FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Suporte para Séries Temporais em Plataforma *e-Science*

José Tiago Paiva Antunes Magalhães

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. João Correia Lopes

27 de Julho de 2015

Suporte para Séries Temporais em Plataforma *e-Science*

José Tiago Paiva Antunes Magalhães

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Prof. Maria Cristina Ribeiro
Arguente: Prof. Maria Benedita Malheiro
Vogal: Prof. João Correia Lopes

---

27 de Julho de 2015

Resumo

Nos últimos anos têm ganho popularidade soluções de gestão de dados que não seguem a abordagem objeto-relacional tradicional, nos casos em que não é necessário manter as propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (ACID) nem é necessária a utilização de SQL para a interrogação às bases de dados onde são guardados esses dados. As bases de dados NoSQL diferem da abordagem relacional por usarem estruturas de dados chave-valor, coluna, grafo ou documento, e estão a ser cada vez mais usadas em aplicações que tratam a chamada *Big data*.

No domínio de aplicação das Ciências da Terra, nomeadamente a utilização de sensores laser (*LiDAR*) para a análise das condições de vento em instalações de torres eólicas de produção de energia elétrica, são produzidas séries temporais usadas posteriormente por investigadores da área nos seus trabalhos de investigação. Devido à enorme quantidade de informação que é produzida por estes sensores, torna-se ineficiente a utilização de base de dados relacionais para o armazenamento das séries temporais produzidas.

O objetivo deste trabalho consiste na produção de uma plataforma *e-Science*, inovadora para investigadores, focada no armazenamento e disponibilização de toda a meta-informação e informação necessária à correta interpretação dos resultados científicos obtidos, nomeadamente a rastreabilidade e a reprodutibilidade dos dados. Esta plataforma será suportada por duas abordagens distintas: uma com recurso a uma base de dados relacional e uma outra com recurso a uma base de dados NoSQL. Para a sua implementação foi estudado com detalhe o impacto que as duas abordagens terão no desempenho da solução, nomeadamente no armazenamento e disponibilização dos dados para os investigadores.

Os testes efetuados permitem concluir que a abordagem que faz uso de um sistema de base de dados NoSQL apresenta um desempenho muito superior na generalidade dos casos, quando comparada com a primeira abordagem. O estudo realizado revela ainda que as limitações descobertas na primeira abordagem inviabilizam a sua utilização para grandes quantidades de dados. Por fim, a segunda abordagem revela-se mais escalável que a primeira, propriedade importante para se conseguir manter a disponibilidade da plataforma, à medida que o número de utilizadores e a quantidade de dados aumentam.

Abstract

In the last few years database management systems solutions, that do not follow the traditional object-relational approach, have gained popularity in specific cases where it is not necessary to maintain ACID properties or use SQL to query the database. The NoSQL databases differ from the relational approach because they use different kinds of structures to store the data like key-value data structures, columns, graphs or documents and they are being increasingly used in applications that deal with the so called Big data.

In the scope of Earth Sciences, including the use of laser sensors (LiDAR) for the analysis of wind conditions on wind towers facilities for electricity production, time series are produced and later used by researchers in their work research. Due to the huge amount of information that is produced by these sensors, it is inefficient to use relational database management systems to store time series.

The objective of this dissertation is to develop an innovative e-Science platform focused on the storage and availability of all meta-information and information necessary for correct interpretation of the scientific results, including repeatability and reproducibility. This platform is supported by two approaches: one using a relational database and another one using a NoSQL database. For its implementation it was studied, in detail, the impact that the two approaches have in the performance of the solution, especially in the storage and availability of data.

The tests performed showed that the approach which uses a NoSQL database management system provides a far superior performance compared with the first approach. The study also reveals that the limitations discovered in the first approach makes it unfeasible for a large amount of data. Finally, the second approach proves to be more scalable than the first which is an important property to be able to maintain the platform availability as the number of users and amount of data increase.

Agradecimentos

O meu percurso não teria sido possível sem a ajuda de muitas pessoas que me seguiram nesta caminhada e a quem estou eternamente grato.

Em primeiro lugar, quero agradecer aos meus pais pelo empenho, apoio e dedicação que sempre demonstraram para comigo e pelo esforço financeiro que fizeram para tornar possível o meu percurso académico.

Em segundo lugar, quero agradecer à minha namorada, Bárbara, pela paciência e compreensão demonstradas em todo o meu percurso, mesmo em alturas de maior ausência da minha parte.

Em terceiro lugar, quero agradecer ao meu orientador, Professor João Correia Lopes, pelos ensinamentos que me transmitiu e pela alta disponibilidade demonstrada durante a realização desta dissertação.

Em último lugar, quero agradecer a todos os meus amigos que me acompanharam nesta caminhada, sem eles este percurso não teria possível.

José Magalhães

"First, solve the problem.
Then, write the code."

John Johnson

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ACID	Atomicity, Consistency, Isolation, Durability
ACM	Association for Computing Machinery
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BJSON	Binary JavaScript Object Notation
BLOB	Binary Large Object
CAP	Consistency, Availability, Partition tolerance
CRUD	Create, Remove, Update, Delete
CSS	Cascading Style Sheets
DAO	Data Access Object
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LiDAR	Light Detection And Ranging
MVC	Model View Controller
ORM	Object Relational Mapping
PDF	Portable Document Format
RDBMS	Relational DataBase Management System
REST	Representational State Transfer
SQL	Structured Query Language
VNC	Virtual Network Computing
XML	eXtensible Markup Language

Capítulo 1

Introdução

1.1 Contexto e Motivação

As infraestruturas de *cloud computing* (por exemplo, Amazon[1]) e da nova geração de plataformas de computação intensiva de dados (por exemplo, DISC[2], Google MapReduce[3], Hadoop[4] e Dryad[5]) estão vocacionadas para gestão e processamento de grandes quantidades de dados.

A gestão de dados científicos produzidos por um indivíduo ou uma instituição no decurso da sua investigação é cada vez mais importante, uma vez que esses mesmos dados têm um elevado custo de produção, apresentam utilidade para as comunidades científicas e apoiam os resultados da investigação que poderão vir a ter impacto na sociedade. O emergente "Quarto Paradigma da Ciência" [Col10], que surge do acesso cada vez mais generalizado a poderosas capacidades de computação em ambientes em rede, permite que os investigadores produzam e manipulem um maior conjunto de dados [Ber08]. Assim, a gestão de dados científicos é um dos maiores desafios da era da informação — um "dilúvio de dados" [HT03] que precisa de ser tratado para garantir a reprodutibilidade dos resultados da investigação, permitindo a colocação de novas questões e aumentando a visibilidade dos investigadores e instituições.

De acordo com Shannon Bohle, "e-Science é a aplicação de tecnologias de computação para a realização de investigação científica moderna, incluindo a preparação, a experimentação, a recolha de dados, divulgação de resultados, armazenamento a longo prazo e acessibilidade a todos os materiais produzidos através do processo científico" [Boh13].

Muitas disciplinas científicas estão cada vez mais dependentes de dados, resultado de uma inundação de dados científicos de uma nova geração de experiências, simulações, sensores e satélites. A investigação científica exige, cada vez mais, um maior grau de colaboração e partilha de

[1] http://aws.amazon.com/
[2] http://www.pdl.cmu.edu/DISC/
[3] http://research.google.com/archive/mapreduce.html
[4] http://hadoop.apache.org
[5] http://research.microsoft.com/en-us/projects/dryad

dados envolvendo sinergias entre equipas de diferentes disciplinas, o que leva a comunidades com muitos investigadores. Esta mudança de paradigma, de trabalhar sozinho com os seus próprios dados, para trabalhar em equipa com dados de várias fontes e, finalmente, a trabalhar em equipas distribuídas e com grandes conjuntos de dados exige um processo de reformulação e modernização de plataformas de suporte à investigação.

1.2 Objetivos e Resultados Esperados

Nos últimos anos têm ganho popularidade soluções de gestão de dados que não seguem a abordagem objeto-relacional tradicional (por exemplo, PostgreSQL[6]) nos casos em que não é necessário manter transações ACID (por exemplo por não haver atualizações de dados) nem é necessária a utilização de SQL para a interrogação às bases de dados onde são guardados esses dados. As bases de dados NoSQL (*Not Only SQL*) diferem da abordagem relacional (tabelas) por usarem estruturas de dados chave-valor (Dynamo — Amazon), coluna (Cassandra — Instagram), grafo (Virtuoso — DBpedia) ou documento (MongoDB — eBay), e estão a ser cada vez mais usadas em aplicações que tratam a chamada *Big Data* e em aplicações em tempo real.

O objetivo deste trabalho consiste na produção de uma plataforma *e-Science*, inovadora para investigadores, focada no armazenamento e disponibilização de toda a meta-informação e informação necessária à correta interpretação dos resultados científicos obtidos, nomeadamente a rastreabilidade e a reprodutibilidade dos dados. Esta plataforma será suportada por duas abordagens distintas: uma com recurso a uma base de dados relacional e uma outra com recurso a uma base de dados NoSQL. Para a sua implementação será estudado com detalhe o impacto que as duas abordagens terão no desempenho da solução, nomeadamente no armazenamento e disponibilização dos dados para os investigadores.

Embora as abordagens NoSQL estejam nos dias de hoje a ser usadas amplamente para o tratamento de *Big Data*, não existe nenhuma plataforma *e-Science* que tire partido deste tipo de soluções para o armazenamento dos objetos de investigação. Tirando partido desta abordagem espera-se que o protótipo a ser apresentado tenha melhor desempenho em toda a gestão de dados científicos que suportam a investigação, comparativamente com as soluções já existentes.

1.3 Estrutura do Documento

Para além da introdução, este documento contém mais sete capítulos, com a seguinte estrutura:

- O Capítulo 2, Gestão de dados científicos, apresenta a definição, explicação e trabalhos relacionados nos campos em que o projeto está inserido. Apresenta também um conjunto de exemplos de soluções existentes para a gestão de dados científicos.

- O Capítulo 3, Bases de dados NoSQL, analisa em detalhe um conjunto de soluções que estão a ser hoje usadas para o armazenamento de grandes quantidades de informação.

[6] http://www.postgresql.org/

- O Capítulo 4, Descrição do Problema e Abordagem, explica detalhadamente em que consiste o problema que se pretende resolver e as abordagens escolhidas.
- O Capítulo 5, Análise de Requisitos e Arquitetura, descreve detalhadamente a fase de conceção da plataforma, que inclui a análise de requisitos e desenho da arquitetura da solução.
- O Capítulo 6, Implementação, analisa as decisões tomadas durante a implementação e é feita uma análise crítica às ferramentas utilizadas.
- O Capítulo 7, Resultados, apresenta a metodologia de testes aplicada, os testes efetuados, os resultados obtidos e a discussão desses mesmos resultados.
- O Capítulo 8, Conclusões, apresenta as conclusões tiradas deste trabalho e o trabalho futuro que poderá vir a ser desenvolvido.

Capítulo 2

Gestão de Dados Científicos

Neste capítulo são analisadas as soluções já existentes para a gestão de dados científicos. Existem atualmente vários projetos que visam a gestão de dados científicos financiados pela União Europeia, América e Austrália. Nas próximas secções são descritos esses projetos e serão apresentadas as suas diferenças.

2.1 Introdução

As infraestruturas de investigação são uma componente essencial do Espaço Europeu de Investigação (ERA[1]) pois reúnem uma grande variedade de intervenientes na procura de soluções para os problemas científicos enfrentados pela sociedade. Estas infraestruturas oferecem oportunidades de pesquisa únicas para investigadores de diferentes países e de diferentes disciplinas, atraem jovens cientistas e ajudam a formar comunidades científicas. Por estas razões desempenham um papel cada vez mais importante no avanço do conhecimento e no desenvolvimento de tecnologia para ajudar a Europa a competir numa economia do conhecimento cada vez mais globalizado [Eur13].

2.2 ESFRI — Fórum Estratégico Europeu para as Infraestruturas de Investigação

O Fórum Estratégico Europeu para as Infraestruturas de Investigação (ESFRI[2]) é um instrumento estratégico europeu para desenvolver a integração científica da Europa e para reforçar o seu alcance internacional. O acesso competitivo e aberto às infraestruturas de investigação de alta qualidade oferece suporte à aferição da qualidade das atividades de cientistas europeus e atrai os melhores investigadores de todo o mundo.

[1] http://ec.europa.eu/research/era/index_en.htm
[2] http://ec.europa.eu/research/infrastructures/index_en.cfm?pg=esfri

A missão do ESFRI é apoiar uma abordagem coerente e estratégica para a elaboração de políticas de infraestruturas de investigação na Europa, e para facilitar iniciativas multilaterais[3] que conduzam a uma melhor utilização e desenvolvimento de infraestruturas de investigação, a nível comunitário e internacional.

Desde que foi formado em 2002, o ESFRI testemunhou avanços significativos no sentido da unidade e impacto internacional no domínio das infraestruturas de investigação. A publicação do primeiro Roteiro para as infraestruturas de investigação europeias, em 2006, e a sua atualização em 2008, foi um fator chave para que vários projetos estejam agora a entrar na fase de execução.

Publicado pela primeira vez em 2006 [Eur06], com 35 projetos e atualizado em 2008 [Eur08], elevando o número de infraestruturas de investigação para 44, este roteiro é um processo em constante atualização. A última atualização com foco em projetos que lidam com energia, alimentos e biologia foi publicado em dezembro de 2010 [Eur11].

2.2.1 Atualização do Roteiro 2016

O processo de atualização do Roteiro 2016[4] foi lançado em Setembro de 2014 e no âmbito desta atualização o ESFRI esteve aberto a novas propostas ou propostas de modernização de infraestruturas de interesse europeu que correspondessem às necessidades a longo prazo das comunidades europeias de investigação, abrangendo todas as áreas de investigação científica.

2.3 HubZero

HubZero[5] é uma plataforma de *software* de código aberto para a construção de *websites* conhecidos por *hubs*, que dão suporte à colaboração científica, investigação e educação. Foi criado por investigadores da Purdue University para apoiar o projeto *nanoHUB.org* [KMB+08], suportando atualmente dezenas de *hubs*. Desde 2007, o uso da plataforma tem-se expandido para suportar um número crescente de *hubs* nas áreas da engenharia farmacêutica (*pharmaHUB.org*), transferência de calor (*thermalHUB.org*), sistemas micro eletromecânicos (*memsHUB.org*), saúde (*IndianaCTSI.Org*), tratamento do cancro (*cceHUB.org*) e engenharia (*globalHUB.org*). Estes *websites* que estão padronizados com a plataforma têm as seguintes funcionalidades:

- Ferramentas interativas de simulação.
- Áreas de desenvolvimento, incluindo controlo de código fonte e acompanhamento de *bugs*.
- Áreas de projeto para colaboração científica.
- Mecanismos para carregamento e partilha de recursos resultantes da investigação.
- Fóruns de suporte a investigadores.

[3] http://www.e-ir.info/2011/01/15/multilateralism-the-ideological-matrix-of-the-european-union/

[4] http://ec.europa.eu/research/infrastructures/index_en.cfm?pg=esfri-roadmap

[5] https://hubzero.org/

- Estatísticas acerca dos utilizadores e padrões de uso.

2.3.1 *Middleware*

A característica mais importante e inovadora desta plataforma é o facto de combinar ferramentas colaborativas com o *middleware* que providencia o acesso a ferramentas interativas de simulação (Figura 2.1). Estas ferramentas podem ser desenhadas por qualquer utilizador para simulação e processamento de dados. À primeira vista estas ferramentas parecem pequenas e simples Java *applets*[6] incorporadas no *browser*, mas na verdade correm num *cluster* de máquinas hospedadas perto do servidor *Web* e são projetadas para o *browser* do utilizador utilizando VNC [RSFWH98]. Cada ferramenta é executada num ambiente virtual restrito implementado usando OpenVZ[7], que controla o acesso ao sistema de ficheiros, rede e outros processos do servidor [MK10].

Estas ferramentas que existem nos *hubs* correm num ambiente gráfico X Windows System[8] que precisa de ser criado, executando em cada recipiente — ambiente virtual — um conjunto de processos que requerem vários segundos para serem iniciados corretamente. Como a sua iniciação é demorada, o *hub* faz um pré-carregamento deste ambiente gráfico em vários recipientes. Assim, quando um utilizador requer uma ferramenta, o *hub* seleciona um recipiente apto, inicia a ferramenta nesse recipiente e envia para o *browser* do utilizador instruções de como se conetar.

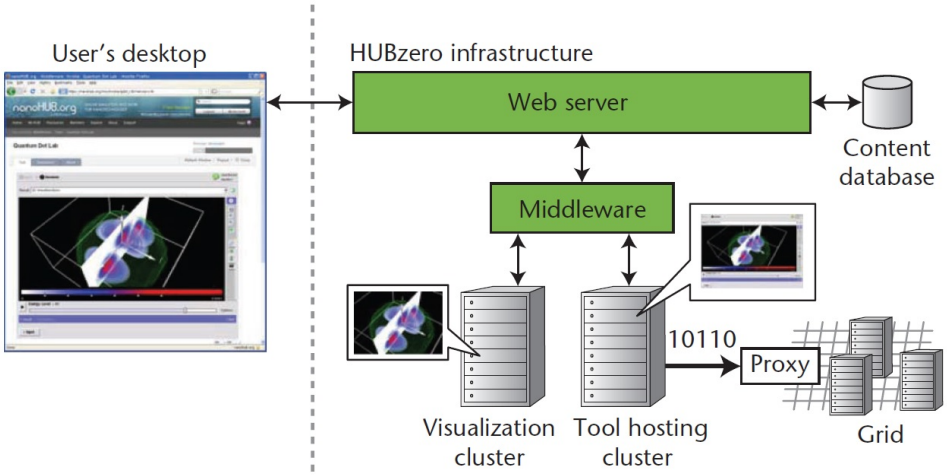

Figura 2.1: A Arquitetura do HubZero [MK10].

2.3.2 Desenvolvimento de Ferramentas

O HubZero fornece aos desenvolvedores uma área completa de trabalho que inclui um repositório para o código desenvolvido e uma *wiki*[9] para as notas do projeto. Cada membro do projeto

[6] http://docs.oracle.com/javase/tutorial/deployment/applet/
[7] http://openvz.org/
[8] http://www.opengroup.org/desktop/x/
[9] http://www.britannica.com/EBchecked/topic/1192819/wiki

tem acesso a esta área de trabalho de modo a promover o trabalho colaborativo. Quando uma ferramenta é concluída, passa à fase de testes e, posteriormente, é publicada. Os desenvolvedores têm a opção de manter o código aberto ou fechado. Após a ferramenta ser publicada, pode ser acedida por todos os utilizadores do *Hub* onde a ferramenta foi desenvolvida [MK10].

2.4 Projeto EUDAT

Nos últimos anos têm sido feitos esforços financeiros por parte da União Europeia e pelos estados membros para que fosse criada uma plataforma que suportasse múltiplas comunidades científicas [Eur10]. Como resultado desta necessidade surgiu em Outubro de 2011 o projeto European Data Infrastructure[10] (EUDAT).

O projeto EUDAT visa abordar os desafios da gestão dos dados científicos e assegurar uma abordagem coerente ao acesso e preservação dos dados de investigação no âmbito da proliferação dos dados produzidos por equipamentos científicos, potenciando assim a visão de uma infraestrutura colaborativa de dados (CDI) (Figura 2.2). Esta infraestrutura tem como objetivo fornecer um conjunto de serviços que são partilhados entre as diferentes comunidades científicas [Han14].

Embora as comunidades científicas de diferentes disciplinas tenham diferentes ambições e abordagens — em especial no que diz respeito à organização e conteúdo dos dados — compartilham também muitas exigências básicas. Estas exigências básicas comuns tornam possível para o projeto EUDAT estabelecer serviços de dados comuns, projetados para suportar múltiplas comunidades de investigação. Os benefícios associados com a criação de um tal quadro colaborativo resulta num melhor aproveitamento dos esforços. Ao fornecer serviços genéricos para as comunidades científicas existentes, o CDI permite que estas comunidades possam concentrar a maior parte do seu esforço e investimento em serviços que são específicos da sua área de investigação. O CDI também fornece aos investigadores individuais e a pequenas comunidades científicas o acesso a estes serviços compartilhados sofisticados, eliminando assim a necessidade de investimento de capital em larga escala no desenvolvimento de uma infraestrutura para esse efeito [LWE+13].

2.4.1 Serviços EUDAT

O projeto EUDAT engloba uma série de serviços essenciais que compõem a infraestrutura colaborativa de dados (Figura 2.3) [EERS15], [ESR15], apresentados a seguir:

- Serviço B2SAFE[11] para replicação segura de dados baseado em *iRODS*[12].
- Serviço de identificadores persistentes (PID) baseado no *Handle System*[13] e *EPIC handle federation*[14].

[10] http://eudat.eu/
[11] http://www.eudat.eu/b2safe
[12] http://irods.org/about/overview/
[13] http://www.handle.net/
[14] http://www.pidconsortium.eu/

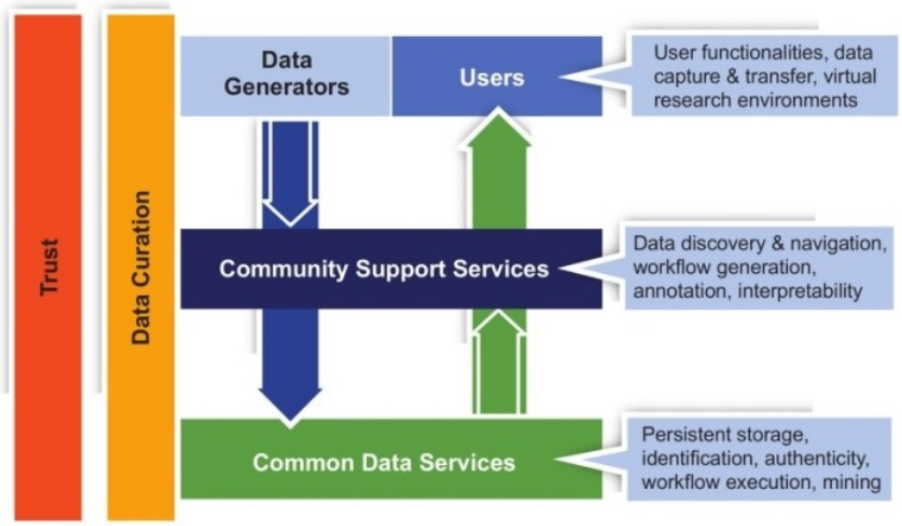

Figura 2.2: Infraestrutura Colaborativa de Dados (CDI) [LWE+13].

- Serviço B2SHARE[15] para armazenar e compartilhar documentos científicos e pequenos conjuntos de dados.

2.4.1.1 B2SAFE — Replicação Segura de Dados

Nos ecossistemas de armazenamento de dados de hoje, as grandes bases de dados devem oferecer um serviço robusto, seguro e altamente disponível para a replicação de dados. Este serviço deve permitir que os repositórios das comunidades científicas possam ser replicados por três principais razões: para se proteger contra perda de dados a longo prazo, ou seja, para garantir a sua preservação, para otimizar o acesso aos dados por parte de utilizadores de diferentes regiões do mundo e, por fim, para trazer os dados para mais perto das máquinas que executam computação intensiva para análise desses dados. O B2SAFE é um serviço que faz parte da infraestrutura colaborativa de dados e foi criado com o intuito de satisfazer os três pontos citados anteriormente.

2.4.1.2 B2STAGE — Processamento de Dados

O objetivo do serviço B2STAGE é oferecer um conjunto de ferramentas confiáveis, eficientes e fáceis de usar que permitam monitorizar não só a transferência de grandes quantidades dados entre as estruturas de armazenamento EUDAT e as áreas de computação de alto desempenho utilizadas para o processamento futuro desses mesmos dados, mas também fornecer meios para reinserir os resultados obtidos nas estruturas de armazenamento EUDAT.

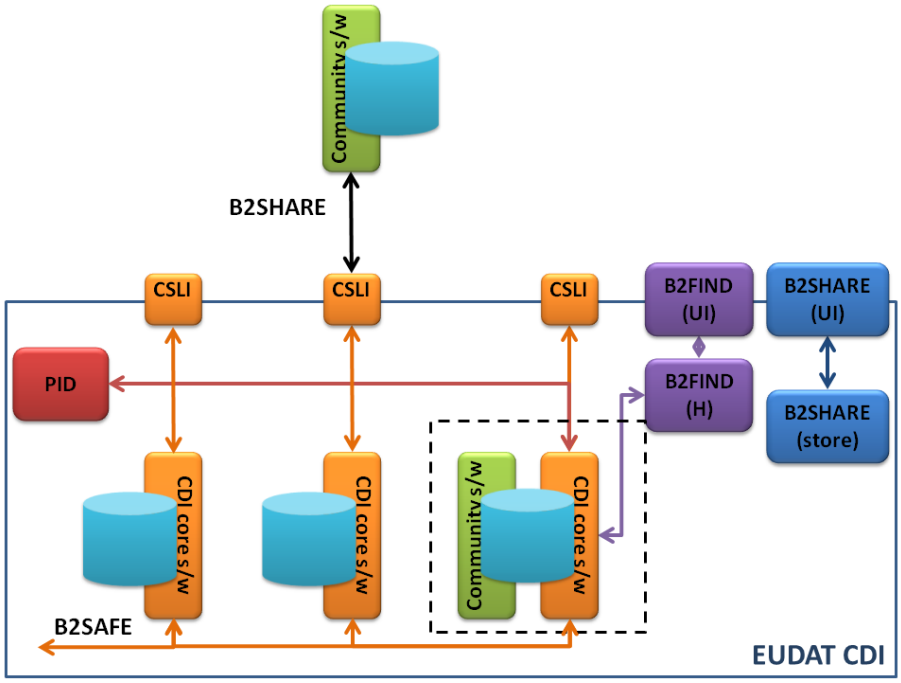

Figura 2.3: Serviços EUDAT [KDBHk14].

2.4.1.3 B2FIND — Metadados

Problemas complexos necessitam cada vez mais de uma abordagem transdisciplinar baseada em dados provenientes de várias áreas de investigação. Neste contexto, garantir que os dados de várias disciplinas estejam disponíveis numa infraestrutura de colaboração pode ser extremamente benéfico. Esta exigência é compartilhada entre as comunidades de investigação, não só para permitir que os seus dados científicos estejam mais visíveis, mas também para tornar possível trabalhar com dados provenientes de outras disciplinas.

O B2FIND é um catálogo de metadados (Figura 2.4) que permite o acesso aos metadados dos dados armazenados no domínio EUDAT. Assim, este serviço atua como um portal que permite aos investigadores encontrar facilmente coleções de dados científicos gerados quer por várias comunidades, quer pelos serviços EUDAT, e aceder a essas coleções de dados através das referências dadas para os repositórios de dados relevantes.

2.4.1.4 B2SHARE — Armazenamento e Partilha de Documentos Científicos

A maioria dos investigadores precisa de uma forma confiável e de fácil utilização para armazenar e compartilhar os seus dados de pesquisa. Os investigadores têm muitas vezes um grande número de arquivos pequenos e, embora a informação desses pequenos conjuntos de dados seja importante, muitas vezes, não pertencem a investigadores de grandes comunidades que seguem estratégias de gestão de dados bem definidas. Este tipo de dados são muitas vezes armazenados

[15] https://b2share.eudat.eu/

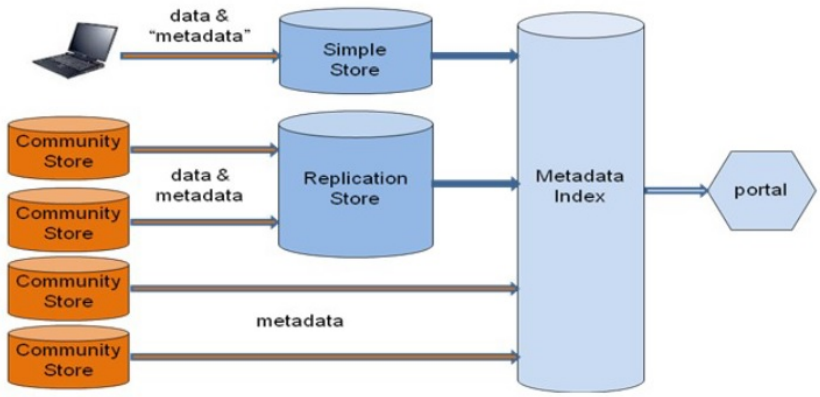

Figura 2.4: Serviço de Metadados [LWE+13].

em máquinas que não se encontram na rede, como computadores portáteis, *desktops* ou servidores locais, arriscando-se assim a perda de dados científicos. Neste contexto, outros investigadores não têm fácil acesso aos dados e o armazenamento é muitas vezes inseguro. De modo a combater estes problemas, o projeto EUDAT criou o serviço B2SHARE.

2.5 ANDS — Australian National Data Service

A investigação científica está a tornar-se cada vez mais intensiva no que toca à produção de dados e os dados cada vez mais complexos. Além disso, os problemas abordados são cada vez mais em larga escala e abrangem várias disciplinas. Por conseguinte, e como resultado de avaliações governamentais, o Departamento de Inovação, Indústria, Ciência, Investigação e Ensino Superior Australiano (DIISRTE) tem investido na melhoria da capacidade do setor de investigação científica da Austrália em usar e reutilizar dados científicos [Dep12].

