente na investigação de métodos e instrumentos para aceder, organizar e extrair conhecimento de vastos volumes de dados digitais. Isto porque a ponte entre o *Big Data* e o conhecimento escondido neste tipo de dados é crucial a todas as áreas consideradas uma prioridade a nível nacional (C. L. P. Chen & Zhang, 2014; Sagiroglu & Sinanc, 2013).

C. L. P. Chen e Zhang (2014) argumentam que retirar conhecimento valioso do *Big Data* se irá tornar a competição básica das organizações dos dias de hoje e prosseguem referindo que existem muitas vantagens que podem ser obtidas através do aproveitamento do *Big Data*. Os autores demonstram ainda as vantagens que as 560 organizações inquiridas associam ao *Big Data*. Os resultados, retirados de (C. L. P. Chen & Zhang, 2014), são apresentados na Figura 9, na qual, o eixo vertical, representa a percentagem de organizações que pensa que o *Big Data* as pode ajudar num determinado aspeto. Nesta figura, destaca-se que 51% das organizações inquiridas acreditam que o *Big Data* as irá auxiliar a melhorar a eficiência operacional.

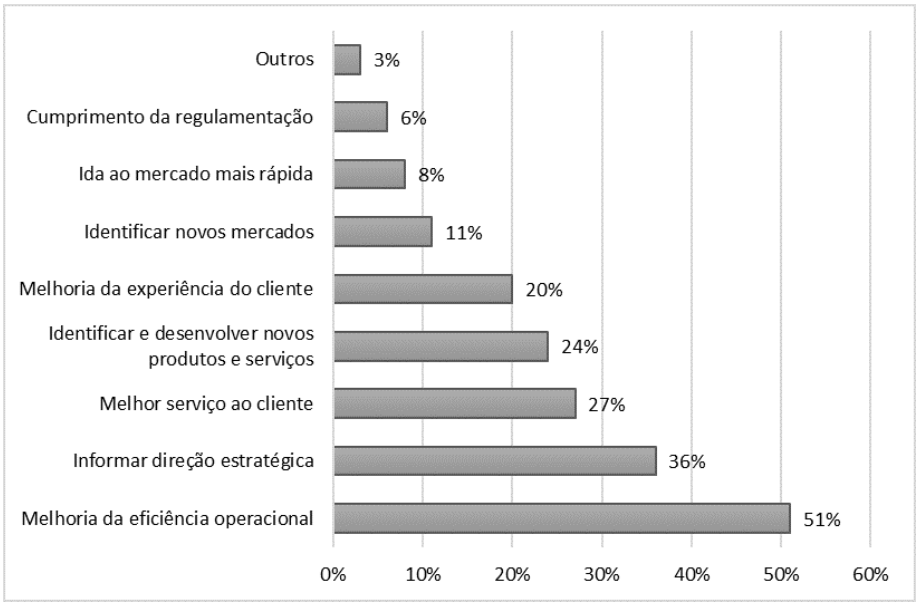


Figura 9 - Oportunidades do Big Data. Adaptado de (C. L. P. Chen & Zhang, 2014).

Segundo Sagiroglu e Sinanc (2013), qualquer organização pode beneficiar do *Big Data* e são muitos os exemplos de aplicações deste na literatura, incluindo: astronomia, ciência atmosférica, ciências

da vida, desastres naturais e gestão de recursos, setor privado, vigilância militar, redes sociais, *web logs*, documentos, fotografias, áudios, vídeos, sensores, entre muitos outros.

Como forma de agregar os contributos de Chandarana e Vijayalakshmi (2014), Manyika et al. (2011) e Sagiroglu e Sinanc (2013), apresenta-se, na Figura 10, as várias áreas onde o *Big Data* pode desempenhar um papel fundamental.

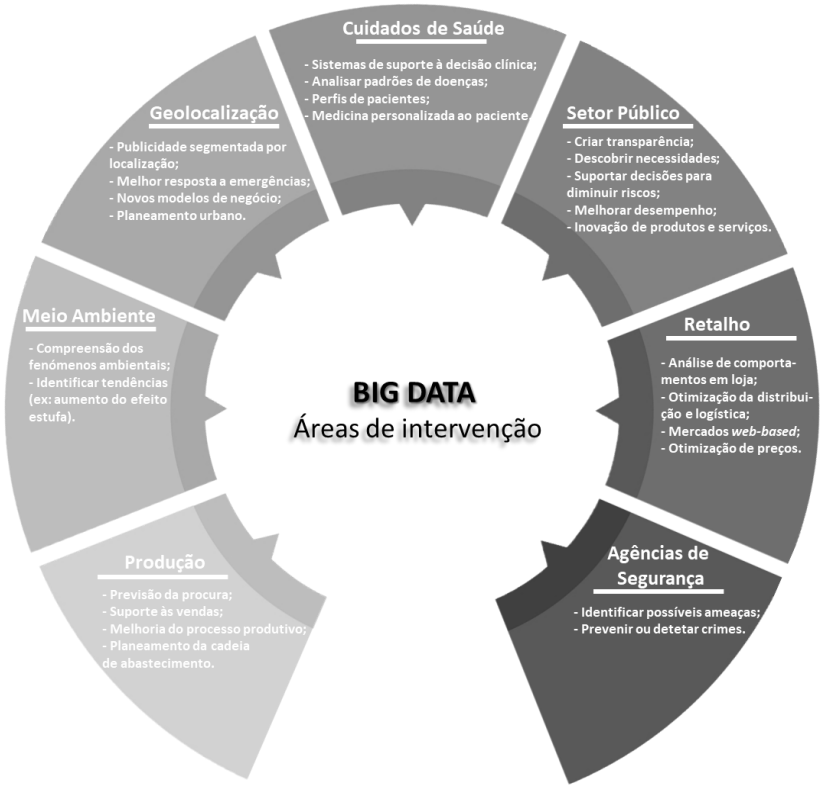


Figura 10 - Áreas de Intervenção do Big Data. Baseado em (Chandarana & Vijayalakshmi, 2014; Manyika et al., 2011; Sagiroglu & Sinanc, 2013).

2.1.3 Desafios e Dilemas do *Big Data*

O *Big Data* proporciona muitas oportunidades e áreas de intervenção, como se pode constatar na subsecção anterior, contudo, este volume de dados cada vez maior na era do *Big Data* acarreta também vários dilemas (M. Chen et al., 2014).

Assim, de seguida serão expostos os vários desafios que a literatura estudada associa ao *Big Data*. De referir que os desafios serão divididos em quatro categorias, baseadas em (C. Costa & Santos, 2017), e que se julgam representativas dos problemas identificados. Em cada uma das categorias ("Desafios Gerais associados ao *Big Data"*, "Desafios Técnicos no Ciclo de Vida do *Big Data"*, "*Big Data* em Ambientes Seguros, Privados e Monitorizados" e "Mudança Organizacional") será feita uma breve

descrição da mesma e apresentada uma tabela que especifica os desafios identificados (coluna "Desafio") e exibe algumas considerações sobre os mesmos (coluna "Descrição").

Desafios Gerais associados ao *Big Data*

Esta categoria incluí desafios de índole mais geral, isto é, que não estão intimamente ligados com as dificuldades que se enfrentam no desenvolvimento de sistemas *Big Data*. Inclui, portanto, preocupações relacionadas a construções teóricas (exemplo: definição de Big Data) ou decisões de mais alto nível (exemplo: debate sobre qualidade *versus* quantidade). A Tabela 1 sintetiza os desafios identificados na literatura e que se considera que se enquadram nesta categoria.

Tabela 1 - Desafios Gerais associados ao Big Data.

Desafio	Descrição
Definição de *Big Data*	É necessária uma definição consensual, rigorosa e holística de *Big Data*, que inclua um modelo estrutural e uma descrição formal, assim como um sistema teórico de *Data Science*. Na sua falta assiste-se a muitas discussões que mais parecem especulação comercial, do que investigação científica (M. Chen et al., 2014).
Falta de *Standards* em *Big Data*	Muitas soluções de *Big Data* argumentam que conseguem melhorar o processamento e análise de dados, contudo, não existe nenhum *standard* de avaliação unificado ou um *benchmark* capaz de comparar a eficiência dessas soluções, com métodos rigorosos. Apenas se consegue saber o desempenho delas à posteriori, isto é, depois de implementar e implantar o sistema (M. Chen et al., 2014). Esta falta de *standards* é ainda agravada pela constante evolução tecnológica no ambiente *Big Data* (Baru, Bhandarkar, Nambiar, Poess, & Rabl, 2013; C. Costa & Santos, 2017).
Quantidade *versus* Qualidade	Segundo Kaisler, Armour, Espinosa, e Money (2013), tem havido o hábito dos utilizadores recolherem o máximo de dados possível, talvez por acreditarem que mais dados se traduzem numa explicação perfeita do fenómeno em estudo. Porém, tudo varia de acordo com o contexto, pelo que, não se deve assumir que recolher mais dados é sempre o melhor. Aliás, se forem armazenadas grandes quantidades de dados que sejam irrelevantes ou não tenham qualidade suficiente as conclusões a retirar dos mesmos estarão comprometidas (Kaisler et al., 2013; Katal, Wazid, & Goudar, 2013). Assim, decidir quais os dados relevantes, quais as fontes de dados mais apropriadas e quando é que os dados têm a qualidade suficiente para suportar decisões e acrescentar valor, permanecem como desafio (Chandarana & Vijayalakshmi, 2014; C. Costa & Santos, 2017; Kaisler et al., 2013; Katal et al., 2013).

Desafios Técnicos no Ciclo de Vida do *Big Data*

Incluem-se nesta categoria os desafios que advêm de dificuldades técnicas, em tarefas como por exemplo recolha, limpeza, transformação e integração dos dados, assim como no seu armazenamento, processamento ou governança. Os desafios identificados na literatura como pertencentes a esta categoria, apresentam-se na Tabela 2.

Tabela 2 - Desafios Técnicos no Ciclo de Vida do Big Data.

Desafio	Descrição
Armazenamento e Processamento de Dados	Os dados estão a ser criados por várias pessoas, vários serviços e a qualquer momento, de tal forma que as tecnologias tradicionais e as redes de comunicação já não conseguem lidar com as quantidades de dados geradas diariamente (Kaisler et al., 2013). Torna-se crucial a escalabilidade, repensando armazenamento e algoritmos, de forma a extrair valor do *Big Data* (C. L. P. Chen & Zhang, 2014; C. Costa & Santos, 2017; Hashem et al., 2015). O processamento paralelo também é de supra importância para garantir disponibilidade, eficiência e elasticidade dos dados, caso contrário, processar *Big Data* implicaria grandes quantidades do tempo e, consecutivamente, os dados adequados não estriam disponíveis a tempo de suportar a tomada de decisão (M. Chen et al., 2014; C. Costa & Santos, 2017; Kaisler et al., 2013; Katal et al., 2013).
Garantir a Qualidade dos Dados	Garantir a qualidade dos dados e adicionar valor através da preparação dos mesmos torna-se desafiante em *Big Data* (C. L. P. Chen & Zhang, 2014; C. Costa & Santos, 2017). Aliás, por vezes, quando se lida com dados que fluem a grande velocidade, não se pode despender tempo a fazer a limpeza dos mesmos, antes do seu consumo (Chandarana & Vijayalakshmi, 2014). Como consequência podem surgir problemas distintos, assim, têm de ser definidas estratégias para os tratar, porque, tal como argumenta Khan et al (2014), não se podem tomar decisões, tendo por base um conjunto de dados que não são confiáveis. No sentido de combater este desafio, M. Chen et al. (2014) argumentam ser necessária a definição de métodos que automaticamente detetem e corrijam alguns problemas com os dados.
Heterogeneidade dos Dados	Devido às distintas fontes de dados utilizadas, a heterogeneidade dos dados é uma característica sempre presente no *Big Data* (M. Chen et al., 2014). Esta característica é ainda mais desafiante, porque traz implicações na qualidade e integração dos dados, visto que, as técnicas tradicionais de análise de dados esperam dados homogéneos (C. Costa & Santos, 2017). Além disso, a natureza não estruturada destes dados acarreta grandes desafios de transformações, de forma a suportarem as tarefas analíticas (C. Costa & Santos, 2017). Os dados em *stream*, por exemplo, podem alterar dinamicamente o seu formato, obrigando a alterações no seu processamento (Kaisler et al., 2013). Trabalhar com dados não estruturados é altamente custoso, porque se trata de dados completamente desorganizados e em bruto (Katal et al., 2013).
Cooperação entre os intervenientes	Segundo M. Chen et al. (2014), a análise em *Big Data* é uma investigação interdisciplinar, que requer especialistas de diferentes áreas a cooperar para alcançar o máximo do potencial do *Big Data*. O mesmo autor, argumenta que é necessário criar uma arquitetura em rede, para garantir que todas as condições estão ao dispor dos intervenientes e ajudá-los a otimizar as suas capacidades ao ponto de alcançar os objetivos. Esta rede auxiliaria os intervenientes, por exemplo, a utilizar e perceber diferentes tipos de dados, de áreas diferentes da sua, mas que são dominadas por outro interveniente que está a cooperar consigo (M. Chen et al., 2014).
Visualização em *Big Data*	Em *Big Data*, devido às características dos dados, é necessário repensar as abordagens tradicionais, a fim de juntar aparência e funcionalidade (C. L. P. Chen & Zhang, 2014). Para extrair valor do *Big Data* são necessárias visualizações que consigam transmitir dados provenientes de várias fontes distintas, de forma interativa e fácil de utilizar, de tal forma que os utilizadores fiquem satisfeitos e utilizem efetivamente as visualizações (M. Chen et al., 2014; C. Costa & Santos, 2017; Krishnan, 2013).

Desafio	Descrição
Governança dos Dados	Em ambientes *Big Data* é muito complexo fazer a governança dos massivos volumes de dados, provenientes de fontes distintas, com diferentes propósitos, diferente público-alvo e consequentemente diferentes restrições (Hashem et al., 2015). Gerir um ambiente tão heterogéneo e definir políticas de acesso para todos os cenários pode tornar-se impossível sem as ferramentas adequadas (C. Costa & Santos, 2017).

Big Data em Ambientes Seguros, Privados e Monitorizados

Nos dias de hoje a privacidade e segurança dos dados costuma ser uma das temáticas que mais preocupam as organizações (M. Chen et al., 2014). Assim, aglomeram-se nesta categoria os desafios que resultam das preocupações de manter os dados seguros e privados, envolvendo a prevenção de riscos e a proteção contra ataques. Os desafios que se inserem nesta problemática encontram-se resumidos na Tabela 3.

Tabela 3 - Big Data em Ambientes Seguros, Privados e Monitorizados.

Desafio	Descrição
Planos de Prevenção e Mitigação de Riscos	Os utilizadores querem ter a garantia de que os seus dados não sofrem *leaks*, isto é, não vêm parar ao domínio público (Chandarana & Vijayalakshmi, 2014). Porém, devido às características do *Big Data*, surgem mais riscos, causando que os métodos de proteção tradicionais tenham de ser repensados. Além disso, por vezes, as organizações não têm conhecimento para lidar com *Big Data*, subcontratando outras que ficam com acesso a dados sensíveis. Assim, é relevante planear um modelo de segurança para *Big Data*, para as organizações especificarem os seus riscos e prevenirem atividade ilegal ou ciberataques (M. Chen et al., 2014; C. Costa & Santos, 2017). Outras preocupações, como por exemplo a integridade (os dados são apenas modificados pelo dono e entidades autorizadas) e a definição de políticas de acesso são também destacadas (Hashem et al., 2015; Manyika et al., 2011).
Propriedade dos dados	Este é um desafio crítico, particularmente na área das redes sociais. Os servidores destas redes sociais guardam grandes volumes de dados que, na verdade, não são propriedade dessas empresas, embora elas atuam como se fosse e a legislação atual tenda a beneficiá-las. Os verdadeiros proprietários dos dados são as pessoas que criaram as páginas ou contas. Existem ainda discussões, para encontrar formas de partilhar e vender dados sem perder o controlo sobre eles (Chandarana & Vijayalakshmi, 2014; Kaisler et al., 2013).

Desafio	Descrição
Legislação	Kaisler et al. (2013) referem que a IDC aplicou o termo "sombra digital" para se referir à informação acerca de um individuo que é tratada para formar uma "imagem" do mesmo. O problema é quanta desta informação, queremos manter privada. Toda esta informação sensível disponível, quando combinada com outras fontes externas leva à inferência de factos sobre a pessoa, por vezes altamente intrusivos, como por exemplo a habituação, o local de trabalho, entre outros. Posto isto, claramente, o *Big Data* deve de ser assegurado por leis e regulamentação quanto à privacidade e segurança (C. Costa & Santos, 2017; Kaisler et al., 2013; Katal et al., 2013; Manyika et al., 2011). Kaisler et al. (2013) colocam algumas questões em aberto: os dados sobre um individuo podem ser guardados num único repositório ou em repositórios distintos? ou nem podem ser recolhidos? Deve-se definir uma alta agregação de dados para não ser possível inferir sobre o indivíduo?

Mudança Organizacional

Nesta categoria, considera-se que se enquadram os problemas/desafios relacionados com a implementação, adoção e utilização efetiva do *Big Data* nas organizações. Estes desafios podem ser, por exemplo, consequência do desconhecimento do valor do *Big Data*, por parte da organização. Apresentam-se na Tabela 4 os desafios relacionados com esta categoria.

Tabela 4 - Mudança Organizacional.

Desafio	Descrição
Transmitir o Valor do *Big Data*	Apesar do *Big Data* soar apelativo para grande parte das organizações, frequentemente, os líderes organizacionais não percebem ou não dão a devida importância ao seu valor e a como o extrair (Manyika et al., 2011). Em muitas áreas as organizações precisam de monitorizar tendências e ganhar vantagem sobre os seus competidores. Porém, para tal, é necessário que os líderes percebam como é que o *Big Data* pode acrescentar valor, que encontrem recursos humanos capacitados para lidar com o fenómeno e que os motivem e conduzam iniciativas a favor do *Big Data* como suporte à decisão (C. Costa & Santos, 2017; Manyika et al., 2011).
Conduzir Mudança Organizacional	Conduzir a mudança organizacional para que se implemente e utilize efetivamente *Big Data*, é importante e acarreta muitos desafios, tais como: a necessidade de uma liderança adequada, e de profissionais para lidar com *Big Data*; a compreensão e utilização das tecnologias de *Big Data*; e a necessidade de mudar a cultura organizacional. Além disso, as iniciativas em *Big Data* requerem abordagens multidisciplinares, com colaboração entre uma equipa que inclua profissionais com diferentes valências, a fim de atingir os objetivos e entregar resultados úteis para a organização (M. Chen et al., 2014; C. Costa & Santos, 2017).

Por análise das quatro tabelas apresentadas nesta subsecção, que sumarizam os principais desafios associados ao *Big Data*, encontrados na literatura, verifica-se que conceber um sistema capaz de suportar análises e extrair valor, de forma eficaz e eficiente, de dados com as características de *Big Data*, acarreta enormes dilemas.

De entre os desafios associados ao *Big Data*, vários autores referem que a segurança e privacidade são aqueles que mais preocupam as organizações, incluindo questões concetuais, técnicas e legais (M. Chen et al., 2014; C. Costa & Santos, 2017; Kaisler et al., 2013; Katal et al., 2013).

De realçar que os desafios apresentados são também relevantes para identificar tópicos de investigação nesta área (C. Costa & Santos, 2017).

2.1.4 Processamento: Do Tradicional ao *Big Data*

O processamento tradicional de dados, segundo Krishnan (2013), pode ser definido como a recolha, o processamento e gestão de dados e que tem como resultado a extração de informação para os consumidores finais. Os dados recolhidos são por natureza estruturados e discretos em volume, sendo que todo o processo é pré-definido, baseado nos requisitos conhecidos e, tarefas como qualidade e limpeza dos dados não se revelam tão problemáticas, quando comparadas com contextos *Big Data*. O ciclo de processamento tradicional de dados é ilustrado na Figura 11.

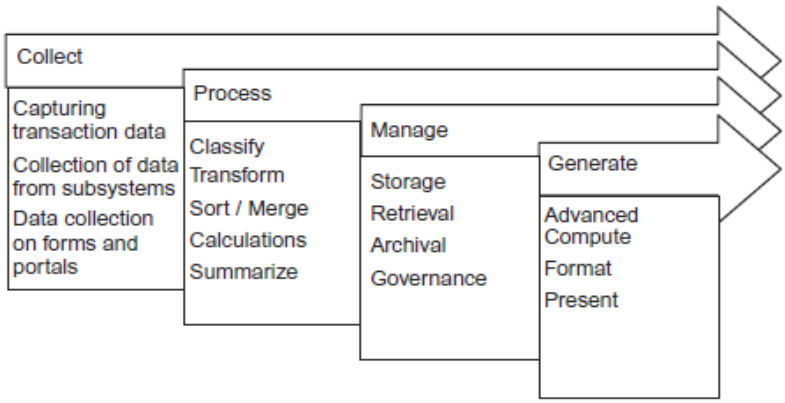


Figura 11 - Ciclo de processamento tradicional de dados. Retirado de (Krishnan, 2013).

O processamento de *Big Data* (Figura 12) difere do tradicional (Figura 11). No processamento tradicional, primeiro exploram-se os dados e criam-se um conjunto de requisitos analíticos, que levam à *data discovery* e à criação de modelos de dados, seguidos da estruturação da base de dados para carregar os dados. Por sua vez, no processamento de *Big Data*, os dados são primeiramente recolhidos e armazenados numa plataforma, de seguida, são aplicados os metadados e a estrutura dos dados é criada. A partir do momento em que os dados possuem estrutura, já podem então ser transformados e analisados (Krishnan, 2013).

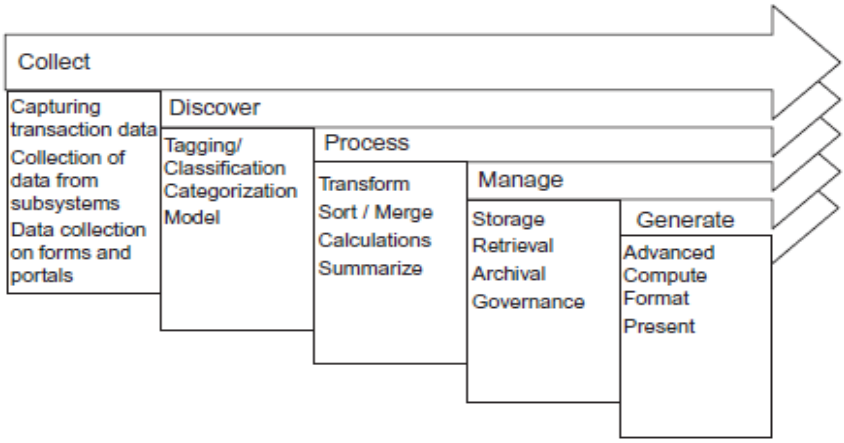


Figura 12 - Ciclo de processamento Big Data. Retirado de (Krishnan, 2013).

Para processar os dados de uma forma flexível, segundo Krishnan (2013), uma arquitetura orientada a bases de dados não iriar obter bom desempenho. Segundo o autor, para processar *Big Data*, uma arquitetura orientada a ficheiros é mais adequada. Porém, para conceber uma arquitetura de processamento eficiente, necessitamos de perceber o fluxo de processamento de *Big Data*, por isso, Krishnan (2013) propõe uma visão geral de alto nível do fluxo de processamento de *Big Data*, assente em 4 fases, ilustradas na Figura 13 e brevemente descritas de seguida, com base no autor.

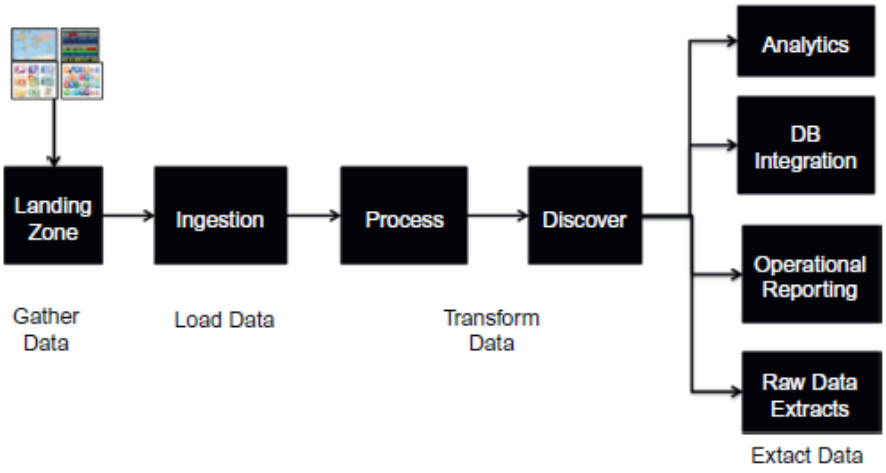


Figura 13 - Fluxo de processamento de Big Data. Retirado de (Krishnan, 2013).

- **Recolha:** os dados são recebidos de diferentes fontes e carregados para o sistema de ficheiros designado por *landing zone*.
- **Carregamento:** os dados são carregados com a aplicação dos metadados, já com alguma estrutura e preparados para a fase de transformação. O processo de carregamento divide o

volume de dados original em arquivos mais pequenos. É também nesta fase que se podem definir partições horizontais ou verticais.

- **Transformação:** os dados são tratados, aplicando-se as regras de negócio e processando o seu conteúdo. Tipicamente, esta fase acarreta vários passos e pode tornar-se complexa de gerir.
- **Extração:** os dados resultantes das fases anteriores podem ser extraídos para fins de análise, relatórios operacionais, integração com um *data warehouse* e para propósitos de visualização.

Apresentadas as fases principais do fluxo de processamento de *Big Data*, que possibilitam processar os dados de uma forma flexível, importa também saber quais são os tipos de processamento de dados existentes. De acordo com Du (2015), o processamento de dados pode ser de três tipos:

- **Processamento em *batch*:** é utilizado para processar dados em lotes. Os dados são lidos da fonte de dados, processados e, por fim, escritos na fonte de destino.
- **Processamento em *real-time*:** consiste em processar os dados e obter os resultados quase de imediato.
- **Processamento em *stream*:** os dados são continuamente processados e atua-se sobre os mesmos em *live stream* para obter os resultados.

2.2 Armazenamento de Dados

O armazenamento de dados é uma das temáticas importantes na era do *Big Data*, tal como foi sendo percetível ao longo da secção 2.1. Posto isto, nesta secção será apresentado um enquadramento sobre os principais conceitos que dizem respeito a esta temática. Iniciar-se-á com uma comparação entre as tradicionais bases de dados SQL, as bases de dados NoSQL e as bases de dados NewSQL, que tentam agregar as vantagens das duas primeiras. Seguir-se-á o conceito de *Data Warehouse* e a distinção entre os sistemas analíticos (OLAP), que costumam suportar os *Data Warehouses* e os sistemas operativos (OLTP), que tipicamente sustentam as bases de dados operacionais. Por fim, apresentar-se-á o conceito de *Big Data Warehouse*, que evolui do tradicional *Data Warehouse* e terminar-se-á abordando os modelos de dados necessários para assegurar o adequado armazenamento e consulta dos dados.

2.2.1 Bases de Dados SQL, NoSQL e NewSQL

As bases de dados relacionais tradicionais, utilizam o SQL (*Structured Query Language*) como linguagem *standard* e armazenam os dados de acordo com as suas relações, em tabelas constituídas

por linhas e colunas. Além disso, este tipo de bases de dados implica a utilização de vários tipos de chaves, sendo a chave primária uma das mais importantes, identificando univocamente cada linha da tabela. As propriedades ACID (*Atomicity, Consistency, Isolation, Durability*) constituem uma relevante propriedade das bases de dados relacionais (Fatima & Wasnik, 2017).

Durante várias décadas, estes sistemas de bases de dados relacionais alcançaram um grande nível de confiabilidade e estabilidade, constituindo-se como poderosos mecanismos de armazenamento e consulta de dados estruturados, com garantia de consistência (Gessert, Wingerath, Friedrich, & Ritter, 2017). Porém, ao longo dos últimos anos tem existido um aumento notável das quantidades de dados geradas, de tal forma que se exige um processamento de elevada velocidade, esquemas flexíveis e bases de dados distribuídas. Por consequência, as bases de dados relacionais tradicionais, não conseguem lidar com os requisitos de escalabilidade exigidos e, muitas vezes falham ao tentar fazê-lo (Fatima & Wasnik, 2017; Gessert et al., 2017; Krishnan, 2013; Vaish, 2013).

Assim, para lidar com as exigências colocadas pela massiva quantidade de dados, gerada a taxas que aumentam drasticamente, nasceram duas novas classes de bases de dados, conhecidas como NoSQL e NewSQL (Fatima & Wasnik, 2017). Contudo, nenhuma destas soluções de bases de dados que foram surgindo (SQL, NoSQL, NewSQL) é perfeita ou surge para substituir as existentes. Posto isto, conhecer as opções à disposição, as suas vantagens e desvantagens e os cenários para os quais são mais ou menos recomendáveis é crítico para tomar uma decisão informada (Vaish, 2013). Leavitt (2010) concorda com esta visão, referindo que as pessoas irão aprender a olhar para os seus dados e selecionar as bases de dados que melhor se adequam às suas necessidades.

As bases de dados NoSQL são a nova abordagem para a conceção de bases de dados capazes de lidar com vastas quantidades de dados distribuídos (C. L. P. Chen & Zhang, 2014). A motivação para o aparecimento e crescimento da notoriedade destas bases de dados, surge da falta de escalabilidade e do facto dos modelos relacionais serem inadequados para lidar com *Big Data* (C. Costa & Santos, 2017).

O NoSQL, segundo Vaish (2013), embora seja também tratado por "Not Only SQL", foi pensado originalmente como a combinação das duas palavras "No" e "SQL", o que significa "Não querer utilizar SQL" ou "querer aceder às bases de dados sem utilizar qualquer sintaxe SQL". Por outro lado, C. L. P. Chen e Zhang (2014) consideram que apesar do termo NoSQL poder gerar más interpretações este não evita por completo o SQL, dado que existem sistemas que apenas evitam algumas funcionalidades, como por exemplo esquemas de tabelas com formatos fixos ou *joins*.

Vaish (2013) prossegue clarificando que NoSQL não é uma base de dados, nem um tipo de base de dados, é sim um termo genérico para referir todas as bases de dados que não seguem os modelos

relacionais tradicionais e que tentam resolver os problemas de escalabilidade e disponibilidade contra a atomicidade ou consistência. Já Leavitt (2010) é mais sucinto, considerando que as bases de dados NoSQL, seguem diferentes abordagens, por isso, a única coisa que têm em comum é o facto de não serem relacionais.

Estas bases de dados possuem algumas características principais e vantagens face às tradicionais que são apresentadas de seguida (Cattell, 2010; Leavitt, 2010):

- Facilidade para trabalhar por parte dos *developers* que não estão acostumados ao SQL;
- Replicar e particionar dados em muitos servidores;
- Escalabilidade horizontal, que envolve menos custos do que possuir apenas uma máquina com grande poder computacional;
- Grande parte das bases de dados NoSQL são *open-source*, envolvendo baixos custos;
- Maior velocidade de processamento;
- Evitar complexidade desnecessária, não suportando propriedades ACID;
- Dinamicamente adicionar novos atributos aos registos;
- Simplicidade e flexibilidade dos modelos.

Porém, também existem desvantagens e Leavitt (2010) identifica as seguintes:

- **Complexidade** - o facto de não trabalhar com SQL requer programação, o que pode ser complexo e consumir muito tempo;
- **Não fiabilidade** - a exclusão do ACID retira uma camada de fiabilidade;
- **Não consistência** - também pela exclusão do ACID. Permite melhorar desempenho e escalabilidade, mas pode ser crítico para alguns cenários, como aqueles que envolvem transações bancárias;
- **Desconhecimento da tecnologia** - muitas organizações não estão familiarizadas com o NoSQL, por isso, podem sentir que não têm o conhecimento suficiente para o utilizar;
- **Falta de suporte** - ao contrário das bases de dados relacionais tradicionais, muitas soluções NoSQL não possuem suporte ao cliente ou ferramentas de gestão.

Em suma, as propriedades ACID são uma das diferenças mais marcantes, entre o SQL e o NoSQL, visto que, o primeiro tem um especial foco neste conceito, enquanto o segundo o sacrifica. Posto isto, alguns autores sugerem o BASE (*Basic Availability, Soft state, Eventual consistency*) em detrimento do ACID, de forma a alcançar melhor desempenho, escalabilidade e disponibilidade, ao invés de

consistência e serialização (Cattell, 2010; Fatima & Wasnik, 2017; Gessert et al., 2017; Vaish, 2013). A Tabela 5 apresenta as propriedades ACID e BASE segundo (Vaish, 2013).

Tabela 5 - Propriedades ACID e propriedades BASE.

ACID (*Atomicity, Consistency, Isolation, Durability*)	**BASE** (*Basic Availability, Soft state, Eventual consistency*)
Atomicidade: toda a transação tem de ser bem-sucedida, caso contrário é revertida	Disponibilidade básica: cada pedido tem garantidamente uma resposta
Consistência: uma transação não pode deixar a base de dados num estado inconsistente	Estado flexível: o estado do sistema pode mudar ao longo do tempo
Isolamento: uma transação não pode interferir com outra	Eventual consistência: a base de dados pode estar momentaneamente inconsistente, mas será eventualmente consistente
Durabilidade: uma transação completa persiste, mesmo depois de reiniciar a aplicação	

Associados a estes dois conceitos mencionados acima surge, frequentemente, o teorema CAP (*Consistency, Availability, Partition tolerance*), o qual afirma que qualquer sistema de partilha de dados em rede, apenas pode ter, simultaneamente, duas de três propriedades desejadas (Brewer, 2012). A Figura 14, sintetiza as propriedades do teorema CAP e classifica várias bases de dados NoSQL segundo as duas propriedades a que conseguem responder, baseando-se nos contributos de Brewer (2012) e Han, Haihong, Le, e Du (2011).

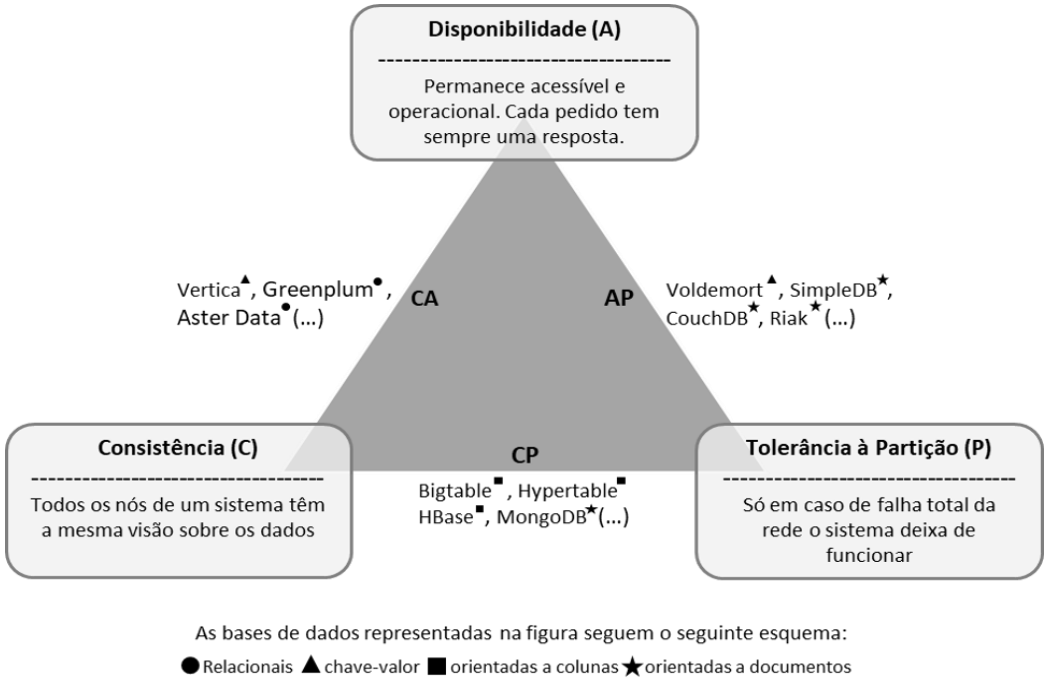


Figura 14 - Teorema de CAP e classificação de algumas bases de dados NoSQL segundo o mesmo.

Por análise da Figura 14, verifica-se que as bases de dados relacionais e as bases de dados NoSQL que incluem algumas das características das relacionais, conseguem garantir consistência e

disponibilidade. Por outro lado, se o objetivo for garantir tolerância à partição, as bases de dados NoSQL terão de abdicar da consistência ou da disponibilidade.

Existe um grande número de bases de dados NoSQL, de tal forma que enumerar e avaliar cada uma delas é uma tarefa quase impossível (Edlich, 2018). Tendo isto em consideração, estas bases de dados são frequentemente divididas em quatro tipos diferentes, de acordo com o seu modelo de dados, ilustradas na Figura 15 e descritas de seguida (M. Chen et al., 2014; C. Costa & Santos, 2017; Grolinger, Higashino, Tiwari, & Capretz, 2013; Krishnan, 2013; Siddiqa, Karim, & Gani, 2017).

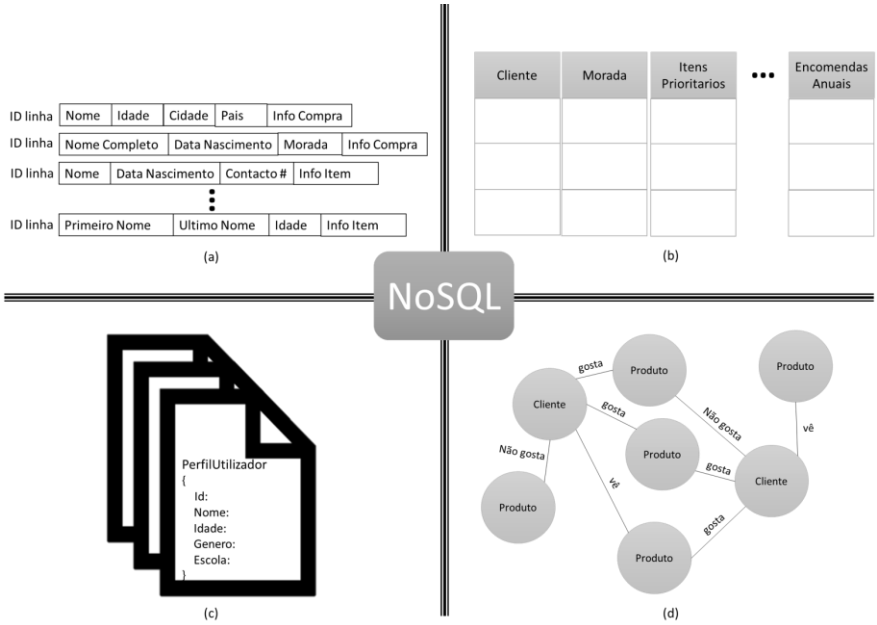


Figura 15 - Tipos de bases de dados NoSQL: (a) chave-valor; (b) orientada a colunas; (c) orientada a documentos; (d) grafos. Adaptada de (Siddiqa et al., 2017).

- **Bases de dados chave-valor** –utilizam uma tabela de *hash*, na qual existe uma chave única e um apontador para um determinado valor de tipo arbitrário. Este modelo não possui esquema e é muito eficiente a armazenar dados distribuídos de tipo arbitrário. No entanto, quando são necessárias operações mais complexas sobre os dados ou qualquer cenário exigindo relações, este modelo já não é apropriado. Além disso, os valores armazenados são opacos, isto é, as *queries* só podem ser feitas através das chaves. Exemplos: Voldemort[3] e Dynamo[4].

[3] http://www.project-voldemort.com/voldemort/
[4] https://aws.amazon.com/dynamodb/

- **Bases de dados orientadas a colunas** – pode ser visto como uma extensão do modelo chave-valor, com a adição de colunas e famílias de colunas. Com esta adição, conseguem-se otimizar as consultas aos dados. Estas bases de dados são muito inspiradas na Bigtable[5] da Google. Exemplos: Bigtable e HBase[6].
- **Bases de dados orientadas a documentos** – armazenam dados em formatos de documentos, frequentemente representados em JSON (*JavaScript Object Notation*), capazes de conter dados com estruturas complexas. São mais uma extensão do modelo chave-valor, utilizando chaves para localizar os documentos guardados. O esquema destas bases de dados é flexível e permite indexar os documentos pela chave ou pelos valores dos documentos. Exemplos: MongoDB[7] e CouchDB[8].
- **Bases de dados de grafos** – baseiam-se na teoria de grafos, na qual os objetos podem ser representados como vértices e as relações entre os mesmos por arestas. São especializados na gestão de dados altamente relacionados, sendo, portanto, muito eficientes em relacionamentos entre diferentes entidades. Estas bases de dados são adequadas a cenários que usam dados de redes sociais para reconhecimento de padrões e sistemas de recomendação. Exemplos: Neo4J[9], GraphDB[10].

Tal como foi referido anteriormente, para além do NoSQL surgiu mais recentemente o NewSQL. Este termo refere uma classe de sistemas de gestão de bases de dados relacionais modernos e muito promissores, que utilizam o SQL como linguagem e que são capazes de igualar o desempenho escalável dos sistemas NoSQL para OLTP (*Online Transaction Processing*), ao mesmo tempo que mantêm as propriedades ACID das bases de dados tradicionais. NuoDB[11] e VoltDB[12] são exemplos de bases de dados deste tipo (C. Costa & Santos, 2017; Cuzzocrea, Song, & Davis, 2011; Fatima & Wasnik, 2017; Kaur & Sachdeva, 2017).

Tendo em conta que foco desta dissertação recai sobre o Druid, uma tecnologia NoSQL cujo armazenamento é orientado a colunas, não se considera relevante alongar a discussão sobre os

[5] https://cloud.google.com/bigtable/
[6] https://hbase.apache.org/
[7] https://www.mongodb.com/
[8] http://couchdb.apache.org/
[9] https://neo4j.com/
[10] http://graphdb.ontotext.com/
[11] https://www.nuodb.com/
[12] https://www.voltdb.com/

diferentes tipos de bases de dados NoSQL e NewSQL. No capítulo 3, neste documento, será então efetuada uma descrição mais pormenorizada sobre o Druid.

2.2.2 Sistemas de *Data Warehousing*

Até meados dos anos 80, as bases de dados armazenavam apenas dados operacionais, ou seja, dados criados pelas operações de negócio incluídas nos processos diários de gestão (compras, vendas...) (Golfarelli & Rizzi, 2009). Porém, desde o início dos anos 90 que as organizações sentem a necessidade de realizar análises de dados mais sofisticadas, de forma a auxiliar o processo de tomada de decisão e fazer frente à crescente competitividade do mercado. Contudo, as bases de dados operacionais não satisfaziam estes requisitos de análise dos dados. Assim sendo, os *Data Warehouses* surgiram para suprir as necessidades analíticas (Vaisman & Zimnyi, 2014).

Segundo Sá (2009), o termo *Data Warehouse* foi aplicado pela primeira vez por Inmon (1992), sendo que existia desde 1988, o termo "*Information Warehouse*" que se pode considerar como o antecessor do termo *Data Warehouse* e que foi atribuído por investigadores da IBM.

Inmon (2005) define *Data Warehouse* como uma coleção de registos orientados por assunto, integrados, não voláteis, variáveis ao longo do tempo e que possui o objetivo de suportar a tomada de decisão. Esta definição sistematiza um conjunto de características, nomeadamente:

- **Orientados por assunto** – o *Data Warehouse* foca-se nas necessidades analíticas (assuntos) das diferentes áreas da organização (exemplo: clientes, fornecedores), sendo que estas necessidade variam de organização para organização.
- **Integrados** – é dos aspetos mais importantes. Os dados selecionados e armazenados no *Data Warehouse* advêm de diversas fontes heterogéneas e são integrados e consolidados de forma a conceber uma visão coerente. Implica o tratamento dos dados, de forma a resolver qualquer tipo de incoerência.
- **Não voláteis** – os dados no *Data Warehouse* são estáveis, isto é, não se realizam continuamente operações sobre os registos, tais como inserções, atualizações e eliminações. Depois dos dados serem carregados para o *Data Warehouse*, não podem ser alterados ou eliminados, apenas se podem ir fazendo refrescamentos periodicamente para acrescentar os dados entretanto acumulados nas bases de dados operacionais ou realizar atualizações evolutivas através de SCD (*slowly changing dimensions*).

- **Variáveis ao longo do tempo** – os *Data Warehouses* fornecem informação sob uma perspetiva histórica, como tal, os dados têm de possuir uma dimensão temporal. Este tipo de sistemas armazena dados respeitantes a um período que pode variar entre 5 e 10 anos, enquanto os sistemas operacionais armazenam dados com um horizonte temporal relativamente curto.

Noutra perspetiva, Kimball (1996) define *Data Warehouse* como uma cópia dos registos transacionais, especialmente estruturados para responder a interrogações e análises. M. Y. Santos e Ramos (2006) consideram que um *Data Warehouse* é um repositório construído especificamente para a consolidação da informação num formato válido e consistente, permitindo aos seus utilizadores a análise de dados de uma forma seletiva.

Jiawei Han, Kamber, e Pei (2011) referem que o facto de vários autores tentarem definir o termo *Data Warehouse* de tantas formas diferentes, dificulta a formulação de uma definição rigorosa. Contudo, os autores defendem uma visão que tem vários pontos em comum com as anteriores, referindo-se ao *Data Warehouse* como um repositório que é mantido separadamente dos sistemas operacionais da organização, que integra informação proveniente de diversas fontes e que disponibiliza uma plataforma sólida de dados históricos consolidados para análise e auxílio do processo de tomada de decisão.

Associado a *Data Warehouse*, surge muitas vezes o termo *Data Warehousing* ou sistemas de *Data Warehouse* o que pode gerar alguma confusão. Importa então clarificar que *Data Warehousing* diz respeito aos métodos, técnicas e ferramentas utilizadas para suportar os "trabalhadores do conhecimento" (gestores, diretores, analistas) a conduzir análises aos dados que ajudem a melhorar o processo de tomada de decisão, assim como, os recursos informacionais (Golfarelli & Rizzi, 2009).

Existem na literatura diferentes propostas de arquitetura para *Data Warehouses*. Vaisman e Zimnyi (2014) apresentam uma arquitetura genérica, ilustrada na Figura 16, que se divide em quatro camadas principais:

- **Camada *Back-End*** – é composta pelas ferramentas de ETL (*Extraction*, *Transformation* and *Loading*) utilizadas para extrair dados das bases de dados operacionais e outras fontes (internas ou externas à organização), transformá-los (limpeza, correção de erros, tratamento de *nulls*...) e carregá-los para alimentar o *Data Warehouse*. Inclui também a área de *Data Staging*, que é uma base de dados intermédia, na qual os processos de extração e transformação dos dados ocorrem, antes do carregamento final para o *Data Warehouse*.

- **Camada *Data Warehouse*** – inclui o *Data Warehouse* organizacional e/ou *Data Marts* e repositórios de metadados que guardam informação sobre o *Data Warehouse* e o seu conteúdo.
- **Camada OLAP** – é constituída pelo servidor OLAP, que fornece uma visão multidimensional dos dados presentes no *Data Warehouse* ou nos *Data Marts*.
- **Camada *Front-End*** – inclui ferramentas que fornecem aos utilizadores um conjunto de capacidades analíticas para explorar e analisar os dados e que são úteis para auxiliar a tomada de decisão. Estas ferramentas incluem: OLAP, *reporting*, análises estatísticas, ou *Data Mining*.

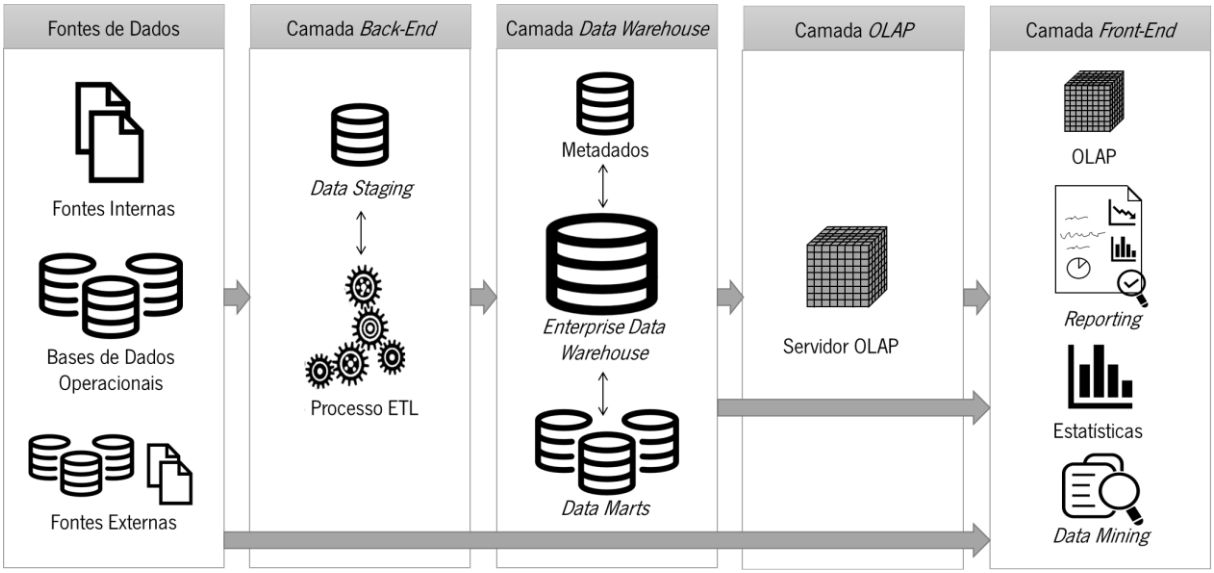


Figura 16 - Arquitetura típica de um Data Warehouse. Adaptado de (Vaisman & Zimnyi, 2014).

Tal como referido anteriormente existem várias propostas de arquitetura, pelo que, podem ocorrer casos em que alguns dos componentes presentes na Figura 16 não sejam utilizados. Há situações em que existe apenas o *Data Warehouse* organizacional, sem qualquer *Data Mart*, ou situações em que não existe *Data Warehouse* organizacional. Construir um *Data Warehouse* organizacional é uma tarefa complexa e custosa, enquanto que os *Data Marts* são mais fáceis e rápidos de implementar. Contudo, esta vantagem pode tornar-se um problema, no caso dos vários *Data Marts* independentes precisarem de ser integrados num *Data Warehouse* para toda a organização (Vaisman & Zimnyi, 2014).

Em alguns casos, não existe um servidor OLAP e as ferramentas da camada *Front-End* acedem diretamente ao *Data Warehouse* (representado pela seta que liga as camadas *Data Warehouse* e *Front-End*). Pode verificar-se ainda a situação em que não existe nem servidor OLAP, nem *Data Warehouse*, mas sim um *Data Warehouse* Virtual, que consiste num conjunto de vistas materializadas ligadas diretamente às bases de dados operacionais (representado pela seta que liga as Fontes de Dados à camada *Font-End*). Este tipo de *Data Warehouse* é fácil de implementar, mas não representa uma solução

real de *Data Warehouse*, uma vez que não possui dados históricos, metadados centralizados, nem a capacidade de fazer a limpeza e transformação dos dados. Além disso, estes sistemas podem causar um impacto severo no desempenho das bases de dados operacionais (Vaisman & Zimnyi, 2014).

Na Figura 16 e subsequentes parágrafos, verifica-se também o aparecimento do termo *Data Mart*, sendo por isso relevante descrevê-lo. Segundo Jiawei Han et al. (2011), um *Data Mart* é um subconjunto dos dados de toda a empresa, que possui valor para um grupo específico de utilizadores. Por exemplo, um *Data Mart* de marketing, provavelmente incluirá assuntos tais como: clientes, produtos e vendas. Ou seja, enquanto um *Data Warehouse* é organizacional, isto é, tem o objetivo de ser o repositório que permite a análise de toda a informação da organização, um *Data Mart* é departamental, englobando subconjuntos de dados que interessam a áreas/departamentos da organização.

Tipicamente, um *Data Warehouse* possui um esquema multidimensional, que utiliza dois elementos: factos e dimensões. Os factos são medidas numéricas, como por exemplo unidades vendidas ou lucro gerado, enquanto que as dimensões são as perspetivas através das quais se podem analisar os factos (Jiawei Han et al., 2011).

De acordo com Di Tria, Lefons, e Tangorra (2014), os *Data Warehouses* tradicionais são desenhados segundo duas abordagens distintas:

- **Orientada aos dados** – consiste na reengenharia das fontes de dados. É criado um esquema multidimensional, tendo em conta os dados disponíveis. Contudo, esta metodologia pode criar um modelo que não satisfaz os requisitos ou objetivos do negócio.
- **Orientada aos requisitos** – tem por base os objetivos do negócio, resultantes das necessidades dos responsáveis pela tomada de decisão. Esta metodologia define esquemas multidimensionais através dos objetivos de negócio, sendo as fontes de dados consideradas mais tarde, durante o processo de ETL. Pode acontecer que na altura de popular o *Data Warehouse*, não existam os dados que são necessários ou que sejam descartadas algumas fontes de dados contendo informação potencialmente interessante.

Estas duas abordagens podem ser combinadas, resultando numa abordagem híbrida que considera as vantagens de ambas. Em contrapartida, as abordagens híbridas são mais complexas, por terem de integrar e considerar em simultâneo, as fontes de dados e os requisitos.

2.2.3 Sistemas Operacionais *versus* Sistemas Analíticos

Numa organização, normalmente é possível identificar sistemas operacionais (OLTP) e sistemas analíticos (OLAP). Desta forma, importa salientar as principais diferenças entre eles.

Os sistemas OLTP tipicamente têm a função de registar as transações que ocorrem no funcionamento diário das organizações, como por exemplo movimentos contabilísticos, movimentos em armazém e encomendas. As tarefas associadas a este tipo de sistemas são estruturadas e repetitivas e consistem em transações que envolvem poucos registos, atómicas e isoladas. Além disso, a consistência e a recuperação dos dados são aspetos críticos e a maximização do processamento das transações é a métrica de desempenho chave (Chaudhuri & Dayal, 1997).

Em contraste com as bases de dados operacionais (que suportam OLTP), o *Data Warehouse* suporta OLAP. Estes sistemas são projetados para o suporte à tomada de decisão, como tal, os dados históricos, sumarizados e consolidados são mais importantes que o máximo detalhe de registos individuais. A preocupação essencial destes sistemas passa por otimizar o processamento de *queries* e os tempos de resposta (Chaudhuri & Dayal, 1997).

Kimball e Ross (2013) recorrem a uma metáfora para explicar a diferença entre os sistemas operacionais e os sistemas analíticos. Para estes autores, face às características de ambos, os sistemas operacionais fazem as rodas de uma organização girar, enquanto que os sistemas analíticos vêm as rodas da organização girar, para avaliar o seu desempenho e tomar decisões. Ou seja, os primeiros tratam de todas as operações diárias da organização, refletindo os dados mais atualizados, enquanto que os segundos vão assistindo e guardando um histórico que lhes permita analisar e aferir se os processos operacionais da organização estão no caminho certo.

Dado que as bases de dados operacionais estão otimizadas para operações OLTP, tentar executar *queries* OLAP complexas nestas bases de dados iria resultar em desempenhos inaceitáveis. Além disso, o suporte à tomada de decisão requer dados de várias fontes (internas e externas) que podem estar em falta nas bases de dados operacionais e que necessitam de passar por processos de transformação e consolidação, visto que, podem possuir diferentes formatos, inconsistências ou qualidade insuficiente (Chaudhuri & Dayal, 1997; Vaisman & Zimnyi, 2014).

As operações típicas de OLAP incluem *roll-up* (aumentar o nível de agregação) e *drill-down* (diminuir o nível de agregação ou aumentar o nível de detalhe) ao longo de uma ou mais hierarquias de dimensão, *slice and dice* (seleção e projeção dos dados) e *pivot* (reorientar a vista multidimensional sobre os dados) (Chaudhuri & Dayal, 1997; Vaisman & Zimnyi, 2014).

Com o objetivo de sintetizar as diferenças entre as bases de dados operacionais e os *Data Warehouses,* a Tabela 6 sistematiza as principais características destes dois tipos de repositórios.

Tabela 6 – Bases de Dados Operacionais versus Data Warehouses. Baseado em (Sá, 2009; M. Y. Santos & Ramos, 2006).

