INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema de armazenamento de grande volume de dados para sistemas energéticos inteligentes

GIL MACHADO SILVA PINHEIRO
outubro de 2017

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Big Data Storage

Sistema de armazenamento de grande volume de dados para sistemas energéticos inteligentes

Gil Machado da Silva Pinheiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Isabel Praça

Co-orientador: Eugénia Margarida Vinagre

Júri:

Presidente:

Vogais:

Porto, Outubro 2017

Resumo

O sector elétrico encontra-se num ponto fulcral da sua evolução. A mais recente conferência sobre as alterações climáticas das Nações Unidas (COP21 ou CMP11), realizada em Paris, veio novamente apelar ao desenvolvimento de estratégias para travar o aquecimento global. A crescente utilização de fontes de energia renovável distribuídas, e o associado fluxo bidirecional de energia, bem como a utilização de metodologias inteligentes de gestão destes recursos, é um dos aspetos chave no âmbito das *Smart Grids* (redes inteligentes). Este novo paradigma, para além dos benefícios monetários para os consumidores, tem associado o potencial de reduzir as emissões de carbono.

Esta dissertação é realizada no âmbito do Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e o Desenvolvimento (GECAD) sediado no Instituto Superior de Engenharia do Instituto Politécnico do Porto (ISEP-IPP) no domínio de sistemas energéticos inteligentes.

O projeto propõe e implementa um sistema de armazenamento e processamento distribuído associado a tecnologias Big Data, que integra diferentes fontes de dados, permite a realização de análises em tempo-real e aplicação de algoritmos de *data mining*.

Palavras-chave: Armazenamento BigData, Smart Grids, Apache Spark, Cassandra, SMACK, YCSB

Abstract

The utilities industry is at a pivotal point in its evolution. The most recent United Nations Climate Change Conference (COP21 or CMP11) held in Paris, raised awareness for the deployment of strategies to slow down global warming. The increasing adoption of distributed renewable energy sources, and the bidirectional flow of energy, as well as the intelligent management of these resources, is one of the main key aspects of Smart Grids. This new paradigm, which beyond the financial benefits for consumers, has the potential to reduce carbon emissions.

This dissertation was developed at the Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD) in the Institute of Engineering Polytechnic of Porto (ISEP-IPP), under the domain of intelligent power energy systems.

The project proposes and implements a distributed storage and processing system using Big Data Technologies, capable of integrating several data sources, streaming real-time analysis and the usage of data mining techniques.

Keywords: BigData Storage, Smart Grids, Apache Spark, Cassandra, SMACK, YCSB

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

ACID	*Atomicity, Consistency, Isolation and Durability*
BASE	*Basic Availability, Soft-State and Eventual Consistency*
CAP	*Consistency, Availability and Partition Tolerance*
DStream	*Discretized Stream*
EB	*Exabyte*
FFE	*Fuzzy Front End*
GFS	*Google File System*
HA	High Availability
HDFS	*Hadoop Distributed File System*
JSON	*Javascript Object Notation*
MB	*Megabyte*
NCD	New Concept Development Model
NoSQL	*Not Only SQL*
NPD	*New Product Development*
OLTP	*Online Transaction Processing*
P2P	*Peer-to-Peer*
PLC	*Programmable Logic Controller*
PMU	*Phasor Measuring Unit*
RDD	*Resilient Distributed Dataset*
RHEL	*Red Hat Enterprise Linux*
SQL	*Structured Query Language*
TB	*Terabyte*
XML	*eXtensible Markup Language*

ZB *Zettabyte*

1 Introdução

Análise em tempo-real é declarada como uma das principais estratégias de processamento de grandes quantidades de dados no futuro (Asay, 2015). Este tipo de análise permite às organizações reagir mais rapidamente e tomar decisões mais acertadas. Na atualidade, existe um conjunto de ferramentas Big Data para este tipo de análise, que beneficiam as mais variadas indústrias.

O setor elétrico deve explorar a adoção destas tecnologias no contexto das *smart grids*. A implementação destas redes inteligentes implica a captura e monitorização de equipamentos que produzem informação a grandes taxas de amostragem. O volume de dados produzido poderá ser demasiado grande para ser eficientemente gerido por sistemas informáticos tradicionais (Aiello and Pagani, 2014).

No âmbito de sistemas energéticos inteligentes, o GECAD pretende explorar as vias de análise em tempo-real de medições periódicas no contexto de Big Data. De momento, a captura de dados é realizada por analisadores de energia localizados em vários edifícios do campus do ISEP e armazenados numa base de dados não distribuída. Como solução alternativa, esta dissertação baseia-se na implementação de um sistema distribuído Big Data, com o objetivo de armazenar e analisar toda a informação produzida.

O capítulo 2 descreve em detalhe o contexto do projeto e os objetivos, o problema a resolver e o valor associado ao projeto. O capítulo 3 destaca o estado de arte de *Big Data* e a sua definição na literatura, enuncia as principais ferramentas e arquiteturas, e apresenta algumas soluções no contexto das *smart grids*. O capítulo 4 apresenta e justifica a solução proposta e as metodologias de avaliação do sistema. Por fim, o capítulo 5 detalha o processo de implementação e apresenta os resultados obtidos dos testes de avaliação.

2 Enquadramento

As subseções seguintes descrevem o enquadramento do projeto, explicando o contexto em que se insere, detalhando qual o problema a resolver, os objetivos a cumprir e os resultados esperados. Por fim, é apresentado a análise de valor associada à concretização do projeto e uma proposta de negócio acompanhada de um o modelo de negócio *canvas*.

2.1 Contexto

Smart Grids são as novas redes elétricas do futuro. Ao contrário da comunicação unidirecional da rede elétrica tradicional (distribuidor para o consumidor), as *smart grids* integram tecnologias de comunicação e de informática num sistema resiliente, auto recuperável e de comunicação bidirecional, que capacita as empresas de energia de total visibilidade e controlo em tempo-real sobre os seus serviços e recursos da rede (Farhangi, 2010). A implementação destas redes inteligentes surge da necessidade de estratégias mais eficientes de produção e distribuição, otimização dos recursos, integração de energias renováveis, melhorias a nível de fiabilidade da rede, e de novos requisitos a nível de prestação de serviços junto dos consumidores (ISGAN, 2014). Devido à integração de energias renováveis na rede, as *Smart Grids* terão também um papel importante na redução da pegada de carbono (Pratt *et al.*, 2010).

A Figura 1 apresenta uma comparação entre a rede tradicional e as redes inteligentes. Da figura salienta-se a introdução de novos elementos no sistema (e.g. veículos elétricos) e novos canais de comunicação entre os diferentes elementos da rede. A introdução destes elementos traduz-se na integração de novo *hardware* e subsistemas de *software* capazes de lidar com o crescente volume de dados e os variados canais de comunicação.

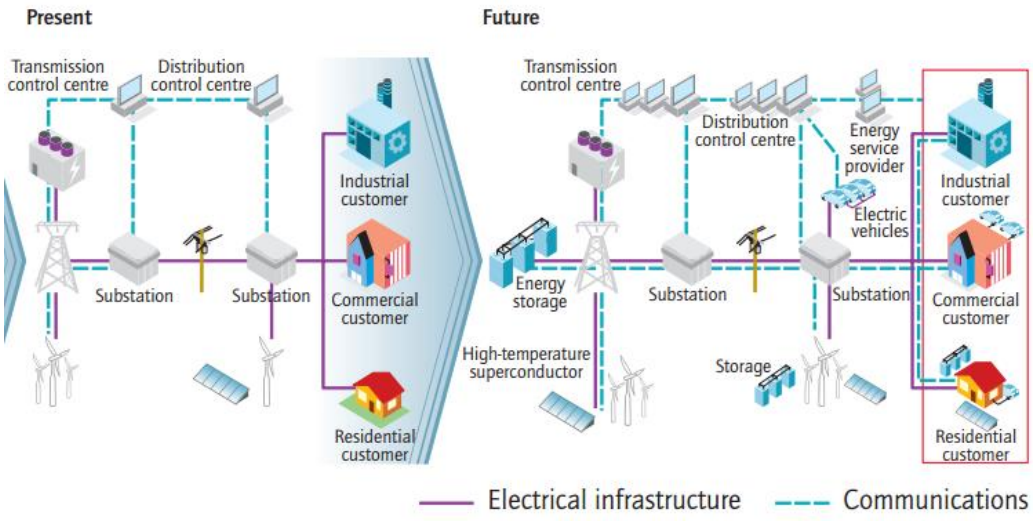

Figura 1: Sistemas elétricos mais inteligentes. Fonte: (IEA 2011, p.6, Figura 1).

No âmbito do *hardware* destaca-se a utilização de medidores inteligentes (*Smart Meters*), *Phasor Measuring Units* (PMUs) e diversas redes de sensores integrados.

Os medidores inteligentes são a substituição digital dos contadores mecânicos. Estes aparelhos medem o consumo de eletricidade em intervalos de tempo, tipicamente de 15 ou 30 minutos, e são o elemento principal que estabelece a comunicação bidirecional entre o consumidor e o distribuidor. Medidores inteligentes têm um papel importante nas estratégias de distribuição da eletricidade, ao providenciarem informação atualizada dos consumos dos clientes. A informação recolhida é analisada para descobrir padrões de consumo, que podem ser utilizados, por exemplo, na criação de planos de tarifas dinâmicos ajustados a cada consumidor.

PMUs são considerados um dos dispositivos mais importantes no futuro das redes elétricas. Estes equipamentos medem as ondas elétricas num determinado ponto da rede a taxas de 10 a 60 observações por segundo. Ao comparar valores de diversas unidades em diferentes localizações da rede, é possível ter uma visão atualizada das condições da infraestrutura, permitindo o diagnóstico atempado de anomalias funcionais e acelerando ações de prevenção e reparação.

Outras fontes de dados devem ser consideradas, por exemplo, para correlacionar causas e efeitos de modo a extrair verdadeiro valor da informação. A monitorização em tempo real da rede permite rapidamente a deteção de falhas, mas em certos casos não as causas. De certa forma, com a integração de medidores inteligentes e de dados meteorológicos/ambientais, poderá ser possível correlacionar efeitos com causas. No âmbito de prevenção de falhas na rede, poderão ser criados modelos de previsão a partir de informação histórica ambiental e da rede que atempadamente alertam sobre a possibilidade de falhas futuras.

A contribuição das tecnologias *big data* no contexto das *smart grids* prende-se principalmente nas capacidades de integração e análise de grandes conjuntos de dados de natureza variada. A Tabela 1 (Lai and Lai, 2015) enumera algumas potenciais aplicações *big data* associadas às *smart grids*.

Tabela 1: Potenciais aplicações *Big Data* em *Smart Grids*. Fonte: (Lai & Lai 2015, Tabela 1).

1	Previsão e análise da situação económica e impacto social
2	Desenvolvimento de raciocínio científico para tomadas de decisão
3	Análise de desempenho em sistemas de armazenamento e de geração de energias
4	Balanceamento de cargas de acordo com consumos de energia e de eficiência
5	Análise comportamental dos consumidores
6	Utilização de medidores inteligentes e dispositivos eletrónicos inteligentes para análises de estimação de estados
7	Técnicas analíticas aplicadas à determinação de preços e implementação de programas de incentivos
8	Otimização da infraestrutura da rede
9	Previsão de intensidades de consumo e geração de energia
10	Gestão de recursos
11	Técnicas analíticas aplicadas à qualidade de serviço

Apesar da relação entre as duas áreas ser promissora, é necessário evidenciar a importância dos desafios à implementação dos sistemas associados a cada uma. Umas das maiores barreiras ao avanço das *Smart Grids* são os desafios relativos a segurança dos sistemas (Khurana, Hadley and Frincke, 2010) e privacidade dos dados (Efthymiou and Kalogridis, 2010). Para uma implementação bem-sucedida e que seja aceite pelo público-geral, deve ser realizada uma abordagem holística que considere estes dois fatores como parte das principais prioridades dos sistemas.

2.2 Problema e Objetivos

De momento o GECAD possui uma rede de analisadores de energia distribuídos por três edifícios no *campus* universitário. Estes dispositivos realizam medições em intervalos de tempo reduzidos (10 a 15 segundos), de dados de consumo de energia de tomadas, luzes e sistemas de ar condicionado, e também de produção de energia resultante de painéis fotovoltaicos. Acesso aos analisadores é conseguido através de PLCs (*Programmable Logic Controllers*), que se encontram ligados à rede local do GECAD.

A Figura 2 apresenta a planta do edifício N e as salas afetadas pela monitorização dos principais analisadores indicadas pelos símbolos da legenda. No total, existem 5 analisadores de energia de monitorização de consumo de sistemas HVAC, tomadas e luzes, e um inversor que captura informação de produção de energia.

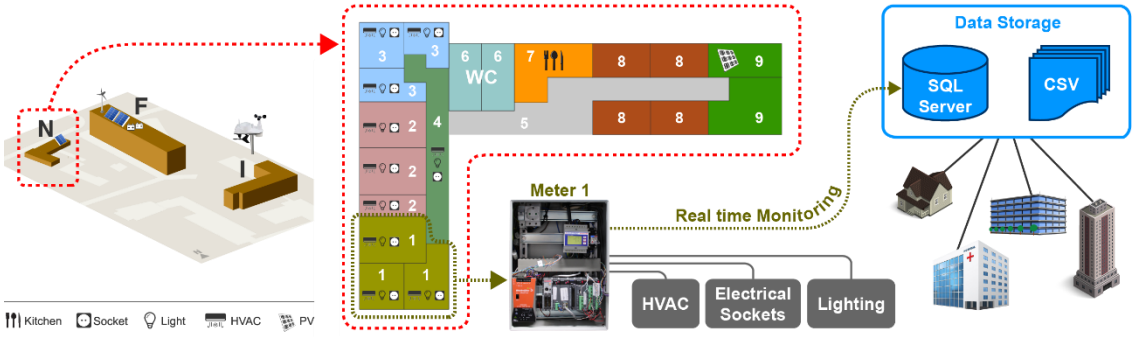

Figura 2: Planta do edifício N realçando os principais analisadores de energia ativos.

O armazenamento dos dados recolhidos está principalmente a ser armazenado num único servidor SQL Server. Para além desta informação, o GECAD recebe e armazena, no mesmo servidor, conjuntos de dados de parceiros externos nos projetos internacionais DREAM-GO[1] e FUSE-IT[2].

[1] Sítio: http://www.dream-go.ipp.pt
[2] Sítio: https://itea3.org/project/fuse-it.html

O objetivo principal deste trabalho é a especificação e desenvolvimento de um repositório de dados, centralizando toda a informação disponível, permitindo a aplicação de técnicas de extração de conhecimento e análise em tempo-real. O grupo pretende também incluir outras fontes de dados, como condições meteorológicas e informação eólica. A Tabela 2 contém um exemplo dos tipos de dados que o sistema terá de armazenar.

Tabela 2: Excerto de dados relativos ao consumo de energia elétrica em iluminação, tomadas e sistema de ar condicionado.

Date	Hour	N1_P1 AC	N1_P2 Iluminação	N1_P3 Tomadas
25-07-2014	00:00:00	12,96	0,00	216,00
25-07-2014	00:00:10	14,21	0,00	211,00
25-07-2014	00:00:20	12,94	0,00	215,00
25-07-2014	00:00:30	12,99	0,00	207,00

Para a realização do projeto não foi disponibilizado qualquer tipo de orçamento financeiro e foram utilizados recursos de hardware já existentes. A Tabela 3 enumera os principais requisitos do projeto.

Tabela 3: Requisitos do projeto.

Funcionais	Não funcionais
Capturar e armazenar dados de forma continuada de diferentes fontes de dados	Resiliente a falhas para prevenir a perda de dados
Capaz de introduzir informação em massa	Medidas de segurança e privacidade
Realizar análise em tempo-real	Fiabilidade e Eficiência
Suportar a aplicação de algoritmos de *data mining*	Desempenho, diversidade e flexibilidade

2.3 Análise de Valor

O processo de negócio e de inovação é decomposto por três fases sequenciais: *fuzzy front end* (FFE), *new product development* (NPD) e comercialização (Belliveau *et al.*, 2002). Durante o processo, uma boa gestão e desempenho na fase inicial FFE, contribui positivamente para o sucesso de um novo produto ou serviço (Kim and Wilemon, 2002). FFE é uma fase embrionária de identificação de oportunidades e de formulação de ideias na base do processo de desenvolvimento de um novo produto ou serviço. Em contraste com a fase NPD, a FFE é caracterizada por ser uma fase destruturada e experimental, em que as ideias são suscetíveis à mudança e cujo o seu abandono não resulta em grandes danos para a organização.

Com o objetivo de potenciar o sucesso da fase FFE, Peter Koen (2001) define o *New Concept Development Model* (NCD). O modelo NCD, representado na Figura 3, define cinco componentes chave, potenciados pelo apoio de ações executivas da organização e influenciados por fatores externos. O processo tem inicio na identificação de oportunidades ou na idealização do produto ou serviço e termina após a sua conceptualização.

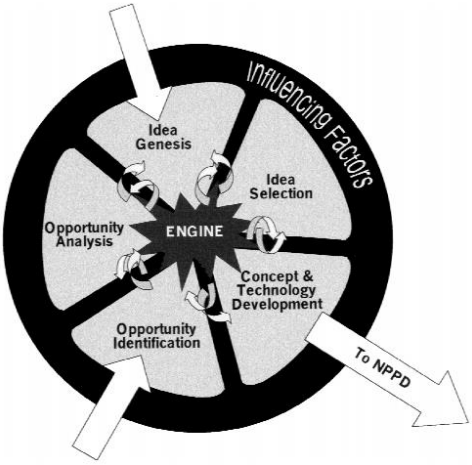

Figura 3: Diagrama do NCD. Fonte: (Koen *et al.*, 2001, fig. 1)

Os cinco elementos chave no modelo NCD incluem: *opportunity identification* (i); *opportunity Analysis* (ii); *Idea Genesis* (iii); *Idea Selection* (iv) e *Concept & Technology Development* (v).

(i) Neste elemento, a organização identifica oportunidades que pretende aproveitar. Tipicamente alinhado com os objetivos do negócio, a base deste elemento é a utilização de métodos de visionamento do futuro. O processo pode ser resultante de atividades formais, como uma análise aos fatores externos, ou de atividades informais, por exemplo, sessões de discussão coletiva. Neste passo, podem ser consideradas técnicas como a análise de mercados, tendências tecnológicas ou análise de padrões de consumidores.

(ii) A análise de oportunidades tem como objetivo determinar quais das oportunidades realmente devem ser perseguidas. As metodologias a utilizar neste elemento são semelhantes às usadas no processo de identificação. A organização pode recorrer a análise de competidores, estudos detalhados de segmentos de mercado ou a identificação de necessidades dos consumidores, que não sejam satisfeitas pelos produtos existentes no mercado.

(iii) Este terceiro elemento diz respeito aos processos evolucionais de idealização e de enriquecimento de ideias. As ideias são construídas, modificadas e até abandonadas. Neste processo, recomenda-se, a utilização de uma equipa envolvida em sessões de *brainstorming*. Estas sessões, devem ter em consideração informação sobre as necessidades de consumidores e avanços tecnológicos. A organização pode ainda contribuir para o processo, ao adotar uma cultura organizacional que fomente a criação de ideias, como por exemplo, fornecendo incentivos aos seus colaboradores, designando posições de liderança do processo a pessoas com espirito criativo e envolvendo os colaboradores em sessões de *networking* e de partilha de conhecimento.

(iv) Tomar uma boa decisão no processo de seleção de ideias é fundamental para o sucesso da mesma. No entanto, não existe um processo que garanta um resultado positivo. Para tal, a seleção pode recorrer a metodologias de *anchored scales*. Estas metodologias justificam a seleção, ao quantificar a adequabilidade estratégica da ideia e a probabilidade de sucesso técnico e comercial.

(v) A definição do conceito é o elemento final do NCD. O decisor deve construir um plano de negócio ou uma proposta formal do projeto que inclua uma avaliação de objetivos a cumprir e de critérios relevantes para a organização.