O Serviço Nacional de Dados Nacional Australiano (ANDS) foi criado em Janeiro de 2009 na sequência da criação do Projeto ANDS. Este projeto foi originalmente criado como parte da iniciativa do departamento de Estratégia Nacional de Infraestruturas Colaborativas para garantir que os dados científicos são usados o mais eficazmente possível pelos investigadores australianos [The07], [Dep12].

Como é referido no plano de negócios de 2012-13, "ANDS existe para revolucionar o ambiente de pesquisa de dados científicos da Austrália, fazendo com que as coleções de dados científicos tenham mais valor" [Dep12]. Assim, este projeto visa garantir melhores dados — melhor descritos, mais ligados, mais integrados e organizados, mais acessíveis, mais facilmente utilizados para novos fins — permitindo que outras novas questões possam ser investigadas [pt15].

2.5.1 Australian Research Data Commons

Para que os objetivos acima citados sejam satisfeitos, o ANDS está a construir o Australian Research Data Commons (ARDC). O termo *commons* refere-se tradicionalmente aos dados científicos que são disponibilizados para o uso da comunidade. Esta abordagem irá possibilitar que as informações sobre os dados científicos australianos estejam no mesmo local e inseridos num contexto adequado [pt10].

Por outras palavras, o Australian Research Data Commons é uma agregação das coleções de dados científicos australianos e das suas descrições, incluindo as informações necessárias para apoiar a sua reutilização, as relações entre os vários elementos envolvidos (os dados, os investigadores que os produziram, os instrumentos que os recolheram e as instituições onde trabalham), e a infraestrutura necessária para permitir e apoiar esses mesmos dados [pt10].

Um facto interessante e diferenciador de todas as outras plataformas já apresentadas é que o ANDS não detém realmente os dados, mas sim o caminho para aceder a esses dados [pt10].

2.6 Projeto *Windscanner.eu*

A unidade *Windscanner*[16] é um conjunto de infraestruturas de investigação que foram inicialmente concebidas pela DTU Wind Energy durante o período 2009–2012 e que serão estabelecidas em toda a Europa após uma fase de preparação entre 2012-2015. Esta unidade consiste num sistema de medição e análise, usando sensores laser, que pode gerar mapas detalhados das condições do vento [Uni13].

Devido à grande quantidade de dados que são recolhidos e à sua importância crescente, foi desenhada e encontra-se em produção uma plataforma *e-Science* que irá suportar todo o processo de fluxo de dados: recolha, processamento, visualização, pesquisa e preservação a longo prazo. Está desenhada para os diferentes grupos de utilizadores, permitindo o acesso aos dados brutos, a visualização dos resultados e o acesso aos resultados finais que podem ser, por exemplo, publicações científicas, gráficos, etc. Com o objetivo de garantir o acesso livre e em qualquer lugar aos dados e metadados produzidos durante as campanhas de medição, a arquitetura contempla três diferentes tipos de nós: nós no local da campanha, nós locais e o nó central [Uni13].

Cada nó no local de campanha, ou seja, cada nó no local onde são feitas um conjunto de medições e onde são recolhidos dados provenientes dos equipamentos, lida com a comunicação de baixo nível entre os vários dispositivos no local da medição. A sua finalidade é, assim, coordenar os dispositivos e extrair os dados brutos. Os nós locais atuam como intermediários entre os nós anteriormente descritos que produzem dados e o nó central que recolhe os dados. Os dados brutos são processados e validados utilizando procedimentos de controlo de qualidade. Os dados da campanha considerados pelo seu proprietário como de acesso livre, são enviados pelo nó local para o nó central [Uni13].

[16] http://www.windscanner.eu/

O nó central contém a plataforma de *e-Science* que garante não só o armazenamento de todos os dados e metadados produzidos durante as campanhas, mas também o armazenamento de outros objetos de investigação produzidos por investigadores, tais como: conjuntos de dados, documentos PDF, rotinas, arquivos, etc. Ao realizar uma simulação, os investigadores podem selecionar os dados brutos ou processados, descarregá-los e enviar mais tarde os resultados do trabalho realizado. Alternativamente, um investigador pode criar um procedimento (em MATLAB[17], Mathematica[18] ou R[19]) a ser aplicado ao conjunto de dados selecionado e receber um identificador para aceder aos resultados (dados derivados) quando estes estiverem prontos [Uni13].

2.7 Conclusões

Após a descrição detalhada nas secções anteriores do projeto EUDAT, ANDS e HubZero é possível concluir que o projeto HubZero é o mais maduro nesta fase, com suporte para a criação de ferramentas que são executadas com recurso à computação em GRID para resultados mais rápidos. Por outro lado, o projeto europeu EUDAT é o mais ambicioso estando a construir uma plataforma colaborativa de dados da qual fazem parte um conjunto de serviços acima descritos que vão para além do que a plataforma HubZero tem para oferecer. Por fim, a plataforma ANDS foca-se, essencialmente, em tornar os dados científicos produzidos rastreáveis, agregando numa plataforma um conjunto de fontes que dão origem a um catálogo de dados, isto é, a plataforma não armazena os dados, apenas aponta o caminho para onde esses dados estão disponíveis.

A plataforma de *e-Science* para o projeto *Windscanner* aposta numa arquitetura centralizada com um nó central que agrega todos os dados e metadados produzidos nos vários locais de campanha. Este tipo de abordagem pode trazer problemas quando falamos em grande quantidades de dados a serem transferidos para o nó central vindas de várias fontes de dados.

[17] http://www.mathworks.com/
[18] http://www.wolfram.com/mathematica/
[19] http://www.r-project.org/

Capítulo 3

Bases de Dados NoSQL

Neste capítulo são analisados os sistemas de base de dados que não seguem o modelo objeto-relacional, as razões do seu aparecimento e o impacto que estão a ter no armazenamento de grandes quantidades de informação.

3.1 Introdução

Os sistemas de gestão de base de dados relacionais (RDMBS) são hoje a tecnologia predominante para o armazenamento de dados estruturados em aplicações *Web*. Desde o artigo de Edgar Frank Codd "Um modelo relacional de dados para grandes bases de dados partilhadas" [Cod70] de 1970, este tipo de armazenamento, que tem por base um modelo relacional, tem sido amplamente adotado e muitas vezes é pensado como a única alternativa para o armazenamento de dados acessível por vários clientes de uma forma consistente. Embora tenha havido diferentes abordagens ao longo dos anos, tais como base de dados orientada a objetos ou base de dados XML [Bou05], essas tecnologias nunca ganharam a mesma aprovação e participação de mercado dos RDBMS.

Nos últimos anos, este tipo de sistema de armazenamento tem sido questionado e discutido tanto por investigadores quanto por empresas, o que levou ao surgimento de uma grande variedade de bases de dados alternativas. O movimento, bem como os novos tipos de armazenamento de dados, são comummente reconhecidos por sistemas de base de dados *Not Only SQL* (NoSQL), "usado para descrever o crescente uso de base de dados não relacionais entre os desenvolvedores da *Web*" [Oba09].

3.2 As Razões do Aparecimento do NoSQL

Durante anos, a fim de melhorar o desempenho nos servidores de bases de dados, os seus administradores tiveram de comprar plataformas mais poderosas à medida que a carga na base de

dados aumentava (escalonamento vertical) em vez de distribuir a base de dados através de múltiplos servidores (escalonamento horizontal). Os RDBMS não costumam escalar facilmente, mas as bases de dados NoSQL foram concebidas para expandir facilmente e tirar proveito dos novos nós geralmente projetados com *hardware* de baixo custo [MAI14]. Assim como as taxas de transferência de dados têm crescido, o volume da informação armazenada tem também crescido em massa. Algumas bases de dados NoSQL fornecem uma taxa de transferência de dados significativamente maior do que os RDBMS tradicionais. Por exemplo, a Google é capaz de processar 20 PB de informação por dia armazenada em Bigtable [CDG+06] via MapReduce [Lai09b].

Apesar das muitas melhorias na gestão dos RDBMS ao longo dos anos, os sistemas RDBMS complexos só podem ser mantidos com a ajuda de gestores de bases dados. Esses gestores estão profundamente envolvidos na conceção, instalação e ajuste contínuo dos sistemas RDBMS. Por outro lado, as bases de dados NoSQL geralmente são projetadas a partir do zero para possibilitar menos manutenção: reparação automática, distribuição de dados e modelos de dados mais simples [Lai09a].

A maioria das bases de dados NoSQL são concebidas para armazenar estruturas de dados que são simples ou mais semelhantes às de linguagens de programação orientada a objetos, em comparação com as estruturas de dados relacionais. Isto é particularmente importante para aplicações com estruturas de dados de baixa complexidade que dificilmente poderão beneficiar com as características de uma base de dados relacional.

3.3 Classificação das Bases de Dados NoSQL

Nos últimos anos têm sido desenvolvidas uma variedade de base de dados NoSQL, principalmente por empresas, para atender às suas necessidades específicas de desempenho, escalabilidade e manutenção.

Por causa desta variedade de abordagens e sobreposições em relação aos requisitos não funcionais e conjunto de funcionalidades, têm surgido várias formas para classificar as bases de dados NoSQL, usando diferentes categorias. Estas diferentes categorias serão explicadas nos pontos seguintes.

3.3.1 Modelo de Dados

A principal diferença entre as bases de dados NoSQL e as bases de dados relacionais é o modelo de dados. As bases de dados NoSQLclassificam-se numa das quatro categorias seguintes: documento, grafo, chave-valor ou coluna [Sad14].

3.3.1.1 Modelo em Documento

Enquanto que as bases de dados relacionais armazenam os dados em linhas e colunas, as bases de dados que seguem o modelo documento armazenam os dados em documentos (Figura 3.1). Estes documentos usam normalmente uma estrutura semelhante a JSON [Sad14]. Os documentos fornecem uma forma intuitiva e natural de modelar os dados que está alinhada com a programação orientada a objetos, pois cada documento é efetivamente um objeto. Estes documentos podem conter um ou mais campos, onde cada campo contém um valor que pode ser um conjunto de carateres, uma data, um binário ou um *array*. Uma das vantagens sobre as bases de dados relacionais é que, ao invés de se espalhar uma entrada por várias colunas e tabelas, cada registo e os seus dados associados são normalmente armazenados juntos num único documento, simplificando o acesso aos dados e a redução ou até eliminação da necessidade de transações complexas [Mon14].

Numa base de dados de documentos, a noção de estrutura é dinâmica, isto é, cada documento pode conter campos diferentes. Essa flexibilidade pode ser particularmente útil para a modelação de dados não estruturados e polimórficos. Além disso, as bases de dados de documentos oferecem a robustez na consulta dos dados que tem vindo a ser procurada nas bases de dados relacionais, isto é, os dados podem ser procurados com base em qualquer um dos campos de um documento.

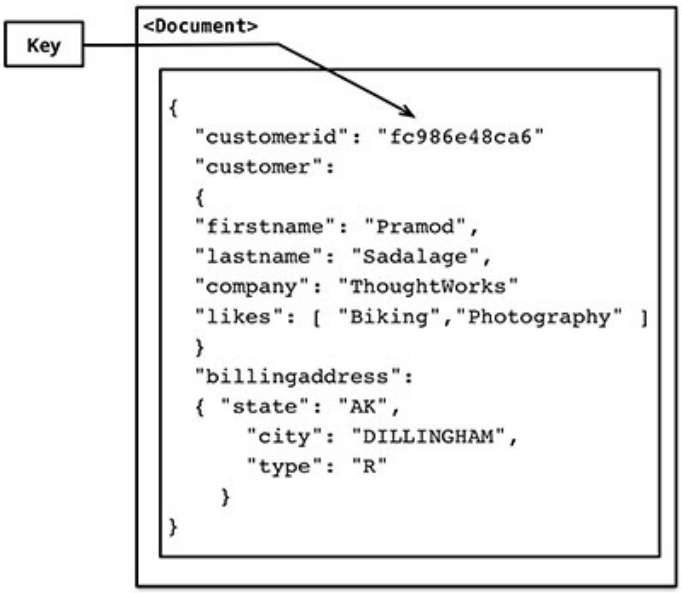

Figura 3.1: Base de Dados em Documento [Sad14].

MongoDB

O MongoDB é uma base de dados gratuita escrita em C++, desenvolvida em código aberto e que segue o modelo acima descrito. De acordo com os seus desenvolvedores, o principal objetivo do MongoDB é fechar a lacuna entre as bases de dados chave-valor, conhecidas pela sua rapidez e escalabilidade, e os tradicionais sistemas de gestão de bases de dados, que seguem o modelo relacional e têm ao seu dispor um conjunto vasto de funcionalidades.

A abstração e unidade de dados usada no MongoDB é o documento, uma estrutura de dados comparável a um documento XML ou a um documento JSON. De facto, os documentos MongoDB persistem na base de dados num formato chamado BJSON que é muito semelhante ao JSON, mas numa representação binária, por razões de eficiência e para conseguir tipos de dados adicionais. O MongoDB oferece os seguintes tipos de dados para os campos do documento:

- tipos básicos: *boolean*, *integer*, *double*.
- tipos de sequência de caracteres: *string* (sequências de caracteres codificados em UTF-8), expressão regular, JavaScript.
- objeto (objetos BJSON).
- *object_id* é um tipo de dados de 12 B usado pelo o MongoDB para um campo denominado *_id* que identifica documentos dentro de coleções. O tipo de dados *object_id* é composto pelos seguintes componentes:
  - *timestamp* em segundos desde *epoch* (primeiros 4 B).
  - *Id* da máquina que atribui o valor do *_id* do objeto (próximos 3 B).
  - *Id* do processo MongoDB (próximos 2 B).
  - Contador (últimos 3 B).
- *Null*.
- *Array*.
- Date.

3.3.1.2 Modelo em Grafo

As Bases de dados em grafo usam como estrutura grafos com nós, arestas e propriedades para representar dados. Na prática, os dados são modelados como uma rede de relações entre os elementos. Este modelo possibilita o armazenamento de nós, também designados entidades, e relações entre entidades [Mon14]. Estas entidades possuem propriedades (por exemplo nome, idade) que as descrevem e as tornam únicas, tal como acontece na programação orientada a objetos, em que os objetos de cada classe possuem um conjunto de características que os identificam.

As relações entre as entidades são representadas pelas arestas do grafo que podem também conter propriedades. Estas relações são a base deste modelo porque a maior parte dos dados consultados são derivados a partir destas relações.

Embora a adição de nós e relações seja simples, a sua modificação não o é pois as alterações efetuadas têm que ser aplicadas a todos os nós e a todas as relações nos dados existentes. Por isso, é necessária uma especificação bem estruturada para modelar as relações no domínio de forma a minimizar a necessidade de alterações nas relações existentes [Sad14].

A organização do grafo permite que os dados sejam armazenados uma vez e, em seguida, interpretados de maneiras diferentes com base nas relações entre as entidades. A travessia entre relações é muito rápida pois não é necessário calcular as relações no momento da interrogação à base de dados uma vez que estas persistem na base dados. Isto difere das bases de dados relacionais, uma vez que nesse modelo as combinações entre uma ou mais tabelas baseadas num campo comum entre elas (*join*) acontecem no momento da consulta. O Neo4j[1] é um exemplo de um sistema de base de dados que segue este modelo.

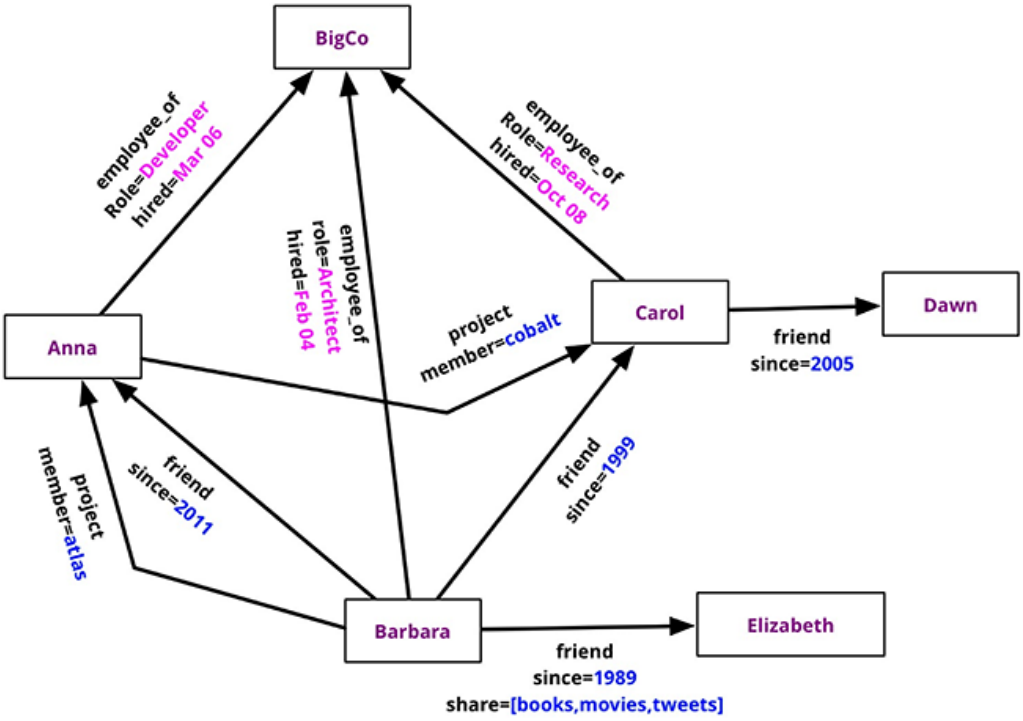

Figura 3.2: Base de Dados em Grafo [Sad14].

3.3.1.3 Modelo em Chave-valor

Do ponto de vista do modelo de dados, base de dados de chave-valor são o tipo mais básico de bases de dados NoSQL [WS10]. Cada item é armazenado como uma chave juntamente com o seu valor. O valor, no entanto, é totalmente opaco ao sistema, isto é, os dados só podem ser interrogados pela chave. Este tipo de solução é altamente escalável uma vez que o acesso aos dados é feito sempre pela chave.

Amazon Dynamo

O Amazon Dynamo é um dos muitos sistemas de bases de dados usados pela Amazon. A Amazon construiu o Dynamo com uma interface chave-valor simples que armazena os valores como BLOB.

[1] http://neo4j.com/

As operações estão limitadas a um par chave-valor de cada vez, de modo que as operações de atualização estão reduzidas a chaves individuais não permitindo referências cruzadas para outros pares chave-valor ou operações que abranjam mais de um par chave-valor. A interface do Dynamo tem como funcionalidades:

- *get(chave)*, retornando a lista de objetos e o contexto.
- *put(chave, contexto, objeto)*, não sendo retornado nada.

A operação *get()* pode retornar mais do que um objeto se houver conflitos de versão para objetos armazenados na chave dada. É também retornado um contexto, em que os metadados do sistema, tais como a versão de objeto, são armazenados. Na operação *put()* os clientes têm de fornecer esse contexto como um parâmetro para além de chave e do objeto. Os valores das chaves e objetos não são interpretados pelo Dynamo, mas tratado como um *array* de bytes opaco. A chave é usada pelo algoritmo MD5 para determinar os nós de armazenamento responsáveis por este par chave-valor.

3.3.1.4 Modelo Orientado à Coluna

Bases de dados orientadas à coluna usam um mapa ordenado multidimensional distribuído[2] para armazenar dados. Este tipo de base de dados armazena os dados em famílias de colunas. Famílias de colunas são grupos de dados relacionados que são frequentemente acedidos juntos. Cada família pode ser comparada a um conjunto de linhas de uma tabela de uma base de dados relacional onde a chave identifica a linha que é formada por várias colunas (Figura 3.3). A grande diferença em relação ao Modelo Relacional é que as várias linhas não têm que ter o mesmo número ou tipo de coluna, e estas podem ser adicionadas a qualquer linha em qualquer momento, sem a obrigatoriedade de as adicionar às outras linhas.

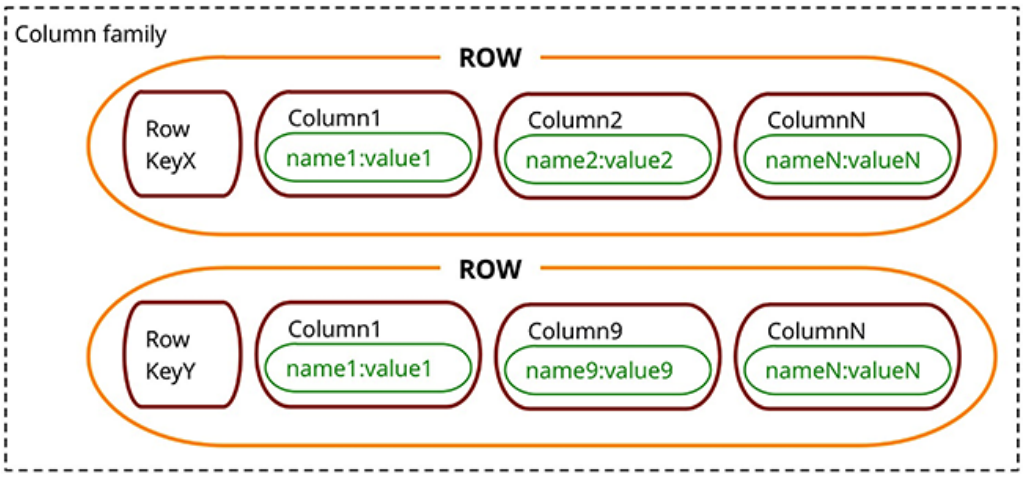

Figura 3.3: Base de Dados Orientado à Coluna [Sad14].

[2] http://jimbojw.com/wiki/index.php?title=Understanding_Hbase_and_BigTable

No caso de uma coluna conter outras colunas, essa coluna é designada de super-coluna.

Apache Cassandra
O Apache Cassandra foi originalmente desenvolvido pelo Facebook e tornado num projeto em código aberto em 2008. Cassandra é um sistema de armazenamento distribuído para gerir dados estruturados que está desenhado para ser altamente escalável. Partilha estratégias de implementação e *design* das bases de dados tradicionais, mas não suporta um modelo relacional completo. Em vez disso, disponibiliza um modelo de dados simples que suporta controlo dinâmico sobre o formato e *layout* dos dados. Uma instância de Cassandra consiste tipicamente numa tabela que representa um mapa distribuído multidimensional indexado por uma chave. Uma tabela é estruturada pelas seguintes dimensões:

- linhas, que são identificadas por uma chave de comprimento arbitrário. Operações em linhas são atómicas, não importando quantas colunas estão a ser lidas ou escritas.
- Família de colunas, que podem ocorrer num número arbitrário por linha.
- Colunas, que têm um nome associado e armazenam um número indeterminado de valores por linha, que são identificados por um *timestamp*. Cada linha numa tabela pode ter um número diferente de colunas, podendo a tabela não ser um retângulo.
- Super-colunas, que têm um nome e um conjunto arbitrário de colunas que lhes estão associadas. Assim, os valores em Cassandra são acedidos pelo triplo (chave da linha, chave da coluna, *timestamp*) em que chave da coluna é "coluna-família:coluna" para colunas simples contidas na família da coluna ou "coluna-família:supercoluna:coluna" para colunas inseridas numa super-coluna.

3.4 Consistência

A maior parte das bases de dados NoSQL oferecem consistência eventual, um tipo fraco de consistência. A consistência eventual é um modelo de consistência usado em base de dados distribuídas para alcançar alta disponibilidade. Este modelo de consistência não garante que cada processo que acede a um registo vê a mesma versão do mesmo, isto é, não é garantido que uma leitura retorne sempre a versão do registo mais recente. Quando há uma atualização a um registo apenas é garantido que, eventualmente, todos os processos irão ver essa atualização.

3.4.1 Teorema de CAP

O teorema de *Consistency, Availability, and Partition tolerance* (CAP) foi proposto por Eric Brewer no simpósio Princípios dos Sistemas Distribuídos da ACM[3] em Julho de 2000, onde defendeu que não é possível para um sistema distribuído fornecer estes três requisitos em simultâneo [Bre00]:

[3] http://acm.org/

- **Consistência**, isto é, se o sistema fica num estado consistente após uma operação. Um sistema distribuído é normalmente considerado consistente se, depois de uma operação de escrita, os dados ficam imediatamente disponíveis para serem utilizados. Por outras palavras, uma leitura retorna sempre a escrita mais recente.

- **Disponibilidade**, isto é, todos os pedidos recebem uma resposta quer seja de sucesso ou de falha. Garantir 100% de disponibilidade num sistema significa que o sistema está disponível em qualquer instante de tempo.

- **Tolerância a Falhas**, isto é, a capacidade do sistema continuar a funcionar, apesar da perda de mensagens ou falha parcial [GL02]. Por exemplo, numa rede com múltiplas sub-redes onde os nós A e B estão numa sub-rede e os nós C e D estão noutra, uma partição de rede ocorre se o *switch* entre as 2 sub-redes falhar. Neste caso os nós A e B não conseguirão comunicar com os nós C e D, o que significa que o sistema não é tolerante a falhas.

Doze anos depois, Eric Brewer num artigo intitulado "*CAP Twelve Years Later: How the 'Rules' Have Changed*" explica que "a forma mais fácil de entender o CAP é pensar em dois nós em lados opostos de uma partição. Permitir que pelo menos um nó atualize o seu estado fará com que os nós se tornem inconsistentes, perdendo, assim, o C. Da mesma forma, se a escolha é a de preservar a consistência, um lado da partição deve agir como se não estivesse disponível, perdendo, assim, o A. Somente quando os dois nós comunicam é possível preservar tanto a consistência como a disponibilidade, perdendo assim o P" [Bre12]. Sobre o impacto do teorema sobre os modelos de base dados, Eric Brewer refere que "a crença geral é que para sistemas de grande escala, os *designers* não podem perder P e, portanto, têm uma escolha difícil entre C e A" concluindo que "o movimento NoSQL tem a ver com a criação de soluções que se focam na disponibilidade em primeiro lugar e na consistência em segundo; bases de dados que seguem as propriedades ACID (atomicidade, consistência, isolamento e durabilidade) fazem o oposto" [Bre12].

No entanto, Eric Brewer revela que a crença geral pela qual o movimento NoSQL se rege é enganosa "em primeiro lugar, porque as partições são raras, não há motivo para perder C ou A quando o sistema não está particionado. Em segundo lugar, a escolha entre C e A pode ocorrer muitas vezes dentro do mesmo sistema; não só os subsistemas podem fazer escolhas diferentes, mas também a escolha pode mudar de acordo com a operação ou até mesmo com dados específicos ou utilizadores envolvidos. Finalmente, todas as três propriedades são contínuas. Disponibilidade é obviamente contínua de 0 a 100 por cento, mas também há muitos níveis de consistência, e até mesmo as partições têm nuances, incluindo desacordo dentro do sistema sobre se existe ou não uma partição" [Bre12].

3.4.2 ACID vs BASE

O modelo ACID (atomicidade, consistência, isolamento e durabilidade) foi criado em 1983 [HR83], mas as suas propriedades haviam sido propostas por Jim Gray em 1981 [Gra81]. Este modelo define que cada transação nas bases de dados deve ser:

- **Atómica**, isto é, deve ser ou totalmente sucedida ou toda a transação falhar e, nesse caso, a base dados é deixada no mesmo estado em que se encontrava antes da operação ter executado.
- **Consistente**, isto é, a transação não pode deixar a base de dados num estado inconsistente. Se a transação que é feita viola regras de consistência da base de dados, essa transação é revertida e a base de dados é deixada no último estado consistente conhecido.
- **Isolada**, isto é, a transação não pode interferir com outras transações. O resultado de duas ou mais transações que executem simultaneamente deve ser o mesmo que o resultado de executar as transações em série.
- **Durável**, isto é, a transação uma vez sucedida deve persistir e não ser revertida, mesmo em caso de falha posterior do *hardware*. Esta propriedade não garante que os dados armazenados sejam permanentes, uma vez que estes podem ser alterados por uma transação diferente sem perder a propriedade de durabilidade.

Para um número crescente de aplicações e casos de uso incluindo aplicações *Web*, especialmente em grande escala, disponibilidade e tolerância a falhas são mais importantes que a consistência. Este facto é facilmente entendido com o exemplo da Amazon. Se toda a vez que alguém está no processo de compra de um livro, a Amazon bloqueasse parte da base de dados até que a compra fosse terminada a fim de garantir que todos os visitantes do mundo vissem o número exato de livros em *stock*, o sistema sofreria de baixa disponibilidade, o que impediria os seus clientes de realizar as suas compras. Essas aplicações têm de ser confiáveis o que implica disponibilidade e redundância e a consequentemente distribuição entre dois ou mais nós. Como estas propriedades são difíceis de alcançar com as propriedades ACID, abordagens como BASE (*Basically Available, Soft state, Eventually consistent*) são aplicadas.

A abordagem BASE, de acordo com Brewer, perde as propriedades ACID de consistência estrita e de isolamento em favor da "disponibilidade e desempenho" [Bre00]. A sigla BASE é um acrónimo para as seguintes propriedades:

- **Basicamente Disponível**, isto é, mesmo que alguns dos componentes falhe, o sistema subjacente permanece funcional e com capacidade de resposta. No entanto, os dados retornados podem não ser consistentes ou os mais recentes.
- **Estado leve**, isto é, o estado da base de dados pode mudar ao longo do tempo. Mesmo sem qualquer entrada de dados, a base de dados pode mudar ao longo do tempo devido às operações internas para alcançar consistência.
- **Eventualmente Consistente**[4], isto é, a base de dados irá, mais tarde ou mais cedo, tornar-se consistente a partir do momento em que deixa de receber dados. Os dados recebidos serão propagados para todos os nós, mas o sistema permanece disponível para receber novas entradas, não verificando a consistência de cada transação antes de processar a próxima.