Bases de Dados Operacionais	***Data Warehouses***
Objetivos operacionais, operações diárias (OLTP)	Objetivos analíticos, suporte à tomada de decisão (OLAP)
Acessos de leitura/escrita	Acessos só de leitura
Orientado às aplicações	Orientado aos assuntos
Acesso por transações simples predefinidas	Acesso por questões *ad-hoc,* queries complexas e relatórios periódicos
Dados correntes, atómicos, isolados e relacionais (normalizados)	Dados históricos, sumarizados, integrados e multidimensionais
Acesso a poucos registos de cada vez	Acesso a muitos registos de cada vez
Dados atualizados em *real-time*	Carregamento periódico de mais dados
Estrutura otimizada para atualizações	Estrutura otimizada para processamento de questões

Atualmente, com o aparecimento do *Big Data* tem surgido o desafio de implementar OLAP e *Data Warehouses* nestes contextos, com o objetivo de se conseguir recolher, transformar, carregar e analisar vastos conjuntos de dados. Este desafio constitui um tópico emergente da investigação nesta área, visto que computar cubos OLAP em ambientes com estas características, acarreta dificuldades. Estas dificuldades devem-se, normalmente, a dois fatores intrínsecos do *Big Data*: o tamanho, que se torna realmente vasto neste tipo de conjuntos de dados; e a complexidade, que pode mesmo ser muito elevada em alguns conjuntos de dados (Cuzzocrea, 2015; Cuzzocrea, Bellatreche, & Song, 2013).

2.2.4 *Big Data Warehouses*

Para iniciar esta subsecção, importa salientar que os objetivos principais de um *Data Warehouse* permanecem os mesmos, assim como a necessidade das organizações implementarem estes sistemas, para auxiliar os seus processos de tomada de decisão. Contudo, o volume, tipo e formato dos dados atualmente existentes, colocam em causa as regras de processamento dos *Data Warehouses* tradicionais. Estas regras foram concebidas para lidar com dados relacionais, como tal, não podem ser aplicadas em textos, imagens, vídeos ou dados gerados por sensores (Krishnan, 2013).

Mohanty, Jagadeesh, e Srivatsa (2013) também são da opinião que a influência da *web*, dos dispositivos móveis e outras tecnologias que foram surgindo, desafiaram as abordagens e processos tradicionais. Os dados já não estão centralizados e apenas limitados aos sistemas da organização,

estando cada vez mais distribuídos, com diferentes estruturas e a crescer a taxas exponenciais. Segundo estes autores, as organizações terão de desenvolver um ecossistema de plataformas que utilize o tradicional *Data Warehouse* e o *Big Data* através de uma arquitetura híbrida, um *Big Data Warehouse*.

De acordo com Jukić, Sharma, Nestorov, e Jukić (2015) tem existido o equívoco de que o *Big Data* e os *Data Warehouses* são incompatíveis. Contudo, os autores levaram a cabo um trabalho no qual demonstram que o *Big Data* aumenta as capacidades do *Data Warehouse*, conduzindo a análises de dados e descobertas nos mesmos, que seriam impossíveis recorrendo apenas às tecnologias tradicionais.

A Figura 17 demonstra as dificuldades de lidar com dados que possuem características de *Big Data* e retirar conclusões dos mesmos. Se as organizações armazenarem, utilizarem e analisarem os dados de forma adequada, conjugando as potencialidades do *Big Data* e dos *Data Warehouses*, conseguirão ter uma visão holística dos seus processos de negócio e retirar conclusões adequadas. Caso contrário, poderá suceder como ilustrado na Figura 17, em que cada utilizador retira uma conclusão enviesada, porque não consegue ter uma visão completa (Krishnan, 2013; Wu, Zhu, Wu, & Ding, 2014).

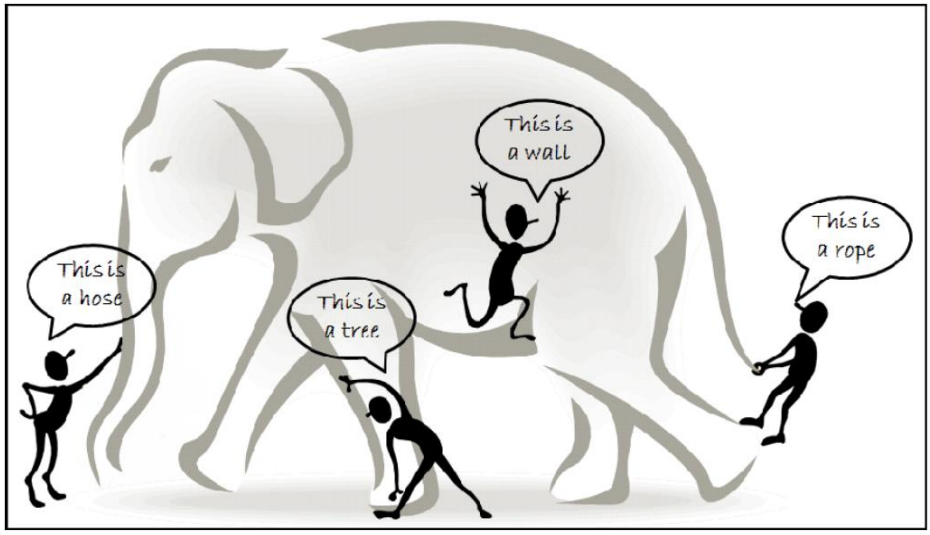


Figura 17 - A visão limitada de cada utilizador conduz a conclusões erradas. Retirado de (Wu, Zhu, Wu, & Ding, 2014).

Di Tria et al. (2014) reconhecem que os *Big Data Warehouses* diferem substancialmente dos *Data Warehouses* tradicionais, uma vez que o seu esquema deve ser baseado em novos modelos lógicos que permitam maior flexibilidade que os modelos relacionais. Por isso, os autores consideram que as metodologias para projetar um *Data Warehouse* devem ser híbridas (para considerar os benefícios das abordagens de *Data Warehouse* tradicionais, orientadas a dados e orientadas a requisitos), incrementais (com base nas abordagens ágeis) e automáticas (para a sua conceção ser mais rápida, mesmo tendo de considerar várias fontes de dados). De acordo com estas características, Di Tria et al. (2014) propõe uma

metodologia híbrida (GrHyMM - *Graph-based Hybrid Multidimensional Model*), considerando tanto as fontes de dados como os requisitos identificados pelos responsáveis pelo processo de tomada de decisão.

Contudo, M. Y. Santos et al. (2017) consideram que o conceito de *Big Data Warehouse* ainda é muito jovem, pelo que faltam abordagens metodológicas que guiem a implementação deste tipo de sistemas. O conceito tem vindo a ganhar interesse, destacando-se a necessidade de fazer alterações aos *Data Warehouses* tradicionais para alcançarem novas características, relevantes em ambientes *Big Data*, nomeadamente (Mohanty et al., 2013; M. Y. Santos, Costa, et al., 2017):

- Processamento de dados altamente distribuído;
- Escalabilidade para suportar dados, utilizadores e análises crescentes, com baixos custos;
- Integrar diversos tipos de dados, incluindo fontes de dados internas ou externas à organização;
- Analisar grandes volumes de dados sem recorrer a amostras;
- Operações *real-time* (processamento em *streaming*, atualizações frequentes...);
- Cargas de trabalho analíticas complexas (exemplo: consultas *ad hoc*, *data mining*, análises exploratórias, vistas materializadas);

De uma forma geral, os *Big Data Warehouses* podem ser implementados seguindo a estratégia "*lift and shift*" ou "*rip and replace*". A primeira consiste em aumentar as capacidades do *Data Warehouse* tradicional com tecnologias *Big Data* (exemplo: Hadoop ou NoSQL), adicionando-se componentes melhores, mais baratos ou mais rápidos. Esta estratégia segue uma abordagem por caso de uso (Clegg, 2015). Na estratégia "*rip and replace*" o *Data Warehouse* tradicional é completamente substituído por tecnologias *Big Data*, representando uma estratégia de modernização viável, quando a solução atual possui deficiências ou não serve os objetivos do negócio. Contudo, esta estratégia pode ser dispendiosa e disruptiva para as organizações com muitos anos de investimentos nas tecnologias tradicionais. Apesar desta desvantagem, a estratégia "*rip and replace*" pode tornar-se atrativa para *start-ups*, pequenas empresas ou grupos de investigação com recursos limitados, visto que grande parte das tecnologias *Big Data* são *open-source* e livres para aquisição e utilização (Russom, 2016; M. Y. Santos, Costa, et al., 2017).

Para terminar, tal como anteriormente referido, Di Tria et al. (2014) consideram que as metodologias de *Big Data Warehouse* devem adotar novos modelos lógicos, como aqueles que são utilizados pelas bases de dados NoSQL, de forma a assegurar escalabilidade, flexibilidade, e melhor desempenho. No contexto desta dissertação, procurar-se-á aplicar uma arquitetura de *Big Data*

Warehouse, na qual o Druid tenha um papel central no armazenamento e análise de vastos conjuntos de dados (históricos e em *real-time*) com alto desempenho (Yang et al., 2014).

2.2.5 Modelos de Dados

As bases de dados constituem uma componente essencial nos sistemas de informação atuais, sendo que, uma das suas características fundamentais é o facto de fornecerem níveis de abstração sobre os dados, isto é, ocultam detalhes do armazenamento dos dados e destacam características essenciais para uma melhor perceção dos mesmos por parte dos utilizadores. Os modelos de dados – conjunto de conceitos que pode ser usado para descrever a estrutura da base de dados – providenciam os meios necessários para alcançar esta abstração (Elmasri & Navathe, 2015; Vaisman & Zimnyi, 2014).

Os modelos de dados são peças centrais em *Business Intelligence* e *Analytics*, uma vez que asseguram que as necessidades analíticas são devidamente consideradas e permitem que sejam efetuadas análises aos dados através de diferentes perspetivas (M. Y. Santos & Costa, 2016b).

Segundo Elmasri e Navathe (2015), existem vários modelos de dados, que podem ser categorizados de acordo com o tipo de conceitos que utilizam para descrever a estrutura da base de dados e o nível de abstração que proporcionam. A Tabela 7 apresenta as categorias de modelos de dados consideradas por estes autores e também mencionadas por Vaisman e Zimnyi (2014).

Tabela 7 - Categorias de Modelos de Dados. Baseado em (Elmasri & Navathe, 2015; Vaisman & Zimnyi, 2014).

Modelo de Dados	Descrição
Concetual	Fornece conceitos que são muito próximos da forma como os utilizadores compreendem os dados. É um modelo de alto nível e, como tal, não inclui pormenores de implementação. Entidades, atributos e relações são conceitos normalmente associados a este modelo.
Lógico	Providencia conceitos que podem ser facilmente compreendidos pelos utilizadores, mas que não possuem um detalhe muito distante da forma como os dados são armazenados. Ou seja, traduz e detalha o modelo concetual e serve de base ao modelo físico.
Físico	Trata-se de um modelo de baixo nível, como tal, detalha a forma como os dados estão armazenados. Este modelo, parte do modelo lógico e customiza-o para uma plataforma de gestão de base de dados específica (exemplo: SQL Server). Este tipo de modelos é geralmente concebido para especialistas e não para utilizadores em geral.

No que à implementação de *Data Warehouses* diz respeito, Dehdouh, Bentayeb, Boussaid, e Kabachi (2015) também distinguem estes três níveis de abstração (Modelo Concetual, Modelo Lógico e Modelo Físico), apresentados na Figura 18. Os modelos lógicos visam a reorganização dos dados de acordo com a arquitetura de armazenamento mais adequada a potenciar os sistemas de gestão de base de dados. Desta forma, tal como é visível na Figura 18, o modelo físico situa-se entre o modelo concetual e o modelo físico, fornecendo mais detalhes que o primeiro sobre a estruturação dos dados e as suas relações e preparando a transição para o nível físico. Estas características, segundo os autores, fazem do modelo lógico o mais decisivo no processo de modelação.

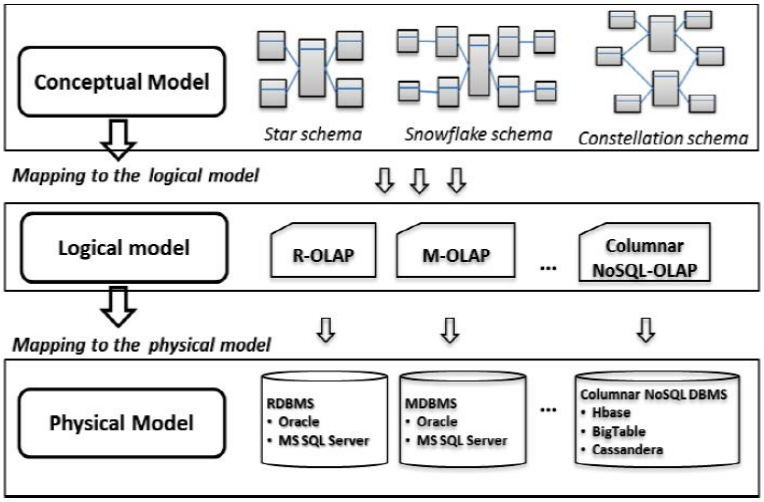


Figura 18 - Modelos utilizados na implementação de um Data Warehouse. Retirado de (Dehdouh et al., 2015).

O esquema multidimensional é o modelo concetual de referência na construção de *Data Warehouses*, sendo que os modelos mais populares são o modelo em estrela, o modelo em floco de neve e o modelo em constelação, representados na Figura 18 (Dehdouh et al., 2015). Dada a importância destes três modelos multidimensionais é apresentada, de seguida, uma descrição sucinta de cada um deles (Elmasri & Navathe, 2015; Vaisman & Zimnyi, 2014):

- **Modelo em Estrela** – consiste numa tabela de factos central e um conjunto de tabelas em volta, uma para cada dimensão;
- **Modelo em Floco de Neve** – consiste numa variação do modelo em estrela. As tabelas das dimensões são normalizadas, organizadas em hierarquias;
- **Modelo em Constelação** – consiste num conjunto de tabelas de factos que partilham tabelas de dimensões.

Ao nível do modelo lógico, tradicionalmente têm sido utilizadas três abordagens: ROLAP (*Relational-OLAP*), MOLAP (*Multidimensional-OLAP*) e HOLAP (*Hybrid-OLAP*). Contudo, o modelo lógico

adotado por estas abordagens tradicionais não é adequado para vastos volumes de dados, inerentes a contextos *Big Data*. Para suprir este problema têm surgido novos modelos, como os modelos NoSQL orientados a colunas. Tal como se pode observar na Figura 18, estes novos modelos (na figura exemplificados por "Columnar NoSQL-OLAP") têm a missão de fazer a associação entre os modelos concetuais tradicionais e as bases de dados NoSQL adotadas no modelo físico (Dehdouh et al., 2015).

M. Y. Santos e Costa (2016a) realçam que em contextos *Big Data*, devido às características das bases de dados NoSQL, as tarefas de modelação têm de ser vistas por outra perspetiva. Contudo, segundo os autores, estas bases de dados também necessitam de modelos que assegurem o adequado armazenamento e consulta dos dados. Além disso, mesmo que numa fase inicial de recolha de dados se possa ser mais flexível, a certa altura terá de se proporcionar alguma estrutura aos dados, de forma a permitir análises por parte dos utilizadores.

De acordo com M. Y. Santos e Costa (2016a), se se pensar na vasta quantidade de bases de dados relacionais atualmente implementadas e se fizer o exercício de prever a necessidade de transferir estas bases de dados para formatos colunares, devido ao crescente volume de dados, por exemplo, conclui-se que a importância dos modelos de dados em *Big Data* aumenta. Em contextos que exijam uma rápida transição do ambiente tradicional para o ambiente *Big Data*, a proposta de regras objetivas para esta transição beneficiaria os utilizadores e asseguraria o correto armazenamento dos dados.

Neste sentido, o trabalho de M. Y. Santos e Costa (2016a) propõe um conjunto de regras para a transição automática entre o ambiente tradicional e o ambiente *Big Data*, considerando dois objetivos: a identificação de um modelo de dados colunar a aplicar, por exemplo no HBase, para suportar necessidades operacionais; e a identificação de um modelo de dados tabular, a aplicar por exemplo no Hive[13], para suportar as necessidades analíticas. Os mesmos autores, em M. Y. Santos e Costa (2016b) apresentam um exemplo para a construção de um *Big Data Warehouse*, propondo um conjunto de regras para transformar um modelo multidimensional (normalmente utilizado para modelar um *Data Warehouse* tradicional) num modelo de dados tabular, adequado à implementação no Hive.

Dehdouh et al. (2015) propõem três tipos de modelos lógicos: NLA (*Normalized Logical Approach*), DLA (*Denormalized Logical Approach*), e DLA-CF (*Denormalized Logical Approach by using Column Family*). Para além de descreverem cada um dos modelos e de demonstrarem como se pode tirar partido dos mesmos para a construção de *Data Warehouses*, os autores comparam o desempenho dos três modelos para suportar análises. Os modelos desnormalizados (DLA e DLA-CF) revelaram-se

[13] https://hive.apache.org/

mais adequados para responder a análises, uma vez que obtiveram um tempo de execução três vezes inferior aos modelos normalizados (NLA).

No âmbito desta dissertação pretende-se ter a capacidade de analisar os dados históricos e em *real-time*, em contextos *Big Data*, da forma mais rápida possível. Para tal, o modelo de dados utilizado terá de estar preparado de forma a otimizar este tipo de consultas.

2.3 *Real-time* em *Big Data Warehouses*

A análise dos dados para prever tendências de mercado ou melhorar o desempenho das organizações esteve sempre presente na tarefa de liderar uma organização. Contudo, nos dias de hoje, caracterizados por uma competição cada vez mais agressiva e pela rápida mudança das necessidades dos clientes e das tecnologias disponíveis, os responsáveis pela tomada de decisão já não estão satisfeitos com relatórios ou *dashboards* pré-configurados. Pelo contrário, pretendem fazer queries *ad hoc* que retornem respostas rapidamente, tendo por base informação em *real-time* e pretendem ter esta visão ao seu dispor quando e onde necessitarem (Azvine, Cui, & Nauck, 2005; Sahay & Ranjan, 2008).

Desta forma, as plataformas tradicionais de BI (*Business Intelligence*), nas quais se programavam refrescamentos diários, semanais ou mensais aos dados, já não satisfazem os requisitos das organizações (Liu, Lftikhar, & Xie, 2014). A integração de novos dados no *Data Warehouse* não pode ocorrer da forma tradicional (*offline*), uma vez que isto acarreta que enquanto o refrescamento ocorre os utilizadores e aplicações OLAP não possam consultar os dados. Para evitar este problema, estas tarefas eram calendarizadas para janelas temporais em que os sistemas não eram utilizados (exemplo: após o horário de trabalho). Contudo, esta solução é inviável para empresas que operam durante todo o dia, não restando janelas temporais em que os sistemas não sejam utilizados ou para aquelas que necessitam da informação da forma mais atempada possível (R. J. Santos & Bernardino, 2008). Daqui surge a necessidade de concretização de um *Real-Time Data Warehouse* (RTDW), tema detalhado mais à frente nesta secção.

Mohamed e Al-Jaroodi (2014) reforçam a importância de fazer análises em *real-time* (ou tempo oportuno), em contextos *Big Data*, referindo que este é um dos aspetos essênciais para alcançar o sucesso em muitas organizações e contextos, visto que, em muitos casos, se a informação e análise da mesma não ocorrer em tempo oportuno esta deixará de ter valor. Os autores discutem ainda que *real-time Big Data* tem muitas aplicações importantes, como por exemplo em decisões militares, e que a

correta implementação deste tipo de sistemas pode conduzir a vantagens e benefícios significativos como salvar a vida de milhares de pessoas, melhorar a qualidade de vida, reduzir riscos e aumentar lucros.

Lebdaoui, Orhanou, e Elhajji (2014) acrescentam que, para atingir os requisitos de *real-time* é importante ter em atenção a gestão dos dados provenientes de diversas fontes e o tratamento e entrega dos mesmos para análise em tempo útil. A integração dos dados é efetuada por um processo de ETL contínuo, para garantir que a informação é entregue da forma mais rápida possível (Liu et al., 2014).

De acordo com um *survey* de *Big Data* efetuado a 274 organizações na Europa, existe uma tendência clara por tornar os dados disponíveis para análise em *real-time* e, 70% dos respondentes indica a necessidade de processamento *real-time*. Contudo, apenas 9% das organizações fez progressos neste aspeto, devido à complexidade e desafios técnicos no processamento *real-time*. Isto demonstra a importância do *real-time* em contextos analíticos de *Big Data* e todo o trabalho que ainda há por fazer, visto que apesar de lhe ser reconhecida importância, poucas organizações conseguiram progressos nesta área (Liu et al., 2014).

Na ótica de Mohamed e Al-Jaroodi (2014), os desafios que a implementação de *real-time Big Data* acarreta incluem: transferência de dados em *real-time*, descoberta de situações excecionais em *real-time*, *real-time analytics* e tomada de decisão em *real-time*.

Importa clarificar que *near real-time* surge normalmente associado ao *real-time*, sendo que no contexto desta dissertação será utilizado apenas o termo *real-time*. Este termo não possui uma definição consensual, sendo que segundo Azvine et al. (2005) pode significar:

- O requisito para obter latência zero num processo;
- Que um processo tem acesso à informação sempre que necessário;
- Que um processo fornece informação sempre que a gestão necessita;
- A capacidade de obter medidas de desempenho e análises não apenas sobre dados históricos, mas também sobre dados atuais.

Para a concretização do *real-time*, em contextos tradicionais ou em contextos Big Data, diferentes abordagens têm sido seguidas, as quais são de seguida apresentadas.

Abordagens com tecnologias tradicionais

Numa abordagem tradicional (fora de contextos *Big Data*), começou por ser identificada a necessidade de possuir um *Real-Time Data Warehouse* (RTDW), tal como já foi anteriormente referido. O RTDW é visto como uma extensão ao DW tradicional, capaz de capturar as rápidas alterações nos dados,

assim como suportar o processo de análise e decisão em *real-time.* O grande desafio é o acesso aos dados sem atrasos de processamento (Li & Mao, 2015).

Farooq e Sarwar (2010) levaram a cabo um trabalho, no qual referem que para além do desafio da integração dos dados em *real-time* (pelo processo de ETL), há também que ter em conta a manutenção do modelo multidimensional do *Data Warehouse*, isto porque, na ótica dos autores, não existe nenhuma tecnologia de *real-time* ETL que consiga auxiliar a reduzir a latência, caso esta seja causada pelos modelos. No seu estudo, os autores concluem que os modelos multidimensionais semiestruturados (exemplo: modelo XML) trazem vantagens sobre os estruturados (exemplo: modelo relacional), reduzindo o tempo de reposta na execução de *queries* ao RTDW.

Outros autores têm tentado endereçar o problema de reduzir a latência para que os dados possam ser acedidos em *real-time* a fim de constituir uma importante vantagem competitiva. Vaisman e Zimnyi (2014) argumentam que as necessidades por dados em *real-time* e a diminuição da latência são dependentes do contexto, sendo que a estratégia de aumentar a frequência do processo de ETL (denominada por *near real-time* ETL) é a mais simples. Contudo, esta estratégia não é suficiente quando a latência precisa de ser drasticamente reduzida.

Freudenreich et al. (2013) argumentam que o processo de integração dos dados no DW pode ser otimizado se, ao invés de se tratar o processo de ETL como uma fase separada e executando numa infraestrutura independente, se mover os dados para o DW através de um processo de ELT. De seguida utilizam-se as capacidades de processamento do DW para fazer a transformação dos dados e materialização de vistas para que estes possam ser acedidos. Assim, segundo os autores, faz-se uma melhor utilização dos recursos, porque se processam apenas os dados realmente necessários e se evita ter de esperar que a fase de pré-processamento tenha de terminar para se poder analisar os dados.

Uma abordagem clássica é a criação de partições *real-time*. Neste caso os dados *real-time* e os dados históricos são guardados em tabelas separadas. A tabela com os dados em *real-time* terá o mesmo esquema da tabela com os dados históricos, mas apenas terá disponível, por exemplo, os dados de um dia, de forma a garantir a rápida consulta dos dados. Neste exemplo de granularidade, no final do dia os dados da tabela de *real-time* são integrados na tabela com os dados históricos. Na análise dos dados, são considerados os dados provenientes de ambas as tabelas. Todavia, esta abordagem pode comprometer a qualidade dos dados (Kimball & Ross, 2013; Vaisman & Zimnyi, 2014). R. J. Santos e Bernardino (2008) contribuem com uma abordagem semelhante de carregamento contínuo de dados para o DW. Nesta abordagem são criadas tabelas temporárias com estrutura semelhante às tabelas do DW, sem a definição de qualquer tipo de chaves, índices ou restrições de qualquer tipo. As consultas de

dados consideram dados de ambas as tabelas e, ao fim do tempo desejado, os dados das tabelas temporárias passam para o DW.

Li e Mao (2015) propõem uma *framework* de *real-time* ETL, apresentada na Figura 19, que separa o processamento de dados históricos do processamento de dados *real-time* e combina uma área de armazenamento dinâmico com uma tecnologia de replicação dinâmica, de forma a evitar degradação de desempenho entre as queries OLAP e as atualizações OLTP. Os dados em *real-time* passam por uma área de armazenamento dinâmica que faz a sua gestão e manutenção, utilizando técnicas de replicação. De seguida, os dados são armazenados no DW, na área destinada aos dados em *real-time*. O processamento dos dados históricos é efetuado em *batch* diretamente para a área do DW que lhes é destinada.

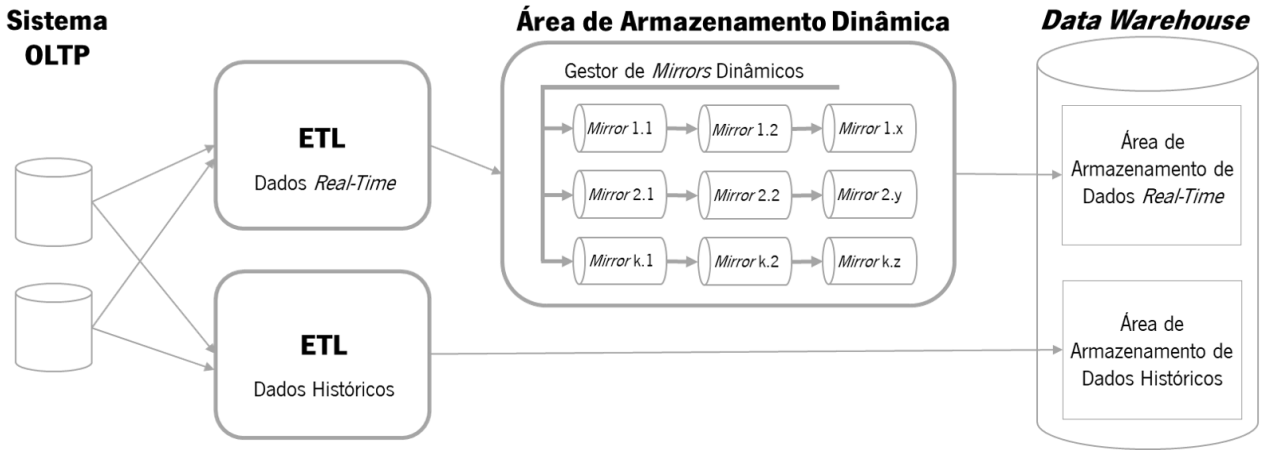


Figura 19 - Framework RTDW. Adaptada de (Li & Mao, 2015).

Golab e Johnson (2014) alertam que apesar da separação entre os dados *real-time* e os dados históricos ser uma solução tipicamente apontada, ainda não é completamente claro como é que esta divisão deve ser feita. Para além disso, os dados históricos fornecem contexto para interpretar os novos dados. Os autores destacam ainda alguns desafios e problemas em aberto, tais como a implementação de técnicas de controlo para garantir que as consultas não são bloqueadas pela atualização dos dados ou a implementação de técnicas ao nível da organização dos dados, como o particionamento horizontal por um atributo temporal.

Abordagens com tecnologias Big Data

Uma das arquiteturas identificadas na literatura é a arquitetura *Lambda*, proposta por Marz e Warren (2015) e apresentada na Figura 20. A ideia principal desta arquitetura é pensar no sistema de *Big Data* como um conjunto de camadas que satisfazem necessidades particulares e é desenhada para lidar com grandes conjuntos de dados, sejam eles históricos ou recolhidos em *real-time* (C. Costa & Santos, 2017; Marz & Warren, 2015).

A arquitetura está dividida em três camadas principais: *Batch Layer*, *Serving Layer* e *Speed Layer*. Na *Batch Layer,* o *Master Dataset* armazena todos os dados. Como é impensável ler de uma fonte de dados que pode ter *petabytes* de informação sempre que uma *query* é executada, a *Batch Layer* disponibiliza *Batch Views* como resultado das funções que executa. Nesta fase entra a *Serving Layer*, uma base de dados distribuída que carrega as *Batch Views* (pré-computações do *Master Dataset*) e torna possível que sejam acedidas. Desta forma, a *Serving Layer* suporta *queries* arbitrárias, melhorando o desempenho porque se evita que cada *query* tenha de consultar todo o *Master Dataset*. Apenas com estas duas camadas, as *Batch Views* rapidamente ficam desatualizadas, uma vez que vão chegando novos dados enquanto está a ser percorrido o processo de pré-computação das *Batch Views*, entre a *Batch Layer* e a *Serving Layer*, o que compromete os requisitos de contextos que necessitem de dados em *real-time*. Assim, os autores propõem ainda a *Speed Layer*, que tem o objetivo que computar funções nos dados em *real-time,* de forma tão rápida quanto as aplicações ou contextos necessitarem. Quando as *Batch Views* com os novos dados estejam processadas, as *Real-Time Views* com esses dados podem ser descartadas (C. Costa & Santos, 2017; Marz & Warren, 2015).

Esta arquitetura é um paradigma, ou seja, as tecnologias que implementam cada um dos *Layers* são independentes da ideia base. A *Batch Layer* pode usar tecnologias como por exemplo o Hadoop, enquanto que a *Speed Layer*, tipicamente é implementada recorrendo a tecnologias como o Storm[14] e Spark[15]. Por fim, a *Serving Layer* pode ser implementada através de sistemas como o HBase ou Cassandra[16] (Liu et al., 2014).

Além desta arquitetura, têm surgido outras na literatura que tentam endereçar o desafio de disponibilizar para análise dados históricos e em *real-time* e que dão mais detalhes quanto às tecnologias utilizadas e dificuldades encontradas.

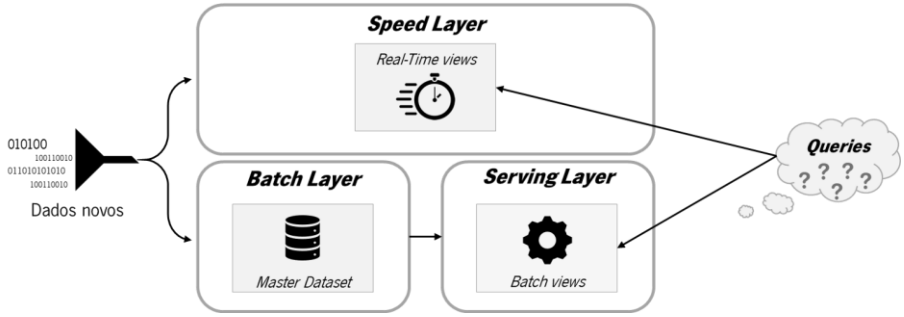


Figura 20 – Arquitetura Lambda. Adaptada de (Marz & Warren, 2015).

[14] http://storm.apache.org/
[15] https://spark.apache.org/
[16] http://cassandra.apache.org/

Mishne, Dalton, Li, Sharma, e Lin (2013) ilustram os desafios de se desenvolver um sistema que suporte processamento *real-time* em contextos *Big Data*. Os autores começaram por fazer uma implementação baseada no Hadoop, mas que se revelou inadequada para processamento de dados com baixa latência, consequentemente, tiveram de fazer uma segunda implementação especialmente desenhada para a tarefa. Baseados na experiência que este trabalho lhes concedeu, os autores consideram importante que, como trabalho futuro, se investigue e tente desenvolver plataformas que consigam lidar tanto com vastos volumes de dados, como com a velocidade com que esses dados podem fluir na rede. Cha e Wachowicz (2015) argumentam que o Hadoop é agora um competidor viável aos sistemas existentes, mas concordam que tem de ser desenvolvida investigação, a fim de colmatar esta falha em conceber plataformas para *Real-Time Big Data Analytics.*

No seu trabalho, G. J. Chen et al. (2016) consideram que as soluções em *real-time* são importantes porque fornecem informação valiosa, em tempo útil, à medida que os eventos vão acontecendo, permitindo que se possa agir e decidir rapidamente. Estes autores referem também que é relevante considerar cinco importantes decisões, quando se pretende implementar um sistema de processamento *real-time*, nomeadamente: a facilidade de utilização, o desempenho, a tolerância a faltas, a escalabilidade e a exatidão dos dados. Baseados nos *tradeoffs* entre as várias alternativas, os autores identificam e discutem a arquitetura que implementaram no Facebook. De uma forma resumida, na sua arquitetura, os eventos que vão sucedendo nas fontes de dados são registados e enviados, através da tecnologia de distribuição de mensagens Scribe[17], para os sistemas de *streaming* (Puma[19], Stylus[19] e Swift[19]) e para o armazenamento dos dados (Laser[19], Scuba[19] e Hive, cada um respondendo a diferentes tipos de *queries*). A arquitetura considera ainda o Presto[18] para fazer consultas aos dados armazenados no Hive.

Apesar dos contributos mais recentes, continua a não ser possível identificar muitos trabalhos que enderecem a questão de construir uma plataforma capaz de considerar dados históricos e dados em *real-time* e que ao mesmo tempo garanta o rápido acesso e análise dos mesmos, um *Real-Time Big Data Warehouse* (RTBDW). Além disso, as arquiteturas identificadas continuam a considerar diferentes tecnologias para armazenar os dados *Real-Time* e os dados históricos. Um exemplo disto é a arquitetura proposta por M. Y. Santos, e Sá, et al. (2017), na qual armazenam os dados *real-time* no Cassandra e os dados históricos no Hive, utilizando depois o Presto para fazer *queries* a ambas as tecnologias de

[17] Tecnologia em desenvolvimento, sem *website* oficial.
[18] https://prestodb.io/

armazenamento, de forma a conseguir consultar os dados de forma integrada. O trabalho de Lima (2017) também utiliza uma abordagem semelhante à anterior, comparando o desempenho do Hive e do Cassandra, mas considerando diferentes tabelas para os propósitos de *real-time* e os propósitos históricos. Os melhores resultados deste autor foram numa abordagem de criação de uma tabela particionada por hora, no Hive, para receber os eventos em *real-time*, sendo que depois os dados desta tabela são movidos para uma tabela com diferentes características e que conserva a informação histórica.

O Druid, tecnologia em foco nesta dissertação, é concebido para suportar análises exploratórias em vastos volumes de dados, históricos e em *real-time*, pelo que será importante perceber em que contextos é que esta tecnologia poderá efetivamente servir um RTBDW e obter melhor desempenho que as soluções existentes na literatura (como aquela proposta por M. Y. Santos, e Sá, et al. (2017)) colmatando todas as dificuldades que foram relatadas nos trabalhos que tentaram endereçar desafios semelhantes (Yang et al., 2014).

Na comunidade científica existem ainda muito poucos trabalhos que explorem a incorporação do Druid e que demonstrem o seu desempenho, numa arquitetura com contextos semelhantes aos desta dissertação. O trabalho de Fangjin Yang et al. (2017) é um dos poucos exemplos propondo uma arquitetura *open source* denominada de *Real-time Analytics Data Stack* (RADStack). Esta arquitetura é baseada na arquitetura *Lambda* e utiliza o Druid como *Serving Layer* em detrimento do HBase, por exemplo. Contudo, na comunidade técnica existem mais exemplos da implementação do Druid como se verifica na página "Powered by Druid"[19] do *website* do Druid.

2.4 Mapa de Conceitos

De forma a sistematizar o enquadramento concetual é apresentado, na Figura 21, um mapa integrando os conceitos mais relevantes que foram abordados neste capítulo. Para cada um dos conceitos são apresentadas algumas informações suportadas pela revisão de literatura, nomeadamente: a forma como cada conceito se define, os desafios que encaram, as oportunidades que se vislumbram e as suas limitações. É ainda evidenciado como é que os conceitos se relacionam entre si.

[19] http://druid.io/druid-powered.html

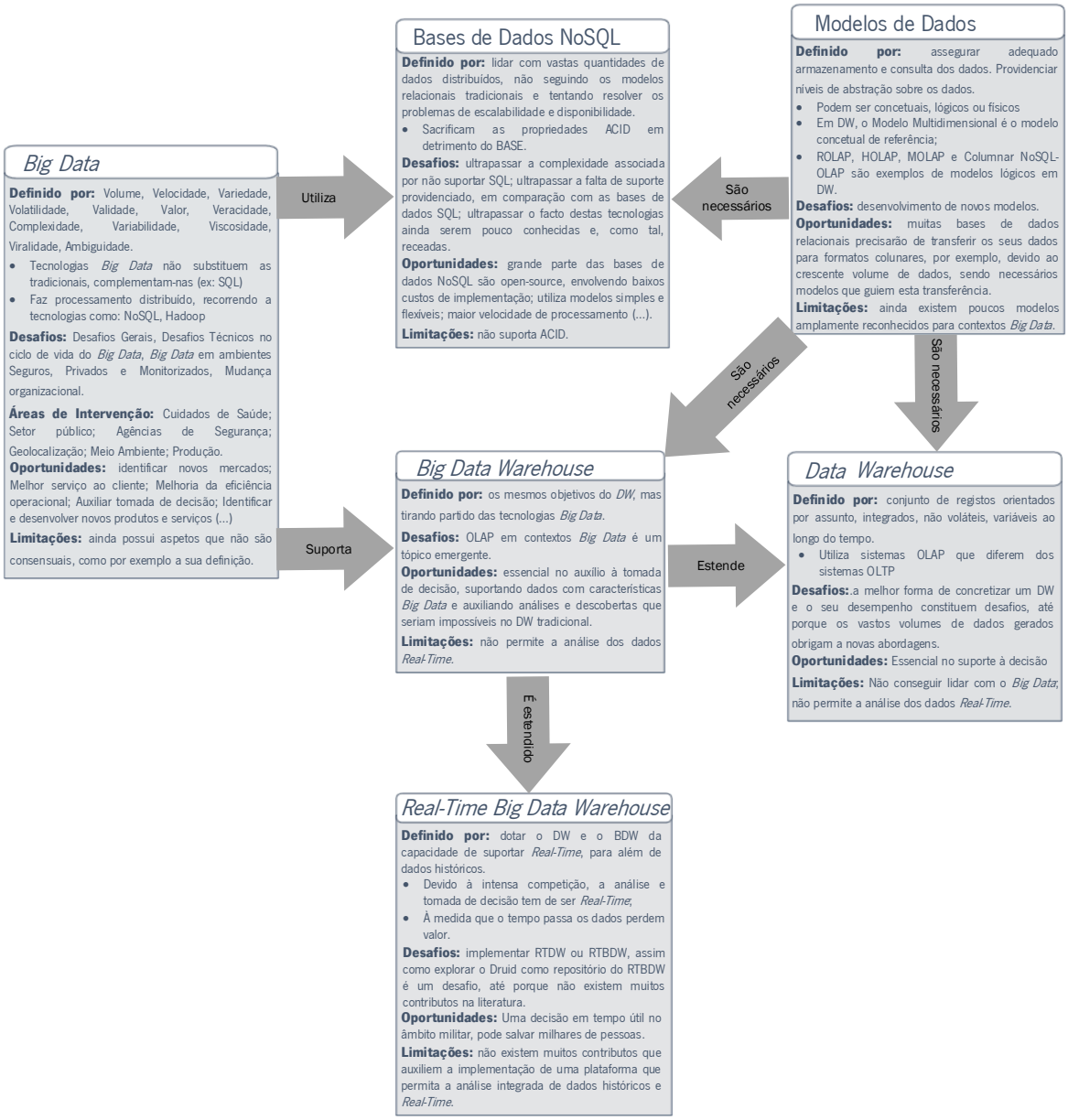


Figura 21 - Mapa de Conceitos do Enquadramento Concetual.

Terminado o enquadramento concetual é possível depreender que a ambiguidade frequentemente associada a alguns dos conceitos pode ser dissipada pelo estudo da literatura científica e técnica.

Vive-se numa época em que as organizações e as pessoas que as constituem têm de lidar com uma pressão cada vez maior para fazer mais, melhor e mais rápido, com cada vez menos recursos. Desta forma urge tirar o melhor partido possível das tecnologias à disposição, a fim de suportar as organizações no seu dia a dia e nos processos de tomada de decisão que têm de ser rápidos e acertados. Neste sentido, as organizações terão de perceber os contextos para os quais a utilização singular das tecnologias tradicionais já não é adequada e alicerçar os seus processos em tecnologias *Big Data* ou na combinação destas com as tecnologias tradicionais.

O *Data Warehouse*, nos dias que correm, é um exemplo da inadequação para lidar com os dados que as organizações necessitam de analisar diariamente. A forma tradicional como os *Data Warehouses* estão construídos não suporta as características do *Big Data*, de tal forma que as organizações terão de optar por outras estratégias para ter uma visão central da sua informação. Essas estratégias passam por dotar os *Data Warehouses* com as tecnologias *Big Data*, originando-se um *Big Data Warehouse*, que responderá a objetivos semelhantes ao *Data Warehouse*, ao mesmo tempo que supre as suas limitações.

Contudo, há ainda um importante aspeto a considerar, que se prende com o facto de as organizações necessitarem não só de analisar dados históricos, mas também dados em *Real-Time*, integrados num único repositório. Assim, é importante munir o *Big Data Warehouse* da capacidade de integrar os dados em *Real-Time*, concebendo-se um *Real-Time Big Data Warehouse*. Além disso, para assegurar o adequado armazenamento e acesso aos dados é importante considerar e desenvolver novos Modelos de Dados.

Na comunidade científica há poucos contributos demonstrando a implementação deste tipo de plataformas. Existem ainda menos contributos que considerem o Druid como o repositório do *Real-Time Big Data Warehouse*, comprovando-se o carácter inovador desta dissertação.

No enquadramento concetual procurou-se utilizar artigos relevantes na área e auxiliar a criação de uma base firme de conhecimento em volta das temáticas endereçadas, ajudando a perceber cada um dos conceitos, os desafios encontrados, as áreas que carecem mais de investigação e a enquadrar melhor esta dissertação.

3. ENQUADRAMENTO TECNOLÓGICO

Este capítulo apresenta as tecnologias que são relevantes no contexto desta dissertação. O capítulo será essencialmente focado no Druid, por ser a tecnologia em foco nesta dissertação, contudo, também será apresentado o ecossistema Hadoop e outras tecnologias que podem ser relevantes no contexto deste trabalho, uma vez que a sua integração com o Druid permitirá propor uma arquitetura de RTBDW. O capítulo termina com a especificação do protocolo de testes efetuado nesta dissertação e com a infraestrutura tecnológica que permitiu a execução dos mesmos.

3.1 Ecossistema do Hadoop

O Hadoop é um projeto *open-source* que permite o armazenamento e processamento de vastos conjuntos de dados, de forma distribuída, num *cluster* escalável com múltiplos *nodes* e utilizando *commodity hardware*, o que possibilita a sua utilização incorrendo em baixos custos (Apache Hadoop, 2018; C. L. P. Chen & Zhang, 2014; C. Costa & Santos, 2017; Krishnan, 2013). Este projeto é inspirado no *Google File System* (GFS) e no paradigma de programação MapReduce (Zikopoulos et al., 2011). O Hadoop inclui dois componentes principais: o *Hadoop Distributed File System* (HDFS) e o Hadoop MapReduce, que é uma *framework* de processamento distribuído (C. L. P. Chen & Zhang, 2014; C. Costa & Santos, 2017).

No que diz respeito ao HDFS, os dados são partidos em ficheiros mais pequenos (denominados *blocks*) distribuídos e replicados pelos vários *nodes* do *cluster.* Desta forma, as funções de *Map* e de *Reduce* podem ser executadas em subconjuntos mais pequenos que a vasta fonte de dados original, o que providencia alguns requisitos, como por exemplo escalabilidade, disponibilidade e tolerância a falhas, importantes em contextos de processamento *Big Data* (C. Costa & Santos, 2017; Krishnan, 2013; Zikopoulos et al., 2011).

Por sua vez, o Hadoop MapReduce é um modelo de programação e um motor de execução para o processamento distribuído de vastas fontes de dados armazenadas no HDFS. O MapReduce é baseado num método de dividir e conquistar, isto é, divide um problema complexo em vários problemas mais simples e, no final, combina a solução de cada problema simples, a fim de retornar a solução para o problema original. Este processo é implementado em duas fases, a fase de *Map* e a fase de *Reduce* (C. L. P. Chen & Zhang, 2014; C. Costa & Santos, 2017).

Os principais componentes que constituem o HDFS, dividem-se em dois tipos de *nodes*: o *NameNode*, responsável por armazenar metadados sobre os *blocks* e os *nodes;* e o *DataNode*, que armazena os blocos de dados (C. Costa & Santos, 2017; Krishnan, 2013). Quanto ao Hadoop MapReduce, este também é constituído por dois tipos de *nodes*: o *JobTracker*, responsável por agendar jobs e distribuir tarefas pelo outro tipo de *nodes*, denominado *TaskTracker* (C. L. P. Chen & Zhang, 2014). Ao longo dos anos, o Hadoop tem evoluído, tendo sido feita uma transição do MapReduce para o YARN (ou MapReduce 2.0), que repensa o *JobTracker* e o *TaskTracker*, substituindo-os por um *ResourceManager*, um *NodeManager* e um *ApplicationMaster*, de forma a resolver alguns dos problemas identificados no Hadoop MapReduce (C. Costa & Santos, 2017).

Os principais módulos que constituem o Hadoop são os seguintes (Apache Hadoop, 2018):

- *Common* – suportam os outros módulos do Hadoop;
- HDFS: sistema de ficheiros distribuído que permite o armazenamento e acesso a grandes conjuntos de dados;
- Hadoop YARN: *framework* de gestão de tarefas e de recursos do *cluster*;
- Hadoop MapReduce - utilizado para o processamento paralelo de grandes conjuntos de dados.

Além disto, existem muitos projetos relacionados com o Hadoop e que formam o ecossistema Hadoop, como por exemplo: Ambari[20], Avro[21], Cassandra, Chukwa[22], HBase, Hive, Mahout[23], Pig[24], Spark, Tez [25]e Zookeeper[26] (Apache Hadoop, 2018).

Importa referir que alguns destes projetos (e outros que não se encontram listados) serão abordados na secção 3.3.

Para terminar, é relevante referir que o Druid também possui relação com o ecossistema Hadoop, uma vez que utiliza o HDFS e o Zookeeper, sendo que esta utilização será abordada na secção seguinte (Yang et al., 2014).

[20] https://ambari.apache.org/
[21] https://avro.apache.org/
[22] http://chukwa.apache.org/
[23] https://mahout.apache.org/
[24] https://pig.apache.org/
[25] https://tez.apache.org/
[26] https://zookeeper.apache.org/

3.2 Druid

O Druid foi disponibilizado como uma solução *open-source* em 2012, por desenvolvedores da Metamarkets[27]. Este sistema é um repositório de dados distribuído, orientado a colunas, que possui uma arquitetura *shared-nothing* e que utiliza estruturas avançadas de indexação para permitir análises exploratórias sobre vastos conjuntos de dados históricos e *real-time,* suportando rápidas agregações de dados, filtros flexíveis e baixa latência na ingestão de dados (Chambi et al., 2016; Díaz, Martín, & Rubio, 2016; Yang et al., 2014, 2017).

A principal motivação para a criação do Druid advém do facto de plataformas como o Hadoop armazenarem grandes quantidades de dados, mas não garantirem o quão rapidamente estes dados podem ser ingeridos, armazenados e acedidos. Também os sistemas de bases de dados relacionais e as bases de dados NoSQL se revelaram inadequadas para estes contextos. O Druid surgiu então com o objetivo de suprir estes problemas, providenciando uma plataforma adequada para contextos que requerem baixa latência na ingestão e consulta do dados, assim como análises flexíveis e rápidas com diferentes agregações sobre os dados (Díaz et al., 2016; Yang et al., 2014). Um dos desafios desta tecnologia é permitir que os utilizadores consigam fundamentar as suas decisões, de uma forma atempada, em factos que derivam da análise de dados recebidos *real-time*. O tempo desde que um evento ocorre até ao tempo em que este seja consultável, determina o quão rapidamente as partes interessadas podem reagir a potenciais situações anómalas nos seus sistemas. Em 2016, o Druid já era adotado por organizações como: eBay, Yahoo e Netflix (Chambi et al., 2016).

Por fim, é relevante referir que o Druid possui integração com o Ambari, o que facilita a sua instalação, configuração e manutenção (Hortonworks, 2018b).

3.2.1 Arquitetura

Um *cluster* Druid é composto por diferentes tipos de *nodes*, cada um deles concebido para fazer um conjunto de tarefas muito preciso, de forma a separar responsabilidades e simplificar a complexidade do sistema. Numa arquitetura distribuída, cada *node* estará presente numa máquina e será totalmente independente dos outros *nodes* no *cluster*. Desta forma, os *nodes* não partilham dados nem recursos materiais com outros *nodes* do *cluster*, por isso, falhas *intra-cluster* terão um impacto mínimo na disponibilidade dos dados (Chambi et al., 2016; Yang et al., 2014).

[27] https://metamarkets.com/

A designação Druid surgiu da classe Druid presente em muitos *role-playing games* (RPG): é um mutante, capaz de adquirir diferentes formas para cumprir papéis distintos (Yang et al., 2014). Esta inspiração pelo nome reflete a capacidade desta tecnologia se adaptar a diferentes papéis, através dos vários *nodes*, sendo capaz de lidar com dados históricos e dados que são recebidos em *real-time*, juntando todos os *nodes* num único sistema. A composição de um *cluster* Druid e o fluxo de dados que ocorre através do mesmo é apresentada na Figura 22, sendo que os *nodes* da arquitetura serão a seguir explicados.

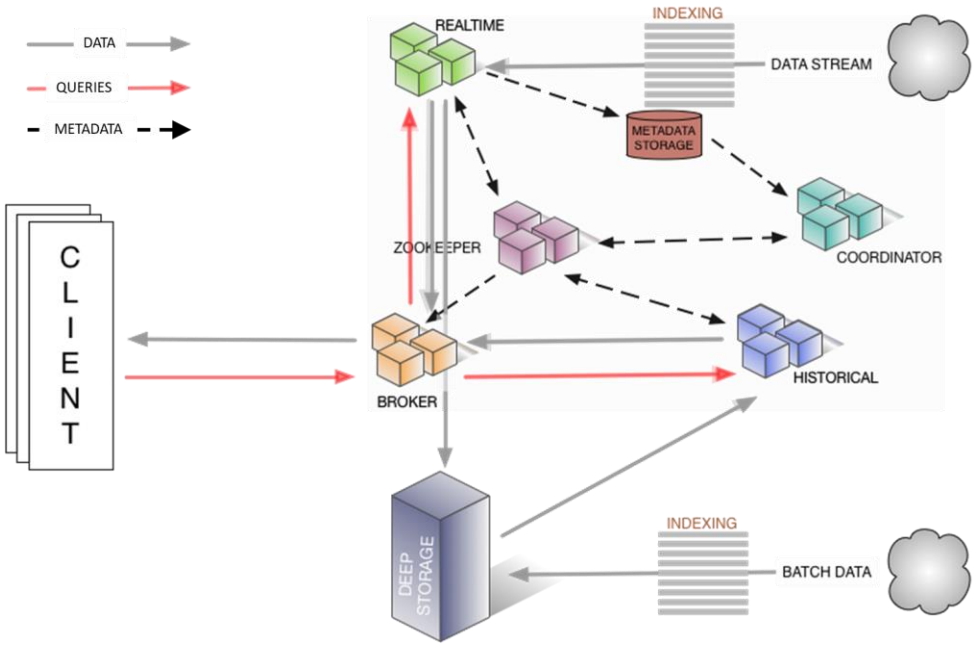


Figura 22 - Visão Geral de um cluster Druid. Baseado em (Druid, 2018d).

Os ***Real-time nodes*** encapsulam a funcionalidade de ingerir, consultar e criar segmentos de dados a partir de *streams* de eventos. Os eventos indexados através destes *nodes* ficam de imediato disponíveis para serem consultados. Estes *nodes* só estão preocupados com os eventos que ocorrem numa janela temporal relativamente pequena (exemplo: horas) e periodicamente entregam *batches* de eventos imutáveis, que recolheram durante esta janela temporal, para os *historical nodes*, especializados em lidar com este tipo de dados. Os *nodes* reportam o seu estado e os dados que servem, utilizando o Zookeeper[28] (Yang et al., 2014, 2017).

Os *Real-time nodes* empregam uma *log structured merge tree*. Estes *nodes* mantêm um *in-memory index buffer* para todos os eventos que vão sendo recebidos, sendo que estes índices são incrementais e diretamente consultáveis. Periodicamente, ou quando um número máximo de linhas (parametrizável)

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

é atingido, os *real-time nodes* persistem os seus *in-memory indexes* para o disco, num processo que converte os dados armazenados neste buffer para segmentos orientados a colunas, descritos adiante. Os índices armazenados são carregados para *off-heap memory* para que possam continuar a ser consultáveis. Durante todo este processo, ilustrado na Figura 23, os dados estão sempre disponíveis para consulta (Yang et al., 2014, 2017).

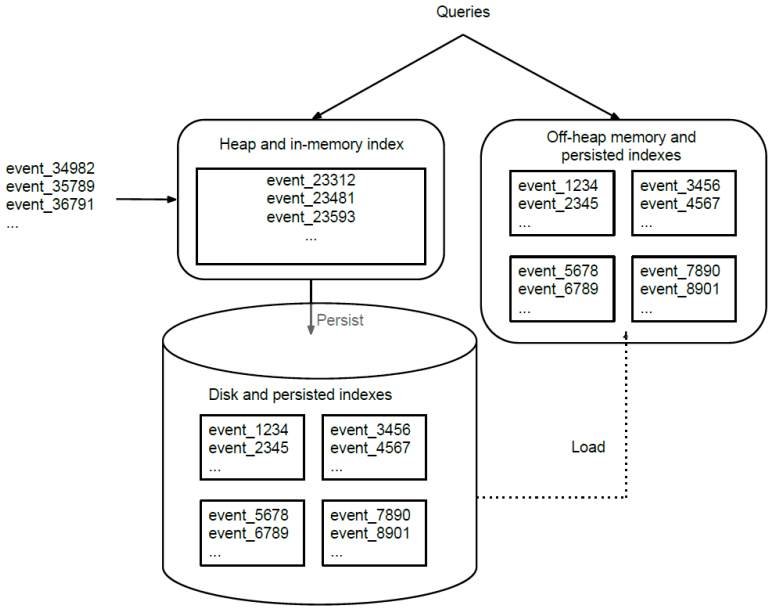


Figura 23 - Processo de persistir os índices em memória. Retirado de (Yang et al., 2014).

Periodicamente, cada *real-time node,* agenda uma tarefa que procura por todos os índices persistidos localmente e os integra, construindo um bloco imutável de dados, denominado por segmento e que contém todos os eventos que foram ingeridos por este *node*, durante um período de tempo. Durante a fase de entrega (*handoff stage*), o *real-time node* carrega este segmento para um armazenamento permanente de dados, normalmente um sistema de ficheiros distribuídos como o HDFS, que o Druid denomina por *deep storage* (Yang et al., 2014, 2017).

A Figura 24 evidencia o conjunto de operações realizadas por um *real-time node*. No exemplo, o *node* inicia às 13:37 e anuncia que irá servir o segmento de dados para o intervalo entre as 13:00 e as 14:00 horas. Este intervalo de uma hora é designado *segment granularity* e é configurável. A cada 10 minutos o *node* irá persistir os seus *in-memory indexes* para o disco. Este intervalo de 10 minutos denomina-se *window period* e também é parametrizável. Perto do final da hora corrente, quando o *node* começar a receber eventos pertencentes ao intervalo entre as 14:00 e as 15:00 horas, o *node* prepara-se para receber dados da próxima hora, criando um novo *in-memory índex* e anunciando que está a servir o segmento de dados entre as 14:00 e as 15:00 horas. Às 14:10, que é o fim da hora mais o *window period* (minimiza o risco de perda de dados decorrente de atrasos), o *node* junta todos os índices

persistidos entre as 13:00 e as 14:00 horas num único segmento imutável e inicia a fase de entrega dos dados para o *deep storage*. Depois, é criada uma entrada no repositório de metadados (pode ser implementado, por exemplo em MySQL ou PostgreSQL) para indicar que foi criado um segmento. Os *historical nodes* podem consultar esta entrada no repositório de metadados e transferir e servir o segmento. Finalmente, quando o segmento já está carregado e é consultável, o *node* liberta todas as informações sobre este segmento e anuncia que já não o serve (Yang et al., 2014, 2017).

