UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO AUGUSTO CORDEIRO

SERVIDOR E APLICATIVO WEB PARA ARMAZENAMENTO E ANÁLISE DE DADOS DE LOCALIZAÇÃO DE DISPOSITIVOS MÓVEIS

BAURU, SP

2017

MARCELO AUGUSTO CORDEIRO

SERVIDOR E APLICATIVO WEB PARA ARMAZENAMENTO E ANÁLISE DE DADOS DE LOCALIZAÇÃO DE DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso de graduação apresentado à disciplina Projeto e Implementação de Sistemas do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho”, como requisito parcial para obtenção do título de bacharel.

Orientador: Prof. Dr. Eduardo Martins Morgado

BAURU, SP

2017

Cordeiro, Marcelo Augusto.
Servidor e aplicativo web para armazenamento e análise de dados de localização de dispositivos móveis / Marcelo Augusto Cordeiro, 2017
50 p. : il.

Orientador: Eduardo Martins Morgado

Monografia (Graduação)–Universidade Estadual Paulista. Faculdade de Ciências, Bauru, 2017

1. Big Data. 2. Aplicação Web. 3. Spark. 4. MySQL 5. NodeJS I. Universidade Estadual Paulista. Faculdade de Ciências. II. Título.

MARCELO AUGUSTO CORDEIRO

SERVIDOR E APLICATIVO WEB PARA ARMAZENAMENTO E ANÁLISE DE DADOS DE LOCALIZAÇÃO DE DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso de graduação apresentado à disciplina Projeto e Implementação de Sistemas do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho”, como requisito parcial para obtenção do título de bacharel.

Aprovado em 13 / 02 / 2017

BANCA EXAMINADORA

Prof. Dr. Eduardo Martins Morgado
Orientador
Faculdade de Ciências
Universidade Estadual Paulista - Bauru

Profa. Dra. Simone das Graças Domingues Prado
Faculdade de Ciências
Universidade Estadual Paulista - Bauru

Profa. Dra. Márcia Aparecida Zanoli Meira e Silva
Faculdade de Ciências
Universidade Estadual Paulista - Bauru

Dedico este trabalho em memória de minha avó materna, Izaira de Sousa Silva.

AGRADECIMENTOS

Agradeço aos meus pais Elizabeth Ramos da Silva Cordeiro e Valdemir Manoel Cordeiro e irmã Francyelly Giovany Cordeiro, os quais sem o suporte e apoio incondicional nunca conseguiria chegar até aqui.

Agradeço também aos meus amigos e colegas da turma de 2012 do Bacharelado em Ciência da Computação e do LTIA – Laboratório de Tecnologia da Informação Aplicada, da Unesp de Bauru, sem os quais não conseguiria ter sobrevivido os longos anos de faculdade.

Aos meus amigos do Colégio Técnico Industrial de Bauru, cuja amizades tenho o prazer de desfrutar até hoje.

Ao meu orientador, mentor e amigo, professor Eduardo Morgado, um dos melhores educadores que tive o prazer de conhecer durante a graduação, e sem o qual este trabalho não seria possível.

Por fim, agradeço a todos colegas, amigos e familiares não citados aqui diretamente, mas que contribuíram e me apoiaram durante essa jornada.

Muito obrigado.

“Tenha medo, mas faça assim mesmo.
O importante é a ação. Não fique esperando
para ser confiante. Simplesmente faça o que tem
que fazer e a confiança eventualmente surgirá.”
Carrie Fisher

RESUMO

A internet das coisas está cada vez mais presente no nosso cotidiano, com uma quantidade cada vez maior de dispositivos conectados à internet. Para lidar eficientemente com a grande quantidade de dados gerada por todos esses dispositivos, se faz necessário o uso de ferramentas de big data. Adicionalmente, dado que o sinal tradicional de GPS não é adequado para obter a localização de dispositivos em ambientes fechados, este projeto teve como objetivo desenvolver um servidor e um aplicativo Web para armazenamento e visualização de dados de localização de dispositivos gerados a partir de uma série de sensores de Wi-Fi ou *bluetooth* fixados em pontos pré-determinados dentro de um prédio. Foi desenvolvido um *cluster* utilizando servidores Elastic Compute Cloud da Amazon Web Services, com Spark instalado e configurado para funcionar juntamente com MySQL, e o protótipo de um aplicativo Web, feito em NodeJS, para se conectar aos servidores e exibir os dados armazenados na base de dados.

PALAVRAS-CHAVE: Big Data, Aplicação Web, Spark, MySQL, NodeJS.

ABSTRACT

The internet of things is increasingly present in our daily lives, with an increasing number of devices connected to the internet. To efficiently handle the large amount of data generated by all these devices, it is necessary to use big data tools. In addition, since the traditional GPS signal is not suitable for locating devices indoors, this project aimed to develop a server and Web application for storing and viewing location data of devices generated from a series of Wi-Fi or bluetooth sensors fixed at predetermined points inside a building. A cluster was developed using Elastic Compute Cloud servers from Amazon Web Services, with Spark installed and configured to work alongside MySQL, and the prototype of a Web application, made in NodeJS, to connect to the servers and display the data stored in the database.

KEY-WORDS: Big Data, Web Applications, Spark, MySQL, NodeJS.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

As técnicas de localização sem fio podem ser divididas em dois grupos principais: técnicas de auto posicionamento e técnicas de posicionamento remoto.

Na primeira técnica, o dispositivo móvel utiliza sinais transmitidos por antenas para calcular a sua própria posição. Na segunda técnica, a posição do dispositivo móvel é calculada por uma série de receptores que medem os sinais recebidos e enviados pelo dispositivo (ZEIMPEKIS; GIAGLIS; LEKAKOS, 2003).

Os tradicionais sistemas de GPS (*Global Positioning System*) utilizam a técnica de auto posicionamento para calcular sua posição no globo terrestre baseado nos sinais recebidos de 24 satélites posicionados na órbita terrestre com 20.200 quilômetros de distância entre cada um (DJUKNIC; RICHTON, 2001).

Entretanto, a força do sinal GPS não é suficiente para penetrar na maioria dos prédios. A reflexão do sinal muitas vezes permite a leitura em ambientes fechados, porém o cálculo da posição não será confiável (CHEN; KOTZ, 2000). Portanto, são necessárias soluções diferentes para se criar um sistema de geoposicionamento que funcione em ambientes fechados.

Uma das melhores maneiras de se abordar esse problema é através da Internet das Coisas (*Internet of Things* – IoT).

O termo IoT foi utilizado pela primeira vez por Kevin Ashton em 1999 (ASHTON, 2009), e pode ser definido como “A habilidade de comunicação, conectividade, e computação de dispositivos compartilhando dados via internet para ajudar a melhorar produtos, serviços, capacidade de resposta e qualidade de vida.” (SMITH, 2015, tradução nossa)[1].

Por exemplo, utilizando uma série de sensores Wi-Fi posicionados em pontos fixos dentro de um prédio, com a triangulação do sinal é possível calcular a posição de dispositivos conectados à rede Wi-Fi (BLECKY, 2016).

Para oferecer uma posição confiável, é necessário que estes sensores coletem e transmitam a força do sinal Wi-Fi em cada dispositivo com uma alta frequência.

Utilizando como exemplo o prédio do Laboratório de Tecnologia da Informação Aplicada (LTIA) da Faculdade de Ciências da Unesp de Bauru, em um dia

[1] Texto original: “The communication, connectivity, and computing ability of devices sharing data via the Internet to help improve products, services, responsiveness, and quality of life.”