[4] tradução de *eventually consistent*.

3.5 Resumo

A Tabela 3.1 apresenta um resumo das propriedades dos dois acrónimos. A grande diferença entre estes dois conjuntos de propriedades é que, as propriedades BASE, caraterísticas das bases de dados NoSQL, abdicam da consistência estrita a favor da alta disponibilidade dos dados.

Tabela 3.1: ACID vs BASE [Bre00].

ACID	**BASE**
Consistência forte	Consistência fraca
Isolamento	Disponibilidade primeiro
Conservativa (pessimista)	Agressiva (otimista)
Dificuldade de evolução	Rápida, fácil evolução

Capítulo 4

Descrição do Problema e Abordagem

Através do estudo do estado de arte no Capítulo 3 foi possível concluir que um dos grandes desafios da era da informação é armazenar e processar todos os dados que são produzidos, tendo surgido nos últimos anos um conjunto de soluções para armazenamento de grandes quantidades de dados de forma escalável, eficiente e de baixo custo.

Neste capítulo é descrito o problema que se pretende resolver e são apresentadas as abordagens seguidas para a sua resolução.

4.1 Problema

No decurso de uma investigação científica é produzida, por vários sensores e equipamentos, uma enorme quantidade de dados que precisam de ser armazenados e geridos para, posteriormente, poderem ser consultados e analisados. No domínio de aplicação das Ciências da Terra, nomeadamente na utilização de sensores laser (*LiDAR*) para a análise das condições de vento em instalações de torres eólicas de produção de energia elétrica, são produzidas séries temporais. Devido à enorme quantidade de informação que é produzida por estes sensores, torna-se ineficiente a utilização de base de dados relacionais para o armazenamento das séries temporais produzidas devido, essencialmente, aos problemas inerentes de escalabilidade, descritos no Capítulo 3, que o modelo relacional apresenta quando é necessário gerir um número elevado de tuplos, na ordem dos milhares de milhões.

Pela análise da Figura 4.1, que representa o desempenho na consulta de dados numa tabela de um sistema de base de dados relacional, neste caso PostgreSQL, é possível concluir que esta solução apresenta problemas de escalabilidade. Quando a tabela contém poucas linhas, o tempo de consulta dos dados é relativamente reduzido, sendo os resultados apresentados em poucos segundos. No entanto, o tempo de execução aumenta rapidamente à medida que o número de entradas na tabela aumenta. Aplicando o procedimento a uma tabela com mil milhões de entradas, o tempo de consulta é superior a 24 h, o que sugere que o PostgreSQL não escala eficientemente para um grande conjunto de dados, devido, presumivelmente, a uma má escolha do algoritmo de consulta de dados [Jac09]. Enquanto que o *query planner* da base dados escolhe uma estratégia de

agregação com base numa *hash table* para pequenas tabelas, em tabelas maiores a abordagem é alterada para uma ordenação por agrupamento das colunas, que é viável quando se lida com alguns milhões de linhas, mas muito pouco eficiente quando se lida com mil milhões de linhas. A dificuldade do PostgreSQL não é em armazenar os dados, mas sim em devolvê-los quando estamos perante grandes quantidades de informação.

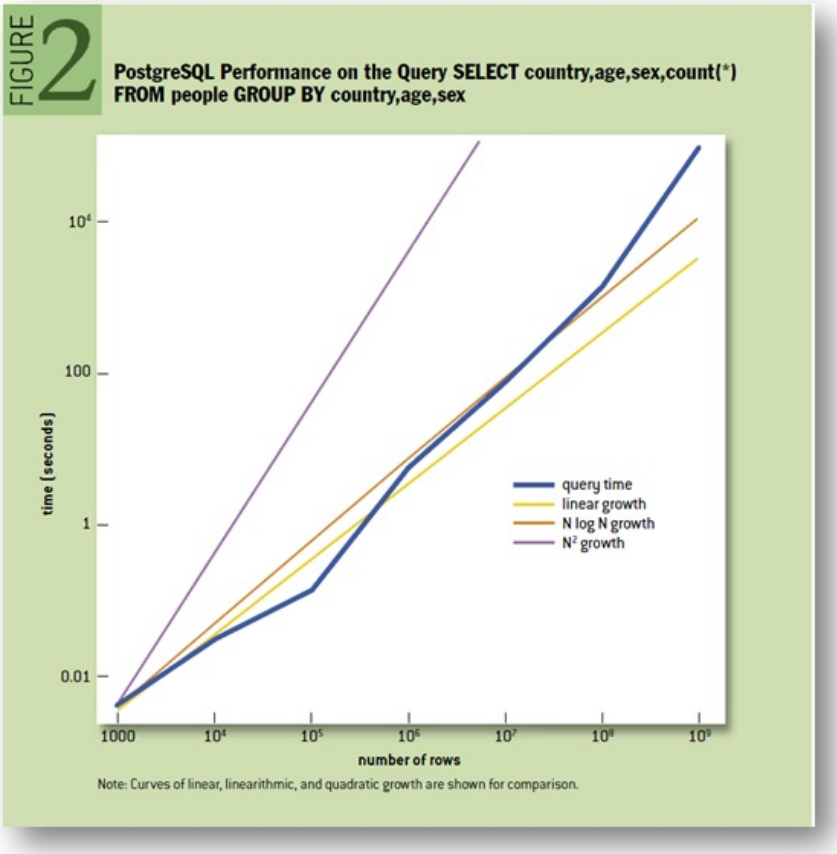

Figura 4.1: Desempenho de um Sistema de Base de Dados SQL [Jac09].

Os sistemas de bases de dados tradicionais foram desenhados para o processamento eficiente de transações tais como: inserir, atualizar, procurar e retornar pequenas quantidades de informação num grande conjunto de dados. O problema surge quando queremos consultar uma grande quantidade de dados acumulados ao longo de meses ou anos. Para além disso, os sistemas de base dados relacionais seguem um esquema rígido de dados tornando custosa a sua alteração. Como as séries temporais a armazenar são produzidas por diferentes sensores e equipamentos, a sua estrutura não é uniforme, implicando a criação de uma tabela para cada tipo de série temporal e a consequente alteração do esquema de dados, ou a criação de um *array* de tamanho variável para armazenar qualquer tipo de série temporal, implicando a criação de uma tabela adicional que descreva o que cada posição do *array* representa.

Apesar de terem surgido soluções para a gestão de dados científicos, estas implicam o uso de API compreensivelmente limitadas a um conjunto de funções mais generalizadas para cobrir o maior número de áreas científicas possíveis e focadas no armazenamento e preservação dos dados, como é o caso do projeto EUDAT.

4.2 Abordagem

Como descrito na Secção 1.2, o objetivo deste trabalho consiste na criação de um protótipo de uma plataforma *e-Science*, focado no desempenho e escalabilidade, que seja capaz de resolver os problemas descritos anteriormente. Como uma plataforma de *e-Science* engloba um conjunto de conceitos e funcionalidades que vão muito além do problema descrito — preparação, experimentação, recolha de dados, divulgação de resultados, armazenamento a longo prazo, acessibilidade a todos os materiais produzidos através do processo científico, etc. — será impossível implementar todas estas funcionalidades e, por isso, o foco deste trabalho será no desempenho e escalabilidade da solução em relação ao armazenamento, descarregamento e visualização das séries temporais produzidas por equipamentos e sensores.

A plataforma *e-Science* segue duas abordagens distintas para a gestão dos dados armazenados: a primeira faz uso de uma base de dados relacional e a segunda faz uso de uma base de dados NoSQL. A ideia passa por estudar o impacto que as duas abordagens têm na performance da plataforma e, ao mesmo tempo, avaliar a escalabilidade de cada uma.

4.3 Resumo e Conclusões

Durante este capítulo foi identificado e descrito o problema a que se pretende responder com este trabalho. A identificação do problema de escalabilidade inerente aos sistemas de base de dados relacionais levou a que um dos principais objetivos deste trabalho, além do desenvolvimento de uma plataforma *e-Science*, fosse estudar o impacto que um sistema de base de dados relacional teria no desempenho da plataforma. Para que fosse possível realizar esse estudo, foi necessário abordar o problema recorrendo a duas abordagens distintas: uma fazendo uso de uma base de dados relacional e outra com recurso a uma base de dados NoSQL. O próximo capítulo apresenta os cenários de utilização da plataforma e arquitetura que foi desenhada para a implementação das duas abordagens.

Capítulo 5

Análise de Requisitos e Arquitetura

Este capítulo detalha a primeira fase do trabalho que corresponde à conceção da plataforma *e-Science* desenvolvida. As secções seguintes fornecem uma análise e descrição dos requisitos do sistema, cenários de utilização e atores envolvidos bem como a arquitetura do sistema.

5.1 Cenário de Utilização

Um cenário de utilização "é uma descrição de uma potencial situação de negócio que pode ser encontrada pelos utilizadores de um sistema — o foco está em requisitos funcionais e não em questões de desenho da arquitetura do sistema" [Amb04]. Os atores (humanos ou máquinas) são identificados e os cenários de uso onde eles participam são descritos usando narrativas de utilização.

5.1.1 Atores

A Figura 5.1 representa os atores que interagem na plataforma:

- O ator ***Public*** tem acesso aos dados publicados na plataforma sendo-lhe permitido procurar e descarregar recursos que estejam marcados como públicos.
- O ator ***Visitor*** é qualquer utilizador não autenticado. Pode fazer o *login* na plataforma ou registar-se para ter acesso a outras funcionalidades.
- O ator ***Researcher***, representa um investigador no sistema que é capaz de aceder a dados e metadados e gerir os seus objetos de investigação.
- O ator ***Data Provider*** interage com o sistema para fazer o carregamento de dados e metadados relacionados com campanhas. Os fornecedores de dados são, geralmente, máquinas que utilizam serviços fornecidos pela plataforma de *e-Science* para realizar as operações referidas.
- O ator ***Campaign Manager*** é responsável por descrever as suas campanhas e realizar todas as tarefas relacionadas com validação de dados.

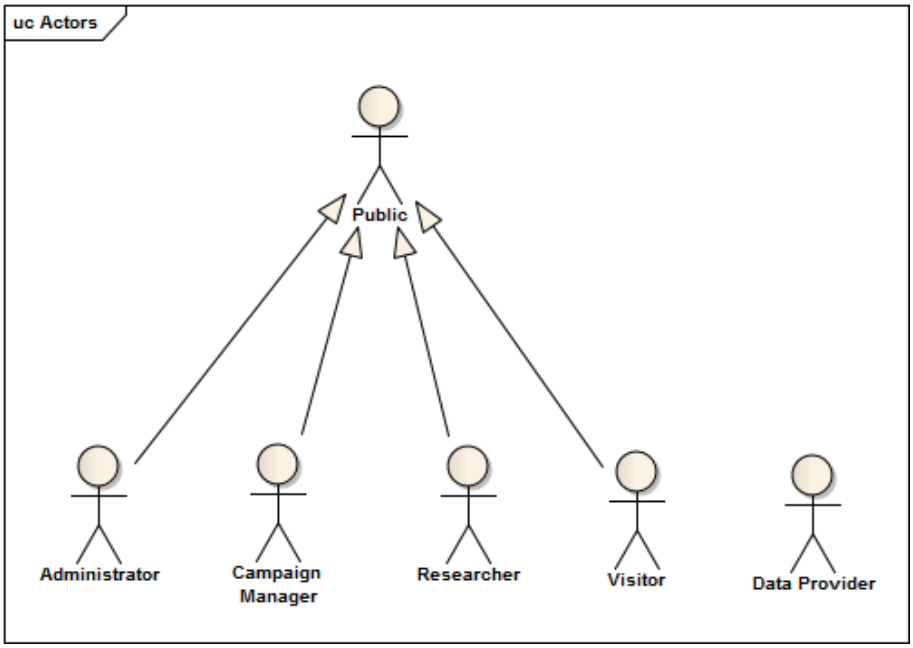

Figura 5.1: Atores que interagem com a Plataforma.

- O ator ***Administrator*** gere as contas dos utilizadores e concede permissões para que os ***Data Providers*** e ***Researchers*** possam desempenhar as suas funções. Diferentes ***Researchers*** podem ter diferentes privilégios de leitura ou de escrita em determinados recursos, de acordo com as políticas de acesso a serem definidas.

5.1.2 Narrativas de Utilização

Na Tabela 5.1 são enunciados os possíveis cenários de utilização seguindo o paradigma das narrativas de utilização[1]. A cada narrativa de utilização foi atribuída uma prioridade (alta, média ou baixa) sendo que as classificadas com prioridade alta foram o foco durante a implementação. As classificadas com prioridade baixa representam funcionalidades extra que poderão ser implementadas em trabalho futuro.

Tabela 5.1: Narrativas de Utilização

Identificador	**Nome**	**Descrição**	**Prio.**
US01	*Register*	Como*Visitor* quero registar uma nova conta de modo a que tenha acesso à informação de perfil.	Baixa
US02	*Login*	Como *Visitor* quero autenticar-me de modo a que tenha acesso a informação reservada.	Média

Continua na próxima página...

[1]Narrativas de utilização (*user stories*) são escritas de acordo com o seguinte modelo: Como <tipo de utilizador> pretendo <objetivo> de modo que <mais-valia>.

Tabela 5.1 – continuação da página anterior

Identificador	**Nome**	**Descrição**	**Prio.**
US11	*Create campaign*	Como *Campaign Manager* quero criar uma nova campanha de modo a que um *Researcher* possa carregar dados relativos a essa campanha.	Alta
US12	*Create Site*	Como *Campaign Manager* quero criar um novo lugar de campanha de modo a que o possa associar a uma campanha.	Alta
US13	*Create Equipment*	Como *Campaign Manager* quero criar um novo equipamento de modo a que o possa associar a um ou mais *datasets* produzidos.	Alta
US14	*Create Template*	Como *Campaign Manager* quero criar um novo *template* de modo a que seja possível associá-lo a um dataset.	Alta
US15	*Create Station*	Como *Campaign Manager* quero criar uma nova estação de modo a que possa associá-la a um *dataset* e equipamento num determinado período de tempo.	Alta
US16	*Create scenario*	Como *Campaign Manager* quero criar um novo cenário de modo a que o *Researcher* ou *Data Provider* possam carregar recursos relativos a esse cenário.	Alta
US17	*Create dataset*	Como *Campaign Manager* quero criar um novo *dataset* de modo a que o *Data Provider* possa carregar dados relativos a esse *dataset*.	Alta
US18	*Set access permissions*	Como *Campaign Manager* quero definir as permissões de acesso às campanhas de modo a que utilizadores indicados possam aceder às campanhas.	Baixa
US19	*Publish dataset*	Como *Campaign Manager* quero validar e publicar *datasets* e metadados associados de modo a que os *Researchers* possam aceder aos *datasets*.	Baixa
US21	*Upload raw data*	Como *Data Provider* quero carregar dados e metadados relativos a *datasets* de modo a que a plataforma possa armazená-los.	Alta
US31	*Search*	Como *Researcher* quero filtrar os *datasets* existentes por data e outros atributos de modo a que possa facilmente encontrar os valores que procuro.	Alta
US32	*Upload resource*	Como *Researcher* quero carregar informação relativa a uma campanha ou cenário (relatórios, documentos descritivos, outros documentos) de modo a que a plataforma possa armazená-los.	Baixa
US33	*Publish resource*	Como *Researcher* quero publicar um novo documento de modo a que outros investigadores possam aceder a esse novo documento.	Baixa
US34	*Download dataset*	Como *Researcher* quero descarregar um *dataset* em formato HDF5 de modo a que possa analisá-lo.	Média

Continua na próxima página...

Tabela 5.1 – continuação da página anterior

Identificador	Nome	Descrição	Prio.
US35	*See dataset*	Como *Researcher* quero visualizar um *dataset* de modo a que possa analisá-lo.	Média
US41	*Accounts management*	Como *Administrator* quero criar novas contas e garantir permissões de acesso de modo a que *Researcher* e *Data Provider* de modo a que estes possam desempenhar os seus papeis.	Baixa
US42	*Monitor hub*	Como *Administrator* quero monitorizar o estado do sistema de modo que a plataforma esteja operacional.	Média

5.1.2.1 Casos de Uso

Para melhor perceção daquilo que precisa de ser implementado, desdobraram-se as narrativas de utilização em pacotes de casos de uso: Autenticação (Figura 5.2), Campanha & Cenário (Figura 5.3) e Dataset (Figura 5.4). Aconselha-se, também, a visualização do Anexo B onde é apresentada uma série de *screenshots* que exemplifica alguns casos de utilização.

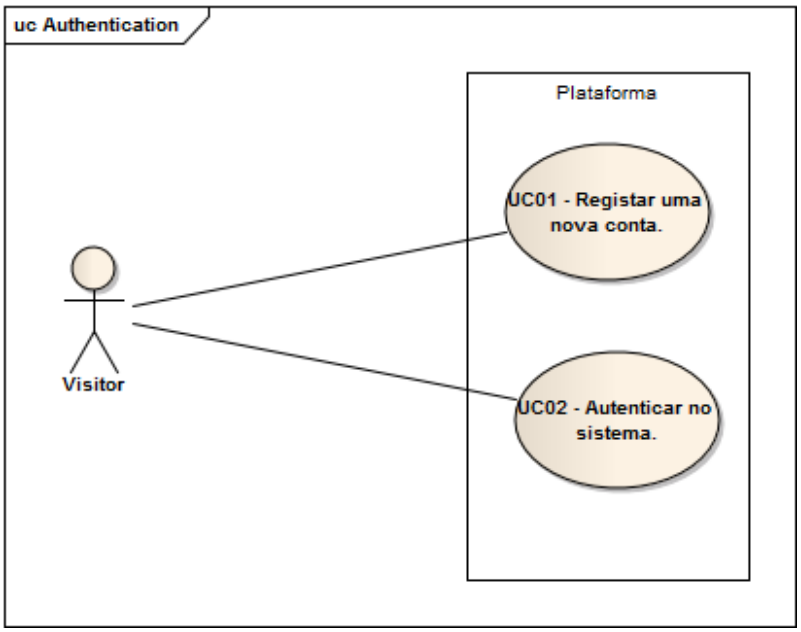

Figura 5.2: Pacote Autenticação.

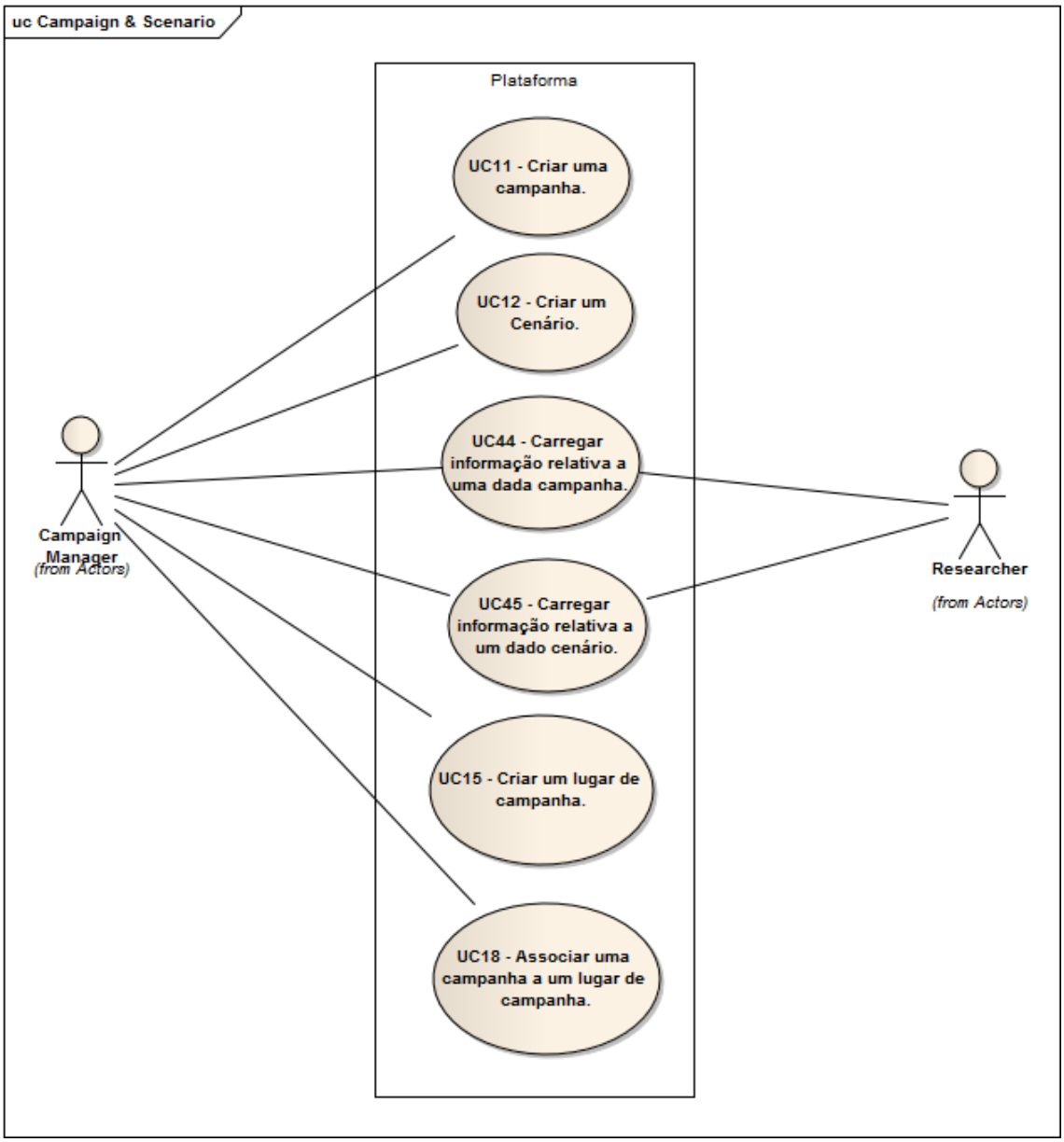

Figura 5.3: Pacote Campanha & Cenário.

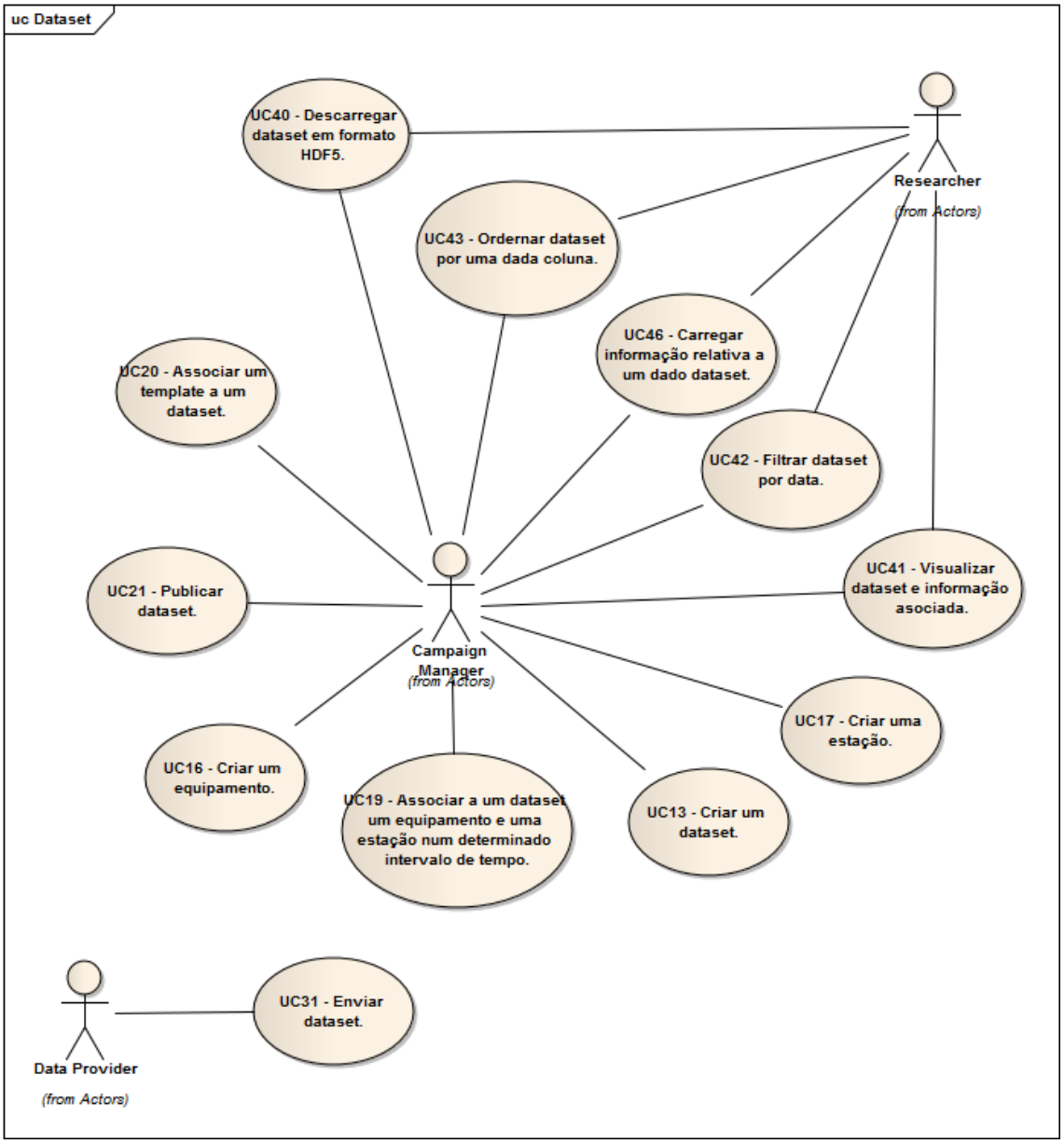

Figura 5.4: Pacote Dataset.

5.1.3 Requisitos Técnicos

Requisitos técnicos dizem respeito a aspetos não-funcionais do sistema, tais como questões relacionadas com o desempenho, fiabilidade, ambiente de desenvolvimento, etc [Amb04]. É de extrema importância identificar estes requisitos não-funcionais pois estes impõem limitações na arquitetura da plataforma a ser projetada.

A plataforma deve disponibilizar uma aplicação *Web* com interfaces intuitivas para os utilizadores e API adequadas para acesso por programas externos. A plataforma deve disponibilizar armazenamento suficiente para o número de campanhas e medições previstas e escalar à medida que o número de investigadores e o número de campanhas aumenta. Finalmente, a plataforma deve estar em conformidade com os níveis de serviço gerais que se aplicam a este tipo de infraestruturas:

Disponibilidade — tempo de atividade do sistema elevado.

Tolerância a falhas — o sistema deve ser capaz de continuar em funcionamento mesmo após uma falha de *hardware* ou *software*.

Desempenho — investigadores devem esperar um tempo de resposta aceitável.

Escalabilidade — a capacidade de aumentar o número de utilizadores e de recursos mantendo um desempenho similar.

Segurança — integridade do sistema e dos seus utilizadores, isto é, a integridade das ações dos utilizadores e dos dados associados.

Suportabilidade — a facilidade com a qual um sistema implementado pode ser administrado, incluindo tarefas tais como a monitorização do sistema, reparação de problemas que surgem e modernização dos componentes de *hardware* e *software*.

5.2 Modelo Conceptual do Domínio

A modelação conceptual do domínio "é a tarefa de descobrir os diferentes tipos de entidades que representam os conceitos, os seus relacionamentos, pertinentes para o espaço do problema" [WL12]. De forma a perceber melhor o domínio do problema foi criado um modelo conceptual que representa o chamado "Universo do Discurso". As Figuras 5.5 e 5.6 apresentam fragmentos do modelo concetual usando diagramas de classe UML. Estes diagramas descrevem de forma inequívoca e completa o domínio do problema. As entidades do domínio são apresentadas como caixas e as relações entre elas são representadas como linhas com a multiplicidade presente nas extremidades e, por vezes, com atribuição de um nome para melhorar a legibilidade.

Campanha

A Figura 5.5 representa a entidade ***Campaign*** e as suas relações. Uma *Campaign* consiste num conjunto de medições feitas num único ***Site***, por uma determinada instituição ou equipa, com uma

data de início e uma data de fim. Cada *Campaign* possui um gestor de campanha que é responsável pelo ciclo de vida da campanha, isto é, inserção de informação relativa à campanha, validação e publicação dos dados medidos. O gestor pode definir um conjunto de utilizadores ou instituições que têm acesso à campanha.

Durante uma campanha, uma equipa, provavelmente, irá realizar diferentes tipos de medições: em diferentes partes do *site*, usando outros equipamentos ou mudando a sua configuração. Assim, medições contínuas formam um ***Dataset*** e estão associadas a um ***Scenario***.

De forma a documentar as medições, a plataforma armazena ***Campaign Documents***. Estes documentos podem ser associados tanto a uma campanha como a um determinado cenário. Desta forma, é possível, por exemplo, descrever as conclusões tiradas após a realização de uma campanha ou cenário. Cada *Dataset* possui também associado um *Campaign Document* que descreve a configuração dos equipamentos usados para produzir o referido *Dataset*.