No âmbito do projeto, o grupo GECAD sentiu a necessidade de explorar as tendências associadas às áreas de investigação estudadas. Análises às tendências tecnológicas no contexto das *smart grids* e aos fatores que influenciam o grupo, conduziram à ideia de utilizar ferramentas *big data* para solucionar um problema atual. Durante o processo de maturação da ideia, procurou-se opções mais orientadas para a análise em tempo-real, conduzindo à presença em *webinars* de diferentes tecnologias. Por fim, a conceptualização da ideia resulta na solução proposta no capítulo 4.

2.3.1 Proposta de Valor

A definição de uma proposta de valor é um processo de desenvolvimento de construção de valor para a experiência do cliente alvo, com o objetivo final de criação de riqueza para a própria empresa (Barnes, Blake and Pinder, 2009). Na sua base, descreve a mistura de produto(s) e/ou serviço(s), relação e "imagem" que a empresa oferece aos seus clientes, e comunica explicitamente o que a empresa faz de melhor ou de diferente da competição (Kaplan and Norton, 2004). A proposta de valor pode também ser usada internamente como uma diretriz que guia os diferentes esforços organizacionais na mesma direção. Tipicamente, começa-se por identificar o segmento-alvo de clientes, identifica-se os problemas dos clientes, as suas necessidades e o que valorizam, e define-se os benefícios da própria oferta. É especialmente importante a identificação dos fatores de valorização, uma vez que são os consumidores que atribuem o verdadeiro valor da oferta da empresa.

Esta atribuição de valor refere-se ao conceito de valor percebido. Este conceito descreve que os atributos que constituem valor são altamente idiossincráticos entre indivíduos, podendo ser fatores como o preço, comodidade, o que se procura num produto ou a relação entre qualidade e preço (Zeithaml, 1988). Woodall (2003) decompõe valor para o cliente em cinco formas principais. Estas formas ditam que o cliente, de certa forma, realiza um balanço entre benefícios e sacrifícios, deriva valor a partir do uso do produto e perceção dos seus atributos, e realiza comparações com pontos de referência objetivos (e.g. outros produtos). Ainda na mesma publicação, o autor evidencia a natureza longitudinal de uma proposta de valor, ou seja, cada indivíduo valoriza diferentes coisas em diferentes alturas. Uma associação entre benefícios procurados e oferecidos altamente contribui para a decisão de compra e correlaciona-se também com a lealdade do cliente perante a marca (Yang and Peterson, 2004).

Em grande parte dos casos, as organizações têm grandes dificuldades na determinação do valor dos seus ativos e do que os seus clientes valorizam, pelo que devem recorrer a técnicas de modelação e conceptualização de valor. Nicola e Eduarda (2012) propõem um modelo conceptual de decomposição de valor para o cliente, que alinha os conceitos de valor percebido e valor para o cliente com as áreas de investigação de redes de valor e de redes colaborativas. O modelo alinha as diferentes formas de valor, as posições temporais e os benefícios e

sacrifícios percebidos, tanto por parte da empresa como dos clientes, com os ativos tangíveis e intangíveis da organização. Desta forma, a organização é capaz de alinhar as suas ofertas com as expectativas dos clientes, potenciando a retenção dos mesmos e o aumento das receitas.

No âmbito deste projeto, a proposta de valor associada, centra-se principalmente na expansão das capacidades do grupo, no que diz respeito à investigação suportada por conjuntos de dados maiores e por novas metodologias de análise previamente não exploradas. Para além disso, o sistema implementado poderá também acelerar o ritmo das investigações, resultando num aumento de produtividade e de descoberta de conhecimento.

2.3.2 Cenários de Negócio

Um possível cenário de negócio seria a prestação de serviços que consistiriam na implementação de um sistema semelhante, personalizado aos requisitos de cada cliente. Por exemplo, a empresa Xarevision, sediada no Porto, trabalhou com o Continente na implementação de sistemas de armazenamento e descoberta de conhecimento, baseado na utilização da ferramenta Apache Hadoop (Macedo and Caxias, 2015).

A Figura 4 apresenta o modelo de negócio *canvas* para o cenário apresentado. Os clientes recorreriam a esta prestação e serviço devido à falta de tempo ou experiência necessárias para a implementação de uma solução *big data*. Como o projeto se insere no contexto de sistemas energéticos inteligentes, os clientes-alvo poderiam ser operadores das redes de distribuição (DSOs), operadores de s*mart grids* ou *microgrids*, parques tecnológicos e industriais, campus universitários e gestores de edifícios.

<table>
<tr>
<td rowspan="2">Key Partners
• Fornecedores de serviços na cloud (AWS ou GCE)
• Fornecedores de hardware especializado</td>
<td>Key Activities
• Análise das fontes de informação
• Investigação e desenvolvimento
• Descoberta de Conhecimento
• Administração de Sistemas</td>
<td rowspan="2">Value Proposition
• Armazenamento de dados fiável, seguro e resiliente
• Melhor desempenho
• Descoberta de conhecimento
• Benefícios Financeiros
• Aumento produtividade</td>
<td>Customer Relationships
• Assistência pós-venda
• Equipa de desenvolvimento dedicada</td>
<td rowspan="2">Customer Segments
• DSOs
• Parques Tecnológicos e Industriais
• Campus Universitários
• Gestores de Edifícios</td>
</tr>
<tr>
<td>Key Resources
• Engenheiros especializados
• Data Scientists
• Infraestrutura de suporte (hardware e software)</td>
<td>Channels
• Presencial</td>
</tr>
<tr>
<td colspan="3">Cost Structure
• Custos com pessoal técnico especializado
• Hardware e Software de desenvolvimento</td>
<td colspan="2">Revenue Streams
• Principal: Prestação do serviço de implementação de uma solução Big Data
• Secundários: Serviços de apoio/manutenção pós-implementação</td>
</tr>
</table>

Figura 4: Modelo de Negócio *Canvas*.

Seria procurada a construção de relações com parceiros provedores de infraestruturas que suportassem as soluções a implementar, quer fossem de armazenamento e computação na *cloud* ou vendedores de *hardware* especializado. A prestação do serviço envolveria um processo moroso de constante comunicação bidirecional, uma vez que em certos casos poderia ser necessário o conhecimento de parte das infraestruturas informáticas dos clientes. Desta maneira, adequar-se-ia uma negociação *Win-Win* em que as duas partes estão envolvidas em conjunto para alcançar os mesmos objetivos e construírem relações de confiança.

A Tabela 4 esclarece o valor do projeto para o cliente numa perspetiva longitudinal. Como se trata de um serviço a fase *transaction* é considerada como sendo o período de desenvolvimento da solução e *Ex Post* a fase de utilização da solução assim que completa. *Disposition* é a fase de depreciação do serviço ou disposição.

Tabela 4: Alinhamento dos benefícios e sacrifícios do projeto numa perspetiva longitudinal.

	Fases Temporais			
	Ex Ante	**Transaction**	**Ex Post**	**Disposition**
Benefícios	Inovação, Qualidade	Customização, Qualidade técnica	Fiabilidade, Flexibilidade, Financeiros	Reutilização do sistema para outros efeitos
Sacrifícios	Preço	Tempo, Custos de Relação	Custos de manutenção e uso	

3 Big Data

Nos últimos 20 anos, a quantidade de informação gerada evoluiu de forma exponencial. As empresas deparam-se com novos desafios na gestão e manutenção de dados, sendo necessárias soluções alternativas adaptadas às novas realidades. Segundo o estudo "The Digital Universe" (Turner *et al.*, 2014), 132 EB (*Exabyte*) de dados foram criados durante o ano de 2005, valor que subiu para 4.4 ZB (*Zettabyte*) durante o ano 2013. Desde então, é estimada uma duplicação em tamanho a cada dois anos, sendo que todas as previsões apontam para o valor de 44 ZB de informação anual no ano 2020.

Nos dias de hoje, a influência do fenómeno Big Data atinge quase todas as áreas de negócio. Casos reais comprovam que qualquer área pode beneficiar de sistemas que extraem conhecimento a partir de dados, levando a decisões mais acertadas que produzem maior valor para as organizações.

No setor da saúde, acesso a informação valiosa pode contribuir: no melhoramento dos planos de tratamento e acompanhamento de pacientes; diagnósticos preliminares de doenças mais eficazes; redução de custos nos cuidados. Os grupos MeriTalk e EMC (2014) consultaram 150 executivos de agências da área de cuidados de saúde e investigação. Desse grupo, 59% acreditam que o cumprimento dos objetivos das organizações a que pertencem irá depender da capacidade de implementar soluções Big Data, no entanto apenas 29% possui engenheiros informáticos preparados para implementar soluções Big Data.

A indústria do retalho teve de se reinventar às novas realidades do comércio. Atualmente, o consumidor tem cada vez mais poder na decisão de compra. As redes sociais capacitaram os consumidores de métodos de partilha mais facilitados, influenciando as mentes de possíveis compradores. A indústria deve procurar a integração de novas fontes de dados que vão para além da informação transacional, para perceber o que motiva os seus clientes. A WalmartLabs, do gigante de retalho Walmart, é composta por múltiplas equipas de investigação que têm como objetivo desenvolver soluções ajustadas aos clientes. Um exemplo é o projeto

SocialGenome, que implementa metodologias analíticas sobre dados originários de redes sociais como o Facebook, Twitter e Youtube, e identifica os interesses dos seus clientes percebendo também os seus contextos (Dezyre, 2015). Soluções como esta permitem à empresa alcançar novos clientes e ao mesmo tempo aproximar-se das pessoas.

Cada vez mais é reforçado o investimento em recursos de energia renovável, permitindo ir ao encontro das preocupações ambientais e de sustentabilidade dos países. Com o objetivo de progredir nesse sentido, já estão a ser explorados novos conceitos como o investimento em energias renováveis e a implementação de sistemas energéticos inteligentes (*Smart Cities*, *Smart Grids*, *Smart Meters*). Na adaptação das atuais redes elétricas para estes novos paradigmas, a informação disponível é cada vez maior, mais variada e com elevada cadência. Sistemas de armazenamento direcionados para Big Data em conjunto com métodos de análise em tempo real podem, por exemplo, levar a uma gestão mais eficiente dos sistemas elétricos e deteção de falhas na estrutura da rede, minimizando tempos de resposta a eventos críticos (como *Blackouts*) e reduzindo os custos associados.

É importante realçar que em Big Data o conceito de descoberta de conhecimento não é algo novo. Algoritmos que trabalham sobre dados, procurando a obtenção de conhecimento, já se encontravam integrados em aplicações de contexto empresarial (*Business Intelligence).* Big Data vem introduzir novos desafios, de volume, variedade de informação, de velocidade de análise, para os quais os sistemas informáticos ainda não se encontravam preparados.

3.1 Definição e os Vs de Big Data

À medida que as organizações se sentiam mais confortáveis com a área, também crescia a adoção do termo "Big Data", por vezes incorporado num plano estratégico de marketing. Empresas que oferecem soluções Big Data, acabaram por caracterizar o termo à sua maneira e desta forma, não existe uma clara definição do termo.

A primeira utilização documentada do termo é realizada em 1997 pelos investigadores da NASA, cita-se M. Cox e D. Ellsworth (1997):

> *"Processos de visualização [computacional da dinâmica de fluidos] fornece um desafio interessante para sistemas computacionais: os conjuntos de dados são geralmente muito grandes, taxando as capacidades da memória principal, disco local, e até mesmo de discos remotos. Nós chamamos a isto o problema de big data." (tradução).*

Mais recentemente, a Oracle (2015) descreve Big Data como sendo uma estratégia de gestão de informação que tem de incluir e integrar novos tipos de dados e processos de gestão com informação tradicional já existentes. A IBM define Big Data simplesmente como sendo a habilidade de capturar, reter e perceber novos tipos de dados não estruturados (Zikopoulos *et al.*, 2014, p. 39). Para além de evidenciarem a importância da variedade dos tipos dados, outras definições chamam a atenção para a eficiência temporal dos processos de análise dos dados e a aplicabilidade deles em casos reais.

Apesar da diversidade de opiniões, a revisão da literatura aponta para um consenso geral, tanto por organizações como por académicos, no que diz respeito a três características que são conhecidas como os 3Vs de Big Data.

3.1.1 Os 3 Vs Tradicionais

Doug Laney (2001) escreve uma publicação onde explora os problemas emergentes nas estratégias de armazenamento e gestão de dados da altura. Nesse documento, o autor coloca os novos desafios de gestão de dados num plano tridimensional: *Data Volume, Data Variety e Data Velocity*.

Volume – Refere-se à dimensão dos dados em termos de armazenamento. Quando se menciona Big Data associa-se sempre a grandes quantidades de informação. Geralmente os conjuntos de dados em Big Data são caracterizados por dimensões na ordem dos TB (*Terabyte)* para cima, embora não exista um patamar a partir do qual a classificação como Big Data seja adequada. As infraestruturas devem estar preparadas para serem capazes de escalar horizontalmente e lidar com as implicações de armazenamento e computação distribuídos.

Variedade – Para além do volume de dados, o que torna difícil a integração dos dados é a multiplicidade de representações que os caracterizam. Num só conjunto, os dados podem ter o formato áudio/vídeo, grafos que representam interligações (e.g. redes sociais), informação "crua" de sensores ou equipamentos eletrónicos, entre outras estruturas. Desta forma, os dados são tipicamente categorizados como sendo: estruturados, semiestruturados ou não estruturados.

Dados estruturados assumem à partida semânticas bem definidas, com características comuns entre si, estabelecendo inter-relações bem delineadas. Por essas razões, conduzem a um elevado nível de inflexibilidade à mudança. Este tipo de dados alimentou durante anos as implementações de sistemas de bases de dados relacionais.

Informação semiestruturada implica heterogeneidade em termos semânticos, o que aumenta a dificuldade nos processos de consolidação, integração e posterior análise. Neste tipo de dados um esquema que descreva a estrutura é opcional. Desta forma, são caracterizados como sendo *schemaless* e auto descritivos, ou seja, o esquema não é necessariamente definido de forma clara, podendo estar implicitamente escondido (Abiteboul, Buneman and Suciu, 1999, p. 19). Este nível de abstração garante um maior nível de flexibilidade e escalamento. Dentro deste tipo são considerados formatos como por exemplo o XML (*eXtensible Markup Language*) e JSON (*JavaScript Object Notation)*, e estruturas de dados como grafos.

Os dados não estruturados são caracterizados pela inexistência de um modelo de dados, o que faz com que a sua análise seja mais complexa. A maioria deste tipo de informação é texto puro (e.g. documentos Word/PDF, E-Mail, publicações no Facebook/Twitter, etc.), no entanto existem também dados não textuais como formatos de áudio, vídeo e imagens.

Velocidade – Armazenamento de dados apenas para alimentar aplicações de negócio significa um desperdício de valor. Esta dimensão evidencia a importância do tempo de resposta das metodologias de processamento de dados. Não basta apenas analisar e esperar indefinidamente por resultados, é necessário pensar em quais as metodologias que fazem sentido serem aplicadas tendo em consideração os tempos de produção de resultados. As atuais ferramentas associadas a Big Data encontram-se divididas em duas implementações diferentes no que diz respeito à análise dos dados.

Apache Hadoop, uma das ferramentas mais conhecidas, foi inicialmente construída para efetuar análise em *batches* (blocos/fornadas). Este tipo de processamento, tipicamente realizado periodicamente, é aplicado aos dados depois de estes já se encontrarem armazenados. O processamento *batch* é mais indicado para análises complexas sobre grandes conjuntos de dados geralmente com um tempo de resposta na ordem de vários minutos ou horas.

A outra alternativa é optar por uma solução de análise de *streaming*/*near real-time*. Este tipo surge da necessidade de reagir a eventos o mais rapidamente possível. Análise *streaming* é geralmente descrita como análise de dados em movimento, aplicada sobre muitos conjuntos de dados de pequena dimensão à medida que estes entram no sistema.

3.1.2 Novos Vs

A adoção personalizada do termo levou também à definição de novos Vs. Cada organização identificou certas propriedades dos dados que considerou importantes para a sua área de negócio. Uma pesquisa na Internet como "The Vs of Big Data" devolve diferentes Vs (quatro, cinco e por vezes até sete Vs). A Tabela 5 enumera algumas definições dadas por diferentes autores e organizações de outros Vs, para além dos referidos na secção anterior, associados a Big Data.

Tabela 5: Outros Vs de Big Data.

V	Descrição
Valor	Novas fontes de informação podem resultar em grande valor, mas muita das vezes encontra-se disperso (Chen, Mao and Liu, 2014, p. 173).
	O valor dos dados varia significativamente. O desafio é identificar o que é valioso e transformar e extrair a informação para análise (Oracle, 2013, p. 4)
Veracidade	Refere-se aos problemas de qualidade e confiabilidade dos dados (Zikopoulos *et al.*, 2014, p. 11)
Variabilidade	Os fluxos de dados são altamente inconsistentes com diferentes períodos de intensidade (SAS, 2014)

Veracidade pode ser particularmente interessante para empresas que monitorizam o consumo de eletricidade, já que é essencial que os dados sejam verdadeiros e não resultantes de funcionamento defeituoso dos equipamentos ou de ataques maliciosos.

Em 2013, Doug Laney (2013) argumenta que as novas dimensões não são qualidades definidoras dos dados em Big Data e que a preocupação com estas vertentes pode levar à definição incorreta de prioridades e fracos planeamentos.

3.2 Teorema de CAP e ACID vs BASE

Todas as soluções Big Data baseiam-se na implementação de sistemas distribuídos. No inicio do milénio, Eric Brewer (2000) introduz o teorema CAP (*Consistency*, *Availability* e *Partition Tolerance*), que influenciou as arquiteturas dos sistemas de armazenamento distribuídos. *Consistency* dita que qualquer leitura feita ao sistema de armazenamento obtém a imagem mais atualizada dos dados. *Availability* refere-se ao nível de disponibilidade do sistema para a concretização de pedidos. *Partition Tolerance* descreve que o sistema é tolerante a partições da rede e aos riscos associados de perda de mensagens. Na sua base, o teorema define que num sistema distribuído de armazenamento é impossível garantir todas as propriedades em simultâneo e que devem ser realizados *trade-offs* no desenho dos sistemas.

As bases de dados relacionais baseiam-se nas propriedades ACID (*Atomicity, Consistent, Isolated, Durable*). Introduzidas por Jim Gray (1981), estas propriedades garantem a consistência associada ao teorema CAP. Com o objetivo de cumprir novos requisitos de disponibilidade dos sistemas, Eric Brewer cria o conceito BASE (*Basic-Availability, Soft-State* e *Eventual Consistency)* no final da década de 90 (Brewer, 2012, p. 24). A Tabela 6 descreve as propriedades dos dois modelos.

Tabela 6: Modelos de consistência ACID e BASE.

Modelo	Propriedade	Descrição
ACID	*Atomic*	Todas as partes constituintes de uma transação têm que ser executadas com sucesso.
	Consistent	Todas as transações obedecem aos mesmos protocolos/regras legais da base de dados. Como resultado, a base de dados nunca se encontra num estado inconsistente.
	Isolated	Múltiplas transações não interferem entre si.
	Durable	Os resultados das transações são permanentes, mesmo após a ocorrência de falhas no sistema.
BASE	*Basic Availability*	O sistema procura alta disponibilidade, respondendo a todos os pedidos, mesmo que a resposta seja um erro ou dados inconsistentes
	Soft-State	O estado do sistema ao longo do tempo é maioritariamente inconsistente devido à filosofia de eventual consistência dos dados.
	Eventual Consistency	Alterações aos dados são eventualmente propagadas pelo sistema. Leituras à base de dados podem retornar dados ainda não atualizados.