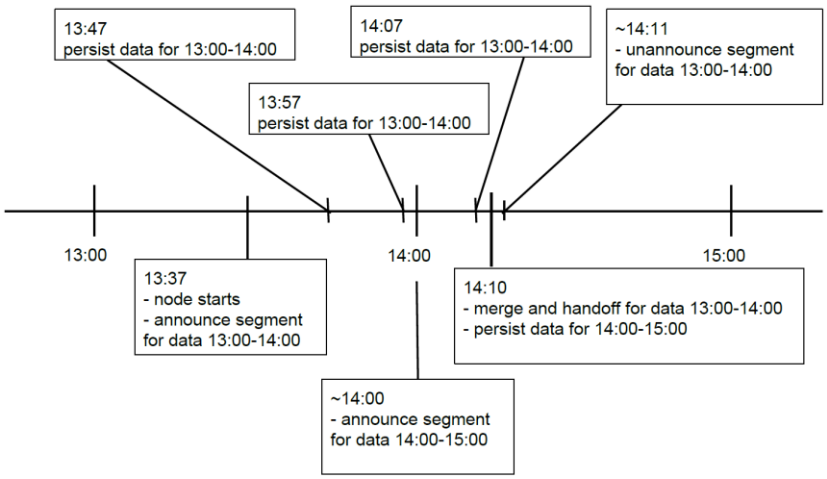


Figura 24 – Operações de um real-time node. Retirado de (Yang et al., 2014).

Os ***Historical nodes*** são dedicados a servir dados históricos. Estes *nodes* encapsulam a funcionalidade de carregar e servir segmentos imutáveis de dados, criados pelos *real-time nodes* ou inseridos em *batch*. Quando são inseridos em *batch*, os segmentos são diretamente carregados para o *deep storage,* sendo que também são criadas entradas no repositório de metadados, por cada segmento criado. Tal como os *real-time nodes*, estes *nodes* reportam o seu estado e os dados que estão a servir ao Zookeeper. Através do Zookeeper também são enviadas instruções para carregar ou eliminar segmentos, assim como informações sobre a localização destes segmentos no *deep storage* e como os descomprimir e processar. Antes de um *historical node* transferir um segmento do *deep storage*, primeiro verifica a sua *local cache*, que mantém informação sobre os segmentos que já existem no *node*. Se verificar que a *local cache* não contém informação sobre o segmento, irá transferi-lo e, assim que o processo estiver completo, irá informar o Zookeeper que serve o segmento e este passa a ser consultável (Yang et al., 2014, 2017). Este processo encontra-se ilustrado na Figura 25.

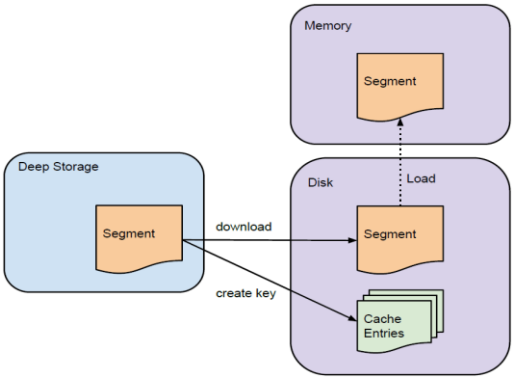


Figura 25 – Processo até disponibilizar os dados nos historical nodes. Retirado de (Yang et al., 2014).

De referir que os *historical nodes* podem ser agrupados em diferentes camadas, onde todos os *nodes* de uma camada têm uma configuração idêntica. O propósito das camadas é permitir dar maior ou menor prioridade aos segmentos, de acordo com a sua importância ou frequência com que são acedidos. Por exemplo, pode construir-se uma camada com melhores recursos computacionais para servir segmentos de dados que são frequentemente acedidos e uma camada com menores recursos computacionais para servir segmentos menos consultados (Yang et al., 2014).

Por fim, é relevante referir que o Druid, por defeito, utiliza um motor de armazenamento com uma estrutura mapeada em memória, ao invés de operar numa estrutura totalmente em memória. Desta forma, o Druid depende do sistema operativo para fazer *page in* e *page out* dos segmentos da memória. Dado que os segmentos só podem ser consultados (responder a *queries*) se estiverem carregados em memória, o motor utilizado para mapear os segmentos em memória, permite que os segmentos recentes permaneçam em memória, enquanto que aqueles que nunca são consultados são libertados da memória (*page out*). Utilizando esta abordagem, existe a limitação de quando uma *query* necessita de carregar mais segmentos para memória (*page in*) do que a capacidade do *node.* Nestes casos o desempenho será afetado pelo custo de muitas operações de *page in* e *page out* da memória. Contudo, esta abordagem permite incorrer em menos custos do que aqueles que um motor totalmente em memória acarreta (uma vez que a memória RAM é um recurso mais caro que o disco). Se o desempenho for crítico e o preço da mais elevado da infraestrutura não for um problema, o utilizador pode optar por configurar o motor do Druid para trabalhar só em memória (Druid, 2018d, 2018g; Yang et al., 2014).

Broker nodes atuam como *query routers* dos *historical* e *real-time nodes*. Os *broker nodes* compreendem os metadados publicados no Zookeeper sobre os segmentos consultáveis e onde é que estes se localizam. Estes nodes são responsáveis por fazer *route* das *queries*, de tal forma que estas consultem os corretos *historical* e *real-time nodes.* Além disso, também juntam os resultados parciais,

retornados pelos *historical* e *real-time nodes*, antes de devolver o resultado final consolidado aos utilizadores (Chambi et al., 2016; Yang et al., 2014, 2017)

Um aspeto importante destes nodes é que também possuem uma *cache*. Desta forma, sempre que o *broker node* recebe uma *query*, primeiro irá mapeá-la com um conjunto de segmentos, sendo que, os resultados para certos segmentos podem já existir em *cache*, não sendo necessário voltar a computá-los. No caso de os resultados não existirem em *cache*, o *broker node* irá encaminhar a *query* para os corretos *historical* e *real-time nodes* e no fim combinará os resultados. Quando os *historical nodes* retornarem resultados, o *broker* irá guardá-los em *cache*, para futura utilização. Os dados em *real-time* nunca são guardados em *cache*, porque não seria adequado, visto que estão constantemente a ser alterados. Este processo é demonstrado na Figura 26. De referir que a *cache* também pode atuar como um nível adicional de durabilidade dos dados, visto que, no caso dos *historical nodes* falharem, ainda será possível consultar os resultados, se estes já estiverem em *cache* (Yang et al., 2014).

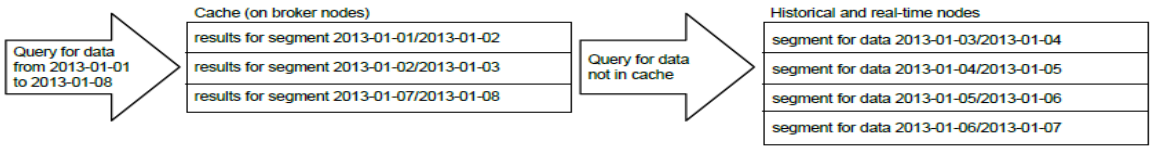


Figura 26 - Broker nodes combinam os segmentos em cache com os que não estão. Retirado de (Yang et al., 2014).

A tarefa principal dos ***Coordinator nodes*** reside em gerir a distribuição dos segmentos entre os *historical nodes* do *cluster*. Estes *nodes* aplicam um conjunto de regras e comunicam aos *historical nodes* para carregar dados, eliminar dados desatualizados, replicar dados e mover dados para equilibrar a carga do *cluster*. O Druid utiliza um *multi-version concurrency control swapping protocol*, que é responsável por eliminar algum segmento que contenha dados que são completamente obsoletos pelos novos segmentos, mantendo, assim, visualizações estáveis. Periodicamente, o *coordinator node* compara o estado atual do *cluster* com o estado expectável e, à semelhança do que acontece com os outros *nodes*, mantém uma conexão com o Zookeeper para estar ao corrente de informação sobre o *cluster*. Além disso, estes *nodes* também mantêm uma conexão com o repositório de metadados, que contém informações importantes sobre os segmentos servidos pelos *historical nodes*, assim como regras sobre como os segmentos devem ser criados, destruídos e replicados no *cluster* (Chambi et al., 2016; Yang et al., 2014, 2017).

Os ***Indexing Service nodes*** formam um *cluster* de *workers* para ingerir dados em *batch* e em *real-time* para o sistema, assim como fazer alterações aos dados já armazenados. O *Indexing Service* possui uma arquitetura *master-slave*, composta por três componentes principais: *peon,* capaz de executar uma tarefa de cada vez; *middle manager*, que gere os *peons* e um *overlord* que gere a

distribuição de tarefas pelos *middle managers.* Os *overlords* e os *middle managers* podem executar no mesmo *node* ou distribuídos por diferentes *nodes*, enquanto que os *middle managers* e os *peons* executam sempre nos mesmos *nodes.* Estes *nodes* suprem algumas das limitações apontadas aos *real-time nodes,* como por exemplo o facto de ser necessário reiniciar os *real-time nodes* para que alterações no modelo de dados tenham efeito (Druid, 2018j, 2018d).

Na Figura 27 apresenta-se uma visão geral do *Indexing Service*, na qual se pode verificar a distribuição dos vários componentes e as interações entre os mesmos.

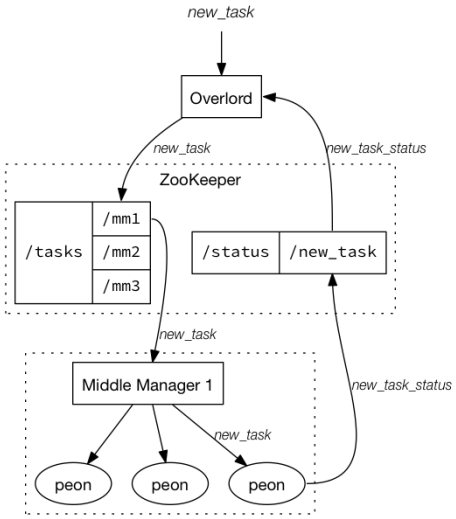


Figura 27 - Visão Geral do Indexing Service. Retirado de (Druid, 2018j).

3.2.2 Formato de Armazenamento dos Dados

As tabelas do Druid (denominadas por *data sources*) são coleções de *timestamped events*, particionados num conjunto de segmentos de dados, tipicamente possuindo entre 5 e 10 milhões de linhas. Os segmentos definem-se como coleções de registos de um certo período de tempo e representam a unidade fundamental de armazenamento no Druid, sendo que a replicação e distribuição é feita ao nível dos segmentos (Yang et al., 2014, 2017).

O Druid requer sempre uma coluna *timestamp*, como método de simplificar as políticas de distribuição de dados, as políticas de retenção de dados e um primeiro nível de poda sobre as *queries* (*query pruning*). As tabelas do Druid são particionadas em intervalos de tempo bem definidos, tipicamente uma hora ou um dia, e podem adicionalmente considerar valores de outras colunas, para atingir o tamanho desejado para o segmento (utilizando particionamento, que é explicado adiante). A granularidade temporal para particionar os segmentos (*segment granularity*) é uma função que considera o volume dos dados e o intervalo de tempo. Uma fonte de dados com *timestamps* espalhados ao longo

do ano é melhor particionada por dia, enquanto que uma fonte de dados com *timestamps* espalhados ao longo de um dia é melhor particionada por hora (Yang et al., 2014, 2017).

Os segmentos são armazenados num formato colunar, visto que guardar informação agregada em colunas, ao invés de linhas, traz várias vantagens de desempenho, como por exemplo o facto do motor de *queries* apenas ler as colunas necessárias para responder a uma *query*. Uma das características dos segmentos é o facto de serem imutáveis, o que permite consistência e que múltiplas *threads* possam utilizar o mesmo segmento em simultâneo, auxiliando a alcançar um melhor desempenho de leitura (Yang et al., 2014, 2017).

Os segmentos no Druid são identificados por um identificador da tabela (*data source*), o intervalo de tempo dos dados e a versão (que incrementa quando um novo segmento é criado). Estes metadados são utilizados pelo sistema para fazer o controlo de versões anteriormente referido, no qual os segmentos com versões mais recentes que servem dados no mesmo intervalo de tempo que os segmentos com versões mais antigas, os substituem.

Os formatos dos dados do Druid consistem em três tipos diferentes (Chambi et al., 2016; Yang et al., 2017):

- ***Timestamps*** – representa a data do evento e são armazenados em colunas com do tipo *long*.
- **Dimensões** – atributos utilizados pelas *queries* OLAP para filtrar os dados. São equivalentes às dimensões dos *Data Warehouses* tradicionais e são armazenadas em colunas do tipo *string*.
- **Métricas** – são equivalentes às métricas dos *Data Warehouses* tradicionais, contendo valores numéricos utilizados pelas *queries* OLAP para fazer agregações e cálculos (exemplo: SUM, COUNT, AVG...). Podem ser armazenadas em colunas do tipo *int*, *float*, *long* ou *double*.

Dependendo do tipo de coluna, diferentes métodos de compressão podem ser utilizados para reduzir o custo de armazenar a coluna em memória ou no disco (Yang et al., 2017).

3.2.3 Query Granularity

Uma das características do Druid é a sua capacidade para fazer agregações sobre os dados durante a sua ingestão, num processo denominado de *roll-up*. Esta funcionalidade permite reduzir as necessidades de armazenamento, visto que os dados agregados constituem uma redução no número de registos que são necessários armazenar. A documentação refere que esta redução pode atingir um fator 100, em relação aos dados originais (Druid, 2018e).

Contudo, esta redução na necessidade de armazenar tantos registos possui o custo de se perder o detalhe sobre os dados. Á medida que os dados são mais agregados, o detalhe disponível sobre os mesmos é cada vez menor, ou seja, a granularidade definida para a agregação constituiu a granularidade mínima a que será possível analisar os dados. A definição da granularidade a utilizar para agregar os dados durante a sua ingestão é feita através da propriedade "queryGranularity", sendo que a granularidade mínima suportada é o milissegundo (Druid, 2018e).

Na Figura 28 demonstra-se um exemplo de aplicação desta funcionalidade com a propriedade "queryGranularity" definida como hora.

Timestamp	productName	quantity
2018-01-01T10:11:45Z	Product x	3
2018-01-01T10:11:45Z	Product y	2
2018-01-01T10:31:22Z	Product y	5
2018-01-01T10:55:59Z	Product y	1
2018-01-01T11:11:24Z	Product x	4
2018-01-01T11:25:07Z	Product x	9

Timestamp	productName	quantity
2018-01-01T10	Product x	3
2018-01-01T10	Product y	8
2018-01-01T11	Product x	13

Figura 28 - Aplicação da Query Granularity hora.

3.2.4 Particionamento dos Segmentos de Dados

Os segmentos de dados do Druid, tal como já foi mencionado, são sempre particionados baseados no atributo *timestamp*, contudo, podem ser combinadas outras estratégias para particionar os segmentos, de tal forma que atinjam o tamanho desejado (Druid, 2018b).

O Druid suporta particionamento *hashed* (baseado no *hash* das dimensões de cada linha) e particionamento por dimensão (baseado nos diferentes valores que uma dimensão pode assumir). O primeiro tipo é recomendado pela documentação, visto que melhora o desempenho na indexação dos dados e gera segmentos de tamanhos mais uniformes, em comparação com o segundo tipo de particionamento (Druid, 2018b).

Na Figura 29 apresenta-se um exemplo da aplicação de *hashed partitions.* Tal como se verifica na figura, foi aplicada a propriedade "numShards=3" que divide o segmento de dados original em 3 *shards* semelhantes. De referir que os *shards* possuem as mesmas características que os segmentos e

são tratados pelo Druid da mesma forma. Na prática, o segmento de dados original foi dividido em 3 segmentos que servem dados do mesmo intervalo de tempo ("Monday", por exemplo). Contudo, por uma questão de nomenclatura e distinção entre conceitos, nesta dissertação, refere-se segmento de dados quando se pretende abordar os segmentos que não foram particionados em vários segmentos para o mesmo intervalo de tempo e refere-se *shards* quando se pretende abordar os vários segmentos particionados que foram gerados a partir dos segmentos originais (Druid, 2018b, 2018m).

Figura 29 - Exemplo da Aplicação de Hashed Partitions.

3.2.5 Filtros

As *queries* OLAP frequentemente perguntam por totais agregados de certas métricas, filtradas por algumas dimensões, durante um intervalo de tempo. Para otimizar a resposta a este tipo de *queries* o Druid utiliza índices. Na Tabela 8 apresenta-se uma amostra de dados que será útil para o exemplo que será apresentado de seguida.

Tabela 8 - Exemplo de dados. Retirado de (Yang et al., 2017).

Timestamp	Publisher	Advertiser	Gender	City	Click	Price
2011-01-01T01:01:35Z	bieberfever.com	google.com	Male	San Francisco	0	0.65
2011-01-01T01:03:63Z	bieberfever.com	google.com	Male	Waterloo	0	0.62
2011-01-01T01:04:51Z	bieberfever.com	google.com	Male	Calgary	1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	Taiyuan	0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	New York	0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	Vancouver	1	1.53

Um exemplo de *query* à Tabela 8, pode ser "Quantos cliques ocorreram e que receita foi gerada pelos *publishers* bieberfever.com e ultratrimfast.com?". Para cada *publisher* é possível identificar num *array* salientando em que linhas da tabela o *publisher* em questão está presente. Esta informação pode ser guardada num *array* de binários, no qual os índices do *array* representam as linhas da tabela. Se um

publisher em particular está presente na linha da tabela, o índice do *array* para a linha em questão é marcado com o valor 1. Esta abordagem, demonstrada na Figura 30, de mapear os valores das colunas para índices de linhas, forma os índices invertidos (Yang et al., 2014, 2017).

```
bieberfever.com   -> rows [0, 1, 2] -> [1][1][1][0][0][0]
ultratrimfast.com -> rows [3, 4, 5] -> [0][0][0][1][1][1]
```

Figura 30 - Índice invertido. Retirado de (Yang et al., 2017).

Para responder à *query* referida no exemplo o Druid tem de agregar duas métricas, sendo que, não é necessário analisar toda a coluna das métricas, fruto da utilização dos índices. Por exemplo, para o publisher "bieberfever.com" apenas será necessário considerar as linhas 0, 1 e 2 na coluna das métricas (Yang et al., 2014, 2017). Tipicamente, o Druid utiliza ainda algoritmos de compressão de *bitmaps,* nomeadamente os algoritmos Concise e Roaring (Chambi et al., 2016).

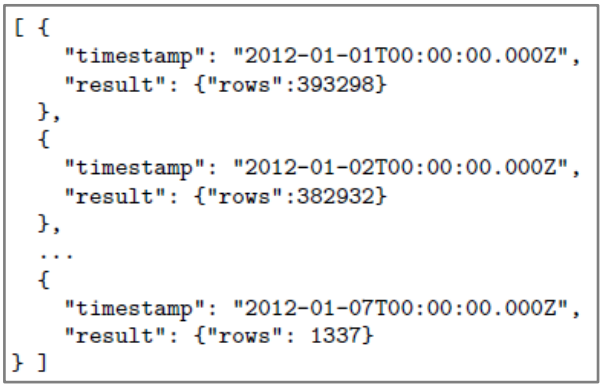
3.2.6 *Query* API

O Druid tem a sua própria linguagem e aceita as *queries* como pedidos POST. O corpo do pedido POST é um objeto JSON, contendo pares chave-valor que especificam os vários parâmetros das *queries*. Tipicamente, uma *query* contém o nome da tabela, a granularidade, o intervalo de tempo de interesse, o tipo de pedido e as métricas utilizadas para fazer as agregações. As *queries* podem ainda conter filtros, sendo que, quando são especificados filtros, apenas o subconjunto dos dados que pertence ao filtro será analisado. Os resultados da *query* também serão um objeto JSON, contendo as métricas agregadas, num intervalo de tempo (Yang et al., 2014). A Figura 31 apresenta um exemplo de uma *query,* do tipo *timeseries*, que retorna a contagem de linhas, na tabela "wikipedia", entre 2013-01-01 e 2013-01-08, filtradas pelas linhas cujo valor da dimensão "page" equivale a "Ke$ha".

```
{
  "queryType"    : "timeseries",
  "dataSource"   : "wikipedia",
  "intervals"    : "2013-01-01/2013-01-08",
  "filter"       : {
                     "type" : "selector",
                     "dimension" : "page",
                     "value" : "Ke$ha"
                   },
  "granularity"  : "day",
  "aggregations" : [{"type":"count", "name":"rows"}]
}
```

Figura 31 - Exemplo de query ao Druid. Retirado de (Yang et al., 2014).

Os resultados da *query* são integrados num *array* JSON, como se apresenta na Figura 32.



```
[ {
    "timestamp": "2012-01-01T00:00:00.000Z",
    "result": {"rows":393298}
  },
  {
    "timestamp": "2012-01-02T00:00:00.000Z",
    "result": {"rows":382932}
  },
  ...
  {
    "timestamp": "2012-01-07T00:00:00.000Z",
    "result": {"rows": 1337}
} ]
```

Figura 32 - Resultado da query exemplo ao Druid. Retirado de (Yang et al., 2014).

O Druid suporta muitos tipos de agregações, incluindo somatórios, mínimos, máximos, agregações javaScript e agregações complexas, como por exemplo estimativa de cardinalidade e estimativa de quantis aproximados, sendo que os resultados das agregações podem ser combinados em expressões matemáticas para formar outras agregações (Druid, 2018a; Yang et al., 2014). Além de agregações durante a ingestão de dados o Druid suporta ainda agregações executadas nas *queries*, denominadas de *Post-Aggregations* (Druid, 2018l).

Importa referir que, recentemente, o Druid adicionou a funcionalidade SQL, providenciada através de um *parser* e de um *planner* baseado no Apache Calcite[29], embora deixem a nota que esta é ainda uma versão experimental. Sendo assim, esta versão não suporta algumas das funcionalidades suportadas pelo Druid, como por exemplo DataSketches[30] *aggregators*. O oposto também se verifica, ou seja, o Druid não suporta algumas funcionalidades SQL, como por exemplo *joins* (Druid, 2018n).

3.2.7 Extensões ao Druid

O Druid implementa um sistema de extensões que permite adicionar funcionalidades em tempo de execução. As extensões são normalmente utilizadas para adicionar o suporte para o *deep storage* (HDFS ou S3, por exemplo), repositórios de metadados (MySQL ou PostgreSQL, por exemplo), novas agregações, novos formatos de dados de entrada (ORC e Avro, por exemplo), entre outros. Clusters produtivos utilizam, no mínimo, duas extensões: uma para o *deep storage* e outra para o repositório de metadados (Druid, 2018f).

As extensões do Druid dividem-se em dois grupos: extensões *core* (são mantidas atualizadas pelos contribuidores do Druid) e extensões propostas pelos membros da comunidade do Druid (não são

[29] https://calcite.apache.org/
[30] http://druid.io/docs/latest/development/extensions-core/datasketches-aggregators.html

mantidas pelos contribuídos do Druid e podem ter passado por menos testes de validação) (Druid, 2018f).

De entre as extensões propostas pela comunidade existem algumas que se destacam, proporcionando ferramentas de visualização para analisar os dados do Druid (Superset e Pivot, por exemplo) ou clientes para fazer *queries* sobre os dados armazenados no Druid, implementados em diferentes linguagens, nomeadamente: python, R, SQL, java, scala, php, entre outras (Druid, 2018c).

O Tranquility[31] é outro projeto relevante, uma vez que auxilia na ingestão de dados no Druid, via *streaming* (Druid, 2018o).

3.3 Tecnologias de Possível Integração com o Druid

Nesta dissertação é estudado o Druid no processamento analítico de dados em contextos de BDW e RTBDW, como tal, é necessário estudar as tecnologias que poderão integrar com o Druid, numa arquitetura capaz de responder aos requisitos que se exigem. Desta forma, apresentam-se nesta secção um conjunto de tecnologias, divididas nas seguintes categorias: processamento *streaming* (Tabela 9), SQL-on-Hadoop (Tabela 10) e visualização de dados (Tabela 11).

Tabela 9 - Tecnologias para o Processamento em Streaming.

Tecnologia	Descrição
Kafka	Sistema que permite a publicação e subscrição de mensagens em *real-time.* As mensagens são associadas a partições distribuídas pelos vários nós de um *cluster*, de forma a obter tolerância a falhas. Um cluster Kafka retém as mensagens publicadas durante um tempo configurável, a partir do qual são descartadas (C. L. P. Chen & Zhang, 2014; Liu et al., 2014).
Flume	É um sistema distribuído, confiável e disponível, que recolhe, agrega e transfere grandes quantidades de dados de eventos (normalmente dados de logs) de forma eficiente, tipicamente para o HDFS (C. L. P. Chen & Zhang, 2014; Liu et al., 2014).
Spark Streaming	Segundo os autores esta é uma tecnologia que torna mais fácil o processamento de aplicações *streaming*, de uma forma escalável e tolerante a falhas. Além disso, possui integração com o Spark, permitindo assim combinar o processamento *streaming* e *batch* e com *queries* interativas (Apache Hadoop, 2018; Apache Spark, 2018b; Liu et al., 2014).
Storm	É um sistema de computação em *real-time* para processar *streams* com baixa latência, garantindo que todos os dados são processados. É também caracterizado por ser

[31] https://github.com/druid-io/tranquility

	escalável e tolerante a falhas (C. L. P. Chen & Zhang, 2014; Liu et al., 2014; Marz & Warren, 2015).

Tabela 10 - Tecnologias SQL-on-Hadoop.

Tecnologia	Descrição
Hive (LLAP)	O Hive LLAP combina um mecanismo inteligente de *in-memory caching* com servidores de *query* persistentes, de forma a providenciar rápidas respostas às *queries* SQL, sem sacrificar a escalabilidade pela qual o Hive e o Hadoop são conhecidos (Hortonworks, 2018c).
Presto	É um motor de *queries* SQL distribuído, *open-source* e que é utilizado para executar *queries* analíticas interativas a grandes conjuntos de dados. Esta tecnologia permite fazer *queries* sobre distintas fontes de dados, incluindo o Hive, Cassandra, bases de dados relacionais e até bases de dados proprietárias (Presto, 2016).
Impala	É um sistema de *ad-hoc querying* escalável, que providencia uma linguagem semelhante ao SQL para executar *queries* sobre os dados armazenados no HDFS e HBase. Esta tecnologia processa os dados em memória o que permite retornar os resultados das *queries* rapidamente (Liu et al., 2014).
Spark	Trata-se de um motor de *analytics* para o processamento de grandes quantidades de dados, e que é capaz de combinar SQL, processamento *streaming*, *machine learning* e outras operações analíticas complexas. O *Spark* suporta ainda o processamento *in-memory,* de forma a suportar melhores desempenhos (Apache Spark, 2018a; Liu et al., 2014).

Tabela 11 - Tecnologias para a Visualização de Dados.

Tecnologia	Descrição
Superset	É uma plataforma *web* de suporte à componente de análise e visualização de dados em sistemas de *business intelligence*. Possui algumas funcionalidades relevantes, como por exemplo: criação e partilha de *dashboards*, utilizando as várias visualizações disponibilizadas, definições de segurança e permissões de acesso, integração com repositórios de dados SQL, através do SQLAlchemy. Possui ainda uma forte integração com o Druid (Apache Superset, 2018).
Tableau	Trata-se de uma ferramenta de visualização que torna a exploração dos dados mais intuitiva e que é otimizada para disponibilizar aos utilizadores todas as colunas provenientes dos dados, permitindo-lhes juntá-las (diferentes tabelas, por exemplo). Possui conectores para diversas tecnologias, como por exemplo o Hive (C. L. P. Chen & Zhang, 2014).
Pivot	É uma tecnologia que permite análises exploratórias sobre dados de eventos, permitindo efetuar operações OLAP, de forma intuitiva, à medida que os dados vão chegando. O Pivot foi concebido para tirar o máximo de vantagem das funcionalidades do Druid. Esta tecnologia começou por ser um projeto *open-source* e foi desenvolvida

	recorrendo ao projeto *open-source* Plywood, no entanto tornou-se uma ferramenta comercial (Imply, 2018).

3.4 Protocolo de Testes

Primeiramente, na secção 1.2 desta dissertação é possível identificar objetivos de índole concetual e experimental. Os objetivos do primeiro tipo já foram sendo abordados ao longo deste capítulo, na qual foi efetuado um enquadramento tecnológico do Druid e das tecnologias que lhe surgem habitualmente associadas. Contudo, para alcançar os objetivos de índole experimental e para melhor sustentar os objetivos teóricos, é necessário definir como será levada a cabo a vertente experimental desta dissertação.

Assim, a fase experimental estará dividida em dois aspetos principais, apresentados de seguida:

- Druid para Processamento Analítico de Dados em BDW – avaliação e estudo do comportamento do Druid em contextos de *Big Data Warehousing*, vocacionados para o processamento de vastos volumes de dados históricos (processamento em *batch*);
- Druid para Processamento Analítico de Dados em RTBDW - avaliação e estudo do comportamento do Druid em contextos de *Real-Time Big Data Warehousing*., no qual dados os dados recebidos têm de ser processados à medida que são recolhidos (processamento em *streaming*).

3.5 Infraestrutura de Testes

Para efetivar o protocolo de testes previamente apresentado, e que inclui duas fases, foi utilizada uma infraestrutura de testes idêntica àquela que foi utilizada por E. Costa (2017), E. Costa, Costa, e Santos (2017), permitindo que seja feita uma comparação rigorosa entre o desempenho do Druid e os resultados obtidos pelo Hive e Presto, apresentados no trabalho dos autores acima nomeados. Desta forma, os contributos desta dissertação complementarão outros contributos científicos e técnicos, para além de constituir mais um avanço na investigação do projeto no qual esta dissertação se integra.

Desta forma, a infraestrutura de testes consiste num cluster Hadoop de 5 *nodes* (1 HDFS *NameNode* e 4 HDFS *Data Nodes*), cada um dos quais com as seguintes características de *hardware*:

- 1 Intel Core i5, quad core, com *clock speed* entre 3.1-3.3 GHz;

- 32 GB de DDR3 *Random Access Memory* (RAM), com *clock speed* de 1333 MHz e 24 GB disponível para processamento de queries;
- 1 Samsung 850 EVO 500GB Solid State Drive (SSD) com uma capacidade de leitura de 540 MB/s e uma capacidade de escrita de 520 MB/s;
- 1 *gigabit Ethernet card* conectado através de cabos *Cat5e Ethernet* e um *gigabit Ethernet switch*.

O sistema operativo utilizado em todos os *nodes* do cluster é o CentOS 7 com um sistema de ficheiros XFS. A distribuição do Hadoop em utilização foi atualizada para a versão Hortonworks Data Platform (HDP) 2.6.4, por disponibilizar uma versão mais recente e estável do Druid, face àquela que era utilizada por E. Costa (2017) e E. Costa et al. (2017). As configurações utilizadas por estes autores não foram modificadas, ou seja, a distribuição do Hadoop em utilização continua com as configurações por defeito e mantem-se um fator de replicação 2.

Por fim, foi instalada a versão 0.10.1 do Druid e a versão 0.15.0 do Superset, via Ambari, sendo que todas as configurações foram deixadas por defeito na instalação. Mais tarde, para responder à componente experimental desta dissertação foi necessário adicionar ou alterar algumas propriedades no Druid ou noutros serviços com os quais se relaciona (ex: Map Reduce, YARN, Hive), sendo que estas alterações serão apresentadas e explicadas no capítulo 4. De referir que os componentes do Druid e do Superset foram distribuídos pelos vários *nodes* do cluster, da forma que está ilustrada na Figura 33. No *NameNode* ficaram alojados o Druid *Broker*, o Druid *Coordinator*, o Druid *Overlord,* o *Druid Router* e o Superset enquanto que os Druid *Historicals* e os Druid *MiddleManagers* ficaram distribuídos pelos *DataNodes* do cluster.

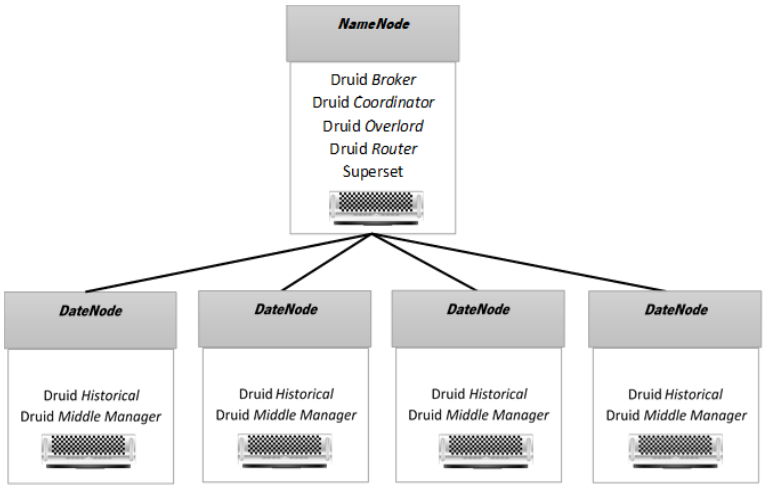


Figura 33 - Distribuição dos Componentes do Druid e do Superset pelo Cluster.

4. DRUID PARA PROCESSAMENTO ANALÍTICO DE DADOS EM BDW

Neste capítulo pretende-se avaliar o Druid no processamento de dados em contextos de *Big Data Warehousing* com vastos volumes de dados históricos e perceber de que forma a modelação e o particionamento dos dados afetam o seu desempenho. Também será objetivo estudar e aferir a integração do Druid com o Hive e as vantagens que dela advêm. Além disso, considera-se relevante comparar o desempenho obtido, com aquele que pode ser alcançado recorrendo a outras tecnologias SQL-on-Hadoop. Neste sentido, novamente se realça que será efetuada uma comparação com os resultados do Hive (*On* Tez) e do Presto, retirados de E. Costa (2017) e E. Costa et al. (2017). Para todos os cenários de teste apresentados neste capítulo será utilizado o *Star Schema Bechmark* (SSB).

Este capítulo inicia-se com a apresentação do conjunto de dados utilizado e a exposição e explicação dos cenários de teste que serão utilizados. Seguidamente, apresentam-se os resultados obtidos nos vários cenários. Tal como já foi referido, também se abordará o potencial da integração entre o Hive e o Druid, assim como o comportamento destas tecnologias em ambientes multiutilizador. Por fim, é efetuada uma análise dos resultados face ao tipo de queries utilizadas e uma síntese geral de resultados.

Os resultados obtidos serão utilizados como base para identificar em que cenários a utilização do Druid é mais vantajosa e para definir algumas recomendações de utilização, a fim de otimizar a utilização e o desempenho desta tecnologia.

4.1 Conjunto de Dados

Tal como foi anteriormente referido, o conjunto de dados utilizado para efetuar os testes necessários é o *Star Schema Benchmark* (SSB), à semelhança do que é feito na parte experimental do trabalho de E. Costa (2017) e E. Costa et al. (2017).

O SSB consiste num DW tradicional de vendas modelado de acordo com estruturas multidimensionais (estrelas). Além disso, este *benchmark* inclui treze *queries* utilizadas para avaliar o desempenho no processamento analítico de dados (OLAP) (O'Neil, O'Neil, & Chen, 2009). Importa referir que o SSB é baseado no TPC-H[32], com algumas modificações propostas por O'Neil et al. (2009), fundamentadas em muitos dos princípios de Kimball e Ross (2013) e que estão relacionadas sobretudo

[32] http://www.tpc.org/tpch/

com a transformação do esquema relacional do TPC-H num esquema em estrela. Contudo, algumas destas alterações não foram implementadas, uma vez que estas não foram utilizadas nos trabalhos de E. Costa (2017) e E. Costa et al. (2017), sendo necessário manter o mesmo cenário de teste para que seja possível a comparação entre as diversas tecnologias e entre os resultados daqueles estudos e os apresentados nesta dissertação. Neste contexto, não foi efetuada a diminuição do *scale factor* das dimensões CUSTOMER e SUPPLIER (apresentadas de seguida) e a dimensão temporal utilizada inclui menos atributos do que aqueles que são definidos por O'Neil et al. (2009).

A Figura 34 apresenta o modelo de dados em estrela que serviu de base ao *benchmark* conduzido por E. Costa (2017) e E. Costa et al. (2017), e o processo de desnormalização que originou os 2 modelos de dados utilizados nos vários testes apresentados neste capítulo: o modelo com todos os atributos disponíveis no esquema de dados e o modelo com o subconjunto dos atributos necessários para responder às *queries* do SSB. A utilização destes dois modelos permitirá comparar os resultados aqui obtidos com os resultados obtidos por E. Costa (2017) e E. Costa et al. (2017), visto que estes autores utilizaram o primeiro modelo, com todos os atributos disponíveis. No entanto, e para perceber o impacto que o modelo de dados tem no Druid, ao nível dos atributos disponíveis, este trabalho analisa um cenário alternativo, comparando os resultados alcançados no primeiro e no segundo modelo.

O modelo de dados do SSB inclui uma tabela de factos, LINEORDER, ligada a 4 tabelas de dimensão, CUSTOMER, SUPPLIER, DATE e PART. Realça-se que a desnormalização da tabela "LINEORDER", apresentada na Figura 34, originou 53 atributos (no modelo com todos os atributos), devido aos dois relacionamentos que a tabela "DATE" mantém com a tabela LINEORDER: 9 dos quais por intermédio do atributo "Orderdate" (prefixo "od_") e outros 9 por intermédio do atributo "Commitdate" (prefixo "cd_"). Existem ainda 11 atributos provenientes apenas da tabela "LINEORDER" que não possuem qualquer prefixo. Os restantes atributos, oriundos das tabelas "CUSTOMER", "SUPPLIER" e "PART" apresentam os prefixos "c_", "s_" e "p_", respetivamente.

No que diz respeito à criação do modelo com os atributos necessários, ilustrado na Figura 34, o processo passou por uma análise das *queries* do SSB, com o objetivo de se gerar um modelo orientado a responder a todas elas, sem considerar atributos adicionais não utilizados por estas. A única exceção é o "od_datestandard" que não é utilizado nas queries do SSB, mas é necessário porque os segmentos de dados no Druid requerem sempre um *timestamp*.

Neste trabalho, o SSB *benchmark* é utilizado com diferentes *Scale Factors* (SFs), nomeadamente 30, 100 e 300. O SF influencia o tamanho do conjunto de dados gerado, por exemplo, um SF de 30 irá gerar uma tabela de factos ("LINEORDER") com cerca de 180.000.000 linhas, influenciado também as

restantes tabelas (calculadas de acordo com as fórmulas apresentadas na Figura 34, junto ao nome de cada tabela).

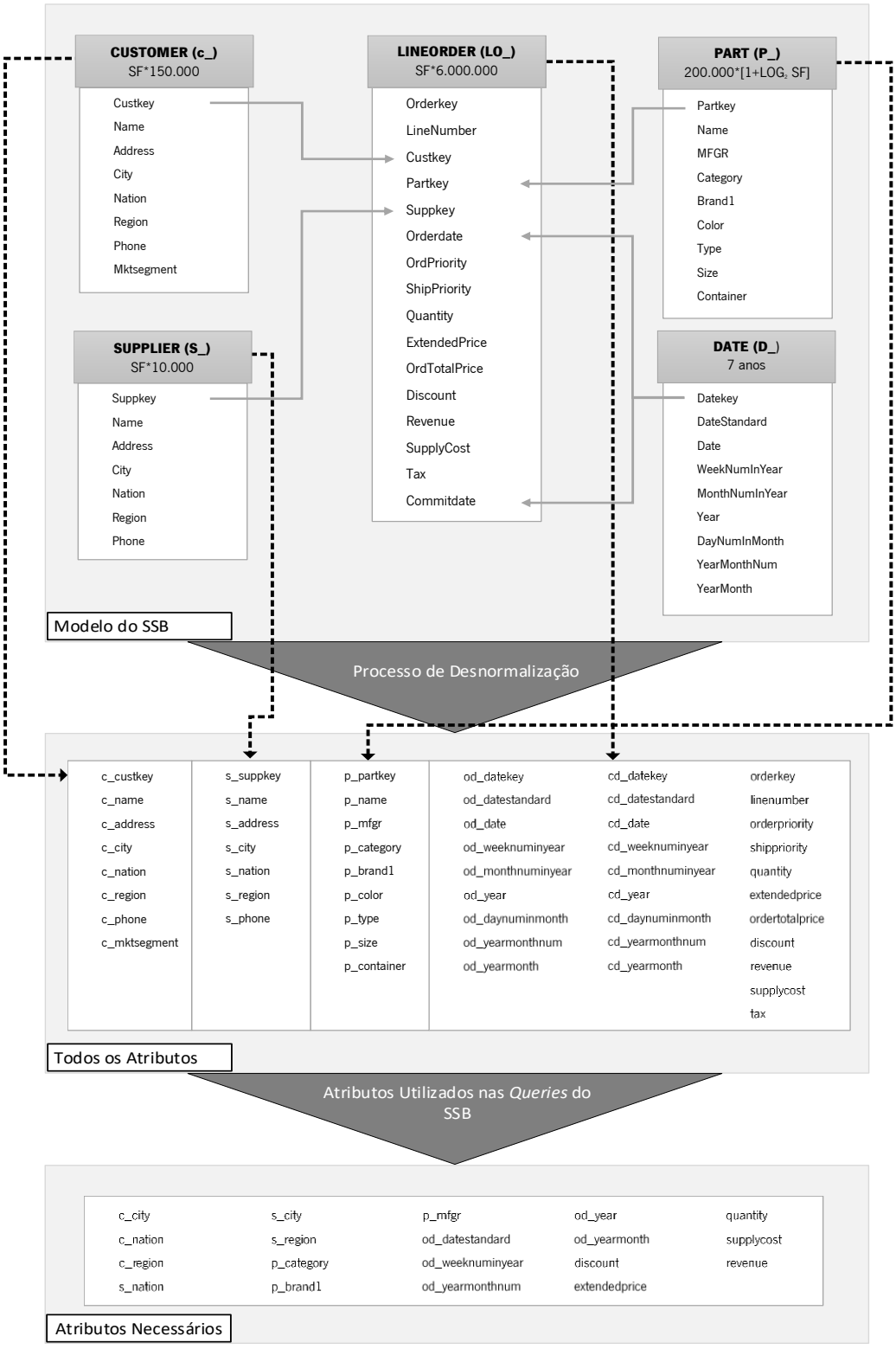


Figura 34 - Processo de Desnormalização do SSB.

Por fim, destaca-se que apenas serão utilizados modelos desnormalizados, uma vez que o Druid não suporta *joins*, pelo que não seria possível implementar o modelo em estrela do SSB (Yang et al., 2014, 2017). Apesar desta característica ser muitas vezes apontada como uma limitação do Druid, abordagens recentes na implementação de *Big Data Warehouses* têm-se focado em estruturas totalmente desnormalizadas para suportar os requisitos analíticos (Dehdouh et al., 2015; M. Y. Santos & Costa, 2016b, 2016a; M. Y. Santos, Martinho, & Costa, 2017). Dado que o Druid não suporta *joins*, neste trabalho são utilizadas as 13 queries do SSB, adaptadas para consultar modelos desnormalizados. Estas *queries* podem ser consultadas no Apêndice 1 - Queries e são designadas de Q1.1, Q1.2, Q1.3, Q2.1, Q2.2, Q2.3, Q3.1, Q3.2, Q3.3, Q3.4, Q4.1, Q4.2 e Q4.3.

4.1.1 Cardinalidade

Antes de avançar para a execução dos vários testes considera-se importante fazer uma análise exploratória dos dados, quer ao nível da sua cardinalidade (apresentada nesta subsecção), quer ao nível da sua distribuição (apresentada na subsecção seguinte). Esta análise será útil para classificar as várias *queries* do SSB (secção 4.6), assim como poderá ser um auxílio na avaliação dos resultados obtidos.

Importa referir que esta análise foi efetuada recorrendo ao modelo de dados desnormalizado, apenas para aqueles atributos que são utilizados nas *queries* do SSB. Além disso, foi considerada a amostra de dados gerada pelo SF 1 (cerca de 6.000.000 linhas). Uma vez que se trata de um *dataset* gerado automaticamente, as amostras produzidas por diferentes SFs seguem distribuições e cardinalidades semelhantes, pelo que esta amostra é suficiente para avaliar as características dos dados.

A Tabela 12 apresenta a cardinalidade dos vários atributos do modelo considerado (todos os atributos utilizados nas queries do SSB, excluindo métricas, como por exemplo: “revenue”, “extendedprice”), obtidas através de várias *queries* para calcular os valores distintos presentes em cada atributo.

Tabela 12 - Cardinalidade dos Atributos do Modelo Desnormalizado do SSB.

Atributo	Cardinalidade
c_city	225
c_nation	25
c_region	5
s_city	225
s_nation	25
s_region	5

p_mfgr	5
p_category	25
p_brand1	975
od_weeknuminyear	53
od_year	7
od_yearmonthnum	80
od_yearmonth	80

4.1.2 Distribuição

A análise da distribuição dos atributos foi realizada recorrendo à ferramenta Superset. Seguidamente, recorrendo a várias figuras será apresentada a análise de distribuição efetuada aos vários atributos, dividida pela tabela da qual os atributos advieram.

A Figura 35 apresenta vários gráficos que ajudam a verificar a distribuição dos atributos relacionados com "CUSTOMER". É possível verificar que os atributos "c_region" e "c_nation" possuem uma distribuição relativamente uniforme, com os valores a variarem entre 1.19M-1.21M (milhões de registos) e 246k-233k (mil registos), respetivamente. Quanto ao atributo "c_city", devido à sua maior cardinalidade é difícil apresentar na figura toda a distribuição que o caracteriza. Contudo, pela análise dinâmica do gráfico apresentado na figura, verifica-se que os valores variam entre 30.9k-24.2k.

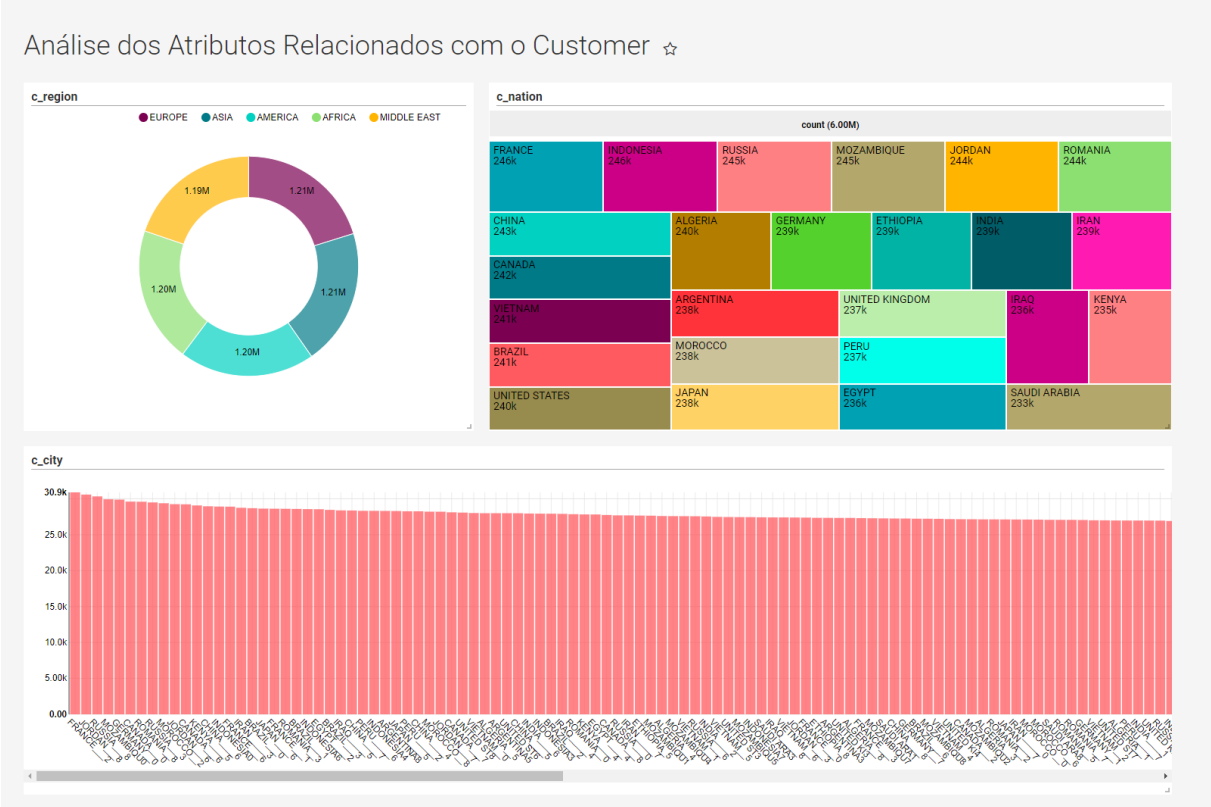


Figura 35 - Distribuição dos Atributos Relacionados com "CUSTOMER".

Na Figura 36, evidencia-se a distribuição dos atributos relacionados com "SUPPLIER", recorrendo a uma abordagem semelhante à anterior. Mais uma vez, conclui-se que o atributo "s_region" tem uma distribuição uniforme, variando entre 1.17M-1.22M. O atributo "s_nation" e o atributo "s_city" não apresentam distribuições tão uniformes quanto o atributo anterior, sendo que os seus valores variam entre 262k-217k e 35.6k-19.9k (não visível na figura, mas analisável recorrendo ao gráfico dinâmico), respetivamente.

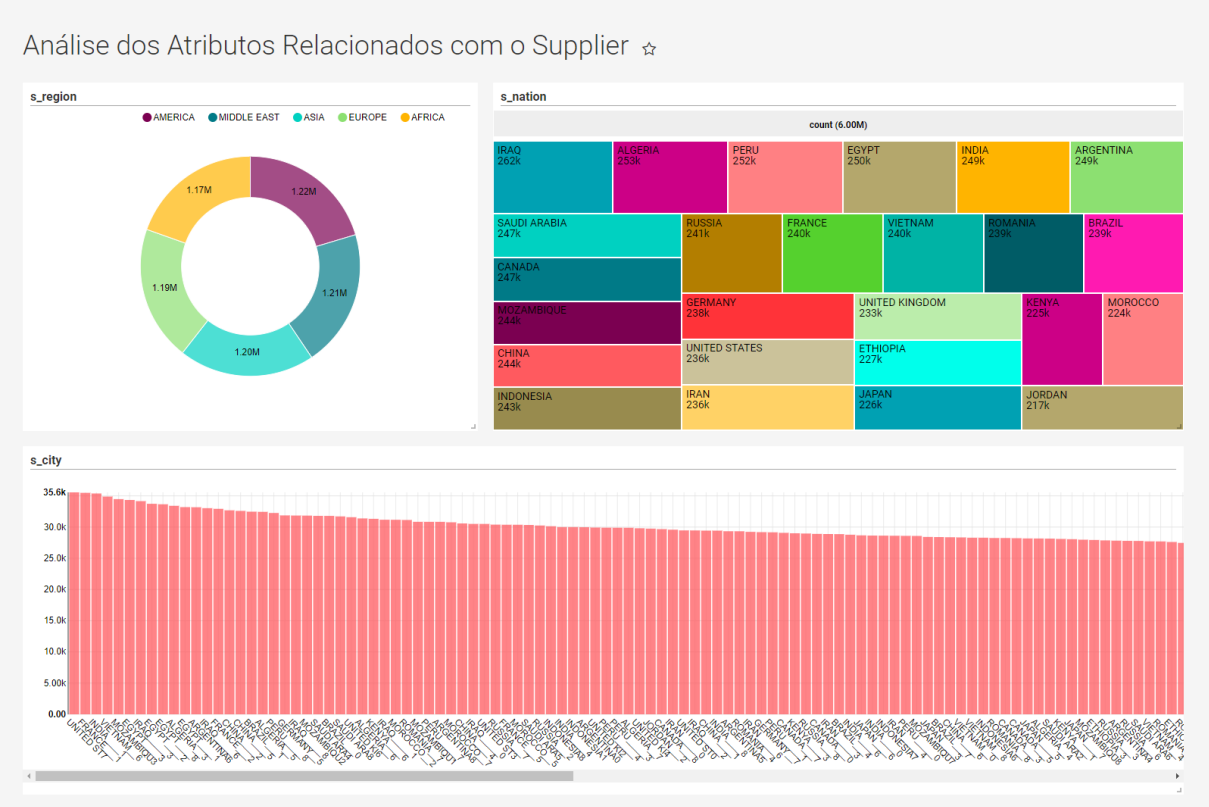


Figura 36 - Distribuição dos Atributos Relacionados com "SUPPLIER".

A Figura 37, ilustra a análise realizada aos atributos que advêm da tabela "Part". Quanto a estes atributos verifica-se que o "p_mfgr" possui uma distribuição regular entre os valores 1.19M-1.21M. O atributo "p_category" não é tão uniforme na distribuição pelos valores possíveis, uma vez este atributo se distribui no intervalo 247k-235k. Por fim, a distribuição do atributo "p_brand1" varia entre 7.66k-5.41k.

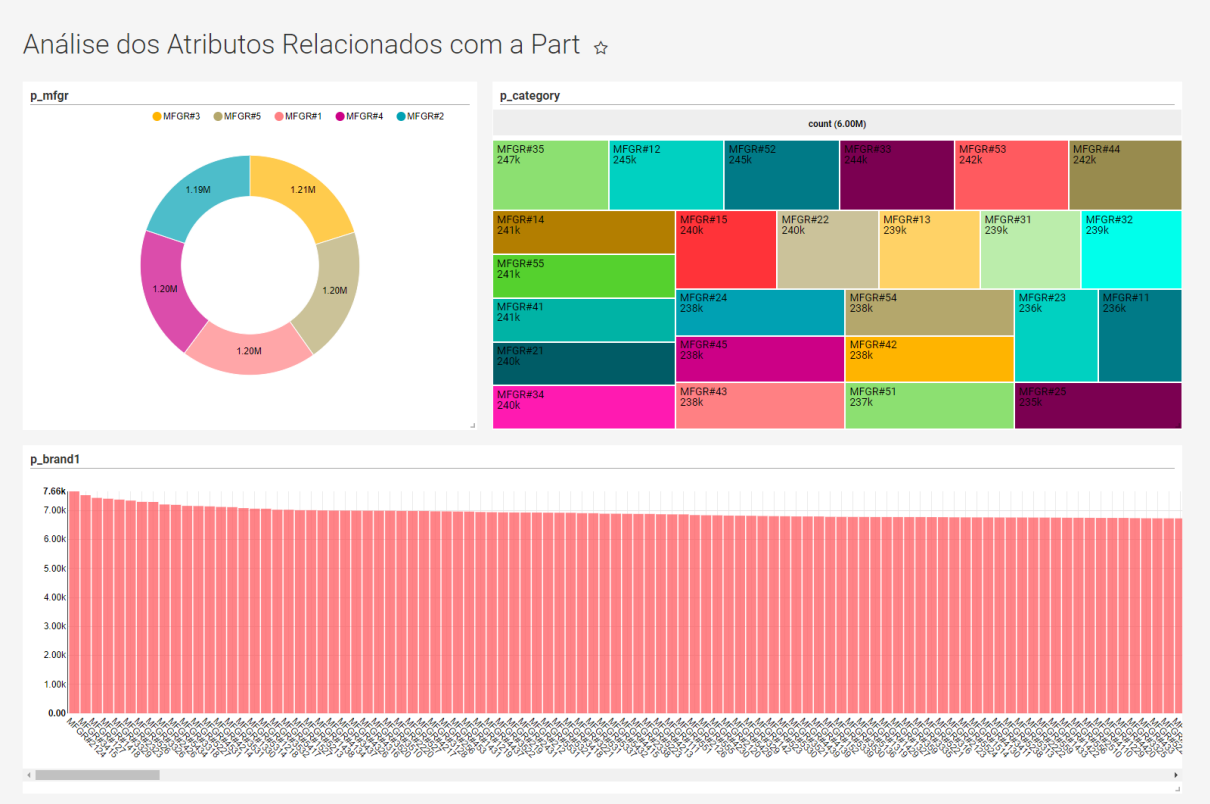


Figura 37 - Distribuição dos Atributos Relacionados com "PART".