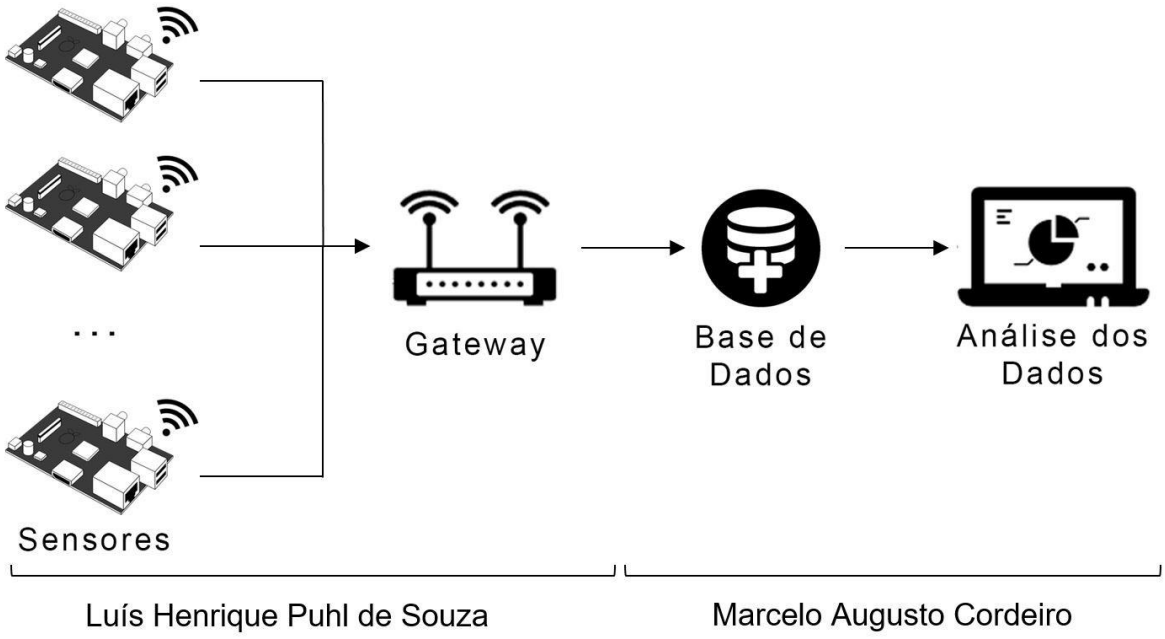
comum, é observado uma média de 30 dispositivos conectados à rede Wi-Fi. Considerando um sensor que a cada 30 segundos colete 1 kB de dados de cada dispositivo, por mês, seriam coletados mais de 2 GB de dados. Portanto, para garantir um sistema escalável, é necessário a utilização de técnicas de *big data* para armazenar e manipular esses dados.

O melhor modo de se definir *big data* ainda é discutido por pesquisadores, mas uma definição simples é a de que “se é necessário se preocupar com o tamanho dos dados, então é *big data*.” (ESPOSITO, 2015, tradução nossa)[2].

Adicionalmente, a constante monitoração da posição de dispositivos dentro de um prédio pode gerar informações valiosas que atualmente não são exploradas.

A Figura 1 ilustra a arquitetura simplificada de um projeto de IoT, mostrando a relação deste projeto com o do aluno Luís Henrique Puhl de Souza, também do Bacharelado em Ciências da Computação da Unesp de Bauru, responsável por desenvolver parte do projeto como seu Trabalho de Conclusão de Curso.

Figura 1: Divisão do Sistema



Fonte: elaborado pelo autor

Em um sistema de IoT, o *gateway* é o ponto de acesso único que conecta todos os sensores à internet. Deste modo, a base de dados não precisa se conectar

[2] Texto original: “If I have to worry about the size of the data, then the data is big.”

com cada um dos sensores, todos os dados coletados já ficam agrupados e filtrados no *gateway* (KONSEK, 2015).

A estrutura do sistema aqui proposto pode ser facilmente escalada para diversas finalidades, desde o controle de funcionários dentro de uma empresa até um sistema completo de GPS para ambientes subterrâneos.

Por exemplo, considerando um ambiente onde a maior parte das pessoas está sempre carregando consigo pelo menos um dispositivo conectado à internet, monitorar a posição destes dispositivos significa também monitorar a posição das pessoas naquele ambiente.

Essa informação pode ser utilizada para as mais diversas finalidades, como para determinar as áreas do prédio onde se tem a maior concentração de pessoas durante um determinado período de tempo. Em ambientes comerciais, por exemplo, essa informação pode ser utilizada para melhor distribuição dos funcionários nas áreas com maior número de clientes.

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver um servidor e um aplicativo Web para armazenamento e visualização de dados de localização de dispositivos gerados a partir de uma série de sensores de Wi-Fi ou *bluetooth* fixados em pontos pré-determinados dentro de um prédio.

1.1.2 Objetivos específicos

- Estudar ferramentas para armazenamento e processamento de *big data*;
- Definir a melhor ferramenta de *big data* para tratar a quantidade de dados gerados pelos sensores;
- Construir uma base de dados apropriada para a quantidade de dados do projeto;
- Planejar a estrutura do aplicativo Web;
- Estudar as tecnologias Web mais atuais;
- Definir as tecnologias Web que serão usadas para o desenvolvimento do aplicativo;

- Implementar o aplicativo Web;
- Testar a integridade do aplicativo perante situações não-ideais;
- Realizar os ajustes necessários para garantir um aplicativo estável e seguro.

1.2 Organização da Monografia

Além deste capítulo introdutório, esta monografia está organizada em outros quatro capítulos:

- Capítulo 2 – Fundamentação Teórica: revisão da literatura disponível sobre temas relevantes a este trabalho, como internet das coisas e *big data*.
- Capítulo 3 – Materiais e Métodos: definição e apresentação dos métodos e da metodologia científica utilizados no desenvolvimento deste trabalho;
- Capítulo 4 – Desenvolvimento: detalhamento de como cada etapa do projeto foi desenvolvida e os problemas encontrados em cada uma delas;
- Capítulo 5 – Conclusão: revisão e análise dos resultados produzidos por este trabalho, além de propor trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

> "A próxima era da computação será fora do ambiente de computadores tradicionais. Com o paradigma da internet das coisas, muitos dos objetos que nos rodeiam estarão conectados de uma forma ou de outra. Identificação por Rádio Frequência (RFID) e as tecnologias de redes de sensores irão evoluir para satisfazer este novo desafio, no qual informações e sistemas de comunicação estarão invisivelmente integrados ao ambiente à nossa volta. Isso irá resultar na geração de quantidades enormes de dados, que precisam ser armazenados, processados e apresentados de uma forma eficiente e facilmente interpretável."[3] (GUBBI, *et al.*, 2013, tradução nossa)

2.1 Internet das Coisas

Internet das coisas é atualmente a tecnologia emergente mais empolgante entre desenvolvedores do mundo inteiro (GARTNER, 2015). Segundo Esposito (2015), uma pesquisa realizada em 2015 com mais de 500 profissionais da área de tecnologia da informação mostrou que IoT já é relevante para 58% do mercado de tecnologia, e 87% acreditam que será relevante no futuro de suas empresas.

Um dos maiores problemas da computação tradicional é a forma de se criar e capturar dados. A grande maioria de dados disponíveis na internet atualmente foram criados ou capturados por seres humanos, que possuem atenção, tempo e precisão limitados. Como consequência, é possível processar apenas uma pequena fração de toda a informação disponível à nossa volta (ASHTON, 2009).

Para explorar essa quantidade enorme de dados, é necessário computadores e redes de sensores que se encaixem no meio ambiente humano, ao invés de forçar humanos aos ambientes deles, o que torna todo o processo de captura e processamento de dados imperceptível e independente da destreza humana (WEISER, GOLD e BROWN, 1999).

2.2 Big Data

Acredita-se que o conceito de *big data* tenha se originado do advento de empresas de pesquisas que precisavam agregar e processar enormes quantidades de dados não estruturados (AMAZON WEB SERVICES, 2016b).