No contexto de Big Data, grande parte das bases de dados NoSQL (*Not Only SQL*) seguem a filosofia BASE, sacrificando a consistência dos dados de maneira a garantir melhor disponibilidade e desempenho. Isto significa, que ao trabalhar com bases de dados NoSQL é assumido como pressuposto, que é normal trabalhar-se com informação que não é a mais atualizada. Mais tarde, surgiram as bases de dados NewSQL que pretendiam manter as propriedades ACID e a escalabilidade e desempenho associados a sistemas NoSQL.

3.3 Tecnologias Big Data

O número de tecnologias no contexto de *big data* é vasto. O ecossistema *big* data inclui tecnologias como bases de dados e sistemas distribuídos de ficheiros, *software* dedicado a análise de dados e *data mining*, ferramentas de integração e transformação de dados, motores de pesquisa sobre os dados, serviços de *clusters* e aplicações de visualização de dados.

Este subcapítulo, foca-se na descrição das diferentes bases de dados e nas duas *frameworks* mais relevantes dos últimos anos (Apache Hadoop e Apache Spark).

3.3.1 BD NoSQL

NoSQL, por vezes referenciado por "Not Only SQL", caracteriza-se por ser um conjunto de sistemas de bases de dados que pretendem oferecer alternativas à rigidez de sistemas relacionais e à linguagem SQL (*Structured Query Language*). Foram desenhadas para reduzir a complexidade de integração de um sistema de armazenamento com o código de aplicações, facilitando ao programador o armazenamento de informação.

Talvez um dos aspetos mais importantes, seja o facto de alguns dos tipos de bases de dados NoSQL permitem modelos de dados dinâmicos, o que os torna ideais para o armazenamento de tipos de dados semiestruturados e não estruturados. Desta forma, a base de dados não impõe muitas restrições sobre a estrutura da informação armazenada, sendo muito mais fácil a adaptação de aplicações de negócio a novos requisitos.

Ao contrário das bases de dados relacionais que dividem relações entre diferentes tabelas, estes sistemas armazenam toda a informação relacionada numa única estrutura, evitando o uso de operações de agregação de dados a favor de melhor desempenho e escalabilidade. E*stas* bases de dados estão desenhadas para garantir disponibilidade dos dados através de técnicas automáticas de distribuição de armazenamento sobre várias máquinas (*sharding*), sendo ideais para casos de aplicabilidade em que o grande volume de dados é um problema.

Dentro de NoSQL existem quatro tipos de bases de dados:

3.3.1.1 Chave-Valor

O armazenamento nestas bases de dados é o mais simples das quatro e é semelhante a uma tabela *hash*. A informação é armazenada tendo em conta dois campos: uma chave única que identifica cada registo e o valor a ser armazenado, que pode assumir qualquer tipo de dados (primitivo ou não). A manipulação da base de dados é feita a partir de APIs que transparecem operações de PUT, GET e DELETE, para adicionar, obter e remover registos. Os *lookups* são feitos utilizando o valor das chaves, tornando-os eficientes e facilmente escaláveis. Ainda dentro deste universo, existem propriedades distintas que caracterizam diferentes bases de dados. Alguns destes sistemas suportam consistência configurável, ordenação das chaves, armazenamento em memória RAM ou armazenamento em discos convencionais. A utilização deste tipo de base dados não deve ser considerado se forem pretendidos inquéritos baseados em informação contida nos registos. As bases de dados mais utilizadas neste grupo incluem Redis, Riak e Memcached.

3.3.1.2 Documento

Em bases de dados deste tipo, o armazenamento dos dados é feito em documentos cuja estrutura é semelhante à estrutura JSON agrupados em coleções. A modelação de um documento assemelha-se a programação orientada a objetos, em que cada documento possui um conjunto de campos de diversos tipos de dados (e.g. *string, date, array, etc.*). A flexibilidade desta estratégia de armazenamento é ainda mais evidenciada pelo fato de que é possível adicionar campos a documentos já persistidos, e por consequência, diferentes documentos na mesma coleção podem conter campos diferentes.

Em comparação com as bases de dados chave-valor, o sistema tem conhecimento do tipo de dados que se encontra em cada registo e permite assim, a realização de inquéritos baseados em valores particulares a cada campo. No entanto, esta funcionalidade implica a criação de índices específicos que podem comprometer o desempenho do sistema.

Este sistema será o mais indicado para o armazenamento de dados semiestruturados e para aplicações que necessitem de acesso aos dados especificando condições de filtragem. Nesta categoria destaca-se MongoDB, CouchDB e CouchBase.

3.3.1.3 Família de Colunas

Bases de dados deste tipo apresentam algumas semelhanças com bases de dados relacionais e bases de dados chave-valor. A estrutura de armazenamento é mais complexa comparativamente às duas anteriores e inclui 5 elementos: *keyspace* ou *keystore*, colunas, super-colunas, famílias de colunas e super-família de colunas.

Keyspace serve de contentor para múltiplas famílias de colunas. Estas famílias podem ser pensadas como tabelas das bases de dados relacionais, uma vez que armazenam múltiplas linhas. Cada linha é representada por um tuplo chave-valor, no qual a chave identifica de forma única o registo, e mapeia um conjunto de colunas. Às colunas, estão associadas um nome, um valor, e uma *timestamp* que dita a data de inserção ou última modificação. As colunas podem

ser agrupadas formando uma super-coluna. Por fim, diferentes famílias de colunas podem ser agrupadas para formar uma super-família de colunas.

A Figura 5 ilustra um exemplo de um modelo de dados em Apache Cassandra adequado ao problema descrito na secção 2.2. No caso demonstrado, todos os registos associados a um analisador de energia são agrupados por uma chave de partição e a informação "cresce" horizontalmente.

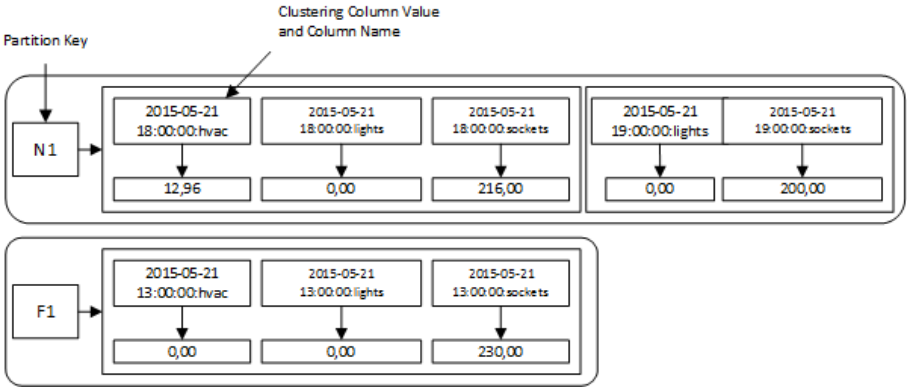

Figura 5: Modelo de dados de um exemplo implementado em Apache Cassandra.

Esta estrutura de dados peculiar permite que diferentes linhas possuam diferentes números e tipos de dados nas suas colunas. O tratamento de valores nulos é simplesmente realizado pela ausência de colunas associadas. No entanto, o tamanho de partição deve ser previamente estimado, uma vez que o descontrolo do seu tamanho pode resultar em problemas de desempenho. Uma vez que estas bases de dados usam a chave para efetuar pesquisas bastante rápidas, inquéritos que pretendam utilizar valores contidos nas colunas, só são possíveis após a criação desaconselhada de índices secundários. Outras bases de dados utilizadas nesta categoria incluem HBase e HyperTable.

3.3.1.4 Grafos

Tal como o nome indica estas bases de dados especializam-se no armazenamento de grafos e ao contrário das restantes bases de dados NoSQL, este tipo garante propriedades ACID. De uma maneira simples, um grafo é uma estrutura de dados caracterizada por um conjunto de vértices e um conjunto de arestas. Na modelação de um grafo, os vértices podem ser contextualizados a partir do armazenamento de diversos atributos sobre a forma de chave-valor. A ligação das arestas com os vértices estabelece uma relação explícita entre os vértices. Esta ligação deve ter sempre um nome para adicionar claridade semântica à estrutura do grafo. Poder ainda ser não direcional ou direcional, caso a aresta possua uma orientação de um vértice para outro, e possuir um conjunto de propriedades (e.g. peso, qualidade, custo, etc.) que podem ser ponderadas quando se percorre pelo grafo.

A aplicabilidade dos grafos abrange muitas áreas das ciências. Em informática, os grafos são estruturas ideais para armazenar relações entre utilizadores de uma rede social ou manter informação de uma topologia de uma rede ativa. Nestas bases de dados especializadas, o armazenamento das relações é feito diretamente. As três bases de dados deste tipo mais

influentes no contexto de NoSQL são: Neo4J, Titan e OrientDB. Ao contrário das outras duas, Titan não possui um sistema de armazenamento próprio e necessita de recorrer a Apache Cassandra, Apache HBase ou BerkeleyDB para persistir o grafo.

3.3.2 BD NewSQL

O termo NewSQL surge numa publicação de 2011 da empresa The 451 Group (Aslett, 2011) para descrever um novo conjunto de bases de dados relacionais, já existentes na altura, que prometiam melhorar a performance e escalabilidade de soluções MySQL. Bases de dados deste tipo são caracterizadas por tentarem oferecer o melhor de dois mundos, a escalabilidade que caracteriza os sistemas NoSQL, e a integridade, consistência e controlo de transações, associados às tradicionais bases de dados relacionais.

Para tal, apoiam-se no conceito ACID para o controlo de transações sobre um modelo de dados relacional, implementam interfaces SQL para as interações com o sistema, suportados por um sistema de concorrência não bloqueante entre as operações de leitura e escrita, e são capazes de serem utilizados de forma distribuída por vários nós autossuficientes. A implementação de uma base de dados NewSQL é ideal para empresas que queiram aplicar a sua experiencia e migrar aplicações SQL já existentes para um sistema ACID, capaz de lidar com os problemas de b*ig data*.

Atualmente, os sistemas NewSQL podem ser categorizados pelas diferentes abordagens adotadas na implementação das interfaces SQL e na forma como resolvem os problemas de escalabilidade e desempenho.

3.3.2.1 Novas arquiteturas de bases de dados

Este primeiro tipo refere-se a sistemas de bases de dados desenhados a partir de raiz, para cumprir os requisitos de desempenho e escalabilidade. Para se conseguir escalabilidade, a arquitetura é desenhada para o sistema funcionar de forma distribuída, em que cada nó é responsável por um subconjunto dos dados. Em termos de desempenho, estes sistemas são bastante mais rápidos que sistemas relacionais tradicionais, uma vez que procuram trabalhar com a memória RAM como sistema de armazenamento principal e disco para garantir durabilidade do sistema. Nesta categoria destacam-se VoltDB, MemSQL e NuoDB.

3.3.2.2 Sistemas transparentes de *clustering* e *sharding*

Caracterizados por reterem os sistemas transacionais no formato original, estes sistemas introduzem uma camada intermédia que permite a adoção de técnicas de *clustering* e *sharding*, que melhorem as bases de dados a nível de desempenho e escalabilidade.

Um grande beneficio deste tipo de sistema, é que reutilizam a experiência e implementações que as empresas já possuem, evitando em grande parte a reescrita de código e processos de migração. As principais soluções nesta categoria incluem dbShards e ScaleArc. Ambas são produtos comerciais que providenciam técnicas de *sharding* transparentes sobre bases de dados já existentes e balanceamento dinâmico de tráfego.

3.3.3 Frameworks

O papel das *frameworks* é a de proporcionarem um conjunto de ferramentas que resolvem múltiplos problemas numa linguagem comum. Apache Hadoop é uma das principais *frameworks open-source*, que propulsionou a evolução e o crescimento da área de *big data*. Mais recentemente, Apache Spark assume uma posição de renome, ao preencher as fraquezas de Hadoop no processamento analítico de dados. Desta forma, é importante salientar estas duas *frameworks* e as suas contribuições na área de *big data*.

3.3.3.1 Apache Hadoop

É uma *framework open-source*, escrita na linguagem de programação Java, que permite o armazenamento e processamento de grandes conjuntos de dados de uma forma distribuída. Hadoop foi durante muito tempo a ferramenta de eleição para muitos projetos de Big Data, e é a base de muitas ferramentas que pretendem expandir o conjunto de funcionalidades original (Apache HBase, Pig, Spark, Hive). A ferramenta é composta por quatro módulos: *Hadoop Common*, bibliotecas que apoiam os outros módulos; *Hadoop Distributed File System* (HDFS), camada de armazenamento distribuída e de acesso aos dados; *Hadoop YARN*, módulo que realiza a gestão de *clusters* (nós) e de agendamento de tarefas; *Hadoop MapReduce*, componente de programação de métodos de processamento de dados que executam paralelamente.

HDFS

Baseado no trabalho realizado pela Google no *Google File System* (GFS), é um sistema de ficheiros distribuído. O desenho da arquitetura do sistema foi pensado assumindo um conjunto de filosofias especificados na documentação oficial (The Apache Software Foundation, 2016), das quais se destacam:

- **Acesso aos Dados** – HDFS é desenhado para o processamento de dados em blocos e não se encontra otimizado para operações *ad-hoc* interativas.
- **Modelo de acesso *WORM*** – O acesso aos dados segue uma filosofia *write-once-read-many*. Alterações a ficheiros que já estejam armazenados são desaconselhadas, evitando problemas de coerência e melhorando das taxas de transferência.
- **Deslocação de computação** – Processamento dos dados é mais eficiente se este for efetuado perto da localização dos dados. São fornecidas *interfaces* para as aplicações se aproximarem dos dados, minimizando o congestionamento na rede e melhorando as taxas de transferência do sistema.

A arquitetura de armazenamento do sistema HDFS segue uma configuração *Master-Slave*, composta por duas entidades: NameNode e DataNodes. O NameNode guarda todos os meta-dados necessários para a gestão do sistema de ficheiros (localização dos ficheiros, permissões, *namespaces* e cotas de disco.

Durante a gravação de um ficheiro, o seu conteúdo é dividido em blocos e replicado (tipicamente 3 vezes) por múltiplos *DataNodes*. Se uma aplicação cliente pretender consultar um ficheiro, contacta o *NameNode* para receber as localizações dos blocos, e em seguida, lê diretamente do *DataNode* mais próximo. Para a escrita, o *NameNode* é solicitado para devolver quais os *DataNodes* adequados e a escrita procede-se diretamente sobre os nós. O *design* de utilização normal deste sistema possui um *NameNode* por cluster e múltiplos *DataNodes* por cluster.

Hadoop MapReduce

É uma *framework* para desenvolver aplicações que executam, paralelamente, métodos de análise sobre grandes conjuntos de dados armazenados nos *clusters*. A programação dos métodos segue o modelo de programação MapReduce introduzido pela Google (Dean and Ghemawat, 2004). Uma operação MapReduce é dividida em duas funções: Map, função que recebe um par de entrada e produz um conjunto intermediário de pares chave-valor, e no fim agrupa valores associados à mesma chave e encaminha-os para a função Reduce; Reduce converge todos os valores intermediários associados à mesma chave intermediária, possivelmente criando um conjunto menor de valores. A Figura 6 ilustra um exemplo de uma operação MapReduce que consiste em contar o número de palavras agrupadas pelo número de caracteres.

Map	**Combine**	**Reduce**
6: Hadoop	6:[Hadoop]	6:1
5: Spark	5:[Spark, Neo4J, Titan]	5:3
5: Neo4j	4:[Riak]	4:1
5: Titan		
4: Riak		

Figura 6: Exemplo de uma operação MapReduce.

Em Hadoop, a distribuição das funções pelas diferentes máquinas é orquestrada pelo Hadoop YARN. YARN segue uma arquitetura *Master-Slave* semelhante à do HDFS, em que existe um mestre *Resource Manager* (tipicamente noutra máquina que não a do *NameNode*) que comunica com os diferentes *NodeManagers* das restantes máquinas. Os gestores de cada nó informam o *ResourceManager* sobre os seus recursos disponíveis e processam as funções de *Map* ou *Reduce* de acordo com as decisões tomadas pelo mestre.

Ferramentas relacionadas-

Apache Hadoop revelou-se uma das mais populares ferramentas de Big Data, no entanto a sua orientação para análises em blocos, a acentuada curva de aprendizagem na implementação de funções *MapReduce* mais complexas e a incapacidade de servir muitos casos de uso, conduziu ao aparecimento de um conjunto de ferramentas que introduziram novas funcionalidades. A Tabela 7 enumera algumas das principais ferramentas que surgiram para colmatar as deficiências de Hadoop.

Tabela 7: Outras ferramentas Big Data.

Ingestão de dados	
Apache Flume	Serviço distribuído para capturar, agregar e servir grandes quantidades de dados a diferentes sistemas de armazenamento. Inicialmente desenvolvido apenas para trabalhar com HDFS.
Apache Kafka	Sistema distribuído que segue uma arquitetura *publish-subscribe* para capturar e servir dados semelhante a filas de mensagens. Diferentes produtores submetem informação para tópicos, e consumidores subscritos aos tópicos são alertados de novos dados. Ferramenta útil no desenvolvimento de *pipelines* de análise de dados em movimento, a partir de integração com Spark, Storm ou Flink.
Apache Sqoop	Ferramenta para transferir dados de bases de dados relacionais para HDFS.
Processamento de dados	
Apache Spark	*Framework* de computação distribuída alternativa ao *Hadoop MapReduce*. Dispõe de bibliotecas de algoritmos de aprendizagem, processamento de grafos, análise de dados em movimento e manipulação de dados numa linguagem semelhante a SQL.
Apache Storm	Ferramenta de processamento e análise de grandes fluxos de dados em tempo-real. Semelhante a SparkStreaming.
Apache Flink	Outra *framework* de processamento de dados em movimento. Pode ser comparada diretamente com Spark, uma vez que também dispõem de bibliotecas de processamento de grafos e algoritmos de aprendizagem máquina.
Apache Pig	Motor de processamento de dados que executa por cima de Hadoop MapReduce. Dispõe de uma linguagem procedimental, PigLatin, para realizar operações sobre dados (e.g. *join, filter, sort*). Estas operações são traduzidas em funções MapReduce automaticamente.
Apache Giraph	Ferramenta de processamento de grafos em Hadoop utilizando o modelo de programação MapReduce.
Gestão de Sistemas	
Apache Mesos	Gestor de recursos e agendador de tarefas de um *cluster*. Tipicamente usado para gerir *clusters* que não são de Hadoop, como por exemplo, Spark *clusters*.
Apache Zookeeper	Serviço de coordenação para a criação de sistemas distribuídos altamente disponíveis. Principalmente usado para cobrir pontos de falha únicos em sistemas, como é no caso de *clusters* geridos por YARN ou Mesos.

3.3.3.2 Apache Spark

Apache Spark é uma *framework open source* desenvolvida em 2009 pelo grupo AMPLab da Universidade da Califórnia em *Berkeley* e aberta à comunidade como projeto Apache em 2010. Na sua base, é um conjunto de ferramentas de processamento analítico de dados distribuídos, construído para garantir velocidade de análise, facilidade de uso e suporte a métodos analíticos sofisticados.

Spark surge para substituir o paradigma MapReduce utilizado até à data pelo Hadoop MapReduce na realização de processamento de dados, introduzindo o conceito de *resilient distributed dataset* (RDD) em memória, chegando a ser 100 vezes mais rápido do que MapReduce na mesma operação, ou 10 vezes mais se o RDD estiver no disco (Xin *et al.*, 2013).

Cada um dos módulos oferece um conjunto de bibliotecas que permitem a criação de aplicações nas linguagens de programação Scala, Java e Python. No entanto, em certos módulos, nem todas as funcionalidades se encontram disponíveis em Java ou Python.

Sendo *open*-source, programadores podem contribuir para o repositório comunitário de pacotes *third-party*, expandindo ainda mais o conjunto de funcionalidades. Até à data, este repositório contém mais de 150 bibliotecas com funcionalidades que vão desde conetores com diversos sistemas de armazenamento, até implementações de algoritmos de aprendizagem.