Finalmente, realizaram-se mais algumas análises exploratórias, de forma a adquirir uma maior perceção sobre os dados base para os vários cenários de teste deste trabalho. Um exemplo destas análises exploratórias é apresentado na Figura 38, na qual se verifica a distribuição do número de linhas por mês, assim como os valores de algumas métricas.

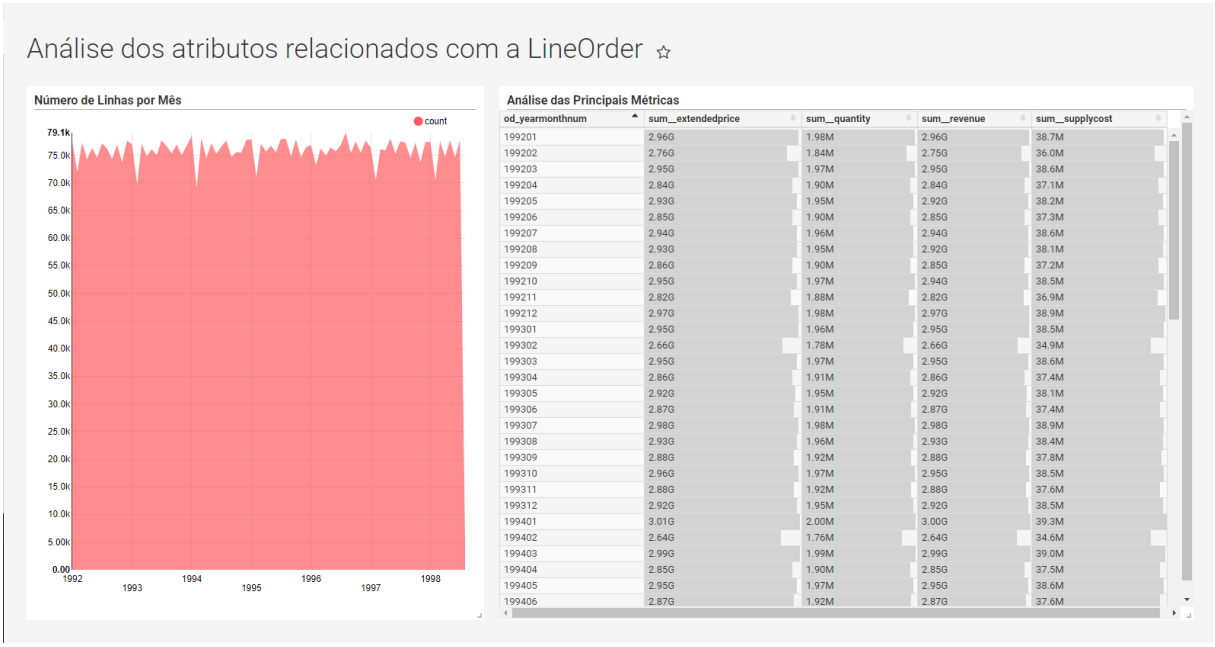


Figura 38 - Distribuição dos Atributos Relacionados com "LINEORDER".

4.2 Cenários de Teste

Neste capítulo, tal como anteriormente mencionado, pretende-se investigar o impacto que a modelação e o particionamento dos segmentos de dados têm no desempenho do Druid, quer seja no tempo de processamento das *queries*, quer seja noutros aspetos, como por exemplo, o espaço ocupado. Para efetuar esta análise aplicar-se-ão vários cenários de teste, explicados de seguida:

- **Cenário T** - grupo de cenários de teste, no qual se utiliza um modelo desnormalizado com todos os atributos. Este grupo de cenários está assim subdividido em:
    - **Cenário TS** - cenário no qual se estudará de que forma a definição do *segment granularity* impacta o Druid. Testar-se-ão as granularidades dia, mês, trimestre e semestre;
    - **Cenário TSP** - cenário no qual se analisará qual o impacto causado no Druid, pela utilização de *hashed partitions,* em junção com a definição do *segment granularity*.
- **Cenário N** - grupo de cenários de teste, no qual se utiliza um modelo desnormalizado contendo apenas os atributos necessários para responder às *queries* do SSB. Este grupo de cenários subdivide-se em:
    - **Cenário NS** - cenário semelhante ao TS, com o objetivo acrescido de analisar o impacto que a redução do número de atributos considerados no modelo tem no Druid. Também se irão utilizar as granularidades dia, mês, trimestre e semestre;
    - **Cenário NSQ** - cenário no qual se investigará de que forma é que a definição da *query granularity* impacta o Druid. Testar-se-á, como *query granularity,* o dia, a semana e o mês.
    - **Cenário NSQP** - cenário que possui o objetivo de estudar os resultados alcançados pela combinação da definição do *segment granularity*, da *query granularity* e de *hashed partitions.*

A definição das siglas atribuídas à identificação dos cenários segue a seguinte nomenclatura: T (**T**odos os Atributos), N (Atributos **N**ecessários), S (***S****egment Granularity*), Q (***Q****uery Granularity*) e P (*Hashed* ***P****artitions*).

É relevante referir que a escolha das granularidades a utilizar no *segment granularity*, segundo a documentação do Druid, é uma função entre o volume de dados e o intervalo de tempo pelo qual os dados se distribuem. Por exemplo, uma fonte de dados com *timestamps* espalhados ao longo do ano obterá um melhor desempenho particionado por dia, enquanto que uma fonte de dados com *timestamps*

espalhados ao longo do dia irá obter um melhor desempenho particionado por hora. Assim, selecionaram-se as granularidades dia, mês, trimestre e semestre, excluindo-se granularidades superiores (como, por exemplo, ano), visto que se iriam gerar grandes segmentos de dados, degradando o desempenho e exigindo uma grande quantidade de memória para os processar, uma vez que na ingestão de dados (feita através de um job MapReduce) é gerado um número de *reduces* igual ao número de *shards*/segmentos. Isto significa que se a fonte de dados possuir 7 anos e se definir o *segment granularity* como ano, serão gerados 7 segmentos e consecutivamente 7 *reduces* (Chambi et al., 2016; Druid, 2018m, 2018h; Yang et al., 2014). Mesmo assim, com as granularidades definidas, para algumas tabelas estavam a ser originados erros de memória. Para ultrapassar este problema, aumentou-se a memória máxima disponível para o MapReduce e o Yarn atribuírem a cada *container* para 12 GB. Contudo, para o SF de 100, por exemplo, o carregamento de uma tabela com o *segment granularity* definido ao semestre, e sem qualquer tipo de partições ou *query granularity* definidas, falhou devido a um erro de memória, tal como se demonstra na Figura 39.

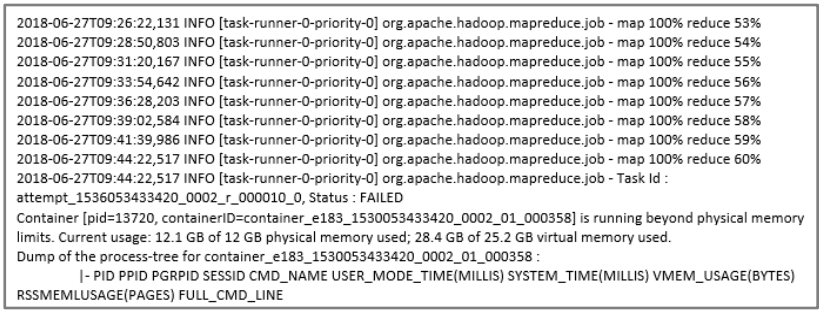


Figura 39 - Erro na Ingestão de Dados para o Druid.

No que à *query granularity* diz respeito, face às características dos dados e às queries do SSB, a granularidade mínima possível de ser utilizada é o dia, visto que o *timestamp* da fonte de dados do SSB tem informação diária (não tem informação da hora, por exemplo). Além disto, selecionou-se ainda a granularidade semana, por permitir uma comparação com alguma distância temporal. Por fim, como se observou que apenas a *query* 1.3 do SSB exigia um atributo com informação mensal, acrescentou-se também a *query granularity,* mês de forma a ter mais um cenário de comparação em que o efeito da *query granularity* pudesse ser mais evidente, sendo que, a *query* 1.3 nos cenários em que a *query granularity* estiver definida como mês, não irá retornar resultados. É importante referir que a *query granularity* não foi aplicada ao modelo com todos os atributos, visto que, este modelo possui registos únicos (por considerar atributos que são identificadores únicos, como por exemplo o "orderkey") e, como

tal, a *query granularity* não iria fazer qualquer agregação nos dados (o efeito seria o mesmo a não utilizar esta funcionalidade) (Druid, 2018h, 2018i).

Por fim, no que a partições diz respeito, a documentação do Druid refere que o mais recomendável é a definição de *hashed partitions* (baseadas no *hash* de todas as dimensões em cada linha), tal como já foi anteriormente referido, uma vez que melhora o desempenho na indexação dos dados e cria segmentos mais uniformes relativamente às partições do tipo *single-dimension*. Optou-se ainda por definir as *hashed partitions*, configurando a propriedade "numShards", a fim de ter um maior controlo sobre o número de partições geradas para cada intervalo de tempo e porque a documentação do Druid refere que desta forma a ingestão de dados será mais rápida (Druid, 2018b, 2018m).

O número de *hashed partitions* a testar será uma função entre o tamanho dos segmentos gerados, o número de linhas por segmento e ainda a evolução do desempenho entre os diferentes testes. Segundo a documentação do Druid, é recomendável que os segmentos tenham tamanhos uniformes compreendidos entre 300 MB e 700 MB e um número de linhas por segmento de cerca de 5.000.000. Sublinha-se que a documentação é contraditória, uma vez que Druid (2018m) refere o intervalo anteriormente mencionado, enquanto que Druid (2018b) aponta para um intervalo de 500 MB a 1 GB.

Através destes testes, será possível aferir qual a melhor estratégia a utilizar num BDW implementado no Druid. Para tal comparar-se-ão os resultados obtidos para os vários cenários de teste, a fim de se perceber a influência do *segment granularity*, do *query granularity* e das *hashed partitions* no desempenho do Druid e no espaço ocupado, assim como a evolução dos resultados ao aumento do volume de dados, uma vez que serão aplicados 3 SFs diferentes (30, 100, 300). Além disso, e como já foi mencionado, comparar-se-á o desempenho do Druid com o Hive e o Presto, recorrendo aos resultados obtidos por E. Costa (2017) e E. Costa et al. (2017) o que possibilitará perceber se o Druid tem capacidade para substituir estas tecnologias ou para ser utilizado como complemento às mesmas.

A Figura 40 sintetiza os cenários de teste que serão analisados neste capítulo. Salienta-se que além dos testes já abordados será ainda conduzido um teste para estudar o potencial da integração entre o Hive e o Druid e alguns testes em ambientes multiutilizador, que serão detalhados numa fase mais adiantada deste capítulo.

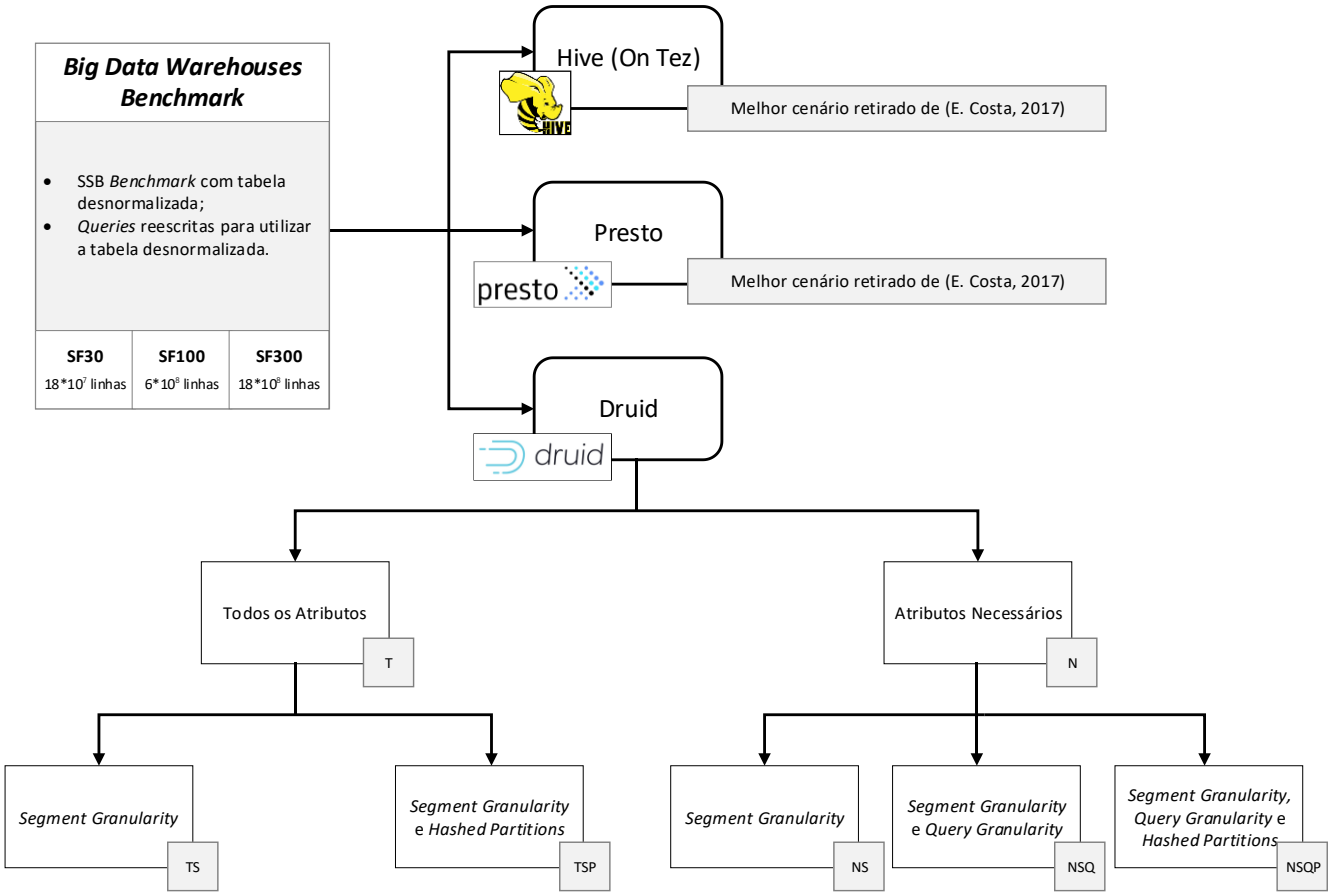


Figura 40 - Cenários de Teste ao Druid para Processamento Analítico de dados em BDW.

4.2.1 Preparação dos Cenários de Teste

Antes da apresentação dos resultados obtidos é importante detalhar a preparação dos cenários de teste, isto é, todas as tarefas necessárias para preparar e executar os vários testes.

Numa primeira fase é necessário gerar a fonte de dados do SSB com o SF pretendido e guardá-la no HDFS, sendo que, de seguida, é necessário criar no Druid as tabelas para os vários cenários. Estas tabelas são criadas através de uma tarefa para ingerir os dados em *batch* denominada por *Hadoop-ingestion task*, que executa no cluster Hadoop (Druid, 2018b). Na Figura 41, apresenta-se um exemplo de ingestão de dados no Druid. No exemplo ilustrado está a ser carregada uma tabela do SF 300, utilizando o modelo só com os atributos necessários para responder às queries do SSB, com a *segment granularity* definida como trimestre ("*quarter*"), a *query granularity* definida como dia ("*day*") e ainda um número de 15 *hashed partitions*. De referir que no "tuningConfig" foram ainda definidas as propriedades "maxRowsInMemory" e "numBackgroundPersistThreads" para valores apontados como recomendáveis

para tornar a ingestão de dados mais rápida (Druid, 2018b). As restantes tarefas de ingestão de dados podem ser consultadas em: https://github.com/jmcorreia/Druid_SSB_Benchmark.

```
{
  "type" : "index_hadoop",
  "spec" : {
    "ioConfig" : {
      "type" : "hadoop",
      "inputSpec" : {
        "type" : "static",
        "paths" : "/user/lid4/correia/ssb/ssb_analytical_objects.analytical_obj300/"
      }
    },
    "dataSchema" : {
      "dataSource" : "analytical_obj300_SQuarter_QDay_PHashed15",
      "granularitySpec" : {
        "type" : "uniform",
        "segmentGranularity" : "quarter",
        "queryGranularity" : "day",
        "intervals" : ["1991/1999"]
      },
      "parser" : {
        "type" : "hadoopyString",
        "parseSpec" : {
          "format" : "tsv",
          "columns" : ["c_custkey","c_name","c_address","c_city","c_nation","c_region","c_phone","c_mktsegment","s_suppkey","s_name","s_address","s_city",
                       "s_nation","s_region","s_phone","p_partkey","p_name","p_mfgr","p_category","p_brand1","p_color","p_type","p_size","p_container",
                       "od_datekey","od_datestandard","od_date","od_weeknuminyear","od_monthnuminyear","od_year","od_daynuminmonth","od_yearmonthnum",
                       "od_yearmonth","cd_datekey","cd_datestandard","cd_date","cd_weeknuminyear","cd_monthnuminyear","cd_year","cd_daynuminmonth",
                       "cd_yearmonthnum","cd_yearmonth","orderkey","linenumber","orderpriority","shippriority","quantity","extendedprice","ordertotalprice",
                       "discount","revenue","supplycost","tax","shipmode"],
          "delimiter":"|",
          "dimensionsSpec" : {
            "dimensions" : ["c_city", "c_nation", "c_region", "s_city", "s_nation", "s_region", "p_mfgr", "p_category", "p_brand1",
                            "od_weeknuminyear", "od_year", "od_yearmonthnum", "od_yearmonth"]
          },
          "timestampSpec" : {
            "format" : "yyyy-MM-dd",
            "column" : "od_datestandard"
          }
        }
      },
      "metricsSpec" : [
        {
          "type" : "doubleSum",
          "name" : "quantity",
          "fieldName" : "quantity"
        },
        {
          "type" : "doubleSum",
          "name" : "extendedprice",
          "fieldName" : "extendedprice"
        },
        {
          "type" : "doubleSum",
          "name" : "discount",
          "fieldName" : "discount"
        },
        {
          "type" : "doubleSum",
          "name" : "revenue",
          "fieldName" : "revenue"
        },
        {
          "type" : "doubleSum",
          "name" : "supplycost",
          "fieldName" : "supplycost"
        }
      ]
    },
    "tuningConfig" : {
      "type" : "hadoop",
      "maxRowsInMemory" : "200000",
      "numBackgroundPersistThreads":"1",
      "partitionsSpec": {
        "type": "hashed",
        "numShards": 15
      },
      "jobProperties" : {
      }
    }
  }
}
```

Figura 41 - Exemplo de Ingestão de Dados no Druid.

A Figura 42 apresenta um exemplo da submissão da tarefa de ingestão apresentada anteriormente, através do método POST suportado pelo protocolo HTTP (Druid, 2018p).

```
[admin@node5 correia]$ curl -X 'POST' -H 'Content-Type:application/json' -d @gerarSSB
/necessary_attributes/analytical_obj300_SQuarter_QDay_PHashed15_druid.json localhost:
8090/druid/indexer/v1/task
{"task":"index_hadoop_analytical_obj300_SQuarter_QDay_PHashed15_2018-07-06T10:01:31.1
50Z"}[admin@node5 correia]$
```

Figura 42 - Submeter Tarefa de Ingestão de Dados no Druid.

Submetida a tarefa de ingestão, o seu estado pode ser acompanhado recorrendo à "Overlord Console" do Druid, evidenciada na Figura 43.

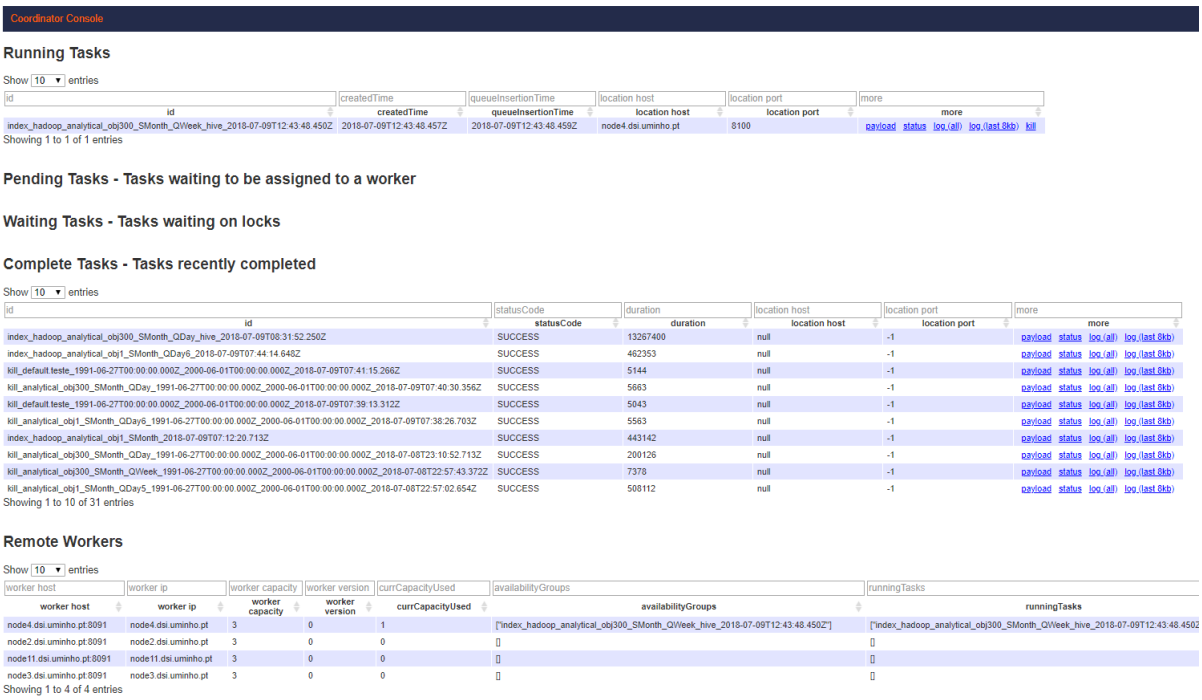


Figura 43 - Druid Overlord Console.

Se a tabela for carregada com sucesso, pode ser consultada na "Coordinator Console" do Druid, apresentada na Figura 44, na qual é possível verificar várias informações, como por exemplo o número de segmentos de dados e *shards* originados, o tamanho de cada um deles, a granularidade, os atributos que constituem cada um dos segmentos, entre outras ações. De referir que as tabelas são criadas com um fator de replicação 2 (configuração por defeito) e será analisado o espaço que ocupam no HDFS (*deep storage* utilizado pelo Druid) e nos *historical nodes* do Druid. Os valores apresentados não incluirão a replicação, sendo que, como está a ser usado um fator de replicação 2, o valor total será o dobro daquele que é apresentado como ocupado nos *historical nodes* do Druid.

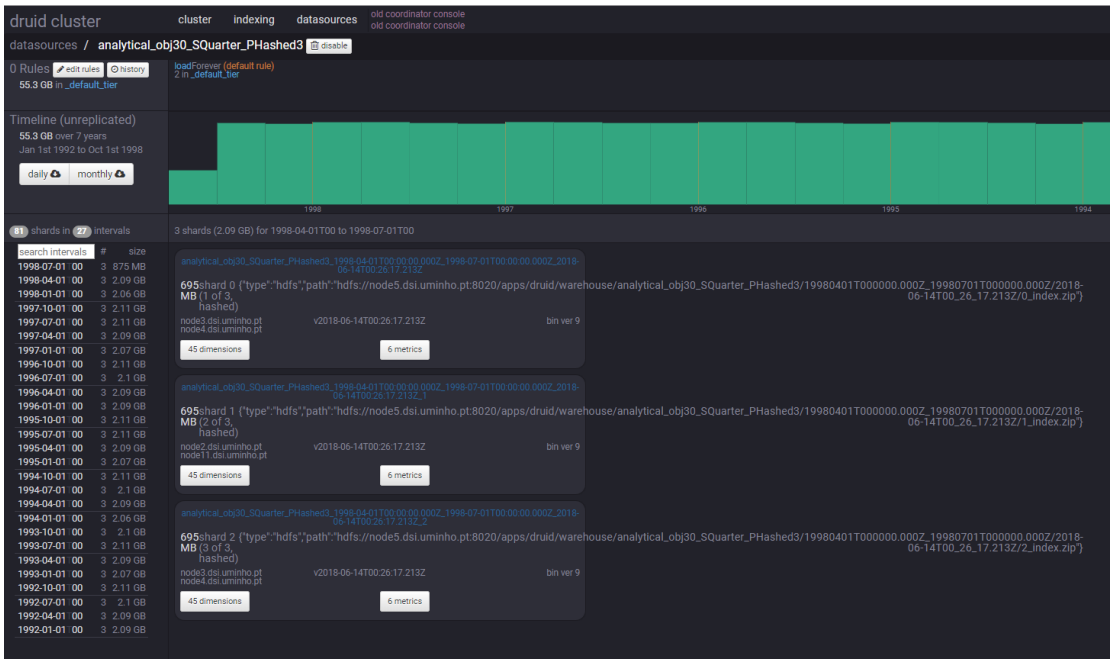


Figura 44 - Druid Coordinator Console.

Depois de carregadas as tabelas para o Druid, é necessário definir de que forma é que se irão executar as várias *queries* do SSB. Neste sentido, dado que estas *queries* estão implementadas em SQL e tendo em conta que esta é uma linguagem *standard* optou-se por utilizar a funcionalidade de SQL providenciada pelo Druid, ao invés da sua linguagem nativa baseada em JSON. Para utilizar esta funcionalidade foi necessário definir a propriedade "druid.sql.enable = true", nas configurações do Druid *Broker*. É relevante mencionar que a documentação do Druid refere que o SQL é uma funcionalidade ainda experimental, contudo, todas as queries do SSB foram suportadas por esta funcionalidade. Por fim, sublinha-se que, segundo a documentação, para além de uma pequena sobrecarga na tradução do SQL para a linguagem nativa do Druid, não existe qualquer outra perda de desempenho a considerar (Druid, 2018n).

Com o objetivo de se obterem resultados mais rigorosos foram criadas scripts responsáveis por executar 4 vezes cada uma das 13 queries do SSB, abordagem já seguida no trabalho de E. Costa (2017), E. Costa et al. (2017). Estas *scripts* terão ainda variações para executar em ambiente multiutilizador, com ou sem o mecanismo de *cache* ativo e efetuando as *queries* diretamente ao Druid ou através do Hive. Todas estas *scripts* estão disponíveis em https://github.com/jmcorreia/Druid_SSB_Benchmark. Por fim, é importante deixar a nota que durante a análise dos resultados obtidos não será utilizada apenas a média dos quatro resultados de cada *query*, mas também o primeiro resultado. Dessa forma será possível perceber melhor o efeito da *cache* do Druid, uma vez que no primeiro resultado a *cache* não tem ainda impacto, porque a *query* não foi

executada anteriormente. Após a execução das primeiras *queries* é possível começar a perceber as vantagens da *cache* no processamento dos dados.

4.3 Resultados Obtidos

Finalizada a análise do conjunto de dados e apresentados os vários cenários de teste, nesta secção serão apresentados e analisados os resultados obtidos que constituirão uma base para tecer algumas recomendações sobre a implementação de um BDW no Druid e qual o desempenho desta tecnologia face às alternativas.

Antes da apresentação dos resultados obtidos, considera-se relevante esclarecer a nomenclatura que irá ser seguida para o nome das tabelas guardadas no Druid e ainda os símbolos utilizados nas tabelas informativas. Assim, o nome das tabelas do Druid segue a lógica S{*Segment Granularity*}_Q{*Query Granularity*}_PHashed{Número de Partições}. Seguindo esta notação, uma das tabelas nomeia-se, por exemplo, "SMonth_QDay_PHashed2". Nos cenários em que a tabela não possui *hashed partitions* ou *query granularity*, estas são excluídas do nome, como se verifica nos casos "SMonth", "SMonth_PHashed2" ou "SMonth_QDay".

Quanto aos símbolos utilizados para representar as características das tabelas carregadas no Druid, esclarece-se de seguida o seu significado:


Espaço ocupado no HDFS (utilizado como *deep strorage* do Druid).


Espaço médio ocupado por cada *shard* ou segmento.

Espaço ocupado nos *historical nodes* do Druid.

Número de atributos do modelo utilizado.


Número de segmentos.


Número médio de linhas por *shard* ou segmento.


Número de *shards* gerados.


Desempenho.

Destaca-se, ainda, que será recorrente circundar a verde os melhores desempenhos obtidos e será também frequente apresentar não só a média de resultados obtidos, mas também o resultado obtido na *run* 1 das *queries*, de forma a perceber-se melhor o efeito da *cache* do Druid (uma vez que na primeira execução da *query*, os resultados necessários ainda não estarão em *cache).*

4.3.1 Cenário T – Modelação com Todos os Atributos

Este cenário, tal como anteriormente mencionado, engloba todos os cenários de teste que têm por base um modelo desnormalizado com todos os atributos do modelo original do SSB. Desta forma, serão aqui apresentados os tempos de processamento e várias informações sobre cada uma das tabelas utilizadas, para possibilitar a análise do impacto da definição da *Segment Granularity* e *Hashed Partitions* no Druid.

É importante destacar que, dado o elevado tempo de ingestão das tabelas do SF 300, optou-se por uma estratégia para minimizar a criação de muitas tabelas para o SF em questão. O tempo de ingestão de cada tabela (com o modelo com todos os atributos), do SF 30 para o SF 100, evoluiu de uma média de cerca de 1 hora e 19 minutos para cerca de 4 horas e 14 minutos, pelo que ingerir as várias tabelas para o SF 300 demoraria um tempo que podia prejudicar a evolução do restante trabalho. Desta forma, considerou-se suficientemente representativo realizar todos os cenários para o SF 30 e SF 100 e, baseado nos resultados obtidos, escolher os cenários considerados relevantes para explorar com SF 300.

Cenário TS - Segment Granularity

A Tabela 13 apresenta as características das várias tabelas utilizadas neste cenário. Pela análise desta tabela, verifica-se que a *segment granularity* influencia o espaço de armazenamento das tabelas, sendo que, quanto menos detalhada for a granularidade definida menor o espaço ocupado. Para o SF 30, por exemplo, a tabela com granularidade dia (tabela "SDay") ocupa 29 GB no HDFS e 69 GB nos *historical nodes* do Druid, enquanto que para guardar os mesmos dados numa tabela com granularidade *Semester* o espaço ocupado reduz para 20 GB no HDFS e 36 GB no Druid. Além disso, por comparação do espaço ocupado pelas tabelas no SF 30 e SF 100, verifica-se que existe um aumento quase na mesma proporção, isto é, 100/30 é igual a cerca de 3.33 e, para o caso da tabela "SDay" o espaço ocupado evoluiu numa proporção de 3.41 no HDFS e de 3.30 no Druid.

Tabela 13 - Informação Geral sobre as Tabelas do Cenário TS.

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
30	SDay	29	68	2406	2406	29	53	0.07
	SMonth	23	45	80	80	564		2.25
	SQuarter	21	39	27	27	1452		6.67
	SSemester	20	36	14	14	2542		12.86
100	SDay	99	227	2406	2406	94		0.25
	SMonth	76	149	80	80	1860		7.50

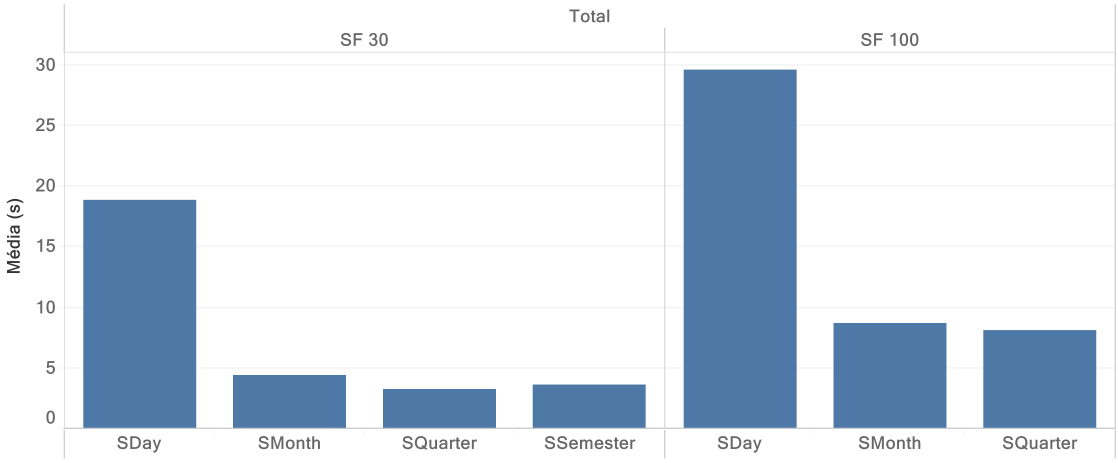
SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
	SQuarter	71	130	27	27	4810		22.22
	SSemester	Erro de memória.						

Analisadas as características das tabelas utilizadas neste cenário, importa agora observar os resultados obtidos por cada uma delas. Assim, na Figura 45, apresentam-se os tempos médios de processamento obtidos pelas várias tabelas, em cada uma das *queries*, sendo que os melhores resultados (menores tempos de processamento) por *query* estão circundados a verde. Através da exploração da figura, conclui-se que a tabela com granularidade trimestre ("SQuarter") obtém os melhores resultados em 8 *queries* para o SF 30 e em 9 *queries* para o SF 100. Contudo, neste SF e nas *queries* Q1.2 e Q1.3, os tempos obtidos são iguais aos obtidos pela "SMonth". Além disto, verifica-se ainda que a tabela "SMonth" apresentou um melhor desempenho no SF 100 face ao apresentado para SF 30, uma vez que obteve um menor tempo de processamento em 6 *queries*, enquanto que para o SF 30 tinha obtido o menor tempo de processamento em apenas 3 *queries*.

		Query												
SF	Tabela	Q 1.1	Q 1.2	Q 1.3	Q 2.1	Q 2.2	Q 2.3	Q 3.1	Q 3.2	Q 3.3	Q 3.4	Q 4.1	Q 4.2	Q 4.3
SF 30	SDay	2.90	0.85	1.76	7.99	0.85	0.41	1.74	0.39	0.35	0.34	0.61	0.36	0.28
	SMonth	0.51	0.09	0.06	1.26	0.28	0.15	0.55	0.47	0.17	0.06	0.50	0.15	0.11
	SQuarter	0.17	0.06	0.03	0.85	0.25	0.13	0.40	0.29	0.15	0.06	0.51	0.21	0.11
	SSemester	0.23	0.07	0.03	0.82	0.31	0.15	0.39	0.33	0.17	0.05	0.56	0.28	0.18
SF 100	SDay	2.10	0.93	0.53	12.03	1.41	0.76	3.38	1.65	0.61	1.42	3.75	0.62	0.40
	SMonth	0.47	0.15	0.08	2.21	0.78	0.41	0.99	0.90	0.43	0.14	1.46	0.43	0.27
	SQuarter	0.53	0.15	0.08	1.76	0.71	0.37	0.94	0.62	0.38	0.12	1.58	0.52	0.32

Figura 45 - Tempos Médios de Processamento de cada Query (em segundos) - Cenário TS.

Considerando agora a Figura 46, é possível verificar os tempos totais de processamento por tabela (média dos tempos totais das quatros execuções das *queries*), tendo em contra ambos os SF. Assim, confirma-se que, de facto, a tabela "SDay" obteve um desempenho bastante inferior às restantes e que a tabela "SQuarter" obteve o melhor desempenho em ambos os SFs. Mais se observa que, tal como referido anteriormente, a tabela "SMonth" aproximou-se do desempenho da "SQuarter", quando comparada a evolução de ambas do SF 30 para o SF 100. Enquanto que no SF 30 a tabela "SQuarter" obteve um tempo médio de processamento das *queries* 26.42% inferior à tabela "SMonth", para o SF 100 a diferença reduziu para 7.21%.



	SF 30				SF 100		
	SDay	SMonth	SQuarter	SSemester	SDay	SMonth	SQuarter
Run 1 (s)	42.06	8.96	5.43	5.66	63.05	14.70	12.71
Média (s)	18.85	4.37	3.21	3.58	29.58	8.71	8.08
Dif. Média (%)		-76.84%	-26.42%	11.32%		-70.56%	-7.21%

Figura 46 - Tempos Totais de Processamento das Queries - Cenário TS.

Confrontando os resultados obtidos por cada tabela, com as suas características anteriormente apresentadas, verifica-se que as tabelas que obtiveram melhores desempenhos, no SF 30, "SMonth" e "SQuarter", possuem segmentos de dados que ocupam, em média, 564 MB e 1452 MB, respetivamente e, um número médio de linhas por segmento de $2.25*10^6$, no caso da "SMonth" e de $6.67*10^6$, no caso da "SQuarter". Estes valores encontram-se mais próximos dos valores recomendados pela documentação do Druid (cerca de $5*10^6$ linhas por segmento/*shard* e um espaço médio entre 500 MB - 1 GB), do que as restantes tabelas, o que pode explicar o desempenho alcançado (Druid, 2018m, 2018b).

Para o SF 100, mais uma vez, a tabela "SDay" é caracterizada por muitos *shards* pequenos (2406 *shards*), uma vez que possuem um tamanho médio de 94 MB e um número médio de linhas de $0.25*10^6$, distante dos valores recomendados. Quanto às tabelas "SMonth" e "SQuarter", embora tenham obtido melhor desempenho, as suas características afastaram-se das recomendadas dado o aumento do volume de dados. Os segmentos de dados da tabela "SQuarter" possuem agora um espaço médico ocupado de 4810 MB e um número médio de linhas de $22.22*10^6$. Isto explica que a tabela "SMonth" tenha evoluído o seu desempenho (aproximando-se do desempenho da "SQuarter"), uma vez que se encontra mais próxima dos valores recomendados. Ainda que os valores da tabela "SQuarter" sejam superiores ao recomendado, os resultados indicam, para já, que é preferível ter um menor número

de ficheiros de maior dimensão, por oposição a muitos ficheiros de menor dimensão, que é o caso da tabela "SDay".

Cenário TSP - Segment Granularity e Hashed Partitions

Dado que tanto para o SF 30, como para o SF 100, se verificou que existia ainda alguma diferença de valores face ao recomendado, quer no espaço médio ocupado, quer no número médio de linhas por *shard*, neste cenário irá explorar-se a definição de *hashed partitions* e o seu impacto no desempenho e nas características das tabelas geradas.

É importante voltar a referir que a definição do número de *hashed partitions* irá ter em conta se as características das tabelas estão próximas do recomendado e se o desempenho obtido está a melhorar ou piorar. Por exemplo, para uma tabela que ainda não está no intervalo de valores aconselhado (espaço e número de linhas por *shard*), vai-se aumentando o número de partições até que as características fiquem mais próximas desses valores. Depois, observa-se a evolução do desempenho e quando a tabela já estiver com características semelhantes às recomendadas e o seu desempenho tiver piorado, considera-se irrelevante aumentar o número de partições.

A Tabela 14 apresenta as características das várias tabelas utilizadas neste cenário. Examinando esta tabela é possível verificar que a definição de *hashed partitions* implica o consumo de mais espaço de armazenamento. Por exemplo, para o SF 30, se se comparar as tabelas "SSemester_PHashed5" e "SSemester" (Tabela 13), verifica-se que o espaço ocupado pela primeira é 43.5% superior no HDFS (evoluiu de 20 GB para 29 GB) e 60.5% superior no Druid (evoluiu de 36 GB para 58 GB). Quando se analisam as tabelas do SF 100, como por exemplo a "SQuarter_PHashed20" e a "SQuarter" (Tabela 13) o impacto é ainda maior, visto que a primeira ocupa 77% mais espaço no HDFS (evoluiu de 71 GB para 126 GB) e requer um armazenamento 115% maior no Druid (evoluiu de 130 GB para 279 GB).

Tabela 14 - Informação Geral sobre as Tabelas do Cenário TSP.

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
30	SMonth_PHashed2	27	57	80	160	355	53	1.13
	SQuarter_PHashed2	25	49	27	54	904		3.33
	SQuarter_PHashed3	27	55		81	683		2.22
	SQuarter_PHashed4	29	60		108	555		1.67
	SQuarter_PHashed5	30	64		135	471		1.33
	SSemester_PHashed2	24	44	14	28	1581		6.43
	SSemester_PHashed3	26	50		42	1192		4.29
	SSemester_PHashed4	28	54		56	969		3.21
	SSemester_PHashed5	29	58		70	822		2.57

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
100	SMonth_PHashed2	91	187	80	160	1169		3.75
	SMonth_PHashed3	101	213		240	888		2.5
	SMonth_PHashed4	107	232		320	725		1.88
	SQuarter_PHashed3	92	183	27	81	2256		7.41
	SQuarter_PHashed4	98	198		108	1830		5.56
	SQuarter_PHashed5	102	210		135	1680		4.45
	SQuarter_PHashed6	106	220		162	1360		3.70
	SQuarter_PHashed10	115	247		270	915		2.22
	SQuarter_PHashed15	122	267		405	660		1.48
	SQuarter_PHashed20	126	279		540	517		1.11
	SSemester_PHashed3	87	166	14	42	3950		14.29
	SSemester_PHashed4	92	180		56	3210		10.72
	SSemester_PHashed5	96	190		70	2710		8.57
	SSemester_PHashed10	108	224		140	1600		4.29
	SSemester_PHashed15	115	243		210	1157		2.86
	SSemester_PHashed20	119	256		280	914		2.14
	SSemester_PHashed25	122	266		350	760		1.71

Importa agora perceber se o aumento no espaço de armazenamento, ditado pela definição de *hashed partitions*, produz uma melhoria no desempenho do processamento de queries. Desta forma, na Figura 47 são apresentados os resultados obtidos por cada uma das tabelas, deste cenário e do cenário TS (para ser mais fácil a comparação), na execução das várias *queries*.

Por análise da figura supracitada é possível verificar que, para o SF 30, a tabela "SQuarter_PHashed3" obteve os menores tempos médios de processamento em 6 das 13 *queries*. Além disso, é visível que a tabela em questão registou melhor desempenho que a "SQuarter" em 11 das 13 *queries*, revelando-se aqui um impacto positivo da definição de *hashed partitions*. Esta tendência também se verifica para o SF 100, visto que a tabela "SMonth_PHashed2", por exemplo, alcançou tempos médios de processamento menores que a tabela "SMonth" em 11 *queries*. Continuando a analisar o SF 100, a tabela "SSemester_PHashed20" foi a que obteve melhores resultados médios num maior número de queries (4 de 13). No entanto, através de uma análise mais pormenorizada, conclui-se que esta tabela não terá sido aquela que exigiu menor tempo para processar todas as *queries*, tendo em conta os tempos mais elevados nas *queries* Q2.1, Q3.1, Q3.2 e Q4.1. Refira-se, ainda, que tempos mais elevados na resposta a estas *queries* e à Q1.1 é uma tendência transversal a todas as tabelas.

		Query												
SF	Tabela	Q 1.1	Q 1.2	Q 1.3	Q 2.1	Q 2.2	Q 2.3	Q 3.1	Q 3.2	Q 3.3	Q 3.4	Q 4.1	Q 4.2	Q 4.3
SF 30	SMonth	0.51	0.09	0.06	1.26	0.28	0.15	0.55	0.47	0.17	0.06	0.50	0.15	0.11
	SMonth_PHashed2	0.57	0.11	0.06	1.54	0.28	0.14	0.67	0.55	0.18	0.09	0.48	0.14	0.11
	SQuarter	0.17	0.06	0.03	0.85	0.25	0.13	0.40	0.29	0.15	0.06	0.51	0.21	0.11
	SQuarter_PHashed2	0.13	0.05	0.03	0.93	0.19	0.14	0.50	0.48	0.13	0.06	0.50	0.15	0.11
	SQuarter_PHashed3	0.10	0.03	0.04	0.38	0.18	0.11	0.26	0.16	0.13	0.06	0.40	0.14	0.09
	SQuarter_PHashed4	0.17	0.09	0.03	0.86	0.16	0.11	0.43	0.51	0.12	0.05	0.41	0.12	0.10
	SQuarter_PHashed5	0.09	0.05	0.03	1.29	0.20	0.11	0.56	0.50	0.15	0.07	0.50	0.13	0.10
	SSemester	0.23	0.07	0.03	0.82	0.31	0.15	0.39	0.33	0.17	0.05	0.56	0.28	0.18
	SSemester_PHashed2	0.15	0.05	0.02	0.77	0.23	0.12	0.41	0.37	0.16	0.06	0.47	0.18	0.11
	SSemester_PHashed3	0.11	0.03	0.03	0.69	0.18	0.12	0.46	0.32	0.19	0.06	0.51	0.16	0.11
	SSemester_PHashed4	0.15	0.07	0.03	0.88	0.18	0.12	0.52	0.37	0.14	0.05	0.46	0.14	0.09
	SSemester_PHashed5	0.21	0.07	0.04	0.95	0.18	0.10	0.55	0.47	0.20	0.05	0.44	0.13	0.09
SF 100	SMonth	0.47	0.15	0.08	2.21	0.78	0.41	0.99	0.90	0.43	0.14	1.46	0.43	0.27
	SMonth_PHashed2	0.21	0.09	0.04	1.88	0.58	0.31	1.16	1.03	0.36	0.11	1.34	0.37	0.23
	SMonth_PHashed3	0.32	0.08	0.06	2.09	0.58	0.31	1.26	1.09	0.40	0.13	1.30	0.35	0.24
	SMonth_PHashed4	0.48	0.08	0.06	2.50	0.66	0.33	1.29	1.07	0.46	0.15	1.61	0.38	0.33
	SQuarter	0.53	0.15	0.08	1.76	0.71	0.37	0.94	0.62	0.38	0.12	1.58	0.52	0.32
	SQuarter_PHashed3	0.33	0.08	0.04	1.87	0.55	0.31	1.05	0.86	0.42	0.11	1.32	0.36	0.26
	SQuarter_PHashed4	0.42	0.07	0.05	1.80	0.62	0.34	0.99	0.78	0.39	0.14	1.33	0.42	0.25
	SQuarter_PHashed5	0.27	0.06	0.04	2.20	0.57	0.31	1.18	1.18	0.32	0.13	1.29	0.34	0.24
	SQuarter_PHashed6	0.32	0.07	0.05	1.20	0.55	0.31	0.81	0.44	0.33	0.13	1.21	0.42	0.28
	SQuarter_PHashed10	0.50	0.12	0.10	3.51	0.60	0.33	1.69	1.67	0.44	0.15	1.39	0.35	0.25
	SQuarter_PHashed15	0.32	0.11	0.09	3.09	0.58	0.31	1.49	1.43	0.40	0.15	1.46	0.30	0.20
	SQuarter_PHashed20	0.47	0.13	0.08	3.33	0.64	0.37	1.56	1.41	0.47	0.19	1.52	0.34	0.24
	SSemester_PHashed3	0.42	0.10	0.06	1.83	0.67	0.35	0.98	0.62	0.43	0.17	1.34	0.49	0.28
	SSemester_PHashed4	0.46	0.07	0.05	1.58	0.66	0.35	0.90	0.52	0.37	0.13	1.30	0.43	0.30
	SSemester_PHashed5	0.48	0.08	0.07	2.25	0.63	0.32	1.07	0.80	0.42	0.13	1.48	0.37	0.31
	SSemester_PHashed10	0.34	0.07	0.05	1.20	0.57	0.31	0.83	0.45	0.33	0.14	1.28	0.35	0.25
	SSemester_PHashed15	0.36	0.08	0.05	2.36	0.60	0.32	1.48	1.39	0.41	0.17	1.33	0.36	0.25
	SSemester_PHashed20	0.25	0.06	0.06	2.75	0.48	0.29	1.39	1.55	0.40	0.16	1.37	0.30	0.21
	SSemester_PHashed25	0.33	0.07	0.06	2.68	0.60	0.31	1.35	1.27	0.42	0.18	1.37	0.35	0.23

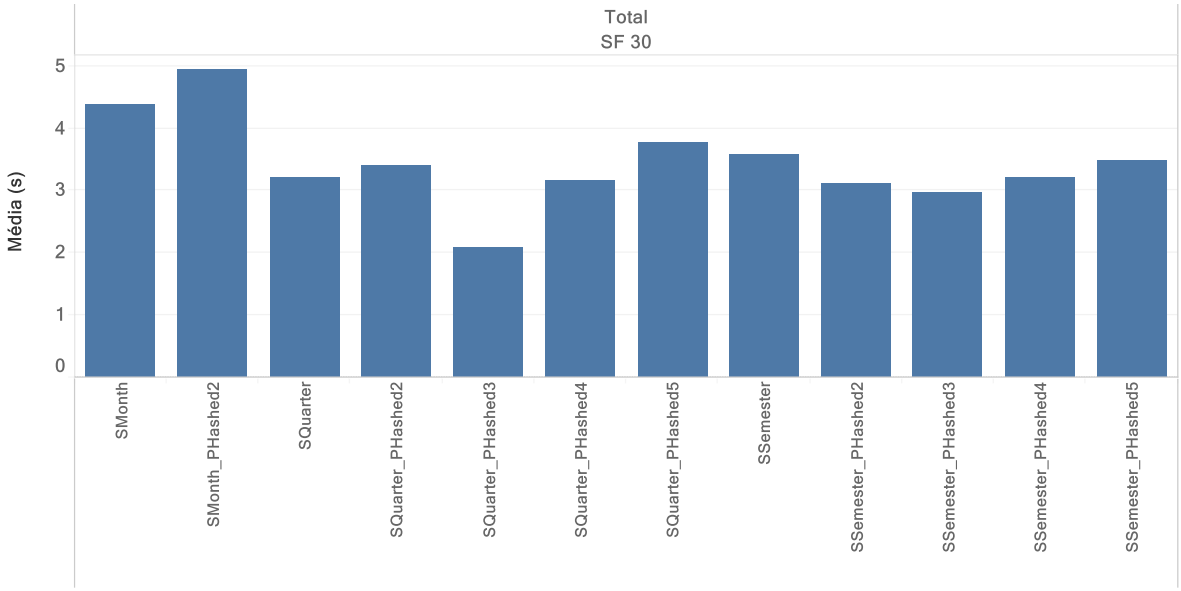
Figura 47 - Tempos Médios de Processamento de cada Query (em segundos) - Cenário TSP.

Analisando agora a Figura 48, na qual estão expressos os tempos de processamento agregados de todas as *queries* para o SF 30, é possível confirmar em que casos é que a definição de *hashed partitions* permitiu que fossem atingidos melhores desempenhos do que aqueles que tinham sido alcançados no cenário TS. Um destes casos é o da tabela "SQuarter_PHashed3" que obteve os melhores resultados para o SF em questão, 2.09 segundos de média das quatro execuções das *queries* (a primeira execução demorou 2.72 segundos), enquanto que a tabela "SQuarter" havia atingido o desempenho de 3.21 segundos (5.43 na primeira execução das *queries*). Ou seja, neste caso a definição de *hashed partitions* permitiu uma melhoria de 34.9% no tempo total médio de processamento de todas as *queries*.

Comparando a evolução do desempenho das tabelas de granularidade trimestre com as suas características, conclui-se que a tabela sem partições armazenava 27 *shards* que, em média, ocupavam 1452 MB e possuíam $6.67*10^6$ linhas (Tabela 13). Ou seja, não apresentava as características apontadas como recomendadas pelo Druid (já anteriormente mencionadas), pelo que, à priori, se esperasse que a definição de partições causasse uma melhoria no desempenho, pelo facto de dividir os dados num maior número de *shards* com características mais próximas das recomendadas. Contudo, os testes à tabela "SQuarter_PHashed2" revelaram um pior desempenho que a tabela não particionada, mesmo possuindo esta tabela características já enquadradas no recomendável (54 *shards* com 904 MB e $3.33*10^6$ linhas

em média), o que evidencia também alguma instabilidade nos resultados produzidos pela aplicação desta estratégia. A tabela "SQuarter_PHashed3", tal como já foi referido, alcançou os melhores resultados e é constituída por 81 *shards* com 683 MB e 2.22*10⁶ linhas em média. Continuando com a granularidade *quarter*, a definição de 4 ou mais *hashed partitions* começou a gerar tabelas com *shards* mais pequenos, o que piorou o desempenho (Druid, 2018b, 2018m).

Ainda na Figura 48, verifica-se que a definição de partições na tabela com granularidade *month* não se revelou vantajosa. A tabela "SMonth" já possuía 80 *shards* com 564 MB e 2.25 *10⁶ linhas em média e a definição de *hashed partitions* gerou um maior número de *shards* mais pequenos, degradando o desempenho.



	SF 30											
	SMonth	SMonth_PHashed2	SQuarter	SQuarter_PHashed2	SQuarter_PHashed3	SQuarter_PHashed4	SQuarter_PHashed5	SSemester	SSemester_PHashed2	SSemester_PHashed3	SSemester_PHashed4	SSemester_PHashed5
Run 1 (s)	8.96	9.99	5.43	5.73	2.72	5.95	6.61	5.66	5.16	4.57	5.66	6.31
Média (s)	4.37	4.93	3.21	3.40	2.09	3.15	3.77	3.58	3.10	2.96	3.22	3.48
Dif. Média (%)		12.89%		5.75%	-38.48%	50.53%	19.75%		-13.26%	-4.45%	8.58%	8.12%

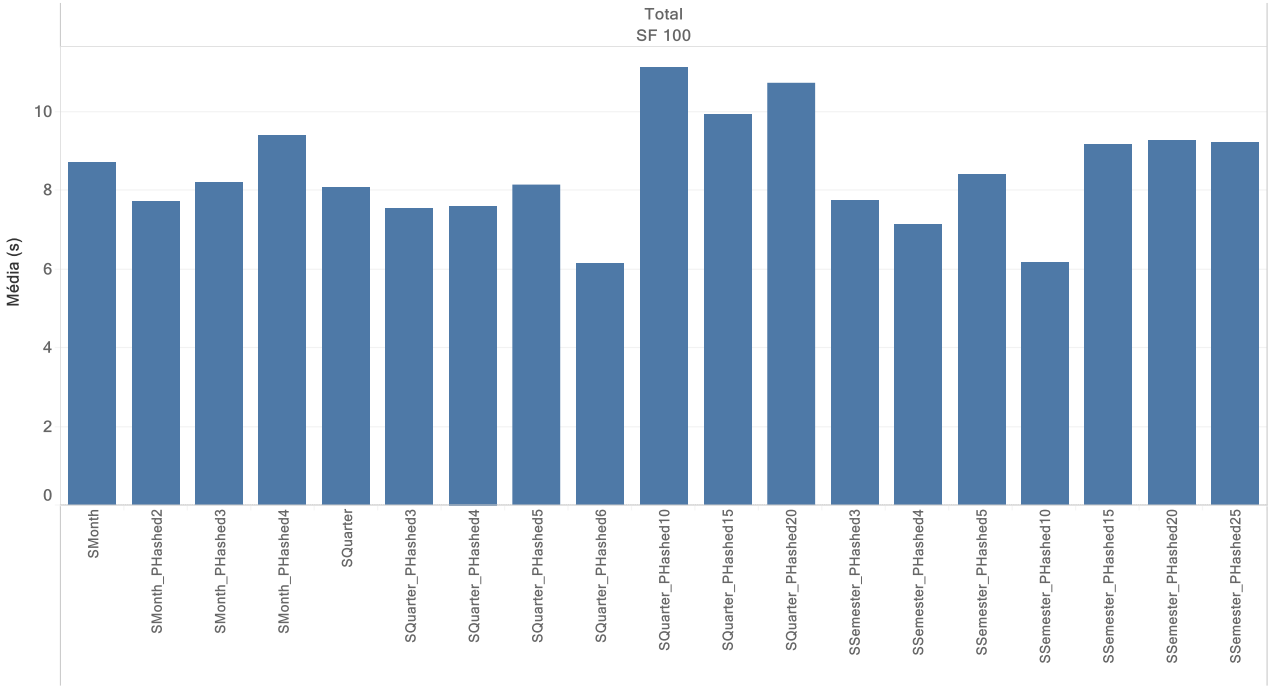
Figura 48 - Tempos Totais de Processamento das Queries, SF30 - Cenário TSP.