---

[3] Texto original: "The next wave in the era of computing will be outside the realm of the traditional desktop. In the Internet of Things (IoT) paradigm, many of the objects that surround us will be on the network in one form or another. Radio Frequency IDentification (RFID) and sensor network technologies will rise to meet this new challenge, in which information and communication systems are invisibly embedded in the environment around us. This results in the generation of enormous amounts of data which have to be stored, processed and presented in a seamless, efficient, and easily interpretable form."

Atualmente, é comum utilizar os 5 Vs da *big data* para analisar se os dados a serem armazenados e processados se encaixam nos conceitos fundamentais de *big data*:

- Volume: o enorme volume de dados disponíveis atualmente apresenta um dos maiores desafios para a computação tradicional. Muitas empresas já possuem uma grande quantidade de dados armazenados e arquivados em forma de *logs* que bases de dados relacionais tradicionais não são capazes de processar. Para lidar eficientemente com este volume de dados são necessárias ferramentas que explorem massivamente o conceito de processamento paralelo (DUMBILL, 2016), como Hadoop e Spark.

- Velocidade: segundo Taurion (2014), a velocidade é um dos Vs mais importantes da *big data*. Com a computação se inserindo cada vez mais em sistemas de decisão críticos, como por exemplo carros autônomos, a habilidade de processar todos os dados disponíveis sobre o ambiente em tempo o mais próximo possível do real é extremamente importante para que decisões sejam feitas no instante em que a situação ocorre.

- Variedade: sendo possível trabalhar com quantidades tão grande de dados, também é necessário estar preparado para lidar com uma variedade de dados tão grande quanto. Estima-se que cerca de 90% de todos os dados no mundo são não estruturados (FIGUEIREDO, 2015), o que os tornam impossíveis de serem processados eficientemente em bases de dados relacionais.

- Veracidade: de acordo com um estudo realizado pela IBM em 2014 (IBM CORPORATION, 2014), um entre cada três líderes não confiam totalmente nos dados que recebem. Porém, quando se trabalha com grandes quantidades de dados, é inviável esperar que esses dados sejam conferidos por um humano para garantir sua veracidade. Portanto, é essencial que o sistema possua ferramentas de mineração poderosas o suficiente para verificar a veracidade dos dados autonomamente.

- Valor: uma vez que a quantidade de dados já é grande, eles só devem ser armazenados e processados quando representam algum valor para o objetivo da empresa ou instituição. Para isso, são novamente necessárias ferramentas de

mineração que consigam separar e agregar apenas os dados que apresentem este valor.

Até poucos anos atrás, poucas empresas possuíam os recursos para investir em tecnologias capazes de processar grandes quantidades de dados. Porém, em 2009, The Apache Software Foundation liberou o código do Hadoop, em desenvolvimento desde 2006, divulgando o modelo de programação MapReduce, capaz de armazenar e processar enormes quantidades de dados através de um sistema de arquivos distribuídos e tolerante a falhas (chamado de HDFS - Hadoop Distributed File System).

Como resultado, Hadoop difundiu o conceito e uso de *big data* no mercado de tecnologia e se tornou a ferramenta mais utilizada até hoje (KUNTAMUKKALA, 2014).

Desde então, tecnologias de *big data* são utilizadas quando a quantidade de dados é grande, complexa e variada e difícil de ser adequadamente armazenada, administrada e processada através de bases de dados e programas tradicionais.

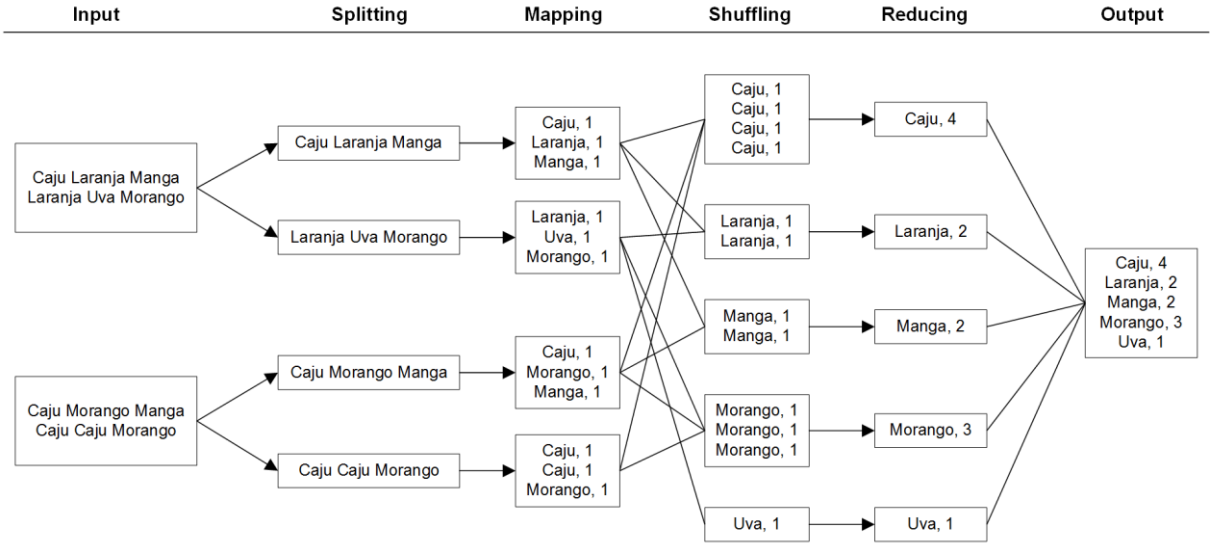
2.2.1 MapReduce

MapReduce é um modelo de programação criado para ser utilizado no Hadoop no processamento de enormes quantidades de dados. Recebe esse nome por se dividir em duas funções principais: *mapping* e *reducing*.

Para realizar o mapeamento, os arquivos desejados são recebidos do HDFS e então mapeados para pares do tipo <chave, valor> através de uma lógica de negócios definida pelo desenvolvedor. Estes resultados são então ordenados e reduzidos através de mais uma lógica de negócios para produzir o resultado final desejado.

A Figura 2 ilustra este processo para, como exemplo, realizar a contagem de palavras em dois arquivos:

Figura 2: Esquema de funcionamento do MapReduce



Fonte: elaborado pelo autor

Cada etapa do algoritmo possui uma função específica:

- *Input*: entrada dos dois arquivos, normalmente capturados de um HDFS.
- *Splitting*: as linhas dos arquivos são separadas em blocos independentes entre si.
- *Mapping*: mapeia cada palavra do bloco para uma chave do tipo <palavra, 1>.
- *Shuffling*: ordena os elementos comuns entre os blocos para facilitar a redução.
- *Reducing*: reduz todos os blocos de acordo com a lógica de negócios desejada, neste caso, somar os valores que possuem a mesma chave.
- *Output*: arquivo com a contagem de palavras, normalmente enviado para ser armazenado de volta no HDFS.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentados os materiais e métodos utilizados para o desenvolvimento deste trabalho.