Core

SparkCore contém um conjunto de funcionalidades básicas que suportam o sistema inteiro. É aqui que estão implementados componentes para gestão e agendamento de tarefas, gestão de memória, recuperação de falhas e interação com fontes de dados. Neste módulo, é exposta uma API que define a abstração de dados *Resilient Distributed Dataset* (RDD), que é fundamental para o funcionamento do Spark e dos módulos complementares. Um RDD é uma coleção de registos, particionada, *read-only*, resiliente e imutável, que pode ser construído a partir de dados de um sistema de armazenamento (e.g. HDFS, bases de dados NoSQL, etc.) ou transformações aplicadas sobre outros RDDs (Zaharia *et al.*, 2012). As operações que podem ocorrer sobre um RDD são: Transformações, funções que produzem outro RDD (e.g. agregações ou filtragens); e Ações, operações que produzem um resultado (e.g. *reduce* ou *count*). Pode-se pensar num RDD, como se se tratasse de um intermediário entre dois sistemas de armazenamento (*Input* e *Output*), no qual são efetuadas operações de análise de forma distribuída por um *cluster* e, sempre que possível, em memória.

SparkStreaming

Extensão do módulo *core* de Spark que permite a criação de aplicações que processam dados à medida que estes vão chegando ao sistema. Para consumir dados existem bibliotecas de integração com diversas fontes de dados como Apache Kafka, Flume, HDFS, Kinesis ou Twitter.

SparkStreaming divide os dados recebidos em *micro-batches*, construindo uma estrutura denominada DStream (*Discretized Streams*). Uma DStream é representada por um conjunto de RDDs que contém dados de um determinado intervalo de tempo. Qualquer operação aplicada sobre uma DStream é aplicada aos RDDs. Para além de possuir integração com as fontes anteriormente mencionadas, o utilizador pode programar *Receivers* customizados que trabalhem com outros sistemas fonte através de *sockets* TCP (*Transmission Control Protocol*). Após a criação de uma DStream, podem ser aplicadas operações de transformação (e.g. *map*, *reduce*, *filter*, etc.), algoritmos dos módulos SparkMLlib e SparkGraphX, ou inquéritos SQL usando SparkSQL

MLlib

MLlib é uma biblioteca de algoritmos de aprendizagem automática. Integra-se automaticamente com o Spark para executar os algoritmos de forma escalável e distribuída sobre grandes conjuntos de dados. Atualmente suporta algoritmos comuns de aprendizagem incluindo modelos de classificação e regressão, *clustering*, redução dimensional e filtragem colaborativa.

Para além desta biblioteca, Spark pode ser integrado com a ferramenta Apache Mahout e com a linguagem de programação R através da biblioteca SparkR. Mahout foi originalmente desenvolvido para trabalhar com soluções Hadoop e executar os métodos seguindo o modelo MapReduce, no entanto, desde a versão 0.10, tem vindo a integrar novas ferramentas como o Spark. SparkR é um pacote para a linguagem R, em que é possível importar RDDs e realizar operações de agregação, filtragem e seleção no ambiente R.

SparkSQL

SparkSQL, lançado em 2014, é um dos módulos mais recentes que compõem a *framework* Spark. É uma evolução do sistema Shark, uma interface relacional, que implementava funcionalidades normalmente presentes em sistemas RDMBS para manipulação de dados estruturados (Xin *et al.*, 2013). SparkSQL é uma ferramenta que possibilita a manipulação de dados através de inquéritos escritos na linguagem SQL e conectividade com diversas bases de dados relacionais através de drivers JDBC (*Java Database Connectivity*).

GraphX (alpha)

É um componente de Spark para construção e computação paralela de grafos, mas não é uma base de dados orientada a grafos como Neo4j ou Titan. GraphX trabalha sobre uma variante do sistema Pregel apresentado pela Google (Malewicz, Austern and Bik, 2010). A ferramenta inclui um conjunto de operadores para a criação e manipulação, de forma distribuída, de grafos e vários algoritmos orientados para a análise dos mesmos. De momento, os algoritmos suportados são: *PageRank*, *ConnectedComponent* e *TriangleCount*.

3.4 Arquiteturas

Em implementações de Big Data existem duas importantes abordagens arquiteturais diferentes, Lamda[3] e Kappa[4]. As duas arquiteturas são desenhadas para resolver os problemas inerentes a cenários de tempo-real e de análises periódicas sobre dados históricos. Da mesma forma, é normal utilizarem o mesmo conjunto de tecnologias de bases de dados e de *frameworks*, em que a diferença reside na implementação do armazenamento, e análise e disponibilização dos resultados, associada aos dados históricos.

Arquitetura Lambda

Lambda, introduzida por Nathan Marz (2011; Marz and Warren, 2015), define três camadas para construção de uma solução Big Data: camada de análise em *batch*, camada de análise em tempo-real e uma camada de armazenamento de baixa latência.

Assim que novos dados entram no sistema, estes são servidos, ao mesmo tempo, à camada *batch* e à camada de tempo-real, em que cada uma das partes efetua o seu tipo de análise. Os resultados das operações de análise são posteriormente armazenados numa base de dados.

[3] Sítio: http://lambda-architecture.net
[4] Sítio: http://www.kappa-architecture.com

A Figura 7 demonstra um exemplo de uma arquitetura lambda, em que é usado Hadoop para análise em *batch* e Storm para análise em tempo-real. No exemplo, utiliza-se apenas uma base de dados para servir os dois tipos de resultados em tabelas, no entanto, outras implementações efetuam a separação ao nível acima, recorrendo até duas bases de dados diferentes, o que pode implicar a execução de operações de agregação de resultados.

A maior desvantagem deste tipo de solução, é a manutenção do código necessária para produzir os mesmos resultados obtidos em diferentes sistemas distribuídos, por vezes desenvolvidos em linguagens de programação diferentes.

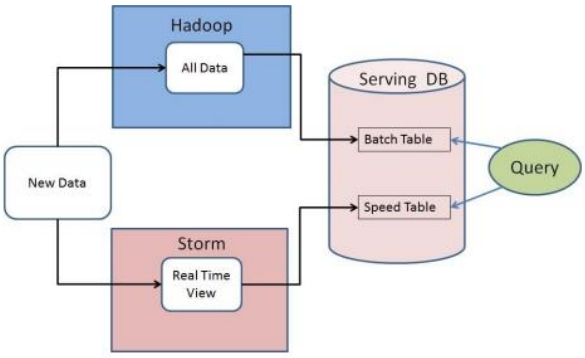

Figura 7: Exemplo de uma implementação seguindo a arquitetura Lambda. Fonte: (Astakhov & Chayel 2015, Figure 2).

Um sistema desenhado desta forma consegue cumprir todos os possíveis cenários de processamento de dados, mas introduz um maior nível de complexidade na gestão do sistema dependendo das diferentes ferramentas usadas.

Arquitetura Kappa

Kappa, inicialmente implementada no LinkedIn e posteriormente definida por Jay Kreps (2014), é uma arquitetura que se foca principalmente nos dados em movimento. Neste tipo de solução, não existe uma ferramenta que armazene toda informação no seu estado puro indefinidamente, como se verifica, por exemplo, com a utilização de Apache Hadoop na camada de *batch* da arquitetura Lambda.

A base deste tipo de topologia, é unificação das camadas de *batch* e de tempo-real em apenas uma, capaz de resolver os dois problemas e mantendo o desenvolvimento de processos de análise numa só tecnologia. Para tal, utiliza um sistema de ficheiros *log* imutáveis (p.e. Apache Kafka ou Apache Samza) para reter a informação no seu estado puro durante um período de tempo concordante com o intervalo de tempo de execução de análises em *batch*. Por exemplo, se é necessário reprocessar em intervalos de 30 dias, a informação reside no sistema de ficheiros o mesmo período de tempo, e as análises são efetuadas através da mesma *framework* que realiza análises em *streaming*. Assim, este tipo de solução não é indicado para casos de uso que necessitem de toda a informação fonte a qualquer altura, sendo indicada para situações onde o período de retenção de dados é conhecido

3.5 Soluções Big Data no Contexto das Smart Grids

Embora nem todas sejam diretas implementações das arquiteturas previamente apresentadas, a Tabela 8 descreve algumas soluções, no contexto geral de Big Data e de Smart Grids, que utilizam ferramentas consideradas nos dois tipos de arquiteturas.

Tabela 8: Exemplos de implementações da arquitetura Lambda.

Referência	**Descrição**
(Merlino and Yang, 2014)	Implementaram uma solução baseada na arquitetura Lambda. Apache Kafka ingere dados de fontes externas e alimenta as duas camadas de análise. Hadoop é usado na camada *batch* e Storm para análise de tempo-real. Apache Druid é usado como base de dados para servir aplicações cliente.
(Edelson, 2015)	Sugere uma implementação da arquitetura Lambda, recorrendo a Spark, Akka, Kafka e Cassandra. Spark é capaz de realizar ambos os tipos de análise. Cassandra é escolhida para persistir e servir os dados.
(Mayilvaganan and Sabitha, 2013)	Sugerem um ambiente na *cloud* baseado na arquitetura Lambda, que utilizaria Apache Cassandra como sistema principal de armazenamento de dados de consumo de clientes e informação meteorológica. Hadoop e MapReduce seriam utilizados para a descoberta de padrões e previsões de consumo.
(Zhou *et al.*, 2016)	Destacam a importância de informação obtida em redes constituídas por PMUs e o problema no armazenamento e processamento dos dados. Identificam dois tipos de aplicações em tempo-real, com requisitos de tempos de resposta diferentes. Embora os autores tenham explorado tecnologias NoSQL, a arquitetura descrita inclui: openHistorian 2.0 como sistema de armazenamento de dados ACID, facilmente integrável com *software* SCADA. A análise de dados periódica é realizada num *cluster* com Spark e HDFS.
(Cheng *et al.*, 2015)	Introduzem um sistema de armazenamento e de análise de dados no contexto de IoT e *smart cities* denominado CiDAP. O sistema utiliza dados de mais de 15 000 sensores instalados na cidade de Santander em Espanha durante um período de 3 meses. CiDAP utiliza CouchDB para armazenamento e acesso aos dados, e um *cluster* de HDFS e Spark para análises mais complexas. Realizam experiências preliminares sobre uma instância de CouchDB, e avaliam as taxas de dois tipos de inquéritos por segundo. Atingem os maiores valores de 471 e 34, para inquéritos simples e complexos respetivamente.
(Ganesh H.B., Kumar S. and Poornachandran, 2015)	Usam dados em intervalos de uma hora de PMUs obtidos a partir da Texas Synchrophasor Network[5] e alimentam um cluster de 4 nós com Apache Spark e Cassandra.
(Wang and Xiao, 2012)	Dados provenientes de um sistema, denominado CAC, de monitorização de condições de rede de energia espalhada por cidades chinesas, são direcionados para um *cluster* de 10 máquinas a correr Hadoop e HBase em máquinas virtuais individuais.

[5] Sítio: https://web.ecs.baylor.edu/faculty/grady/Texas_Synchrophasor_Network.html

A implementação de um sistema distribuído é uma tarefa complexa e uma vez que as abordagens enumeradas não especificam detalhes de implementação (e.g. *hardware* usado) é impossível avaliar e comparar as soluções apresentadas de forma pragmática.

3.6 Avaliação Tecnologias

Um dos maiores requisitos de aplicações no contexto das *smart grids* é o baixo tempo de latência presente, tanto na ingestão dos dados por parte dos sistemas de armazenamento, como nos tempos de resposta nas análises dos dados. Como tal, a avaliação das abordagens foca-se na avaliação de desempenho (*benchmarks*) realizada por terceiros.

A escolha de *benchmarks* incidiu nas que realizaram testes às tecnologias mencionadas na secção 3.3. Desta forma, a avaliação apresentada não é representativa de todo o espectro de *big data*, mas serve de base para a tomada de decisão das ferramentas consideradas na solução proposta. Uma vez que as tecnologias mencionadas sofrem atualizações regularmente, foram selecionados *benchmarks* que não precedessem o ano 2015.

Em (Swaminathan and Elmasri, 2016), os autores comparam MongoDB, Cassandra e HBase, em diferentes cargas de trabalho, com conjuntos de dados de diferentes tamanhos (1GB, 4GB, 10GB e 40GB), à medida que aumentam o número de máquinas no *cluster*. No cenário de leituras e escritas, Cassandra revelou-se ser a base de dados com melhor desempenho, seguida de Hbase e MongoDB. Os resultados das experiências indicam que Cassandra é a base de dados com melhor desempenho em cenários de carga de leitura e escrita, e na execução de *range queries* (leitura de vários registos em blocos). MongoDB é a base de dados indicada para cenários de apenas leituras e Hbase para apenas escritas. Por fim, quando se apenas considera o conjunto de dados de 40GB, Cassandra destaca-se em 4 das 5 cargas de trabalho.

Numa comparação entre Redis, Cassandra e MySQL, Redis é a base de dados com melhor desempenho na generalidade das diferentes cargas de trabalho, atingindo o maior número de operações por segundo e os menores valores de latência, seguida de MySQL e em último lugar Cassandra (Souza and Santos, 2015).

Carlos Costa e Maribel Yasmine Santos (2016), realizam uma análise ao estado de arte de *benchmarks* a bases de dados NoSQL. Desta análise, os autores retiram conclusões relativas a Cassandra, Hbase e MongoDB. No geral, Cassandra é que mais beneficia do aumento de nós no *cluster*¸ demonstra bons níveis de escalabilidade e demonstra bom desempenho em cargas de trabalho mistas. Hbase assegura dos melhores desempenhos em cenários de escrita intensiva, mas sofre em casos que envolvam leituras e escritas simultâneas. MongoDB apresenta os melhores resultados em situações de leituras intensivas, no entanto, é a base de dados, na generalidade dos casos, menos escalável das três.

Em (Ribeiro and Bernardino, 2013), os autores realizam experiências de avaliação de desempenho que comparam MySQL, Drizzle TokuDB, NuoDB, VoltDB e StormDB. As experiências utilizam dois conjuntos de dados (1GB e 10GB) e seguem o modelo de dados relacional. No processo de carregamento do conjunto mais pequeno, TokuDB foi o sistema que menos demorou. Na avaliação dos tempos de execução a uma bateria de inquéritos, TokuDB e StormDB são os sistemas com melhor desempenho em ambos os conjuntos de dados. Importante salientar que MySQL sofre de problemas de escalabilidade ao desempenhar bastante pior nos testes com o conjunto de dados de 10GB. As bases de dados VoltDB e NuoDB excederam a memória RAM utilizável devido à sua natureza *in-memory*.

Concluindo, no que diz respeito aos estudos procurados, salienta-se a presença frequente de Cassandra, Hbase e MongoDB nas avaliações de desempenho, corroborando também o nível de popularidade associado a estes sistemas. É importante relembrar que apesar de serem todas bases de dados NoSQL, MongoDB é uma base de dados possui um modelo de dados bastante diferente das outras duas, critério que tem mais importância do que resultados de desempenho. Por outras razões, Hbase deve ser considerada se já existe experiência prévia a trabalhar com Apache Hadoop.

No caso das bases de dados NewSQL, o grau de adoção não se encontra ao mesmo nível das bases de dados NoSQL, o que influencia o número de *benchmarks* independentes recentes e a consolidação de uma metodologia comum a diferentes autores.

Os resultados de avaliações de desempenho são fundamentais para a tomada de decisão de uma base de dados. No entanto, quando se comparam os tipos de sistemas apresentados, critérios como a necessidade de seguir padrões ACID ou BASE, o modelo de dados a adotar e as capacidades na manipulação da informação armazenada, assumem equivalente ou mais importância do que os níveis de desempenho.

4 Solução Proposta

A solução proposta, baseada na SMACK (Spark, Mesos, Akka, Cassandra, Kafka) *stack*, consiste na integração das diferentes aplicações apresentadas na Figura 8 e foca-se principalmente na flexibilidade e integração das diferentes ferramentas. A Figura 9 ilustra a estrutura da solução proposta.

A SMACK *stack* consiste na implementação de uma solução que se situa numa posição intermédia entre as duas grandes arquiteturas. Distancia-se da arquitetura Kappa, uma vez que mantém as funcionalidades de análise em *batch* esperadas na tipologia Lambda, mas mantém todo o desenvolvimento de serviços e aplicações de análise numa única *framework*, Apache Spark (Estrada and Ruiz, 2016).

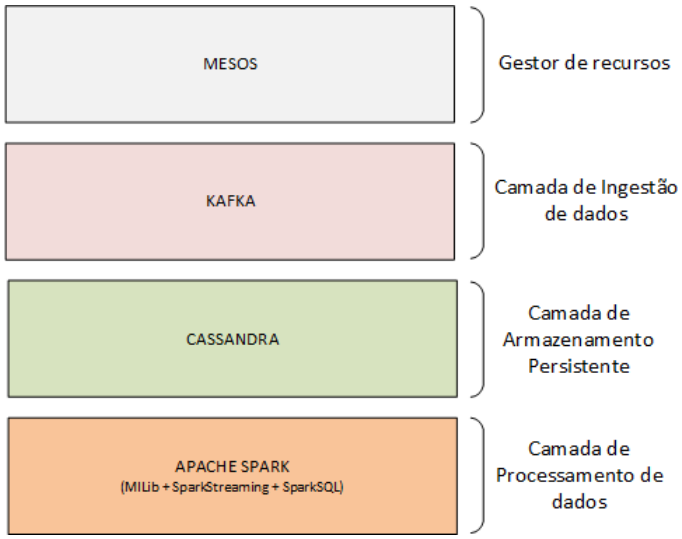

Figura 8: *STACK* de ferramentas principais.

Spark foi escolhido uma vez que permite a construção de uma *pipeline* capaz de realizar análise em tempo-real e aplicar algoritmos de aprendizagem máquina numa única *framework*, com níveis de desempenho superiores à utilização de MapReduce. Por forma a executar aplicações de processamento de Spark, um gestor de recursos tem que ser escolhido. A opção de Hadoop YARN foi preterida a favor de Apache Mesos, uma vez que a *stack* não envolve nenhuma tecnologia Hadoop para o qual o YARN está otimizado e pelo facto de o Mesos ser compatível com a base de dados escolhida. O Mesos abstrai os recursos de *hardware*, permitindo a construção e execução de sistemas distribuídos (*frameworks*) em contentores isolados. Para isso, cada *framework* implementa um *scheduler* (agendador) responsável por comunicar com o Mesos e um executor que inicializa as tarefas de execução.

A ingestão dos dados será realizada pelas ferramentas Apache Kafka e conectores relacionados. O objetivo é que as diversas fontes periódicas enviem os seus dados para o Kafka para armazenamento intermédio disponível também para operações de agregação ou reencaminhamento com o uso de Spark.

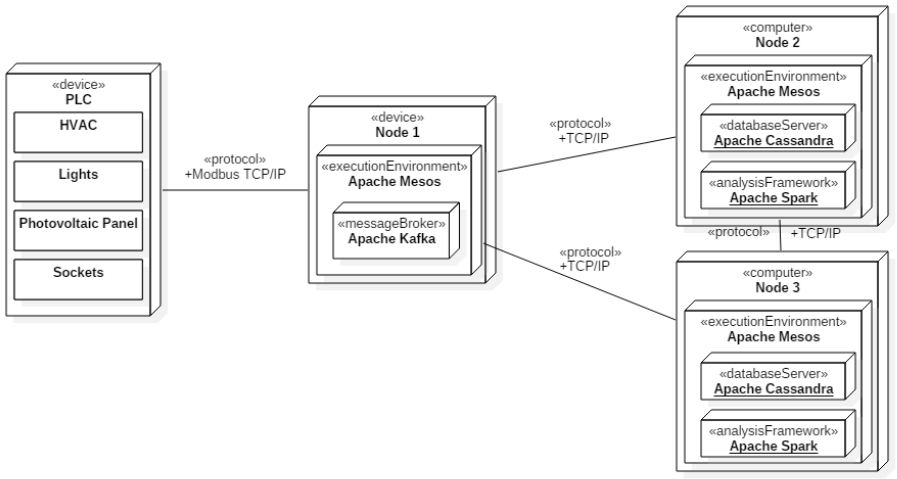

Figura 9: Arquitetura do sistema distribuído (UML *Deployment Diagram*).