Recorrendo agora à Figura 49, é possível fazer uma análise idêntica à anterior, mas agora sobre os resultados obtidos para o SF 100. Importa relembrar que não são demonstrados resultados para a tabela "SSemester", no SF 100, visto que houve erro no seu carregamento. Desta vez, no que diz respeito à granularidade *month*, verifica-se uma melhoria de desempenho quando se definem *hashed partitions*.

Este comportamento explica-se pelo facto de a fonte de dados ser agora o SF 100, ou seja, possui um maior volume de dados, originando segmentos de dados maiores para a granularidade *month* e, passíveis de serem particionados, a fim de se alcançarem melhores desempenhos. A tabela não particionada "SMonth" possuía 80 *shards* com 1860 MB e $7.50*10^6$ linhas em média, enquanto que a "SMonth_PHashed2" já possuía 160 *shards* com 1169 MB e $3.75*10^6$ linhas em média.

Continuando o estudo dos resultados apresentados na Figura 49, comprovam-se as conclusões também retiradas na análise à Figura 48, isto é, também para os testes realizados a este SF se observam melhorias de desempenho resultantes da definição de *hashed partitions* e alguns comportamentos instáveis. Quanto aos comportamentos instáveis, estes verificam-se tanto na granularidade *quarter*, como na granularidade *semester*, visto que, na primeira delas, houve uma melhoria ligeira de desempenho até à definição de 4 *hashed partitions,* seguida de uma degradação no desempenho da tabela "SQuarter_PHashed5" e acompanhada de uma melhoria na tabela "SQuarter_PHashed6" que foi aquela que obteve os tempos médios de processamentos mais baixos para o SF 100. De referir que na granularidade *semester* se verifica uma tendência semelhante. No que diz respeito às melhorias de desempenho, a figura em análise demonstra que o particionamento das tabelas produziu melhores resultados em todas as granularidades, dependendo do número de partições adotado e das características que a adoção desse número de partições produziu nas tabelas em questão.

Para finalizar, tal como já foi referido, a tabela "SQuarter_PHashed6" obteve os melhores tempos totais médios de processamento (e também se se tiver apenas em consideração a *run* 1 das *queries*), sendo que atingiu um tempo cerca de 24.26% inferior ao da tabela "SQuarter" (24.74% melhor desempenho que o da tabela "SQuarter_PHashed5").

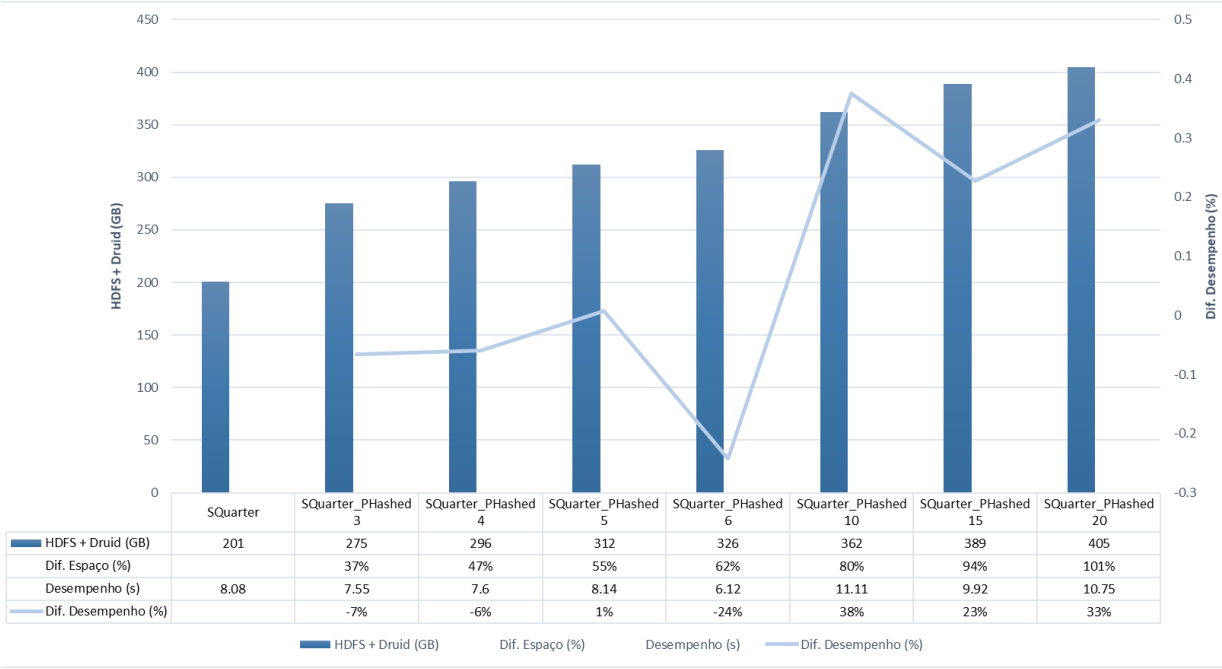


SF 100

	SMonth	SMonth_PHashed2	SMonth_PHashed3	SMonth_PHashed4	SQuarter	SQuarter_PHashed3	SQuarter_PHashed4	SQuarter_PHashed5	SQuarter_PHashed6	SQuarter_PHashed10	SQuarter_PHashed15	SQuarter_PHashed20	SSemester_PHashed3	SSemester_PHashed4	SSemester_PHashed5	SSemester_PHashed10	SSemester_PHashed15	SSemester_PHashed20	SSemester_PHashed25
Run 1 (s)	14.70	12.86	13.81	16.44	12.71	11.50	12.51	13.76	9.02	20.98	17.66	19.65	11.31	11.15	15.00	9.22	16.31	16.35	16.27
Média (s)	8.71	7.71	8.21	9.39	8.08	7.55	7.60	8.14	6.12	11.11	9.92	10.75	7.73	7.14	8.41	6.17	9.15	9.27	9.21
Dif. Média (%)		-11.50%	6.56%	14.37%		-6.54%	0.70%	6.98%	-24.74%	81.42%	-10.72%	8.38%		-7.67%	17.76%	-26.63%	48.37%	1.26%	-0.57%

Figura 49 - Tempos Totais de Processamento das Queries, SF 100 - Cenário TSP.

A Figura 50 providencia uma análise comparativa do aumento no espaço ocupado e das diferenças no tempo médio de processamento das *queries*, causadas pela definição de *hashed partitions*. Para a elaboração desta figura apenas foi tida em consideração a tabela "SQuarter" e as tabelas particionadas que advêm dela, no SF 100. Assim, verifica-se que a tabela "SQuarter_PHashed6" que atingiu o melhor desempenho, acarretou um espaço de armazenamento 62% superior à tabela não particionada ("SQuarter"), isto para atingir um tempo de processamento 24% inferior a essa tabela. Através da análise da mesma figura é ainda percetível, mais uma vez, que o desempenho obtido pela utilização desta funcionalidade pode ser um pouco instável, visto que, por vezes, as tabelas particionadas obtiveram pior desempenho (exemplo: "SQuarter_PHashed5").



	SQuarter	SQuarter_PHashed 3	SQuarter_PHashed 4	SQuarter_PHashed 5	SQuarter_PHashed 6	SQuarter_PHashed 10	SQuarter_PHashed 15	SQuarter_PHashed 20
HDFS + Druid (GB)	201	275	296	312	326	362	389	405
Dif. Espaço (%)		37%	47%	55%	62%	80%	94%	101%
Desempenho (s)	8.08	7.55	7.6	8.14	6.12	11.11	9.92	10.75
Dif. Desempenho (%)		-7%	-6%	1%	-24%	38%	23%	33%

Figura 50 - Comparação entre o Aumento no Espaço Ocupado e as Diferenças no Desempenho.

Finalmente, na Tabela 15, apresentam-se as tabelas que obtiveram os melhores resultados no cenário T, assim como as características de cada uma delas.

Tabela 15 - Melhores Desempenhos no Cenário T.

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(x*10⁶)	(s)
30	SQuarter_PHashed3	27	55	27	81	683	2.22	2.09
100	SQuarter_PHashed6	106	220	27	162	1360	3.70	6.12

4.3.2 Cenário N – Modelação com os Atributos Necessários

Ao longo deste cenário, tal como a designação faz antever, serão apresentados e discutidos os resultados obtidos nos vários testes que partilham em comum o facto de utilizarem como base um modelo desnormalizado do SSB orientado às *queries* pré-definidas, ou seja, que tem em conta apenas os atributos necessários para responder a todas as *queries* em avaliação.

A estruturação e discussão desta subsecção será idêntica àquela que foi utilizada para o cenário T. Como objetivos, para além daqueles que são comuns ao cenário T (analisar o impacto da definição da *Segment Granularity* e *Hashed Partitions* no Druid) existem agora os objetivos acrescidos de perceber o impacto da definição da *query granularity* e da redução de atributos do modelo de dados, no

desempenho e nas características das várias tabelas, nomeadamente no espaço necessário para armazenamento.

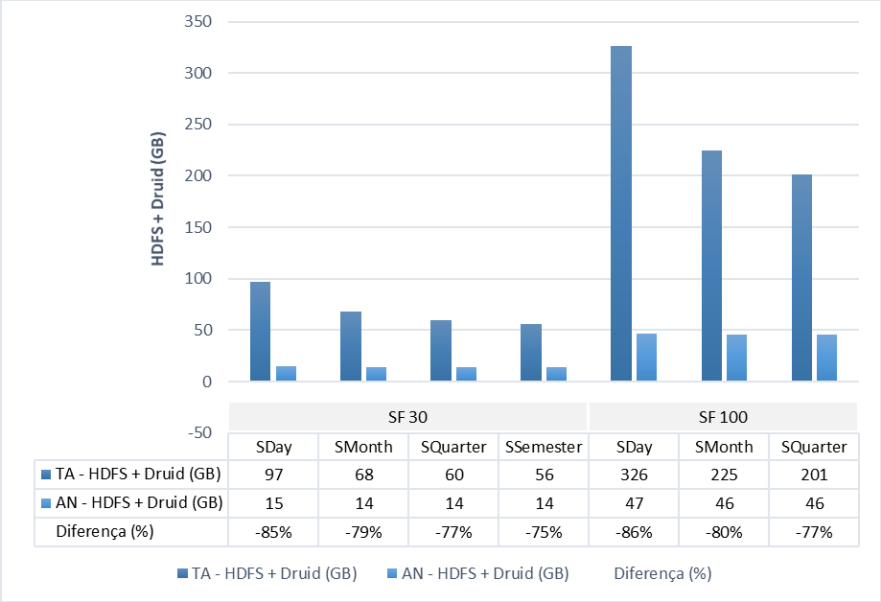
Cenário NS - Segment Granularity

A Tabela 16 apresenta então as características das tabelas utilizadas neste cenário, e que irão permitir perceber as alterações produzidas pela transformação no modelo de dados, que considera agora 18 atributos, ao invés dos 53 utilizados no cenário T.

Tabela 16 - Informação Geral sobre as Tabelas do Cenário NS.

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
30	SDay	6	9	2406	2406	4		0.07
	SMonth	5	9	80	80	106		2.25
	SQuarter	5	9	27	27	315		6.67
	SSemester	5	9	14	14	607		12.86
100	SDay	18	29	2406	2406	12	18	0.25
	SMonth	18	28	80	80	354		7.50
	SQuarter	18	28	27	27	1050		22.22
	SSemester	18	28	14	14	2020		42.86
300	SMonth	54	85	80	80	1060		22.50
	SQuarter	54	85	27	27	3150		66.67

Para efetuar então uma comparação do cenário NS e do cenário TS, no que diz respeito ao espaço ocupado, a Figura 51 será um importante auxílio a este estudo. Nesta figura, apresenta-se o espaço total ocupado (HDFS + Druid) pelo modelo com todos os atributos (representado pela sigla TA), pelo modelo com os atributos necessários (representado pela sigla AN) e a diferença percentual entre ambos, para o SF 30 e SF 100. Posto isto, por análise da figura em questão, percebe-se que a redução de 53 para 18 atributos considerados no modelo de dados, causou um decréscimo no espaço necessário ao armazenamento das tabelas, que se situa entre os 75% e os 86%.



	SF 30				SF 100		
	SDay	SMonth	SQuarter	SSemester	SDay	SMonth	SQuarter
TA - HDFS + Druid (GB)	97	68	60	56	326	225	201
AN - HDFS + Druid (GB)	15	14	14	14	47	46	46
Diferença (%)	-85%	-79%	-77%	-75%	-86%	-80%	-77%

Figura 51 - Diferença no Espaço Ocupado pelas Tabelas do Cenário TS e do Cenário NS.

Além disto, importa referir também as diferenças no tempo de ingestão das tabelas de ambos os cenários. Para o cenário TS, as tabelas do SF 30 obtiveram um tempo médio de ingestão de cerca de 1 hora e 19 minutos, enquanto que as tabelas do SF 100 necessitaram em média de cerca de 4 horas e 14 minutos. Já para o cenário NS, as tabelas do SF 30 demoraram em média 29 minutos (cerca de 63% menos tempo que no cenário TS), as do SF 100 alcançaram uma média de cerca de 1 hora e 22 minutos (aproximadamente 68% menos tempo que no cenário TS) e as tabelas do SF 300 prolongaram-se, em média, durante 3 horas e 43 minutos (cerca de 12% menos do que a média das tabelas do SF 100 do cenário TS).

Desta forma, verifica-se que o número de atributos considerados no modelo de dados tem consequências tanto no espaço de armazenamento como no tempo de ingestão, dado que quanto mais atributos forem considerados, maior o número de índices que terá de ser calculado durante a ingestão de dados, consumindo mais tempo e mais espaço de armazenamento.

Importa agora verificar se também existiu alguma melhoria no desempenho. Neste sentido, a Figura 52 apresenta os tempos médios de processamento de cada *query*, tanto para as tabelas baseadas no modelo com todos os atributos (sigla "TA" na figura), como para as baseadas no modelo com os atributos necessários (sigla "AN" na figura). Analisando a figura em questão, verifica-se que a tabela "SMonth" obteve melhor desempenho num maior número de *queries* em todos os SFs, ao contrário do que sucedera no cenário TS, no qual a tabela "SQuarter" tinha conseguido atingir os melhores resultados.

Além disto, verifica-se que, no geral, as tabelas que usam o modelo com os atributos necessários obtiveram melhores resultados face às tabelas que utilizam o modelo com todos os atributos.

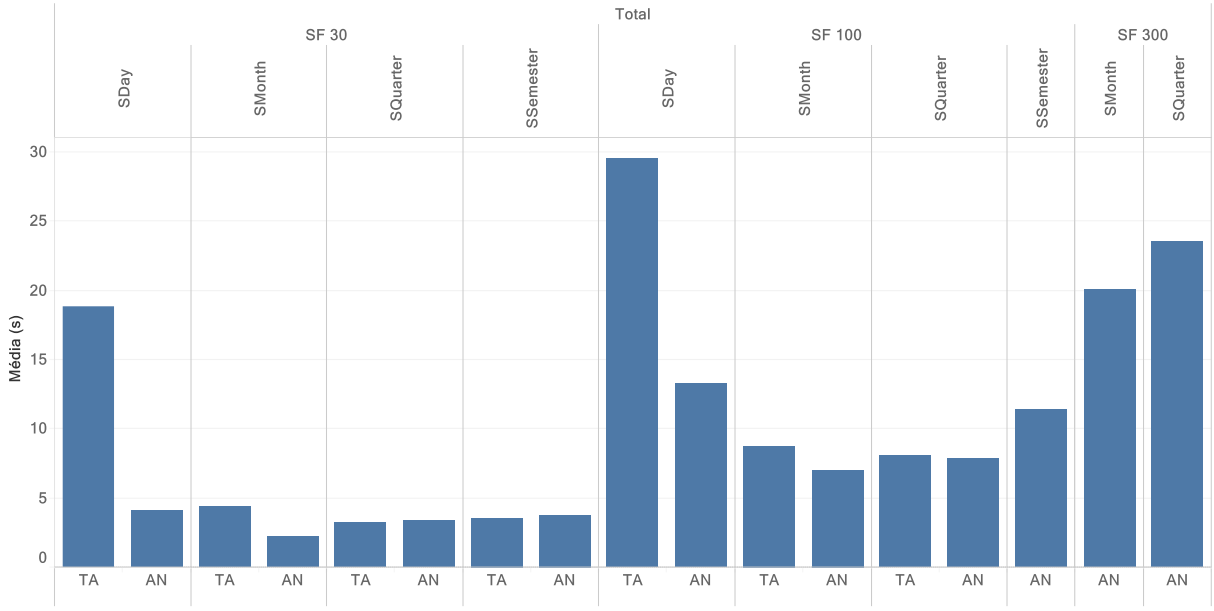
			Query												
SF	Tabela	Atributos	Q 1.1	Q 1.2	Q 1.3	Q 2.1	Q 2.2	Q 2.3	Q 3.1	Q 3.2	Q 3.3	Q 3.4	Q 4.1	Q 4.2	Q 4.3
SF 30	SDay	TA	2.90	0.85	1.76	7.99	0.85	0.41	1.74	0.39	0.35	0.34	0.61	0.36	0.28
		AN	0.17	0.09	0.08	0.96	0.38	0.27	0.39	0.29	0.28	0.20	0.49	0.24	0.23
	SMonth	TA	0.51	0.09	0.06	1.26	0.28	0.15	0.55	0.47	0.17	0.06	0.50	0.15	0.11
		AN	0.12	0.05	0.03	0.50	0.16	0.11	0.28	0.17	0.14	0.06	0.38	0.12	0.09
	SQuarter	TA	0.17	0.06	0.03	0.85	0.25	0.13	0.40	0.29	0.15	0.06	0.51	0.21	0.11
		AN	0.15	0.06	0.02	0.94	0.22	0.14	0.54	0.38	0.15	0.06	0.42	0.17	0.12
	SSemester	TA	0.23	0.07	0.03	0.82	0.31	0.15	0.39	0.33	0.17	0.05	0.56	0.28	0.18
		AN	0.29	0.06	0.03	0.80	0.27	0.15	0.44	0.41	0.18	0.07	0.59	0.30	0.18
SF 100	SDay	TA	2.10	0.93	0.53	12.03	1.41	0.76	3.38	1.65	0.61	1.42	3.75	0.62	0.40
		AN	2.12	1.08	0.57	4.11	0.71	0.46	0.89	0.56	0.43	0.28	1.25	0.47	0.38
	SMonth	TA	0.47	0.15	0.08	2.21	0.78	0.41	0.99	0.90	0.43	0.14	1.46	0.43	0.27
		AN	0.39	0.14	0.07	1.57	0.61	0.33	0.93	0.50	0.44	0.17	1.23	0.38	0.25
	SQuarter	TA	0.53	0.15	0.08	1.76	0.71	0.37	0.94	0.62	0.38	0.12	1.58	0.52	0.32
		AN	0.44	0.14	0.05	1.68	0.57	0.38	1.10	0.66	0.36	0.16	1.50	0.48	0.30
	SSemester	AN	0.79	0.14	0.06	2.41	1.21	0.42	1.49	0.94	0.53	0.18	1.80	0.94	0.48
SF 300	SMonth	AN	0.88	0.33	0.13	4.63	1.57	0.98	2.69	1.55	1.04	0.36	4.07	1.08	0.72
	SQuarter	AN	1.27	0.34	0.11	5.21	2.07	1.01	2.86	1.83	1.06	0.43	4.89	1.51	0.96

Figura 52 - Tempos Médios de Processamento de cada Query (em segundos) - Cenário NS.

Contudo, para validar se de facto as tabelas do modelo com os atributos necessários obtiveram melhores resultados é relevante analisar também a Figura 53, que proporciona uma análise sobre os tempos totais médios de processamentos das *queries*. Esta permite confirmar que, de facto, a "SMonth" foi a tabela que obteve o melhor desempenho em todos os SFs (apesar de não ser possível, para o SF 300, realizar uma comparação entre os cenários TA e AN). Em adição, também se confirma que as tabelas com menor número de atributos obtiveram quase sempre um melhor desempenho que as tabelas com todos os atributos. A redução no tempo despendido a processar as *queries* variou entre 3.15% e 78.39%. A exceção foram as tabelas "SQuarter" e "SSemester" do SF 30, com uma perda de desempenho a rondar os 5%.

Tendo em conta que o Druid é uma base de dados orientada a colunas, à priori, não se esperava que a redução do número de atributos tivesse um grande impacto no desempenho. No entanto, os segmentos de dados no Druid são carregados para memória, tal como já foi explicado no capítulo 3. Assim, face às configurações definidas, o sistema operativo vai gerir que segmentos de dados é que são carregados até, no limite, esgotar a memória definida. Neste caso as queries, para além de recorrer aos dados em memória vão também recorrer aos dados armazenados no disco, o que reduz o desempenho. No caso em que o modelo de dados tem menos atributos, cada segmento de dados vai ocupar menos espaço (tal como este cenário já demonstrou), apesar de conter a mesma informação em termos do número de linhas (o que deixou de existir foram atributos não necessários para responder às queries). Desta forma, como os segmentos de dados serão mais pequenos, será possível carregar mais segmentos

de dados para memória, do que no cenário com todos os atributos o que se traduzirá numa melhoria do desempenho (Druid, 2018g, 2018d).



	SF 30								SF 100								SF 300			
	SDay		SMonth		SQuarter		SSemester		SDay		SMonth		SQuarter		SSemester		SMonth		SQuarter	
	TA	AN	TA	AN	TA	AN	TA	AN	TA	AN	TA	AN	TA	AN	AN		AN		AN	
Run 1 (s)	42.06	5.83	8.96	3.36	5.43	5.49	5.66	6.24	63.05	29.32	14.70	10.82	12.71	11.57	18.63		30.95		37.40	
Média (s)	18.85	4.07	4.37	2.22	3.21	3.37	3.58	3.77	29.58	13.30	8.71	7.02	8.08	7.83	11.39		20.02		23.55	
Dif. Média (%)		-78.39%		-49.19%		4.94%		5.41%		-55.02%		-19.39%		-3.15%						17.64%

Figura 53 - Tempos Totais de Processamento das Queries - Cenário NS.

Cenário NSQ - Segment Granularity e Query Granularity

Neste cenário explorar-se-á o impacto causado pela definição da *query granularity*. Importa relembrar que esta funcionalidade está intrinsecamente ligada ao modelo de dados, visto que é responsável por fazer a agregação dos dados durante a sua ingestão e, para que esta agregação ocorra, o modelo de dados não pode gerar todas as linhas distintas, tal como já foi explicado no capítulo 3.

É ainda relevante mencionar que, a fim de poder ter uma granularidade que sumarizasse mais os dados, e assim avaliar melhor o potencial desta funcionalidade, foi adicionada a *query granularity* mês. Contudo, nos testes em que esta granularidade for utilizada, a *query* 1.3 não poderá ser respondida, visto que utiliza o atributo "od_weeknuminyear", que foi removido para que este teste pudesse ser efetuado (este atributo possui o detalhe da semana, pelo que, se fosse utilizado inibiria a sumarização dos dados, para a granularidade mês).

Através da análise da Tabela 17 é possível aferir como evoluíram as características das tabelas, face à definição da *query granularity*. Desta forma, confirma-se que esta funcionalidade tem um impacto

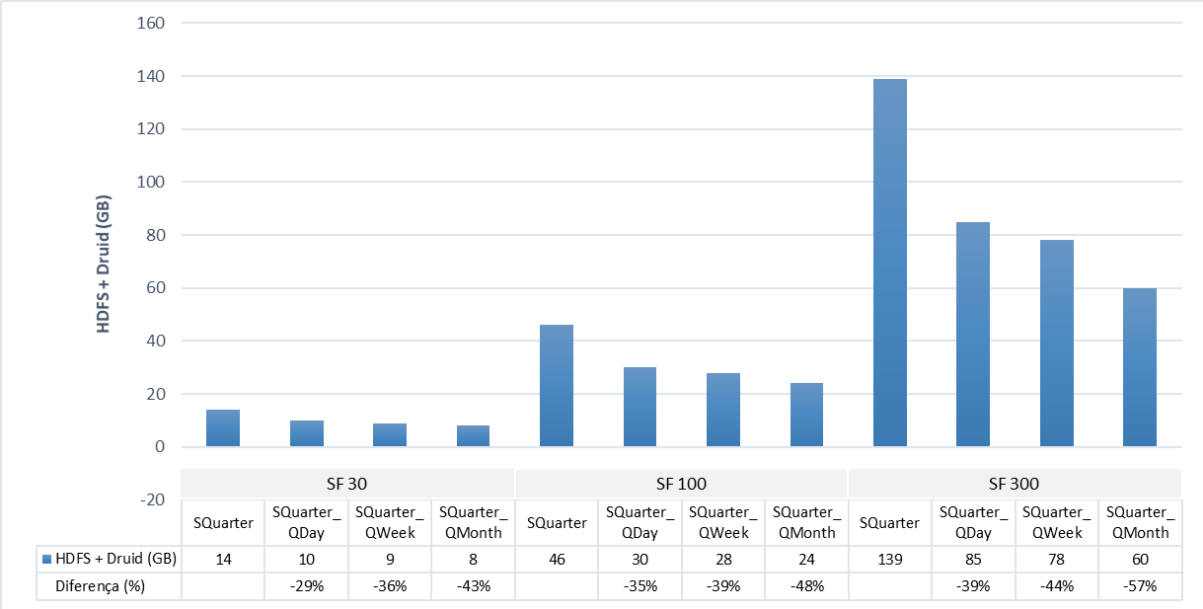
positivo (de redução) no espaço necessário para armazenamento das tabelas, assim como no tamanho e número de linhas em média guardadas num *shard*. Este impacto deve-se ao facto da ação da *query granularity* promover uma redução do número de linhas antes da sua ingestão (através da agregação), não havendo necessidade de guardar todos os dados. Para o SF 300, a tabela "SQuarter_QDay", por exemplo, promoveu uma redução do número de linhas por *shard* de 0.76% (de $66.67*10^6$ para $66.16*10^6$). Quando se analisa a tabela "SQuarter_QWeek", este impacto aumenta, tendo alcançado uma redução de 3.55% (de $66.67*10^6$ para $64.30*10^6$). Por fim, se a granularidade for definida para o nível do mês ("SQuarter_QMonth") a redução do número de linhas por *shard* atinge os 18.73% (de $66.67*10^6$ para $54.18*10^6$).

Tabela 17 - Informação Geral sobre as Tabelas do Cenário NSQ.

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
30	SMonth_QDay	4	6	80	80	74	18	2.25
	SMonth_QWeek	4	5			66		2.24
	SQuarter_QDay	4	6	27	27	220		6.67
	SQuarter_QWeek	4	5			195		6.64
	SQuarter_QMonth	3	5			178	17	6.52
	SSemester_QDay	4	6	14	14	425	18	12.85
	SSemester_QWeek	4	5			376		12.81
	SSemester_QMonth	3	5			343	17	12.58
100	SMonth_QDay	12	18	80	80	226	18	7.48
	SMonth_QWeek	11	17			206		7.41
	SQuarter_QDay	12	18	27	27	667		22.17
	SQuarter_QWeek	11	17			611		21.95
	SQuarter_QMonth	10	14			526	17	20.70
	SSemester_QDay	12	18	14	14	1290	18	42.75
	SSemester_QWeek	11	17			1180		42.34
	SSemester_QMonth	10	14			1010	17	39.91
300	SMonth_QDay	35	50	80	80	630	18	22.33
	SMonth_QWeek	32	46			574		21.70
	SQuarter_QDay	35	50	27	27	1870		66.16
	SQuarter_QWeek	32	46			1700		64.30
	SQuarter_QMonth	25	35			1310	17	54.18

A Figura 54 permite que se faça agora uma análise à diferença no espaço ocupado pelas tabelas, causada pela aplicação da *query granularity*. Nesta figura foi apenas utilizada a tabela "SQuarter" e as tabelas criadas a partir desta com definição de *query granularity*, nos três SFs (30, 100, 300), de forma a facilitar a análise (isto porque o impacto nas outras tabelas é semelhante e pode ser derivado a partir da Tabela 16 e da Tabela 17). Assim, por análise desta figura, verifica-se que no mínimo houve uma

redução de 29% no espaço necessário para armazenamento da tabela (espaço no HDFS + Druid) e, no máximo, existiu uma redução de 57%.



	SF 30				SF 100				SF 300			
	SQuarter	SQuarter_QDay	SQuarter_QWeek	SQuarter_QMonth	SQuarter	SQuarter_QDay	SQuarter_QWeek	SQuarter_QMonth	SQuarter	SQuarter_QDay	SQuarter_QWeek	SQuarter_QMonth
HDFS + Druid (GB)	14	10	9	8	46	30	28	24	139	85	78	60
Diferença (%)		-29%	-36%	-43%		-35%	-39%	-48%		-39%	-44%	-57%

Figura 54 - Diferença no Espaço Ocupado, causada pela Aplicação da Query Granularity.

Já foi possível perceber-se que a *query granularity* tem efetivamente um efeito positivo nas características das tabelas (ao nível do espaço ocupado e número de linhas guardadas). Sendo assim, importa agora perceber se este efeito positivo é acompanhado por uma melhoria de desempenho. Para tal, começando pela análise dos tempos médios de processamento de cada *query*, expostos na Figura 55, é percetível que, no geral, o desempenho alcançado pelas tabelas que tiram partido da *query granularity* é superior ao desempenho das tabelas com o *raw data* (dados sem agregações), visto que obtêm os menores tempos num maior número de *queries*, em todos os SFs.

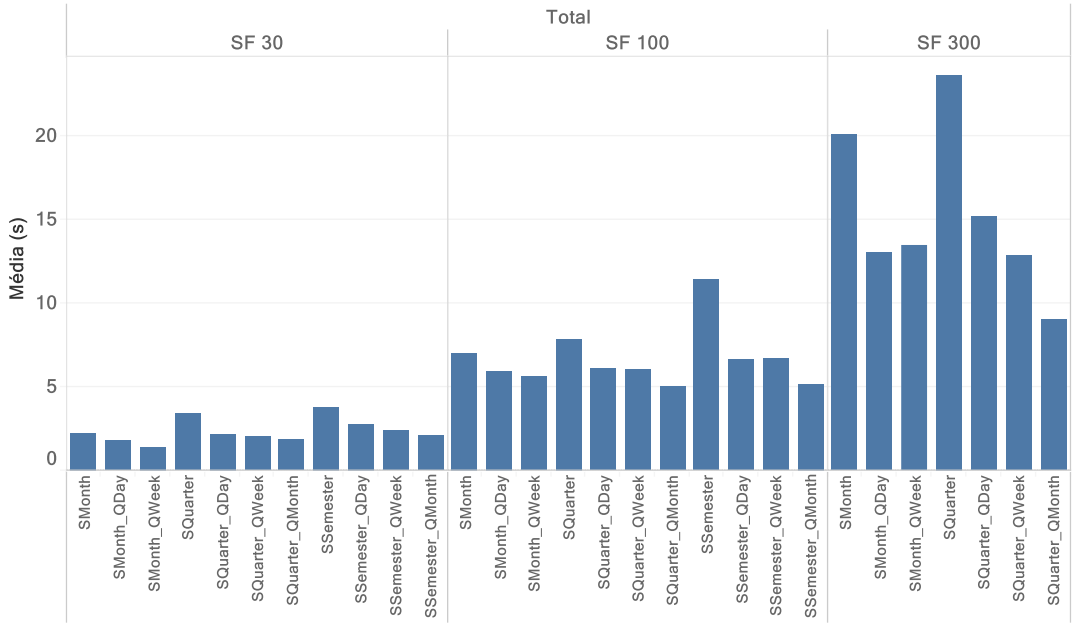
		Query												
SF	Tabela	Q 1.1	Q 1.2	Q 1.3	Q 2.1	Q 2.2	Q 2.3	Q 3.1	Q 3.2	Q 3.3	Q 3.4	Q 4.1	Q 4.2	Q 4.3
SF 30	SMonth	0.12	0.05	0.03	0.50	0.16	0.11	0.28	0.17	0.14	0.06	0.38	0.12	0.09
	SMonth_QDay	0.10	0.05	0.03	0.40	0.16	0.11	0.20	0.08	0.09	0.06	0.31	0.11	0.07
	SMonth_QWeek	0.10	0.05	0.03	0.31	0.21	0.11	0.13	0.04	0.03	0.03	0.23	0.08	0.06
	SQuarter	0.15	0.06	0.02	0.94	0.22	0.14	0.54	0.38	0.15	0.06	0.42	0.17	0.12
	SQuarter_QDay	0.14	0.06	0.02	0.58	0.24	0.12	0.22	0.10	0.11	0.05	0.34	0.12	0.07
	SQuarter_QWeek	0.14	0.05	0.04	0.66	0.21	0.14	0.17	0.05	0.05	0.03	0.30	0.09	0.06
	SQuarter_QMonth	0.13	0.05		0.59	0.25	0.16	0.18	0.04	0.03	0.02	0.31	0.08	0.05
	SSemester	0.29	0.06	0.03	0.80	0.27	0.15	0.44	0.41	0.18	0.07	0.59	0.30	0.18
	SSemester_QDay	0.24	0.06	0.03	0.64	0.23	0.15	0.28	0.12	0.13	0.11	0.42	0.21	0.11
	SSemester_QWeek	0.25	0.05	0.04	0.54	0.34	0.16	0.20	0.05	0.04	0.03	0.43	0.16	0.08
	SSemester_QMonth	0.24	0.05		0.54	0.24	0.19	0.15	0.04	0.05	0.04	0.31	0.13	0.06
SF 100	SMonth	0.39	0.14	0.07	1.57	0.61	0.33	0.93	0.50	0.44	0.17	1.23	0.38	0.25
	SMonth_QDay	0.34	0.18	0.06	2.40	0.70	0.30	0.54	0.08	0.07	0.03	0.90	0.21	0.10
	SMonth_QWeek	0.31	0.20	0.10	2.39	0.61	0.37	0.41	0.06	0.04	0.03	0.83	0.20	0.09
	SQuarter	0.44	0.14	0.05	1.68	0.57	0.38	1.10	0.66	0.36	0.16	1.50	0.48	0.30
	SQuarter_QDay	0.47	0.16	0.05	1.90	0.70	0.37	0.56	0.11	0.09	0.04	1.05	0.35	0.20
	SQuarter_QWeek	0.46	0.14	0.06	2.30	0.58	0.39	0.46	0.06	0.06	0.03	1.06	0.31	0.11
	SQuarter_QMonth	0.49	0.14		1.84	0.58	0.35	0.38	0.05	0.03	0.03	0.81	0.22	0.08
	SSemester	0.79	0.14	0.06	2.41	1.21	0.42	1.49	0.94	0.53	0.18	1.80	0.94	0.48
	SSemester_QDay	0.74	0.13	0.06	1.53	0.85	0.44	0.64	0.12	0.11	0.05	1.22	0.51	0.22
	SSemester_QWeek	0.73	0.13	0.07	1.96	0.74	0.48	0.53	0.07	0.04	0.04	1.30	0.43	0.15
	SSemester_QMonth	0.66	0.14		1.45	0.67	0.40	0.45	0.05	0.03	0.03	0.89	0.32	0.08
SF 300	SMonth	0.88	0.33	0.13	4.63	1.57	0.98	2.69	1.55	1.04	0.36	4.07	1.08	0.72
	SMonth_QDay	0.86	0.34	0.11	3.96	1.84	0.91	1.05	0.15	0.11	0.07	2.78	0.58	0.23
	SMonth_QWeek	0.82	0.32	0.13	4.20	1.89	0.88	1.02	0.11	0.07	0.06	3.31	0.49	0.15
	SQuarter	1.27	0.34	0.11	5.21	2.07	1.01	2.86	1.83	1.06	0.43	4.89	1.51	0.96
	SQuarter_QDay	1.25	0.38	0.10	4.61	1.85	1.17	1.20	0.16	0.11	0.05	3.21	0.73	0.28
	SQuarter_QWeek	1.24	0.37	0.16	3.93	1.68	1.03	1.10	0.12	0.07	0.04	2.20	0.72	0.22
	SQuarter_QMonth	0.97	0.26		3.05	1.03	0.71	0.88	0.07	0.03	0.04	1.50	0.37	0.07

Figura 55 - Tempos Médios de Processamento de cada Query (em segundos) - Cenário NSQ.

Recorrendo agora à Figura 56, complementa-se a análise da evolução do desempenho ditado pela utilização de *query granularity*, confirmando-se que a aplicação desta funcionalidade foi vantajosa em todos os SFs, uma vez que as tabelas que recorreram a ela obtiveram os melhores resultados. No caso do SF 30, a tabela que obteve melhor desempenho foi a "SMonth_QWeek", que conseguiu uma redução de cerca de 36% (de 2.22s para 1.42s) na média de tempo necessário para processar todas as *queries,* face à tabela "SMonth" que não possui os dados agregados. Analisando outro exemplo, desta vez para o SF 300, a "SQuarter_QMonth" obteve os melhores resultados que são cerca de 62% inferiores à média obtida pela "SQuarter".

É interessante destacar que, no caso do SF 100, se não tivesse sido aplicada a *query granularity* mês, a tabela "SMonth_QWeek" teria alcançado o melhor desempenho do SF em questão. Outra nota relevante é o resultado obtido pela tabela "SMonth_QWeek", para o SF 300, ser pior do que o obtido pela "SMonth_QDay", demonstrando, mais uma vez, que os resultados podem, por vezes, ser um pouco instáveis. Contudo, no geral, existe uma tendência para que os melhores desempenhos a considerar em cada *segment granularity*, estejam presentes na tabela que mais agrega os dados, ou seja, que aplica

na *query granularity* um período menos detalhado (a *query granularity* mês, por exemplo, no geral, obtém melhores resultados que a *query granularity* semana, considerando o mesmo *segment granularity*).



SF 30

	SMonth	SMonth_QDay	SMonth_QWeek	SQuarter	SQuarter_QDay	SQuarter_QWeek	SQuarter_QMonth	SSemester	SSemester_QDay	SSemester_QWeek	SSemester_QMonth
Run 1 (s)	3.36	2.54	2.11	5.49	3.70	3.34	3.11	6.24	4.04	4.00	3.47
Média (s)	2.22	1.76	1.42	3.37	2.17	1.97	1.88	3.77	2.73	2.36	2.05
Dif. Média (%)		-20.50%	-19.76%		-35.63%	-9.24%	-4.47%		-27.57%	-13.49%	-13.41%

SF 100

	SMonth	SMonth_QDay	SMonth_QWeek	SQuarter	SQuarter_QDay	SQuarter_QWeek	SQuarter_QMonth	SSemester	SSemester_QDay	SSemester_QWeek	SSemester_QMonth
Run 1 (s)	10.82	11.03	11.22	11.57	10.35	10.23	9.35	18.63	11.50	12.56	9.45
Média (s)	7.02	5.91	5.63	7.83	6.03	6.03	5.00	11.39	6.63	6.68	5.18
Dif. Média (%)		-15.77%	-4.78%		-22.90%	-0.14%	-16.98%		-41.81%	0.78%	-22.50%

SF 300

	SMonth	SMonth_QDay	SMonth_QWeek	SQuarter	SQuarter_QDay	SQuarter_QWeek	SQuarter_QMonth
Run 1 (s)	30.95	23.22	22.76	37.40	25.91	23.65	16.45
Média (s)	20.02	13.01	13.45	23.55	15.10	12.86	8.99
Dif. Média (%)		-35.01%	3.37%		-35.88%	-14.84%	-30.09%

Figura 56 - Tempos Totais de Processamento das Queries - Cenário NSQ.

Antes de terminar as considerações sobre este cenário, é importante destacar que as melhorias de desempenho e a redução do espaço de armazenamento, causadas pela aplicação da *query granularity*, acarretam um custo. Isto é, à medida que se agregam os dados, perde-se a capacidade de

consultar os eventos originais, com o máximo detalhe. Ou seja, a *query granularity* determina a granularidade mínima a que irá ser possível consultar os dados agregados. Assim, é importante que antes da aplicação desta funcionalidade se esteja ciente que o aumento de desempenho (pela aplicação da *query granularity*) acarreta menor flexibilidade na exploração dos dados, o que influencia as operações de *drill-down* (Druid, 2018d).

Refira-se, ainda, que se os dados do SSB possuíssem um detalhe temporal mais específico (exemplo: hora ou segundo, ao invés de dia) seria expectável um impacto superior. A utilização da *query granularity* mês foi precisamente para se poder observar uma maior influência sobre os dados do SSB, tendo em conta as suas características. Se se considerarem dados de cliques num *website*, com informação temporal ao milissegundo, por exemplo, estes dados não terão grande valor para serem analisados individualmente. Nestes casos o ganho pela aplicação da *query granularity* poderá ser maior (aplicado sobre detalhes temporais que originalmente são muito específicos) e sem grandes custos, visto que os registos individuais normalmente não possuem interesse (Druid, 2018d).

Cenário NSQP - Segment Granularity, Query Granularity e Hashed Partitions

As tabelas utilizadas no cenário NSQ, possuíam características próximas ou até abaixo do espaço médio recomendado por *shard*. Contudo, esta redução no espaço ocupado, face às tabelas que foram sendo utilizadas no cenário T, deveu-se à diminuição do número de atributos considerados no modelo de dados, tal como se verificou anteriormente. Assim, o número médio de linhas por *shard*, em alguns casos, continua a apresentar valores ainda distantes dos recomendados.

Posto isto, neste cenário estudar-se-á se a aplicação de *hashed partitions* irá ser capaz de melhorar o desempenho obtido no cenário NSQ. É relevante referir que a escolha da *segment granularity* a utilizar no SF 300 fez-se de acordo com os melhores resultados do cenário NSQ e NSQP para o SF 30 e SF 100. Sendo assim, foram selecionados como *segment granularity* o mês e o trimestre. Optou-se por não utilizar mais combinações de granularidades, por se considerar que já existia um número de testes suficientemente representativo e porque cada tabela do SF 300 demora em média 3 horas e 43 minutos a ser carregada, o que podia atrasar o decorrer da parte experimental da dissertação.

Mesmo assim, este cenário possui um grande número de tabelas, pelo que a apresentação dos resultados de todas elas seria complexa. Neste sentido, a análise realizada neste cenário irá contemplar apenas as tabelas do SF 300 que serão suficientes para retirar as conclusões necessárias sobre este cenário, sendo que, os resultados dos restantes SFs podem ser consultados nos apêndices 4 e 5. No

final será apresentada uma tabela resumo com os melhores resultados e que incluirá também o SF 30 e o SF 100.

A Tabela 18 apresenta as características das várias tabelas utilizadas neste cenário para o SF 300. A tabela completa com informações sobre os restantes SFs pode ser consultada no Apêndice 2 – Tabela Completa do Cenário NSQP. Tal como sucedera no cenário TSP, pela análise da Tabela 18, volta a verificar-se que a aplicação de *hashed partitions* acarreta um incremento no espaço necessário para armazenar as tabelas. Se se considerar, por exemplo, as tabelas "SQuarter_QMonth" (Tabela 17) e "SQuarter_QMonth_PHashed20" verifica-se que a última ocupa (no HDFS e Druid) 18.3% mais que a primeira (evoluiu de 60 GB para 71 GB). Destaque-se, no entanto, que este impacto é bastante inferior àqueles que foram observados no cenário TSP, pelo que, também neste aspeto a redução do número de atributos e a aplicação da *query granularity* terão tido um papel fundamental.

Tabela 18 - Informação Geral sobre as Tabelas do Cenário NSQP.

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
300	SMonth_QDay_PHashed5	37	56	80	400	141	18	4.47
	SMonth_QDay_PHashed10	39	59		800	74		2.23
	SMonth_QWeek_PHashed5	34	50		400	124		4.34
	SMonth_QWeek_PHashed10	35	51		800	64		2.17
	SQuarter_QDay_PHashed5	37	56	27	135	416		13.23
	SQuarter_QDay_PHashed10	39	59		270	219		6.62
	SQuarter_QDay_PHashed15	39	61		405	150		4.41
	SQuarter_QDay_PHashed20	40	62		540	115		3.31
	SQuarter_QWeek_PHashed5	34	50		135	367		12.86
	SQuarter_QWeek_PHashed10	35	51		270	190		6.43
	SQuarter_QWeek_PHashed15	36	53		405	130		4.29
	SQuarter_QWeek_PHashed20	36	54		540	99		3.21
	SQuarter_QMonth_PHashed5	27	40		135	296	17	10.84
	SQuarter_QMonth_PHashed10	28	41		270	152		5.42
	SQuarter_QMonth_PHashed15	29	42		405	103		3.61
	SQuarter_QMonth_PHashed20	29	42		540	78		2.71

De forma a analisar agora o impacto das *hashed partitions* no desempenho obtido, apresentam-se, na Figura 57, os tempos médios de processamento das queries do SSB alcançados pelas várias tabelas utilizadas no SF 300. Os resultados para o SF 30 e SF 100 estão disponíveis no Apêndice 3 – Tempos de Processamento por *Query* (segundos)- Cenário NSQP . Estudando a figura supracitada verifica-se que as tabelas particionadas (através de *hashed partitions*), no geral, foram sempre capazes de atingir melhores desempenhos do que as tabelas não particionadas, isto é, obtiveram tempos menores

num maior número de queries. A tabela "SMonth_QDay_PHashed5", por exemplo, foi superior à tabela "SMonth_QDay" para 8 das 13 queries. Já a tabela "SMonth_QWeek_PHashed5" foi capaz de superar a tabela "SMonth_QWeek" em 10 queries. Porém, curiosamente, a tabela que foi superior num maior número de queries, no SF 300, foi a tabela "SQuarter_QMonth" que não é particionada. Importa agora confirmar se o facto de ter obtido menores tempos num maior número de *queries*, implicou a obtenção de um tempo menor para processar todas elas.

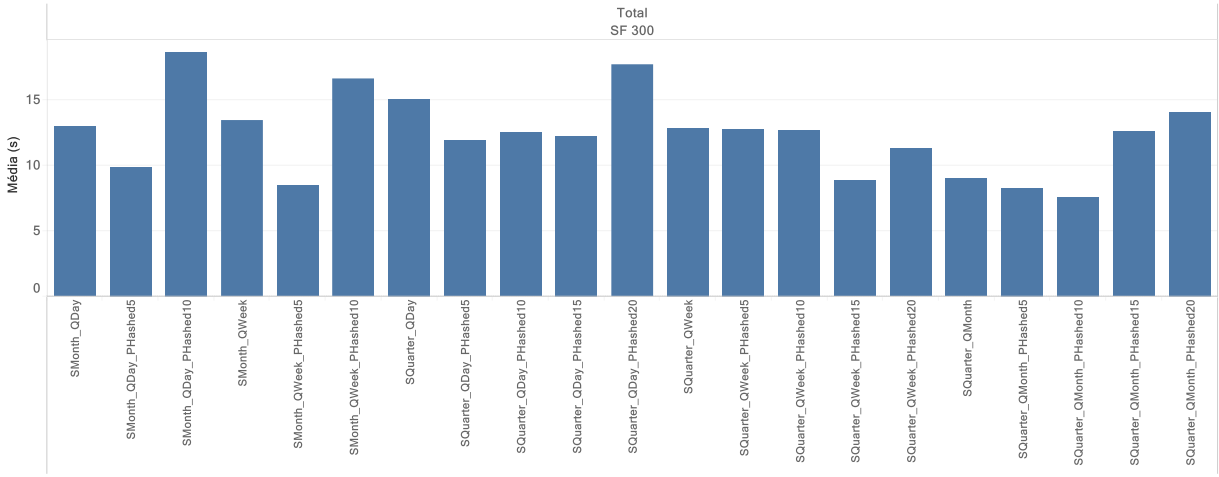
		Query												
SF	Tabela	Q 1.1	Q 1.2	Q 1.3	Q 2.1	Q 2.2	Q 2.3	Q 3.1	Q 3.2	Q 3.3	Q 3.4	Q 4.1	Q 4.2	Q 4.3
SF 300	SMonth_QDay	0.86	0.34	0.11	3.96	1.84	0.91	1.05	0.15	0.11	0.07	2.78	0.58	0.23
	SMonth_QDay_PHashed5	0.63	0.12	0.07	2.40	1.44	0.77	1.17	0.24	0.30	0.10	1.95	0.47	0.24
	SMonth_QDay_PHashed10	0.62	0.10	0.08	9.30	1.48	0.87	2.10	0.72	0.42	0.15	1.99	0.58	0.29
	SMonth_QWeek	0.82	0.32	0.13	4.20	1.89	0.88	1.02	0.11	0.07	0.06	3.31	0.49	0.15
	SMonth_QWeek_PHashed5	0.50	0.09	0.05	2.23	1.30	0.86	0.90	0.15	0.09	0.05	1.69	0.40	0.17
	SMonth_QWeek_PHashed10	0.51	0.09	0.10	8.95	1.65	0.87	1.30	0.70	0.14	0.18	1.59	0.40	0.19
	SQuarter_QDay	1.25	0.38	0.10	4.61	1.85	1.17	1.20	0.16	0.11	0.05	3.21	0.73	0.28
	SQuarter_QDay_PHashed5	0.63	0.11	0.06	3.75	1.43	0.82	1.31	0.25	0.24	0.07	2.28	0.58	0.35
	SQuarter_QDay_PHashed10	0.59	0.11	0.08	3.59	1.47	0.82	1.46	0.36	0.46	0.13	2.51	0.57	0.35
	SQuarter_QDay_PHashed15	0.57	0.10	0.06	3.02	1.52	0.82	1.56	0.42	0.50	0.16	2.45	0.67	0.37
	SQuarter_QDay_PHashed20	0.49	0.10	0.04	7.11	1.38	0.92	2.02	1.26	0.52	0.20	2.56	0.71	0.39
	SQuarter_QWeek	1.24	0.37	0.16	3.93	1.68	1.03	1.10	0.12	0.07	0.04	2.20	0.72	0.22
	SQuarter_QWeek_PHashed5	0.51	0.11	0.07	4.50	1.55	1.08	1.36	0.15	0.15	0.05	2.59	0.47	0.18
	SQuarter_QWeek_PHashed10	0.50	0.12	0.09	4.76	1.51	0.82	1.19	0.16	0.30	0.10	2.54	0.43	0.21
	SQuarter_QWeek_PHashed15	0.47	0.08	0.05	2.29	1.31	0.82	1.05	0.15	0.13	0.05	1.76	0.47	0.22
	SQuarter_QWeek_PHashed20	0.85	0.13	0.07	4.52	1.21	0.79	1.20	0.17	0.14	0.06	1.57	0.41	0.20
	SQuarter_QMonth	0.97	0.26		3.05	1.03	0.71	0.88	0.07	0.03	0.04	1.50	0.37	0.07
	SQuarter_QMonth_PHashed5	0.75	0.08		2.43	1.18	0.78	0.83	0.08	0.04	0.03	1.61	0.31	0.12
	SQuarter_QMonth_PHashed10	0.46	0.08		2.35	1.07	0.76	0.85	0.10	0.05	0.04	1.38	0.34	0.14
	SQuarter_QMonth_PHashed15	0.59	0.10		6.43	1.38	0.77	1.05	0.14	0.12	0.06	1.42	0.37	0.17
	SQuarter_QMonth_PHashed20	0.56	0.11		7.68	1.35	0.74	0.86	0.15	0.18	0.14	1.72	0.36	0.19

Figura 57 - Tempos Médios de Processamento de cada Query (em segundos) - Cenário NSQP.

Como forma de complementar a análise anterior, a Figura 58 providencia uma visão sobre os tempos totais (média dos totais das quatro execuções das *queries*) de processamento das *queries* para todas as tabelas do SF 300. A mesma apresentação para as tabelas do SF 30 é consultável no Apêndice 4 - Tempos Totais de Processamento, SF 30 - Cenário NSQP, enquanto que os resultados para as tabelas do SF 100 estão presentes no Apêndice 5 - Tempos Totais de Processamento, SF 100 - Cenário NSQP. Através da figura supramencionada, conclui-se que a tabela "SQuarter_QMonth_PHashed10" foi a que conseguiu menor tempo total médio de processamento com 7.60 segundos (e melhor tempo também se apenas se considerar apenas a primeira execução das *queries*), um tempo cerca de 15.5% inferior ao tempo médio observado para a tabela não particionada ("SQuarter_QMonth"), comprovando-se uma melhoria causada pela aplicação de *hashed partitions*.

Para a tabela "SMonth_QWeek", por exemplo, também se obtiveram melhorias causadas pelo particionamento, visto que a tabela "SMonth_QWeek_PHashed5" atingiu um desempenho 36.91% superior, quando comparado com a tabela não particionada que lhe deu origem. Estes resultados comprovam que não é apenas o espaço médio ocupado por cada *shard* que influencia o desempenho

obtido, visto que, a tabela "SMonth_QWeek" já possuía um tamanho médio de 574 MB, supostamente enquadrado nos valores considerados recomendáveis. Contudo, tal como anteriormente referido, existiu uma grande diminuição deste espaço médio ocupado, fruto da redução dos atributos considerados no modelo. Desta forma, era expectável que o desempenho pudesse ser melhorado se as tabelas fossem



	SMonth_QDay	SMonth_QDay_PHashed5	SMonth_QDay_PHashed10	SMonth_QWeek	SMonth_QWeek_PHashed5	SMonth_QWeek_PHashed10	SQuarter_QDay	SQuarter_QDay_PHashed5	SQuarter_QDay_PHashed10	SQuarter_QDay_PHashed15	SQuarter_QDay_PHashed20	SQuarter_QWeek	SQuarter_QWeek_PHashed5	SQuarter_QWeek_PHashed10	SQuarter_QWeek_PHashed15	SQuarter_QWeek_PHashed20	SQuarter_QMonth	SQuarter_QMonth_PHashed5	SQuarter_QMonth_PHashed10	SQuarter_QMonth_PHashed15	SQuarter_QMonth_PHashed20
Run 1 (s)	23.22	16.26	32.07	22.76	13.73	30.91	25.91	21.20	19.40	18.02	29.61	23.65	21.12	20.94	14.35	23.81	16.45	14.71	13.63	22.46	24.00
Média (s)	13.01	9.89	18.69	13.45	8.49	16.65	15.10	11.89	12.52	12.23	17.70	12.86	12.76	12.73	8.85	11.31	8.99	8.24	7.60	12.59	14.05
Dif. Média (%)		-23.98%	88.89%		-36.91%	96.21%		-21.30%	5.32%	-2.32%	44.71%		-0.79%	-0.20%	-30.51%	27.84%		-8.35%	-7.74%	65.56%	11.60%

Figura 58 - Tempos Totais de Processamento das Queries, SF 300 - Cenário NSQP.

particionadas, uma vez que, cada *shard* possuía, em média, $21.70*10^6$ linhas, que são valores distantes daqueles que caracterizavam as tabelas que alcançaram os melhores resultados no cenário T.

É ainda relevante fazer uma análise comparativa do aumento no espaço ocupado, causado pela definição de *hashed partitions*, contrabalançado com as diferenças de desempenho obtidas. Com esse intuito, foi desenvolvido esse estudo apenas para a tabelas "SQuarter_QMonth" e as tabelas particionadas originadas por esta, para o SF 300. Esta análise é apresentada na Figura 59, a partir da qual se conclui que para atingir uma melhoria de desempenho de 15%, foi necessário precisamente mais 15% de espaço de armazenamento. Relembra-se que no cenário TSP, para alcançar uma melhoria de 24% no desempenho foi necessário um aumento de 62% no espaço de armazenamento, pelo que também aqui se observaram melhorias que advieram do modelo de dados e da aplicação da *query granularity*.