3.1 Materiais

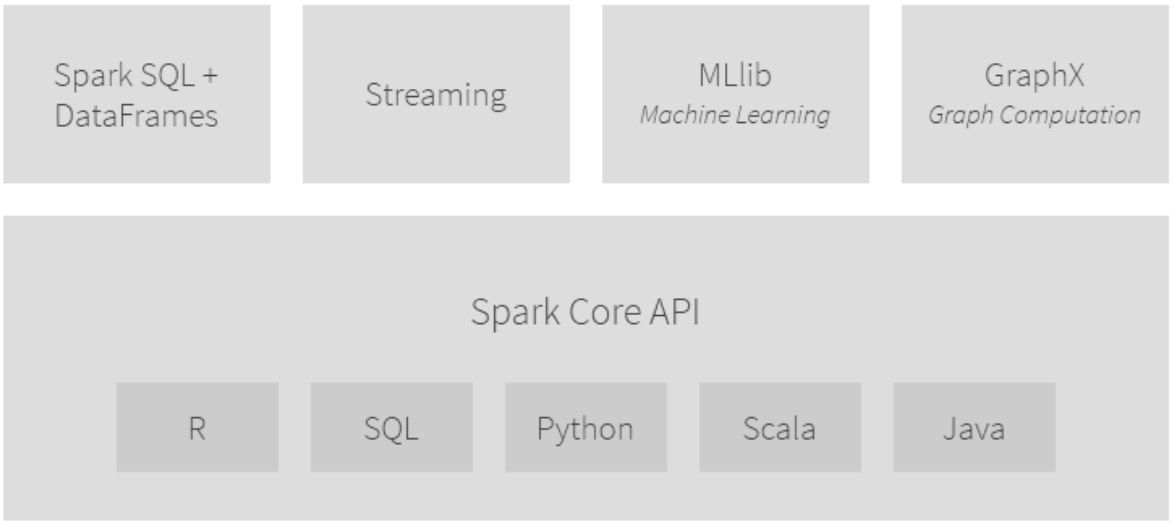
3.1.1 Spark

Spark (THE APACHE SOFTWARE FOUNDATION, 2016b) é um framework para processamento *in-memory* de *big data* em *clusters* desenvolvido em 2009 na Universidade da Califórnia, Berkeley, que doou o projeto para a The Apache Software Foundation em 2010, que o mantém desde então.

Graças à sua facilidade de uso e velocidade, o Spark se tornou a ferramenta para *big data* que mais cresce no mercado. Se comparado com Hadoop, Spark executa até 100 vezes mais rápido em memória e até 10 vezes mais rápido em disco. Ademais, segundo Esposito (2015), 39% dos usuários de Hadoop reportaram já estarem, também, utilizando Spark.

O *Spark Ecosystem*, representado na Figura 3, inclui ferramentas avançadas para as mais diversas finalidades, como aprendizado de máquina e grafos. Além disso, um dos maiores diferencias do Spark é o módulo *Spark SQL*, que, mesmo se tratando de uma base de dados não relacional, permite o uso de SQL de forma extremamente rápida em suas consultas.

Figura 3: Spark Ecosystem

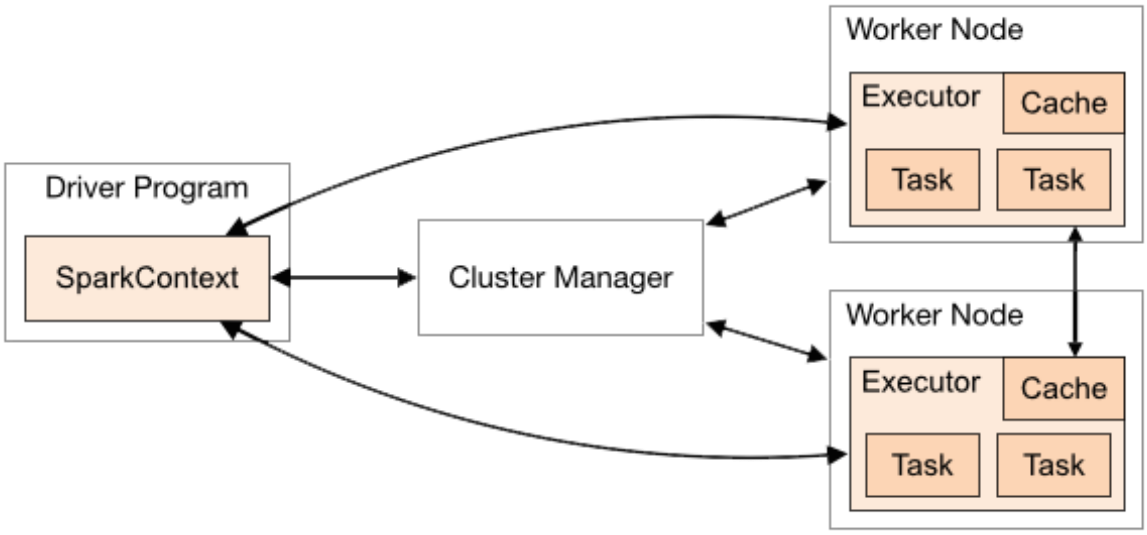


Fonte: https://databricks.com/spark/about

Spark, assim como Hadoop, trabalha com o conceito de uma máquina mestre, onde está o *Driver Program*, e máquinas escravas, chamadas de *Worker Nodes*, onde a informação é de fato armazenada. A máquina mestre se comunica principalmente com o *Cluster Manager*, um programa responsável por manter os endereços de cada máquina escrava no *cluster*, assim como o endereço de suas cópias. Por padrão, cada *Worker Node* será sempre replicado três vezes, assim, se uma das máquinas escravas falhar, o *Cluster Manager* automaticamente irá buscar os dados desejados em uma de suas duas cópias (KUNTAMUKKALA, 2014). Estas cópias também serão utilizadas caso uma máquina escrava pare de enviar o *heartbeat* para a máquina mestre, um sinal enviado periodicamente para indicar seu funcionamento.

Como mostra a Figura 4, cada tarefa solicitada pela máquina mestre é enviada para o *Cluster Manager*, que então a encaminha para a máquina escrava com o dado solicitado. A execução da tarefa é feita na máquina escrava e seu resultado é enviado de volta para o *Cluster Manager*, que o repassa para a máquina mestre.

Figura 4: Esquema de funcionamento do Spark

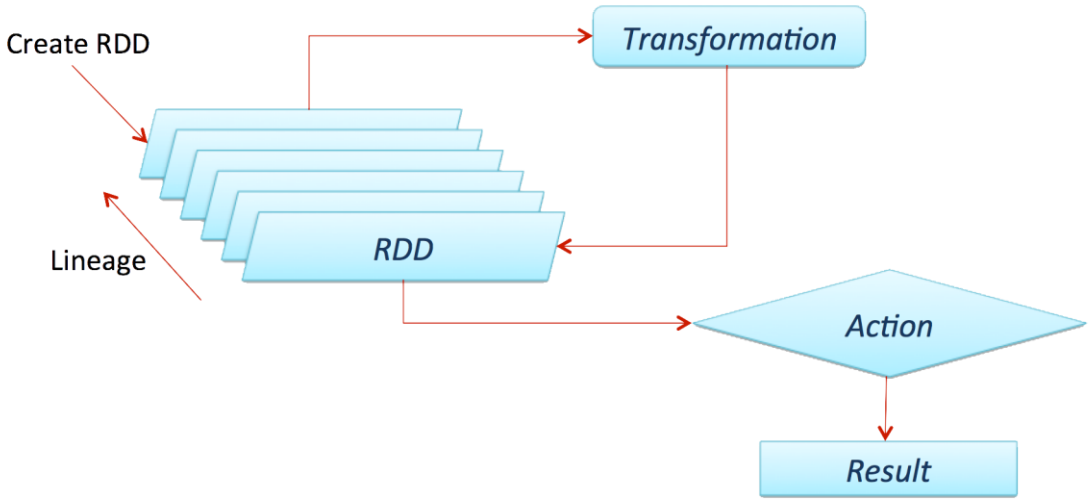


Fonte: http://spark.apache.org/docs/1.3.0/cluster-overview.html

Além disso, Spark introduz o uso de RDDs (Resilient Distributed Dataset), que funcionam como uma coleção imutável e distribuída de dados, particionados entre várias máquinas em um *cluster*. Opera com base em duas funções principais: transformações e ações.