Uma vez que Spark ocupará grande parte da memória disponível das máquinas e o problema não necessita propriedades ACID, optou-se pela base de dados NoSQL Cassandra. Sendo uma base de dados deste tipo, os padrões relacionados com bases de dados relacionais não se aplicam. A escolha desta base de dados prende-se nas necessidades de desempenho que oferece e na fácil integração com a *framework* Spark através do conetor implementado pela empresa DataStax [6] . Como já mencionado, Cassandra pode servir como sistema de armazenamento de grafos com a ferramenta Titan. Desta forma, na eventualidade de ser necessário armazenar informação indicada para grafos (e.g. topologia da rede de sensores), reutiliza-se conhecimento e diminui-se o tempo de desenvolvimento.

[6] Sítio: https://github.com/datastax/spark-cassandra-connector

O modelo de dados a usa basear-se-á em *Time Series Data* e seguirá as sugestões recomendadas em (McFadin, 2015). Para garantir que não há perda de dados em caso de ocorrência de falhas dos nós de armazenamento, Cassandra será configurado para replicar os dados pelo menos uma vez. Num *cluster* de dois nós, esta configuração, permite que o sistema continue a funcionar após a falha de um dos nós. A arquitetura apresentada possui um ponto único de falha do sistema no nó 1. Mediante os recursos disponíveis poderão ser adicionados mais nós, para gerir vários Mesos *Leaders* e Kafka *Brokers* garantindo um sistema distribuído de alta disponibilidade.

4.1 Avaliação do Sistema

O sucesso da implementação do sistema dependerá essencialmente na capacidade de cumprir os requisitos especificados na Tabela 3, particularmente na capacidade de realização de análise em tempo-real. O cálculo contínuo de médias de consumo, e a criação e aplicação de um modelo de aprendizagem, em tempo-real, focado na previsão de consumos, servirão como prova do cumprimento deste requisito.

Uma vez que a o projeto se trata de uma solução alternativa à solução monolítica já existente no grupo GECAD, a avaliação do sistema recai na comparação destes dois sistemas e não com os mencionados na secção 3.4. Assim, a avaliação realizada através da formulação e execução de vários testes de desempenho e casos de estudos. Os testes a efetuar, baseiam-se na execução de *benchmarks* aos sistemas de armazenamento, e tem como objetivo determinar grau de escalabilidade através da observação do número de operações por segundo. O caso de estudo, compara os sistemas numa perspetiva exploratória e analítica dos dados, tendo como objetivo perceber se a distribuição de processamento em Spark resulta em melhorias nos tempos de resposta de diferentes inquéritos.

4.1.1 Testes de desempenho

O desempenho geral dos sistemas será avaliado utilizando a ferramenta YCSB[7] (*Yahoo*! *Cloud Serving Benchmark*), especificamente desenhada para avaliar soluções NoSQL (Cooper *et al.*, 2010). O YCSB, possui uma interface para cada base de dados, incluindo uma para interagir com *drivers* JDBC, que usa para executar diferentes cargas de trabalho padronizadas. As cargas de trabalho, descritas na Tabela 9, são executas sobre um modelo de dados chave-valor não relacional. Este é constituído por 11 colunas de texto, em que a primeira coluna é dedicada à chave primária e as restantes 9 a campos de texto com conteúdo aleatório.

[7]Sítio: https://github.com/brianfrankcooper/YCSB

Tabela 9: Cargas de trabalho do YCSB.

Carga de Trabalho	Descrição
A	Mistura de 50/50 leituras e escritas.
B	Carga de trabalho de execução de leituras. Rácio de 95/5 de leituras e escritas.
C	Apenas leituras.
D	Leitura dos registos mais recentes.
E	Leitura de pequenos blocos de registos agrupados.
F	Leitura, modificação e escrita. A aplicação efetua leituras individuais, modifica a informação e volta a guardar a informação.

A execução dos testes YCSB é parametrizável, incluindo as seguintes variáveis de entrada:

- **workload –** A carga de trabalho a ser executada;
- **threads** – O número de *threads* concorrentes para a execução da carga de trabalho. Pode ser usada para simular mais clientes a aceder à base de dados;
- **measurementtype –** Formato de apresentação dos resultados obtidos (histograma ou série temporal;
- **recordcount** – Número de registos a serem inseridos na fase de preparação;
- **operationcount** – Número de operações a ser atingido na execução da *workload*;
- **maxexecutiontime –** Tempo de execução máximo em segundos da *benchmark*.

Como *output*, a ferramenta disponibiliza informação sobre: *runtime*, tempo de execução total; *throughput*, taxa de execução de operações por segundo; *averagelatency,* latência média de todas as operações em milissegundos; e *95thPercentileLatency* e *99thPercentileLatency*, os valores de latência para 95% e 99% das operações efetuadas respetivamente.

A Tabela 10 enumera os diferentes testes de desempenho a executar. É importante salientar, que o cliente YCSB responsável pela execução, será instalado em máquinas não locais aos servidores de base de dados.

Tabela 10: Testes de desempenho YCSB.

Testes	Objetivo	Variáveis de entrada		Variável de Saída
AD1.1	Observar o comportamento de Cassandra c/ apenas um nó.	YCSB **Load phase** **recordcount** 500 000	Cassandra **ConsistencyLevel** ONE **ReplicationFactor** 1	**Overall Throughput (ops/sec)**
AD1.2	Observar as taxas de inserção de dados em ambas as arquiteturas. Verificar se a Cassandra consegue ingerir mais informação por segundo.	**Load phase** **recordcount** 500 000	**ConsistencyLevel** TWO **ReplicationFactor** 2	
AD2.1	Observar os valores de latência sobre diferentes cenários de carga.	**workload** A **operationcount** 250000		
AD2.2		**workload** D **operationcount** 250000 **maxexecutiontime** 60		

Inicialmente, serão considerados dois testes de carregamento de dados. Um primeiro teste (AD1.1) com apenas um servidor de cassandra; e o segundo (AD1.2) com dois nós de Cassandra e um de SQL Server. Outros dois testes (AD2.1 e AD2.2) serão realizados para a execução das cargas de trabalho: A e D respetivamente. As cargas escolhidas são as que mais se adequam às cargas normalmente observadas em conjuntos de dados de séries temporais no contexto do problema.

Com estes testes, pretende-se fundamentar a hipótese alternativa de que o sistema de armazenamento em Cassandra é mais escalável de que MsSQLServer. Como as experiências podem resultar num congestionamento da rede do GECAD, é importante que sejam realizadas em horas de trabalho menos concorridas e que o número de ensaios, não seja elevado ao ponto de ocupar a rede durante bastante tempo. Como tal, serão realizados 10 ensaios em cada um dos testes e será observada a métrica de operações por segundo. O teste estatístico a utilizar é o *Wilcoxon Signed Rank test*, justificável pelo o tamanho controlado das amostras e pelo fato do tipo de dados, normalmente, não descrever uma distribuição normal.

5 Implementação

O início da implementação começa pela instalação e configuração das máquinas representadas na Figura 9. Para tal, o GECAD disponibilizou a utilização de três máquinas individuais situadas nas instalações do edifício N. As máquinas pertencem à rede local do edifício, o que possibilita acesso direto aos PLC através da rede. Porém, o *hardware* cedido apresenta diferentes níveis de restrições na sua utilização. Dois dos computadores encontram-se totalmente dedicados ao desenvolvimento do projeto, sendo que uma deve manter o sistema operativo Windows de origem. A terceira máquina é utilizada periodicamente no alojamento de serviços no âmbito de outros projetos no GECAD, o que significa a partilha de recursos com algumas aplicações e a impossibilidade de mudança a nível do sistema operativo.

A implementação da solução com o Apache Mesos, implicou a utilização de outras duas ferramentas: Apache Zookeeper e Marathon. Apache Zookeeper mantém informação de configuração e estado de aplicações que interagem com a ferramenta Mesos numa estrutura de ficheiros estilo árvore. O Marathon é um serviço que permite o provisionamento de aplicações (p.e. execução de comandos, *jars*, outros serviços) através do ambiente contido do Mesos. O provisionamento é feito através de especificações de aplicações que indicam parâmetros como o comando a executar e os recursos necessários, que por sua vez são geridos pelo Mesos.

A Figura 10 apresenta uma visão detalhada da arquitetura implementada. A arquitetura é composta por 3 nós independentes a correrem o sistema operativo CentOS7. A justificação para a escolha deste sistema operativo é aprofundada no subcapítulo 5.1.

Cada um dos nós executa o serviço Zookeeper, que mantém informações, de forma replicada, dos últimos estados de execução das aplicações do *cluster*, permitindo a recuperação de aplicações a partir do último estado registado. O nó 1 é o único que possui o serviço de mestre do Mesos, servindo de ponto de entrada de comunicação das restantes ferramentas. Desta forma, as instalações de Kafka, Cassandra e Spark são fornecidas em *runtime* ao Mesos, a partir do serviço Marathon.

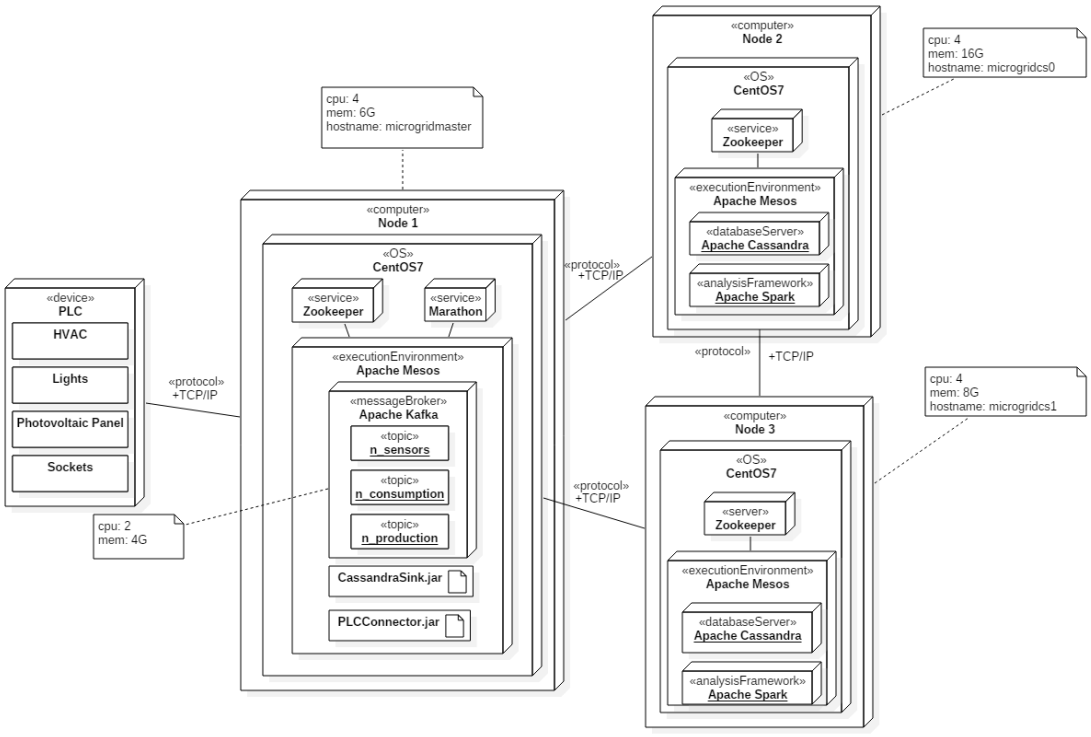

Figura 10: Visão detalhada da arquitetura do sistema distribuído (UML *Deployment Diagram*).

Para realizar a recolha dos dados em tempo-real, foi implementado o conector Java 'PLCConnector.jar', utilizando a API KafkaConnect[8] do Kafka. O conector comunica com o PLC e armazena os dados em tópicos do Kafka na qual a informação se encontra disponível durante um período de 30 dias. Os dados chegam ao destino final da base de dados de Cassandra, através de outro conector Kafka designado como 'CassandraSink.jar'. Este conector já se encontrava implementado pela empresa DataMountaineer, como parte da sua solução StreamReactor [9] . Desta forma, os dados encontram-se em três localizações diferentes. Replicados permanentemente pelo sistema interno da base de dados de Cassandra e temporariamente nas filas de mensagens em Kafka. Por fim, a submissão de tarefas de execução em Spark, é comunicada ao nó mestre e a posterior alocação dos recursos é restringida aos nós 2 e 3.

[8] Sítio: https://kafka.apache.org/documentation/#connectapi
[9] Sítio: https://github.com/datamountaineer/stream-reactor

O subcapítulo seguinte descreve a configuração e instalação de cada uma das ferramentas discutidas, realçando algumas das parametrizações mais importantes e as possíveis interfaces de interação com o sistema. O processo de modelação, migração e aquisição de dados é detalhado no subcapítulo 5.2. No final, são apresentados os resultados dos testes desempenho apresentados anteriormente.

5.1 Configuração e Instalação

A utilização das ferramentas apresentadas na solução proposta é normalmente realizada em sistemas operativos Linux. Desta forma, nas máquinas restritivas, foi utilizada a ferramenta Oracle VM VirtualBox na virtualização de máquinas Linux e foram apontadas como os nós escravos. O nó mestre não recorre a virtualização e o sistema operativo Linux é instalado de raiz. Os subcapítulos seguintes esclarecem alguns dos passos tomados na preparação da arquitetura.

5.1.1 Sistema Operativo

Numa fase inicial de experimentação individual das ferramentas, foi utilizado o sistema operativo Debian[10]. No entanto, os repositórios de pacotes, que permitem facilmente instalar e configurar aplicações, eram limitados. Mais ainda, o suporte a longo termo do sistema Debian 7 "Wheezy" é de cinco anos e termina no ano 2018, fator que pode ser considerado fundamental num contexto empresarial. Assim, a escolha final recaiu no sistema operativo CentOS 7 de instalação mínima.

O sistema escolhido é uma distribuição Linux baseada no sistema RHEL (Red Hat Enterprise Linux), que providencia uma plataforma de computação de nível empresarial, suportada pela comunidade e com um ciclo de vida de 10 anos.

Durante processo de instalação e configuração base do sistema operativo salienta-se os seguintes passos importantes:

- **Nome utilizador:** cada aplicação no *cluster* de Mesos, necessita de explicitar o nome de utilizador na sua execução. Serviços que correm em vários nós, como Cassandra e Spark, necessitam do mesmo nome de utilizador nas máquinas envolvidas. Desta forma, foram configurados utilizadores com o mesmo nome.
- **Seleção sistema de ficheiros**: A seleção do sistema de ficheiros recai no XFS. Este sistema de ficheiros é mais capaz em termos de escalabilidade e de manipulação de grandes quantidades de informação, o que o torna adequado ao contexto da solução.
- **Retenção arquivos de log**: O gestor de serviços do CentOS (systemctl) redireciona mensagens de *log* para um arquivador (journald) que realiza a manutenção de arquivos

[10] Sítio: https://www.debian.org

de mensagens dos serviços do sistema, cuja rotatividade pode ser controlada temporalmente ou por tamanho.

- **Acesso remoto**: A instalação por defeito inclui o serviço SSH (*Secure Shell*) para acesso remoto nas máquinas. Para além do acesso através de utilizador e palavra-passe, foi configurado o acesso através de chaves SSH. Desta forma, as máquinas permitem concretizar *logins* a partir de máquinas confiáveis.

A instalação mínima escolhida é suficiente para a instalação das ferramentas a utilizar, mas não inclui uma interface gráfica de do utilizador. A interação com as máquinas é feita maioritariamente por SSH e através da linha de comandos, com a ajuda do *software* MobaXterm [11], disponível para os sistemas operativos Windows, que inclui também um explorador de ficheiros gráfico através do protocolo SFTP (SSH *File Transfer Protocol*).

Depois de configurado, procedeu-se à experimentação das ferramentas de forma individual. Por consequência, recorreu-se aos sítios oficiais de cada ferramenta para a realização dos *downloads*. Em casos particulares, como o de Cassandra e Kafka, esta metodologia não é a indicada para a utilização conjunta com o Apache Mesos. Isto significa, que foi posteriormente necessário recorrer a projetos desenvolvidos pela comunidade para tal efeito.

Uma solução explorada, foi a da possível utilização da plataforma DC/OS [12] (Datacenter Operating System) desenvolvido pela empresa Mesosphere. O objetivo da plataforma é providenciar uma solução unificada, de todos os serviços/aplicações desenvolvidas para trabalhar com Apache Mesos. Esta solução é a indicada para uma instalação e configuração rápida de toda a SMACK *stack*. No entanto, apresentou restrições a nível de requisitos de *hardware* (demasiado elevados), sendo que um dos detalhes importantes, é a impossibilidade de conjugar os papéis de mestre e escravo na mesma máquina. Embora não utilizado de forma íntegra, a *Mesosphere* é proprietária do único repositório *open-source* ativo de Cassandra em Mesos. Embora parte da plataforma DC/OS, pode ser utilizado individualmente, tendo em conta alguns passos que são explicados no subcapítulo da configuração de Cassandra.

5.1.2 Zookeeper, Mesos e Marathon

O processo de instalação e configuração é semelhante nas três ferramentas. O descarregamento do *software* é realizado a partir de um repositório mantido pela empresa Mesosphere, através do gestor de pacotes 'yum' do CentOS. Este repositório contém diversas versões já compiladas do *software* mencionado, e automatiza a criação de ficheiros e diretórios organizados e prontos para serem utilizados.

[11] Sítio: https://mobaxterm.mobatek.net
[12] Sítio: https://dcos.io

Zookeeper

A instalação do pacote relativo a Zookeeper (mesosphere-zookeeper.x86_64 v3.4.10) cria os diretórios apresentados na Tabela 11. A configuração do servidor Zookeeper é relativamente simples e pode ser realizada em modo *standalone* (apenas uma instância) ou *HA* (*High Availability)*. Como apresentado na arquitetura, o Zookeeper foi instalado nas três máquinas, por forma a replicar a informação que nele é guardada. A qualquer momento, uma das máquinas serve como líder enquanto as outras se encontram em modo *standby*, continuando a responder a pedidos até que dois dos nós falhem.

Tabela 11: Ficheiros e diretórios relevantes da instalação Zookeeper.

	Tipo	**Caminho/Ficheiro**	**Descrição**
Zookeeper	**Configurações**	/etc/zookeeper/conf/zoo.cfg	Principal ficheiro de configuração do servidor. Contém a informação p/ uma instalação HA.
		/var/lib/zookeeper/myid	Ficheiro com o número para a identificação do servidor Zookeeper em modo HÁ
	Executáveis	/opt/mesosphere/zookeeper/bin	Contém os executáveis para inicializar o servidor e uma interface de linha de comandos p/ interações c/ o Zookeeper (zkCli.sh).
	Dados (*snapshots*)	/var/lib/zookeeper/version-2	Localização por defeito dos *snapshots* persistentes em disco.

Para configurar o Zookeeper em modo HA foi necessário configurar as propriedades apresentadas na Figura 11 no ficheiro zoo.cfg. A configuração requer a atribuição do endereço de IP/*hostname* de cada servidor ao seu identificador numérico único. Identificador que deve ser análogo ao encontrado no ficheiro 'myid'. Os números de portas 2888 e 3888 demonstradas, são as portas dedicadas para o protocolo interno de eleição do líder. A porta 2181 é a utilizada para comunicar com aplicações cliente (e.g. Mesos comunicar com o Zookeeper).

```
# Ensemble configuration
server.1=microgridmaster:2888:3888
server.2=microgridcs0:2888:3888
server.3=microgridcs1:2888:3888
```

Figura 11: Excerto de configuração HA de Zookeeper.

Outras propriedades importantes do ficheiro de configuração 'zoo.cfg' incluem: snapRetainCount, o número de *snapshots* mais recentes a reter; autopurge.purgeInterval, o intervalo de tempo em horas para a realização do processo automático de limpeza; e maxClientCnxns, o número máximo de conexões a estabelecer com clientes.