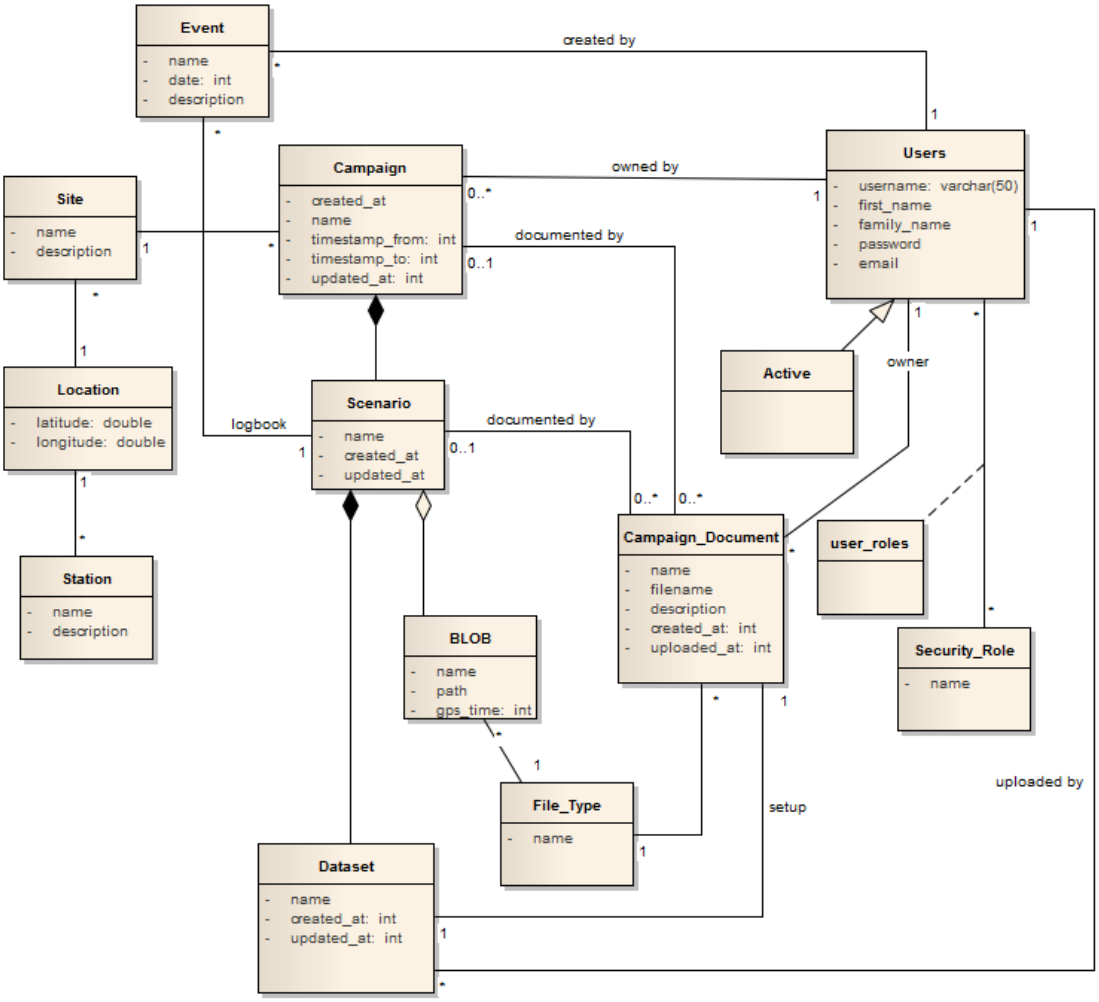

Figura 5.5: Definição da Entidade Campaign

Cenário

Um ***Scenario*** representa um objetivo, com data de início e de fim, a ser cumprido dentro da campanha associada. Dentro de um *Scenario* encontram-se *datasets* produzidos por diferentes equipamentos e medidos no mesmo intervalo de tempo.

Para além disso, cada *Scenario* mantém um *log* de ***Events*** ocorridos que poderão ser relevantes para a análise dos dados subjacentes.

Dataset

A Figura 5.6 representa a entidade ***Dataset***, uma abstração de todas as séries temporais manipuladas pela plataforma, e as suas associações. Os *datasets* armazenados na plataforma consistem em séries temporais de diferentes características dependendo da origem dos dados, ou seja, do equipamento. Assim, cada *Dataset* pode possuir um número variável de valores (colunas) definidos no ***Dataset Template*** que o descreve. *Dataset Template* é a entidade que descreve o que cada valor de um determinado ***Dataset*** representa (nome, unidade, dimensão, etc). Cada *Dataset Template* pode descrever um ou vários *datasets*.

Cada *Dataset* é composto por um conjunto de *measurements*. Cada ***Measurement*** é representado por um *timestamp* e um conjunto variável de valores que são descritos no *Dataset Template* associado. Um *Dataset* pode ser produzido por um ou mais *equipments*.

A entidade ***Equipment*** representa a fonte de dados que origina os *datasets* de acordo com a sua configuração. Cada *Equipment* tem associado um ***Model*** e um ***Type*** e cada *Model* tem associado um ***Manufacturer***.

Durante a produção de um *Dataset* cada *Equipment*, que contribui para a sua produção, tem associado uma ***Station***, uma data de início e uma data de fim. A combinação dos três campos representa a posição exata onde se encontrava o equipamento durante a produção do um *Dataset*.

Um *Dataset* pode ser de dois tipos: ***Measured*** ou ***Derived***. Um *Measured Dataset* representa um conjunto de dados produzidos por *Equipaments*, enquanto que um *Derived Dataset* representa um conjunto de dados derivado de um outro *dataset*, que pode ser *Derived* ou *Measured*. Esta situação é frequente quando é necessário efetuar algum procedimento sobre os dados brutos, como por exemplo, médias de um intervalo de tempo definido.

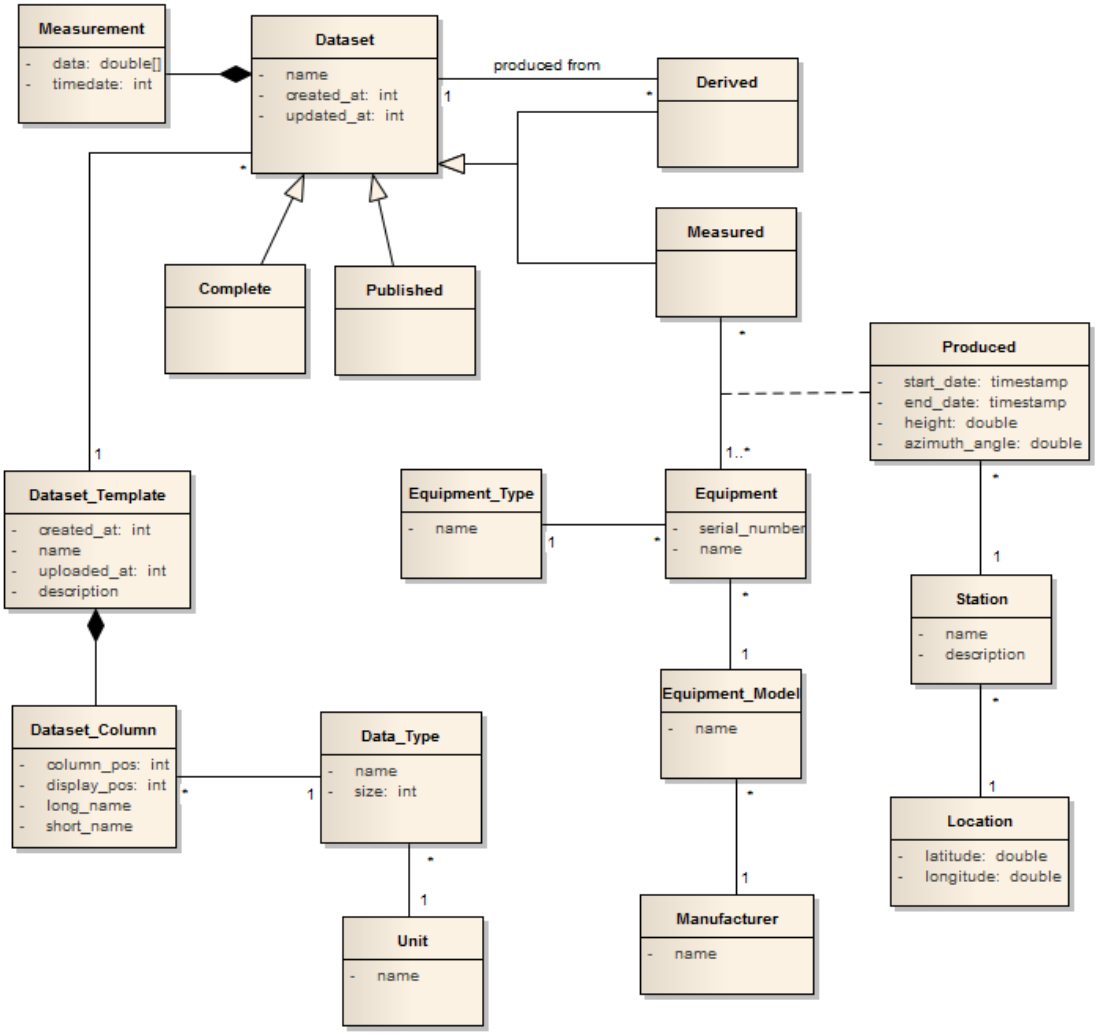

Figura 5.6: Definição da Entidade *Dataset*

5.3 Arquitetura

A Figura 5.7 apresenta os principais componentes do sistema e como estes interagem uns com os outros para alcançar os objetivos propostos. As secções seguintes detalham, em pormenor, cada componente e as suas interações. Para melhor perceber a arquitetura do sistema, nomeadamente a separação lógica das camadas apresentada na Figura 5.7, primeiro é necessário compreender o padrão de arquitetura MVC (*Model-View-Controller*).

5.3.1 Padrão MVC

A plataforma segue o padrão de arquitetura MVC (*Model-View-Controller*). Este padrão de arquitetura permite dividir a solução em três partes interligadas de modo a separar as representações internas dos dados das representações de dados que são apresentadas ao utilizador. As

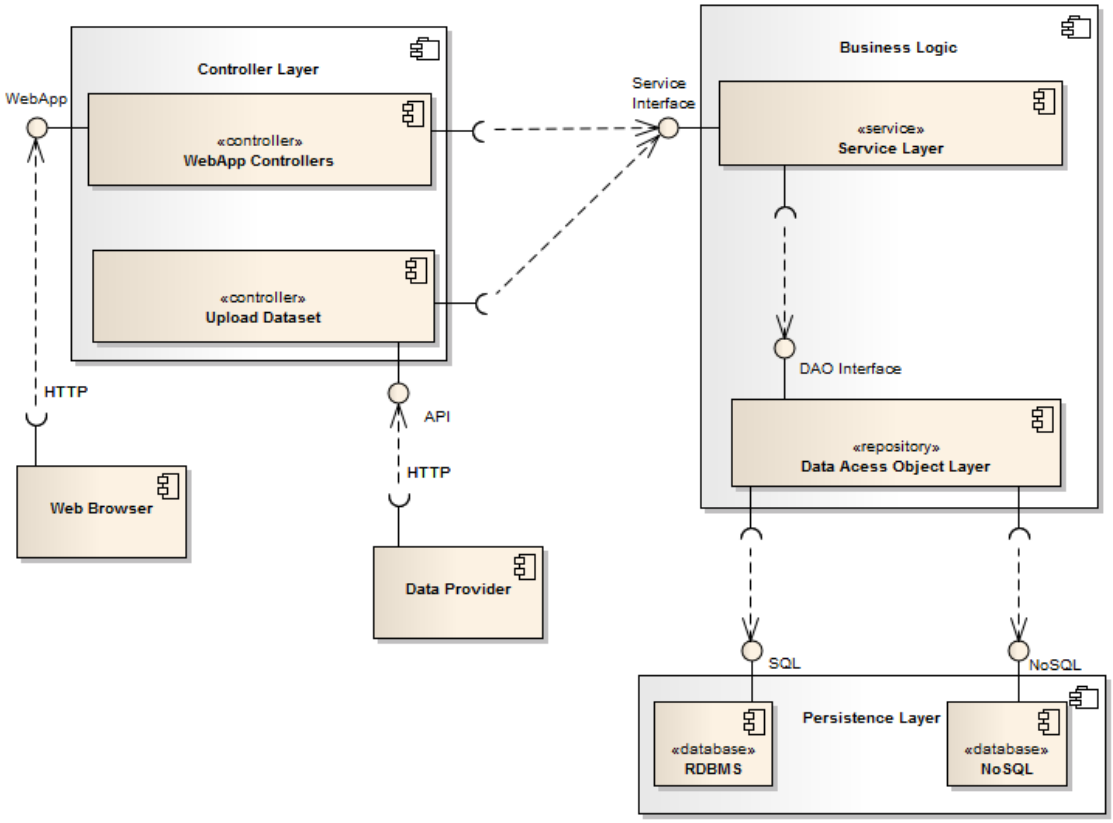

Figura 5.7: Diagrama de Componentes do Sistema.

representações internas devem ser completamente independentes da camada de apresentação, isto é, as representações internas devem permitir várias formas de apresentar os dados em simultâneo. Para isso a solução deve ter as seguintes responsabilidades devidamente separadas: *Model, View, Controller*.

Controller

O *Controller* gere os pedidos de utilizadores. A sua principal função é chamar e coordenar os recursos necessários para executar a ação do utilizador. Normalmente, o *Controller* chamará o *Model* apropriado para a tarefa e, em seguida, seleciona a *View* adequada.

Model

O *Model* representa as entidades e a lógica de negócio da solução. O *Model* disponibiliza ao *Controller* os dados, de acordo com o que foi solicitado pelo utilizador. O modelo de dados será o mesmo, independentemente da forma como este é apresentado ao utilizador, e não depende de nenhum dos outros dois componentes.

View

A *View* oferece formas diferentes de apresentar os dados ao utilizador recebidos a partir do *Model*.

5.3.2 Lógica de Negócio

Depois de ter sido introduzido o padrão de arquitetura usado, esta secção foca-se em explicar a componente *Model* da solução. Esta componente é composta por duas camadas distintas: *Service Layer* e *Data Access Object Layer*.

Service Layer

A camada de serviços é responsável por expor um conjunto de operações disponíveis na plataforma que foram descritas na Tabela 5.1. Esta camada encapsula toda a lógica de negócio da plataforma e faz uso da interface exposta pela camada *Data Access Object* para acesso a dados. Detalhadamente, a camada é composta por um conjunto de interfaces de acordo com as entidades descritas no Domínio do Problema na Secção 5.2, das quais se destacam:

- *Campaign Service* — expõe um conjunto de operações para visualização e manipulação de campanhas, sítios de campanha e *templates*.
- *Scenario Service* — expõe um conjunto de operações para visualização e manipulação de cenários.
- *Measurement Service* — expõe um conjunto de operações para visualização e manipulação (filtragem, ordenação, inserção) de *datasets*.

Data Access Object

A camada de acesso a dados é usada para separar a API de baixo nível, para acesso a dados, das camadas de mais alto nível. Por outras palavras, esta camada é uma abstração sobre como aceder a uma base de dados usando objetos. A camada expõe um conjunto de interfaces que definem as operações CRUD sobre a base de dados.

Para melhor perceção das tarefas desta camada, a Figura 5.8 representa um dos *Data Access Object* existentes, neste caso o responsável pelas operações sobre a entidade *Campaign*. Como é possível visualizar, este DAO implementa uma interface que expõe um conjunto de operações, que serão usadas pelo *Campaign Service*, sobre a entidade *Campaign* que representa dados persistentes que estão armazenados na base de dados.

Esta camada é encapsulada pela camada superior, *Service Layer*, que apenas tem acesso a dados através de pedidos à camada DAO. Este tipo de abordagem traz algumas vantagens pois, caso a camada superior lidasse diretamente com a camada de dados, tornava-se específica da implementação, o que tornaria muito mais trabalhoso a implementação de uma segunda abordagem.

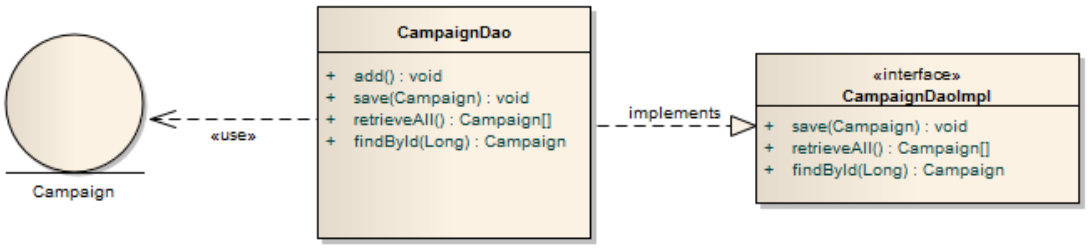

Figura 5.8: Exemplo de um *Data Access Object*.

5.3.3 Controladores

Esta camada, representada na Figura 5.7, é o ponto de entrada do utilizador na plataforma e serve como estrada entre o pedido do utilizador e o serviço adequado a ser chamado. É composta por um conjunto de controladores que seguem a nomenclatura das outras camadas, *<Entity Name>Controller*. A camada encontra-se divida em dois grupos de controladores: *WebApp Controllers* e *UploadDatasetController*. O primeiro grupo contém os controladores da aplicação *Web*, ou seja, aqueles que lidam com todos os pedidos que são realizados através de um *Web Browser*. O segundo está ligado à API que permite que programas externos façam o carregamento de *datasets* para a plataforma no formato HDF5.

WebApp Controllers

WebApp Controllers representa um conjunto de controladores que fazem a ligação entre os pedidos do utilizador e a camada de serviços. Cada pedido é encaminhado para o respetivo *handler*, dos quais se destacam:

- *Campaign Controller* — lida com os pedidos relacionados com Campanhas. Todos os pedidos GET e POST sobre o URI */campaigns* são tratados por este controlador.
- *Equipment Controller* — lida com pedidos de visualização e inserção de equipamentos sobre o caminho */equipments*.
- *Scenario Controller* — lida com os pedidos relacionados com Cenários e *Datasets* sobre os caminhos */campaigns/scenarios* e */campaigns/scenarios/datasets*.

Upload Dataset Controller

Este controlador, responsável por processar os pedidos efetuados por *Data Providers*, expõe uma API para o carregamento de *datasets* em formato HDF5. Assume uma importância significativa, pois representa o ponto de entrada para que aplicações externas à plataforma possam carregar *datasets* que estão a ser produzidos num sítio de campanha.

5.4 Tecnologias Escolhidas

Depois da análise do estado de arte e do desenho da arquitetura, foram escolhidas as tecnologias para o desenvolvimento das duas abordagens. O critério de seleção das ferramentas a serem usadas privilegiou aquelas que representassem projetos de código aberto ou que, pelo menos, fossem de uso gratuito. Em Engenharia de Software são frequentemente usados os termos *front-end* e *back-end* para se referir à separação de conceitos entre a camada de apresentação e a camada de acesso a dados. Ao longo desta secção serão usados estes termos para distinguir estas duas componentes.

Back-end

Para a implementação do *back-end* foi usado JavaEE[2], e um conjunto de *frameworks* das quais se destacam: Spring framework[3], Hibernate[4], The HDF Object Package[5].

JavaEE

Java Enterprise Edition Plataform, ou simplesmente JavaEE, é uma plataforma que expande o JavaSE (Java Standard edition) disponibilizando um conjunto de API para facilitar o desenvolvimento e execução de *software* corporativo[6], nomeadamente serviços *Web* e outras aplicações de larga escala que necessitem de ser escaláveis fiáveis e seguras. A plataforma enfatiza o uso de anotações para configuração e a programação por convenção com vista a diminuir o número de decisões que um programador tem que tomar.

Spring Framework

O Spring é uma poderosa *framework* em código aberto para o desenvolvimento de aplicações em cima de JavaEE. Para o desenvolvimento de aplicações *Web* com recurso a JavaEE são usados EJB (Enterpise Java Beans), um dos componentes da JavaEE API, para o encapsulamento da lógica de negócio de uma aplicação. Para que isto seja possível, é necessário que o servidor de aplicações onde se vai correr a aplicação possua um EJB *Container* que forneça um ambiente de execução. O Spring possibilita o desenvolvimento de aplicações *Web* com recurso a POJO (*Plain Old Java Objects*), em vez de EJB. A vantagem do desenvolvimento apenas com recurso a POJO é que torna-se dispensável um servidor de aplicações Java que implemente toda a JavaEE API, uma vez que não é necessário, por exemplo, um *container* para EJB. Quando se desenvolve uma aplicação complexa em Java as classes devem ser independentes umas das outras o mais possível para aumentar a probabilidade de poderem ser reutilizadas e para que seja possível testá-las de forma independente. O Spring usa o padrão *Dependency Injection* (DI), que permite combinar

[2] http://www.oracle.com/technetwork/java/javaee/overview/index.html
[3] http://projects.spring.io/spring-framework/
[4] http://hibernate.org/
[5] https://www.hdfgroup.org/products/java/hdf-object/index.html
[6] http://www.webopedia.com/TERM/E/enterprise_application.html

essas classes e ao mesmo tempo mantê-las independentes. O que o Spring faz é ligar as classes usando um ficheiro de configuração XML, permitindo assim que todos os objetos sejam iniciados e instanciados pela *framework* e injetados nos sítios indicados.

Hibernate

Como Java é uma linguagem orientada objetos, foi necessária para a primeira abordagem uma *framework* que mapeasse as tabelas da base de dados para classes em Java. O mapeamento objeto-relacional (ORM), é uma técnica de desenvolvimento utilizada para converter dados de uma tabela relacional para linguagens orientadas a objetos. O Hibernate é uma solução ORM para Java que trata do mapeamento entre as tabelas e as classes, abstraindo o uso de *queries* SQL para operações CRUD na base de dados. Embora existam outras soluções ORM para Java, a *framework* Hibernate é a mais utilizada, pois suporta a maior parte dos sistemas de base de dados relacionais. Além disso, esta *framework* possui estratégias inteligentes para diminuir o número de acessos à base de dados e disponibiliza uma forma simples de aceder aos dados.

PostgreSQL

O PostgreSQL[7] é um sistema de gestão de base dados, que embora não seja o mais famoso, é o sistema de base de dados relacional em código aberto mais poderoso e mais avançado. Quando comparado com outros RDBMS, o *PosgreSQL* destaca-se pelo seu alto e completo suporte das funcionalidades de um sistema de base de dados relacional, tais como transações fiáveis garantidas pelas propriedades ACID. Uma das razões para a escolha do *PosgreSQL* para a primeira abordagem, foi o facto de este ser capaz de executar muitas tarefas em simultâneo de forma eficiente. A prova disso é o seu modelo de suporte para concorrência que, ao contrário da maior parte dos sistemas de base de dados relacionais — que usam *locks* para controlo da concorrência — mantém a consistência usando o modelo multi-versão (MVCC). A grande diferença entre este modelo e o modelo de *locks*, é que os *locks* adquiridos para leitura de dados não entram em conflito com os *locks* adquiridos para escrita de dados, ou seja, uma leitura nunca bloqueia uma escrita e vice-versa.

MongoDB

O MongoDB, como anteriormente descrito, é uma base de dados NoSQL em código aberto orientada a documentos projetada para fácil desenvolvimento e escalabilidade. Embora todas as soluções NoSQL garantam a alta disponibilidade e performance, a escolha recaiu nesta solução, para a segunda abordagem, por esta ser a opção melhor documentada devido à grande comunidade que a suporta.

[7] http://www.postgresql.org/

HDF Object Package

O Hierarchical Data Format (HDF) é um conjunto de formatos de ficheiros desenhados para guardar e organizar grandes quantidades de dados. A HDF Object Package é uma biblioteca Java que fornece uma interface orientada a objetos para manipulação de dados de ficheiros HDF. Foi usada para gerar os ficheiros HDF5 a partir de CSV e para manipular ficheiros HDF5 que chegam à plataforma através da API de carregamento de *datasets*.

Tomcat

O Tomcat é um servidor *Web* em código aberto que implementa alguns dos componentes da JavaEE API, como Java Servlets e JavaServer Pages. É responsável por gerir a execução de páginas *Web* e *servlets* para aplicações JavaEE. Este foi o *container* escolhido para correr a solução, uma vez que com o uso da Spring Framework torna-se possível utilizar um servidor *Web* mais leve, que não precise de implementar todas as funcionalidades da JavaEE API.

Front-end

Para implementação da camada de apresentação foram usadas JavaServer Pages (JSP) que permitem a utilização das tecnologias mais conhecidas para o desenvolvimento de interfaces: HTML, CSS e JavaScript.

JavaServer Pages

A JSP[8] é uma tecnologia Java para geração de páginas *Web* dinâmicas baseadas em HTML, CSS, JavaScript, XML, etc. Suporta *taglibs*, que são apoiados por pedaços de código Java que permitem controlar dinamicamente o fluxo da página. A grande vantagem das JSP é o facto de poderem ser combinadas com *Servlets* que lidam com a lógica de negócio e assim aceder facilmente aos dados.

5.5 Resumo

Durante este capítulo foram descritas e priorizadas as ações possíveis que os diferentes atores podem efetuar na plataforma. Posteriormente, foi apresentada a arquitetura da solução e uma análise crítica às ferramentas utilizadas. A arquitetura da solução foi pensada tendo sempre em conta as duas abordagens a ser implementadas, de modo a que as alterações a serem feitas fossem minimizadas. No próximo capítulo é discutida a implementação que resultou da arquitetura desenhada.

[8] http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Capítulo 6

Implementação

Neste capítulo são analisadas as decisões tomadas durante a implementação. São também discutidos alguns pormenores de implementação para melhor perceção das camadas que compõem a solução.

6.1 Introdução

Após o desenho da arquitetura especificada no Capítulo 5, foram tomadas algumas decisões para que a implementação fosse objetiva e clara de forma a não afetar trabalho futuro que possa vir a ser desenvolvido.

Como especificado na arquitetura da plataforma, a lógica de negócio é composta por duas camadas distintas, uma camada de serviços e uma camada de acesso a dados. Como a solução passa pela implementação de duas abordagens diferentes, que serão detalhadas no decorrer deste capítulo, foi decidido que a camada de serviços teria que ser independente da implementação, isto é, que pudesse ser usada nas duas abordagens sem que tivesse que ser alterada. Isto implicou que a camada de serviços seja uma camada de mais alto nível que se abstrai da fonte de dados e está preparada para trabalhar independentemente de como a camada de acesso a dados está implementada.

Outra decisão importante na implementação é a nomenclatura consistente usada. Tal como introduzido no Capítulo 5, na secção de arquitetura do sistema, cada componente do sistema apresenta o nome pré-definido segundo a regra <*Entity Name*><*Layer Name*>. Isto permite rapidamente chegar ao componente procurado, bastando combinar o nome da entidade que se procura com a camada desejada.

Para além daquilo que foi especificado, foram também adotadas outras boas práticas de programação como testes unitários, separação de responsabilidades, tratamentos de exceções e situações de erro e código devidamente documentado.

6.2 Considerações Iniciais

Como especificado no Capítulo 5, a solução segue o padrão de arquitetura MVC. Para a implementação deste padrão foi usado JavaEE[1] e um conjunto de *frameworks* das quais se destacam: Spring framework e Hibernate. Estas tecnologias e outras utilizadas encontram-se detalhadas na Secção 5.4. É importante perceber como o Spring framework, nomeadamente o seu módulo Spring MVC, implementa este padrão. Este módulo é projetado em torno de um *DispatcherServlet* que lida com todas as solicitações e respostas HTTP. O fluxo de processamento de pedidos do Spring MVC *DispatcherServlet* é ilustrado na Figura 6.1.

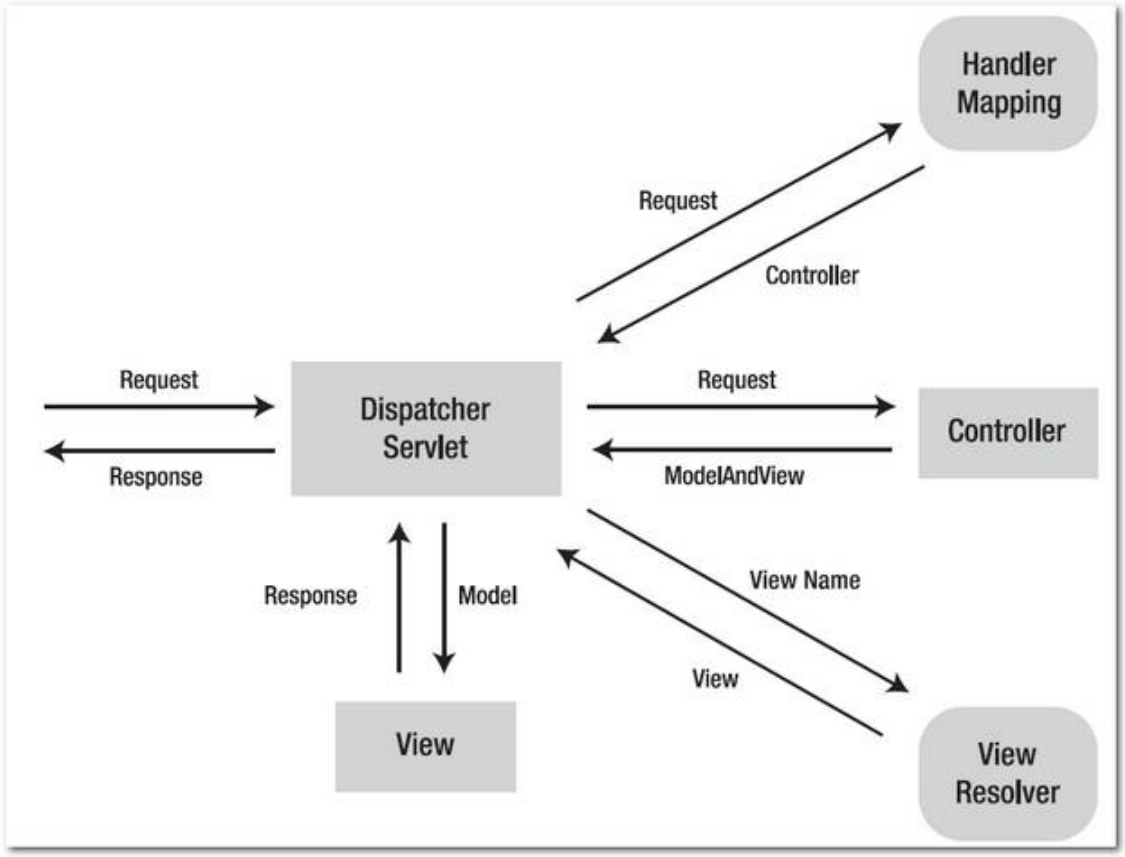

Figura 6.1: Fluxo de Processamento de Pedidos Spring MVC *DispatcherServlet* [Mak08].