Cada transformação aplicada em um RDD gera outro RDD, e então cada ação aplicada sobre esse novo RDD irá gerar um resultado, conforme ilustrado na Figura 5. Porém, por implementar avaliação preguiçosa, uma técnica utilizada para prorrogar o processamento até o ponto em que seu resultado é realmente necessário, as transformações são executadas apenas quando uma ação é aplicada sobre o RDD final gerado por ela.

Figura 5: Esquema de funcionamento de RDDs no Spark

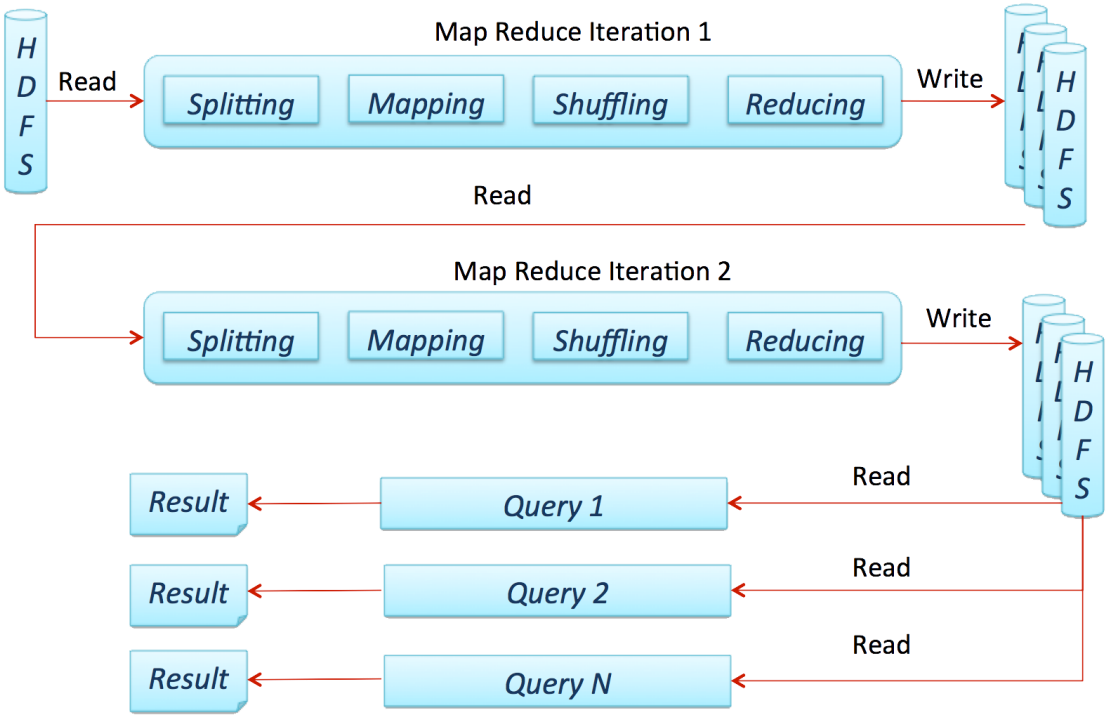


Fonte: https://dzone.com/refcardz/apache-spark

O uso de RDDs permite que o Spark faça a maior parte do processamento das informações na memória principal das máquinas, pois elas são de fato executadas apenas quando o resultado final é solicitado.

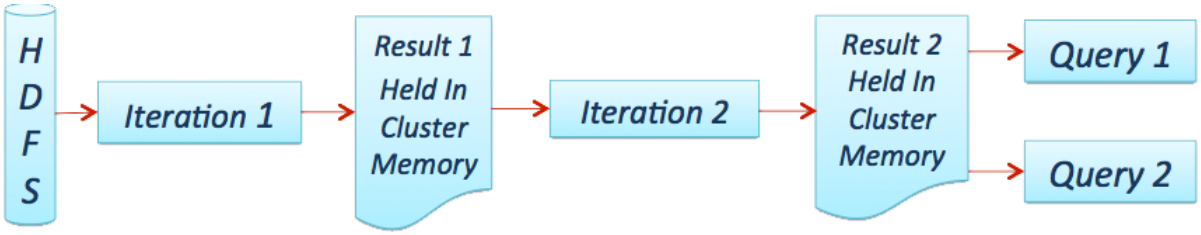
As Figuras 6 e 7 mostram a comparação entre o MapReduce realizado no Hadoop e realizado no Spark.

Figura 6: MapReduce realizado no Hadoop



Fonte: https://dzone.com/refcardz/apache-spark

Figura 7: MapReduce realizado no Spark



Fonte: https://dzone.com/refcardz/apache-spark

Com o Hadoop, o resultado de cada iteração do MapReduce precisa ser gravado em disco, enquanto no Spark o resultado de cada iteração é diretamente utilizado como entrada para a próxima iteração, sem deixar a memória principal das máquinas.

Portanto, o Spark foi escolhido para este trabalho por ser uma das ferramentas de *big data* que mais cresce atualmente e pelo módulo *Spark SQL*, que facilita o tratamento de dados estruturados ao simular uma base de dados relacional, porém ainda assim mantendo todas as vantagens de um sistema de *big data*.

3.1.2 MySQL

O MySQL é um sistema de gerenciamento de banco de dados relacional (RDBMS) de código aberto e é, atualmente, o RDBMS mais usado no mundo (ORACLE CORPORATION, 2016). Segundo Nixon (2014), o MySQL é extremamente poderoso e rápido, até mesmo em máquinas que não possuem um hardware avançado.

Diferentemente das ferramentas de *big data*, o MySQL armazena os dados em tabelas relacionais e utiliza a linguagem SQL para a manipulação destes dados.

O MySQL foi escolhido para este trabalho para ser usado em conjunto com o Spark, conforme detalhado no Capítulo 4, pois oferece uma melhor interação com o aplicativo Web.

3.1.3 Amazon Web Services

Amazon Web Services (AWS) é uma suíte de serviços para computação em nuvem oferecidos pela Amazon.com. Os serviços são oferecidos através de 33 zonas dentro de 12 diferentes regiões geográficas no mundo (AMAZON WEB SERVICES, 2016a).

De acordo com um relatório publicado pela Gartner em 2016, é estimado que os consumidores da AWS estão implementando 10 vezes mais servidores do que as próximas 14 empresas mais populares no segmento combinadas. Consumidores incluem NASA, Pinterest, Netflix e CIA (Agência Central de Inteligência do governo dos Estados Unidos) (LEONG, *et al.*, 2016).

Atualmente a AWS oferece 83 diferentes serviços de nuvem (AMAZON WEB SERVICES, 2016c), sendo os mais difundidos o Amazon Elastic Compute Cloud (Amazon EC2) e o Amazon Simple Storage Service (Amazon S3).

O Amazon EC2 são servidores com fácil acesso que permitem controle total por parte do usuário, o que permite sua configuração e uso com extrema facilidade, especialmente se comparados com a configuração e uso de servidores privados.

O Amazon S3 é um serviço para armazenamento de objetos na nuvem, ou seja, é um servidor otimizado para armazenamento de dados. Assim como o EC2, também possui uma interface intuitiva e simples.

Por fim, criado especificamente para o uso em aplicações de *big data*, o Amazon Elastic MapReduce (Amazon EMR), é um servidor EC2 integrado com

frameworks para processamento de *big data*, como Hadoop e Spark, poupando o usuário da árdua tarefa de instalar e configurar todas as ferramentas necessárias para o funcionamento do *cluster*.

3.1.4 NodeJS

NodeJS é uma plataforma de desenvolvimento JavaScript criada em 2009 por Ryan Dahl (TRAINING.COM, 2016) com o objetivo de ser usada em aplicações em que a comunicação entre servidor e usuário precisa ocorrer em tempo real.