A configuração das comunicações entre o Zookeeper e as outras ferramentas limita-se à inclusão do URL (*Uniform Resource Locator*) do cluster de servidores Zookeeper (p.e. zk://microgridmaster:2181,microgridcs0:2181,microgridcs1:2181/Mesos) no parâmetro de configuração adequado.

Durante a implementação, a interação do utilizador com o *cluster* de máquinas de Zookeeper foi pela interface de linha de comandos 'zkCli.sh'. A Figura 12 apresenta um exemplo da execução dos comandos 'ls' e 'get' a partir da linha de comandos. O primeiro apresenta todos os *znodes* existentes a partir da raiz. O segundo comando obtém os meta-dados do próprio *znode*.

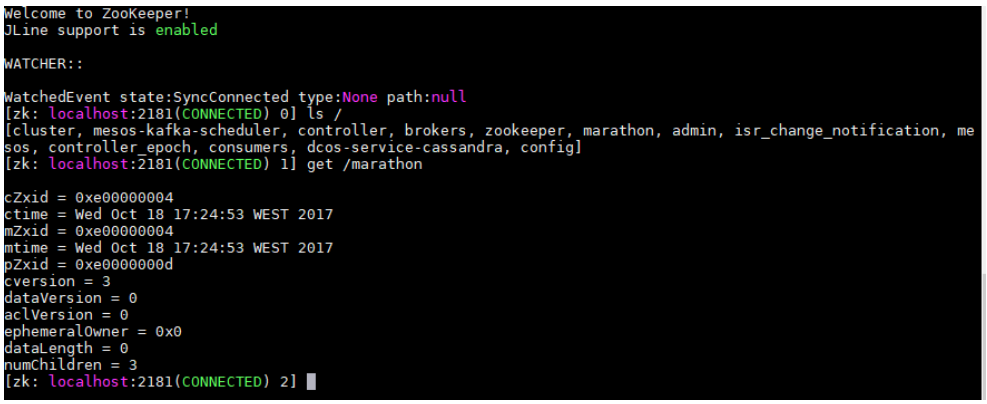

Figura 12: Exemplo de comandos executados pela interface de linha de comandos Zookeeper.

Alternativamente o utilizador pode recorrer a interfaces gráficas de utilizador como o ZooNavigator[13] ou zKui[14].

Mesos

A instalação e configuração do Mesos é mais extensa do que a do Zookeeper, bem como o funcionamento que se aplica à configuração de parâmetros. O pacote relativo ao Mesos (mesos.x86_64 v1.3.0) instala dois serviços de execução, 'mesos-slave' e 'mesos-master', que designam a responsabilidade de cada nó. Apenas a máquina mestre possui o serviço 'mesos-master' ativado, enquanto que o serviço 'mesos-slave' é executado em todas.

A Tabela 12 apresenta os diretórios e ficheiros mais relevantes para a configuração do Mesos. Ao contrário do Zookeeper, os serviços do Mesos não leem apenas um ficheiro de configuração, mas sim um diretório, no qual cada ficheiro representa um parâmetro e o seu conteúdo o valor. Por exemplo, para configurar o URL de Zookeeper (comum ao mestre e escravo), foi criado um ficheiro 'zk' no diretório '/etc/mesos/' com o valor que referencia os servidores Zookeeper.

[13] Sítio: https://github.com/elkozmon/zoonavigator
[14] Sítio: https://github.com/echoma/zkui

Tabela 12: Ficheiros e diretórios relevantes da instalação Mesos.

	Tipo	Caminho/Ficheiro	Descrição
Mesos	**Configurações**	/etc/mesos /etc/mesos-master /etc/mesos-slave	Diretórios que contêm os ficheiros de parametrização a propriedades comuns aos dois serviços, ao serviço mestre e ao serviço escravo respetivamente.
	Executáveis	/usr/bin/mesos-init-wrapper	*Script* invocado pelo serviço respetivo, responsável por aglomerar os parâmetros de configuração e executar os ficheiros de inicialização do servidor Mesos.
		/usr/sbin/mesos-master /usr/sbin/mesos-slave	Ficheiros de inicialização invocados pelo *wrapper* anterior. Nota: No diretório '/usr/bin/' encontram-se outros scripts, fora do contexto do gestor de serviços, para a iniciar o Mesos de forma alternativa.
	Dados	/var/lib/mesos/	Raiz da árvore de diretórios usada pelo Mesos para armazenar informação relativa aos ambientes de execução.
		/var/lib/mesos/slaves	Contém as pastas e ficheiros de cada tarefa de cada *framework* submetida para execução.
		/var/lib/mesos/volumes	Diretório onde se encontram os volumes montados pelo Mesos que dizem respeito a tarefas que necessitam de recursos de disco persistentes (p.e. o serviço de cassandra).

Os aspetos mais importantes da configuração envolveram a definição dos parâmetros responsáveis pelo controlo de acessos às páginas *Web* do Mesos, autenticação de máquinas escravas e *frameworks* com o mestre, os recursos (disco, *cpus*, *gpus*, portas de rede e memória) disponíveis em cada escravo, e as estratégias de recuperação em caso de falha de agentes.

A autenticação configurada para os efeitos mencionados é a autenticação CRAMM-MD5 que vem por defeito com o Mesos. Todas as *framework*/entidades que comunicam com o cluster mesos possuem uma credencial composta por um *principal* e um *secret*. A configuração faz também uso de *access lists* para delimitar as responsabilidades de cada entidade no acesso a recursos (p.e. *endpoints* da aplicação Web do Mesos, recursos de *hardware* das máquinas, etc.).

Com a configuração brevemente descrita atingem-se os seguintes efeitos:

1. Existência de uma credencial *admin* com todos os privilégios;
2. Existência de uma credencial para utilização da aplicação *Web* do Mesos;
3. Escravos estrangeiros ao mestre não se conseguem autenticar e registar com sucesso;
4. *Frameworks* não contempladas pelo mestre, não possuem privilégios de execução no *cluster*;

A configuração poder-se-ia ainda estender à limitação de recursos para cada uma das *frameworks*, mas levantou complicações devido à definição estática dos recursos disponíveis e à incerteza na real utilização dos mesmos. Concluindo, todas as *frameworks* possuem acesso a todos os recursos de *hardware* requisitados desde que estes se encontrem disponíveis.

Marathon

Os diretórios e ficheiros mais relevantes da instalação do pacote relativo ao Marathon (marathon.x86_64 v1.4.5) encontram-se demonstrados na Tabela 13. A configuração do serviço funciona da mesma forma do que a do Mesos. Parâmetros a passar ao serviço de Marathon são individualmente contidos em ficheiros contidos no diretório de configurações. Parâmetros do tipo *boolean* são configurados apenas pela existência do ficheiro no diretório, sem ser necessário ter conteúdo no ficheiro.

À semelhança dos serviços anteriores, o Marathon pode ser executado em modo HA. Para isso basta incluir um ficheiro com o nome 'HA' no diretório de configurações e ativar o serviço em cada nó. Nesse mesmo diretório, encontram-se as configurações necessárias para autenticação HTTP básica de acesso à aplicação Web, as credenciais de acesso para comunicação com o Mesos e o diretório de armazenamento de artefactos.

Tabela 13: Ficheiros e diretórios relevantes da instalação Marathon.

	Tipo	**Caminho/Ficheiro**	**Descrição**
Marathon	**Configurações**	/etc/marathon/conf	Diretórios que contêm os ficheiros de parametrização para o serviço Marathon.
	Executáveis	/usr/bin/marathon	Executável invocado pelo gestor de serviços respetivo, responsável por aglomerar os parâmetros de configuração e iniciar o Marathon.
	Dados	/var/log/store/	Localização do diretório de artefactos, onde se situam os ficheiros utilizados para *deployment* de aplicações. Configurável a partir da propriedade 'artifact_store'.

A interação com o Marathon é realizada a partir da aplicação *web* ou através da realização de pedidos HTTP à API REST. A Figura 13 apresenta parte da página *web* principal de interação com o Marathon, onde é demonstrado os agendadores dos diferentes serviços organizados num grupo de aplicações denominado 'schedulers'. A partir desta página é possível criar aplicações/grupos e visualizar o estado geral de todas as tarefas.

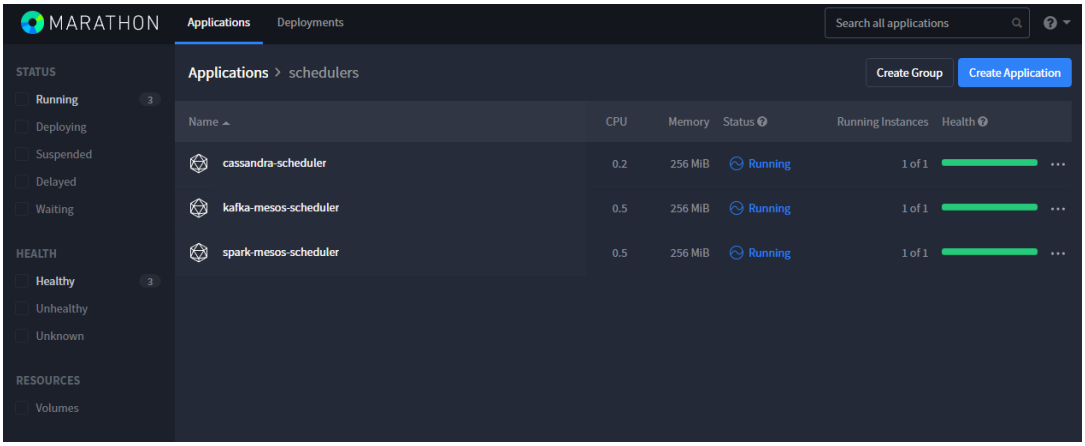

Figura 13: Página *Web* inicial do serviço Marathon.

O utilizador pode aceder a visões detalhadas da aplicação, na qual é apresentada a configuração utilizada na criação da mesma, bem como o seu estado e saúde, e os comandos para operações de escalabilidade, reinicialização e suspensão de execução. Os *deployments* de cada aplicação serão também vísiveis a partir da aplicação *web* principal do Mesos.

As instalações e configurações das restantes ferramentas diferem dos processos anteriores, uma vez que não usam um pacote pertencente ao repositório mencionado. A instalação do serviços é dependente dos pacotes existentes na *artifact store* do Marathon e dos agendadores a serem executados no nó mestre.

5.1.3 Kafka

A instalação de Kafka em Mesos envolve a utilização de um agendador específico e não apenas a partir da distribuição oficial da ferramenta. A Mesosphere dispõe de um agendador Kafka *open-source* num repositório GitHub[15], parte integrante da sua plataforma DC/OS. No entanto, existe um projeto alternativo[16], também mantido pela comunidade, que se ainda encontra ativo e atualizado para as mais recentes distribuições de Kafka. Esta segunda solução foi a escolhida para a instalação e utilização de Kafka, uma vez que não se encontra interligada com a plataforma DC/OS.

O repositório foi clonado e compilei o código fonte para gerar o ficheiro *jar* do agendador de Kafka. A Tabela 14 apresenta uma breve descrição dos ficheiros mais relevantes do projeto clonado.

[15] Sítio: https://github.com/mesosphere/dcos-commons/tree/master/frameworks/kafka
[16] Sítio: https://github.com/mesos/kafka

Tabela 14: Ficheiros relevantes no repositório de Kafka/Mesos.

	Tipo	Caminho/Ficheiro	Descrição
Kafka Scheduler	**Configurações**	kafka-mesos.properties	Ficheiro de propriedades que contém informação para estabelecer comunicação com o agendador.
	Executáveis	kafka-mesos.sh	*Script* utilizado para interagir com uma execução ativa do agendador.
		kafka-mesos-0.10.1.0-SNAPSHOT.jar	Ficheiro *jar* gerado pela compilação do projeto, usado tanto para iniciar o agendador pelo Marathon, como para a comunicação com o mesmo a partir do *script* anterior.

Depois de compilado, o *jar* é movido para o diretório de artefactos do Marathon e é submetida uma definição de aplicação JSON, que contém a informação para a implementação e execução do agendador no *cluster*. A atual definição de aplicação limita a localização do agendador para a máquina mestre, com o objetivo de poupar recursos nos nós 1 e 2.

A gestão e configuração dos serviços Kafka são realizados a partir do executável 'kafka-mesos.sh'. O *script* permite a execução de comandos de manipulação de *brokers* e tópicos, passando as propriedades de configuração através de um ficheiro ou linha de comandos. Uma das propriedades usadas é a '*constraints*', responsável pela capacidade de limitar a inicialização de um servidor Kafka ao nó 1.

5.1.4 Cassandra

À semelhança do serviço Kafka, existem dois agendadores de Cassandra preparados para serem executados em ambientes Mesos. Ambos originalmente criados pela Mesosphere, apenas o projeto[17] que faz parte integrante da plataforma DC/OS se mantém ativo.

O repositório foi clonado e o código fonte compilado para gerar os *jars* do agendador e executor de Cassandra. Os *jars* resultantes foram movidos também para o diretório de artefactos do Marathon. Apesar de ser parte integrante da plataforma DC/OS, o funcionamento dos serviços do sistema mencionado, baseiam-se na submissão de aplicações ao Marathon.

Este repositório, contém uma definição de aplicação Marathon ajustada para trabalhar com outros serviços DC/OS. Definição que teve de ser alterada para eliminar erros e minimizar avisos de falhas de acesso a serviços DC/OS inexistentes. Ao contrário de implementação de Kafka, a definição da aplicação contém variáveis de ambiente que se traduzem nos parâmetros de configuração do serviço Cassandra. A Tabela 15 descreve alguns dos parâmetros de configuração mais relevantes do serviço Cassandra.

[17] Sítio: https://github.com/mesosphere/dcos-cassandra-service

Tabela 15: Parâmetros de configuração mais relevantes do serviço Cassandra.

	Parâmetro de Configuração	Descrição
Cassandra	CASSANDRA_AUTHENTICATOR CASSANDRA_AUTHORIZER	Controlam a metodologia de autenticação c/ a base de dados.
	NODES SEED_NODES SEEDS_URL	Indicam o número de nós totais, o número de nós sementes e o URL que contém a informação de localização dos nós sementes no *cluster*.
	CONCURRENT_READS CONCURRENT_WRITES CONCURRENT_COUNTER_WRITES	Indicam o número de *threads* dedicadas às operações enumeradas. Variáveis mais utilizadas no ajustamento do desempenho de Cassandra.
	PLACEMENT_CONSTRAINT	Condições de restrição da localização dos serviços Cassandra (p.e."hostname:UNLIKE:microgridmaster").

A Interação dos utilizadores com a base de dados pode ser feita a partir da aplicação gráfica DevCenter[18] da DataStax ou pela linha de comandos 'cqlsh' disponível nas distribuições de Cassandra.

5.1.5 Spark

Ao contrário dos dois serviços anteriores, a distribuição de Apache Spark oficial já contém a implementação de um agendador preparado para funcionar com o Mesos. Assim, apenas foi criada a definição de aplicação Marathon, para executar o agendador no nó mestre. No caso do serviço Spark, a distribuição foi instalada num diretório específico, cujo caminho é idêntico nos nós 2 e 3. Desta forma, a distribuição de Spark não é transferida para os nós em questão sempre que é submetida uma nova tarefa para execução.

Os parâmetros de configuração de Spark são impostos pelo utilizador, durante a submissão de uma tarefa de execução ao agendador e alguns dos mais relevantes podem ser visualizados na

[18] Sítio: https://academy.datastax.com/quick-downloads

Tabela 16: Parâmetros de configuração na submissão de tarefas Spark.

	Parâmetro de Configuração	Descrição
Spark	spark.driver-memory spark.driver.cores spark.executor.memory spark.executor.cores	Indicam a quantidade de memória e número de *cores* de processamento requisitados para *os executores*.
	spark.jars spark.jars.packages	Lista de *jars* (dependências) locais ou coordenadas Maven, a incluir na *classpath* do executor e do *driver*.
	spark.serializer	Classe utilizada para serializar objetos para transmitir pela rede. A definição pré-definida de JavaSerializer compromete desempenho a favor de flexibilidade de serialização. Definir serialização Kryo quando o fator velocidade é mais relevante
	spark.dynamicAllocation.enabled	Determina se deve ser usado um mecanismo de escalonamento de executores baseado na carga de trabalho.
	spark.mesos.constraints	Controla a aceitação de ofertas recebidas pelo Mesos.

Interação com o Spark envolve a utilização da *spark-shell* ou através do executável *spark-submit*. O estado das tarefas pode ser visualizado na aplicação *web* alojada pelo agendador ou na aplicação *web* do Mesos.

5.2 Aquisição dos dados

O processo de aquisição de dados começou pela verificação do esquema da base de dados para a qual já se realizava monitorização. A Figura 14 apresenta um excerto do modelo físico de dados da base de dados do edifício N. A base de dados contém uma tabela para cada analisador de energia, cujas colunas são mapeamentos diretos dos parâmetros monitorizados. Os parâmetros relativos ao consumo encontram-se designados por P1, P2 e P3. O valor de produção de energia é armazenado na tabela 'Inverter6_V3' designado por P. Para além do consumo, encontram-se armazenados também campos relativos à qualidade da rede elétrica.

Durante o desenvolvimento do projeto foram alterados os modelos dos analisadores de energia, resultando no versionamento das tabelas existentes. Assim, existem 3 tabelas para cada um dos analisadores com parâmetros diferentes. A segunda versão de analisadores apenas introduziu três novos campos para além dos já existentes na altura. Com a introdução da versão 3, alguns dos analisadores de energia capturam parâmetros de forma diferente dos restantes,

como é o caso da tabela 'Analyzer4_V3', em que a informação armazenada encontra-se multiplicada por 10. No total, foi utilizada a informação retida em 25 tabelas diferentes.

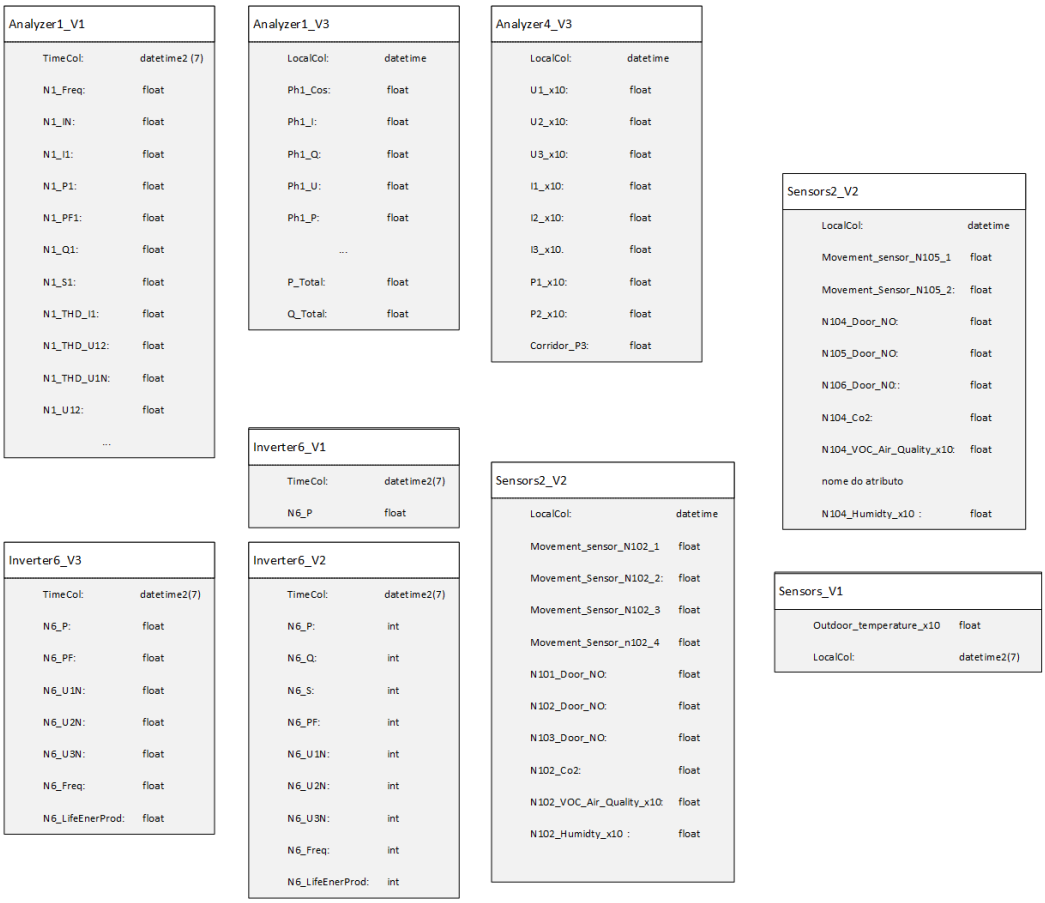

Figura 14: Excerto do modelo físico de dados da base de dados do edifício N.