Quando um pedido HTTP chega ao servidor, acontece o seguinte:

1. Após receber um pedido HTTP, o *DispatcherServlet* consulta o *HandlerMapping* para saber qual o *Controller* apropriado a chamar.

2. O *Controller* recebe o pedido e chama os métodos do serviço apropriado de acordo o pedido HTTP. O serviço irá definir o modelo de dados baseado na lógica de negócio definida e retorna o nome da *View* para o *DispatcherServlet*.

[1] http://www.oracle.com/technetwork/java/javaee/overview/index.html

3. O *DispatcherServlet* chamará o *ViewResolver* para determinar qual a *View* apropriada para o pedido.

4. Uma vez encontrada, o *DispatcherServlet* passa o modelo de dados para a *View* que depois é devolvida para ser renderizada no *Web browser*.

6.3 *Naming* de Recursos

Além de utilizar os verbos HTTP de forma adequada, os nomes dos recursos são, sem dúvida, o conceito mais debatido e mais importante ao criar um serviço *Web* compreensível. Quando os recursos são bem nomeados, uma API torna-se intuitiva e fácil de usar. Cada recurso/entidade descrita na Secção 5.2 tem um URI no sistema que a identifica.

A Tabela 6.1 apresenta os URIs disponíveis na plataforma, a descrição de cada um e nome do controlador que é responsável por gerir os pedidos efetuados.

Tabela 6.1: Naming de Recursos

Controlador	**URI**	**Tipo**	**Descrição**
CampaignController	*/campaigns*	GET	Lista todas as Campanhas.
	/campaigns/<id>	GET	Lista a Campanha <id> e scenários associados.
	/campaigns/add	GET	Formulário para criação de Campanha.
		POST	Adiciona nova Campanha.
ScenarioController	*/campaigns/scenarios/<id>*	GET	Lista o Cenário <id> e *datasets* associados.
	/campaigns/scenarios/add	GET	Formulário para criação de Cenário
		POST	Adiciona novo Cenário.
DatasetController	*/campaigns/scenarios*	GET	Lista *dataset* <id> e informações associadas.
	/datasets/<id>	POST	Retorna *measurements* de acordo com as condições especificadas no *payload*.
	/campaigns/scenarios/datasets/add	GET	Formulário para criação de um *Dataset*.
		POST	Adiciona um novo *Dataset*
	/campaigns/scenarios/datasets /templates/add	POST	Adiciona um novo template.

Continua na próxima página...

Tabela 6.1 – continuação da página anterior

Controlador	URI	Tipo	Descrição
	/campaigns/scenarios/datasets /<id>/produced	POST	Adiciona um equipamento especificado no *payload* como produtor do *dataset* <id>.
EquipmentController	*/equipments*	GET	Lista todos os equipamentos
	/equipments/add	GET	Formulário para criação de um equipamento.
		POST	Adiciona uma novo equipamento.
SiteController	*/sites/add*	GET	Formulário para criação de lugar de campanha.
		POST	Adiciona novo lugar de Campanha.
FileUploadController	*/uploadfile*	POST	Carrega ficheiro HDF5 e armazena na base dados.

6.4 Camada de Acesso a Dados

A camada de acesso a dados, tal como especificado no Capítulo 5, expõe uma interface para acesso transparente aos dados presentes na base dados. Esta camada assume real importância na implementação pois é nela que as duas abordagens diferem. Embora as abordagens sejam diferentes, a interface exposta é a mesma de forma a que a sua utilização seja transparente para as camadas superiores. Na prática isto significa que as camadas superiores desconhecem que tipo de armazenamento está a ser usado. O objetivo desta secção é explicar as diferenças existentes nesta camada e de que forma foram implementadas.

6.4.1 Primeira Abordagem

A primeira abordagem recorre a um RDBMS para armazenar os dados, seguindo o modelo apresentado na Secção 5.2. O PostgreSQL foi o RDBMS escolhido e encontra-se detalhado na Secção 5.4. Tal como nas camadas superiores, para cada entidade existe um *DAO* que gere o acesso aos dados representados por essa entidade. Para que isto seja possível, foi necessário recorrer a uma *framework* que mapeasse as tabelas relacionais em classes Java. A *framework* escolhida foi o Hibernate e encontra-se também detalhada na Secção 5.4.

O *MeasurementDAO* assume maior relevância, pois é aquele que permite o acesso aos *Measurements*. Esta entidade representa cada linha das séries temporais produzidas por equipamentos.

É considerada a informação mais significativa da plataforma, de tal forma que a tabela correspondente na base de dados, no decorrer de uma campanha, armazena quantidades de dados na ordem de mil milhões de linhas.

Um dos problemas desta abordagem foi encaixar a heterogeneidade das séries temporais no modelo de dados uma vez que cada série temporal pode possuir um número variável de valores. Para isso, recorreu-se aos *arrays* em PostgreSQL que permitem armazenar um número variável de valores sem que haja necessidade de os descrever. Assim, é possível que a tabela *measurements* armazene qualquer série temporal.

6.4.2 Segunda Abordagem

A segunda abordagem recorre a um sistema de base de dados NoSQL para armazenamento da entidade *Measurement*. O sistema de base de dados escolhido foi o MongoDB, uma base de dados orientada a documentos que se encontra detalhada na Secção 5.4.

De forma a não perder as vantagens dos RDBMS, todas as outras entidades são armazenadas da mesma forma que a primeira abordagem. A informação relativa às outras entidades é considerada reduzida quando comparada com a informação representada pela entidade *Measurement* e, por isso, não se justifica a passagem das mesmas para uma base de dados NoSQL. O grande fluxo de dados acontece quando um *Data Provider* carrega ficheiros HDF5 para a plataforma ou quando um *Researcher* solicita um *Dataset*. Tanto num caso como noutro isto implica um grande número de inserções ou retorno de dados por parte da base de dados. Nos restantes casos, as entidades são criadas e acedidas pelo utilizador na aplicação *Web* e nunca são acedidas em massa.

Na primeira abordagem recorreu-se a uma *framework* ORM para que o mapeamento entre classes Java e a base de dados fosse efetuado. Neste caso, cada instância *Measurement* é representada por um documento na base de dados seguindo a seguinte estrutura:


```
{
    "_id": "558b3b197760a8f31ca6d7f6",
    "timedate": "2015-06-24T23:19:53.024Z",
    "data": [
        45.34,
        2781.3,
        -345.3,
        2.34,
        23.8,
        2.4,
        -2.14,
        1.67,
        8.35,
        75.4
    ],
    "dataset": 2
}
```


Figura 6.2: Exemplo de um *Measurement* em MongoDB.

- Cada documento contém um _id para representação interna, um *timestamp* que identifica quando a medição foi efetuada, um *array* de dados que contém os valores medidos e o identificador de *dataset* ao qual o *Measurement* em questão pertence.

- Como não foi usada nenhuma *framework* para mapear os documentos para classes em Java, esse mapeamento foi realizado recorrendo a um pequeno módulo desenvolvido — que transforma documentos do MongoDB em entidades representadas por classes Java — sendo transparente para as camadas superiores. Quando um documento é devolvido, este é primeiramente convertido para um objeto de uma classe em Java, comum às duas abordagens, para que possa ser manipulado nas camadas superiores sem que isso implique alguma alteração. Da mesma forma que um documento é convertido para um objeto de uma classe Java comum às duas abordagens, um *Measurement* é convertido para um documento em MongoDB para que possa ser inserido na base de dados.

6.5 Carregamento de *Datasets*

Tal como foi referido no Capítulo 5 e especificado na Tabela 6.1, a solução expõe uma API acessível através do URI */uploadfile* para o carregamento de ficheiros em formato HDF5. Os equipamentos que estão no terreno produzem, normalmente, ficheiros em formato CSV. O problema surge quando é necessário carregar para a plataforma um *dataset* ou parte dele. Os ficheiros produzidos em formato CSV rapidamente atingem um tamanho elevado, tornando moroso o carregamento dos mesmos para a plataforma. Além disso, cada equipamento possui uma estrutura diferente dentro do CSV, isto é, não existe nenhum padrão entre os ficheiros de equipamentos diferentes. Das estruturas observadas concluiu-se que são muito díspares. Dependendo do equipamento, a estrutura do ficheiro pode apresentar ou não um cabeçalho com informação adicional e os nomes das colunas podem estar ou não definidos. Isto acontece porque a estrutura é definida pela marca do equipamento, sendo que os equipamentos no terreno pertencem a várias marcas. O facto das estruturas serem tão dispares trouxe um problema para a plataforma pois o objetivo era que o *carregamento* de *datasets* fosse automatizado independentemente do ficheiro carregado.

6.5.1 HDF5Converter

Para tentar resolver os problemas enunciados foi desenvolvido um módulo externo à plataforma que é capaz de:

1. Reduzir o espaço ocupado pelos ficheiros CSV.

2. Construir uma estrutura que é comum a todos os ficheiros.

3. Carregar um ou vários *datasets* para a plataforma.

O *HDF5Converter* é uma aplicação *stand-alone*[2] em Java que foi criada para automatizar o processo acima descrito. Esta aplicação, que opera independentemente da plataforma, é responsável por converter os ficheiros CSV para o formato HDF5 seguindo uma estrutura predefinida e, posteriormente, carregá-los para a plataforma. Na prática, cada tipo de equipamento possui uma versão deste conversor, adaptada para a estrutura que é usada nos ficheiros CSV produzidos pelo equipamento.

Para resolver o problema de espaço criado pelos ficheiros CSV foi usado o formato HDF5. Este formato foi o escolhido porque está desenhado para guardar e organizar grandes quantidades de dados de forma eficiente[3].

A Figura 6.3 representa a estrutura única dos ficheiros CSV após serem convertidos para HDF5. Cada grupo, representado por uma pasta na imagem, representa um *Dataset* e contém um atributo que identifica o *Dataset* correspondente na plataforma. Dentro de cada grupo encontram-se matrizes extraídas dos ficheiros CSV, que representam partes do *dataset* em questão, sendo que cada linha da matriz representa um *Measurement* desse mesmo *dataset*.

Durante a conversão é questionado pelo conversor ao utilizador a que *dataset* pertence o conjunto de ficheiros CSV. Isto implica que o *dataset* tenha sido previamente criado na plataforma, através do URI */campaigns/scenarios/datasets/add*, para que um id seja gerado e possa ser usado posteriormente no conversor. O *Campaign Manager* está responsável por criar a campanha e adicionar as informações necessárias para que o *dataset* possa ser criado.

6.5.2 Fluxo de Processamento

A Figura 6.4 representa um diagrama de sequência simplificado do processamento de carregamento de um ficheiro HDF5, desde o seu envio até ao armazenamento na base de dados. Quando um *Data Provider*, como é o caso da aplicação *HDF5Converter*, carrega um ficheiro HDF5 através do URI */uploadFile*, é enviado no *payload* do pedido o nome e o conteúdo do ficheiro que será analisado pelo *UploadFileController*. Este controlador, após receção do ficheiro, verifica se a extensão é válida e guarda-o no sistema. Posteriormente, pede ao módulo *HDF5Import* para que um *token* que identifique aquele carregamento seja criado. O *token* criado é enviado como resposta ao *Data Provider*. Seguidamente, o controlador indicará ao *HDF5Import* o caminho no sistema do ficheiro que necessita de ser armazenado na base de dados.

O módulo *HDF5Import* possui um conjunto de *threads*, que depende do número de núcleos do processador da máquina onde a plataforma corre, preparadas para executar tarefas de leitura de *Measurements* a partir do ficheiro e, posteriormente, inseri-los na base de dados. O *Loop Async*, indicado na Figura 6.4, representa essas tarefas que são criadas à medida que o método *importToDatabase()* percorre os grupos do ficheiro. A cada tarefa é atribuído um grupo do ficheiro. A tarefa extrai o id do *dataset*, lê as matrizes presentes convertendo cada linha para uma entidade *Measurement* e invoca a camada *Data Access Object* para que a lista de *Measurements* seja armazenada na base de dados. Como indicado no diagrama, estas tarefas correm paralelamente, não

[2] http://www.webopedia.com/TERM/S/stand_alone.html
[3] https://www.hdfgroup.org/HDF5/whatishdf5.html

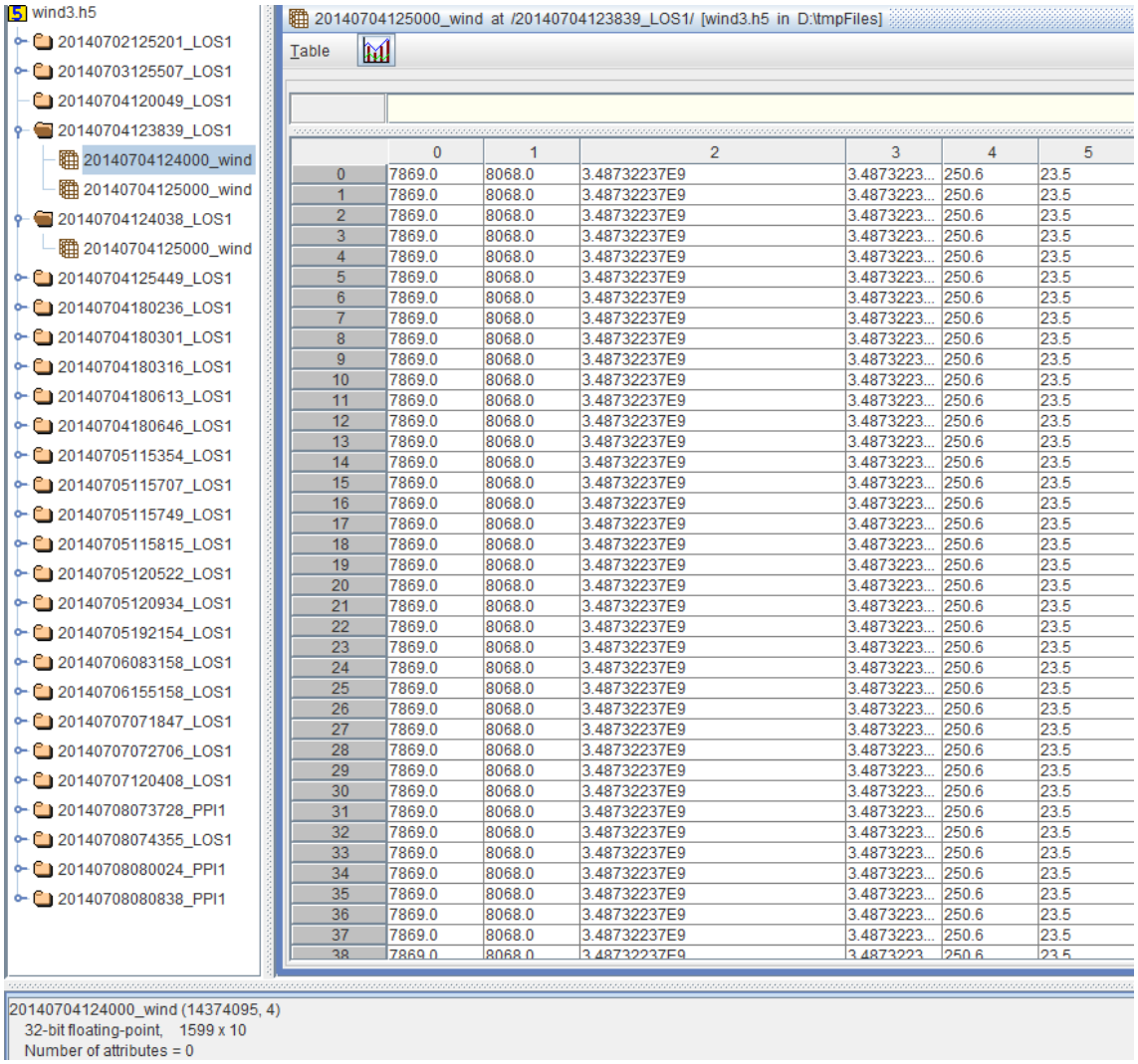

Figura 6.3: Estrutura dos Ficheiros HDF5.

deixando o módulo *HDF5Import* em espera. A camada *Data Access Object* invocará o método *save()*, iterativamente, até que todos os *Measurements* estejam armazenados. Após todas as tarefas concluírem com sucesso, o *token* será marcado como concluído. Durante todo este processo, o *Data Provider* pode questionar o controlador com o *token* recebido para saber se o processo de armazenamento já foi concluído. Esse processo de verificação está representado na caixa *alt* do diagrama.

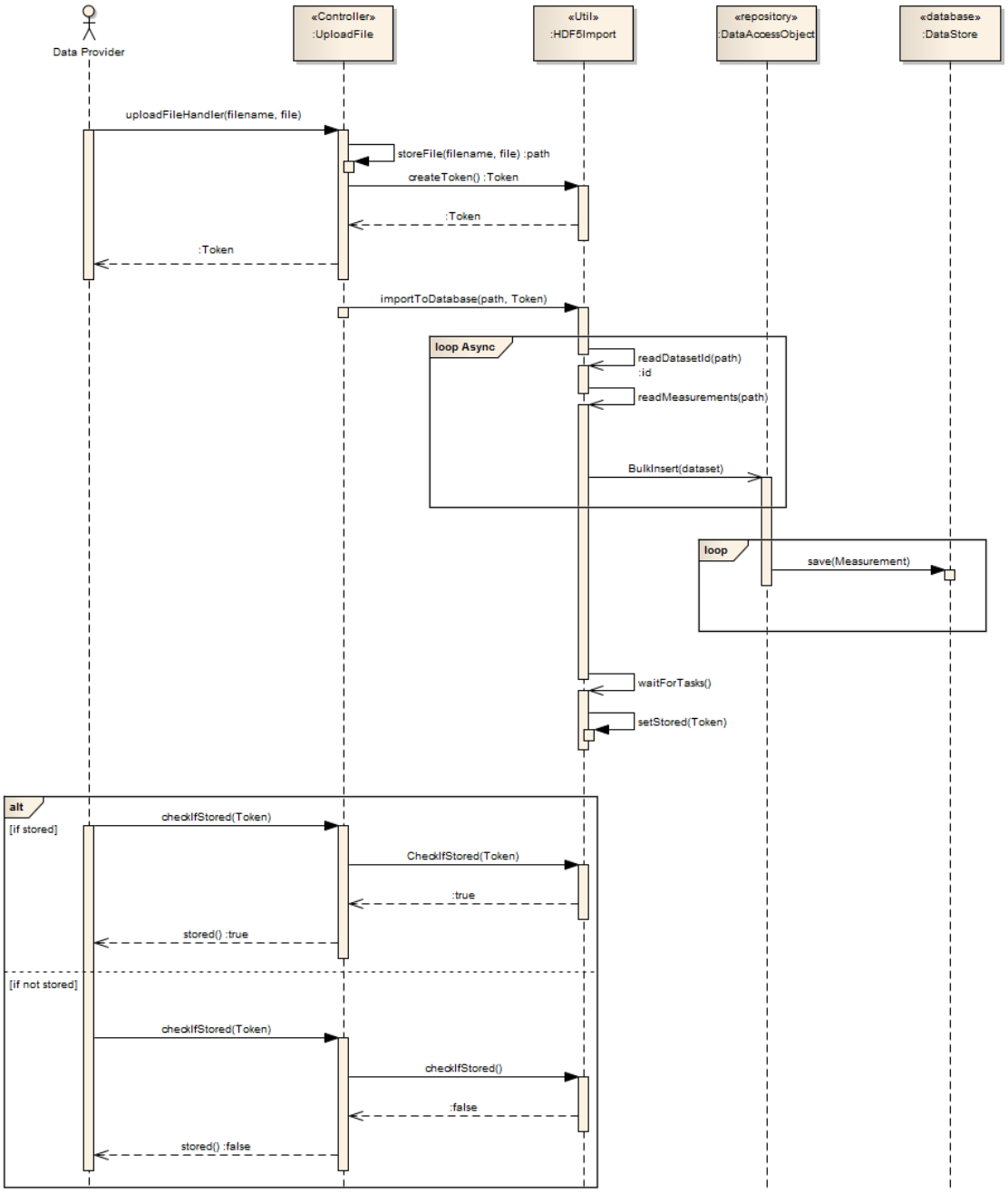

Figura 6.4: Diagrama de Sequência do Carregamento Ficheiro HDF5.

6.6 Otimizações

Durante a implementação da solução foram realizadas algumas otimizações para o melhoramento da performance da plataforma. Esta secção detalha as otimizações mais importantes efetuadas.

6.6.1 Inserção de Dados

Após o desenvolvimento da primeira abordagem constatou-se que a inserção em massa de *Measurements* era consideravelmente lenta. Pior que isso era o facto de, a partir de cerca de cem mil inserções, a exceção *OutOfMemoryException* ser lançada. Isto acontece porque o *Hibernate* guarda em cache todas as instâncias inseridas ao nível da cache de sessão. Para resolver este problema passou-se a usar *StatelessSession* em inserções em massa. *StatelessSession* não disponibiliza nenhum mecanismo de cache, limita-se, simplesmente, a executar as *queries* diretamente na base de dados. Além do tempo de execução ser mais reduzido, a memória utilizada desce drasticamente. De modo a reduzir o tempo de inserção, as inserções em massa passaram a ser efetuadas paralelamente em duas ou mais *threads*, dependendo do número de núcleos disponíveis na máquina onde a solução corre.

Por outro lado, no desenvolvimento da segunda abordagem para a inserção em massa de *Measurements*, fez-se uso dos métodos disponibilizados pela API do MongoDB. Isto permite que o MongoDB execute as operações de inserção em paralelo em grupos de mil operações.

6.6.2 Visualização de Dados

A determinada altura no desenvolvimento verificou-se que um *dataset* não poderia ser retornado quando se lida com grandes quantidades de dados que se deseja exibir numa tabela (milhões de linhas, por exemplo). A estes níveis, enviando os dados para o cliente, e, em seguida, processá-los em JavaScript envolve uma sobrecarga não aceitável, resultando num fraco desempenho da solução. Para resolver este problema passou-se a fazer o processamento do lado do servidor, isto é, a filtragem, paginação e ordenação passaram a ser calculados do lado do servidor. Para isso foi definida uma API entre o cliente e o servidor. Para cada ordenação, mudança de página na tabela ou filtragem, é feito ao servidor um pedido AJAX através do URI */campaigns/scenarios/-datasets/*<*id*>. A Tabela 6.2 apresenta os parâmetros que o servidor espera no *payload* do pedido.

Tabela 6.2: Parâmetros dum Pedido de um *Dataset*

Parâmetro	Tipo	Descrição
draw	Integer	Contador. Pedidos AJAX por omissão são assíncronos. Este contador assegura que os pedidos são executados sequencialmente.
start	Integer	Indica a partir de que registo deve ser retornado. 0 representa o primeiro registo.

Continua na próxima página...

Tabela 6.2 – continuação da página anterior

Parâmetro	Tipo	Descrição
length	Integer	Número de registos que a tabela pode apresentar.
order[i][column]	Integer	Coluna a que uma ordenação deve ser aplicada.
order[i][dir]	String	Direção da ordenação da coluna. Pode ser "asc"ou "desc
dateFrom	String	Data a partir da qual se devem considerar os registos.
dateTo	String	Data até à qual se devem considerar os registos.

Esta API torna a visualização muito mais eficiente pois apenas os dados a serem visualizados naquele momento são devolvidos. Se os parâmetros *dateFrom* e *dateTo* forem especificados, o servidor retornará registos entre o intervalo indicado. A tabela permite visualizar até 50 registos de cada vez.

Após efetuar o pedido, o cliente espera como resposta os parâmetros descritos na Tabela 6.3.

Tabela 6.3: Parâmetros de Resposta dum Pedido de um *Dataset*

Parâmetro	Tipo	Descrição
draw	Integer	Contador recebido no pedido. Pedidos AJAX por omissão são assíncronos. Este contador assegura que os pedidos são executados sequencialmente.
recordsTotal	Integer	Número de registos na base de dados, antes da filtragem.
recordsFiltered	Integer	Número de registos após a filtragem.
data	array	Dados a serem mostrados na tabela.

6.7 Testes

Durante o desenvolvimento da solução foram criados testes unitários para que os vários módulos fossem testados. Os testes foram efetuados usando Junit4 e abrangem a camada de acesso a dados, os serviços e ferramentas desenvolvidas — como é o caso do módulo *HDF5Import*. No caso da camada de acesso a dados, foram escritos testes para as duas abordagens. Para cada camada os testes abrangem os métodos mais utilizados e focam-se, principalmente, em métodos que envolvam inserções ou retorno de dados em massa.

Os testes permitiram fazer grandes mudanças ao código rapidamente, pois indicam se essas mudanças afetaram o que já tinha sido desenvolvido. Além disso, permitiram ter um *feedback* instantâneo das novas funcionalidades à medida que iam sendo desenvolvidas. Isto permite poupar muito tempo no *debugging* da aplicação à procura do erro.

6.8 Resumo

Neste capítulo são descritos os detalhes mais importantes do processo de implementação. A forma como a arquitetura foi desenhada facilitou e acelerou este processo. São também descritas algumas otimizações que foram feitas de forma a aumentar a performance da solução.

As duas abordagens implementadas diferem apenas na camada de acesso a dados, permitindo que a camada de serviços seja independente da implementação. Esta particularidade possibilita, por exemplo, que sejam adicionadas novas abordagens com um esforço reduzido. No próximo capítulo são apresentados e descritos os testes realizados às duas abordagens implementadas.

Capítulo 7

Resultados

A fim de avaliar a qualidade da solução produzida, neste capítulo é apresentada a metodologia de testes aplicada, os testes efetuados, os resultados obtidos e a discussão desses mesmos resultados.

7.1 Metodologia

Tal como referido ao longo deste documento, um dos principais objetivos desta dissertação é estudar o impacto que as duas abordagens têm no desempenho da plataforma, nomeadamente no carregamento e disponibilização de séries temporais. Para isso foram pensados um conjunto de testes que visam medir quantitativamente esse mesmo impacto. Os testes efetuados são maioritariamente considerados testes de carga, pois têm como objetivo analisar o comportamento do sistema por meio do aumento da carga na base de dados.

Os testes realizados podem ser divididos em dois grupos: testes de carregamento ou inserção de dados e testes de seleção de dados. Os dados inseridos correspondem a medições, que são representados pela entidade *Measurement*, de vários *datasets*. Os testes foram realizados para um conjunto de valores que segue uma escala logarítmica de base dez com início em mil registos e fim em mil milhões de registos. Na prática isto significa que cada vez que a carga é aumentada, a nova carga a ser utilizada é calculada multiplicando a carga utilizada no último conjunto de testes por dez. Foram efetuados 50 ensaios para cada teste e calculada a sua média, salvo algumas exceções explicadas nas secções seguintes.

Para a medição do tempo de execução foi usado *nanoTime()*, em Java, em vez de *currentTimeMillis()*, pois é o método que nos garante maior precisão nos resultados. O objetivo do método *nanoTime()* é medir o tempo relativo decorrido entre dois instantes, enquanto que a finalidade do método *currentTimeMillis()* é indicar o tempo absoluto, desde o *Epoch*, segundo o relógio do sistema. A razão para não usar o método *System.currentTimeMillis()* é que o relógio do sistema não é perfeito e, ocasionalmente, precisa de ser corrigido. Esta correção pode acontecer manualmente

ou, no caso da maioria das máquinas, há um processo que corre continuamente e emite pequenas correções ao relógio do sistema. Uma vez que o propósito do *nanoTime()* é medir o tempo decorrido, este não é afetado por qualquer uma dessas pequenas correções[1].

Os testes foram executados num PC *Desktop* com as seguintes especificações: Intel® Core™ i5-4670K 6 MB *cache* 3.80 GHz, 16 GB de RAM DDR3 1600 MHz, 2x *Samsung* SATA HDD 500 GB 7200 RPM em RAID 0. Em termos de *software* a máquina corre Windows 8.1 64 b com PostgreSQL 9.3 e MongoDB 3.0.