Um dos maiores atrativos do NodeJS é seu gerenciador de pacotes, npm, que é atualmente considerado o maior ecossistema de bibliotecas de código aberto do mundo (NODE.JS FOUNDATION, 2016b).

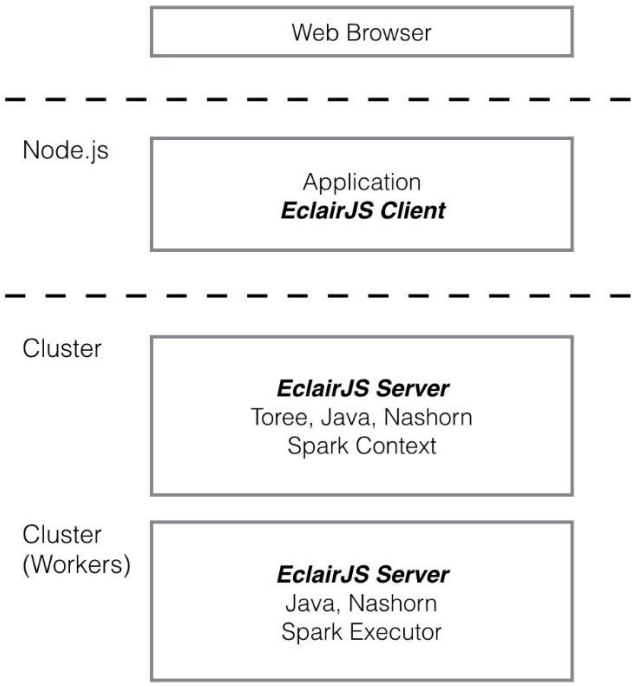
O NodeJS é utilizado neste trabalho como o *back-end* do aplicativo Web, ou seja, é responsável por disponibilizar todas as páginas ao usuário e por toda a comunicação com a base de dados.

3.1.5 EclairJS

EclairJS é uma API que permite acesso à servidores Spark através das linguagens JavaScript e NodeJS. É um projeto novo que começou a ser desenvolvido pela comunidade no início de 2016, portanto ainda não oferece muitas funcionalidades.

O projeto surgiu da união de dois projetos separados, o EclairJS Client e EclairJS Server, que hoje são dois componentes integrados ao projeto, permitindo assim que aplicações escritas em NodeJS sejam executadas remotamente no servidor Spark. A Figura 8 ilustra a estrutura de funcionamento do EclairJS.

Figura 8: Estrutura de funcionamento do EclairJS



Fonte: http://eclairjs.github.io/

3.1.6 Express

O Express.js, ou simplesmente Express, é um *framework* para o desenvolvimento de aplicativos Web em NodeJS. Foi criado em 2010 por TJ Holowaychuk, que descreveu o sistema como um *framework* minimalista, mas com muitas funções disponíveis como *plug-ins*.

Em 2014 o projeto foi adquirido pela empresa StrongLoop, a qual em 2015 foi adquirida pela IBM, que no início de 2016 decidiu então deixar o desenvolvimento do Express sob os cuidados da Node.js Foundation, que o mantém até hoje (NODE.JS FOUNDATION, 2016a). Desde então, o *framework* é considerado um dos mais usados pela comunidade do NodeJS.

Para ser o mais flexível possível, o Express não define uma estrutura padrão para o aplicativo, o que, através de regras de roteamento de páginas no servidor, permite que o aplicativo seja estruturado como o usuário deseja.

3.1.7 Bootstrap

"Bootstrap é o *framework* para HTML, CSS e JavaScript mais popular para o desenvolvimento de projetos Web responsivos e *mobile first*"[4] (BOOTSTRAP, 2016, tradução nossa).

O Bootstrap foi desenvolvido em 2010 por Mark Otto e Jacob Thornton, funcionários do Twitter, como um projeto para encorajar consistência entre as ferramentas internas da empresa, chamado Twitter Blueprint. Em 2011, devido a popularidade do projeto, ele foi renomeado para Bootstrap e lançado como um projeto de código aberto, que é atualmente mantido por um grupo com 7 desenvolvedores principais (BOOTSTRAP, 2016).

Este *framework* foi escolhido para este trabalho pela facilidade em criar aplicativos Web responsivos sem necessitar um grande conhecimento de linguagens de *front-end*, como CSS.

3.2 Metodologia

Considerando que já exista uma rede de sensores conectada à internet, a primeira fase deste projeto foi analisar a quantidade de dados que será gerada pelos sensores e então, através da análise das principais tecnologias para *big data*, como Cassandra (THE APACHE SOFTWARE FOUNDATION, 2016a), Storm (THE APACHE SOFTWARE FOUNDATION, 2016c) e Spark (THE APACHE SOFTWARE FOUNDATION, 2016b), planejar uma base de dados adequada para armazenar e manipular esta quantidade de dados.

A segunda fase do projeto foi planejar e desenvolver o *back-end* do aplicativo, responsável por receber estes dados do *gateway* e agrupá-los na base de dados e então recuperá-los novamente para a exibição no aplicativo Web. Para essa fase do projeto foi necessário o estudo das linguagens Node.js (NODE.JS FOUNDATION, 2016b) e Scala (ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2016), além dos frameworks Express (NODE.JS FOUNDATION, 2016a) e Play (LIGHTBEND, 2016).

Na terceira e última fase do projeto, foi desenvolvido o aplicativo Web para visualizar os dados da base de dados e inferir informações úteis sobre a

[4] Texto original: "Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web."

movimentação dos dispositivos conectados à rede Wi-Fi do prédio, como, por exemplo, a última posição conhecida de cada dispositivo. Nesta fase do projeto foram utilizadas ferramentas como Bootstrap (BOOTSTRAP, 2016).

4 DESENVOLVIMENTO

Neste trabalho, foi configurado um *cluster* de servidores EC2 da AWS e desenvolvido um aplicativo Web para mostrar ao usuário os dados armazenados nestes servidores.

O *cluster* de servidores foi configurado com Spark e pode ser facilmente expandido, portanto está preparado para lidar com grandes quantidades de dados. O servidor é voltado para o uso em sistemas de geolocalização internos, e estes estão prontos para serem conectados à uma rede de sensores que provenham os dados de localização de cada dispositivo dentro de um prédio.

Para que o aplicativo Web consiga se comunicar eficientemente com o servidor, no *cluster* também foi instalado uma base de dados MySQL, configurada para funcionar juntamente com a arquitetura em paralelo do Spark.

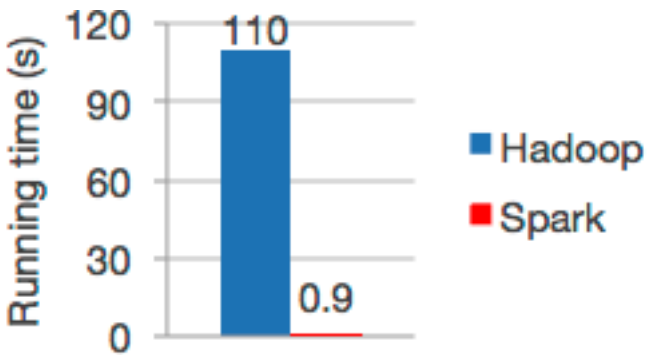
O aplicativo Web foi desenvolvido utilizando o *framework* Express com a linguagem NodeJS para o *back-end*, e o *framework* Bootstrap com as linguagens HTML e CSS para o *front-end*.

4.1 Implementação do Servidor

O servidor da aplicação foi desenvolvido para ser escalável, ou seja, pode ser facilmente expandido para suportar um volume muito maior de dados. Para que isto seja possível, foi necessária pesquisa sobre várias ferramentas de *big data* que funcionam com arquiteturas altamente paralelas, como Hadoop e Spark, ambas desenvolvidas pela Apache Software Foundation.