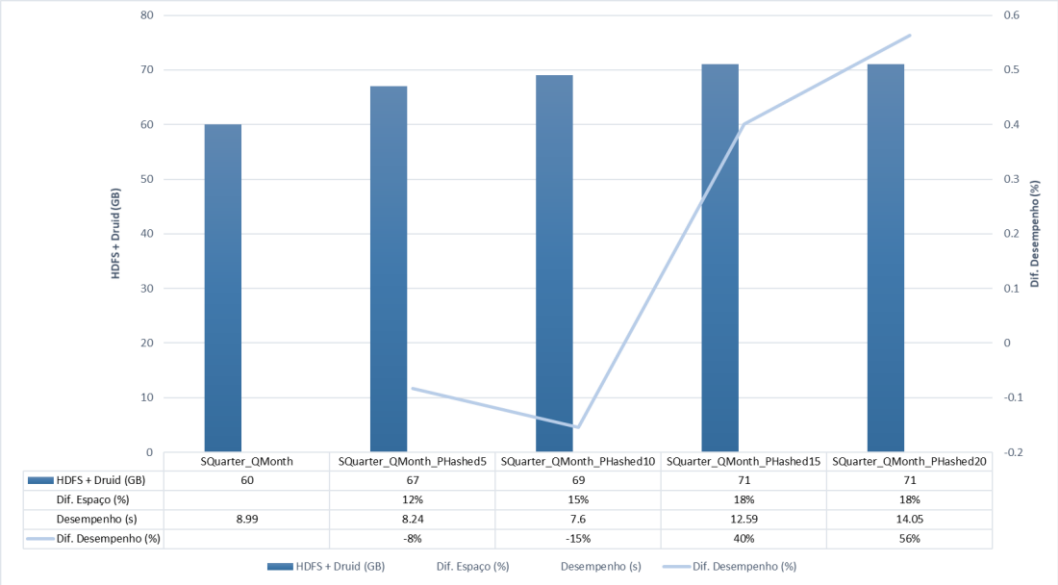


Figura 59 - Comparação entre o Aumento no Espaço Ocupado e as Diferenças no Desempenho - Cenário NSQP.

Finalmente, na Tabela 19, apresentam-se as tabelas que obtiveram os melhores resultados no cenário N, assim como as características de cada uma delas. Comparando estas características com as características das tabelas que obtiveram os melhores resultados no cenário T, verifica-se que existe uma maior influência do número médio de linhas por *shard*, do que do espaço médio ocupado. Além disso, o espaço ocupado mostrou sofrer um forte impacto consoante os atributos considerados no modelo de dados. Desta forma, recomendar o espaço médio que cada *shard* deve ter para se obter um melhor desempenho, pode não ser um método muito fiável. Os resultados demonstram que, preferencialmente, se deve optar por ter em atenção o número médio de linhas por *shard*. É ainda interessante destacar que os melhores resultados no SF 100, tanto no cenário T, como no cenário N, foram alcançados numa tabela com um número médio de linhas por *shard* igual ($3.70*10^6$).

Tabela 19 - Melhores Desempenhos no Cenário N.

SF		Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(x*10⁶)	(s)
T	30	SQuarter_PHashed3	27	55	27	81	683	2.22	2.09
T	100	SQuarter_PHashed6	106	220	27	162	1360	3.70	6.12
N	30	SQuarter_QMonth_PHashed2	3	5	27	54	91	**3.26**	1.35
N	100	SMonth_QWeek_PHashed2	12	17	80	160	106	3.70	3.72
N	300	SQuarter_QMonth_PHashed10	28	41	27	270	152	5.42	7.60

4.4 Hive e Druid: Qual o Potencial da sua Integração?

Nesta subsecção, tal como o título permite antever, explora-se o potencial da integração entre o Hive e o Druid. Para tal, primeiro apresentam-se um conjunto de vantagens da integração destas tecnologias e, de seguida, apresenta-se um *benchmark*. De referir que neste *benchmark* não é objetivo voltar a estudar a influência da *segment granularity*, *query granularity* e *hashed partitions,* no desempenho e características das tabelas, visto que essas questões já foram endereçadas.

Assim, para avaliar esta integração, será utilizada a tabela "SMonth_QDay" para o SF 300, por se tratar de uma tabela sem a definição de *hashed partitions* (mais simples para utilizadores não avançados) e porque se trata de uma configuração já utilizada por Shanklin (2017) na comunidade técnica. Os resultados obtidos no *benchmark* a esta integração serão ainda comparados com os resultados obtidos para a mesma tabela utilizando o Druid como motor de *queries*.

De acordo com Shanklin (2017), o Druid combina o melhor dos repositórios orientados a colunas e índices invertidos. Esta utilização de índices invertidos, segundo o autor, permite que o Druid seja melhor que outras tecnologias SQL-on-Hadoop a responder a *queries* com detalhe muito específico e que necessitam de poucas linhas (5 ou 10, por exemplo) de uma fonte com, potencialmente, milhões de linhas armazenadas. O autor denomina este processo como "encontrar agulhas num palheiro" e refere que à medida que se adicionam mais dimensões, menos linhas são necessárias e mais rapidamente executam as *queries* no Druid.

Tal como já foi referido anteriormente, o Druid não suporta a utilização de *joins*. Existe apenas uma funcionalidade ainda experimental, denominada de *query-time lookups* que permite fazer a junção dos dados do Druid com pequenas tabelas carregadas de sistemas externos (Druid, 2018k). Contudo, os autores do Druid referem que a implementação de *joins* em grande escala não é uma prioridade. Além disso, a funcionalidade do Druid para suportar SQL é também muito recente, pelo que, operações mais complexas como *window functions* ou *joins* se prevê que não venham a ser suportadas num curto espaço de tempo (Druid, 2018n; Shanklin, 2017; Yang et al., 2014).

Apesar da implementação de *joins* não ser uma prioridade para o Druid, esta constitui uma das características do Hive, assim como o suporte de SQL. Posto isto, a integração destes projetos conjuga bem as suas capacidades, beneficiando ambos. O Druid adquire a capacidade de ter a interface SQL providenciada pelo Hive, ou seja, é possível colocar *queries* aos repositórios do Druid através do Hive, para além de criar e eliminar tabelas do Druid através da mesma interface SQL. Isto significa, ainda, que o Druid pode beneficiar da integração do Hive com as ferramentas analíticas (como, por exemplo, o

Tableau) através dos drivers ODBC/JDBC providenciados pela Hortonworks. O Hive, por sua vez, beneficia do potencial dos repositórios do Druid (com informação histórica e *real-time*) com dados agregados, guardados numa estrutura orientada a colunas e com índices invertidos (Shanklin, 2017).

O Hive consegue gerir de forma transparente os repositórios nativos e os repositórios externos, fruto das interfaces providenciadas pelo seu *StorageHandler* (neste caso o *DruidStorageHandler*). Importa ainda referir que o Apache Calcite[33] tem um papel relevante nesta integração, uma vez que é responsável por otimizar, reescrever e dividir o trabalho necessário para responder às *queries*, pelos diferentes motores, baseado nas suas capacidades (Hive, 2017; Shanklin, 2017).

Assim, o Apache Calcite vai gerir o trabalho, de tal forma que as *queries* simples sejam respondidas diretamente pelo Druid, ao passo que as mais complexas vão executar as operações possíveis no Druid e, as restantes serão garantidas pelo Hive. Desta forma, é possível, por exemplo, efetuar um *join* que irá tentar executar partes da *query* no Druid e a junção final é efetuada pelo Hive, auxiliando numa tarefa que o Druid é incapaz de garantir. A Tabela 20 sintetiza a divisão/colaboração de tarefas entre o Hive e o Druid para responderem às *queries* (Shanklin, 2017).

Tabela 20 - Divisão do Trabalho entre Hive e Druid. Adaptado de (Shanklin, 2017).

Camada	Ponto Forte	Funcionalidades
Hive	Operações analíticas complexas e em larga escala	• *Joins* • *Subqueries* • *Window Functions* • Transformações • Agregações Complexas • Ordenações Avançadas • *User Defined Functions* (UDFs) • Interface SQL • Conectores ODBC/JDBC para ferramentas como Tableau
Druid	Queries que utilizam um grande número de dimensões	• Agregações Simples (agrupadas pelas dimensões – *Group By*) • *Top* N • Valores Min/Max • *Timeseries*

[33] https://calcite.apache.org/

Estudadas algumas das vantagens desta integração, de seguida é apresentado um exemplo, com um caso prático, da integração que pode ser realizada entre o Hive e o Druid. Como pré-requisito para a integração destas duas tecnologias é necessário ativar a funcionalidade *Live Long And Process* do Hive (Hive LLAP) no cluster utilizado como infraestrutura, visto que é aqui providenciado o "DruidStorageHandler" (Hive, 2017).

Para tal, foi necessário fazer algumas configurações via Ambari, nomeadamente:

- Ativar *pre-emption* no YARN;
- Ativar *Interactive Query* no Hive;
- Alterar o número de *nodes* utilizados pelo LLAP para 4;
- Alterar o número máximo de *queries* concorrentes para 4;
- Alterar a memória por *Daemon* para 10750 MB;
- Alterar a *In-memory cache per Daemon* para 3300 MB;
- Alterar para 12 GB a memória máxima para o YARN atribuir aos *containers*.

Estas configurações foram definidas tendo em conta que as configurações por defeito atribuídas ao Hive LLAP, não permitiam *queries* concorrentes e apenas utilizavam um nó do cluster, não fazendo paralelismo. A definição destas configurações teve em consideração as recomendações de Hortonworks (2018a).

De forma a ser possível efetuar um *benchmark* à integração entre Hive e Druid, foi então necessário começar por carregar a tabela "SMonth_QDay" do SF 300, tal como tinha sido referido anteriormente. De referir que a tarefa de ingestão desta tabela teve de sofrer algumas alterações (face à tarefa original de carregamento para o Druid), para suprir limitações que foram encontradas no decorrer dos testes. Uma das limitações prendeu-se com o facto das *queries* a partir do Hive não reconhecerem os cálculos "revenue - supplycost" e "extendedprice * discount", pelo que foi necessário criar métricas com estes campos já calculados e adaptar as *queries*. Além disto, o Hive também se revelou incapaz de utilizar as métricas do Druid nos filtros ("quantity", por exemplo), pelo que foi necessário adicioná-las como dimensões.

Por fim, a versão do Hive utilizada possui um problema na criação dos segmentos no Druid (quando se tentam criar tabelas no Druid a partir do Hive), relacionado com a zona horária. O Druid está configurado para trabalhar em UTC (*Coordinated Universal Time*), enquanto que o Hive está configurado

para a zona horária local, neste caso a de Portugal continental. Este problema, resolvido[34] numa versão mais recente do Hive (que ainda não estava integrada na distribuição Hortonworks), impossibilitou criar repositórios no Druid a partir do Hive.

De forma a contornar todos estes problemas os dados foram ingeridos no Druid, tal como havia sido feito na secção 4.2, sendo depois criada uma tabela externa no Hive a referenciar os dados armazenados no Druid. A tarefa de ingestão utilizada está disponível em https://github.com/jmcorreia/Druid_SSB_Benchmark, no ficheiro que se denomina "analytical_obj300_SMonth_QDay_hive_druid.json". Desta tarefa de ingestão é relevante destacar a utilização do JavaScript *aggregator* do Druid que permitiu criar métricas a partir de operações sobre os atributos originais, sem a necessidade de processos de ETL (Druid, 2018a). Destaca-se ainda a necessidade de definir, via Ambari, a propriedade "druid.javascript.enabled = true", para que seja possível utilizar esta funcionalidade.

A Figura 60 demonstra a utilização do JavaScript *aggregator* para a criação dos atributos "net_revenue" (corresponde à operação "revenue – supplycost") e "discounted_price" (corresponde à operação "extendedprice * discount"). Feito isto as *queries* do SSB foram adaptadas para utilizar estes dois novos atributos. Tal como anteriormente, foram desenvolvidas *scripts* para executar cada uma das *queries* 4 vezes, sendo que estas *scripts* (contendo as *queries* adaptadas), podem ser consultadas em https://github.com/jmcorreia/Druid_SSB_Benchmark. Importa ainda referir que serão apresentados os tempos de processamento obtidos recorrendo ao comando "time", visto que também foi esta a abordagem utilizada para recolher os tempos de processamento do Druid e se pretende uma comparação rigorosa. Contudo, e tendo em conta que na comunidade académica e técnica utilizam, por vezes, o tempo de execução devolvido pela Beeline[35] (os trabalhos de E. Costa (2017) e E. Costa et al. (2017) são um exemplo desta utilização) este tempo também foi guardado e pode ser consultado nos apêndices 6 e 7.

[34] https://issues.apache.org/jira/browse/HIVE-19155?attachmentOrder=asc
[35] Cliente JDBC para consultar os dados armazenados no Hive.

```
"metricsSpec" : [
        {
          "type" : "doubleSum",
          "name" : "revenue",
          "fieldName" : "revenue"
        },
        {
          "type": "javascript",
          "name": "net_revenue",
          "fieldNames": ["revenue", "supplycost"],
          "fnAggregate" : "function(current, revenue, supplycost) { return current + (revenue - supplycost); }",
          "fnCombine": "function(partialA, partialB) { return partialA + partialB; }",
          "fnReset": "function() { return 0; }"
        },
        {
          "type": "javascript",
          "name": "discounted_price",
          "fieldNames": ["extendedprice", "discount"],
          "fnAggregate" : "function(current, extendedprice, discount) { return current + (extendedprice * discount); }",
          "fnCombine": "function(partialA, partialB) { return partialA + partialB; }",
          "fnReset": "function() { return 0; }"
        }
      ]
    },
```

Figura 60 - Utilização do Javascript Aggregator para criar atributos "net_revenue" e "discounted_price".

Carregada a tabela para o Druid através de um processo semelhante ao referido na secção 4.2.1, foi necessário criar uma tabela externa no Hive, a referenciar o repositório guardado no Druid, tal como se demonstra na Figura 61.

```
CREATE EXTERNAL TABLE analytical_obj300_SMonth_QDay_hive
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
TBLPROPERTIES ("druid.datasource" = "analytical_obj300_SMonth_QDay_hive");
```

Figura 61 - Criação da Tabela Externa no Hive.

A Figura 62 apresenta os tempos de processamento do Hive a consultar os repositórios do Druid, por comparação com os tempos anteriormente obtidos utilizando o Druid como motor de consultas. Ressalva-se novamente que os tempos recolhidos pela Beeline são consultáveis no Apêndice 6 – Tempos de Processamento Obtidos no Hive (tempo impresso pela Beeline) e no Druid. Pela análise da figura referida, conclui-se que utilizando o Hive como motor de consulta levou a um aumento no tempo de processamento em todas as *queries*, totalizando-se uma perda de 66% de desempenho em relação ao tempo médio obtido na primeira execução das *queries* no Druid e 76% em relação à média das quatro execuções.

	Hive		Druid	
Query	Run 1 (s)	Média (s)	Run 1 (s)	Média (s)
Q 1.1	11.80	5.22	3.32	0.86
Q 1.2	4.59	3.12	1.26	0.34
Q 1.3	3.98	3.49	0.37	0.11
Q 2.1	11.60	7.45	7.67	3.96
Q 2.2	4.09	5.62	2.49	1.84
Q 2.3	3.97	3.94	1.00	0.91
Q 3.1	3.53	4.11	1.06	1.05
Q 3.2	2.46	3.00	0.15	0.15
Q 3.3	3.44	3.13	0.12	0.11
Q 3.4	3.31	3.03	0.07	0.07
Q 4.1	8.07	5.88	4.76	2.78
Q 4.2	3.00	3.66	0.71	0.58
Q 4.3	3.51	3.39	0.25	0.23
Total	67.34	55.03	23.22	13.01
Dif. (%)	——	——	- 66	- 76

Figura 62 - Tempos de Processamento Obtidos no Hive e no Druid.

Estes resultados serão ainda comparados com os tempos obtidos nos trabalhos de E. Costa (2017) e E. Costa et al. (2017) na secção 4.7.

Apesar dos desempenhos terem sido inferiores (tempos mais elevados) aos obtidos utilizando o Druid como motor de consultas, esta integração acarreta outras vantagens, como por exemplo a possibilidade de utilizar o Tableau para fazer análises aos repositórios do Druid, tal como se exemplifica na Figura 63. Verifica-se, também, que o Hive acaba por beneficiar do mecanismo de *cache* do Druid, tendo tempos de execução mais pequenos após a primeira execução das *queries*.

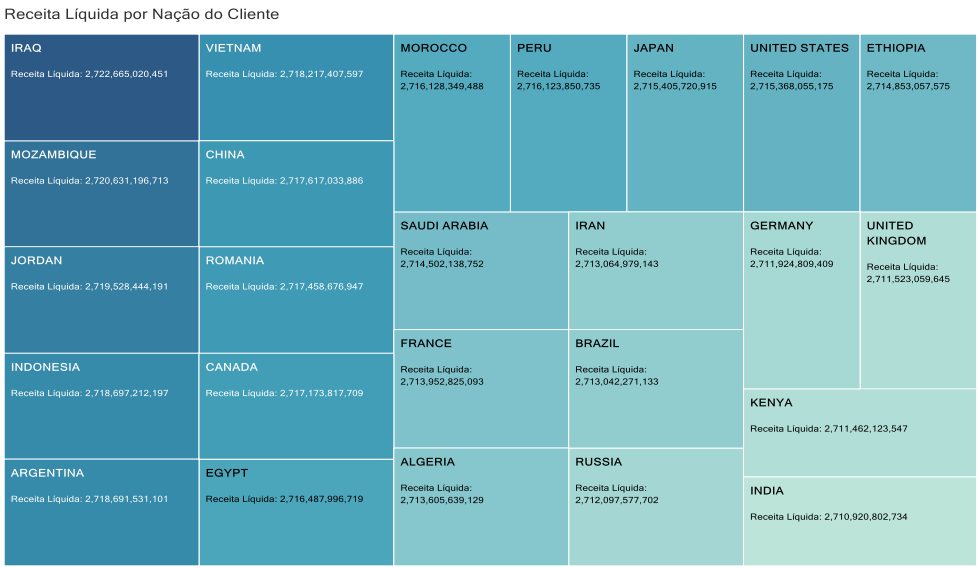


Figura 63 - Dashboard Exemplo no Tableau.

Além disto, Shanklin (2017) refere que no futuro esta integração terá continuidade e melhorias, por exemplo no que diz respeito à velocidade de indexação dos dados. Para terminar, importa destacar que o Hive e a funcionalidades que acrescenta ao Druid, podem constituir um importante auxílio para o crescimento e aceitação desta tecnologia, visto que, numa fase inicial o Druid se revela uma ferramenta complexa. O Hive e a integração do Druid na distribuição Hortorworks permite que utilizadores menos avançados possam configurar e utilizar o Druid sem a necessidade de recorrer à sua linguagem de *query* nativa, e utilizar ferramentas de visualização a que não estão acostumados, entre outros aspetos, que podem afastar possíveis utilizadores (como o facto do Druid, nativamente, não suportar *joins*).

4.5 Hive e Druid em Ambiente Multiutilizador

Em cenários reais, como por exemplo o ambiente do projeto de investigação no qual esta dissertação se enquadra, o BDW não vai ser consultado apenas por um utilizador, podendo haver necessidade de suportar vários em simultâneo. Desta forma, é relevante perceber o desempenho destas tecnologias em ambiente multiutilizador, a fim de detetar qual o padrão nas alterações de desempenho, em comparação com os resultados em ambiente com apenas um utilizador.

Desta forma, as *scripts* utilizadas nos testes anteriores deste capítulo, foram adaptadas para distribuir as *queries* por quatro utilizadores em simultâneo, de forma diferente. Isto é, quando um utilizador está a executar uma *query*, os outros utilizadores estão a executar uma *query* diferente. Recorrendo a este método, pretende-se simular um contexto o mais real possível, no qual vários utilizadores em simultâneo fazem diferentes consultas. Sendo assim, e como foram utilizados quatro utilizadores neste teste, o utilizador 1 começa com a ordem normal das *queries* do SSB, o utilizador 2 começa por executar a *query* Q2.1 e termina a executar a Q1.3, o utilizador 3 começa pela *query* Q3.1 e termina na *query* Q2.3 e assim sucessivamente. As scripts utilizadas encontram-se disponíveis em https://github.com/jmcorreia/Druid_SSB_Benchmark.

A Figura 64 apresenta os resultados alcançados pelo Hive e pelo Druid em ambiente multiutilizador. Desde logo é percetível que o Druid alcançou um desempenho 59% superior no que diz respeito à primeira execução das *queries* e 67% superior na média das quatro execuções das *queries*, em relação ao Hive. Contudo, comparando estes resultados com aqueles que foram anteriormente obtidos num ambiente com um único utilizador, o Hive evidencia que é menos afetado por receber *queries* de diferentes utilizadores em simultâneo, visto que, os tempos obtidos aumentaram cerca de 40% na primeira execução das queries (“Run 1”) e cerca de 49% na média das quatro execuções. Embora

o Druid seja mais afetado que o Hive, neste cenário, o seu desempenho continua a ser satisfatório, uma vez que percentagens de aumento abaixo dos 300%, significam que, num ambiente com quatro utilizadores, o sistema foi capaz de executar as *queries* mais rapidamente do que se as executasse quatro vezes num contexto com um único utilizador.

	Hive		Druid	
Query	Run 1 (s)	Média (s)	Run 1 (s)	Média (s)
Q 1.1	10.53	6.21	4.69	1.19
Q 1.2	6.39	5.84	0.60	0.17
Q 1.3	6.77	5.63	0.16	0.06
Q 2.1	10.66	8.07	8.15	4.38
Q 2.2	6.98	7.18	3.93	3.39
Q 2.3	6.31	6.62	2.25	2.14
Q 3.1	9.56	7.64	4.84	4.83
Q 3.2	5.66	5.57	1.21	1.44
Q 3.3	5.53	5.93	0.90	1.23
Q 3.4	5.09	4.53	1.45	1.29
Q 4.1	9.97	7.44	7.27	4.60
Q 4.2	6.03	6.60	1.88	1.47
Q 4.3	5.13	4.80	1.35	1.23
Total	94.59	82.06	38.69	27.42
Dif. (%)	——	——	- 59	- 67
Aumento (%)	40	49	67	111

Figura 64 - Tempos de Processamentos Obtidos no Hive e Druid em Ambiente Multiutilizador.

No sentido de perceber o efeito da *cache* do Druid, na Figura 65 apresentam-se os resultados obtidos para o utilizador 1, o utilizador 2 e as médias dos 4 utilizadores, tirando ou não partido do mecanismo de *cache* do Druid. É importante relembrar que o utilizador 1 executa as queries do SSB na ordem normal, enquanto que o utilizador 2 começa na *query* Q2.1.

Assim, começando por analisar os resultados que tiram partido da *cache*, na parte da figura circundada a verde, é possível apurar que o utilizador 2 consegue beneficiar do facto do utilizador 1 já ter executado as *queries* Q1.1, Q1.2 e Q1.3 (obtém melhores desempenhos que o utilizador 1 nestas *queries*), ao passo que o utilizador 1 beneficia do facto do utilizador 2 já ter executado as *queries* Q2.1, Q2.2 e Q2.3.

Analisando agora os resultados obtidos com o mecanismo de *cache* desligado, verifica-se que contínua a existir um benefício, ou seja, um aproveitamento do facto das *queries* já terem sido efetuadas anteriormente por outros utilizadores, embora este benefício não seja tão evidente. O utilizador 2, por exemplo, para a *query* Q1.1 obteve agora uma média de 2.76s, que é melhor do que a obtida pelo

utilizador 1 (9.92s). No entanto, o tempo obtido foi superior aos 0.02s necessários pelo utilizador 2, com recurso à *cache*.

Este comportamento benéfico entre utilizadores e *queries* mantém-se, mesmo quando se desliga a *cache* no Druid, porque à medida que vão sendo feitas *queries*, os segmentos de dados que as servem vão sendo carregados para memória, de tal forma que, quando as próximas *queries* são efetuadas, estas podem beneficiar, se alguns dos segmentos de dados que necessitam ainda estiverem carregados em memória. O ganho de desempenho que deixou de existir foi o da *cache* do Druid ao nível do Broker (nó que recebe as *queries*), que guarda os resultados obtidos, por segmento, das *queries* que vão sendo respondidas. Este mecanismo permite que quando se recebe uma *query*, depois de a mapear para os segmentos de dados que serão necessários para devolver uma resposta, se verifique se existem resultados pré-computados para alguns desses segmentos. Caso existam esses resultados serão utilizados, não havendo necessidade de os voltar a computar.

	Cache / Utilizador											
	SIM						NÃO					
	Utilizador 1		Utilizador 2		Média		Utilizador 1		Utilizador 2		Média	
Query	Run 1	Média	Run 1	Média	Run 1	Média	Run 1	Média	Run 1	Média	Run 1	Média
Q 1.1	18.68	4.70	0.02	0.02	4.69	1.19	27.53	9.92	2.84	2.76	9.84	5.00
Q 1.2	2.36	0.61	0.02	0.02	0.60	0.17	1.47	1.36	1.33	1.31	1.42	1.35
Q 1.3	0.60	0.18	0.02	0.02	0.16	0.06	0.38	0.65	0.30	0.36	0.35	0.42
Q 2.1	7.13	4.65	17.98	6.32	8.15	4.38	4.70	4.85	17.88	6.61	7.39	5.25
Q 2.2	1.63	2.51	3.13	3.51	3.93	3.39	5.14	5.70	7.51	3.77	4.43	3.52
Q 2.3	1.59	2.33	3.80	3.25	2.25	2.14	1.26	2.00	2.57	5.32	1.46	2.88
Q 3.1	7.39	4.73	3.85	5.37	4.84	4.83	8.50	5.30	3.71	3.44	5.43	4.28
Q 3.2	1.08	1.22	1.79	1.45	1.21	1.44	0.67	1.41	3.27	2.65	2.15	1.97
Q 3.3	0.83	0.51	1.00	1.11	0.90	1.23	1.00	1.58	1.58	1.13	1.07	1.41
Q 3.4	1.37	1.05	0.28	0.73	1.45	1.29	0.26	1.21	0.16	1.03	1.03	1.14
Q 4.1	4.92	3.86	1.89	3.03	7.27	4.60	1.42	1.61	4.95	4.01	5.53	4.41
Q 4.2	1.23	0.95	2.33	1.51	1.88	1.47	0.46	0.45	1.77	1.96	1.30	1.49
Q 4.3	0.74	0.90	1.17	1.14	1.35	1.23	0.16	0.19	0.60	1.31	1.05	1.14
Total	49.55	28.23	37.27	27.46	38.69	27.42	52.95	36.23	48.46	35.65	42.44	34.25

Figura 65 - Tempos de Processamento por Utilizador, com e sem Cache.

Antes de terminar é relevante realçar que estes resultados foram alcançados recorrendo às configurações por defeito dos sistemas, exceção feita às situações necessárias e que foram referidas. Sendo assim, algumas destas configurações podem não ser as mais adequadas, principalmente para ambientes concorrenciais que normalmente exigem parametrização.

O Druid, tal como já foi mencionado, carrega os seus segmentos de dados para memória, enquanto tiver memória disponível. Se não existir memória suficiente o sistema irá gerir o carregamento e descarregamento de segmentos da memória, sendo que, quando uma *query* é efetuada, se o segmento não estiver carregado em memória será necessário ler dados do disco. Num ambiente com muitos utilizadores, com várias *queries* em simultâneo, certamente haverá necessidade de carregar mais

segmentos para memória o que poderá fazer com que esta atinja o limite e, nesse caso, exigir que se vá ler muitas vezes dados ao disco, degradando o desempenho. A memória disponível para esta operação é uma das configurações que foi deixada por defeito e que podia influenciar os resultados aqui alcançados.

4.6 Estudo dos Resultados face ao Tipo de *Queries*

Nesta secção começa-se por fazer uma caracterização das *queries* do SSB, de acordo com as suas características, nomeadamente: número de colunas e função de agregação utilizada no *SELECT*, os atributos considerados no *WHERE*, *GROUP BY*, *ORDER BY* e respetivas cardinalidades. Este estudo é apresentado na Tabela 21.

Tabela 21 - Caracterização das Queries do SSB.

Query	SELECT		WHERE				GROUP BY			ORDER BY		
	Nº Colunas	Função de Agregação	Nº Condições	Atributos	Cardinalidade	Condição	Nº Atributos	Atributos	Cardinalidade	Nº Atributos	Atributos	Cardinalidade
Q1.1	1	SUM(extendedprice*discount)	3	od_year	7	od_year = 1993						
				discount	Contínuo	discount between 1 and 3						
				quantity	Contínuo	quantity < 25						
Q1.2	1	SUM(extendedprice*discount)	3	od_yearmonthnum	80	od_yearmonthnum = 199401						
				discount	Contínuo	discount between 4 and 6						
				quantity	Contínuo	quantity between 26 and 35						
Q1.3	1	SUM(extendedprice*discount)	4	od_weeknuminyear	53	od_weeknuminyear = 6						
				od_year	7	od_year = 1994						
				discount	Contínuo	discount between 5 and 7						
				quantity	Contínuo	quantity between 26 and 35						
Q2.1	3	SUM(revenue)	2	p_category	25	p_category = 'MFGR#12'	2	od_year	7	2	od_year	7
				s_region	5	s_region = 'AMERICA'		p_brand	975		p_brand	975
Q2.2	3	SUM(revenue)	2	p_brand	975	p_brand1 between 'MFGR#2221' and 'MFGR#2228'	2	od_year	7	2	od_year	7
				s_region	5	s_region = 'ASIA'		p_brand	975		p_brand	975
Q2.3	3	SUM(revenue)	2	p_brand	975	p_brand1= 'MFGR#2239'	2	od_year	7	2	od_year	7
				s_region	5	s_region = 'EUROPE'		p_brand	975		p_brand	975
Q3.1	4	SUM(revenue)	3	c_region	5	c_region = 'ASIA'	3	c_nation	25	2	od_year	7
				s_region	5	s_region = 'ASIA'		s_nation	25		revenue	Contínuo
				od_year	7	od_year >= 1992 and od_year <= 1997		od_year	7			
Q3.2	4	SUM(revenue)	3	c_nation	25	c_nation = 'UNITED STATES'	3	c_city	225	2	od_year	7
				s_nation	25	s_nation = 'UNITED STATES'		s_city	225		revenue	Contínuo
				od_year	7	od_year >= 1992 and od_year <= 1997		od_year	7			
Q3.3	4	SUM(revenue)	3	c_city	225	c_city='UNITED KI1' or c_city='UNITED KI5'	3	c_city	225	2	od_year	7
				s_city	225	s_city='UNITED KI1' or s_city='UNITED KI5'		s_city	225		revenue	Contínuo
				od_year	7	od_year >= 1992 and od_year <= 1997		od_year	7			
Q3.4	4	SUM(revenue)	3	c_city	225	c_city='UNITED KI1' or c_city='UNITED KI5'	3	c_city	225	2	od_year	7
				s_city	225	s_city='UNITED KI1' or s_city='UNITED KI5'		s_city	225		revenue	Contínuo
				od_yearmonth	80	od_yearmonth = 'Dec1997'		od_year	7			
Q4.1	3	sum(revenue - supplycost)	3	c_region	5	c_region = 'AMERICA'	2	od_year	7	2	od_year	7
				s_region	5	s_region = 'AMERICA'		c_nation	25		c_nation	25
				p_mfgr	5	p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2'						
Q4.2	4	sum(revenue - supplycost)	4	c_region	5	c_region = 'AMERICA'	3	od_year	7	3	od_year	7
				s_region	5	s_region = 'AMERICA'		s_nation	25		s_nation	25
				od_year	7	od_year = 1997 or od_year = 1998		p_category	25		p_category	25
				p_mfgr	5	p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2'						
Q4.3	4	sum(revenue - supplycost)	4	c_region	5	c_region = 'AMERICA'	3	od_year	7	3	od_year	7
				s_nation	5	s_nation = 'UNITED STATES'		s_city	225		s_city	225
				od_year	7	od_year = 1997 or od_year = 1998		p_brand	975		p_brand	975
				p_category	25	p_category = 'MFGR#14'						

A Figura 66 sintetiza os resultados previamente apresentados nas secções 4.4 e 4.5 para o Hive e para o Druid, num ambiente com um único utilizador e num ambiente multiutilizador. Foram escolhidos estes cenários, de forma a ser também possível verificar se o comportamento observado no Hive se aproxima daquele que é visível no Druid, sendo importante relembrar que as operações "revenue – supplycost" e "extendedprice * discount" se encontram pré-calculas nas métricas "net_revenue" e "discounted_price", no caso do Hive, o que pode ter alguma influência nos resultados das *queries,* para as quais estes cálculos eram efetuados durante a execução da *query*. Através desta figura é possível verificar que, principalmente no Druid, existe uma clara tendência para a primeira *query* de cada grupo de *queries* demorar mais tempo a ser processada (no grupo de queries Q1.x, por exemplo, a *query* mais demorada é a *query* Q1.1) e o desempenho das seguintes ser gradualmente melhor. Os casos em que este fenómeno não acontece encontram-se circundados a verde na figura.

Quando o motor de consulta utilizado é o Druid, só no ambiente multiutilizador, na passagem da *query* 3.3 para a *query* 3.4 é que o comportamento anteriormente referido não se verificou, sendo que, a diferença na média foi apenas de 1.23s para 1.29s. Por outro lado, quando se utiliza o Hive, já existem mais situações que não seguem este comportamento, tal como é percetível pelos casos destacados na figura. Deixa-se a nota que, curiosamente, se se considerarem os tempos devolvidos pela Beeline (apêndices 6 e 7), o comportamento do Hive aproxima-se muito mais daquele que é observado para o Druid.

Um Utilizador					Multiutilizador				
	Hive		Druid			Hive		Druid	
Query	Run 1 (s)	Média (s)	Run 1 (s)	Média (s)	Query	Run 1 (s)	Média (s)	Run 1 (s)	Média (s)
Q 1.1	11.80	5.22	3.32	0.86	Q 1.1	10.53	6.21	4.69	1.19
Q 1.2	4.59	3.12	1.26	0.34	Q 1.2	6.39	5.84	0.60	0.17
Q 1.3	3.98	3.49	0.37	0.11	Q 1.3	6.77	5.63	0.16	0.06
Q 2.1	11.60	7.45	7.67	3.96	Q 2.1	10.66	8.07	8.15	4.38
Q 2.2	4.09	5.62	2.49	1.84	Q 2.2	6.98	7.18	3.93	3.39
Q 2.3	3.97	3.94	1.00	0.91	Q 2.3	6.31	6.62	2.25	2.14
Q 3.1	3.53	4.11	1.06	1.05	Q 3.1	9.56	7.64	4.84	4.83
Q 3.2	2.46	3.00	0.15	0.15	Q 3.2	5.66	5.57	1.21	1.44
Q 3.3	3.44	3.13	0.12	0.11	Q 3.3	5.53	5.93	0.90	1.23
Q 3.4	3.31	3.03	0.07	0.07	Q 3.4	5.09	4.53	1.45	1.29
Q 4.1	8.07	5.88	4.76	2.78	Q 4.1	9.97	7.44	7.27	4.60
Q 4.2	3.00	3.66	0.71	0.58	Q 4.2	6.03	6.60	1.88	1.47
Q 4.3	3.51	3.39	0.25	0.23	Q 4.3	5.13	4.80	1.35	1.23
Total	67.34	55.03	23.22	13.01	Total	94.59	82.06	38.69	27.42

Figura 66 - Síntese dos Resultados Alcançados pelo Hive e pelo Druid.

Tendo estes resultados como base, é possível concluir que, dentro de um grupo semelhante de *queries*, a alteração de algumas das condições pelas quais os dados são filtrados não produziu, no geral, um impacto negativo no desempenho obtido, mesmo quando se passam a considerar atributos com

maior cardinalidade. Este comportamento explica-se pelo facto do Druid carregar para memória os segmentos de dados necessários para responder às *queries* na primeira vez que é executada. Assim, as *queries* seguintes, sendo semelhantes a *queries* anteriores irão beneficiar do facto de alguns dos segmentos de dados necessários ainda estarem em memória, o que permite alcançar um melhor desempenho. Por força deste mecanismo, é difícil aferir qual o tipo de queries (com maior ou menor número de filtros, maior ou menor granularidade, entre outros aspetos) que, na prática, exige mais tempo, embora seja possível verificar que, no ambiente com um único utilizador, tanto no Druid como no Hive, as queries mais exigentes tenham sido a Q2.1 e Q4.1 (se o critério for a média de tempos de processamento obtidos).

4.7 Síntese de Resultados

Esta secção marca o final do atual capítulo e, como tal, são aqui tecidas algumas considerações sobre os diferentes aspetos abrangidos pelos testes e respetivos resultados alcançados. De entre estas considerações destacam-se as recomendações sobre as melhores estratégias a seguir na implementação de um BDW no Druid e a comparação do desempenho desta tecnologia com outras SQL-on-Hadoop.

Estratégias de Implementação de BDW no Druid

Ao longo dos testes efetuados neste capítulo, foi possível perceber-se que os diferentes aspetos estudados (*segment granularity*, *query granularity*, *hashed partitions* e a modelação) impactam o desempenho e as características das tabelas armazenadas no Druid.

Recorrendo aos testes efetuados no cenário T, conclui-se que a *segment granularity* influencia diretamente o número de segmentos, nos quais os dados de cada tabela se vão subdividir, sendo que uma granularidade muito detalhada (dia, por exemplo) gera muitos segmentos pequenos (em número de linhas e espaço ocupado), degradando o desempenho e ocupando mais espaço de armazenamento, por incapacidade de compactar adequadamente os dados. Por outro lado, a definição de uma *segment granularity* pouco detalhada, pode implicar a criação de tabelas subdivididas em poucos segmentos muito grandes, o que também degrada o desempenho no processamento de *queries*. Ainda no cenário T, a aplicação de *hashed partitions* revelou-se um benefício para o desempenho, embora tenha implicado aumentos na memória ocupada pelas tabelas. A tabela "SQuarter_PHashed6", por exemplo, que obteve os menores tempos de processamento, atingiu um desempenho 24% inferior à tabela sem *hashed partitions* ("SQuarter"), contudo implicou 64% mais espaço de armazenamento.

Na análise aos resultados obtidos nos vários testes do cenário N, evidenciou-se que a modelação tem uma grande influência no espaço ocupado pelas tabelas, no seu tempo de ingestão e no desempenho. Comparando os resultados obtidos no cenário NS, com aqueles que foram obtidos no cenário TS para as tabelas com a mesma *segment granularty*, constata-se que a redução de atributos no modelo de dados foi capaz de gerar uma poupança no espaço de armazenamento entre 75% e 86%. No que ao tempo de ingestão diz respeito, as alterações ao modelo de dados, permitiram uma redução de cerca de 63% nas tabelas do SF 30, 68% nas tabelas do SF 100 e permitiram ainda que as tabelas do SF 300 fossem carregadas 12% mais rapidamente, em média, do que as tabelas do SF 100 no cenário T. Esta melhoria ocorre, visto que é necessário efetuar o cálculo de índices para um menor número de atributos, demorando menos tempo no processo de carregamento das tabelas e ocupando menos espaço de armazenamento, por não ser necessário guardar tantos índices, quando comparado com o cenário T. Finalmente, quanto ao desempenho observou-se que a redução de atributos no modelo, por si só, foi capaz de permitir uma melhoria entre 3.15% e 78.39%, salvo duas exceções que foram as tabelas "SQuarter" e "SSemester" do SF 30, que obtiveram uma perda de desempenho a rondar os 5%.

A *query granularity* revelou-se uma funcionalidade com potencial para otimizar o desempenho e reduzir o espaço necessário de armazenamento. A sua aplicação, no cenário NSQ, garantiu uma redução do espaço ocupado pelas tabelas entre 29% e 57%, dependendo da granularidade utilizada (dia, semana, mês, sendo que quanto menos detalhada for a granularidade definida, maior será a poupança no espaço de armazenamento), quando comparadas com as tabelas no cenário NS que não utilizam esta funcionalidade. Ao nível do desempenho, as tabelas com aplicação de *query granularity* conseguiram, no geral, tempos médios de processamento das *queries* inferiores aos das tabelas que não aplicam esta funcionalidade, sendo que se verificou a tendência dos melhores desempenhos serem obtidos nas tabelas que aplicam uma *query granularity* menos detalhada, ou seja, capaz de agregar mais os dados e consequentemente obter uma maior redução do espaço necessário de armazenamento.

No cenário NSQP, a utilização de *hashed partitions* revelou-se vantajosa para atingir menores tempos de processamento, tal como já tinha sucedido no cenário TSP. No entanto, fruto da aplicação combinada das *hashed partitions*, com a *query granularity* e a redução de atributos no modelo, o aumento no desempenho causado pela utilização de *hashed partitions*, não se repercutiu num aumento tão grande no espaço necessário de armazenamento, quando comparado com o cenário TSP. A tabela "SQuarter_QMonth_PHashed10" foi a que obteve um melhor desempenho no SF 300, tendo alcançado um tempo de processamento médio das *queries* 15% inferior ao alcançado pela tabela "SQuarter_QMonth", sendo que obrigou a um aumento no espaço de armazenamento na mesma

proporção (15%). Relembra-se que no cenário TSP, uma melhoria de 24% no desempenho, pela aplicação de *hashed partitions*, ditou um aumento de 62% no espaço necessário de armazenamento.

A Figura 67 expõe as características (espaço e número de linhas médio por *shard*) das tabelas que obtiveram os melhores desempenhos nos cenários T e N, nos vários SFs estudados. Esta figura permite verificar que recomendar um intervalo de valores para o espaço médio ocupado por *shard* pode não ser uma boa opção, visto que esta característica é fortemente influenciada pelo número de atributos considerados no modelo de dados. O número de linhas médio por *shard* revelou ser um melhor indicador, se se pretender estabelecer intervalos de valores recomendáveis para obter melhores desempenhos. O SF 100 é um bom exemplo do que acaba de ser referido, visto que se se tiver como referência o espaço médio ocupado por *shard*, houve uma redução de 92% (1360 MB para 106 MB), do cenário T para o cenário N, enquanto que, se se considerar como valor de referência o número de linhas médio por *shard*, este valor foi idêntico em ambos os cenários ($3.7*10^6$). Continuando a análise à figura supramencionada, também se evidencia a tendência do número médio de linhas por *shard,* característico das tabelas com melhor desempenho, aumentar à medida que o volume de dados aumenta, sendo que o mesmo se verifica para o espaço médio ocupado por *shard*.

Baseado nestes resultados, um utilizador que pretenda tirar partido das *hashed partitions*, pode definir a propriedade "numShards" para um valor que gere *shards* com número de linhas médio próximo dos referenciados nesta dissertação ou definir a propriedade "targetPartitionSize" com um valor do intervalo recomendado. Se o volume de dados a considerar for próximo de 100 GB, por exemplo, pode definir-se a propriedade "targetPartitionSize" igual a $3.7*10^6$. Estas características, provavelmente, possibilitarão melhorar o desempenho, contudo, para otimizar este desempenho ao máximo será sempre necessário fazer vários testes modificando uma das propriedades em questão. A diferença entre elas é que na propriedade "numShards" cada segmento será dividido no número de *shards* especificado, enquanto que na propriedade "targetPartitionSize" o número de *shards* será calculado automaticamente, tendo em conta o número de linhas por *shard* definido (Druid, 2018b).

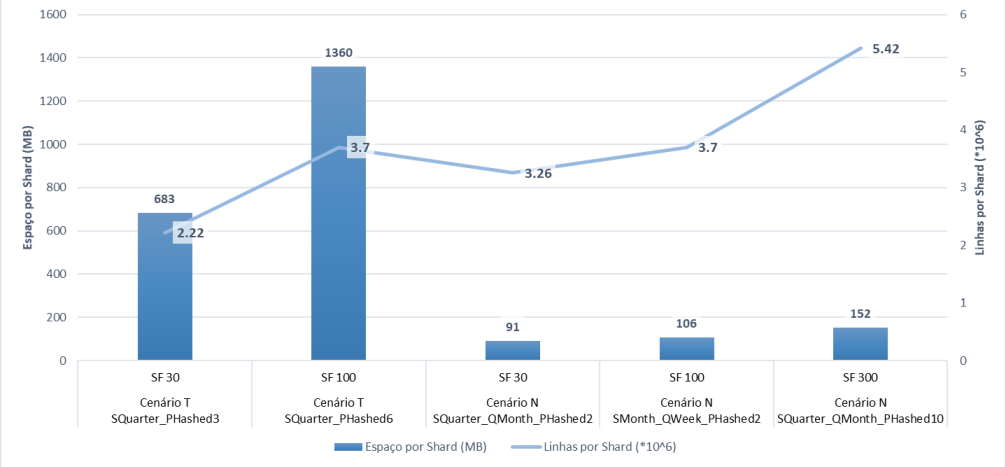


Figura 67 - Características das Tabelas com Melhor Desempenho.

Como forma de perceber a evolução do desempenho face à variação do volume de dados, a Figura 68 apresenta os melhores desempenhos alcançados no cenário T e no cenário N, para os SFs estudados e exibe o aumento que a variação no volume de dados causou no desempenho. Tal permite concluir que a evolução do desempenho com o aumento do volume de dados não é linear. Além disso, recorrendo à mesma imagem, verifica-se que os resultados obtidos no cenário N foram superiores (cerca de 35% no SF 30 e cerca de 39% no SF 100) aos obtidos no cenário T.

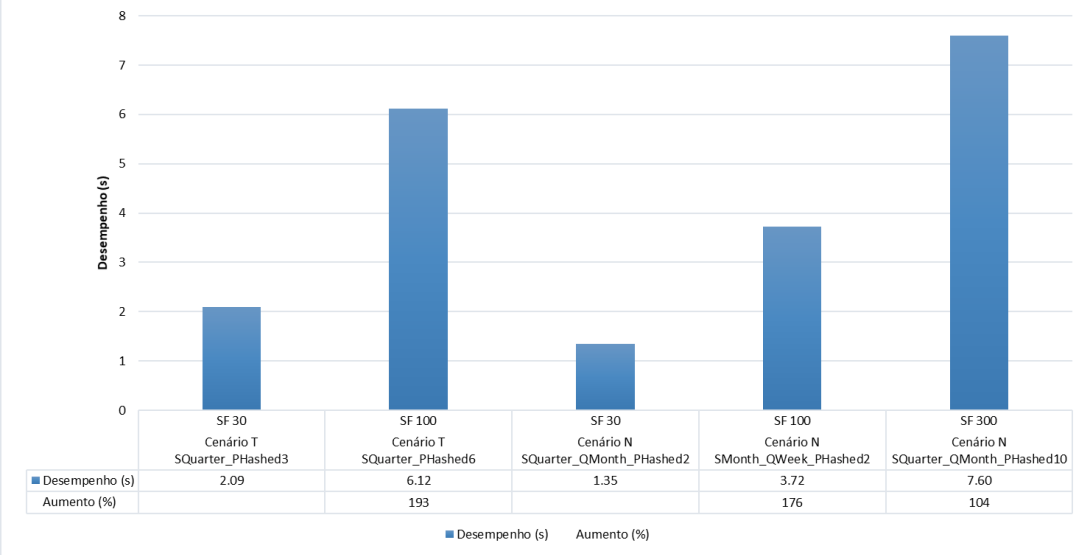


Figura 68 - Evolução do Desempenho face ao Volume de Dados.

No que diz respeito aos testes que foram efetuados ao potencial da integração entre o Hive e o Druid, constatou-se que estas tecnologias podem beneficiar mutuamente. O Druid adquire uma interface SQL para gerir e fazer *queries* sobre os seus repositórios, sendo que estas *queries* incluem *joins* e outras operações complexas que sem esta integração não suportadas pelo Druid. Além disso, passa a ser possível utilizar os conectores ODBC/JDBC do Hive para fazer análises aos repositórios do Druid,

utilizando, por exemplo, o Tableau. O Hive, por sua vez, beneficia da forma otimizada de realizar consultas OLAP tirando partido da forma como o Druid armazena tanto dados históricos, como dados em real-time, alcançando bons desempenhos. Em ambiente multiutilizador, ambas as tecnologias demonstraram um bom desempenho, superior ao tempo que seria necessário para efetuar as *queries* 4 vezes em ambiente com um utilizador.

Por fim, nas secções 4.5 e 4.6, foi ainda possível perceber a influência da *cache* e da gestão dos segmentos que vão sendo carregados para memória no desempenho das *queries* às tabelas do Druid. Verificou-se que a *cache* permite que sejam alcançados em média bons desempenhos e que, em ambiente multiutilizador, os utilizadores possam beneficiar do facto de *queries* muito semelhantes já terem sido executadas por outros utilizadores. Quanto à gestão dos segmentos em memória, evidenciou-se que a primeira *query* de cada grupo de *queries* semelhantes (exemplo: Q1.1 no grupo de *queries* Q1.x) tem tendência para demorar mais tempo a processar, uma vez que, nessa altura, os segmentos necessários são carregados para memória e as *queries* subsequentes podem beneficiar do facto de alguns dos segmentos de que necessitam para ser respondidas, ainda estarem em memória.

Face às considerações que já foram sendo tecidas nesta secção, definem-se as seguintes recomendações na implementação de um BDW no Druid:

1. **Modelação** – estudar o tipo de análises a que uma determinada tabela irá usualmente responder e, mediante os atributos necessários para responder a essas análises, definir o modelo de dados. É relevante ter em atenção que o modelo de dados impacta não só o espaço ocupado e o desempenho na resposta a *queries*, mas também a possível aplicação, ou não, da *query granularity*. Se no modelo de dados forem considerados atributos que possuem valores únicos (chaves incrementais, por exemplo), todas as linhas de registos da tabela serão distintas e, como tal, impossíveis de serem agregadas pela aplicação da *query granularity*.
2. ***Segment Granularity*** – definir uma granularidade capaz de distribuir os dados por segmentos uniformes, ou seja, semelhantes no que diz respeito ao espaço ocupado e número de linhas que possuem e, se possível, que se aproxime de valores recomendáveis. Os resultados desta dissertação demonstram que esta avaliação deve ser feita tendo em conta o número médio de linhas por segmento/ *shard*, num intervalo entre $2.22*10^6$ e $5.42*10^6$, sendo que volumes de dados próximos de 30 GB devem considerar o limite inferior deste intervalo e volumes de dados mais próximos de 300 GB devem considerar o limite superior.

3. ***Query Granularity*** – antes da definição da granularidade a atribuir nesta funcionalidade é importante estar ciente que a mesma implicará o detalhe máximo das análises, ou seja, se for definida uma granularidade dia, os dados serão agregados tendo em conta esse período, perdendo-se a possibilidade de analisar os dados mais detalhadamente. Mediante as necessidades analíticas, definir a granularidade máxima possível para o caso em questão, visto que quanto maior a granularidade mais os dados serão agregados, menor será o espaço ocupado pelos mesmos e melhor será o desempenho alcançado. A *query granularity*, conjugada com a *segment granularity,* constitui um auxílio para se atingirem as características recomendadas à otimização do desempenho, visto que a agregação dos dados diminuirá o número de linhas médio por segmento/*shard*, assim como o espaço médio ocupado.
4. ***Hashed Partitions*** – esta funcionalidade revelou-se capaz de otimizar o desempenho no processamento de *queries*, visto que particiona os dados de uma forma uniforme e parametrizável até que se atinjam características recomendáveis. No entanto, implica um aumento no espaço ocupado o que em alguns cenários pode ser indesejável. Por vezes, também se registou alguma instabilidade nos resultados obtidos, pelo que, a aplicação desta funcionalidade é mais recomendável a utilizadores avançados ou que possuam necessidade de otimizar ao máximo os desempenhos obtidos. Além disso, embora os valores obtidos nesta dissertação possam ser tidos como referência, será sempre necessário efetuar diferentes testes até que se perceba, no cenário em questão, quais as características em termos de número de linhas e espaço médio por *shard*, que melhor otimizam o desempenho.

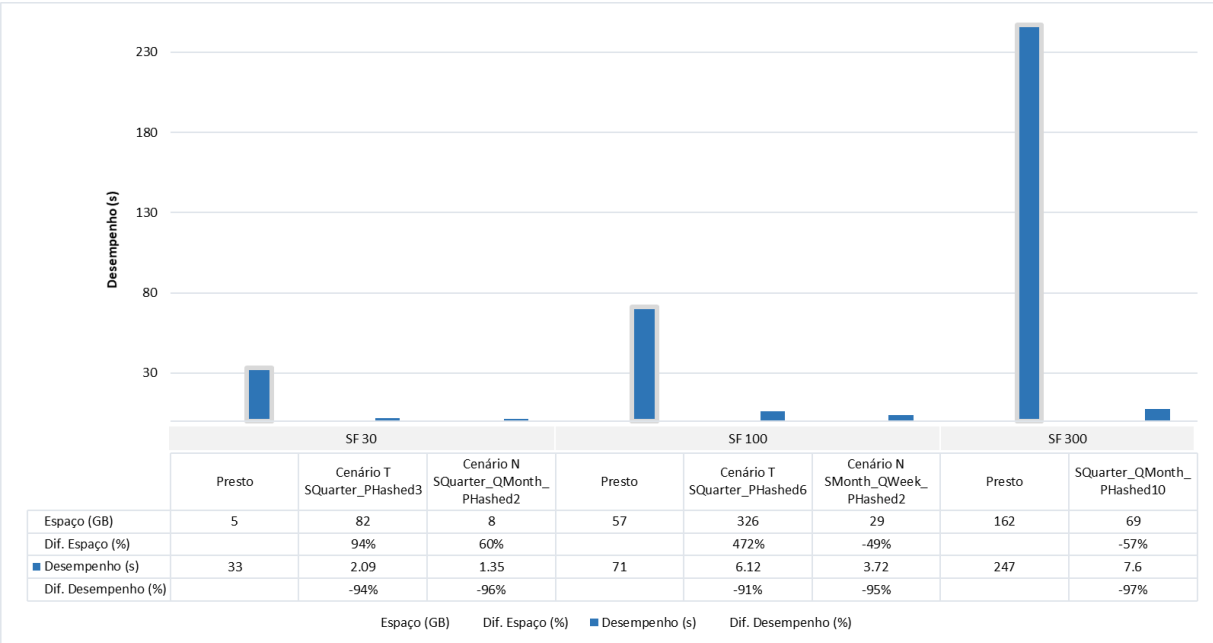
Comparação de Resultados com outras Tecnologias

A Tabela 22 apresenta os melhores resultados obtidos nos trabalhos de E. Costa (2017) e que servirão de referência para avaliar o Druid face a outras tecnologias SQL-on-Hadoop. De referir que os melhores resultados obtidos por este autor foram alcançados com o Presto, em comparação com o Hive, sendo que o Presto alcançou sempre melhor desempenho.

Tabela 22 - Melhores Resultados Obtidos pelo Presto. Baseado em (E. Costa, 2017).

SF	(GB)	(s)
30	5	33
100	57	71
300	162	247

A Figura 69 apresenta os melhores resultados obtidos pelo Presto, retirados dos trabalhos de E. Costa (2017), em comparação com os melhores resultados alcançados pelo Druid (média dos tempos de processamento obtidos durante as quatro execuções das *queries*), nos vários SFs, tanto no cenário T, como no cenário N (neste cenário as tabelas do Druid possuem menos linhas do que as tabelas do Hive consultados pelo Presto, fruto da aplicação das agregações). Para além disso, também está evidenciado na figura a diferença percentual de desempenho entre o Druid e o Presto, assim como o espaço ocupado pelas tabelas de ambas as tecnologias e a diferença percentual entre os espaços ocupados por estas. De referir que, para uma mais fácil identificação, os desempenhos obtidos pelo Presto encontram-se destacados com uma borda de cor diferente. Por análise desta figura, o Druid revela um desempenho muito superior àquele que é alcançado pelo Presto. No mínimo, o Druid apresentou tempos 91% inferiores aos obtidos pelo Presto e, no máximo, 97%. Através da mesma figura, verifica-se que as tabelas do Druid que utilizaram o modelo de dados com todos os atributos (cenário T) ocuparam mais espaço do que as tabelas Hive que armazenam os dados processados pelo Presto. Contudo, no cenário N (utiliza o modelo com menos atributos) as tabelas do Druid ocuparam menos que as do Hive, exceção feita à tabela "SQuarter_QMonth_PHashed2" do SF 30.