O processo de desenho da base de dados em Cassandra teve como principal objetivo consolidar as diferentes tabelas dos analisadores de energia. A Figura 15 apresenta o modelo físico de dados em Cassandra.

O mapeamento para Cassandra resultou em:

- A informação das versões 1 e 2 dos analisadores de energia de consumo são armazenadas apenas na tabela 'old_main_analyzers';
- Os dados da versão 3 dos analisadores de energia de consumo são armazenados na tabela 'main_analyzers';
- As três versões de dados dos inversores são guardadas apenas na tabela 'pv_generation';
- A tabela 'sensors' de Cassandra usa o tipo de dados 'Map' para consolidar todas os diferentes esquemas da rede de sensores;
- Todos os registos encontram-se já organizados temporalmente de ordem descendente através do uso da coluna de *clustering* 'datetime';

- Cada registo nas tabelas é identificado por uma chave primária composta, constituída pelos seguintes campos: id, year, month e datetime;

Este modelo de dados em Cassandra reduziu o número de tabelas a consultar, tirando partido dos campos comuns aos analisadores de energia. No entanto, uma vez que a diferença de campos das versões 1 e 2 para a versão 3 foi significativa, decidiu-se separar em duas tabelas diferentes. Analisadores que não partilham campos em comum não são problema, uma vez que, internamente, valores *null* não são armazenados.

A utilização do tipo de dados *double* para todos os parâmetros monitorizados, garante compatibilidade com os diversos tipos de analisadores atuais e futuros.

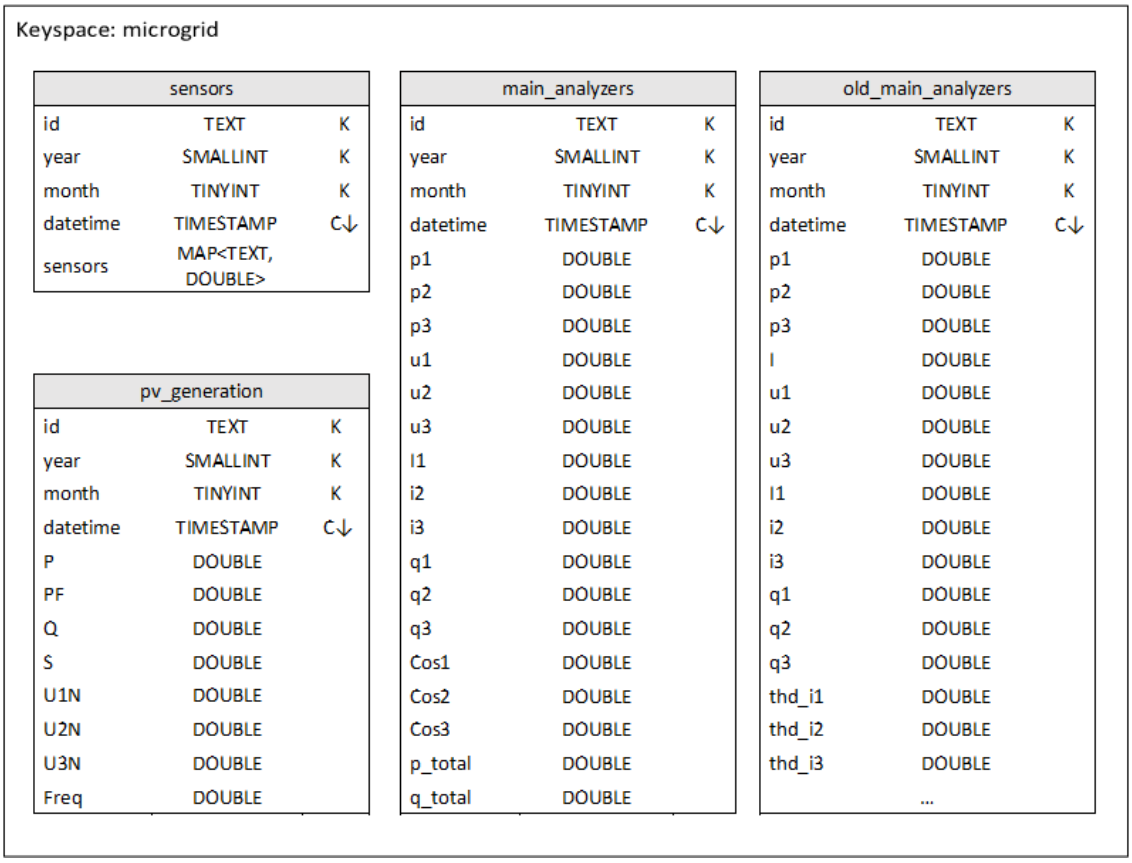

Figura 15: Modelo físico de dados, em notação Chebotko, do edifício N em Cassandra.

Uma vez que a informação se trata de séries temporais, a modelação das tabelas teve em conta o tamanho das partições. Desta forma, todas as tabelas encontram-se particionadas mensalmente através da inclusão das colunas 'year' e 'month' na chave de partição, sendo o mais indicado para o número de campos envolvidos e a cadência dos registos.

O particionamento foi objeto de estudo importante durante a implementação, uma vez que o sistema base de armazenamento de Cassandra sofreu alterações significativas da versão 2.0+ para as versões atuais de 3.0+. Como já mencionado anteriormente na Figura 5, a partição de registos em Cassandra cresce horizontalmente. Assim, a utilização de séries temporais infinitas

implica à utilização de uma metodologia de particionamento. Nestes casos, recorre-se ao cálculo do tamanho da partição no final de intervalos de tempo, contabilizando a cadência do fluxo de dados. Para obter o melhor desempenho na manipulação da partição, o seu tamanho deve rondar ou manter-se abaixo dos 100 Mb.

O cálculo do tamanho da partição para as versões 2.0+ foi inicialmente apresentado no curso de modelação da DataStax DS220 (Datastax Academy, 2016) e é efetuada com a ajuda de duas fórmulas.

A equação 1 determina o número de células N_v de uma partição, tendo em conta o número de linhas esperado. O significado de cada variável é o seguinte: N_c, número de colunas totais; N_{pk} número de colunas da chave primária; N_s número de colunas estáticas.

$$N_v = N_r \times \left(N_c - N_{pk} - N_s\right) + N_s \quad (1)$$

De seguida, é usada a equação 2 para determinar o tamanho da partição total P_s. Esta fórmula utiliza o tamanho do tipo de dados (bytes) de cada coluna. A designação das variáveis é: C_{k_i}, coluna parte da chave-primária; C_{k_j}, coluna estática; C_{r_k}, coluna regular; C_{c_l} coluna de *clustering*.

$$P_s = \sum_i sizeOf\left(C_{k_i}\right) + \sum_j sizeOf\left(C_{k_j}\right) + Nr \times \sum_k \left(sizeOf\left(C_{r_k}\right) + \sum_l sizeOf\left(C_{c_l}\right)\right) + 8 \times N_v\,. \quad (2)$$

Durante o processo de modelação não existia uma formalização para calcular o tamanho da partição na versão 3.0 de Cassandra. Esta versão, contou com inúmeras mudanças ao sistema de armazenamento incluindo a serialização dos dados nas partições. Assim, foi proposta uma revisão do cálculo do tamanho da partição em sessão especial DCAI (*Conference Distributed Computing and Artificial Intelligence*) edição 2017 (Pinheiro *et al.*, 2018).

A equação 3, apresenta a fórmula revista utilizada para calcular o tamanho da partição em Cassandra versão 3.0. A fórmula tem em conta de que já não é efetuada a repetição das colunas de *clustering* para cada coluna regular.

$$P_s = \sum_i sizeOf\left(C_{k_i}\right) + \sum_j sizeOf\left(C_{s_j}\right) + Nr \times \left(\sum_k sizeOf\left(C_{r_k}\right) + \sum_l sizeOf\left(C_{c_l}\right) + 8\right). \quad (3)$$

Para a tabela 'older_main_analyzers' representada na Figura 15, assumindo a cadência de 10 segundos e inexistência de interrupções, estimou-se que o tamanho da mensal da partição tomaria o valor aproximado de 74 Mb, dentro dos parâmetros recomendados. Apesar das restantes tabelas não possuírem tantas colunas, decidiu-se realizar o particionamento da mesma forma, mantendo consistência nos inquéritos realizados à base de dados.

5.2.1 Aquisição em tempo-real

Como mencionado no inicio do capítulo 5, recorreu-se a API do KafkaConnect para implementar o conector de acesso aos PLCs. A API fornece um conjunto de bibliotecas para facilmente interligar sistemas externos ao Kafka. A Figura 16 apresenta o diagrama de implementação de classes do conector desenvolvido. O conector faz uso de duas bibliotecas externas: com.ghgande.j2mod, para estabelecer comunicação e ler parâmetros do PLC; e com.google.code.gson para a construção e serialização de *strings* JSON. O conector utiliza o paradigma de comunicação *polling* na ligação com os PLCs. É ainda possível, configurar o conector para estabelecer ligações com múltiplos PLCs executadas em paralelo.

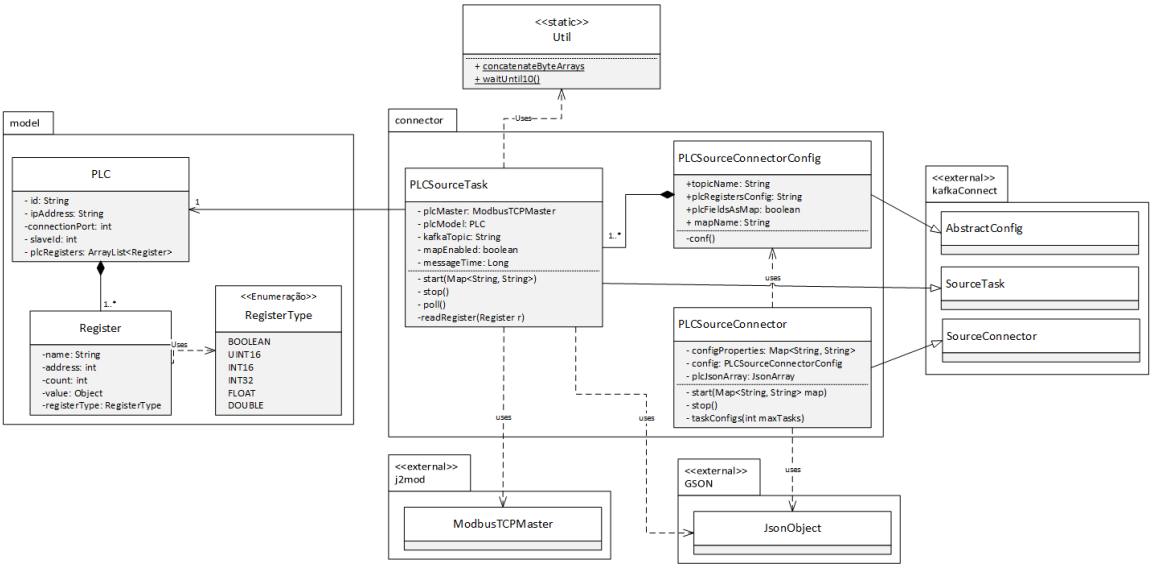

Figura 16: Diagrama de Implementação de Classes do conector 'PLCConnector.jar'.

Cada PLC possui definido um identificador único, que será usado na chave primária de cada registo em Cassandra, a informação de conexão ao equipamento e um conjunto de registos a ler. Cada registo em si é caracterizado por um nome, valor e informação para a leitura do valor. A Tabela 17 descreve as variáveis configuráveis necessárias para a execução do Conector.

Tabela 17: Propriedades de configuração do conector 'PLCConnector.jar'.

Propriedade	Tipo	Obrigatório	Descrição
name	String	Sim	Nome da instancia do conector a utilizar. Deve ser único para cada instância.
tasks.max	Int	Sim	Número de tarefas a utilizar. Deve ser igual ao número de PLCs configurados
topic	String	Sim	O nome do tópico Kafka para onde inserir os registos.
map.enable	boolean	Sim	Se os registos devem ser mapeados para uma estrutura de dados *map* (p.e. o caso da leitura dos sensores).
map.name	String	Não	O nome da estrutura de *map* caso esta seja necessária.
registers	String	Sim	*String* JSON com a configuração de cada PLC e os parâmetros a serem monitorizados. Os nomes dos registos devem corresponder aos nomes das colunas em Cassandra.

Com o objetivo de não sobrecarregar os analisadores e de manter a mesma cadência de dados, o período de tempo entre recolhas não é parametrizável. A Figura 17 apresenta a configuração utilizada para realizar a monitorização do Inversor da produção de energia. É importante realçar que tanto o identificador do PLC, bem como os nomes dos registos definidos na configuração, são mapeados diretamente com o nome das colunas em Cassandra.

```
name=plc-n-generation
connector.class=gecad.connector.PLCSourceConnector
tasks.max=1
topic=n_production
map.enable=false;
registers=[{"plc.id":"N_Inverter6_V3","plc.port":502,"plc.slave":8,"plc.ip":"192.16
8.2.225","plcRegisters":[{"name":"P","address":13,"count":2,"type":"INT16"},{"name"
:"PF","address":17,"count":2,"type":"INT16"},{"name":"U1N","address":10,"count":2,"
type":"INT16"},{"name":"U2N","address":11,"count":2,"type":"INT16"},{"name":"U3N","
address":12,"count":2,"type":"INT16"},{"name":"Freq","address":14,"count":2,"type":
"INT16"},{"name":"LifeEnergyProd","address":18,"count":2,"type":"UINT16"}]}]
```

Figura 17: Exemplo de configuração do conector 'PLCConnector.jar'.

No caso de recolha de dados de consumo, é executada uma instância do conector configurada para todos os 5 analisadores de energia. Outras duas instâncias do conector são executadas para a monitorização dos dados de produção de energia e da informação da rede de sensores. A inicialização e posterior execução das 3 instâncias, é realizada pelo serviço Marathon, restringindo a localização de execução para a máquina mestre.

A movimentação dos dados em Kafka para Cassandra, é feita pelo conector 'CassandraSink.jar' já mencionado anteriormente. A configuração desse conector, permite, em apenas uma instância, redirecionar o conteúdo dos múltiplos tópicos de Kafka, para diferentes tabelas de Cassandra.

5.2.2 Aquisição *Batch*

A migração dos dados existentes no servidor SQL foi conseguida através do *software* Talend Open Studio for Big Data. Esta ferramenta é das poucas ofertas completas de ETL (*Extract Load Transform*), que permite a integração de dados de sistemas tradicionais com tecnologias de Big Data. O seu modo de funcionamento envolve a criação de código java para executar as tarefas, designadas como *jobs*, para todos os componentes de integração.

A escolha deste tipo de *software*, recaiu na necessidade de definir o particionamento mensal nos dados já existentes e realizar o mapeamento que resulta na redução do número de tabelas necessárias em Cassandra.

De forma a automatizar o processo de aquisição de dados foram criados quatro *jobs* principais, um para cada tabela existente em Cassandra, parametrizáveis por datas de inicio e de fim para cada tabela em SQL. Ao todo, o projeto realiza a migração das 25 tabelas mencionados no inicio do capítulo 5. A Figura 18 apresenta um *subjob* que tem como objetivo migrar os dados tabela 'Analyzer1_V2' do servidor SQL.

O processo de migração de uma tabela começa por realizar um inquérito à tabela definida, que obtém os anos e meses de dados existentes entre o intervalo de datas configurado. Para cada resultado anterior, é executado um segundo inquérito que recolhe o conjunto do mês em particular. Este passo é necessário para controlar o uso de memória, garantindo a estabilidade do ambiente de execução em casos em que as tabelas possuam milhões de registos. Em seguida, é realizado o mapeamento das colunas da tabela SQL com a tabela de destino em Cassandra, definindo os valores necessários para a o particionamento mensal. No final de cada iteração é gerada uma SSTable (bloco de dados em Cassandra) posteriormente importada na conclusão de todas as iterações.

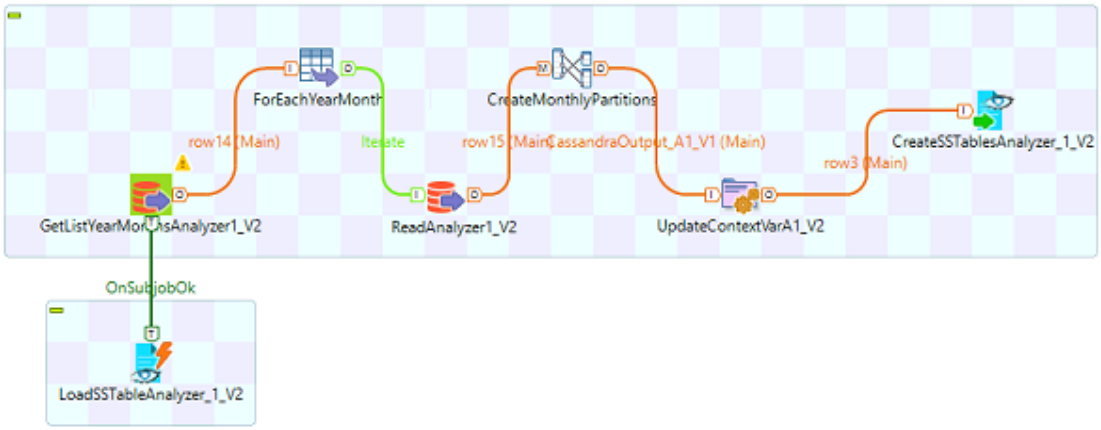

Figura 18: Tarefa *batch* de importação dos registos da tabela 'Analyzer1_V2' do servidor SQL.

A implementação a partir do uso desta ferramenta, possibilitou a criação de um mecanismo de recolha de dados reutilizável e flexível, que pode ser também utilizado para movimentar os mesmos dados para outros sistemas de destino.

5.3 Resultado dos Testes de Desempenho

Os diferentes testes de desempenho foram realizados no mesmo dia, fora do período laboral, e de forma sequencial. A ferramenta de *benchmark* ycsb foi configurada no nó 1, assegurando que a localização do serviço Cassandra não era local à máquina de testes e uma instalação pré-definida do MsSQLServer foi configurada no nó 2. A avaliação dos resultados obtidos foi realizada com recurso ao teste estatístico Wilcoxon Signed Rank Test.

Os resultados obtidos no teste de desempenho AD1.1 encontram-se demonstrados na Tabela 18. Ao observar o número de operações de escrita por segundo, verificou-se que existia uma grande discrepância entre os dois sistemas. A base de dados SQL Server apresenta uma média de taxa de inserção bastante baixa, relativamente a apenas um nó de Cassandra.

Tabela 18: Resultados do teste de avaliação de desempenho AD1.1.