O Spark foi escolhido por ser uma das ferramentas mais modernas disponíveis, podendo ser até 100 vezes mais rápida que o Hadoop, quando funcionando em memória, e até 10 vezes mais rápida quando funcionando em um disso rígido (THE APACHE SOFTWARE FOUNDATION, 2016b), como é mostrado na Figura 9, que compara o tempo de execução de uma tarefa no Hadoop e no Spark.

Figura 9: Tempo de execução do Hadoop e Spark

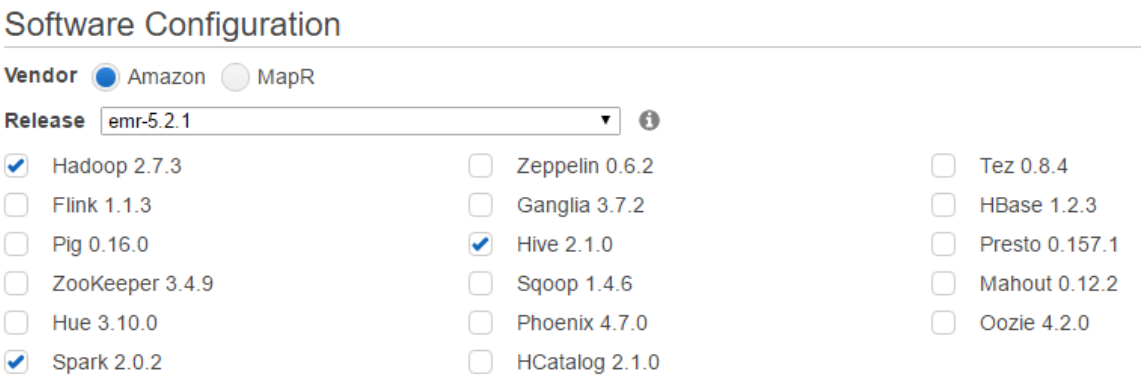


Fonte: baseado em https://spark.apache.org/

O Amazon Web Services (AWS) foi escolhido para hospedar os servidores por sua facilidade em ter seus serviços escalados e por ser atualmente o serviço de *cloud* mais usado e que mais cresce no mundo (AMAZON WEB SERVICES, 2016a).

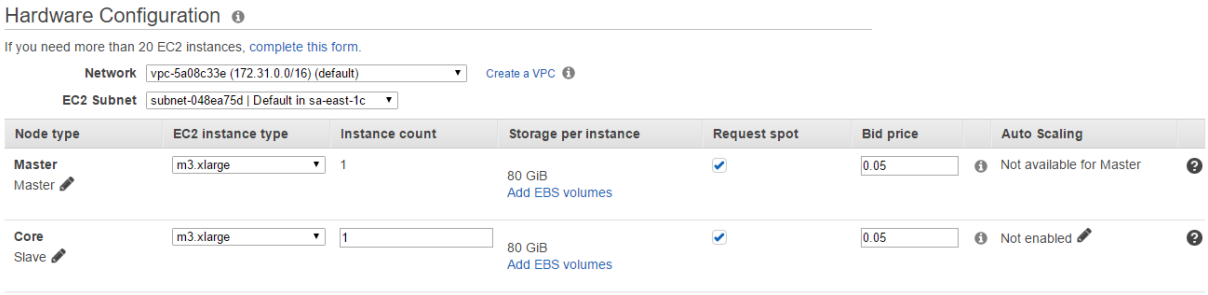
Inicialmente, foram testados os servidores Elastic MapReduce (EMR), que, pelo pagamento de uma pequena taxa adicional, já possuem algumas ferramentas para *big data*, como Spark, instaladas e pré-configuradas. Com as configurações necessárias para este trabalho, como mostrado nas Figuras 10 e 11, a criação dos servidores leva em média 8 horas para ser concluída.

Figura 10: Configurações de software para EMR



Fonte: elaborado pelo autor

Figura 11: Configurações de hardware para EMR



Fonte: elaborado pelo autor

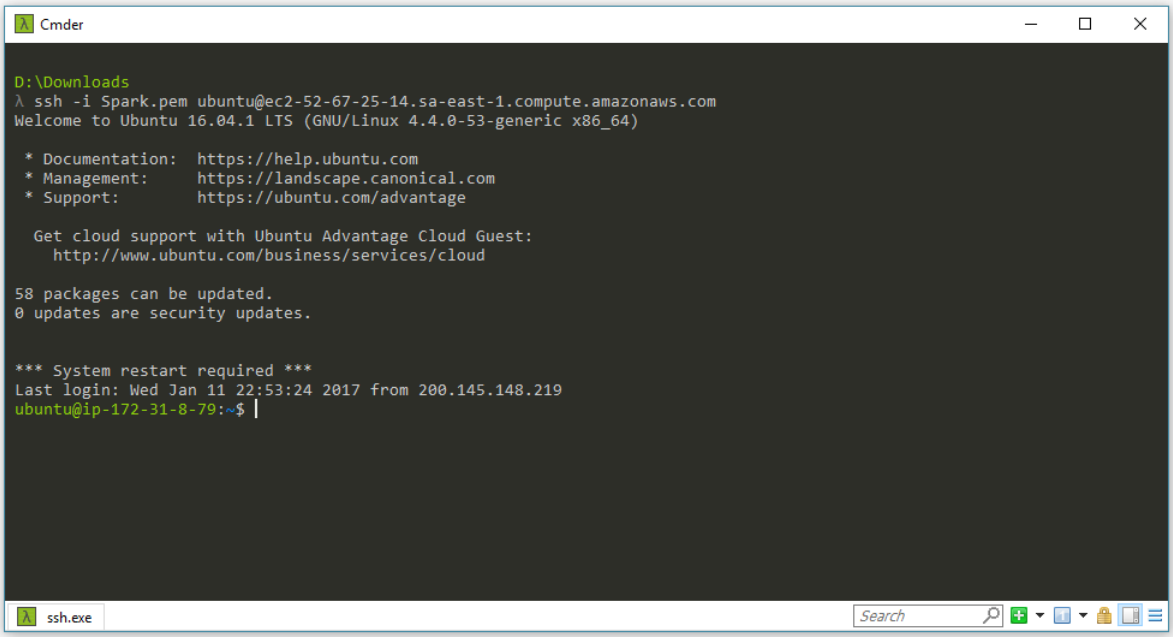
Portanto, decidiu-se não usar servidores EMR para este trabalho, pois se tratando de um ambiente universitário onde os fundos para o projeto eram limitados, foi necessário o uso de servidores que poderiam ser ligados e desligados com facilidade e rapidez conforme fosse necessário a realização de testes.

Foram escolhidos então servidores EC2, do tipo *m4.large*, que apresenta as configurações de hardware recomendadas para o Spark, com o sistema operacional Ubuntu Server 16.04.

Para este projeto foram criados dois servidores, um para a configuração da máquina mestre, ou *master*, e um para a configuração de uma máquina escrava, ou *slave*. As configurações da máquina escrava podem ser facilmente replicadas conforme o sistema é expandido para adição de novas máquinas ao *cluster*.

Os servidores da AWS foram acessados através de *secure shell* (SSH), um protocolo de rede criptográfico que permite acessar remotamente o terminal dos servidores através do uso de uma chave de segurança, como mostra a Figura 12.

Figura 12: Acesso SSH ao servidor da AWS



Fonte: elaborado pelo autor

4.1.1 Instalação e configuração do Hadoop

O Spark utiliza diversas dependências do Hadoop, portanto é necessário ter o Hadoop instalado e configurado em todas as máquinas antes de se instalar o Spark.