7.2 Testes de Inserção

Os testes de inserção, correspondentes à narrativa de utilização US21, consistem na inserção de *Measurements* nos sistemas de base de dados PosgresSQL e MongoDB. Tal como referido anteriormente neste capítulo, os testes foram realizados para mil, dez mil, cem mil, um milhão, cem milhões e mil milhões de inserções. O que se pretende neste conjunto de testes é avaliar o tempo de inserção em cada uma das abordagens. De modo a que os testes fossem mais fiáveis, menos dispersos e mais homogéneos, foram corridos 50 ensaios para cada conjunto de testes, à exceção de cem milhões e mil milhões de inserções, uma vez que nestes casos o tempo de inserção é bastante elevado e não seria prático corrê-los tantas vezes. No entanto, a Figura 7.1 revela que os tempos medidos são praticamente constantes, apresentando pequenas variações entre os ensaios. Isso é comprovado pelo coeficiente de variação calculado, que corresponde a 2,7% no caso do MongoDB e a 3.2% PostgreSQL (ver Anexo A). Com estes dados é possível deduzir que para inserções de maior número (cem milhões e mil milhões) os resultados entre os ensaios seriam homogéneos, o que nos dá alguma segurança nos tempos obtidos para esse número elevado de inserções, ainda que não tenham sido realizados o mesmo número de ensaios.

Os dados usados nos testes de inserção são *Measurements* sintéticos, apenas criados para efeitos de teste. Não foram usados *Measurements* reais porque a leitura de um ficheiro HDF5 implica um *overhead* adicional, que não depende da abordagem e, por isso, não foi considerado. Além disso, a criação de *Measurements* sintéticos permite um maior controlo sobre os dados, pois as suas propriedades são definidas pelo teste a ser executado e não pelo que é lido de um ficheiro. Cada *Measurement* criado é associado a um *Dataset* calculado aleatoriamente entre os valores 1 e 4 inclusive. É importante que as condições iniciais se verifiquem para cada teste de inserção e, por isso, antes de cada teste a base de dados é colocada no seu estado inicial, isto é, sem séries temporais armazenadas.

A Figura 7.2 mostra-nos a evolução do tempo de inserção nas duas abordagens, à medida que se aumenta a carga. Pelo conjunto de pontos medidos foi possível, através de regressão polinomial, neste caso de grau 2, traçar uma linha de tendência que representa uma função que nos permite inferir a relação entre o número de inserções e o tempo de execução para cada uma das abordagens. Com estas funções calculadas é possível estimar qual será o tempo de inserção para números não testados.

[1] https://blogs.oracle.com/dholmes/entry/inside_the_hotspot_vm_clocks

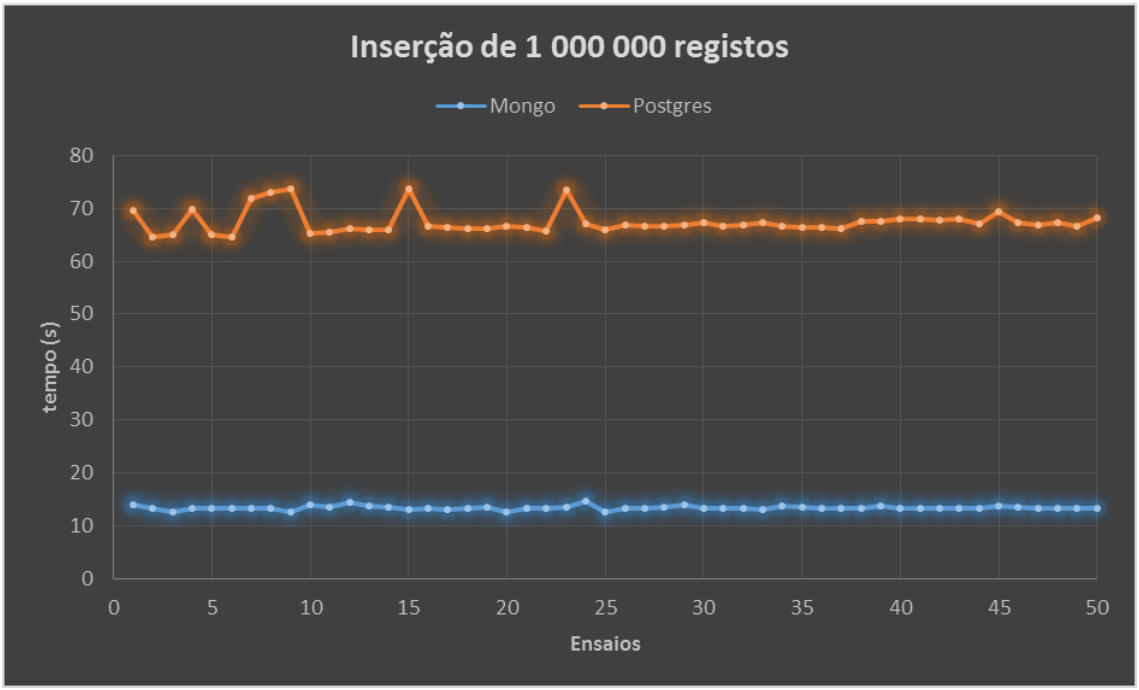

Figura 7.1: Gráfico de 50 Ensaios para Inserção de 1 Milhão de Registos.

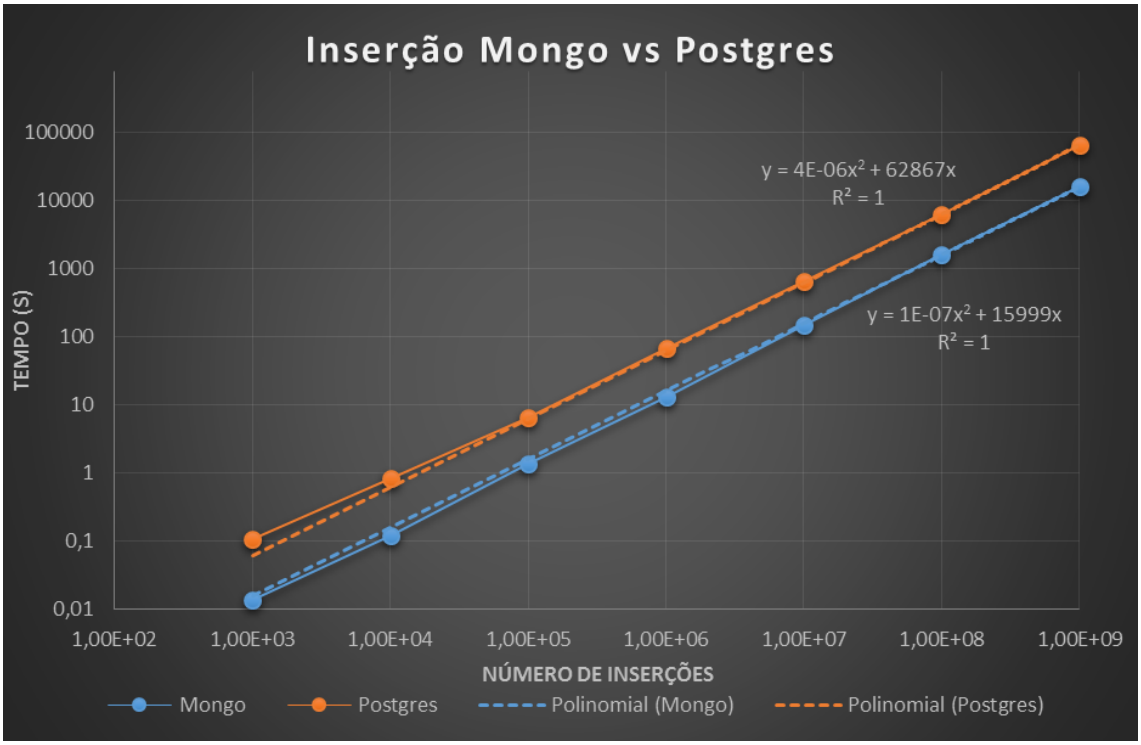

Figura 7.2: Gráfico de Comparação de Inserções entre as Duas Abordagens.

7.3 Testes de Seleção

Os testes de seleção consistem num conjunto de testes que visa medir o tempo que a base de dados demora a devolver os dados. Estes podem ser divididos em 3 grupos diferentes: testes de seleção total dos *Measurements*, testes de seleção de um único *dataset* e testes de seleção de um único *dataset* limitado a um número máximo de resultados. Tal como os testes de inserção, este conjunto de testes é realizado para um número de registos que segue uma escala logarítmica. Na prática, os testes de seleção são efetuados logo após o teste de inserção para o número de registos em questão.

Os testes de seleção, correspondentes às narrativas de utilização US34 e US35, foram desenhados de acordo com o que se estima que irá a acontecer na plataforma mais frequentemente, nomeadamente a visualização e descarregamento de séries temporais.

7.3.1 Seleção Total

O teste de seleção total consiste na seleção de todos os *Measurements* independentemente do *Dataset* a que pertencem. Embora esta situação dificilmente venha a acontecer, pois a visualização e descarregamento ocorre para um conjunto de *Measurements* pertencentes ao mesmo *Dataset*, serviu para avaliar o comportamento das duas abordagens em situações extremas.

Em termos práticos a seleção total das séries temporais consiste na *query SELECT * FROM measurements* em PostgreSQL ou *db.measurements.find()* em MongoDB. Embora os tempos medidos nestes testes não sejam aqueles que revelam maior importância, serviram para proceder a um conjunto de otimizações para que os dados pudessem ser todos selecionados. Num primeiro ensaio verificou-se que a camada *DAO* na primeira abordagem, mais propriamente a interface *MeasurementDAO*, não estava preparada para lidar com valores acima de um milhão de registos. Isto porque tentava carregar todos os dados em memória levando a plataforma, inevitavelmente, a ficar sem memória como nos demonstra a Figura 7.3.

A solução passou por carregar em memória pequenos conjuntos de dados, na ordem dos mil registos, e devolvê-los um de cada vez. A Figura 7.4 demonstra o estado da memória usada pela plataforma durante a seleção de 1 milhão de registos. Como é possível verificar pela análise do gráfico, o consumo de memória foi reduzido drasticamente e, ainda mais importante, mantém-se praticamente constante ao longo da seleção dos dados. Após esta otimização, foi possível executar os testes para todos os valores e obter um gráfico de comparação entre as duas abordagens, representado pela Figura 7.5.

Com o mesmo processo que foi aplicado nos testes de inserção, foi possível obter as funções, representadas no gráfico, que nos permitem estimar o tempo que demorará uma seleção total não testada.

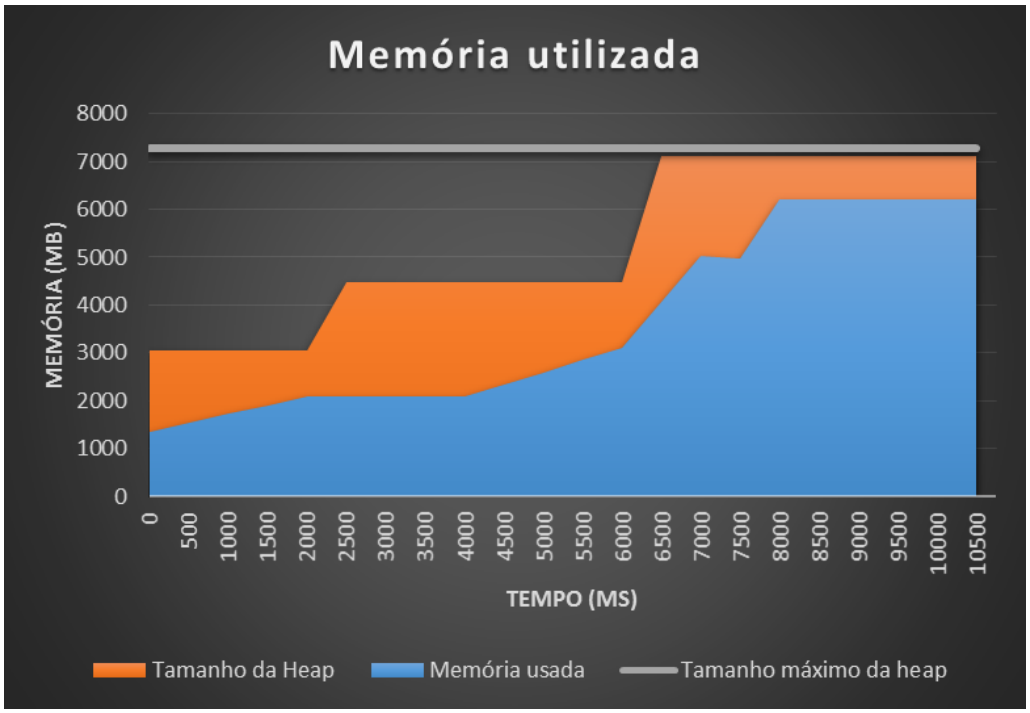

Figura 7.3: Memória Utilizada para a Seleção de 1 Milhão de Registos antes de Otimizações.

7.3.2 Seleção de um *Dataset*

O teste de seleção de um *Dataset* representa uma situação real que irá acontecer cada vez que um utilizador descarregar um *Dataset*. A seleção de um *Dataset* não implica apenas procurar todos os *Measurements* que lhe correspondem, mas também ordená-los por ordem crescente no tempo segundo a *query SELECT * FROM measurements where ds_id = ? ORDER BY timedate*, em PostgreSQL ou *db.measurements.find({dataset: ?}).sort({timedate: 1})* no caso de MongoDB. O caráter "?" corresponde ao id do *dataset* que queremos devolver. Os *Measurements* retornados têm que ser ordenados pelo seu *timestamp* para que o investigador consiga visualizar, no ficheiro, as medições de acordo com o que foi medido pelos *Equipments*.

A Figura 7.6 representa a comparação entre as duas abordagens para o retorno de um *Dataset* completo. É possível verificar pelo gráfico que a partir de 1 milhão de registos não existem valores medidos para o MongoDB. Isto leva-nos ao segundo problema encontrado durante a realização de testes: a partir de um milhão de linhas o MongoDB não consegue ordenar os *Measurements*, pois o conjunto de documentos, envolvidos na operação de ordenação não pode exceder 32 MB. Este problema é apenas resolvido com a criação de índices explicados na Secção 7.4.

7.3.3 Seleção Parcial de um *Dataset*

Uma seleção parcial de um *Dataset* será das situações mais frequentes na plataforma, pois corresponde à ação de visualizar um *Dataset*. Nestes casos, tal como explicado na Secção 6.6.2,

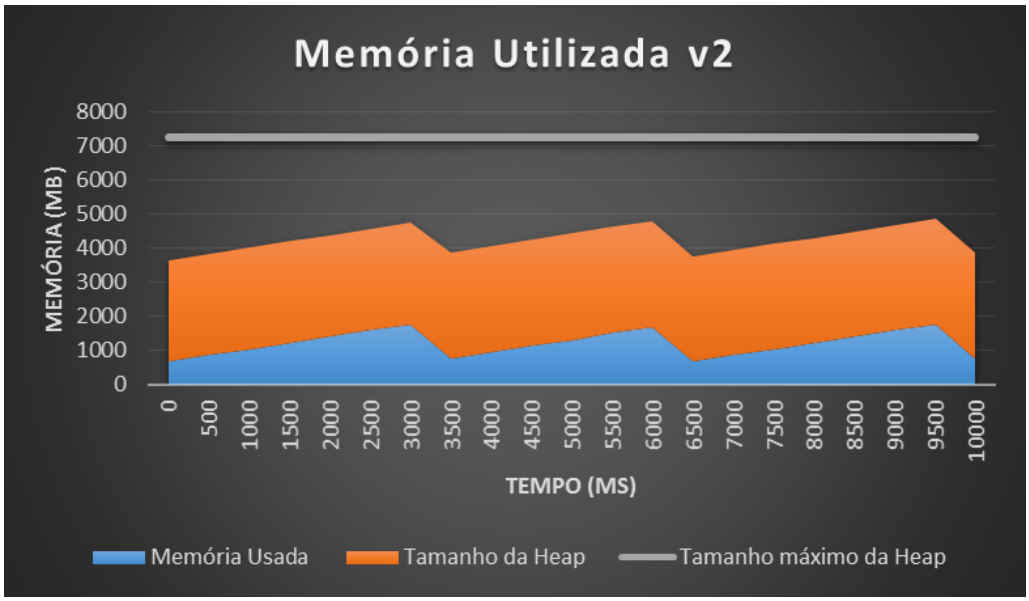

Figura 7.4: Memória utilizada para a Seleção de 1 Milhão de Registos depois de Otimizações.

só é devolvida uma parte *Dataset* correspondente ao que o utilizador solicitou. Por isso, *queries* do tipo *SELECT * FROM measurements where ds_id = ? ORDER BY timedate OFFSET :value LIMIT :max*, em PostgreSQL, e *db.measurements.find({dataset: ?}).sort({timedate: -1}).skip(value).maxResults(max)*, em MongoDB, serão muito frequentes. A variável *value* representa o número de registos que devem ser ignorados e a variável *max* o valor máximo de registos a devolver. Por exemplo, para uma tabela que apresenta 50 registos por cada página, se a primeira página fosse solicitada *value* teria o valor 0 e *max* o valor 50. Caso fosse solicitada a segunda página, *value* passaria para 50 e assim sucessivamente. A Figura 7.7 representa a comparação entre as duas abordagens para a seleção parcial de um *Dataset*.

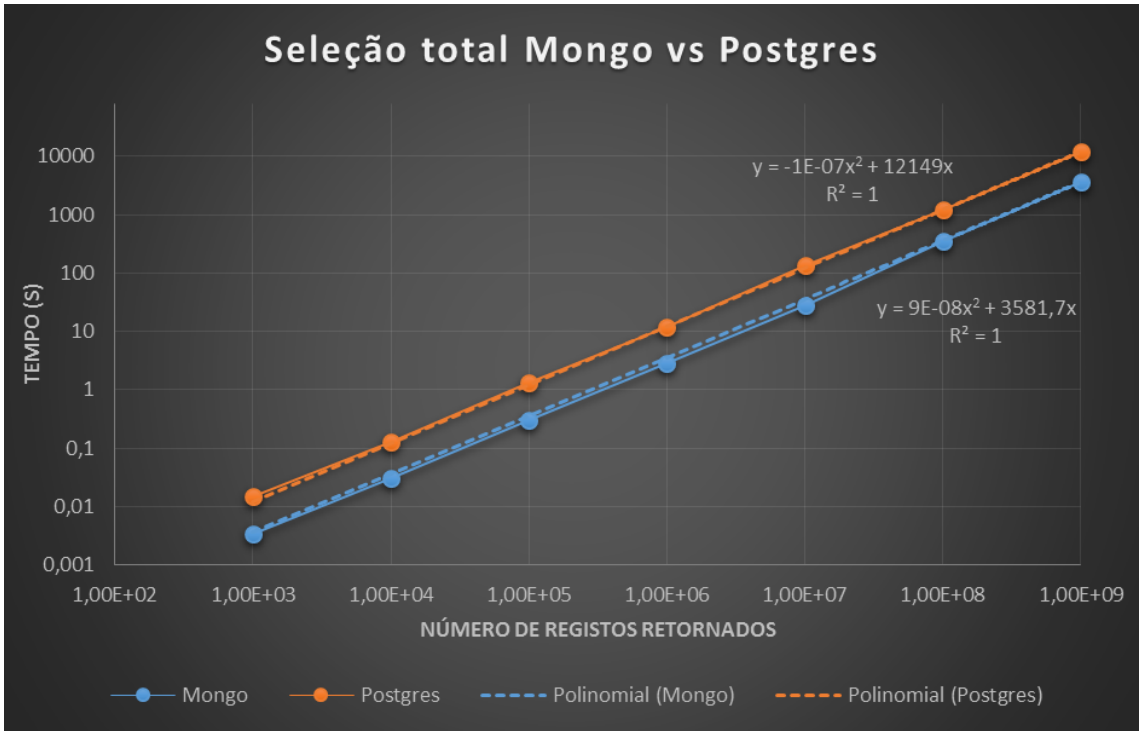

Figura 7.5: Gráfico de Comparação da Seleção total entre as Duas Abordagens.

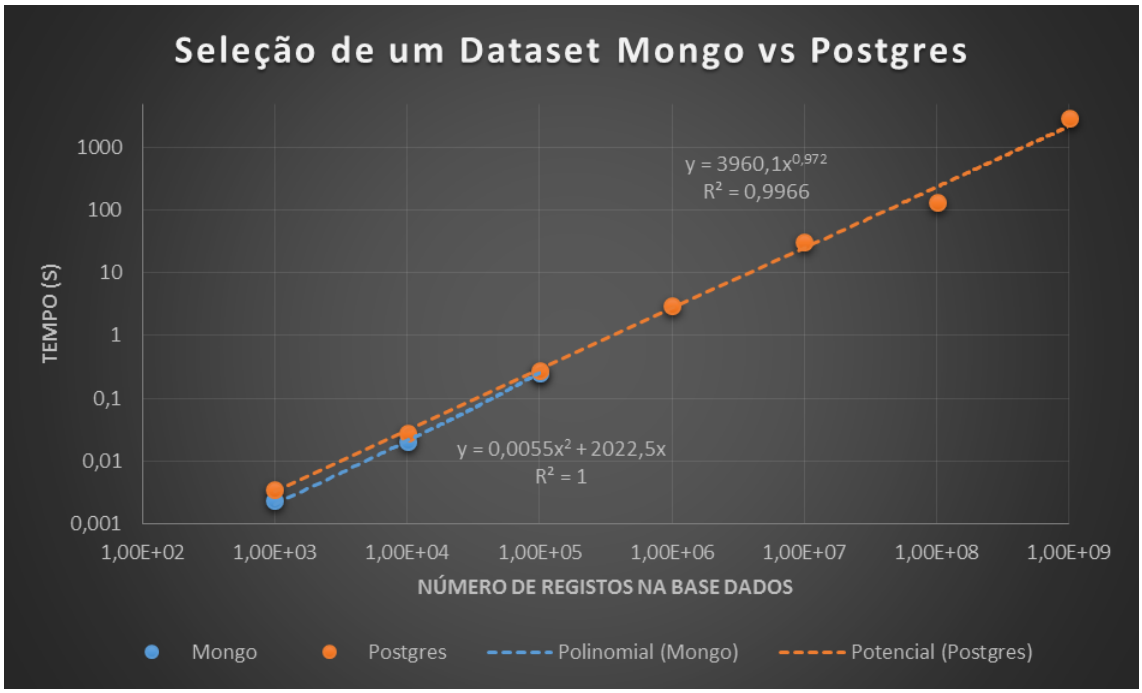

Figura 7.6: Gráfico de Comparação da Seleção de um *Dataset*.

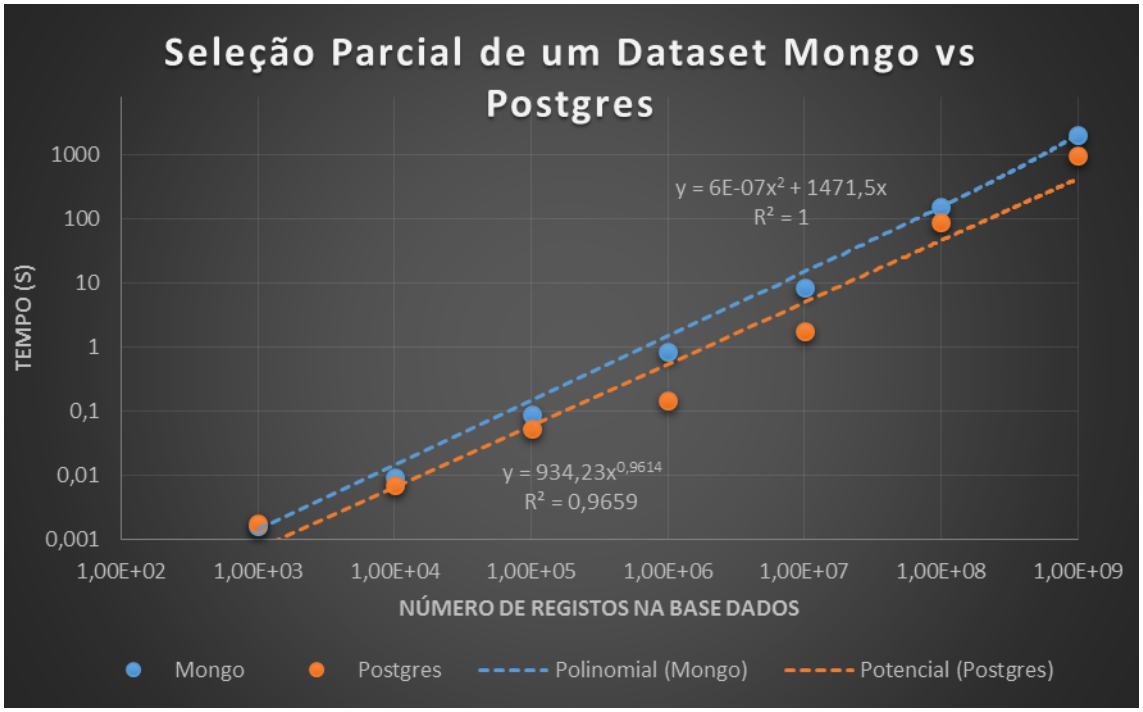

Figura 7.7: Gráfico de Comparação da Seleção Parcial de um *Dataset*.

7.4 Índices

A utilização de índices na solução foi sempre vista como uma otimização que teria que ser feita para melhorar o tempo de resposta das bases de dados. A ideia de otimização passou a ser uma obrigatoriedade a partir do momento em que o MongoDB não consegue ordenar um conjunto de documentos que tenham tamanho superior a 32 MB. Este tamanho é facilmente ultrapassável, uma vez que a plataforma armazena muitos milhões de *Measurements*. Além disso, as operações de ordenação estão sempre presentes, quer no descarregamento de *Datasets*, quer na visualização dos mesmos.

A utilização de índices não é um problema, mas em que campos os aplicar pode vir a ser. Foi necessário realizar um estudo para perceber em que propriedades da entidade *Measurement* faria sentido criar um índice. Para isso usou-se o *query planner* de cada um dos sistemas de base de dados usados. A Figura 7.8 representa os passos executados pelos 2 sistemas de base de dados quando um *Dataset* completo é selecionado. Como é possível verificar os passos executados são:

1. *Scan* sequencial para encontrar todos os *Measurements* que correspondem ao *Dataset* selecionado.

2. Ordenação por *timedate* dos *Measurements* encontrados.

Tanto o primeiro como o segundo passo podem ser extremamente custosos. No primeiro passo todos os *Measurements* têm que ser percorridos para encontrar aqueles que correspondem ao *Dataset* selecionado. No segundo passo, para um número elevado de *Measurements*, terá que ser feito um *external sort*[2].

O primeiro passo pode ser facilmente otimizado com a criação de um índice no campo que identifica o *dataset* a que o *Measurement* pertence. Assim, não será necessário percorrer todos os *Measurements* sequencialmente para encontrar os desejados. Isto não significa que o índice seja sempre usado, se o *query planner* achar que a leitura sequencial de todos *Measurements* é compensatória em relação a uma leitura separada dos *Measurements* desejados, será feita uma leitura sequencial. Na verdade, aceder a registos separadamente é muito mais caro do que lê-los sequencialmente, mas porque nem todos os blocos de memória que guardam a informação da entidade têm que ser visitados, o acesso aleatório acaba por ser mais rápido, dependendo do número de *Measurements* presentes na base de dados.

O segundo passo poderia ser otimizado da mesma forma que o primeiro: criar um índice, neste caso para o campo que representa a data de medição, *timedate*. Tanto o MongoDB como o PostgreSQL têm a capacidade de intersetar múltiplos índices para a mesma *query*. Assim, a ideia era que os índices fossem intersetados no momento da *query* para que não fosse necessário a ordenação dos *Measurements*. No entanto, a interseção de índices não é usada quando o índice usado para ordenação de resultados não faz parte dos índices usados nas cláusulas do *WHERE* em PostgreSQL ou do *find()* em MongoDB. Isto deve-se ao facto de o índice que seria usado

[2] http://web.eecs.utk.edu/~leparker/Courses/CS302-Fall06/Notes/external-sorting2.html

```
QUERY PLAN
text
Sort  (cost=6963.95..7025.85 rows=24760 width=138) (actual time=2
  Sort Key: timedate
  Sort Method: external sort  Disk: 3592kB
  ->  Seq Scan on measurement  (cost=0.00..3379.00 rows=24760 wid
        Filter: (ds_id = 1)
        Rows Removed by Filter: 75168
Total runtime: 33.314 ms
```

```
{
    "queryPlanner": {
        "winningPlan": {
            "stage": "SORT",
            "sortPattern": {
                "timedate": 1
            },
            "inputStage": {
                "stage": "COLLSCAN",
                "filter": {
                    "dataset": {
                        "$eq": 1
                    }
                },
                "direction": "forward"
            }
        }
    },
    "executionStats": {
        "nReturned": 24964,
        "executionTimeMillis": 233,
        "totalDocsExamined": 100000
    }
}
```

Figura 7.8: PostgreSQL vs MongoDB *Query Planner* para Seleção de um *Dataset*

na ordenação ter que ser todo percorrido para possibilitar que os resultados fossem ordenados[3] [4]. Como o *scan* de um índice adiciona *overhead* à operação, a interseção acaba por não ser usada. Como a seleção de um *Dataset* implica sempre uma ordenação dos *measurements* por *timedate*, optou-se por um índice composto que envolvesse os dois campos. No entanto, a ordem dos campos no índice é importante e teve que ser relacionada com os passos indicados na *seleção* de um *Dataset*. Como neste caso o que acontece primeiro será sempre a seleção dos *Measurements* de acordo com o seu *Dataset*, o campo mais à esquerda será o que identifica o *Dataset* seguido do campo *timedate*. Isto fará com que o índice esteja ordenado por *Dataset* e para cada *Dataset* por data. Caso o campo mais à esquerda fosse o *timedate*, os *Measurements* iriam estar ordenados por data e só depois por *Dataset*, o que não nos interessa, pois caso seja necessário retornar um *Dataset* teria que ser feito um *scan* completo ao índice e este acabaria por não ser usado. Por isso, a identificação em primeiro lugar dos passos que são efetuados por cada sistema de gestão

[3] https://jira.mongodb.org/browse/SERVER-3071
[4] http://www.postgresql.org/docs/9.4/static/indexes-bitmap-scans.html

de base de dados foi fulcral para determinar qual o índice correto a construir. Com o índice escolhido deixa de ser necessário percorrer todos os *Measurements* no primeiro passo e deixa também de ser necessário realizar uma ordenação no segundo passo. A Figura 7.9 revela os passos executados após aplicação do índice composto. Como é possível verificar o índice é agora usado para encontrar todos os *Measurements* do *Dataset* pretendido. No primeiro caso, sem índice, a Figura 7.8 mostra-nos que todos os registos foram acedidos. Com a utilização do índice apenas os *Measurements* que pertencem ao *Dataset* pretendido foram acedidos, o que representa uma redução de 75%.

```
QUERY PLAN
text
Sort  (cost=6684.41..6746.00 rows=24637 width=138) (actual time
  Sort Key: timedate
  Sort Method: external sort  Disk: 3568kB
  ->  Bitmap Heap Scan on measurement  (cost=679.35..3116.32 ro
        Recheck Cond: (ds_id = 1)
        ->  Bitmap Index Scan on measurement_ds_id_timedate_idx
              Index Cond: (ds_id = 1)
Total runtime: 24.169 ms
```


```
{
    "queryPlanner": {
        "winningPlan": {
            "stage": "FETCH",
            "inputStage": {
                "stage": "IXSCAN",
                "keyPattern": {
                    "dataset": 1,
                    "timedate": 1
                },
                "indexBounds": {
                    "dataset": [
                        "[1.0, 1.0]"
                    ]
                }
            }
        }
    },
    "executionStats": {
        "nReturned": 25186,
        "totalDocsExamined": 25186
    }
}
```


Figura 7.9: PostgreSQL vs MongoDB *Query Planner* com Índice Composto para Seleção de um *Dataset*.