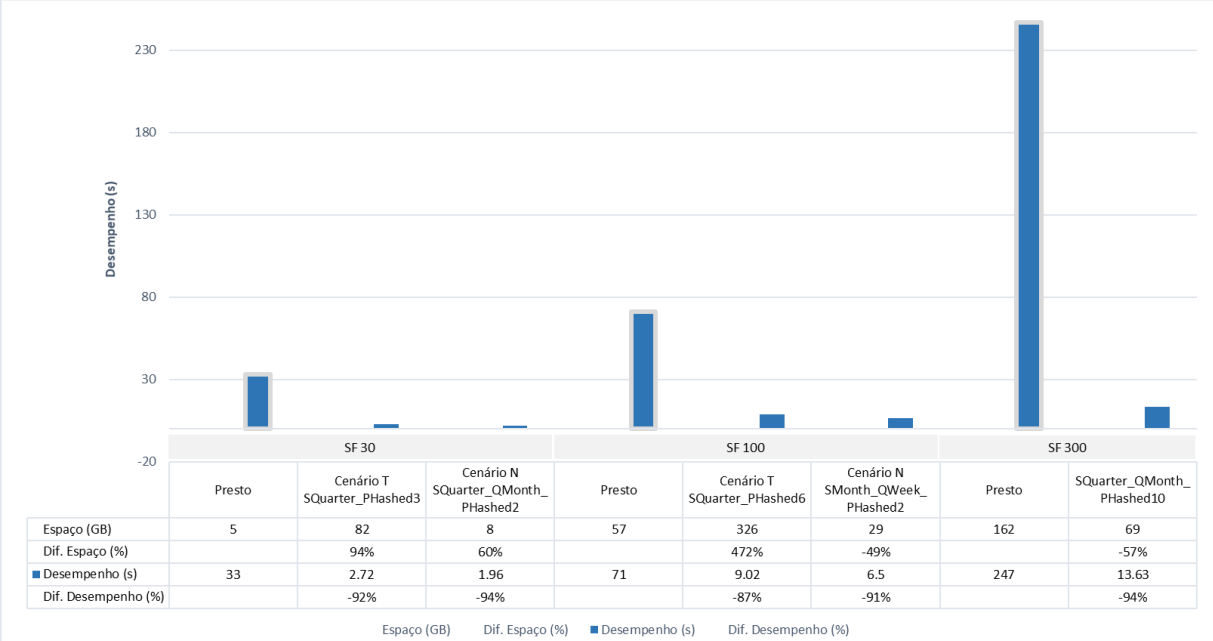


	SF 30			SF 100			SF 300	
	Presto	Cenário T SQuarter_PHashed3	Cenário N SQuarter_QMonth_PHashed2	Presto	Cenário T SQuarter_PHashed6	Cenário N SMonth_QWeek_PHashed2	Presto	SQuarter_QMonth_PHashed10
Espaço (GB)	5	82	8	57	326	29	162	69
Dif. Espaço (%)		94%	60%		472%	-49%		-57%
Desempenho (s)	33	2.09	1.35	71	6.12	3.72	247	7.6
Dif. Desempenho (%)		-94%	-96%		-91%	-95%		-97%

Figura 69 - Comparação entre os Melhores Desempenhos Obtidos pelo Presto e pelo Druid (tempos médios).

A fim de perceber se a *cache* do Druid teve um impacto muito grande nos resultados anteriormente apresentados, na Figura 70 é efetuada uma análise semelhante, desta vez utilizando apenas a média de tempos obtidos na primeira execução das *queries* efetuadas ao Druid. Mais uma vez, verifica-se que o desempenho no Druid é muito superior ao desempenho obtido pelo Presto, mesmo

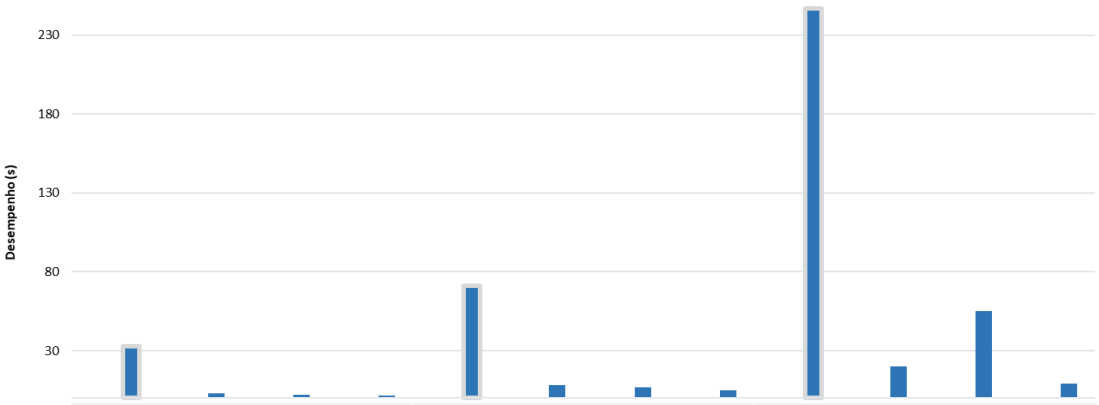
tendo apenas em conta os tempos de processamento obtidos na primeira execução das *queries*, na qual o efeito da *cache* é inexistente (na primeira execução ainda não existem resultados em *cache*). Contudo, efetivamente, existiu uma redução na diferença de desempenho entre as duas tecnologias, situada agora no intervalo 87% - 94%.



	SF 30			SF 100			SF 300	
	Presto	Cenário T SQuarter_PHashed3	Cenário N SQuarter_QMonth_ PHashed2	Presto	Cenário T SQuarter_PHashed6	Cenário N SMonth_QWeek_ PHashed2	Presto	SQuarter_QMonth_ PHashed10
Espaço (GB)	5	82	8	57	326	29	162	69
Dif. Espaço (%)		94%	60%		472%	-49%		-57%
Desempenho (s)	33	2.72	1.96	71	9.02	6.5	247	13.63
Dif. Desempenho (%)		-92%	-94%		-87%	-91%		-94%

Figura 70 - Comparação entre os Melhores Desempenhos Obtidos pelo Presto e pelo Druid (tempos da primeira execução).

Uma vez que se observaram grandes diferenças de desempenho entre as tecnologias aqui abordadas, optou-se por efetuar mais uma análise. Esta análise é apresentada na Figura 71 e efetua uma comparação entre o desempenho obtido pelo Presto e os melhores desempenhos obtidos no Druid, em cada cenário, excluindo-se as tabelas que já foram consideradas anteriormente. Ou seja, utilizam-se as tabelas que obtiveram melhores desempenhos nos cenários TS, NS, NSQ, nos vários SFs, e ainda os desempenhos obtidos na integração entre o Hive e o Druid. Por análise da figura referida, verifica-se que, mesmo assim, o Druid obteve desempenhos bastante superiores aos obtidos pelo Presto, sendo que a diferença nos desempenhos obtidos variou entre 78% e 96% favoráveis ao Druid. Esta análise permite ainda suportar as recomendações que foram expostas nesta secção, visto que mesmo sem grandes preocupações para otimizar as características das tabelas, o Druid foi capaz de obter melhores resultados. Desta forma, a complexidade associada à definição de *hashed partitions* pode ser evitada, tal como foi recomendado, a menos que num caso específico seja necessário otimizar o desempenho ao máximo do seu potencial.



	SF 30				SF 100				SF 300			
	Presto	Cenário T SQuarter	Cenário N SMonth	Cenário N SMonth_ QWeek	Presto	Cenário T SQuarter	Cenário N SMonth	Cenário N SQuarter_ QMonth	Presto	Cenário N SMonth	Cenário N Hive e Druid	Cenário N SQuarter_ QMonth
Espaço (GB)	5	60	14	9	57	201	46	24	162	139	85	60
Dif. Espaço (%)		1100%	180%	80%		253%	-19%	-58%		-14%	-48%	-63%
Desempenho (s)	33	3.21	2.22	1.42	71	8.08	7.02	5	247	20.02	55.03	8.99
Dif. Desempenho (%)		-90%	-93%	-96%		-89%	-90%	-93%		-92%	-78%	-96%

Figura 71 - Desempenhos Obtidos pelo Presto e pelo Druid (tempos médios nos Cenários TS, NS, NSQ e integração Hive e Druid)

Através da comparação de resultados aqui efetuada, verifica-se que o Druid obteve resultados superiores àqueles que foram alcançados pelo Presto e, consequentemente, pelo Hive, visto que nos trabalhos de E. Costa (2017) os resultados obtidos pelo Presto foram sempre superiores aos do Hive. Conclui-se, assim, que o Druid atingiu resultados superiores quando comparado com outras tecnologias SQL-on-Hadoop aqui abordadas (Hive e Presto), perfilando-se, desta forma, como uma possível solução a utilizar no projeto de investigação no qual esta dissertação se enquadra e, de uma forma geral, em contextos de concretização de BDW.

5. Druid para Processamento Analítico de Dados em RTBDW

Analisado o desempenho do Druid como solução de BDW, nesta dissertação é também objetivo perceber como é que o Druid pode ser integrado com outras tecnologias, formando uma arquitetura de RTBDW. Tal como se verificou na análise dos trabalhos relacionados com esta dissertação, no enquadramento concetual, têm sido propostas algumas abordagens que procuram a concretização de um RTBDW, no entanto, em todas elas se identificaram algumas limitações ou foi necessária a utilização de diferentes tecnologias ou várias tabelas com diferentes propósitos. Além disso, esta problemática tem merecido também a atenção do projeto de investigação no qual esta dissertação se enquadra.

O Druid é uma tecnologia que pode solucionar estes problemas, sendo capaz de armazenar dados em *real-time* e dados históricos, gerindo a separação entre os mesmos e efetuando consultas sobre todos os dados, de uma forma transparente ao utilizador. Ou seja, elimina-se a necessidade de utilizar diferentes tecnologias ou várias tabelas com diferentes propósitos para responder à necessidade de implementar um RTBDW que possua bom desempenho.

Assim, ao longo deste capítulo será apresentada uma arquitetura de RTBDW e um caso de aplicação da mesma. É relevante referir que o trabalho efetuado neste capítulo está integrado num projeto mais abrangente de definição de um *Intelligent Data Broker*, estando já em processo de escrita uma publicação científica que descreve os seus princípios. Nesse trabalho, o Druid é utilizado como um repositório para o armazenamento de dados agregados e o cálculo de KPIs (*Key Perfomance Indicator*) em *real-time*, de forma a fornecer informação para uma tomada de decisão atempada e auxiliada pelos KPIs definidos. Por esse motivo, serão aqui apresentados alguns dos KPIs já implementados, sendo que não é objetivo fazer *benchmark* ao desempenho da arquitetura de RTBDW aqui proposta. O objetivo até esta fase era propor uma arquitetura funcional com todos os componentes integrados, sendo que em trabalho futuro se otimizará e avaliará o desempenho da mesma. Refere-se ainda que, nesta fase, será utilizado um modelo desnormalizado com todos os atributos do ALR, pela mesma justificação, ou seja, até esta altura a principal preocupação foi o desenvolvimento de uma solução integrada que será no futuro otimizada.

5.1 Conjunto de Dados

Os dados utilizados como base para o estudo efetuado neste capítulo são provenientes do ALR (*Active Lot Release*), um sistema utilizado pela Bosch e cujos dados estão a ser utilizados no âmbito do projeto de investigação no qual esta dissertação se enquadra. De forma sucinta, o ALR é responsável por auxiliar o controlo de qualidade das peças/produtos nas linhas de produção da fábrica. Este sistema aplica um conjunto de regras aos produtos que constituem um lote, antes deste ser enviado ao cliente. No caso de as regras serem verificadas, o lote pode ser enviado ao cliente. No caso de não se verificarem, o lote terá de ser reparado antes que se possa proceder ao seu envio.

Desta forma, os dados provenientes deste sistema vão sendo gerados em *real-time*, à medida que os lotes vão sendo criados e validados ou invalidados, sendo, portanto, necessário suportar análises sobre os dados à medida que estes chegam, de forma a ser possível tomar decisões sobre os mesmos em tempo útil, ou seja, análises de carácter operacional. Além disso, é necessário ter a capacidade de conservar o histórico com toda a informação para se suportar também outros tipos de análises com maiores intervalos temporais, análises com carácter mais analítico e de descoberta de padrões, suportando decisões num nível mais tático ou estratégico.

Na Figura 72 (desfocada por razões de confidencialidade) esquematiza-se o modelo de dados original do ALR. De referir que os elementos destacados na figura correspondem àqueles que estão no nível mais superior do esquema de dados, englobando outros objetos e/ou *arrays*. Tal como é possível perceber pela análise da figura, estes dados chegam em estrutura JSON, com objetos e *arrays* de objetos. Além disso, o atributo "chave" possui registos identificados como relevantes para associar as regras aos produtos correspondentes, pelo que, estes registos terão de ser transformados em atributos durante o processo de desnormalização (identificados na figura com ligações diferentes).

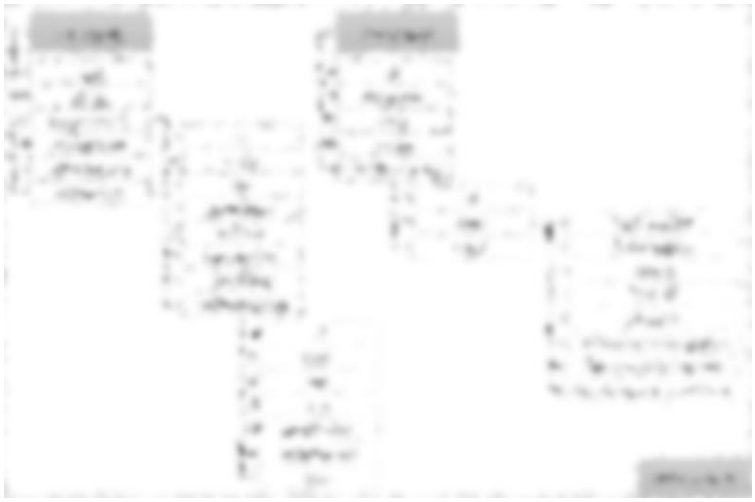


Figura 72 - Modelo de Dados Original do ALR.

Tendo em conta o modelo anterior e as limitações do Druid, é imperativo efetuar um processo de desnormalização (que será efetuado no componente de tratamento dos dados, na arquitetura apresentada na secção seguinte), com o fim de se obter o modelo apresentado na Figura 73 (desfocada por razões de confidencialidade).

Figura 73 - Modelo de Dados Desnormalizado do ALR.

Por fim, importa referir que se tratam de dados confidenciais e, como tal, foi necessário aplicar mecanismos para garantir a privacidade dos mesmos, que serão explicados adiante neste capítulo.

5.2 Arquitetura da Solução

A Figura 74 apresenta a arquitetura da solução de RTBDW implementada, com todas as tecnologias utilizadas especificadas. De referir que em todos os componentes desta arquitetura se podem considerar outras tecnologias, sendo que a escolha por aquelas que se observam na figura teve em conta as que já são utilizadas no âmbito do projeto de investigação no qual esta dissertação se enquadra (como

é o caso do kafka, Spark, Spark Streaming, Hive e Hadoop) e as tecnologias que se integram adequadamente com o Druid e que foram estudadas no enquadramento tecnológico (como é o caso do Superset).

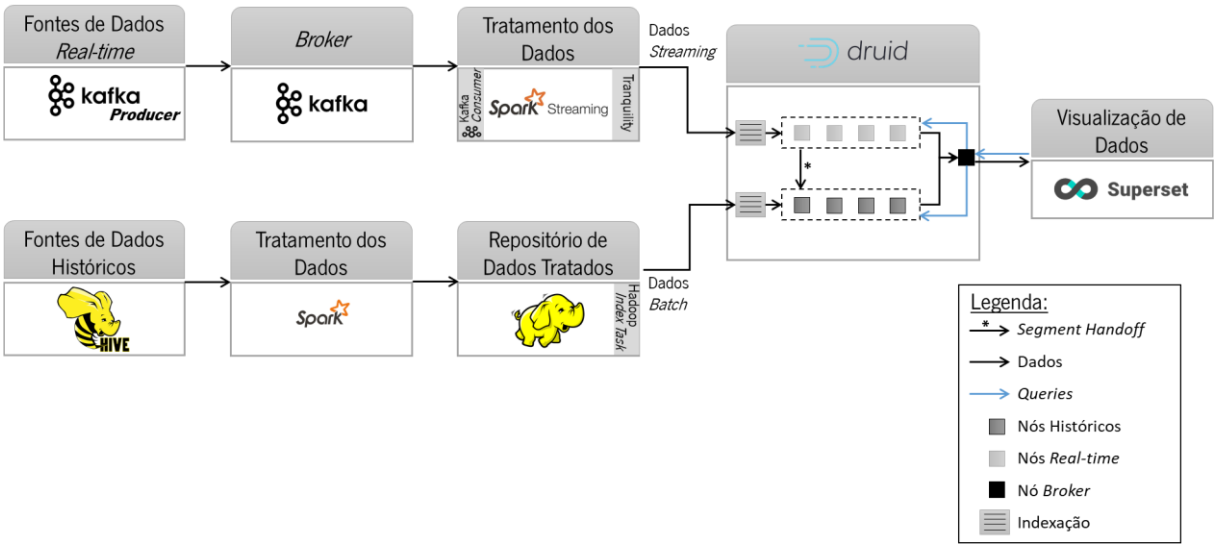


Figura 74 - Arquitetura da Solução de RTBDW Implementada.

No que diz respeito às fontes de dados em *real-time*, considera-se um *producer* implementado em kafka, de forma a simular um contexto real de geração de eventos. Este *producer* irá ingerir aleatoriamente e com uma cadência parametrizável dados de uma amostra recolhida do ALR. Os dados gerados passam por um *Broker,* também implementado em kafka, que disponibiliza os dados para serem consumidos para os propósitos que forem necessários.

Na arquitetura apresentada é possível verificar que o *broker* encaminha os dados para alimentar a fluxo *streaming*, no entanto, podem existir situações em que os mesmos são guardados noutras tabelas ou ficheiros que mais tarde podem alimentar fluxos *batch*, constituindo-se como uma fonte de dados históricos. Continuando na explicação do fluxo *streaming*, os eventos que vão sendo gerados pelo *producer* e encaminhados pelo *broker* passam depois por um processo de tratamento e enriquecimento dos dados, implementado em Spark Streaming. De referir que este componente de tratamento dos dados recebe os dados através de um *consumer* kafka e, depois de efetuar as transformações necessárias, ingere os dados finais no Druid, recorrendo ao Tranquility.

No fluxo *batch* foi utilizado algum histórico de dados do ALR que estava armazenado em tabelas Hive, sendo que estes dados também tiveram de passar por um processo de tratamento e

enriquecimento, implementado no Spark. O componente do Spark, no final, armazena os dados no HDFS, prontos para ser ingeridos no Druid, via Hadoop *Index Task*, assim que necessário.

O componente do Druid recebe os dados em *streaming* e em *batch,* que passam por um processo de indexação, sendo depois gerida, de forma transparente aos utilizadores, a separação entre os dados ingeridos pelas diferentes vias. Os dados que advêm de processos *batch* são imediatamente armazenados nos *historical nodes* do Druid e ficam disponíveis para consulta, enquanto que os dados ingeridos via *streaming* passam pelos *real-time nodes*, ficando imediatamente disponíveis para consulta e, periodicamente dá-se o processo de *segment handoff* (explicado na secção 0), no qual os dados são transferidos dos *real-time nodes* para os *historical nodes,* sendo consultáveis durante todo o processo. O *broker* do Druid é responsável por receber as *queries* e encaminhá-las para os *nodes* que possuam segmentos que auxiliem na resposta a estas *queries*, quer sejam *historical nodes* ou *real-time nodes*. No final, o *broker* do Druid junta os resultados parciais devolvidos pelos vários *nodes* e envia a resposta da *query*.

Por fim, como componente de visualização de dados foi utilizado o Superset, principalmente por ser possível a sua integração com os repositórios do Druid e por estar incluído na distribuição Hortonworks. Realça-se que em todos os componentes poderão ser adicionadas outras tecnologias. Já no decorrer desta dissertação se demonstrou a integração entre o Druid e o Hive e, fruto desta, podia adicionar-se ao componente de visualização de dados, por exemplo, o Tableau.

5.3 Implementação da Solução

Fluxo *Streaming*

No componente de fontes de dados em *real-time*, tal como já foi referido, optou-se por implementar um *producer* em kafka que irá ingerir, de forma aleatória e com uma cadência parametrizável, dados de uma amostra do ALR. Ao ingerir estes dados de forma aleatória, garante-se que os requisitos de privacidade dos mesmos são cumpridos, visto que não será possível retirar conclusões sobre os mesmos, nem inferir o estado do negócio da organização que cedeu os dados. Além disso, desta forma existe um maior controlo sobre a cadência a que os dados são ingeridos, o que permite maior flexibilidade na fase de testes à solução até que esta fique otimizada.

Estes dados são enviados para um tópico no *Broker* Kafka e que permite que sejam consumidos por todos os seus subscritores ou consumidores. Neste caso há a considerar o consumidor do fluxo de *streaming* (um kafka *consumer*), que irá permitir que o Spark Streaming (no componente de tratamento

de dados) receba os dados à medida que vão sendo gerados e efetue os procedimentos necessários para garantir a transformação e enriquecimento dos mesmos até estarem em condições de serem armazenados no Druid.

Os dados consumidos, tal como mencionado anteriormente, possuem requisitos de privacidade, razão essa que motivou a ingestão dos mesmos aleatoriamente. Contudo, estes dados são provenientes de uma amostra, pelo que à medida que se fossem inserindo iam referir sempre as datas provenientes da amostra, o que podia gerar duplicados. Desta forma, efetuou-se um tratamento aos dados para que todas as datas sejam alteradas, face à data de ingestão do registo no componente de tratamento de dados. Ou seja, se o evento que acaba de chegar a este componente disser respeito a um lote validado, por exemplo, esta data será alterada para corresponder à data de entrada do registo no componente, e as restantes datas serão calculadas tendo em conta esta data e as diferenças entre as datas originais, provenientes da amostra.

Ao aplicar esta abordagem garante-se também um mecanismo de proteção de dados adicional, visto que para além dos dados serem ingeridos aleatoriamente, as suas datas foram alteradas, pelo que estão desfasados dos resultados reais do negócio. Seguidamente, aplica-se também a substituição dos valores nulos por *Strings* representativas. Ambos os tratamentos de dados estão ilustrados na Figura 75, sendo que o primeiro, corresponde ao método “treatDates”, enquanto que o segundo é efetuado recorrendo ao método “treatNulls”, ambos implementados na classe “ALRUtils”.

```
public class ALRUtils {

    public static void treatDates(ALR alr) throws ParseException {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssZ");
    sdf.setTimeZone(TimeZone.getTimeZone("Europe/Lisbon"));
    Lot lot = alr.getLot();
    long now = System.currentTimeMillis();
    long diffCreationAndFinishDate = Math.abs(sdf.parse(lot.getCreationDate()).getTime() - sdf.parse(lot.getPallet().getFinishDate()).getTime());
    String finishDate = lot.getPallet().getFinishDate();
    if (lot.getState().equals("CREATED")) {
        lot.setCreationDate(sdf.format(new Timestamp(now)));
        lot.getPallet().setFinishDate(sdf.format(new Timestamp(now - diffCreationAndFinishDate)));
    } else {
        long diffValidAndCreation = Math.abs(sdf.parse(lot.getValidationDate()).getTime() - sdf.parse(lot.getCreationDate()).getTime());
        lot.setValidationDate(sdf.format(new Timestamp(now)));
        lot.setCreationDate(sdf.format(new Timestamp(now - diffValidAndCreation)));
        lot.getPallet().setFinishDate(sdf.format(new Timestamp(now - (diffCreationAndFinishDate + diffValidAndCreation))));
    }
    for (Product product : lot.getPallet().getProducts()) {
        long diffFinishAndProduct = Math.abs(sdf.parse(finishDate).getTime() - sdf.parse(product.getDate()).getTime());
        product.setDate(sdf.format(new Timestamp(sdf.parse(lot.getPallet().getFinishDate()).getTime() - diffFinishAndProduct)));
    }
    if ("CREATED".equals(lot.getState())) {
        lot.setTimestamp(lot.getCreationDate());
    } else {
        lot.setTimestamp(lot.getValidationDate());
    }
}

public static void treatNulls(ALR alr) {
    if (alr.getObservation() == null) {
        alr.setObservation("No observations");
    }
    if (alr.getLot().getObservation() == null) {
        alr.getLot().setObservation("No observations");
    }
    if (alr.getLot().getValidationDate() == null) {
        alr.getLot().setValidationDate("Not applicable");
    }
}
```

Figura 75 - Tratamento efetuado às Datas e aos Valores Nulos.

Posteriormente é necessário efetuar o processo de desnormalização dos dados que, originalmente e como já foi possível verificar, vêm numa estrutura em JSON que é complexa, possuindo *nested objects* (objetos dentro de objetos) e outras particularidades, como por exemplo, o atributo "chave", no *array* "ruleDetails", que se tem de desmultiplicar em vários atributos.

Finalmente, assim que os dados estiverem preparados para serem armazenados, estes são ingeridos no Druid, em *real-time*, recorrendo ao Tranquility. Esta parte do processo é demonstrada na Figura 76. Desta figura, importa destacar que se começa por criar uma "InputStream" a referenciar um ficheiro com as configurações da tarefa de ingestão ("iDBRealTimeRawData"). De referir que esta tarefa de ingestão é semelhante às tarefas utilizadas para ingerir dados em *batch* e pode ser consultada no Apêndice 8. Desta figura é ainda relevante destacar a definição da tabela onde os dados serão armazenados (no Druid as tabelas são denominadas por *DataSources*), sendo neste caso a tabela "ALRRawData". Seguidamente, faz-se o *build* do serviço com todas as configurações, inicia-se o mesmo e enviam-se os dados num formato percebido pelo Tranquility, com o atributo e o valor correspondente. A partir deste momento os dados são imediatamente consultáveis, sendo que a criação das tarefas de ingestão e as *batches* de eventos (para não ser enviado apenas um evento por conexão) são geridas pelo Tranquility, mediante as configurações definidas.

```
joined.javaRDD().foreachPartitionAsync(partitionOfRecords -> {
    InputStream configStream = ALRRawDataConsumer.class.getClassLoader().getResourceAsStream("iDBRealTimeRawData.json");
    TranquilityConfig<PropertiesBasedConfig> config = TranquilityConfig.read(configStream);
    DataSourceConfig<PropertiesBasedConfig> pageViewConf = config.getDataSource("ALRRawData");
    Tranquilizer<Map<String, Object>> druidService = DruidBeams.fromConfig(pageViewConf)
            .buildTranquilizer(pageViewConf.tranquilizerBuilder());
    druidService.start();

    while (partitionOfRecords.hasNext()) {
        Row alr = partitionOfRecords.next();
        final Map<String, Object> obj = ImmutableMap.<String, Object>builder()
                .put("timestamp", alr.get(19))
                .put("lotObservation", alr.get(0))
                .put("lotState", alr.get(1))
                .put("lotUuid", alr.get(2))
                .put("lotValidationDate", alr.get(3))
                .put("lotCreationDate", alr.get(4))
                .put("palletId", alr.get(5))
                .put("palletFinishDate", alr.get(6))
                .put("palletHuCode", alr.get(7))
                .put("palletLine", alr.get(8))
                .put("palletOrigin", alr.get(9))
                .put("palletCode", alr.get(10))
                .put("palletPartNumber", alr.get(11))
                .put("productId", alr.get(12))
                .put("productBox", alr.get(13))
                .put("productDate", alr.get(14))
                .put("productLine", alr.get(15))
                .put("productOrigin", alr.get(16))
                .put("productPartNumber", alr.get(17))
                .put("productSerialNumber", alr.get(18))
                .put("rulesId", alr.get(20) == null ? "Not applicable" : alr.get(20))
                .put("rulesDesignation", alr.get(21) == null ? "Not applicable" : alr.get(21))
                .put("rulesValid", alr.get(22) == null ? "Not applicable" : alr.get(22))
                .put("rulesReason", alr.get(23) == null ? "Not applicable" : alr.get(23))
                .put("classId", alr.get(24) == null ? "Not applicable" : alr.get(24))
                .put("deffectCauseDescription", alr.get(25) == null ? "Not applicable" : alr.get(25))
                .put("flawId", alr.get(26) == null ? "Not applicable" : alr.get(26))
                .put("maxNumberOfOccurrences", alr.get(27) == null ? 0 : alr.get(27))
                .put("numberOfOccurrences", alr.get(28) == null ? 0 : alr.get(28))
                .put("partNumber", alr.get(29) == null ? "Not applicable" : alr.get(29))
                .put("serialNumber", alr.get(30) == null ? "Not applicable" : alr.get(30))
                .put("station", alr.get(31) == null ? "Not applicable" : alr.get(31))
                .build();

        druidService.send(obj);
    }
    druidService.flush();
    druidService.stop();
});
```

Figura 76 - Ingestão de Dados no Druid, recorrendo ao Tranquility.

Fluxo *Batch*

Este fluxo inicia-se com o componente de fontes de dados históricos que, nesta solução, utilizou o Hive, visto que no projeto de investigação que integra esta dissertação, este repositório estava a ser utilizado e já continha algum histórico dos dados provenientes do ALR. Contudo, tal como nos outros componentes, neste também se podem adicionar outras tecnologias, como por exemplo o HDFS. É relevante referir que os dados que chegam em *real-time* também podem constituir uma fonte de dados históricos se isso se justificar em alguns cenários, sendo que esta interação não está representada na arquitetura, uma vez que não foi utilizada nesta solução. Pode considerar-se, por exemplo, para além de ingerir os dados em *real-time* através do fluxo de *streaming*, guardar também os eventos originais noutro repositório (HDFS ou Hive, por exemplo), para poderem funcionar, mais tarde, como uma fonte de dados históricos para outros propósitos. Um destes propósitos pode ser validar os dados que foram inseridos em *real-time* ou até substituí-los, caso se detete algum problema.

Seguidamente, os dados passam pelo processo de tratamento e enriquecimento (semelhante ao descrito no fluxo *streaming*), no componente implementado com o Spark, sendo que ultrapassado este

processo, os dados resultantes são armazenados num repositório com os dados tratados, implementado com o HDFS.

Finalmente, assim que pretendido, é executada uma tarefa de ingestão de dados no Druid em *batch* ("Hadoop *Index Task*"), tal como aquela que está disponível no Apêndice 8.

Visualização de Dados

Nesta componente, tal como já foi referido, foi utilizado o Superset que disponibiliza uma integração com os repositórios do Druid, não tendo sido necessário trabalho adicional nesse sentido. Além disso, esta ferramenta permite que os *Dashboards* sejam automaticamente atualizados a cada 10 segundos ou que a sua atualização seja forçada em qualquer altura, o que é relevante para analisar os dados que chegam em *real-time*. Na Figura 77 é apresentado um exemplo de análise efetuada aos dados do ALR, recorrendo ao Superset.

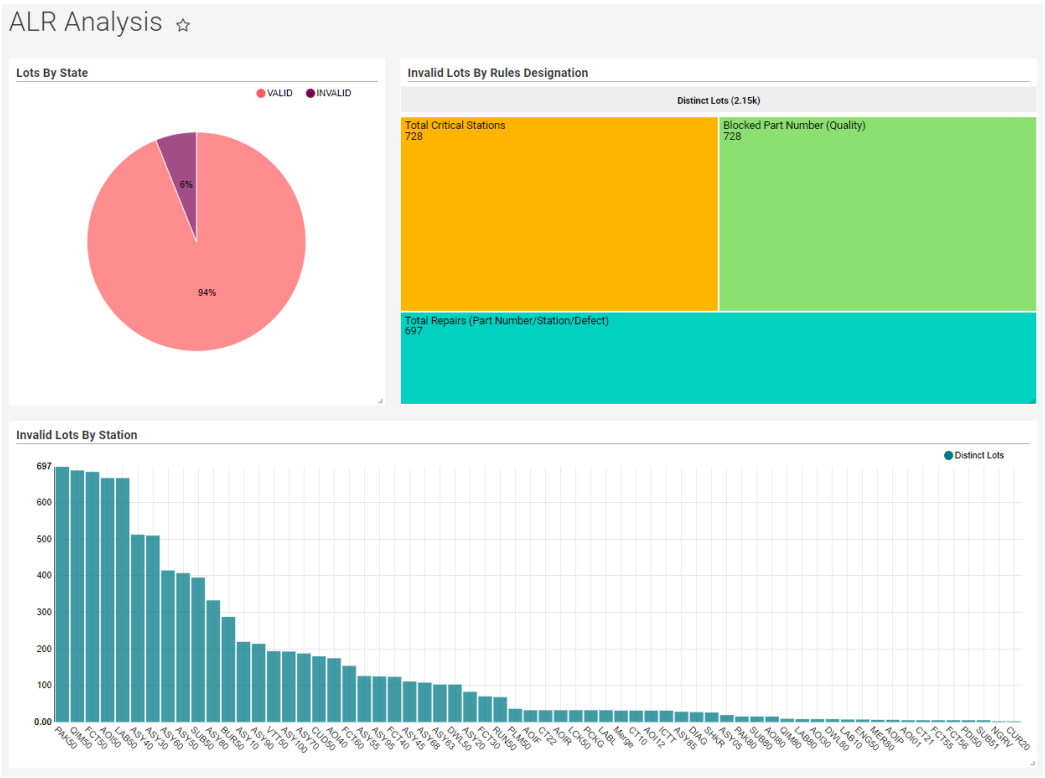


Figura 77 - Exemplo de Análise aos Dados do ALR no Superset.

KPIs

Tal como supramencionado, o trabalho apresentado neste capítulo está integrado num projeto desenvolvido em paralelo e que utiliza o Druid para o armazenamento de dados agregados em *real-time* e para o cálculo de KPIs sobre os dados que vão sendo recebidos (e sobre os dados que vão sendo mantidos em histórico).

Os KPIs utilizados são os seguintes:

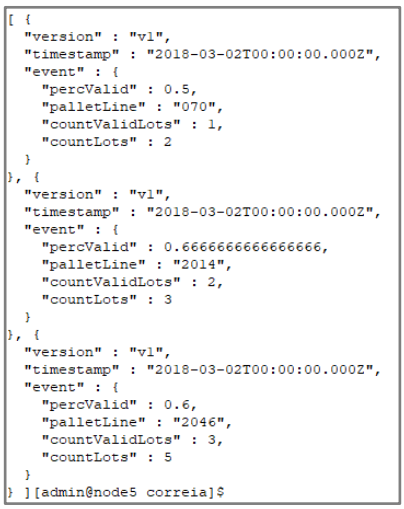
- Número de lotes inválidos, por dia, menor que 5%;
- Percentagem de lotes válidos, por dia, por linha da palete, maior que 90%;
- Número de produtos produzidos, por dia, por linha, maior que a média na última semana.

Seguidamente, na Figura 78, apresenta-se um exemplo da implementação dos KPIs. O exemplo presente na figura é o cálculo do segundo KPI, sendo que foi implementado de forma a que a *query* retorne informação sobre as situações anómalas, ou seja, que estão em incumprimento do KPI, de tal forma que o utilizador possa identificar de imediato esses casos e tomar providências. A implementação do primeiro e do terceiro KPI pode ser consultada nos Apêndices 10 e 11, respetivamente. Importa ainda mencionar que os KPIs foram implementados recorrendo à linguagem nativa do Druid, visto que a linguagem SQL suportada pelo Druid apresentava limitações que impossibilitaram a implementação dos mesmos recorrendo a SQL.

```
{
  "queryType": "groupBy",
  "dataSource": "ALRRawData",
  "descending": "false",
  "dimensions":["palletLine"],
  "metrics":["countValidLots, countLots"],
  "aggregations": [
      {
          "type" : "filtered",
          "filter" : {
              "type" : "selector",
              "dimension" : "lotState",
              "value" : "VALID"
                  },
          "aggregator" : {
              "type" : "cardinality",
              "name" : "countValidLots",
              "fields" : ["lotUuid"],
              "round": true
                  }
          },
          {
              "type" : "cardinality",
              "name" : "countLots",
              "fields" : ["lotUuid"],
              "round": true
          }
   ],
  "postAggregations" : [
      {
          "type"   : "arithmetic",
          "name"   : "percValid",
          "fn"     : "/",
          "fields": [
              {"type" : "hyperUniqueCardinality", "fieldName" : "countValidLots"},
              {"type" : "hyperUniqueCardinality", "fieldName" : "countLots"}
          ]
      }
   ],
  "granularity": "all",
  "intervals": [
    "2018-03-02/2018-03-03"
  ],
  "having": {
   "type": "lessThan",
   "aggregation": "percValid",
   "value": 0.90
  }
}
```

Figura 78 - Cálculo do segundo KPI.

Se executarmos a *query* anterior esta irá retornar o resultado apresentado na Figura 79, demonstrando ao utilizador que as linhas “070”, “2014” e “2076” atravessam situações anómalas.



```
[ {
  "version" : "v1",
  "timestamp" : "2018-03-02T00:00:00.000Z",
  "event" : {
    "percValid" : 0.5,
    "palletLine" : "070",
    "countValidLots" : 1,
    "countLots" : 2
  }
}, {
  "version" : "v1",
  "timestamp" : "2018-03-02T00:00:00.000Z",
  "event" : {
    "percValid" : 0.6666666666666666,
    "palletLine" : "2014",
    "countValidLots" : 2,
    "countLots" : 3
  }
}, {
  "version" : "v1",
  "timestamp" : "2018-03-02T00:00:00.000Z",
  "event" : {
    "percValid" : 0.6,
    "palletLine" : "2046",
    "countValidLots" : 3,
    "countLots" : 5
  }
} ][admin@node5 correia]$
```


Figura 79 - Resultado da query do segundo KPI.

Com este exemplo foi possível dar a conhecer as potencialidades do Druid em contextos de RTBDW. Apesar de ser um trabalho que ainda está em curso, o mesmo revela-se importante para se compreender na globalidade as potencialidades do Druid e a sua capacidade para processamento analítico de dados em contextos de *Big Data*.

6. Conclusões

Este documento, tal como era objetivo começa por expor o enquadramento concetual sobre os principais conceitos associados à temática desta dissertação, nomeadamente: o *Big Data*, os *Data Warehouses*, os *Big Data Warehouses* e o *Real-Time* em *Big Data Warehouses*. Além disso, o enquadramento concetual aborda o estudo dos trabalhos que endereçaram desafios semelhantes aos desta dissertação e termina com um mapa concetual que sistematiza os principais conceitos abordados, assim como as relações entre eles.

Seguidamente, no enquadramento tecnológico foi explorado o ecossistema Hadoop, assim como algumas das tecnologias que podem integrar com o Druid numa arquitetura de RTBDW. Além disso, é apresentado o protocolo de testes aplicado na parte experimental da dissertação, assim como a infraestrutura tecnológica que o suportou. Contudo, no capítulo em questão foi concedida atenção especial sobre o Druid, visto que esta é a tecnologia em foco nesta dissertação. Assim, contextualizou-se a tecnologia, apresentou-se a sua arquitetura e principais características diferenciadoras de outras tecnologias, abordou-se o seu formato de armazenamento e motor de consultas mapeado em memória, assim como outras questões, como o particionamento dos segmentos, a definição de *query granularity*, os índices invertidos que o Druid aplica, a linguagem nativa do Druid e a capacidade que esta tecnologia tem para implementar extensões propostas pela comunidade. Neste capítulo encerrou-se o objetivo de construir uma base firme de conhecimento alicerçada numa revisão de literatura extensa, recente e relevante, conduzindo a um bom enquadramento concetual e tecnológico, importante para ser bem-sucedido ao longo do restante trabalho da dissertação.

Terminado o enquadramento concetual e tecnológico, foi apresentada a fase experimental desta dissertação, que se dividiu em dois tópicos principais: Druid para Processamento Analítico de Dados em BDW e Druid para Processamento Analítico de Dados em RTBDW.

No primeiro tópico começou-se por efetuar um estudo da cardinalidade e distribuição dos dados, relevante para se analisar se as características dos mesmos têm algum impacto no processamento das *queries* que os envolvam. De seguida, foram apresentados os vários cenários de teste e preparada a sua execução. Estes testes utilizaram o SSB para avaliar o desempenho do Druid e da integração do Hive com o Druid em diferentes configurações de *segment granularity*, *query granularity*, *hashed partitions* e de modelo de dados, sendo que os desempenhos obtidos foram ainda comparados com os resultados de outras tecnologias SQL-on-Hadoop em circunstâncias idênticas. Em adição, foi ainda estudado o

potencial da integração do Hive com o Druid, foram efetuados testes em ambientes multiutilizador e estudados os resultados obtidos face ao tipo de *queries*.

Os resultados obtidos no primeiro tópico da fase experimental da dissertação permitiram estudar a influência produzida no desempenho do Druid pela aplicação das propriedades que foram testadas (*segment granularity, query granularity, hashed partitions* e atributos a considerar no modelo de dados). Através da análise dos resultados foi possível verificar que a definição de segmentos de dados com tamanhos uniformes, no que diz respeito ao número de linhas é capaz de otimizar o desempenho. Para tal, devem começar por ser escolhidas *segment granularities* que particionem os dados adequadamente e, preferencialmente, deve utilizar-se um modelo de dados apenas com os atributos necessários, de tal forma que a ingestão de dados seja mais rápida, o espaço de armazenamento necessário seja menor e o impacto da aplicação da *query granularity* possa ser maior. Demonstrou-se ainda que, em todos os cenários, a aplicação de *hashed partitions* foi capaz de otimizar o desempenho. O estudo do potencial da integração entre o Hive e o Druid revelou que esta é benéfica para ambas as tecnologias, uma vez que o Druid beneficia, por exemplo, de uma interface SQL para gerir os seus repositórios e de conectores ODBC/JDBC que podem permitir integração com várias ferramentas de visualização e, o Hive beneficia da forma otimizada como os repositórios do Druid armazenam informação. Por fim, a comparação dos resultados obtidos no capítulo 4, com os resultados obtidos por outras tecnologias SQL-on-Hadoop em circunstâncias semelhantes, demonstrou que o Druid obteve melhores desempenhos que todas elas.

Por fim, no segundo tópico da fase experimental desta dissertação, apresentado no capítulo 5, foi proposta uma arquitetura funcional de RTBDW. Neste capítulo, começou-se por analisar o conjunto de dados, de forma a ser percetível os tratamentos de dados a efetuar e a necessidade de uma arquitetura de RTBDW. Demonstrou-se ainda a implementação da arquitetura proposta, tendo-se apresentado algumas análises aos dados que foram integrados no Druid, em *real-time*, assim como os KPIs que foram desenvolvidos. Este capítulo permitiu demonstrar que o Druid pode fazer parte de uma arquitetura de RTBDW, integrando com várias tecnologias, sendo que como trabalho futuro será necessário avaliar o desempenho desta arquitetura.

6.1 Resultados Obtidos

Esta dissertação tinha como finalidade estudar detalhadamente o Druid, por se tratar de uma tecnologia recente e que pode vir a assumir um papel relevante na implementação de sistemas de BDW e RTBDW, dadas as suas características. Assim, era importante perceber os contextos em que a sua

utilização pode ser recomendada, assim como avaliar o seu desempenho, através de um conjunto alargado de testes. Assim, os objetivos que esta dissertação se propunha atingir, foram todos concretizados, tendo sido:

- Sistematizado o estado da arte no que diz respeito a abordagens de implementação de BDW e RTBDW;
- Contextualizada a tecnologia e os seus principais cenários de utilização;
- Avaliado o desempenho do Druid no processamento analítico de dados num contexto de BDW, utilizando um *benchmark* de referência;
- Estudada e testada a integração entre o Hive e o Druid, em comparação com outras tecnologias SQL-on-Hadoop, utilizando um *benchmark* de referência;
- Estudado o impacto que ambientes multiutilizador podem causar no desempenho do Druid e da integração entre o Hive e o Druid, utilizando um *bechmark* de referência;
- Identificadas boas práticas para a otimização do desempenho do Druid no processamento analítico de dados, em contextos de BDW;
- Implementada e validada uma arquitetura de RTBDW em torno do Druid, recorrendo a um caso de demonstração;

No que diz respeito a contributos científicos, é importante referir que o trabalho efetuado na vertente experimental, no capítulo 4, desta dissertação já deu origem a uma publicação científica que está prestes a ser submetida. Além disso, o trabalho apresentado no capítulo 5, também deu origem a um trabalho paralelo que brevemente resultará numa publicação científica. No que diz respeito ao projeto de investigação no qual esta dissertação se enquadra, também foi dado um contributo importante, visto que se comprovou que o Druid pode atingir melhores resultados que outras tecnologias SQL-on-Hadoop atualmente em utilização no projeto.

6.2 Dificuldades e Limitações

A juventude que caracteriza esta área de investigação, assim como a velocidade com que esta vai evoluindo gerou dificuldades para alcançar os objetivos a que esta dissertação se propunha.

A documentação, por vezes, incoerente, desorganizada ou incompleta providenciada pelo Druid, assim como a existência de poucos estudos na comunidade científica e técnica que abordem esta tecnologia constituiu uma das maiores dificuldades, principalmente na fase inicial. Esta tecnologia tem

bastantes conceitos associados, pelo que se estes não forem corretamente documentados torna-se muito complexo perceber o funcionamento da mesma. Uma das incoerências encontradas na documentação foi evidenciada nesta dissertação, quando o Druid referia, em dois locais distintos da documentação, dois intervalos diferentes recomendáveis para particionar os segmentos de dados. Um acontecimento que evidencia a velocidade e as dificuldades associadas a esta área é o facto de, quando esta dissertação se iniciou o Druid não suportar SQL, funcionalidade mais tarde adicionada e que motivou que tivesse sido feita uma atualização da infraestrutura tecnológica utilizada para considerar a nova versão.

No que diz respeito à integração do Druid com outras tecnologias, para se formar uma arquitetura de RTBDW, esta também se revelou bastante custosa, uma vez que, num curto espaço de tempo foi necessário trabalhar com muitas tecnologias distintas, nomeadamente: kafka, Spark, HDFS, Hive, Druid e Superset, para além do Tableau e Hive LLAP que foram utilizadas nos testes ao desempenho do Druid em contextos de BDW. Estas dificuldades não se prenderam apenas com ter de aprender a trabalhar com todas estas tecnologias, mas principalmente em conseguir integrá-las todas numa arquitetura com o Druid, visto que a documentação neste aspeto é particularmente pouca.

Tendo em conta que esta dissertação produziu um grande número de resultados, fruto da execução dos diferentes cenários de teste, a organização destes revelou-se desafiante.

Por último, a criação de várias tabelas com grandes volumes de dados para que se pudessem efetuar os vários testes, implicou grandes tempos que necessitaram de ser geridos corretamente, sobre pena de comprometer o desenvolvimento do trabalho.

No entanto, fazendo alusão à frase citada no início desta dissertação, apesar de terem surgido dificuldades e limitações que pareciam impossíveis de superar, no final todos os objetivos foram cumpridos.

6.3 Trabalho Futuro

Apesar de se terem atingido todos os objetivos, existem vários aspetos que podem ser estudados no futuro.

No que diz respeito ao processamento do Druid em contextos de BDW, um possível trabalho futuro, seria testar o desempenho da integração entre o Hive e o Druid noutras condições, nomeadamente num modelo em estrela. O Hive, tal como foi referido nesta dissertação, providencia ao Druid um conjunto de funcionalidades, de entre as quais a possibilidade de fazer análises mais complexas sobre os seus dados, como por exemplo *joins*, pelo que seria relevante estudar o desempenho nesses contextos.

Para além disso, seria relevante aplicar as recomendações apresentadas nesta dissertação, noutros cenários, de forma a validar se estas se mantêm boas práticas em diferentes contextos. No que diz respeito ao método de particionamento, foi aplicado o tipo *hashed*, contudo, também seria relevante verificar a influência que o particionamento baseado numa única dimensão pode ter no desempenho e se pode ser preferível em alguns cenários.

Por último, e tal como foi anteriormente mencionado, é relevante que, no futuro, sejam efetuados testes para avaliar o desempenho da arquitetura de RTBDW proposta nesta dissertação e, possivelmente, comparar esta arquitetura com alternativas existentes.

REFERÊNCIAS BIBLIOGRÁFICAS

Amaral, L. (2005). Da Gestão ao Gestor de Sistemas de Informação: Expectativas Fundamentais no Desempenho da Profissão. In A. S. e C. Z. L. Amaral, R. Magalhães, C. C. Morais (Ed.), *Sistemas de Informação Organizacionais* (pp. 49–71). Lisboa: Edições Sílabo.

Apache Hadoop. (2018). Welcome to Apache™ Hadoop®! Retrieved May 1, 2018, from http://hadoop.apache.org/

Apache Spark. (2018a). Apache Spark™ - Unified Analytics Engine for Big Data. Retrieved July 5, 2018, from https://spark.apache.org/

Apache Spark. (2018b). Spark Streaming | Apache Spark. Retrieved July 5, 2018, from https://spark.apache.org/streaming/

Apache Superset. (2018). Apache Superset (incubating) — Apache Superset documentation. Retrieved July 2, 2018, from https://superset.incubator.apache.org/

Azvine, B., Cui, Z., & Nauck, D. D. (2005). Towards real-time business intelligence. *BT Technology Journal*, *23*(3), 214–225. https://doi.org/10.1007/s10550-005-0043-0

Baru, C., Bhandarkar, M., Nambiar, R., Poess, M., & Rabl, T. (2013). Benchmarking Big Data Systems and the BigData Top100 List. *Big Data*, *1*(1), 60–64. https://doi.org/10.1089/big.2013.1509

Brewer, E. (2012). CAP Twelve Years Later: How the "Rules" Have Changed. *COMPUTER*, *45*(2), 23–29. https://doi.org/10.1109/MC.2012.37

Cattell, R. (2010). Scalable SQL and NoSQL data stores. *SIGMOD Record*, *39*(4), 12–27. https://doi.org/10.1145/1978915.1978919

Cha, S., & Wachowicz, M. (2015). Developing a Real-Time Data Analytics Framework Using Hadoop. In B. C. Khan L. (Ed.), *Proceedings - 2015 IEEE International Congress on Big Data, BigData Congress 2015* (pp. 657–660). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/BigDataCongress.2015.102

Chambi, S., Lemire, D., Godin, R., Boukhalfa, K., Allen, C. R., & Yang, F. (2016). Optimizing druid with roaring bitmaps. In D. E. Desai B.C. (Ed.), *ACM International Conference Proceeding Series* (Vol. 11–13–July, pp. 77–86). Association for Computing Machinery. https://doi.org/10.1145/2938503:2938515

Chandarana, P., & Vijayalakshmi, M. (2014). Big data analytics frameworks. In *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications, CSCITA 2014* (pp. 430–434). Mumbai, Maharashtra: IEEE Computer Society.

https://doi.org/10.1109/CSCITA.2014.6839299

Chaudhuri, S., & Dayal, U. (1997). An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, *26*(1), 65–74. https://doi.org/10.1145/248603.248616

Chen, C. L. P., & Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Inf. Sci.*, *275*, 314–347.

Chen, G. J., Wiener, J. L., Iyer, S., Jaiswal, A., Simha, R. L. N., Wang, W., … Yilmaz, S. (2016). Realtime data processing at Facebook. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Vol. 26-June-20, pp. 1087–1098). Association for Computing Machinery. https://doi.org/10.1145/2882903.2904441

Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. *MIS Quarterly: Management Information Systems*, *36*(4), 1165–1188.

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications*, *19*(2), 171–209. https://doi.org/10.1007/s11036-013-0489-0

Clegg, D. (2015). Evolving data warehouse and BI architectures: the Big Data challenge. *Bus. Intell. J*, *20*(1), 19–24.

Costa, C., & Santos, M. Y. (2017). Big Data: State-of-the-art concepts, techniques, technologies, modeling approaches and research challenges. *IAENG International Journal of Computer Science*, *44*(3), 285–301. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85028073996&partnerID=40&md5=51d13b35c2291ef10a6a8174756335c0

Costa, E. (2017). *Organização e Processamento de Dados em Big Data Warehouses baseados em Hive*. Retrieved from https://repositorium.sdum.uminho.pt/handle/1822/53753

Costa, E., Costa, C., & Santos, M. Y. (2017). Efficient big data modelling and organization for hadoop hive-based data warehouses. *Lecture Notes in Business Information Processing*, *299*, 3–16. https://doi.org/10.1007/978-3-319-65930-5_1

Cuzzocrea, A. (2015). Data warehousing and OLAP over Big Data: A survey of the state-of-the-art, open problems and future challenges. *International Journal of Business Process Integration and Management*, *7*(4), 372–377. https://doi.org/10.1504/IJBPIM.2015.073665

Cuzzocrea, A., Bellatreche, L., & Song, I.-Y. (2013). Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions. In *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP* (pp. 67–70). New York, NY, USA: ACM. https://doi.org/10.1145/2513190.2517828

Cuzzocrea, A., Song, I. Y., & Davis, K. C. (2011). Analytics over Large-Scale Multidimensional Data: The Big Data Revolution! *DOLAP '11 Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP*, 101–103. Retrieved from http://dl.acm.org/citation.cfm?id=2064695

De Mauro, A., Greco, M., & Grimaldi, M. (2016). A formal definition of Big Data based on its essential features. *LIBRARY REVIEW*, *65*(3), 122–135. https://doi.org/10.1108/LR-06-2015-0061

Dehdouh, K., Bentayeb, F., Boussaid, O., & Kabachi, N. (2015). Using the column oriented NoSQL model for implementing big data warehouses.

Di Tria, F., Lefons, E., & Tangorra, F. (2014). Design process for big data warehouses. In W. W. K. I. Karypis G. Cao L. (Ed.), *DSAA 2014 - Proceedings of the 2014 IEEE International Conference on Data Science and Advanced Analytics* (pp. 512–518). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/DSAA.2014.7058120

Díaz, M., Martín, C., & Rubio, B. (2016). State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications*, *67*, 99–117. https://doi.org/10.1016/j.jnca.2016.01.010

Druid. (2018a). Druid | Aggregations. Retrieved July 6, 2018, from http://druid.io/docs/latest/querying/aggregations.html

Druid. (2018b). Druid | Batch Data Ingestion. Retrieved June 5, 2018, from http://druid.io/docs/latest/ingestion/batch-ingestion.html

Druid. (2018c). Druid | Community and Third Party Software. Retrieved May 20, 2018, from http://druid.io/libraries.html

Druid. (2018d). Druid | Design. Retrieved July 1, 2018, from http://druid.io/docs/latest/design/design.html

Druid. (2018e). Druid | Druid Concepts. Retrieved February 21, 2018, from http://druid.io/docs/latest/design/

Druid. (2018f). Druid | Extensions. Retrieved May 20, 2018, from http://druid.io/docs/latest/development/extensions.html

Druid. (2018g). Druid | FAQ. Retrieved July 3, 2018, from http://druid.io/faq.html

Druid. (2018h). Druid | Granularities. Retrieved July 5, 2018, from http://druid.io/docs/latest/querying/granularities.html

Druid. (2018i). Druid | Index. Retrieved July 5, 2018, from http://druid.io/docs/latest/design/index.html

Druid. (2018j). Druid | indexing-service. Retrieved January 21, 2018, from

http://druid.io/docs/latest/design/indexing-service.html

Druid. (2018k). Druid | Lookups. Retrieved July 11, 2018, from http://druid.io/docs/latest/querying/lookups.html

Druid. (2018l). Druid | Post-Aggregations. Retrieved January 21, 2018, from http://druid.io/docs/latest/querying/post-aggregations.html

Druid. (2018m). Druid | Segments. Retrieved July 5, 2018, from http://druid.io/docs/latest/design/segments.html

Druid. (2018n). Druid | SQL. Retrieved June 6, 2018, from http://druid.io/docs/latest/querying/sql.html

Druid. (2018o). Druid | stream-ingestion. Retrieved February 9, 2018, from http://druid.io/docs/latest/ingestion/stream-ingestion.html

Druid. (2018p). Druid | Tutorial Batch. Retrieved June 6, 2018, from http://druid.io/docs/latest/tutorials/tutorial-batch.html

Du, D. (2015). *Apache hive essentials*. Birmingham: Packt Publ. Retrieved from http://cds.cern.ch/record/2010043

Dumbill, E. (2013). Making Sense of Big Data. *Big Data*, *1*(1), 1–2. https://doi.org/10.1089/big.2012.1503

Economist, T. (2011). *Beyond the PC*. Retrieved from http://www.economist.com/node/21531109

Edlich, S. (2018). NOSQL Databases. Retrieved January 9, 2018, from http://nosql-database.org/

Ekbia, H., Mattioli, M., Kouper, I., Arave, G., Ghazinejad, A., Bowman, T., ... Sugimoto, C. R. (2015). Big data, bigger dilemmas: A critical review. *Journal of the Association for Information Science and Technology*, *66*(8), 1523–1545. https://doi.org/10.1002/asi.23294

Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of Database Systems* (7th ed.). Pearson.