Load 500 000 records										
Teste	**SQL**	**Cassandra (1)**	**Diff**	**Abs Diff**	**Positive**	**Rank**	**Signed Rank**			
1	371	4269	-3898	3898	-1	6	-6		**Positive Sum**	0
2	342	4230	-3888	3888	-1	4	-4		**Negative Sum**	-49
3	333	4314	-3981	3981	-1	9	-9		**T**	0
4	335	4205	-3870	3870	-1	3	-3		**Tails**	2
5	332	4285	-3953	3953	-1	8	-8		**α**	0.05
6	369	4265	-3896	3896	-1	5	-5		**N**	9
7	343	4269	-3926	3926	-1	7	-7		**T-critic**	5
8	370	4353	-3983	3983	-1	10	-10			
9	377	4213	-3836	3836	-1	2	-2			
10	381	4185	-3804	3804	-1	1	-1			

Já o teste AD1.2, pretendia verificar se dois nós de Cassandra melhorariam a capacidade de inserção comparativamente a apenas uma instância. Na Tabela 19, é possível observar que o desempenho de Cassandra se degradou ligeiramente ao expandir-se o número de nós dedicados. Embora a diferença seja pequena, em nenhum dos 10 testes, a arquitetura maior se desempenhou melhor que a sua contraparte. Este resultado, vai contra a teoria de escalabilidade linear normalmente associada à base de dados. É possível que o facto da introdução de replicação dos dados e do aumento do nível de consistência, tenham contribuído o suficiente para justificar tais valores.

Tabela 19: Resultados do teste de avaliação de desempenho AD1.2.

Load 500 000 records, RF=2, CL=TWO										
Teste	**Cassandra (1)**	**Cassandra (2)**	**Diff**	**Abs Diff**	**Positive**	**Rank**	**Signed Rank**			
1	4269	3876	393	393	1	5	5		**Positive Sum**	55
2	4230	3925	305	305	1	1	1		**Negative Sum**	0
3	4314	3865	449	449	1	8	8		**T**	0
4	4205	3794	411	411	1	7	7		**tails**	2
5	4285	3784	501	501	1	10	10		**α**	0.05
6	4265	3885	380	380	1	4	4		**N**	9
7	4269	3874	395	395	1	6	6		**T-critic**	5
8	4353	3874	479	479	1	9	9			
9	4213	3857	356	356	1	2	2			
10	4185	3809	376	376	1	3	3			

Desta forma, a opção de usar uma arquitetura de dois nós na solução do projeto apenas é justificável pela replicação de informação.

Os resultados da execução da *WorkloadA,* apresentados na

Tabela 20, confirmam novamente, que a base de dados SQL Server desempenhou pior do que Cassandra com dois nós e mantém-se consistente com os resultados obtidos anteriormente. Curiosamente, a introdução de operações de leitura, teve um efeito negativo nos resultados da base de dados Cassandra.

Tabela 20: Resultados do teste de desempenho AD2.1.

Workload A 250 000										
Teste	**SQL**	**Cassandra (2)**	**Diff**	**Abs Diff**	**Positive**	**Rank**	**Signed Rank**			
1	417	3126	-2709	2709	-1	1	-1		**Positive Sum**	0
2	453	3272	-2819	2819	-1	5	-5		**Negative Sum**	-53
3	409	3222	-2813	2813	-1	2	-2		**T**	0
4	453	3377	-2924	2924	-1	10	-10		**Tails**	2
5	421	3239	-2818	2818	-1	4	-4		**α**	0.05
6	446	3322	-2876	2876	-1	9	-9		**N**	9
7	436	3268	-2832	2832	-1	6	-6		**T-critic**	5
8	455	3296	-2841	2841	-1	7	-7			
9	448	3298	-2850	2850	-1	8	-8			
10	434	3247	-2813	2813	-1	2	-2			

Finalmente, na execução da carga de trabalho maioritariamente de leituras, ambas as bases de dados conseguiram atingiram valores superiores de operações por segundo. Este aumento era esperado, uma vez que os sistemas de armazenamento possuem parte dos dados já em *cache*. Apesar da subida substancial no desempenho de SQL Server, esta ainda fica aquém da conseguida pelo sistema de armazenamento Cassandra.

Tabela 21: Resultado do teste de desempenho AD2.1.

Workload D 250 000 (95% Reads 5%Writes)										
Teste	**SQL**	**Cassandra (2)**	**Diff**	**Abs Diff**	**Positive**	**Rank**	**Signed Rank**			
1	2558	6261	-3703	3703	-1	9	-9		**Positive Sum**	0
2	2544	6220	-3676	3676	-1	6	-6		**Negative Sum**	-46
3	2552	6218	-3666	3666	-1	4	-4		**T**	0
4	2551	6231	-3680	3680	-1	8	-8		**Tails**	2
5	2533	6211	-3678	3678	-1	7	-7		**α**	0.05
6	2532	6071	-3539	3539	-1	2	-2		**N**	9
7	2540	5725	-3185	3185	-1	1	-1		**T-critic**	5
8	2560	6116	-3556	3556	-1	3	-3			
9	2538	6211	-3673	3673	-1	5	-5			
10	2508	6279	-3771	3771	-1	10	-10			

De uma forma geral, os resultados podem ser considerados incertos e inconclusivos, apesar de se ter usado o mesmo *hardware*. Os níveis de desempenho apresentados pela instalação SQL Server são muito discrepantes em comparação com a solução NoSQL, indicando potenciais problemas difíceis de diagnosticar. Por um lado, é necessário ter em conta que a ferramenta *YCSB* é orientada para sistemas NoSQL. No entanto, é possível que existam problemas na configuração da base de dados SQL Server ou na utilização do driver JDBC usado nos testes.

6 Conclusão e Trabalho Futuro

O setor elétrico encontra-se no meio de um processo de transformação. A procura pela eficiência na geração, consumo e distribuição de energia, forçam a industria a modernizar a sua infraestrutura, idealizar novos conceitos e adotar novas tecnologias. A implementação generalizada de redes elétricas inteligentes, é cada vez mais reconhecida como uma das principais soluções para o futuro. Neste tipo de soluções, energia e informação são trocadas entre os distribuidores e os consumidores, permitindo a construção de processos de automação ajustados às necessidades de ambas as entidades. O seu sucesso depende em parte na capacidade de lidar com o volume adicional de informação. Novas estratégias de gestão de dados conduzem à exploração de tecnologias na área de Big Data.

Neste contexto, o grupo de investigação de GECAD detém de uma micro rede inteligente ativamente monitorizada. O objetivo principal deste projeto foi o desenho e implementação de uma solução Big Data distribuída e integrada com a micro rede. A solução teria de consolidar o armazenamento e permitir a aplicação de técnicas de análise periódicas e de tempo-real sobre os dados.

Baseada na SMACK *stack*, a solução proposta foi implementada num *cluster* de três máquinas, 1 nó físico e 2 nós virtualizados, gerido pela ferramenta Apache Mesos. Foi desenvolvido um conector em Apache Kafka, que monitoriza ativamente um dos edifícios da micro rede, e persiste os dados capturados na base dados NoSQL Apache Cassandra. Finalmente os requisitos de análises periódicas e de tempo-real são cumpridos em Apache Spark.

Um dos maiores desafios encontrados no processo de desenvolvimento, foi o da integração das diferentes ferramentas com o gestor de recursos Apache Mesos. A escolha do Mesos implicou a utilização de distribuições paralelas aos canais oficiais de cada uma das restantes ferramentas. A modelação de dados em Cassandra, foi um dos pontos fulcrais da solução. Resultante deste

estudo, foi escrito e publicado um *paper*, que formaliza o particionamento de dados temporais na versão de Cassandra 3.0.

Na tentativa de validar a solução implementada, realizaram-se testes de desempenho que compararam a base de dados de Cassandra com SQL Server através do uso da ferramenta YCSB. Em todos os testes comparativos destes dois sistemas, Cassandra revelou-se ser a melhor solução. No entanto, surgem interrogações à validade das experiências. O desempenho de SQL Server obtido, foi bastante baixo e impossível de diagnosticar. A falta de homogeneidade no *hardware* utilizado no ambiente de testes, pode ter contribuído para os resultados obtidos.

Apesar de se ter implementado autenticação em alguns pontos da solução, a implementação atual ainda não conta com canais de comunicação encriptados nem com a encriptação de dados em repouso. A utilização de nós físicos para todos as máquinas do *cluster*, bem como a sua expansão, poderão trazer melhorias a nível de desempenho. Com base na arquitetura implementada é possível evoluir para casos de estudo complexos e que integrem dados de diversas fontes, como, para além da micro rede GECAD, dados de edifícios de parceiros internacionais que colaboram com o GECAD em projetos europeus.

Referências

Abiteboul, S., Buneman, P. and Suciu, D. (1999) *Data on the Web: from relations to semistructured data and XML*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Aiello, M. and Pagani, G. A. (2014) 'The Smart Grid's Data Generating Potentials', in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*. IEEE, pp. 9–16. doi: 10.15439/2014F509.

Asay, M. (2015) *The Future Of Big Data Looks Like Streaming*. Available at: http://readwrite.com/2015/04/02/big-data-streaming-no-more-batch-processing (Accessed: 10 February 2016).

Aslett, M. (2011) *How will the database incumbents respond to NoSQL and NewSQL*. Available at: https://451research.com/report-short?entityId=66963.

Astakhov, V. and Chayel, M. (2015) 'Lambda Architecture for Batch and RealTime Processing on AWS with Spark Streaming and Spark SQL'. Amazon Web Services, p. 12. Available at: https://d0.awsstatic.com/whitepapers/lambda-architecure-on-for-batch-aws.pdf.

Barnes, C., Blake, H. and Pinder, D. (2009) *Creating and delivering your value proposition: Managing customer experience for profit*. 1st edn. London and Philadelphia: Kogan Page.

Belliveau, P. *et al.* (2002) *The PDMA toolbook for new product development*. John Wiley & Sons.

Brewer, E. (2000) *Towards robust distributed systems*, *PODC*. Available at: http://www.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf (Accessed: 17 February 2016).

Brewer, E. (2012) 'CAP twelve years later: How the "rules" have changed', *Computer*. IEEE, 45(2), pp. 23–29. doi: 10.1109/MC.2012.37.

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

Cheng, B. *et al.* (2015) 'Building a Big Data Platform for Smart Cities: Experience and Lessons from Santander', in *2015 IEEE International Congress on Big Data*. IEEE, pp. 592–599. doi: 10.1109/BigDataCongress.2015.91.

Cooper, B. F. *et al.* (2010) 'Benchmarking cloud serving systems with YCSB', in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*. New York, New York, USA: ACM Press, p. 143. doi: 10.1145/1807128.1807152.

Costa, C. and Santos, M. Y. (2016) 'Reinventing the Energy Bill in Smart Cities with NoSQL Technologies', in *Transactions on Engineering Technologies*. Springer, pp. 383–396.

Cox, M. and Ellsworth, D. (1997) 'Application-controlled demand paging for out-of-core visualization', in *Proceedings of the 8th conference on Visualization'*. Los Alamitos: IEEE Computer Society Press, p. 235–ff. Available at: http://dl.acm.org/citation.cfm?id=267068 (Accessed: 16 February 2016).

Datastax Academy (2016) *Physical: Partition Size*. Available at: https://academy.datastax.com/courses/ds220-data-modeling/physical-partition-size (Accessed: 10 July 2016).

Dean, J. and Ghemawat, S. (2004) 'MapReduce: Simplified Data Processing on Large Clusters', in *Proc. of the OSDI - Symp. on Operating Systems Design and Implementation*, pp. 137–149. doi: 10.1145/1327452.1327492.

Dezyre (2015) *How Big Data Analysis helped increase Walmart's Sales turnover?* Available at: https://www.dezyre.com/article/how-big-data-analysis-helped-increase-walmart-s-sales-turnover/109 (Accessed: 27 December 2015).

Edelson, H. (2015) *Lambda Architecture with Spark Streaming, Kafka, Cassandra, Akka, Scala*. Available at: http://www.slideshare.net/helenaedelson/lambda-architecture-with-spark-streaming-kafka-cassandra-akka-scala (Accessed: 15 February 2016).

Efthymiou, C. and Kalogridis, G. (2010) 'Smart Grid Privacy via Anonymization of Smart Metering Data', in *2010 First IEEE International Conference on Smart Grid Communications*. IEEE, pp. 238–243. doi: 10.1109/SMARTGRID.2010.5622050.

Estrada, R. and Ruiz, I. (2016) *Big Data SMACK*. 1st edn. Apress. doi: 10.1007/978-1-4842-2175-4.

Farhangi, H. (2010) 'The path of the smart grid', *IEEE Power and Energy Magazine*. IEEE, 8(1), pp. 18–28. doi: 10.1109/MPE.2009.934876.

Ganesh H.B., B., Kumar S., S. and Poornachandran, P. (2015) 'Apache Spark a Big Data Analytics Platform for Smart Grid', *Procedia Technology*, 21, pp. 171–178. doi: 10.1016/j.protcy.2015.10.085.

Gray, J. (1981) 'The transaction concept: virtues and limitations', in *Proceedings of the 7th International Conference on Very Large Databases*. Cupertino CA USA: VLDB Endowment, pp. 144–154. Available at: http://dl.acm.org/citation.cfm?id=1286831.1286846 (Accessed: 17 February 2016).

IEA (2011) 'Technology Roadmap: Smart Grids'. Paris: International Energy Agency, p. 52. Available at: http://www.iea.org/publications/freepublications/publication/technology-roadmap-smart-grids.html.

ISGAN (2014) *Smart Grid Drivers and Technologies by Country, Economies and Continents*. Available at: http://www.iea-isgan.org/force_down_2.php?num=3.

Kaplan, R. and Norton, D. P. (2004) *Strategy maps: Converting intangible assets into tangible outcomes*. 1st edn. Boston: Harvard Business School Press.

Khurana, H., Hadley, M. and Frincke, D. A. (2010) 'Smart-grid security issues', *IEEE Security & Privacy Magazine*, 8(1), pp. 81–85. doi: 10.1109/MSP.2010.49.

Kim, J. and Wilemon, D. (2002) 'Focusing the fuzzy front-end in new product development', *R and D Management*. Blackwell Publishers Ltd, 32(4), pp. 269–279. doi: 10.1111/1467-9310.00259.

Koen, P. *et al.* (2001) 'Providing clarity and a common language to the "fuzzy front end"', *Research-Technology Management*. Taylor & Francis, 44(2), pp. 46–55.

Kreps, J. (2014) *Questioning the Lambda Architecture*, *O'Reilly*. Available at: https://www.oreilly.com/ideas/questioning-the-lambda-architecture (Accessed: 15 February 2016).

Lai, C. S. and Lai, L. L. (2015) 'Application of Big Data in Smart Grid', in *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, pp. 665–670. doi: 10.1109/SMC.2015.126.

Laney, D. (2001) '3D data management: Controlling data volume, velocity and variety', *META Group Research Note*. Available at: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-

Management-Controlling-Data-Volume-Velocity-and-Variety.pdf (Accessed: 16 February 2016).

Laney, D. (2013) *Batman on Big Data*. Available at: http://blogs.gartner.com/doug-laney/batman-on-big-data/ (Accessed: 20 November 2015).

Macedo, S. and Caxias, T. (2015) 'QTDEI - Big Data - Conceitos e Princípios Fundamentais'. Porto: ISEP.

Malewicz, G., Austern, M. and Bik, A. (2010) 'Pregel: a system for large-scale graph processing', in *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*. New York: ACM Press, pp. 135–146. doi: 10.1145/1807167.1807184.

Marz, N. (2011) *How to Beat The CAP Theorem*. Available at: http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html.

Marz, N. and Warren, J. (2015) *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co. Available at: http://dl.acm.org/citation.cfm?id=2717065 (Accessed: 21 February 2016).

Mayilvaganan, M. and Sabitha, M. (2013) 'A cloud-based architecture for Big-Data analytics in smart grid: A proposal', in *2013 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, pp. 1–4. doi: 10.1109/ICCIC.2013.6724168.

McFadin, P. (2015) *Getting Started with Time Series Data Modeling*. Available at: https://academy.datastax.com/demos/getting-started-time-series-data-modeling (Accessed: 10 February 2016).

MeriTalk and EMC (2014) *The Big Data Cure*. Available at: https://www.meritalk.com/study/the-big-data-cure/.

Merlino, G. and Yang, F. (2014) *Building a Data Pipeline That Handles Billions of Events in Real-Time*. Available at: https://metamarkets.com/2014/building-a-data-pipeline-that-handles-billions-of-events-in-real-time/ (Accessed: 10 February 2016).

Nicola, S. and Ferreira, E. (2012) 'A novel framework for modeling value for the customer, an essay on negotiation', *International Journal of Information Technology & Decision Making*, 11(3), pp. 661–703. doi: 10.1142/S0219622012500162.

Oracle (2013) 'Oracle: Big Data for the Enterprise', p. 16. Available at: http://www.oracle.com/us/products/database/big-data-for-enterprise-519135.pdf.

Oracle (2015) *An Enterprise Architect's Guide to Big Data*. Available at: http://www.oracle.com/technetwork/topics/entarch/articles/oea-big-data-guide-1522052.pdf (Accessed: 18 November 2015).

Pinheiro, G. *et al.* (2018) 'Smart Grids Data Management: A Case for Cassandra', in *Distributed Computing and Artificial Intelligence, 14th International Conference*, p. 87.

Pratt, R. G. *et al.* (2010) *The smart grid: An estimation of the energy and CO2 benefits*. Richland: U.S. Department of Energy.

Ribeiro, D. A. C. and Bernardino, J. F. R. (2013) *Bases de dados NewSQL: uma avaliação experimental*. Instituo Superior de Engenharia de Coimbra.

SAS (2014) *What is Big Data*. Available at: http://www.sas.com/en_us/insights/big-data/what-is-big-data.html (Accessed: 10 November 2015).

Souza, V. C. O. and Santos, M. V. C. (2015) 'Maturing, Consolidation and Performance of NoSQL Databases: Comparative Study', *Proceedings of the annual conference on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1*. Brazilian Computer Society, p. 32.

Swaminathan, S. N. and Elmasri, R. (2016) 'Quantitative Analysis of Scalable NoSQL Databases', in *2016 IEEE International Congress on Big Data (BigData Congress)*. IEEE, pp. 323–326. doi: 10.1109/BigDataCongress.2016.49.

The Apache Software Foundation (2016) *Apache 2.7.2 - HDFS Architecture*. Available at: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (Accessed: 30 January 2016).

Turner, V. *et al.* (2014) 'The Digital Universe of Opportunities: Rich Data and Increasing Value of the Internet of Things', *IDC White Paper*, (April). Available at: http://www.emc.com/leadership/digital-universe/2014iview/it-imperatives.htm%5Cnhttp://idcdocserv.com/1678%5Cnhttp://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf.

Wang, D. and Xiao, L. (2012) 'Storage and Query of Condition Monitoring Data in Smart Grid Based on Hadoop', in *2012 Fourth International Conference on Computational and Information Sciences*. IEEE, pp. 377–380. doi: 10.1109/ICCIS.2012.292.

Woodall, T. (2003) 'Conceptualising "Value for the Customer" Conceptualising "Value for the Customer": An Attributional, Structural and Dispositional Analysis', *Academy of Marketing Science Review*, 2003(12).

Xin, R. S. *et al.* (2013) 'Shark: SQL and rich analytics at scale', in *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*. New York: ACM Press, pp. 13–24. doi: 10.1145/2463676.2465288.

Yang, Z. and Peterson, R. T. (2004) 'Customer perceived value, satisfaction, and loyalty: The role of switching costs', *Psychology and Marketing*, 21(10), pp. 799–822. doi: 10.1002/mar.20030.

Zaharia, M. *et al.* (2012) 'Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing', in *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. Berkeley: USENIX Association, p. 2.

Zeithaml, V. (1988) 'Consumer perceptions of price, quality, and value: a means-end model and synthesis of evidence', *The Journal of marketing*.

Zhou, D. *et al.* (2016) 'Distributed Data Analytics Platform for Wide-Area Synchrophasor Measurement Systems', *IEEE Transactions on Smart Grid*, 7(5), pp. 2397–2405. doi: 10.1109/TSG.2016.2528895.

Zikopoulos, P. *et al.* (2014) *Big Data Beyond the Hype A Guide to Conversations for Today's Data Center*. 1st edn. Edited by R. Melnyk. McGraw-Hill Education.