Bitmap Heap Scan

Na Figura 7.9 é possível constatar que foi realizada uma ordenação por *timedate*, mesmo tendo sido usado o *índice* para encontrar os *Measurements* pretendidos. Isto acontece porque é usado um *Bitmap Index Scan* em vez de *Plain Index Scan*. Um *Plain Index Scan* obtém, através do índice, um

tuplo de cada vez e acede imediatamente à posição de memória em disco onde o tuplo se encontra. Um *Bitmap Heap Scan* obtém, através do índice, todas as posições de memória dos tuplos de uma vez só, ordena-os segundo a sua posição de memória e depois visita-os. A ordenação por posição de memória em disco é efetuada para minimizar o acesso aleatório a memória que é extremamente custoso. A ideia de utilizar uma leitura ordenada por posição de memória em disco, ao invés de uma organizada pelo índice, tem por objetivo minimizar o acesso aleatório a disco — todos os registos correspondentes a um determinado bloco serão lidos quando esse bloco é carregado para memória, evitando que o mesmo bloco tenha que ser lido várias vezes. Como a ordenação é feita por posição de memória dos tuplos e não por índice, a ordem do índice é perdida e, por isso, é necessário realizar uma nova ordenação por *timedate* quando os tuplos são retornados pelo *Bitmap Heap Scan*.

O *query planner* do PostgreSQL entende que é mais custoso fazer um *Plain Index Scan*, em que os dados já viriam ordenados, do que fazer um *Bitmap Heap Scan*, em que é necessário efetuar uma ordenação final caso a *query* inclua a cláusula *ORDER BY*. Já o *query planner* do MongoDB, a partir do momento da criação do índice, usa sempre um *Plain Index Scan*.

7.5 Discussão de Resultados

A Tabela 7.1 apresenta os resultados resumidos dos quatro conjuntos de testes efetuados sem utilização de índices na base de dados. Foi calculado o número de vezes que uma abordagem é mais rápida que a outra, sendo que "(P)" indica que a abordagem com PostgreSQL é a mais rápida e "(M)" que abordagem com MongoDB é a mais rápida. A unidade do tempo foi ajustada conforme o número de registos para que fosse mais legível. No entanto, os cálculos do *speedup* foram efetuados com os valores originais medidos em nanosegundos.

No que toca a inserções e seleção total de dados o MongoDB é claramente mais rápido, cerca de 4,5 vezes em média. Estes valores são facilmente explicáveis pela anatomia das duas bases de dados:

1. O MongoDB não tem que se preocupar com chaves estrangeiras, isto é, não necessita de verificar, para cada inserção, se o valor é realmente uma chave para outra tabela.

2. Não implementa as propriedades ACID e, por isso, não existe noção de transação nem de consistência estrita. A atomicidade da escrita é apenas garantida para cada documento e não para um conjunto de documentos.

3. Por omissão, o MongoDB usa *Acknowledged write concern*[5], isto é, a resposta ao pedido de escrita é dada antes de os dados serem efetivamente escritos em disco.

No conjunto de testes de seleção de um *Dataset* o desempenho das duas abordagens é praticamente a mesma, com ligeira vantagem para o MongoDB. Ainda assim, a partir de um milhão de

[5] http://blog.mongodirector.com/understanding-durability-write-safety-in-mongodb/

registos não existem valores medidos para o MongoDB. Tal como explicado, no MongoDB a operação de ordenação está limitada a um conjunto de documentos não superior a 32 MB. Para explicar o porquê da seleção de um *Dataset* ser ligeiramente mais rápida em MongoDB, a Figura 7.10 mostra-nos o tempo de seleção de um *Dataset* para as duas base de dados, não considerando o tempo de transmissão dos resultados para a plataforma. Como é possível observar, o algoritmo de ordenação usado pelo PostgreSQL é extremamente eficiente, pois executa em menos de 1 ms. Já o algoritmo usado pelo MongoDB revela-se muito mais lento, demorando cerca de 17 ms para ordenar. Esta diferença de valores não corresponde ao que foi verificado quando se realizaram os testes, havendo mesmo vantagem do MongoDB. Isto é explicado pelo tempo de transmissão de dados entre a base de dados e a plataforma. O MongoDB, embora seja muito mais lento a ordenar, consegue transmitir os resultados de forma mais rápida anulando assim a vantagem que o PostgreSQL tinha no fim da ordenação.

No conjunto de testes de ordenação parcial de um *Dataset*, os resultados a devolver foram limitados a 50. Depois da análise efetuada na Figura 7.10, os resultados obtidos eram expectáveis. Neste caso, as bases de dados só têm que retornar no máximo 50 registos, o que torna o I/O *overhead* muito mais reduzido quando comparado com o conjunto de testes anteriores. Como o problema do PostgreSQL esteve sempre relacionado com a transmissão de dados e não com a ordenação dos mesmos, esta abordagem torna-se melhor para um pequeno conjunto de dados. Na prática isto significa que o conjunto de dados a retornar não é significativo ao ponto de o MongoDB conseguir anular a vantagem criada pelo PostgreSQL.

Tabela 7.1: Resumo dos Resultados sem Índice.

Testes sem índices							
Inserção							
	1000 (ms)	10 000 (ms)	100 000 (s)	1 000 000 (s)	10 000 000 (min)	100 000 000 (min)	1 000 000 000 (h)
Postgres	108,32	858,69	6,67	67,39	10,88	105,38	18,54
Mongo	13,94	122,67	1,37	13,42	2,51	26,70	4,47
Speedup	7,77 (M)	7,00 (M)	4,88 (M)	5,02 (M)	4,33 (M)	3,95 (M)	4,15 (M)
Seleção Total							
	1000 (ms)	10 000 (ms)	100 000 (ms)	1 000 000 (s)	10 000 000 (s)	100 000 000 (min)	1 000 000 000 (h)
Postgres	14,89	125,77	1311,49	11,72	135,27	20,20	3,34
Mongo	3,37	30,38	304,28	2,91	28,59	6,00	1,02
Speedup	4,42 (M)	4,14 (M)	4,31(M)	4,03 (M)	4,73 (M)	3,37 (M)	3,28 (M)
Seleção Dataset							
	1000 (ms)	10 000 (ms)	100 000 (ms)	1 000 000 (s)	10 000 000 (s)	100 000 000 (min)	1 000 000 000 (min)
Postgres	3,47	27,72	276,56	3,05	31,83	2,25	48,78
Mongo	2,32	20,75	257,52	NaN	NaN	NaN	NaN
Speedup	1,49 (M)	1,34 (M)	1,07 (M)	NaN	NaN	NaN	NaN
Seleção Parcial Dataset							
	1000 (ms)	10 000 (ms)	100 000 (ms)	1 000 000 (ms)	10 000 000 (s)	100 000 000 (min)	1 000 000 000 (min)
Postgres	1,79	7,03	53,67	144,85	1,77	1,47	16,18
Mongo	1,63	9,40	88,69	867,68	8,63	2,56	33,93
Speedup	1,10 (M)	1,34 (P)	1,65 (P)	5,99 (P)	4,86 (P)	1,74 (P)	2,10 (P)

O MongoDB apresenta, no geral, melhores resultados. Ainda assim, para mil milhões de registos não é aceitável que um utilizador esteja, em média, 34 min à espera, no caso do MongoDB, ou 16 min à espera, no caso do PostgreSQL para conseguir visualizar os primeiros 50 *Measurements* de um *Dataset*. Além disso, a abordagem com MongoDB torna-se inviável se o número

```
QUERY PLAN
text
Sort  (cost=483.69..490.12 rows=2572 width=138) (actual time=1.567..1.614 rows=2572 loops=1)
  Sort Key: timedate
  Sort Method: quicksort  Memory: 780kB
  ->  Seq Scan on measurement  (cost=0.00..338.00 rows=2572 width=138) (actual time=0.011..1.108
        Filter: (ds_id = 1)
        Rows Removed by Filter: 7428
Total runtime: 1.764 ms
```


```
{
    "executionStats": {
        "executionTimeMillis": 17,
        "executionStages": {
            "stage": "SORT",
            "executionTimeMillisEstimate": 20,
            "inputStage": {
                "stage": "COLLSCAN",
                "filter": {
                    "dataset": {
                        "$eq": 1
                    }
                },
                "executionTimeMillisEstimate": 0
            }
        }
    }
}
```


Figura 7.10: Comparação do Tempo de Execução na Seleção de um *Dataset* entre as Duas Abordagens.

total de *Measurements* armazenados for igual ou superior a um milhão. A Tabela 7.2 apresenta os resultados resumidos dos quatro conjuntos de testes efetuados, com recurso a um índice composto detalhado na Secção 7.4.

Ao contrário do que se tinha verificado nos testes sem índice, o MongoDB torna-se agora muito mais rápido na seleção de um *Dataset* comparativamente com o PostgreSQL. À medida que o número de registos na base dados aumenta, essa diferença é ainda mais significativa chegando ao ponto do MongoDB conseguir ser, aproximadamente, 200 vezes mais rápido.

Com a criação do *índice*, o MongoDB, que inicialmente era mais lento na seleção parcial de um *Dataset*, torna-se agora, no mínimo, 4 vezes mais rápido do que o PostgreSQL.

O aumento notório de desempenho nas seleções totais e parciais de *Datasets*, fez com que os tempos para a visualização e descarregamento de um *Dataset* se tornassem aceitáveis. No entanto, para testes de seleção de um *Dataset* esse aumento de desempenho não se verifica no PostgreSQL.

A Tabela 7.3 mostra-nos a comparação entre a mesma base de dados com e sem índices sendo que "I" indica que a versão com índices é mais rápida. Como é possível observar, ao contrário do MongoDB, o desempenho em alguns dos casos até piorou em relação à versão sem índices. Para tentar perceber o que estava a acontecer, correu-se o *query planner*. A Figura 7.11 demonstra o plano utilizado para seleção de um *Dataset*. Como é visível, o *query planner* do PostgreSQL faz uso do *Bitmap Heap Scan* obrigando a que os resultados sejam ordenados por *timedate*. Para

Tabela 7.2: Resumo dos Resultados com Índice.

Testes com índices							
Inserção							
	1000 (ms)	10 000 (ms)	100 000 (s)	1 000 000 (s)	10 000 000 (min)	100 000 000 (min)	1 000 000 000 (h)
Postgres	129,44	653,87	6,28	65,45	11,17	294,00	348,61
Mongo	21,34	153,76	1,68	16,66	3,24	71,02	88,02
Speedup	6,07 (M)	4,25 (M)	3,75 (M)	3,93 (M)	3,45 (M)	4,14 (M)	3,96 (M)
Seleção Dataset							
	1000 (ms)	10 000 (ms)	100 000 (ms)	1 000 000 (ms)	10 000 000 (s)	100 000 000 (s)	1 000 000 000 (s)
Postgres	3,28	27,82	264,72	3386,07	35,88	154,08	1024,85
Mongo	1,30	8,34	77,45	770,16	3,81	4,53	5,28
Speedup	2,53 (M)	3,34 (M)	3,42 (M)	4,40 (M)	9,43 (M)	33,98 (M)	194,08 (M)
Seleção Parcial Dataset							
	1000 (ms)	10 000 (ms)	100 000 (ms)	1 000 000 (ms)	10 000 000 (ms)	100 000 000 (ms)	1 000 000 000 (ms)
Postgres	2,52	5,76	6,76	7,17	30,43	54,23	61,95
Mongo	0,67	0,68	0,69	0,76	0,89	1,29	4,18
Speedup	3,78 (M)	8,43 (M)	9,81 (M)	9,40 (M)	34,12 (M)	42,13 (M)	14,82 (M)

tentar averiguar qual é realmente a melhor estratégia, forçou-se a que o *query planner* usasse *Index Scan* em vez de *Bit Map Heap Scan* através dos comandos: *SET enable_bitmapscan TO OFF;* e *SET enable_seqscan TO OFF;*. Desta forma os resultados não serão ordenados por posição de memória, o que evitará que necessitem de ser ordenados depois de lidos. O resultado é apresentado na Figura 7.12. O tempo foi reduzido para mais de metade, o que significa que o *query planner* não está de todo a escolher o melhor plano, e isso está a ter impacto nos resultados. Só para um grande conjunto de dados, na ordem dos mil milhões, é que o algoritmo *Bitmap Heap Scan* começa a revelar-se realmente melhor escolha em relação a um *scan* completo à tabela.

Tabela 7.3: Resumo dos Resultados com e sem Índice em PostgreSQL.

Postgres sem Índices vs Postgres com Índices							
Seleção Dataset							
	1000 (ms)	10 000 (ms)	100 000 (ms)	1 000 000 (s)	10 000 000 (s)	100 000 000 (min)	1 000 000 000 (min)
Postgres c/index	3,28	27,82	264,72	3,39	35,88	2,57	17,08
Postgres s/index	3,47	27,72	276,56	3,05	31,83	2,25	48,78
Speedup	1,06 (I)	1,00	1,04 (I)	1,11	1,13	1,14	2,86 (I)
Seleção Parcial Dataset							
	1000 (ms)	10 000 (ms)	100 000 (ms)	1 000 000 (ms)	10 000 000 (s)	100 000 000 (s)	1 000 000 000 (s)
Postgresc/index	2,52	5,76	6,76	7,17	0,03	0,05	0,06
Postgres s/index	1,79	7,03	53,67	144,85	1,77	88,27	970,72
Speedup	1,41	1,22 (I)	7,94 (I)	20,21 (I)	58,29 (I)	1627,52 (I)	15669,54 (I)

```
QUERY PLAN
text
Sort  (cost=70581.96..71207.30 rows=250133 width=138) (actual time=358.816..422.907 rows=250140 loops=1)
  Sort Key: timedate
  Sort Method: external merge  Disk: 36144kB
  ->  Bitmap Heap Scan on measurement  (cost=5794.96..30199.62 rows=250133 width=138) (actual time=27.541..149.921
        Recheck Cond: (ds_id = 1)
        ->  Bitmap Index Scan on measurement_ds_id_timedate_idx  (cost=0.00..5732.42 rows=250133 width=0) (actual
              Index Cond: (ds_id = 1)
Total runtime: 433.929 ms
```

Figura 7.11: *Query Planner* para Seleção de um *Dataset* usando *Bitmap Heap Scan*.

A utilização de índices não apresenta só vantagens. Como era expectável, as inserções tornam-se mais lentas pois, para cada inserção, a base de dados tem que garantir a ordenação do índice.

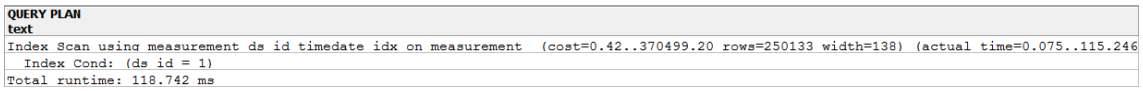

Figura 7.12: *Query Planner* para Seleção de um *Dataset* usando *Index Scan*.

A Tabela 7.4 apresenta o número de vezes que cada uma das bases de dados se torna mais lenta na inserção. Pese embora que para um número reduzido de inserções o impacto dos índices não seja significativo, para um número de inserções na ordem dos cem milhões o tempo de inserção aumenta drasticamente.

Tabela 7.4: Comparação de Inserção com e sem Índices.

Inserções	**# x Mongo mais lento**	**# x Postgres mais lento**
1000	1,5	1
10 000	1,3	1
100 000	1,2	1
1 000 000	1,2	1
10 000 000	1,3	1
100 000 000	2,7	2,8
1 000 000 000	19,7	18,8

Quando existe uma enorme quantidade dados para ser inseridos, tanto o MongoDB como o PostgreSQL aconselham que a criação do índice seja feita após a inserção dos dados de modo a acelerar o processo de inserção. Em relação ao PostgreSQL, é ainda aconselhada a desativação das chaves estrangeiras de forma a acelerar, ainda mais, o processo de inserção, pois assim não haveria necessidade, por parte da base de dados, de verificar se o valor da chave estrangeira correspondia a uma chave primária na tabela referenciada. Na prática, a otimização passaria por desativar a chave estrangeira que identifica o número do *dataset* na tabela *Measurements*, libertando o PostgreSQL de verificações adicionais.

Capítulo 8

Conclusões

O crescente aumento na produção de informação e a necessidade de armazenar e processar todos os dados é hoje um problema real em muitas áreas. A gestão de dados científicos tem um papel fulcral na rastreabilidade e reprodutibilidade dos dados que são produzidos no decurso de uma investigação científica.

O principal objetivo desta dissertação passou por desenhar uma plataforma *e-Science*, focada no armazenamento e disponibilização de séries temporais produzidas por sensores, que pudesse não só fomentar um maior grau de colaboração e partilha de dados entre equipas de diferentes disciplinas, mas também que disponibilizasse um conjunto de funcionalidades que dessem um maior suporte à investigação. No âmbito do projeto *Windscanner.eu* [GLPR14], foi implementado um protótipo dessa plataforma capaz de satisfazer as necessidades dos investigadores.

Numa primeira fase, com o intuito de perceber o impacto que a camada de armazenamento de dados poderia ter na plataforma, foram estudados os sistemas de base de dados que não seguem o modelo relacional, como uma alternativa aos RDBMS. Após um estudo aprofundado dessas bases de dados, foram definidas as funcionalidades que a plataforma devia implementar de acordo com o projeto *Windscanner.eu*.

Numa segunda fase, foi desenhada a arquitetura da plataforma que servirá de base para trabalho futuro. Embora o foco desta solução tenha sido a disponibilização e armazenamento de séries temporais, a arquitetura desenhada está preparada para albergar a maioria das funcionalidades que uma plataforma *e-Science* deverá disponibilizar.

Numa terceira fase, foi implementada a solução seguindo duas abordagens distintas: uma recorrendo a uma base de dados relacional e outra recorrendo à utilização de uma base de dados relacional juntamente com uma base dados NoSQL para armazenar apenas as séries temporais. Embora sejam duas abordagens distintas, a implementação foi feita de modo a que as camadas superiores não fiquem dependentes da implementação, isto é, as camadas que disponibilizam os serviços não são dependentes da abordagem e utilizam a camada de acesso a dados de forma transparente. Isto permitiu que o desenvolvimento fosse mais rápido e mais compacto, sem necessidade de alterações em várias camadas.

Os testes realizados permitiram perceber quais as limitações da solução e implementar algumas otimizações. O facto de se ter lidado com grandes quantidades de dados, na ordem dos mil milhões de registos, obrigou a que os problemas de escalabilidade fossem evidentes. Esses problemas só puderam ser resolvidos com estudo dos mecanismos internos inerentes às bases de dados utilizadas.

A plataforma é, efetivamente, inovadora pois combina o melhor dos dois sistemas de base de dados: uma base de dados não relacional que garante bom desempenho no acesso a grandes quantidades de dados e é facilmente escalável e uma base de dados relacional que guarda toda a meta-informação e permite a utilização do poderoso sistema de *queries* para interseção ou combinação de informação que descreve as séries temporais. Este trabalho acabou por superar os objetivos pois, com os problemas inesperados de escalabilidade que foram aparecendo, foi possível compreender o funcionamento interno dos dois sistemas de base de dados e, com isso, fazer algumas otimizações que não estavam previstas.

A base de dados NoSQL utilizada, MongoDB, revelou um desempenho muito superior ao PostgreSQL. No entanto, nem sempre é vantajoso perder as funcionalidades de um RDBMS em troca de um sistema de base de dados com maior desempenho. Por isso, após os estudos efetuados e os testes realizados, pode-se concluir que os RDBMS e as bases de dados NoSQL podem coexistir no mesmo sistema, tirando-se assim partido das vantagens de cada uma.

Os resultados retirados dessa dissertação foram bastante satisfatórios pois foi possível construir uma solução que é, efetivamente, escalável e apresenta uma excelente desempenho na disponibilização de dados, mesmo lidando com grandes quantidades de informação. No entanto, este protótipo é apenas um passo naquilo que poderá a vir a ser esta plataforma. Os estudos realizados deverão ser uma motivação para o enorme potencial que esta plataforma apresenta e, ao mesmo tempo, servir de base para implementações futuras.

8.1 Trabalho Futuro

De acordo com Shannon Bohle, "e-Science é a aplicação de tecnologias de computação para a realização de investigação científica moderna, incluindo a preparação, a experimentação, a recolha de dados, divulgação de resultados, armazenamento a longo prazo e acessibilidade a todos os materiais produzidos através do processo científico" [Boh13]. A solução implementada foi apenas um caso de estudo que servirá de suporte para implementações futuras. Todas as componentes que fazem parte da definição de *e-Science* deverão ser implementadas futuramente.

Para resolver o problema de escalabilidade dos RDBMS foi usado MongoDB. O MongoDB é apenas um de muitos exemplos de base de dados NoSQL que poderiam ter sido usados. Seria interessante implementar outras abordagens, com recurso a outras bases de dados NoSQL, que seguem um modelo de dados diferente do MongoDB. A estrutura escolhida para cada documento armazenado no MongoDB não implica que tenha sido a melhor. Seria também interessante testar outras estruturas e comparar o desempenho de cada uma.

Durante a implementação, surgiu a dúvida se seria mesmo necessário guardar o conteúdo dos ficheiros HDF5 na base de dados. Seria interessante implementar uma abordagem que apenas guarde a meta-informação na base de dados e mantenha os ficheiros com as séries temporais no sistema de ficheiros. Isto traria um melhoramento drástico na inserção dos dados, visto que deixava de ser necessário inserir *Measurements* na base de dados. Por outro lado, teriam que ser criados um conjunto de funções de manipulação dos ficheiros HDF5 de modo a que fosse possível realizar as opções de filtragem disponíveis na plataforma. A API implementada para a visualização de dados descrita na Secção 6.6.2 não foi pensada para este caso, mas poderia ser usada para este fim. A ideia seria desenhar um serviço *Web* que implementava um conjunto de funções para manipulação de ficheiros e, ao mesmo tempo, expunha esta API REST para troca de dados entre o cliente e o servidor.

Não foi possível, durante a fase de testes, escalar a base de dados horizontalmente devido a limitações de *hardware*. Para tornar a plataforma altamente disponível, a base de dados terá que ser escalada horizontalmente, isto é, acrescentar outras máquinas e distribuir os dados da base de dados pelos vários nós. A ideia seria fazer uso de uma API desenvolvida em trabalhos anteriores [Car14] que é capaz de distribuir os dados de forma transparente, isto é, sem que fosse necessário desenvolver novos módulos para manipulação dos dados nos vários nós.

Referências

[Amb04] Scott W. Ambler. *The Object Primer: Agile Model-Driven Development with UML*. Cambridge University Press, 3rd edition, 2004.

[Ber08] Francine Berman. Got Data? A Guide to Data Preservation in the Information Age: a guide to data preservation in the information age. *Communications of the ACM*, 51(12):50–56, 2008. doi:10.1145/1409360.1409376.

[Boh13] Shannon Bohle. What is E-science and How Should it be Managed?, 2013. URL: http://www.scilogs.com/scientific_and_medical_libraries/what-is-e-science-and-how-should-it-be-managed/.

[Bou05] Ronald Bourret. XML and Databases, 2005. URL: http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[Bre00] Eric Brewer. Towards Robust Distributed Systems, 2000. URL: http://www.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf, doi:10.1145/343477.343502.

[Bre12] E. Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(February):23–29, 2012. doi:10.1109/MC.2012.37.

[Car14] Carlos Carvalheira. Scalable High-Performance Platform for e-Science. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2014.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah a. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes e Robert E. Gruber. Bigtable: A distributed storage system for structured data. *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pages 205–218, 2006. doi:10.1145/1365815.1365816.

[Cod70] E F Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970. doi:10.1145/357980.358007.

[Col10] JP Collins. Sailing on an ocean of 0s and 1s. *Science*, 327(March):1455–1456, 2010.

[Dep12] Department of Education Science and Training. Australian National Data Service (ANDS) BUSINESS PLAN 2012-13. pages 1–118, 2012.

[EERS15] Rob Baxter Epcc, Alison Kennedy Epcc, Johannes Reetz Rzg e Mark Van De Sanden. D2.2.2: EUDAT Services Rolling Plan 2012-2015 (update). Technical report, 2015.

[ESR15] Rob Baxter Epcc, Mark Van De Sanden Surfsara e Johannes Reetz Rzg. D2.2.3: EUDAT Services Rolling Plan 2012-2015 (final). Technical report, 2015.

[Eur06] European Strategy Forum. Strategy Report on Research Infrastructures - Roadmap 2006. Technical report, 2006. URL: http://ec.europa.eu/research/infrastructures/pdf/esfri/esfri_roadmap/roadmap_2006/esfri_roadmap_2006_en.pdf.

[Eur08] European Strategy Forum. Strategy Report on Research Infrastructures - Roadmap 2008. Technical report, 2008. URL: http://ec.europa.eu/research/infrastructures/pdf/esfri/esfri_roadmap/roadmap_2008/esfri_roadmap_update_2008.pdf.

[Eur10] European Commission. Riding the wave: How Europe can gain form the rising tide of scientific data. Final report of the high Levlel Expert Group on Scientific Data. Technical Report October, 2010.

[Eur11] European Strategy Forum. Strategy Report on Research Infrastructures - Roadmap 2010. Technical report, 2011. URL: http://ec.europa.eu/research/infrastructures/pdf/esfri-strategy_report_and_roadmap.pdf, doi:10.2777/23127.

[Eur13] European Commission. Assessing the projects on the ESFRI roadmap A high level expert group report. Technical report, 2013. URL: http://ec.europa.eu/research/infrastructures/pdf/KI0213337ENC_WEB.pdf.

[GL02] Seth Gilbert e Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33:51, 2002. doi:10.1145/564585.564601.

[GLPR14] Filipe Gomes, João Correia Lopes, José Laginha Palma e Luís Frölen Ribeiro. WindS@UP: The e-Science Platform for WindScanner.eu. *Journal of Physics: Conference Series*, 524(The Science of Making Torque from Wind 2014 (TORQUE 2014) 18–20 June 2014, Copenhagen, Denmark), 2014.

[Gra81] Jim Gray. The Transaction Concept: Virtues and Limitations. *Proceedings of the 7th International Conference on Very Large Data Bases*, (1):144–154, 1981. doi:10.1.1.59.5051.

[Han14] Hilary Hanahoe. A recipe for data: lessons from EUDAT. *Information Today Europe*, (March), 2014. URL: http://www.infotoday.eu/Articles/Editorial/Featured-Articles/A-recipe-for-data-lessons-from-EUDAT-95228.aspx.

[HR83] Theo Haerder e Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983. doi:10.1145/289.291.

[HT03] Tony Hey e Anne Trefethen. The Data Deluge: An e-Science Perspective. (January 2003):809–824, 2003.

[Jac09] Adam Jacobs. The Pathologies of Big Data. *Queue*, 7:10, 2009. doi:10.1145/1563821.1563874.

[KDBHk14] Kostas Kavoussanakis, Emanuel Dima, Rob Baxter e Carl Johan Hå kansson. EUDAT user documentation: EUDAT Prime. Technical Report June, 2014.

[KMB+08] Gerhard Klimeck, Michael Mclennan, Sean B. Brophy, George B. Adams e Mark S. Lundstrom. NanoHUB.org: Advancing education and research in nanotechnology. *Computing in Science and Engineering*, 10(2008):17–23, 2008. doi:10.1109/MCSE.2008.120.