Farooq, F., & Sarwar, S. M. (2010). Real-time data warehousing for business intelligence. In *Proceedings of the 8th International Conference on Frontiers of Information Technology, FIT'10*. Islamabad. https://doi.org/10.1145/1943628.1943666

Fatima, H., & Wasnik, K. (2017). Comparison of SQL, NoSQL and NewSQL databases for internet of things. In *IEEE Bombay Section Symposium 2016: Frontiers of Technology: Fuelling Prosperity of Planet and People, IBSS 2016*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/IBSS.2016.7940198

Freudenreich, T., Furtado, P., Koncilia, C., Thiele, M., Waas, F., & Wrembel, R. (2013). An on-demand ELT architecture for real-time BI. *Lecture Notes in Business Information Processing*, *154*, 50–59.

https://doi.org/10.1007/978-3-642-39872-8_4

Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, *35*(2), 137–144. https://doi.org/10.1016/j.ijinfomgt.2014.10.007

Gessert, F., Wingerath, W., Friedrich, S., & Ritter, N. (2017). NoSQL database systems: a survey and decision guidance. *Computer Science - Research and Development*, *32*(3–4), 353–365. https://doi.org/10.1007/s00450-016-0334-3

Golab, L., & Johnson, T. (2014). Data stream warehousing. In *Proceedings - International Conference on Data Engineering* (pp. 1290–1293). Chicago, IL: IEEE Computer Society. https://doi.org/10.1109/ICDE.2014.6816763

Golfarelli, M., & Rizzi, S. (2009). *Data Warehouse Design: Modern Principles and Methodologies* (1st ed.). New York, NY, USA: McGraw-Hill, Inc.

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. M. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing*, *2*(1). https://doi.org/10.1186/2192-113X-2-22

Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. In *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011* (pp. 363–366). Port Elizabeth. https://doi.org/10.1109/ICPCA.2011.6106531

Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques* (3rd ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Ullah Khan, S. (2015). The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, *47*, 98–115. https://doi.org/10.1016/j.is.2014.07.006

Hive. (2017). Druid Integration - Apache Hive - Apache Software Foundation. Retrieved May 21, 2018, from https://cwiki.apache.org/confluence/display/Hive/Druid+Integration

Hortonworks. (2018a). Chapter 1. Optimizing an Apache Hive Data Warehouse - Hortonworks Data Platform. Retrieved July 11, 2018, from https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.4/bk_hive-performance-tuning/content/ch_hive-perf-tuning-intro.html

Hortonworks. (2018b). Druid - Hortonworks. Retrieved February 21, 2018, from https://br.hortonworks.com/open-source/druid/

Hortonworks. (2018c). Interactive SQL on Hadoop with Hive LLAP - Hortonworks. Retrieved July 4, 2018,

from https://br.hortonworks.com/tutorial/interactive-sql-on-hadoop-with-hive-llap/

Imply. (2018). Pivot: A Fast Data Exploration UI for Druid. Retrieved July 20, 2018, from https://imply.io/post/hello-pivot

Inmon, W. H. (1992). *Building the Data Warehouse*. New York, NY, USA: John Wiley & Sons, Inc.

Inmon, W. H. (2005). *Building the Data Warehouse* (4rth ed.). New York, NY, USA: John Wiley & Sons, Inc.

Jukić, N., Sharma, A., Nestorov, S., & Jukić, B. (2015). Augmenting Data Warehouses with Big Data. *Information Systems Management*, *32*(3), 200–209. https://doi.org/10.1080/10580530.2015.1044338

Kaisler, S., Armour, F., Espinosa, J. A., & Money, W. (2013). Big data: Issues and challenges moving forward. In *Proceedings of the Annual Hawaii International Conference on System Sciences* (pp. 995–1004). Wailea, Maui, HI. https://doi.org/10.1109/HICSS.2013.645

Katal, A., Wazid, M., & Goudar, R. H. (2013). Big data: Issues, challenges, tools and Good practices. In *2013 6th International Conference on Contemporary Computing, IC3 2013* (pp. 404–409). Noida. https://doi.org/10.1109/IC3.2013.6612229

Kaur, K., & Sachdeva, M. (2017). Performance evaluation of NewSQL databases. In *Proceedings of the International Conference on Inventive Systems and Control, ICISC 2017*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICISC.2017.8068585

Khan, M. A.-U.-D., Uddin, M. F., & Gupta, N. (2014). Seven V's of Big Data understanding Big Data to extract value. In *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education - "Engineering Education: Industry Involvement and Interdisciplinary Trends", ASEE Zone 1 2014*. Bridgeport, CT: IEEE Computer Society. https://doi.org/10.1109/ASEEZone1.2014.6820689

Kimball, R. (1996). *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. New York, NY, USA: John Wiley & Sons, Inc.

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling* (3rd ed.). Wiley Publishing.

Krishnan, K. (2013). *Data Warehousing in the Age of Big Data* (1st ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Laney, D. (2001). *3D Data Management: Controlling Data Volume, Velocity, and Variety*.

Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? *Computer*, *43*(2), 12–14. https://doi.org/10.1109/MC.2010.58

Lebdaoui, I., Orhanou, G., & Elhajji, S. (2014). An integration adaptation for real-time datawarehousing. *International Journal of Software Engineering and Its Applications*, *8*(11), 115–128. https://doi.org/10.14257/ijseia.2014.8.11.10

Li, X., & Mao, Y. (2015). Real-Time data ETL framework for big real-time data analysis.

Lima, F. L. G. do V. (2017). *Big Data Warehousing em tempo real: da recolha ao processamento de dados*. Retrieved from https://repositorium.sdum.uminho.pt/handle/1822/53679

Liu, X., Lftikhar, N., & Xie, X. (2014). Survey of real-time processing systems for big data. In *ACM International Conference Proceeding Series* (pp. 356–361). Porto: Association for Computing Machinery. https://doi.org/10.1145/2628194.2628251

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). *Big Data: The Next Frontier for Innovation, Competition, and Productivity*.

Marz, N., & Warren, J. (2015). *Big Data: Principles and Best Practices of Scalable Realtime Data Systems* (1st ed.). Greenwich, CT, USA: Manning Publications Co.

Mishne, G., Dalton, J., Li, Z., Sharma, A., & Lin, J. (2013). Fast data in the era of big data: Twitter's real-time related query suggestion architecture. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 1147–1157). New York, NY. https://doi.org/10.1145/2463676.2465290

Mohamed, N., & Al-Jaroodi, J. (2014). Real-time big data analytics: Applications and challenges. In Z. V Smari W.W. (Ed.), *Proceedings of the 2014 International Conference on High Performance Computing and Simulation, HPCS 2014* (pp. 305–310). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/HPCSim.2014.6903700

Mohanty, S., Jagadeesh, M., & Srivatsa, H. (2013). *Big data imperatives: Enterprise big data warehouse, BI implementations and analytics*. *Big Data Imperatives: Enterprise Big Data Warehouse, BI Implementations and Analytics*. Apress Media LLC. https://doi.org/10.1007/978-1-4302-4873-6

O'Neil, P. E., O'Neil, E. J., & Chen, X. (2009). The Star Schema Benchmark ({SSB}).

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *8*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Presto. (2016). Presto | Distributed SQL Query Engine for Big Data. Retrieved July 5, 2018, from https://prestodb.io/

Russom, P. (2016). *Data Warehouse Modernization In the Age of Big Data Analytics*. Retrieved from http://cdn.techhound.net/wp/60138/60138.pdf

Sá, J. V. de O. e. (2009). *Metodologia de Sistemas de Data Warehouse*. Universidade do Minho. Retrieved from https://repositorium.sdum.uminho.pt/handle/1822/10663?mode=full

Sagiroglu, S., & Sinanc, D. (2013). Big data: A review. In *2013 International Conference on Collaboration Technologies and Systems (CTS)* (pp. 42–47). https://doi.org/10.1109/CTS.2013.6567202

Sahay, B. S., & Ranjan, J. (2008). Real time business intelligence in supply chain analytics. *Information Management and Computer Security*, *16*(1), 28–48. https://doi.org/10.1108/09685220810862733

Santos, M. Y., & Costa, C. (2016a). Data Models in NoSQL Databases for Big Data Contexts. In Y. Tan & Y. Shi (Eds.), *Data Mining and Big Data: First International Conference, DMBD 2016, Bali, Indonesia, June 25-30, 2016. Proceedings* (pp. 475–485). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-40973-3_48

Santos, M. Y., & Costa, C. (2016b). Data warehousing in big data: From multidimensional to tabular data models. In D. E. (Ed.), *ACM International Conference Proceeding Series* (Vol. 20–22–July, pp. 51–60). Association for Computing Machinery. https://doi.org/10.1145/2948992.2949024

Santos, M. Y., Costa, C., Galvão, J., Andrade, C., Martinho, B., Lima, F. L. G. do V., & Costa, E. (2017). Evaluating SQL-on-Hadoop for big data warehousing on not-so-good hardware. In D. B. C. Hong J. McClatchey R. (Ed.), *ACM International Conference Proceeding Series* (Vol. Part F1294, pp. 242–252). Association for Computing Machinery. https://doi.org/10.1145/3105831.3105842

Santos, M. Y., Martinho, B., & Costa, C. (2017). Modelling and implementing big data warehouses for decision support. *Journal of Management Analytics*, *4*(2), 111–129. https://doi.org/10.1080/23270012.2017.1304292

Santos, M. Y., & Ramos, I. (2006). *Business Intelligence: Tecnologias da informação na gestão de conhecimento*. FCA-Editora de Informática.

Santos, M. Y., Sá, J. V. de O. e, Andrade, C., Lima, F. L. G. do V., Costa, E., Costa, C., … Galvão, J. (2017). A Big Data system supporting Bosch Braga Industry 4.0 strategy. *International Journal of Information Management*, *37*(6), 750–760. https://doi.org/10.1016/j.ijinfomgt.2017.07.012

Santos, R. J., & Bernardino, J. (2008). Real-time data warehouse loading methodology. In *ACM International Conference Proceeding Series* (Vol. 299, pp. 49–58). Coimbra. https://doi.org/10.1145/1451940.1451949

Shanklin, C. (2017). Ultra-fast OLAP Analytics with Apache Hive and Druid - Hortonworks. Retrieved July 8, 2018, from https://br.hortonworks.com/blog/apache-hive-druid-part-1-3/

Siddiqa, A., Karim, A., & Gani, A. (2017). Big data storage technologies: a survey. *Frontiers of Information*

Technology and Electronic Engineering, *18*(8), 1040–1070. https://doi.org/10.1631/FITEE.1500441

Vaish, G. (2013). *Getting Started with NoSQL*. Packt Publishing.

Vaishnavi, V., & Kuechler, B. (2015). Design Science Research in Information Systems. *Ais*, 45. https://doi.org/10.1007/978-1-4419-5653-8

Vaisman, A., & Zimnyi, E. (2014). *Data Warehouse Systems: Design and Implementation*. Springer Publishing Company, Incorporated.

Ward, J. S., & Barker, A. (2013). Undefined By Data: A Survey of Big Data Definitions. *CoRR*, *abs/1309.5*.

Webster, J., & Watson, R. T. (2002). Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Q.*, *26*(2), xiii–xxiii. Retrieved from http://dl.acm.org/citation.cfm?id=2017160.2017162

Wu, X., Zhu, X., Wu, G.-Q., & Ding, W. (2014). Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, *26*, 97–107.

Yang, F., Merlino, G., Ray, N., Léauté, X., Gupta, H., & Tschetter, E. (2017). The RADStack: Open Source Lambda Architecture for Interactive Analytics. In *HICSS*.

Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., & Ganguli, D. (2014). Druid: A real-time analytical data store. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 157–168). Snowbird, UT: Association for Computing Machinery. https://doi.org/10.1145/2588555.2595631

Zikopoulos, P., Eaton, C., DeRoos, D., Deutsch, T., & Lapis, G. (2011). *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media; 1 edition (October 19, 2011);

APÊNDICES

Apêndice 1 - Queries

Q1.1

```
SELECT SUM(extendedprice*discount) as revenue
FROM $table
WHERE od_year = 1993 AND discount BETWEEN 1 AND 3 AND quantity < 25
```

Q1.2

```
SELECT SUM(extendedprice*discount) as revenue
FROM $table
WHERE od_yearmonthnum = 199401 AND discount BETWEEN 4 AND 6 AND quantity BETWEEN 26 AND 35
```

Q1.3

```
SELECT SUM(extendedprice*discount) as revenue
FROM $table
WHERE od_weeknuminyear = 6 AND od_year = 1994 AND discount BETWEEN 5 AND 7 AND quantity BETWEEN 26 AND 35
```

Q2.1

```
SELECT SUM(revenue), od_year, p_brand1
FROM $table
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY od_year, p_brand1
ORDER BY od_year, p_brand1
```

Q2.2

```
SELECT SUM(revenue), od_year, p_brand1
FROM $table
```

```
WHERE p_brand1 BETWEEN 'MFGR#2221' AND 'MFGR#2228' AND s_region = 'ASIA'
GROUP BY od_year, p_brand1
ORDER BY od_year, p_brand1
```

Q2.3

```
SELECT SUM(revenue), od_year, p_brand1
FROM $table
WHERE p_brand1= 'MFGR#2239' AND s_region = 'EUROPE'
GROUP BY od_year, p_brand1
ORDER BY od_year, p_brand1
```

Q3.1

```
SELECT c_nation, s_nation, od_year, SUM(revenue) as revenue
FROM $table
WHERE c_region = 'ASIA' AND s_region = 'ASIA' AND od_year >= 1992 AND od_year <= 1997
GROUP BY c_nation, s_nation, od_year
ORDER BY od_year ASC, revenue DESC''
```

Q3.2

```
SELECT c_city, s_city, od_year, SUM(revenue) as revenue
FROM $table
WHERE c_nation = 'UNITED STATES' AND s_nation = 'UNITED STATES' AND od_year >= 1992 AND
od_year <= 1997
GROUP BY c_city, s_city, od_year
ORDER BY od_year ASC, revenue DESC''
```

Q3.3

```
SELECT c_city, s_city, od_year, SUM(revenue) as revenue
FROM $table
WHERE (c_city='UNITED KI1' OR c_city='UNITED KI5') AND (s_city='UNITED KI1' OR s_city='UNITED
KI5') AND od_year >= 1992 AND od_year <= 1997
```

GROUP BY c_city, s_city, od_year

ORDER BY od_year ASC, revenue DESC''

Q3.4

SELECT c_city, s_city, od_year, SUM(revenue) as revenue

FROM $table

WHERE (c_city='UNITED KI1' OR c_city='UNITED KI5') AND (s_city='UNITED KI1' OR s_city='UNITED KI5') AND od_yearmonth = 'Dec1997'

GROUP BY c_city, s_city, od_year

ORDER BY od_year ASC, revenue DESC

Q4.1

SELECT od_year, c_nation, SUM(revenue - supplycost) as profit

FROM $table

WHERE c_region = 'AMERICA' AND s_region = 'AMERICA' AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')

GROUP BY od_year, c_nation

ORDER BY od_year, c_nation

Q4.2

SELECT od_year, s_nation, p_category, SUM(revenue - supplycost) as profit

FROM $table

WHERE c_region = 'AMERICA' AND s_region = 'AMERICA' AND (od_year = 1997 OR od_year = 1998) AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')

GROUP BY od_year, s_nation, p_category

ORDER BY od_year, s_nation, p_category

Q4.3

SELECT od_year, s_city, p_brand1, SUM(revenue - supplycost) as profit

FROM $table

WHERE c_region = 'AMERICA' AND s_nation = 'UNITED STATES' AND (od_year = 1997 OR od_year = 1998) AND p_category = 'MFGR#14'

GROUP BY od_year, s_city, p_brand1

ORDER BY od_year, s_city, p_brand1

Apêndice 2 – Tabela Completa do Cenário NSQP

SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
30	SMonth_QDay_PHashed2	4	6	80	160	39	18	1.12
	SMonth_QDay_PHashed3	4	6		240	27		0.75
	SMonth_QWeek_PHashed2	4	6		160	35		1.12
	SMonth_QWeek_PHashed3	4	6		240	24		0.75
	SQuarter_QDay_PHashed2	4	6	27	54	115		3.33
	SQuarter_QDay_PHashed3	4	6		81	79		2.22
	SQuarter_QWeek_PHashed2	4	6		54	102		3.32
	SQuarter_QWeek_PHashed3	4	6		81	70		2.21
	SQuarter_QMonth_PHashed2	3	5		54	91	17	3.26
	SQuarter_QMonth_PHashed3	4	5		81	62		2.17
	SSemester_QDay_PHashed2	4	6	14	28	223	18	6.42
	SSemester_QDay_PHashed3	4	6		42	152		4.28
	SSemester_QWeek_PHashed2	4	6		28	196		6.40
	SSemester_QWeek_PHashed3	4	6		42	135		4.27
	SSemester_QMonth_PHashed2	3	5		28	176	17	6.29
	SSemester_QMonth_PHashed3	4	5		42	120		4.19
100	SMonth_QDay_PHashed2	13	19	80	160	119	18	3.74
	SMonth_QDay_PHashed3	13	20		240	82		2.49
	SMonth_QDay_PHashed4	13	20		320	63		1.87
	SMonth_QWeek_PHashed2	12	17		160	106		3.70
	SMonth_QWeek_PHashed3	12	17		240	72		2.47
	SMonth_QWeek_PHashed4	12	18		320	55		1.85
	SQuarter_QDay_PHashed2	13	19	27	54	354		11.08
	SQuarter_QDay_PHashed3	13	20		81	243		7.39
	SQuarter_QDay_PHashed4	13	20		135	150		4.43
	SQuarter_QDay_PHashed10	14	21		270	79		2.22
	SQuarter_QDay_PHashed15	14	22		405	54		1.48
	SQuarter_QDay_PHashed20	14	22		540	41		1.11
	SQuarter_QWeek_PHashed2	12	17		54	315		10.98
	SQuarter_QWeek_PHashed3	12	17		81	214		7.32
	SQuarter_QWeek_PHashed5	12	18		135	133		4.39
	SQuarter_QWeek_PHashed10	13	19		270	70		2.20
	SQuarter_QWeek_PHashed15	13	19		405	48		1.46
	SQuarter_QWeek_PHashed20	13	20		540	37		1.10

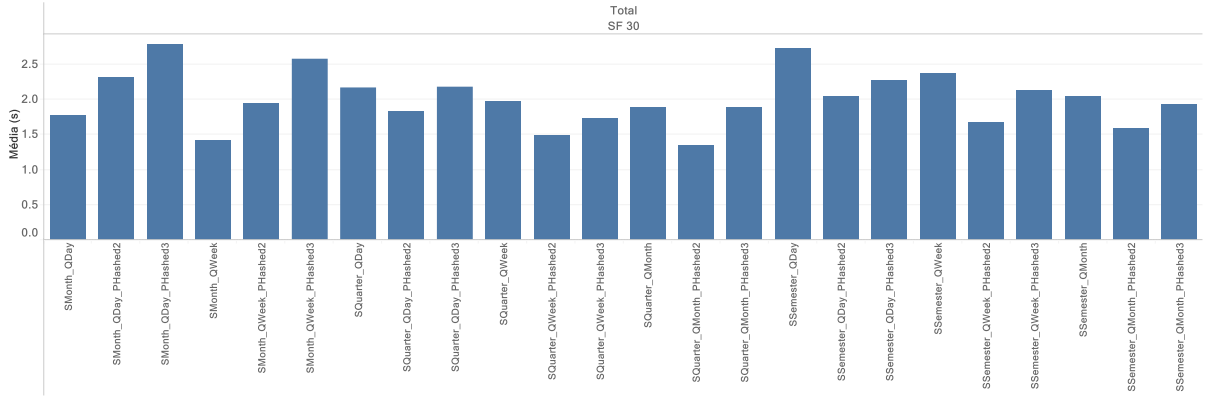
SF	Tabela	(GB)	(GB)	(INT)	(INT)	(MB)	(INT)	(x*10⁶)
	SQuarter_QMonth_PHashed2	10	15		54	280		10.35
	SQuarter_QMonth_PHashed3	11	15		81	189		6.90
	SQuarter_QMonth_PHashed5	11	16		135	116		4.14
	SQuarter_QMonth_PHashed10	11	16		270	60	17	2.07
	SQuarter_QMonth_PHashed15	11	16		405	40		1.38
	SQuarter_QMonth_PHashed20	11	17		540	31		1.03
	SSemester_QDay_PHashed2	13	19		28	682		21.38
	SSemester_QDay_PHashed3	13	20		42	467		14.25
	SSemester_QDay_PHashed5	13	20		70	290		8.55
	SSemester_QDay_PHashed10	14	21		140	152		4.28
	SSemester_QDay_PHashed15	14	22		210	104		2.85
	SSemester_QDay_PHashed20	14	22		280	79		2.14
	SSemester_QDay_PHashed25	14	22		350	63	18	1.71
	SSemester_QWeek_PHashed2	12	17		28	607		21.17
	SSemester_QWeek_PHashed3	12	17		42	412		14.11
	SSemester_QWeek_PHashed5	12	18		70	256		8.47
	SSemester_QWeek_PHashed10	13	19	14	140	134		4.23
	SSemester_QWeek_PHashed15	13	19		210	92		2.82
	SSemester_QWeek_PHashed20	13	20		280	71		2.12
	SSemester_QWeek_PHashed25	13	20		350	57		1.69
	SSemester_QMonth_PHashed2	10	15		28	539		19.96
	SSemester_QMonth_PHashed3	11	15		42	364		13.30
	SSemester_QMonth_PHashed5	11	16		70	223		7.98
	SSemester_QMonth_PHashed10	11	16		140	115	17	3.99
	SSemester_QMonth_PHashed15	11	16		210	78		2.66
	SSemester_QMonth_PHashed20	11	17		280	60		2.00
	SSemester_QMonth_PHashed25	12	17		350	49		1.60
300	SMonth_QDay_PHashed5	37	56		400	141		4.47
	SMonth_QDay_PHashed10	39	59	80	800	74		2.23
	SMonth_QWeek_PHashed5	34	50		400	124		4.34
	SMonth_QWeek_PHashed10	35	51		800	64		2.17
	SQuarter_QDay_PHashed5	37	56		135	416		13.23
	SQuarter_QDay_PHashed10	39	59		270	219		6.62
	SQuarter_QDay_PHashed15	39	61		405	150	18	4.41
	SQuarter_QDay_PHashed20	40	62		540	115		3.31
	SQuarter_QWeek_PHashed5	34	50		135	367		12.86
	SQuarter_QWeek_PHashed10	35	51		270	190		6.43
	SQuarter_QWeek_PHashed15	36	53	27	405	130		4.29
	SQuarter_QWeek_PHashed20	36	54		540	99		3.21
	SQuarter_QMonth_PHashed5	27	40		135	296		10.84
	SQuarter_QMonth_PHashed10	28	41		270	152		5.42
	SQuarter_QMonth_PHashed15	29	42		405	103	17	3.61
	SQuarter_QMonth_PHashed20	29	42		540	78		2.71

Apêndice 3 – Tempos de Processamento por *Query* (segundos)- Cenário NSQP

		Query												
SF	Tabela	Q 1.1	Q 1.2	Q 1.3	Q 2.1	Q 2.2	Q 2.3	Q 3.1	Q 3.2	Q 3.3	Q 3.4	Q 4.1	Q 4.2	Q 4.3
SF 30	SMonth_QDay	0.10	0.05	0.03	0.40	0.16	0.11	0.20	0.08	0.09	0.06	0.31	0.11	0.07
	SMonth_QDay_PHashed2	0.10	0.04	0.03	0.90	0.18	0.14	0.20	0.08	0.14	0.04	0.27	0.10	0.09
	SMonth_QDay_PHashed3	0.11	0.04	0.03	1.35	0.23	0.12	0.22	0.10	0.09	0.05	0.27	0.10	0.08
	SMonth_QWeek	0.10	0.05	0.03	0.31	0.21	0.11	0.13	0.04	0.03	0.03	0.23	0.08	0.06
	SMonth_QWeek_PHashed2	0.11	0.04	0.04	0.76	0.20	0.11	0.15	0.10	0.06	0.05	0.20	0.08	0.06
	SMonth_QWeek_PHashed3	0.10	0.04	0.04	1.41	0.20	0.10	0.14	0.14	0.05	0.03	0.19	0.08	0.06
	SQuarter_QDay	0.14	0.06	0.02	0.58	0.24	0.12	0.22	0.10	0.11	0.05	0.34	0.12	0.07
	SQuarter_QDay_PHashed2	0.09	0.04	0.03	0.40	0.19	0.13	0.21	0.09	0.11	0.05	0.30	0.12	0.08
	SQuarter_QDay_PHashed3	0.07	0.03	0.02	0.81	0.18	0.11	0.25	0.11	0.09	0.04	0.30	0.11	0.07
	SQuarter_QWeek	0.14	0.05	0.04	0.66	0.21	0.14	0.17	0.05	0.05	0.03	0.30	0.09	0.06
	SQuarter_QWeek_PHashed2	0.08	0.04	0.03	0.38	0.17	0.12	0.15	0.05	0.04	0.03	0.24	0.09	0.06
	SQuarter_QWeek_PHashed3	0.14	0.06	0.04	0.56	0.18	0.12	0.14	0.06	0.04	0.03	0.22	0.09	0.06
	SQuarter_QMonth	0.13	0.05		0.59	0.25	0.16	0.18	0.04	0.03	0.02	0.31	0.08	0.05
	SQuarter_QMonth_PHashed2	0.08	0.04		0.39	0.18	0.12	0.14	0.04	0.03	0.02	0.19	0.08	0.05
	SQuarter_QMonth_PHashed3	0.10	0.03		0.76	0.25	0.11	0.20	0.05	0.03	0.04	0.21	0.06	0.05
	SSemester_QDay	0.24	0.06	0.03	0.64	0.23	0.15	0.28	0.12	0.13	0.11	0.42	0.21	0.11
	SSemester_QDay_PHashed2	0.14	0.03	0.02	0.47	0.22	0.12	0.23	0.09	0.09	0.05	0.37	0.13	0.09
	SSemester_QDay_PHashed3	0.11	0.04	0.03	0.71	0.20	0.12	0.26	0.13	0.09	0.04	0.32	0.15	0.08
	SSemester_QWeek	0.25	0.05	0.04	0.54	0.34	0.16	0.20	0.05	0.04	0.03	0.43	0.16	0.08
	SSemester_QWeek_PHashed2	0.14	0.04	0.02	0.45	0.19	0.13	0.16	0.04	0.04	0.03	0.27	0.09	0.06
	SSemester_QWeek_PHashed3	0.12	0.04	0.04	0.80	0.22	0.11	0.21	0.08	0.05	0.02	0.30	0.08	0.07
	SSemester_QMonth	0.24	0.05		0.54	0.24	0.19	0.15	0.04	0.05	0.04	0.31	0.13	0.06
	SSemester_QMonth_PHashed2	0.13	0.03		0.43	0.20	0.12	0.15	0.04	0.03	0.02	0.30	0.08	0.05
	SSemester_QMonth_PHashed3	0.12	0.04		0.73	0.21	0.12	0.19	0.05	0.03	0.02	0.27	0.08	0.05
SF 100	SMonth_QDay	0.34	0.18	0.06	2.40	0.70	0.30	0.54	0.08	0.07	0.03	0.90	0.21	0.10
	SMonth_QDay_PHashed2	0.39	0.10	0.06	1.28	0.59	0.33	0.55	0.16	0.17	0.10	0.73	0.25	0.13
	SMonth_QDay_PHashed3	0.29	0.12	0.07	2.93	0.59	0.30	0.61	0.22	0.28	0.06	0.68	0.20	0.12
	SMonth_QDay_PHashed4	0.39	0.07	0.04	1.90	0.41	0.27	0.52	0.16	0.20	0.06	0.63	0.19	0.13
	SMonth_QWeek	0.31	0.20	0.10	2.39	0.61	0.37	0.41	0.06	0.04	0.03	0.83	0.20	0.09
	SMonth_QWeek_PHashed2	0.32	0.11	0.07	0.86	0.54	0.31	0.37	0.10	0.09	0.07	0.59	0.21	0.10
	SMonth_QWeek_PHashed3	0.40	0.15	0.08	2.94	0.50	0.31	0.45	0.07	0.05	0.03	0.54	0.15	0.09
	SMonth_QWeek_PHashed4	0.25	0.05	0.04	3.18	0.50	0.30	0.49	0.30	0.06	0.04	0.53	0.17	0.15
	SQuarter_QDay	0.47	0.16	0.05	1.90	0.70	0.37	0.56	0.11	0.09	0.04	1.05	0.35	0.20
	SQuarter_QDay_PHashed2	0.25	0.09	0.04	1.93	0.63	0.31	0.55	0.20	0.14	0.05	0.90	0.29	0.13
	SQuarter_QDay_PHashed3	0.43	0.08	0.05	1.69	0.69	0.34	0.52	0.16	0.19	0.07	0.95	0.30	0.14
	SQuarter_QDay_PHashed5	0.39	0.11	0.06	2.44	0.53	0.32	0.64	0.41	0.17	0.05	0.83	0.23	0.15
	SQuarter_QDay_PHashed10	0.40	0.17	0.09	3.06	0.55	0.28	0.85	0.38	0.20	0.07	0.83	0.24	0.16
	SQuarter_QDay_PHashed15	0.59	0.23	0.13	3.50	0.52	0.28	0.80	0.56	0.26	0.07	0.90	0.27	0.17
	SQuarter_QDay_PHashed20	0.64	0.26	0.16	3.75	0.55	0.30	1.14	0.37	0.23	0.08	0.89	0.27	0.18
	SQuarter_QWeek	0.46	0.14	0.06	2.30	0.58	0.39	0.46	0.06	0.06	0.03	1.06	0.31	0.11
	SQuarter_QWeek_PHashed2	0.25	0.11	0.06	1.89	0.65	0.35	0.41	0.09	0.08	0.03	1.10	0.22	0.10
	SQuarter_QWeek_PHashed3	0.31	0.08	0.05	2.07	0.61	0.33	0.40	0.07	0.10	0.04	0.85	0.19	0.09
	SQuarter_QWeek_PHashed5	0.40	0.12	0.09	2.43	0.49	0.30	0.42	0.08	0.06	0.03	0.70	0.19	0.10
	SQuarter_QWeek_PHashed10	0.42	0.16	0.09	3.21	0.50	0.28	0.53	0.09	0.08	0.04	0.57	0.19	0.10
	SQuarter_QWeek_PHashed15	0.51	0.23	0.16	3.72	0.50	0.28	0.46	0.21	0.11	0.05	0.60	0.18	0.10
	SQuarter_QWeek_PHashed20	0.58	0.29	0.15	3.23	0.55	0.29	0.78	0.17	0.12	0.05	0.63	0.20	0.12
	SQuarter_QMonth	0.49	0.14		1.84	0.58	0.35	0.38	0.05	0.03	0.03	0.81	0.22	0.08
	SQuarter_QMonth_PHashed2	0.26	0.10		1.66	0.57	0.32	0.37	0.05	0.03	0.03	0.80	0.19	0.07
	SQuarter_QMonth_PHashed3	0.27	0.06		2.30	0.48	0.32	0.34	0.06	0.03	0.02	0.68	0.19	0.07
	SQuarter_QMonth_PHashed5	0.30	0.14		2.51	0.49	0.30	0.39	0.06	0.04	0.03	0.57	0.14	0.09
	SQuarter_QMonth_PHashed10	0.43	0.19		3.15	0.48	0.30	0.33	0.11	0.05	0.03	0.44	0.14	0.09
	SQuarter_QMonth_PHashed15	0.54	0.23		3.48	0.54	0.28	0.34	0.07	0.05	0.03	0.44	0.14	0.08
	SQuarter_QMonth_PHashed20	0.64	0.27		3.23	0.59	0.29	0.34	0.07	0.05	0.03	0.46	0.15	0.09
	SSemester_QDay	0.74	0.13	0.06	1.53	0.85	0.44	0.64	0.12	0.11	0.05	1.22	0.51	0.22
	SSemester_QDay_PHashed2	0.43	0.08	0.04	1.67	0.66	0.33	0.55	0.13	0.15	0.05	1.01	0.35	0.15
	SSemester_QDay_PHashed3	0.31	0.07	0.06	2.19	0.76	0.34	0.56	0.20	0.19	0.06	1.22	0.37	0.13
	SSemester_QDay_PHashed5	0.36	0.10	0.05	2.26	0.68	0.34	0.68	0.27	0.21	0.06	1.04	0.31	0.16
	SSemester_QDay_PHashed10	0.35	0.12	0.07	2.41	0.58	0.30	0.71	0.38	0.20	0.07	0.96	0.27	0.16
	SSemester_QDay_PHashed15	0.41	0.14	0.08	2.87	0.52	0.35	0.82	0.52	0.22	0.07	0.86	0.26	0.19
	SSemester_QDay_PHashed20	0.44	0.17	0.09	3.44	0.59	0.29	0.85	0.58	0.26	0.08	0.92	0.27	0.19
	SSemester_QDay_PHashed25	0.22	0.07	0.06	3.43	0.53	0.30	0.91	0.49	0.26	0.10	0.96	0.27	0.19
	SSemester_QWeek	0.73	0.13	0.07	1.96	0.74	0.48	0.53	0.07	0.04	0.04	1.30	0.43	0.15
	SSemester_QWeek_PHashed2	0.40	0.10	0.07	2.00	0.55	0.39	0.45	0.09	0.07	0.08	1.10	0.21	0.11

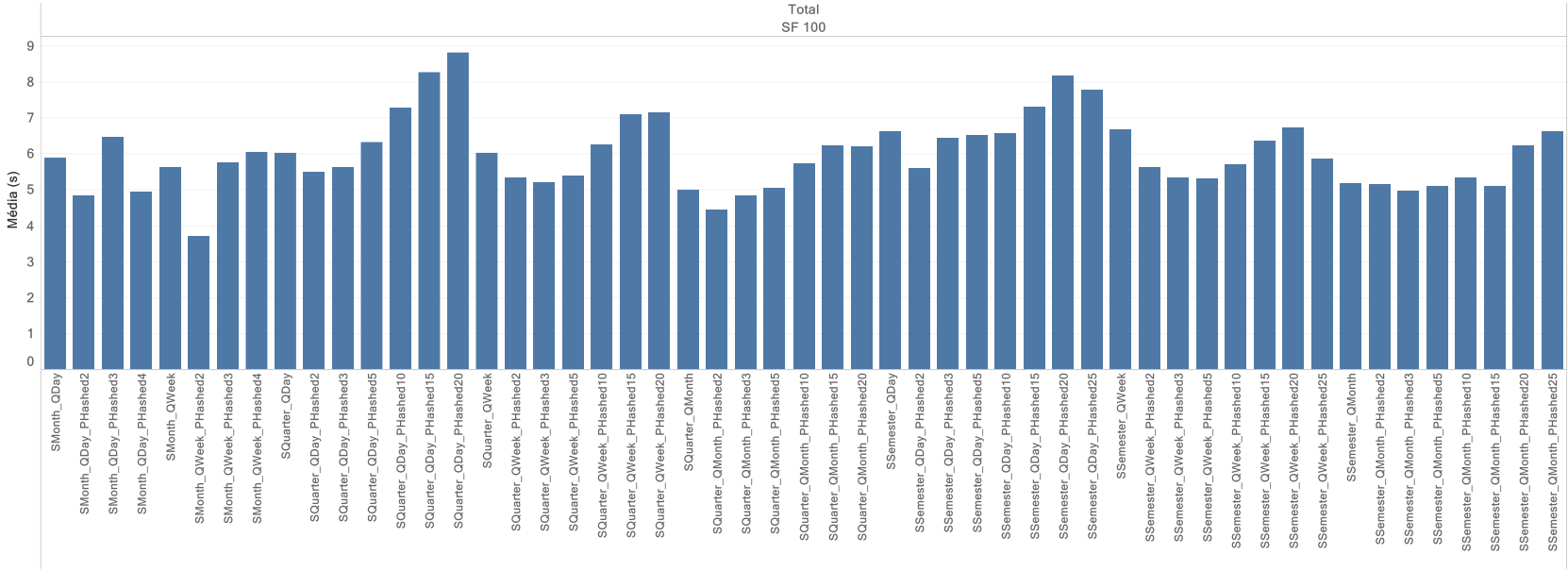
	SSemester_QWeek_PHashed3	0.34	0.08	0.07	2.06	0.59	0.34	0.40	0.08	0.05	0.03	0.92	0.26	0.11
	SSemester_QWeek_PHashed5	0.24	0.07	0.07	2.08	0.56	0.31	0.44	0.08	0.07	0.03	1.03	0.22	0.11
	SSemester_QWeek_PHashed10	0.33	0.11	0.08	2.63	0.47	0.31	0.45	0.14	0.19	0.05	0.65	0.18	0.10
	SSemester_QWeek_PHashed15	0.37	0.15	0.10	3.05	0.51	0.29	0.49	0.17	0.22	0.06	0.65	0.20	0.12
	SSemester_QWeek_PHashed20	0.42	0.19	0.11	3.11	0.52	0.28	0.59	0.25	0.22	0.06	0.68	0.21	0.12
	SSemester_QWeek_PHashed25	0.32	0.12	0.08	2.40	0.58	0.32	0.52	0.21	0.20	0.09	0.67	0.21	0.14
	SSemester_QMonth	0.66	0.14		1.45	0.67	0.40	0.45	0.05	0.03	0.03	0.89	0.32	0.08
	SSemester_QMonth_PHashed2	0.40	0.08		1.85	0.54	0.40	0.36	0.05	0.04	0.04	1.09	0.23	0.08
	SSemester_QMonth_PHashed3	0.39	0.09		1.87	0.57	0.33	0.36	0.05	0.03	0.03	0.96	0.18	0.10
	SSemester_QMonth_PHashed5	0.32	0.10		1.99	0.63	0.33	0.39	0.09	0.07	0.05	0.86	0.21	0.10
	SSemester_QMonth_PHashed10	0.27	0.10		2.71	0.48	0.30	0.45	0.07	0.06	0.04	0.62	0.16	0.08
	SSemester_QMonth_PHashed15	0.38	0.15		2.58	0.45	0.28	0.41	0.07	0.05	0.03	0.49	0.15	0.08
	SSemester_QMonth_PHashed20	0.41	0.18		3.43	0.48	0.31	0.49	0.11	0.08	0.04	0.47	0.16	0.09
	SSemester_QMonth_PHashed25	0.37	0.09		3.37	0.56	0.31	0.46	0.41	0.14	0.06	0.56	0.19	0.12
SF 300	SMonth_QDay	0.86	0.34	0.11	3.96	1.84	0.91	1.05	0.15	0.11	0.07	2.78	0.58	0.23
	SMonth_QDay_PHashed5	0.63	0.12	0.07	2.40	1.44	0.77	1.17	0.24	0.30	0.10	1.95	0.47	0.24
	SMonth_QDay_PHashed10	0.62	0.10	0.08	9.30	1.48	0.87	2.10	0.72	0.42	0.15	1.99	0.58	0.29
	SMonth_QWeek	0.82	0.32	0.13	4.20	1.89	0.88	1.02	0.11	0.07	0.06	3.31	0.49	0.15
	SMonth_QWeek_PHashed5	0.50	0.09	0.05	2.23	1.30	0.86	0.90	0.15	0.09	0.05	1.69	0.40	0.17
	SMonth_QWeek_PHashed10	0.51	0.09	0.10	8.95	1.65	0.87	1.30	0.70	0.14	0.18	1.59	0.40	0.19
	SQuarter_QDay	1.25	0.38	0.10	4.61	1.85	1.17	1.20	0.16	0.11	0.05	3.21	0.73	0.28
	SQuarter_QDay_PHashed5	0.63	0.11	0.06	3.75	1.43	0.82	1.31	0.25	0.24	0.07	2.28	0.58	0.35
	SQuarter_QDay_PHashed10	0.59	0.11	0.08	3.59	1.47	0.82	1.46	0.36	0.46	0.13	2.51	0.57	0.35
	SQuarter_QDay_PHashed15	0.57	0.10	0.06	3.02	1.52	0.82	1.56	0.42	0.50	0.16	2.45	0.67	0.37
	SQuarter_QDay_PHashed20	0.49	0.10	0.04	7.11	1.38	0.92	2.02	1.26	0.52	0.20	2.56	0.71	0.39
	SQuarter_QWeek	1.24	0.37	0.16	3.93	1.68	1.03	1.10	0.12	0.07	0.04	2.20	0.72	0.22
	SQuarter_QWeek_PHashed5	0.51	0.11	0.07	4.50	1.55	1.08	1.36	0.15	0.15	0.05	2.59	0.47	0.18
	SQuarter_QWeek_PHashed10	0.50	0.12	0.09	4.76	1.51	0.82	1.19	0.16	0.30	0.10	2.54	0.43	0.21
	SQuarter_QWeek_PHashed15	0.47	0.08	0.05	2.29	1.31	0.82	1.05	0.15	0.13	0.05	1.76	0.47	0.22
	SQuarter_QWeek_PHashed20	0.85	0.13	0.07	4.52	1.21	0.79	1.20	0.17	0.14	0.06	1.57	0.41	0.20
	SQuarter_QMonth	0.97	0.26		3.05	1.03	0.71	0.88	0.07	0.03	0.04	1.50	0.37	0.07
	SQuarter_QMonth_PHashed5	0.75	0.08		2.43	1.18	0.78	0.83	0.08	0.04	0.03	1.61	0.31	0.12
	SQuarter_QMonth_PHashed10	0.46	0.08		2.35	1.07	0.76	0.85	0.10	0.05	0.04	1.38	0.34	0.14
	SQuarter_QMonth_PHashed15	0.59	0.10		6.43	1.38	0.77	1.05	0.14	0.12	0.06	1.42	0.37	0.17
	SQuarter_QMonth_PHashed20	0.56	0.11		7.68	1.35	0.74	0.86	0.15	0.18	0.14	1.72	0.36	0.19

Apêndice 4 - Tempos Totais de Processamento, SF 30 - Cenário NSQP



	SMonth_QDay	SMonth_QDay_PHashed2	SMonth_QDay_PHashed3	SMonth_QWeek	SMonth_QWeek_PHashed2	SMonth_QWeek_PHashed3	SQuarter_QDay	SQuarter_QDay_PHashed2	SQuarter_QDay_PHashed3	SQuarter_QWeek	SQuarter_QWeek_PHashed2	SQuarter_QWeek_PHashed3	SQuarter_QMonth	SQuarter_QMonth_PHashed2	SQuarter_QMonth_PHashed3	SSemester_QDay	SSemester_QDay_PHashed2	SSemester_QDay_PHashed3	SSemester_QWeek	SSemester_QWeek_PHashed2	SSemester_QWeek_PHashed3	SSemester_QMonth	SSemester_QMonth_PHashed2	SSemester_QMonth_PHashed3
Run 1 (s)	2.54	3.40	4.85	2.11	2.95	4.56	3.70	2.53	3.22	3.34	2.28	2.63	3.11	1.96	2.90	4.04	2.97	3.50	4.00	2.68	3.98	3.47	2.40	3.10
Média (s)	1.76	2.31	2.79	1.42	1.94	2.57	2.17	1.83	2.17	1.97	1.48	1.73	1.88	1.35	1.87	2.73	2.04	2.27	2.36	1.67	2.13	2.05	1.59	1.92
Dif. Média (%)		30.85%	20.87%	-49.26%	37.26%	32.49%	-15.68%	-15.46%	18.48%	-9.39%	-24.84%	16.74%	8.88%	-28.19%	38.66%	45.75%	-25.21%	11.39%	3.85%	-29.51%	28.20%	-4.18%	-22.45%	20.92%

Apêndice 5 - Tempos Totais de Processamento, SF 100 - Cenário NSQP



	SMonth_QDay	SMonth_QDay_PHashed2	SMonth_QDay_PHashed3	SMonth_QDay_PHashed4	SMonth_QWeek	SMonth_QWeek_PHashed2	SMonth_QWeek_PHashed3	SMonth_QWeek_PHashed4	SQuarter_QDay	SQuarter_QDay_PHashed2	SQuarter_QDay_PHashed3	SQuarter_QDay_PHashed5	SQuarter_QDay_PHashed10	SQuarter_QDay_PHashed15	SQuarter_QDay_PHashed20	SQuarter_QWeek	SQuarter_QWeek_PHashed2	SQuarter_QWeek_PHashed3	SQuarter_QWeek_PHashed5	SQuarter_QWeek_PHashed10	SQuarter_QWeek_PHashed15	SQuarter_QWeek_PHashed20	SQuarter_QMonth	SQuarter_QMonth_PHashed2	SQuarter_QMonth_PHashed3	SQuarter_QMonth_PHashed5	SQuarter_QMonth_PHashed10	SQuarter_QMonth_PHashed15	SQuarter_QMonth_PHashed20
Run 1 (s)	11.03	8.12	11.66	10.81	11.22	6.50	10.58	10.85	10.35	9.12	9.58	10.77	13.26	15.82	16.66	10.23	8.89	9.30	9.94	12.21	13.99	14.10	9.35	7.83	8.62	9.11	11.16	12.74	12.31
Média (s)	5.91	4.85	6.47	4.95	5.63	3.72	5.75	6.06	6.03	5.50	5.62	6.34	7.27	8.28	8.83	6.03	5.34	5.21	5.40	6.26	7.11	7.16	5.00	4.45	4.84	5.06	5.74	6.23	6.21
Dif. Média (%)		-18.04%	33.55%	-23.47%	13.66%	-33.91%	54.55%	5.34%	-0.38%	-8.83%	2.22%	12.79%	14.67%	13.82%	6.69%	-31.78%	-11.32%	-2.56%	3.81%	15.88%	13.55%	0.67%	-30.14%	-11.01%	8.67%	4.63%	13.40%	8.51%	-0.23%

	SSemester_QDay	SSemester_QDay_PHashed2	SSemester_QDay_PHashed3	SSemester_QDay_PHashed5	SSemester_QDay_PHashed10	SSemester_QDay_PHashed15	SSemester_QDay_PHashed20	SSemester_QDay_PHashed25	SSemester_QWeek	SSemester_QWeek_PHashed2	SSemester_QWeek_PHashed3	SSemester_QWeek_PHashed5	SSemester_QWeek_PHashed10	SSemester_QWeek_PHashed15	SSemester_QWeek_PHashed20	SSemester_QWeek_PHashed25	SSemester_QMonth	SSemester_QMonth_PHashed2	SSemester_QMonth_PHashed3	SSemester_QMonth_PHashed5	SSemester_QMonth_PHashed10	SSemester_QMonth_PHashed15	SSemester_QMonth_PHashed20	SSemester_QMonth_PHashed25
Run 1 (s)	11.50	9.16	10.13	11.08	11.49	12.87	14.78	13.14	12.56	9.34	9.69	9.05	10.31	11.60	12.72	10.12	9.45	8.41	8.49	9.17	9.09	9.40	11.34	12.20					
Média (s)	6.63	5.61	6.47	6.53	6.58	7.31	8.18	7.78	6.68	5.63	5.33	5.31	5.71	6.38	6.74	5.87	5.18	5.16	4.99	5.12	5.33	5.11	6.24	6.63					
Dif. Média (%)		-15.31%	15.22%	1.02%	0.72%	11.15%	11.81%	-4.83%	-14.17%	-15.77%	-5.33%	-0.37%	7.67%	11.71%	5.57%	-12.89%	-11.81%	-0.34%	-3.26%	2.57%	4.19%	-4.17%	22.07%	6.23%					

Apêndice 6 – Tempos de Processamento Obtidos no Hive (tempo impresso pela Beeline) e no Druid

	Hive		Druid	
Query	Run 1 (s)	Média (s)	Run 1 (s)	Média (s)
Q 1.1	8.60	2.33	3.32	0.86
Q 1.2	1.05	0.42	1.26	0.34
Q 1.3	0.85	0.39	0.37	0.11
Q 2.1	8.60	4.37	7.67	3.96
Q 2.2	1.98	1.96	2.49	1.84
Q 2.3	0.92	0.90	1.00	0.91
Q 3.1	1.40	1.28	1.06	1.05
Q 3.2	0.36	0.38	0.15	0.15
Q 3.3	0.28	0.28	0.12	0.11
Q 3.4	0.28	0.23	0.07	0.07
Q 4.1	4.48	2.67	4.76	2.78
Q 4.2	0.89	0.79	0.71	0.58
Q 4.3	0.51	0.38	0.25	0.23
Total	30.20	16.38	23.22	13.01
Dif. (%)	——	——	- 23	- 21

Apêndice 7 - Tempos de Processamento Obtidos no Hive (tempo impresso pela Beeline) e no Druid em Ambiente Multiutilizador

	Hive		Druid	
Query	Run 1 (s)	Média (s)	Run 1 (s)	Média (s)
Q 1.1	6.50	1.92	4.69	1.19
Q 1.2	0.67	0.48	0.60	0.17
Q 1.3	0.68	0.45	0.16	0.06
Q 2.1	6.03	3.13	8.15	4.38
Q 2.2	2.20	2.09	3.93	3.39
Q 2.3	1.27	1.24	2.25	2.14
Q 3.1	4.86	2.64	4.84	4.83
Q 3.2	0.76	0.73	1.21	1.44
Q 3.3	0.54	0.57	0.90	1.23
Q 3.4	0.42	0.37	1.45	1.29
Q 4.1	5.89	2.80	7.27	4.60
Q 4.2	1.21	1.15	1.88	1.47
Q 4.3	0.86	0.65	1.35	1.23
Total	31.89	18.23	38.69	27.42
Dif. (%)	——	——	21	50
Aumento (%)	6	11	67	111

Apêndice 8 – Tarefa de Ingestão em *Real-time* "iDBRealTimeRawData"


```
{
    "dataSources": {
        "ALRRawData": {
            "spec": {
                "dataSchema": {
                    "dataSource": "ALRRawData",
                    "parser": {
                        "type": "string",
                        "parseSpec": {
                            "timestampSpec": {
                                "column": "timestamp",
                                "format": "auto"
                            },
                            "dimensionsSpec": {
                                "dimensions": [
                                    "lotObservation", "lotState", "lotUuid",
                                    "lotValidationDate", "lotCreationDate", "palletId",
                                    "palletFinishDate", "palletHuCode", "palletLine",
                                    "palletOrigin", "palletCode", "palletPartNumber",
                                    "productId", "productBox", "productDate",
                                    "productLine", "productOrigin", "productPartNumber",
                                    "productSerialNumber", "rulesDesignation",
                                    "rulesValid", "rulesReason", "classId",
                                    "deffectCauseDescription", "flawId",
                                    "partNumber", "serialNumber", "station"
                                ]
                            },
                            "format": "json"
                        }
                    },
                    "granularitySpec": {
                        "type": "uniform",
                        "segmentGranularity": "hour",
                        "queryGranularity": "none"
                    },
                    "metricsSpec": [
                        {
                            "type": "count",
                            "name": "count"
                        },
                        {
                            "type": "doubleSum",
                            "name": "numberOfOccurrences",
                            "fieldName": "numberOfOccurrences"
                        },
                        {
                            "type": "doubleSum",
                            "name": "maxNumberOfOccurrences",
                            "fieldName": "maxNumberOfOccurrences"
                        }
                    ]
                },
                "ioConfig": {
                    "type": "realtime"
                },
                "tuningConfig": {
                    "type": "realtime",
                    "maxRowsInMemory": "100000",
                    "intermediatePersistPeriod": "PT10M",
                    "windowPeriod": "PT10M"
                }
            },
            "properties": {
                "task.partitions": "1",
                "task.replicants": "1"
            }
        }
    },
    "properties": {
        "zookeeper.connect": "$host:$port",
        "druid.discovery.curator.path": "/druid/discovery",
        "druid.selectors.indexing.serviceName": "druid/overlord"
    }
}
```

Apêndice 9 – Tarefa de Ingestão em Batch “iDBBatchRawData”

```
{
  "type" : "index_hadoop",
  "spec" : {
    "ioConfig" : {
      "type" : "hadoop",
      "inputSpec" : {
        "type" : "static",
        "paths" : "/user/lid4/correia/IntelligentDataBroker/batch_alr/"
      }
    },
    "dataSchema" : {
      "dataSource" : "ALRRawData",
      "granularitySpec" : {
        "type" : "uniform",
        "segmentGranularity" : "day",
        "queryGranularity" : "none",
        "intervals" : ["2018-03-02/2018-04-11"]
      },
      "parser" : {
        "type" : "hadoopyString",
        "parseSpec" : {
          "format" : "json",
          "dimensionsSpec" : {
            "dimensions" : [
              "lotObservation", "lotState", "lotUuid", "lotValidationDate",
              "lotCreationDate", "palletId", "palletFinishDate", "palletHuCode"
              "palletLine", "palletOrigin", "palletCode", "palletPartNumber",
              "productId", "productBox", "productDate", "productLine",
              "productOrigin", "productPartNumber", "productSerialNumber",
              "rulesDesignation", "rulesValid", "rulesReason",
              "classId", "deffectCauseDescription", "flawId",
              "partNumber", "serialNumber", "station"
            ]
          },
          "timestampSpec" : {
            "format" : "auto",
            "column" : "timestamp"
          }
        }
      },
      "metricsSpec" : [
        {
          "name" : "count",
          "type" : "count"
        },
        {
          "type" : "doubleSum",
          "name" : "numberOfOccurrences",
          "fieldName" : "numberOfOccurrences"
        },
        {
          "type" : "doubleSum",
          "name" : "maxNumberOfOccurrences",
          "fieldName" : "maxNumberOfOccurrences"
        }
      ]
    },
    "tuningConfig" : {
      "type" : "hadoop",
      "jobProperties" : {
      }
    }
  }
}
```

Apêndice 10 – Primeiro KPI

```
{
   "queryType": "groupBy",
   "dataSource": "ALRRawData",
   "descending": "false",
   "metrics":["countValidLots, countLots"],
   "filter":
        { "type": "not", "field":{"type": "selector", "dimension": "lotState", "value": "CREATED"}}
   ,
   "aggregations": [
        {
            "type" : "filtered",
            "filter" :
            {
                "type" : "selector",
                "dimension" : "lotState",
                "value" : "INVALID"
            },
            "aggregator" :
            {
                "type" : "cardinality",
                "name" : "countDayInvalidLots",
                "fields" : ["lotUuid"],
                "round": true
                }
        },
        {
            "type" : "cardinality",
            "name" : "countDayLots",
            "fields" : ["lotUuid"],
            "round": true
        }
    ],
   "postAggregations" : [
        {
            "type"  : "arithmetic",
            "name"  : "percInvalid",
            "fn"    : "/",
            "fields": [
                {"type" : "hyperUniqueCardinality", "fieldName" : "countDayInvalidLots"},
                {"type" : "hyperUniqueCardinality", "fieldName" : "countDayLots"}
            ]
        }
    ],
   "granularity": "all",
   "having":
        {
            "type": "greaterThan",
            "aggregation": "percInvalid",
            "value": 0.05
        },
   "intervals": ["2018-03-07/2018-03-08"]
}
```

Apêndice 11 – Terceiro KPI

```
{
   "queryType": "groupBy",
   "dataSource": "ALRRawData",
   "descending": "false",
   "dimensions":["productLine"],
   "metrics":["countValidLots, countLots"],
   "filter":
        { "type": "not", "field":{"type": "selector", "dimension": "lotState", "value": "CREATED"}}
   ,
   "aggregations": [
        {
            "type" : "filtered",
            "filter" :
                {
                    "type": "and",
                    "fields":
                    [
                        {
                            "type" : "interval",
                            "dimension" : "__time",
                            "intervals" : ["2018-03-07/2018-03-08"]
                        },
                        {
                            "type" : "selector",
                            "dimension" : "lotState",
                            "value" : "VALID"
                        }
                    ]
                },
            "aggregator" : {
                "type" : "cardinality",
                "name" : "countDayProducts",
                "fields" : ["productSerialNumber", "lotUuid"],
                "round": true
                    }
            },
            {
                "type" : "cardinality",
                "name" : "countProducts",
                "fields" : ["productSerialNumber", "lotUuid"],
                "round": true
            }
    ],
   "postAggregations" : [
        {
            "type"    : "arithmetic",
            "name"    : "percDifference",
            "fn"      : "/",
            "fields" : [
                {"type" : "hyperUniqueCardinality", "fieldName" : "countDayProducts"},
                {
                    "type"  : "arithmetic",
                    "name"  : "avgProducts",
                    "fn"    : "/",
                    "fields": [
                        {"type" : "hyperUniqueCardinality", "fieldName" : "countProducts"},
                        {"type" : "constant", "name": "days", "value" : "6"}
                    ]
                }
            ]
        }
    ],
   "granularity": "all",
   "having":
        {
            "type": "and",
            "havingSpecs":
            [
                {
                    "type": "lessThan",
                    "aggregation": "percDifference",
                    "value": 1
                },
                {
                    "type": "greaterThan",
                    "aggregation": "percDifference",
                    "value": 0
                }
            ]
        },
   "intervals": ["2018-03-02/2018-03-08"]
}
```