[Lai09a] Eric Lai. No to SQL? Anti-database movement gains steam, 2009. URL: http://www.computerworld.com/article/2526317/database-administration/no-to-sql--anti-database-movement-gains-steam.html.

[Lai09b] Eric Lai. Researchers: Databases still beat Google's MapReduce, 2009. URL: http://www.computerworld.com/article/2524259/database-administration/researchers--databases-still-beat-google-s-mapreduce.html.

[LWE+13] Damien Lecarpentier, Peter Wittenburg, Willem Elbers, Alberto Michelini, Riam Kanso, Peter Coveney e Rob Baxter. EUDAT: A New Cross-Disciplinary Data Infrastructure for Science. *International Journal of Digital Curation*, 8(1):279–287, 2013. doi:10.2218/ijdc.v8i1.260.

[MAI14] Mohamed a. Mohamed, Obay G. Altrafi e Mohammed O. Ismail. Relational vs. NoSQL Databases: a survey. *IJCIT International Journal of Computer and Information Technology*, 03(03):598–601, 2014.

[Mak08] Gary Mak. *Spring Recipes: A Problem-Solution Approach*. 1st edition, 2008.

[MK10] Michael Mclennan e Rick Kennell. HUBzero: A platform for dissemination and collaboration in computational science and engineering. *Computing in Science and Engineering*, 12:48–52, 2010. doi:10.1109/MCSE.2010.41.

[Mon14] MongoDB Team. Top 5 Considerations When Evaluating NoSQL Databases Table of Contents, 2014.

[Oba09] Dare Obajanso. Building Scalable Databases: Denormalization, the NoSQL Movement and Digg, 2009. URL: http://www.25hoursaday.com/weblog/2009/09/10/BuildingScalableDatabasesDenormalizationTheNoSQLMovementAndDigg.aspx.

[pt10] ANDS project team. ANDS Discovery Services and the Australian Research Data Commons. Technical report, 2010.

[pt15] ANDS project team. Better Data for Australian Research. Technical report, 2015. URL: http://www.ands.edu.au/andsbrochurejan2015.pdf.

[RSFWH98] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood e Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2:33–38, 1998. doi:10.1109/4236.656066.

[Sad14] Pramod Sadalage. NoSQL Databases: An Overview, 2014. URL: http://www.thoughtworks.com/insights/blog/nosql-databases-overview.

[The07] The ANDS Technical Working Group. Towards the Australian data commons. Technical Report October, 2007. URL: http://ands.org.au/towardstheaustraliandatacommons.pdf.

[Uni13] University of Porto. Windscanner.eu — Establishment of methodologies for data quality assurance and exchange. Technical report, 2013.

[WL12] Michael Witt e West Lafayette. Refactoring HUBzero for Linked Data Categories and Subject Descriptors. In *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries*, pages 149–152, 2012.

[WS10] Silvan Weber e Christof Strauch. NoSQL Databases. *Lecture Notes Stuttgart Media*, pages 1–8, 2010.

Anexo A

Tabelas de Resultados

Neste anexo são apresentados os resultados obtidos para todos os testes realizados. Os tempos apresentados encontram-se em nanosegundos.

A.1 Inserção

Tabela A.1: Tabela Resumo Inserções sem Índice

# Registos	Mongo	Postgres	Speedup Mongo	Coef Var. Mongo	Coef Var. Postgres
1000	13939520,36	108315900,4	7,77	12,10	14,60
10000	122666625,3	858687664,2	7,00	6,69	11,39
100000	1366401559	6674732553	4,88	23,08	3,82
1000000	13421994520	67393000288	5,02	2,70	3,24
10000000	1,50825E+11	6,52701E+11	4,33	3,06	1,08
100000000	1,60192E+12	6,3227E+12	3,95	3,85	7,24
1000000000	1,60987E+13	6,67295E+13	4,15	3,17	3,41

Tabela A.2: Tabela Resumo Inserções com Índice

# Registos	Mongo	Postgres	Speedup Mongo	Coef Var. Mongo	Coef Var. Postgres
1000	21340289	129437868,3	6,07	58,31	27,76
10000	153763132	653872846	4,25	6,25	6,84
100000	1677222224	6282880331	3,75	21,27	8,91
1000000	16660806695	65453664104	3,93	6,74	1,52
10000000	194431358350	670445576561	3,45	2,71	1,22
100000000	4261027006924	17640076082505	4,14	0,00	0,00
1000000000	316863000000000	1255008000000000	3,96	0	0

A.2 Seleção Total

Tabela A.3: Tabela Resumo de Seleção Total

# Registos	Mongo	Postgres	Speedup Mongo	Coef Var. Mongo	Coef Var. Postgres
1000	3370027,52	14886688,16	4,42	2,99	2,91
10000	30379760,5	125770682,6	4,14	4,16	9,51
100000	304281492,4	1311493571	4,31	24,56	10,48
1000000	2906294116	11718041772	4,03	2,65	1,07
10000000	28594225503	1,35274E+11	4,73	0,56	3,80
100000000	3,59854E+11	1,21215E+12	3,37	0,00	0,00
1000000000	3,67016E+12	1,2029E+13	3,28	0,00	0,00

A.3 Seleção de um *Dataset*

Tabela A.4: Tabela Resumo de Seleção de um *Dataset* sem Índice

# Registos	Mongo	Postgres	Speedup Mongo	Coef Variação Mongo	Coef variação Postgres
1000	2322865	3467529	1,49	10,89	9,79
10000	20745125	27719010	1,34	5,30	3,17
100000	257523414	276555253	1,07	4,29	4,43
1000000	NaN	3049875263	NaN	NaN	0,68
10000000	NaN	31834092133	NaN	NaN	0,80
100000000	NaN	135240736287	NaN	NaN	14,10
1000000000	NaN	2926691326621	NaN	NaN	0

Tabela A.5: Tabela Resumo de Seleção de um *Dataset* com Índice

# Registos	Mongo	Postgres	Speedup Mongo	Coef Var. Mongo	Coef Var. Postgres
1000	1298503,1	3284838	2,53	6,33	8,46
10000	8337233	27822572	3,34	14,52	6,06
100000	77453978	264715341	3,42	4,75	3,60
1000000	770162088	3386067391	4,40	2,56	0,89
10000000	3806646808	35880945979	9,43	0,95	1,75
100000000	4534194104	154077474204	33,98	5,95	1,62
1000000000	5280624452	1024848947039	194,08	2,16	2,26

A.4 Seleção Parcial de um *Dataset*

Tabela A.6: Tabela Resumo de Seleção Parcial de um *Dataset* sem Índice

# Registos	Mongo	Postgres	Speedup Mongo	Speedup Postgres	Coef Var. Mongo	Coef Var. Postgres
1000	1630024	1789387	1,10	0,91	6,85	5,58
10000	9403307	7034768	0,75	1,34	5,89	14,37
100000	88694915	53666916	0,61	1,65	5,81	12,86
1000000	867676452	144849896	0,17	5,99	4,56	2,26
10000000	8629690097	1774076969	0,21	4,86	3,91	0,46
100000000	153475774225	88265480250	0,58	1,74	0	0
1000000000	2036010439665	970717733050	0,48	2,10	0	0

Tabela A.7: Tabela Resumo de Seleção Parcial de um *Dataset* com Índice

# Registos	Mongo	Postgres	Speedup Mongo	Coef Var. Mongo	Coef Var. Postgres
1000	666887	2521906	3,78	5,00	5,00
10000	683271	5763288	8,43	6,24	2,65
100000	689501	6763285	9,81	9,63	1,75
1000000	762562	7165569	9,40	12,17	21,53
10000000	891852	30433127	34,12	9,45	20,00
100000000	1287132	54232989	42,13	0,00	0,00
1000000000	4180979	61949349	14,82	7,62	2,75

Anexo B

Casos de Utilização

Neste anexo são apresentados *screenshots* de exemplos de utilização da plataforma.

B.1 Campanha

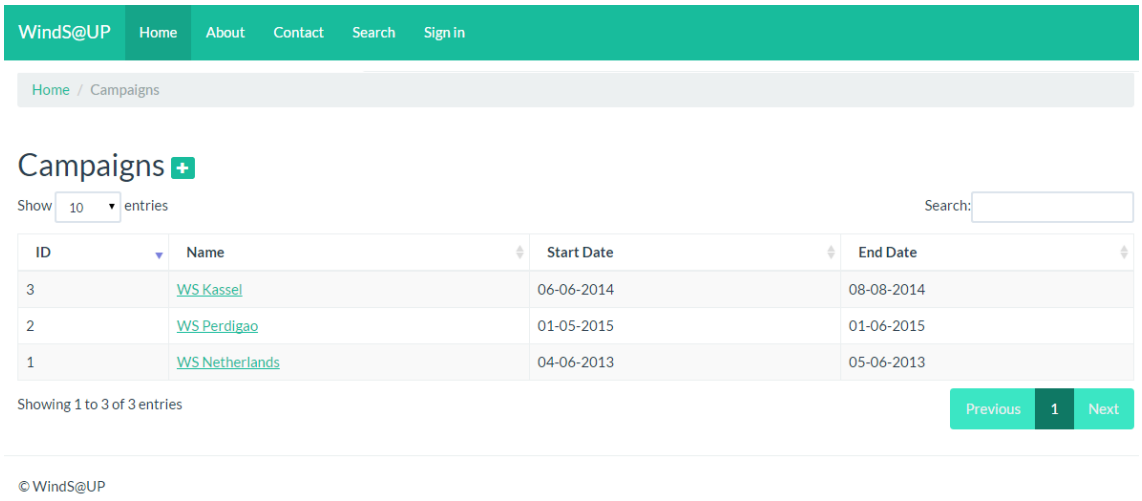

Figura B.1: Visualização de Todas as Campanhas.

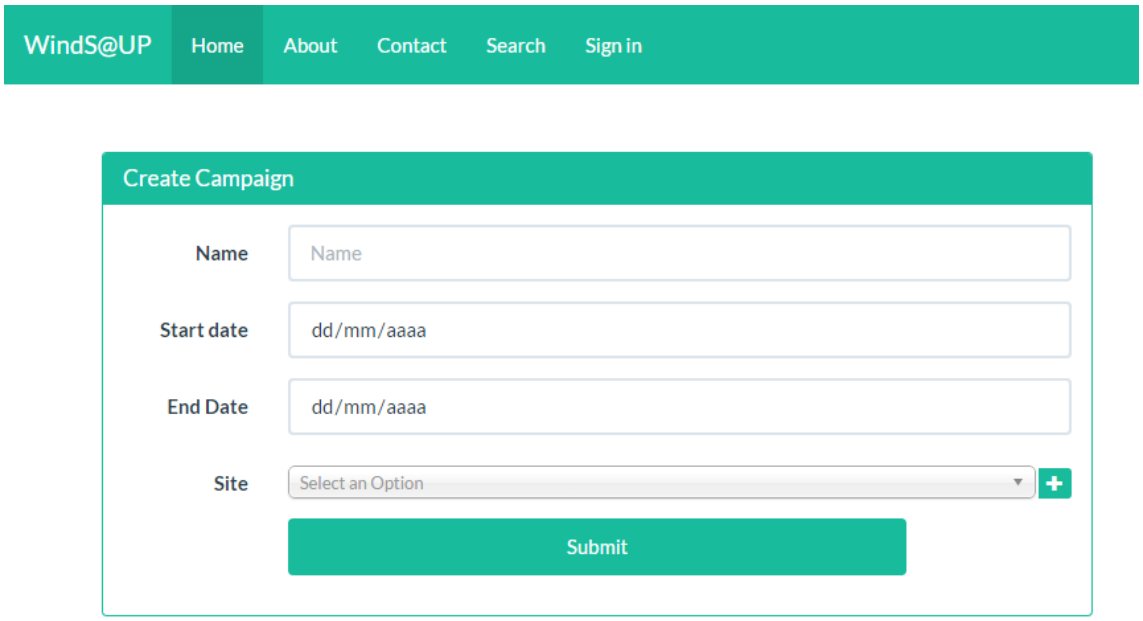

Figura B.2: Criação de uma Campanha.

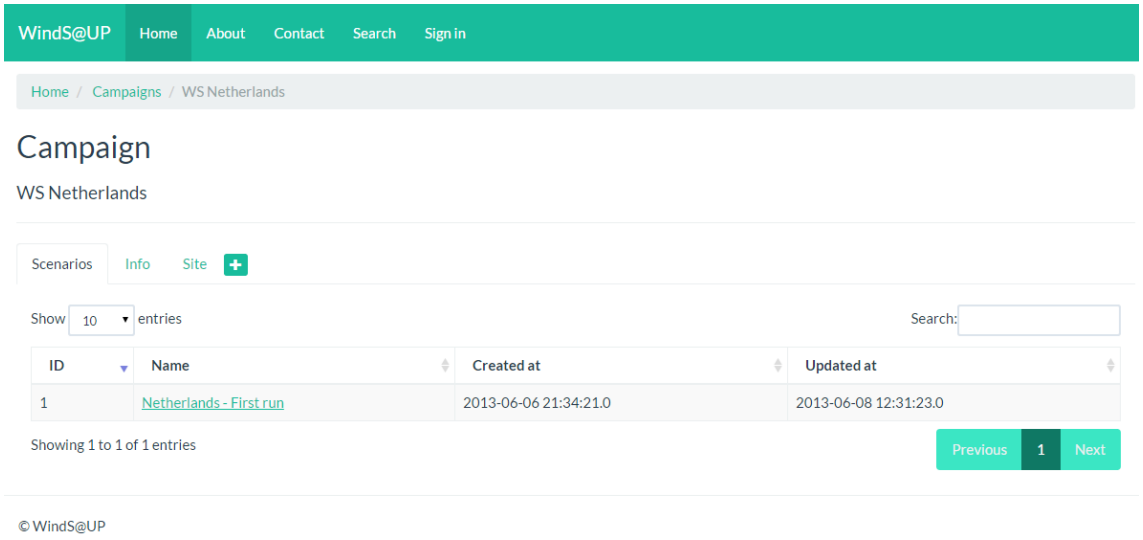

Figura B.3: Detalhes de uma Campanha.

B.2 Cenário

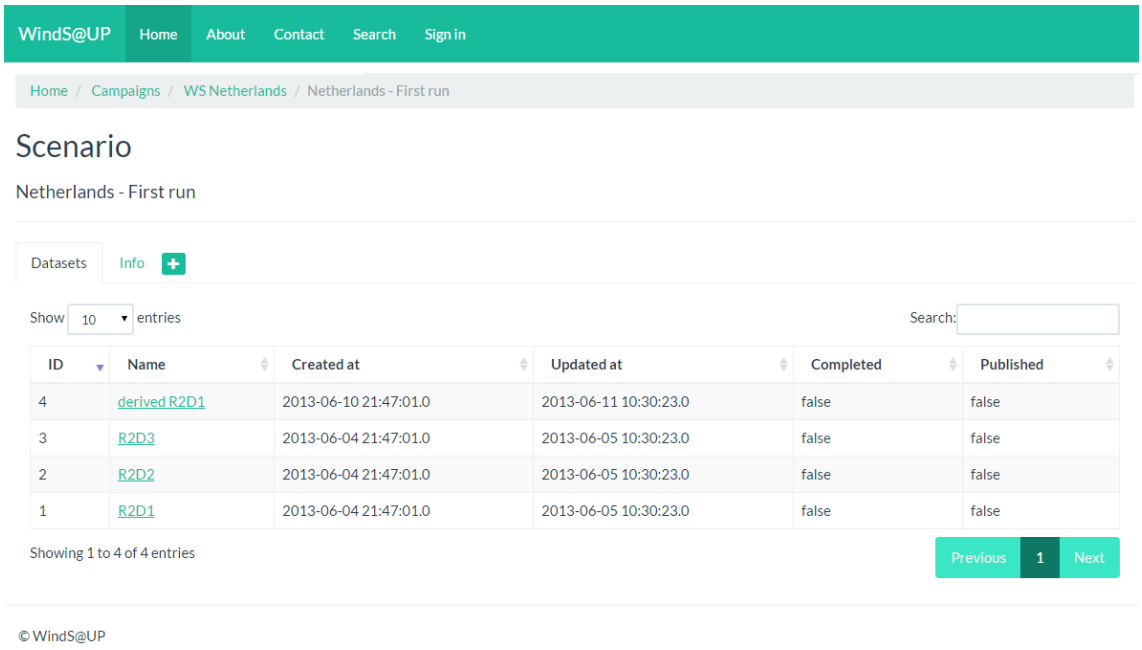

Figura B.4: Detalhes de um Cenário.

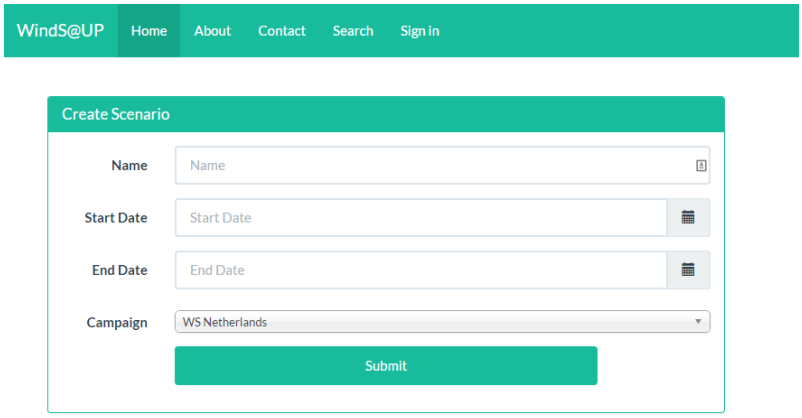

Figura B.5: Criação de um Cenário.

B.3 *Dataset*

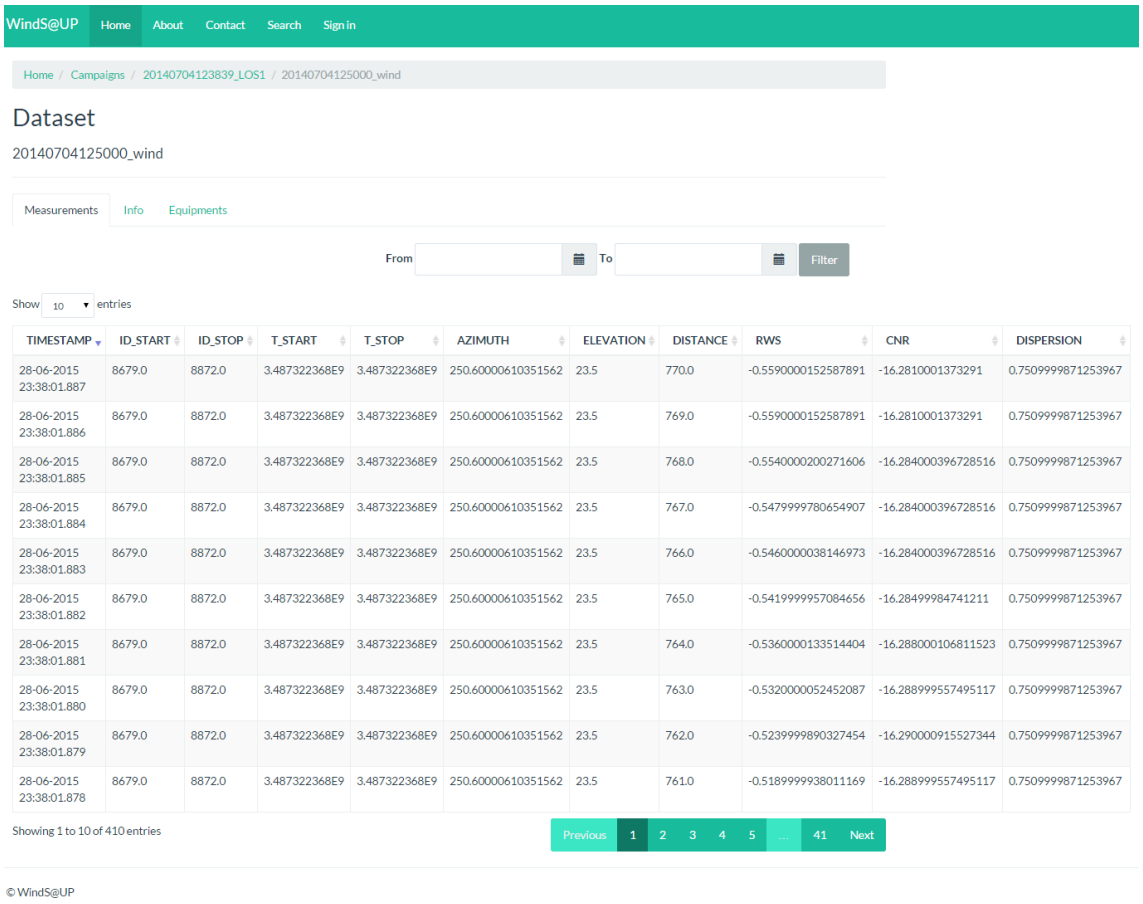

Figura B.6: Detalhes de um *Dataset*.

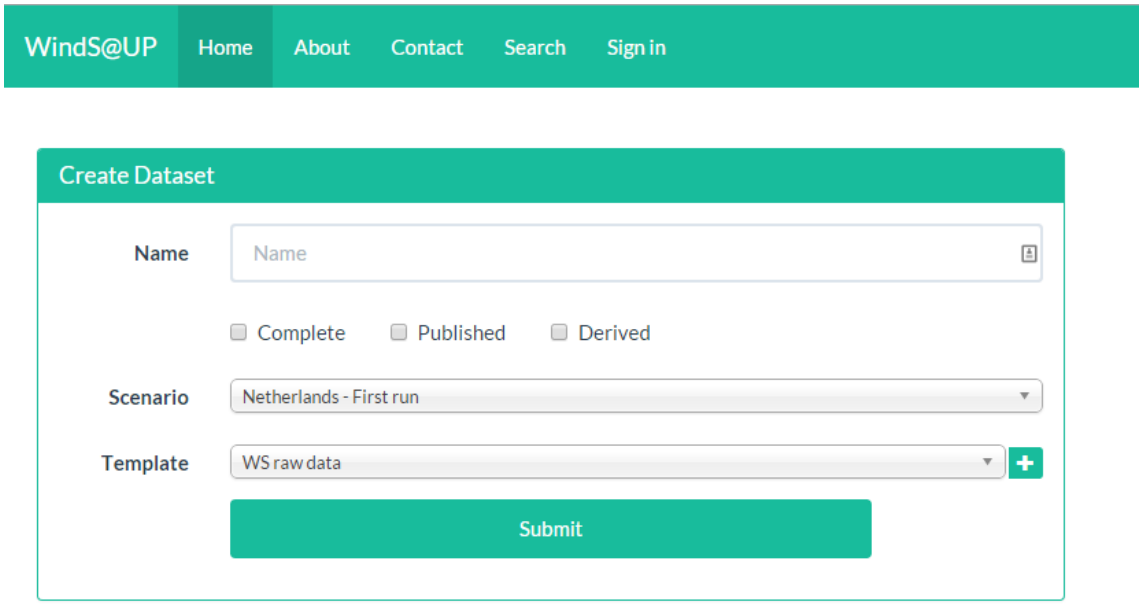

Figura B.7: Criação de um *Dataset*.

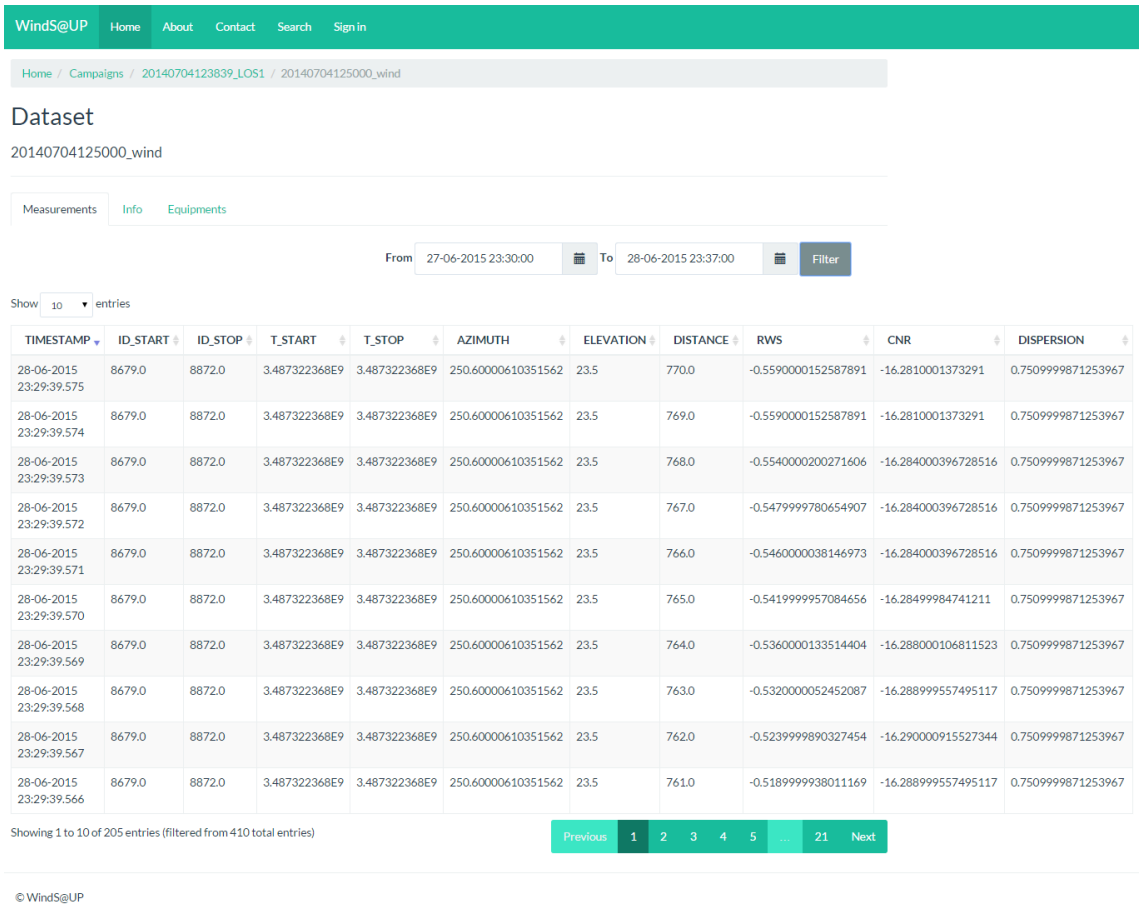

Figura B.8: Filtragem de um *Dataset* por Data.

B.4 Autenticação

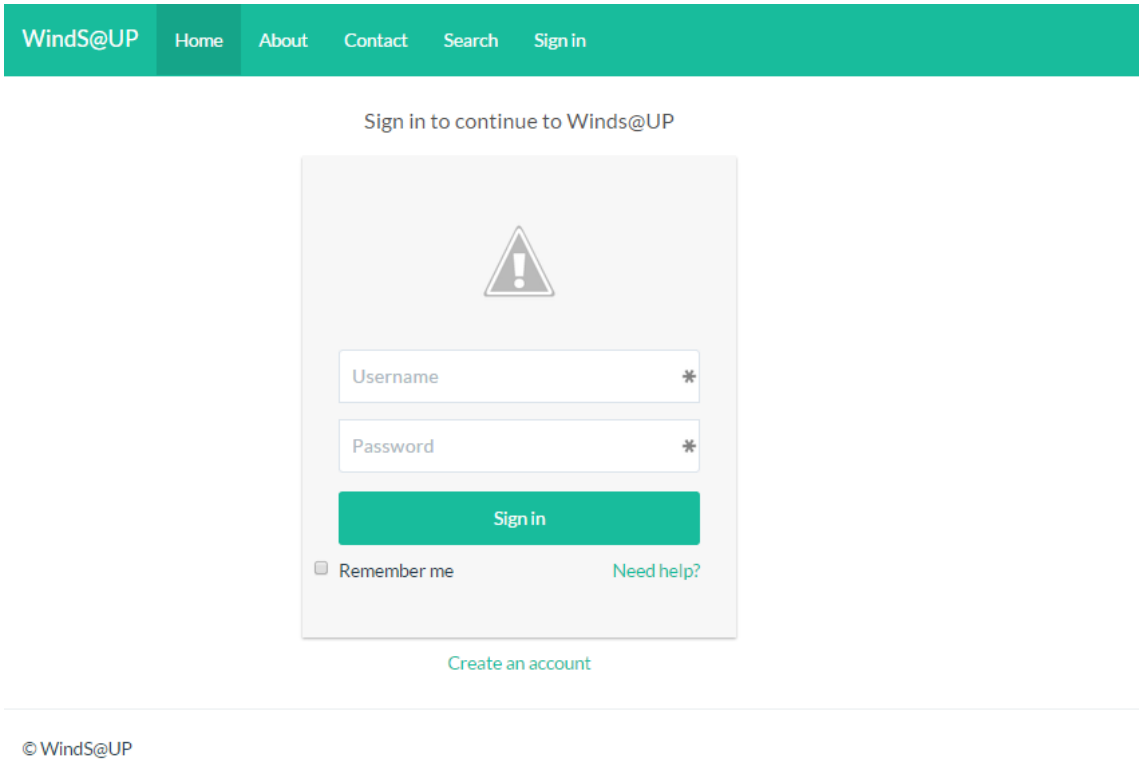

Figura B.9: Autenticação na Plataforma.