O primeiro passo é instalar o Java através do comando:

```
sudo apt-get install openjdk-8-jdk
```

Então é necessário fazer o download do Hadoop, disponível em seu site oficial, e extraí-lo para a pasta “/usr/local/hadoop”. O próximo passo consiste em configurar as variáveis de ambiente do sistema operacional:

```
HADOOP_HOME=/usr/local/hadoop
HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

Uma vez que o Hadoop esteja instalado em todas as máquinas do *cluster*, sua configuração é feita através da edição dos seguintes arquivos:

- $HADOOP_CONF_DIR/core-site.xml: neste arquivo é definido o sistema de arquivos do Hadoop. Por padrão, a localização *localhost* é utilizada, mas para o nosso trabalho utilizaremos o DNS da máquina mestre na porta 9000.
- $HADOOP_CONF_DIR/yarn-site.xml: neste arquivo é configurado o YARN, responsável por separar as funcionalidades de gerenciamento de recursos e agendamento de tarefas para diferentes máquinas. Nele é necessário definir o DNS da máquina mestre como o hospedeiro do gerenciador de recursos (*resource manager host*).
- $HADOOP_CONF_DIR/mapred-site.xml: neste arquivo, o YARN deve ser definido como o *framework* e o DNS da máquina mestre na porta 54311 como o rastreador de tarefas (*job tracker*).
- $HADOOP_CONF_DIR/hdfs-site.xml: neste arquivo é definido o fator de replicação a ser usado. Neste trabalho é usado um fator de replicação de 3 para cada bloco de dados do HDFS.

Uma vez que estes arquivos estejam configurados em todas as máquinas do *cluster*, o próximo passo é configurar a máquina mestre para que ela consiga se comunicar com todas as máquinas escravas, o que é feito alterando-se os seguintes arquivos:

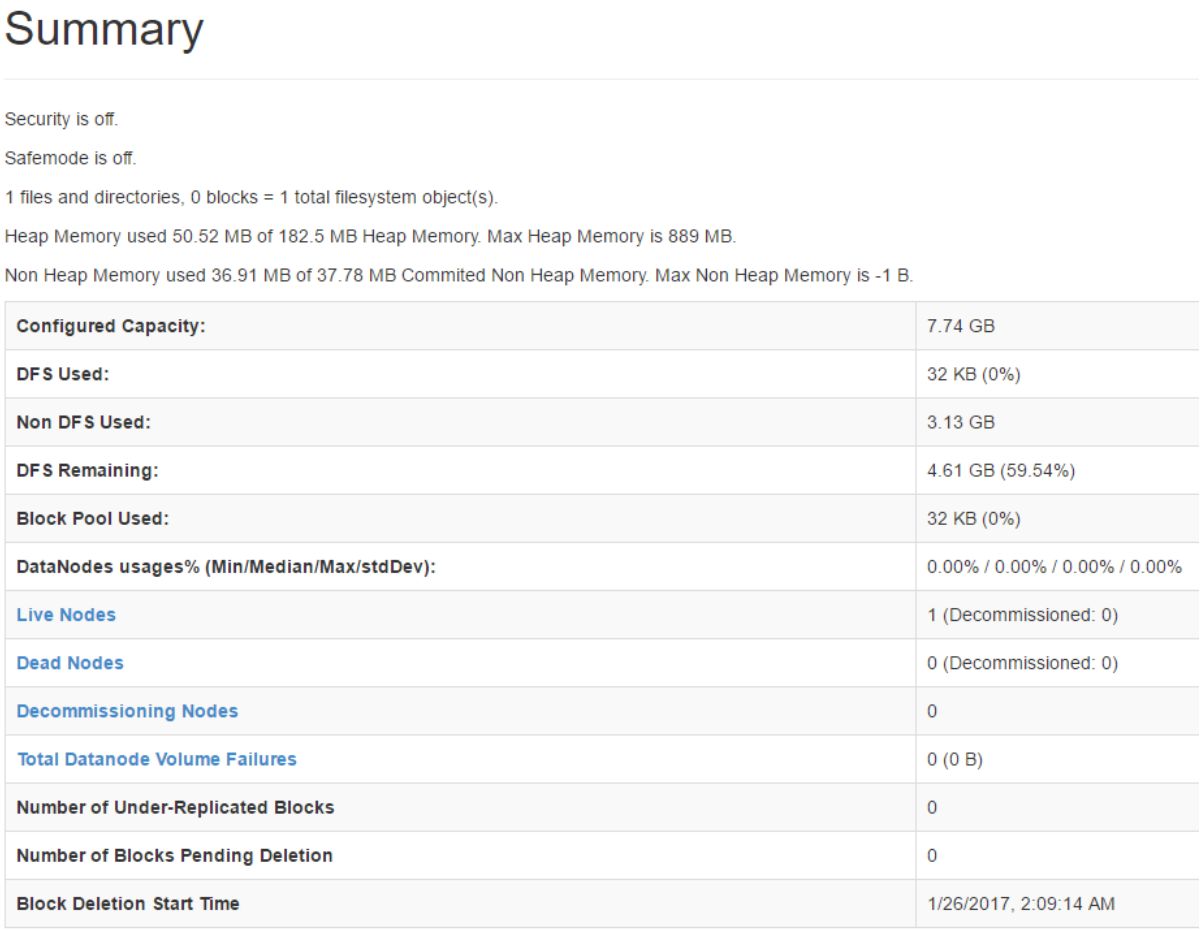
- /etc/hosts: este arquivo é usado para relacionar o nome de servidores com o seu endereço IP ou DNS. Nele deve ser incluso o DNS e nome do servidor, que pode ser obtido em cada máquina através do comando *echo $(hostname)*, de todas as máquinas do *cluster*, inclusive da própria máquina mestre.
- $HADOOP_CONF_DIR/masters: arquivo que contém o nome do servidor da máquina mestre.
- $HADOOP_CONF_DIR/slaves: arquivo que contém uma lista com o nome do servidor de todas as máquinas escravas.

Por fim, tendo estes arquivos configurados corretamente em cada máquina, o *cluster* pode ser iniciado através do comando:

$HADOOP_HOME/sbin/start-dfs.sh

O estado e um resumo do *cluster* podem ser acessados através de um navegador pelo DNS da máquina mestre na porta 50070, como mostra a Figura 13:

Figura 13: Resumo do cluster Hadoop na porta 50070



Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 50.52 MB of 182.5 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 36.91 MB of 37.78 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	7.74 GB
DFS Used:	32 KB (0%)
Non DFS Used:	3.13 GB
DFS Remaining:	4.61 GB (59.54%)
Block Pool Used:	32 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	1/26/2017, 2:09:14 AM

Fonte: elaborado pelo autor

4.1.2 Instalação e configuração do Spark

Tendo o Hadoop instalado e configurado em todas as máquinas do *cluster*, o primeiro passo é instalar o compilador da linguagem Scala, que é utilizada para o funcionamento do Spark, através do seguinte comando:

```
sudo apt-get install scala
```

Em seguida, faça o download do Spark através de seu site oficial e o extraia para a pasta “/usr/local/spark”, e então defina a seguinte variável de ambiente:

SPARK_HOME=/usr/local/spark

Assim como o Hadoop, as configurações do Spark são feitas através da edição de arquivos. O seguinte arquivo deve ser editado em todas as máquinas do *cluster*:

- $SPARK_HOME/conf/spark-env.sh: neste arquivo é definido o DNS da máquina atual e o nível de paralelismo que o Spark deve usar na máquina, chamado de "*spark worker cores*", o que representa o número de *threads* permitidas em cada máquina e não deve ser confundido com o número de núcleos físicos do processador. É recomendado o uso de 3 *threads* para cada núcleo do processador, portanto, para este trabalho foram permitidas 6 *threads*, uma vez que os servidores EC2 do tipo *m4.large* possuem processadores com dois núcleos.

Por fim, na máquina mestre o seguinte arquivo deve ser editado:

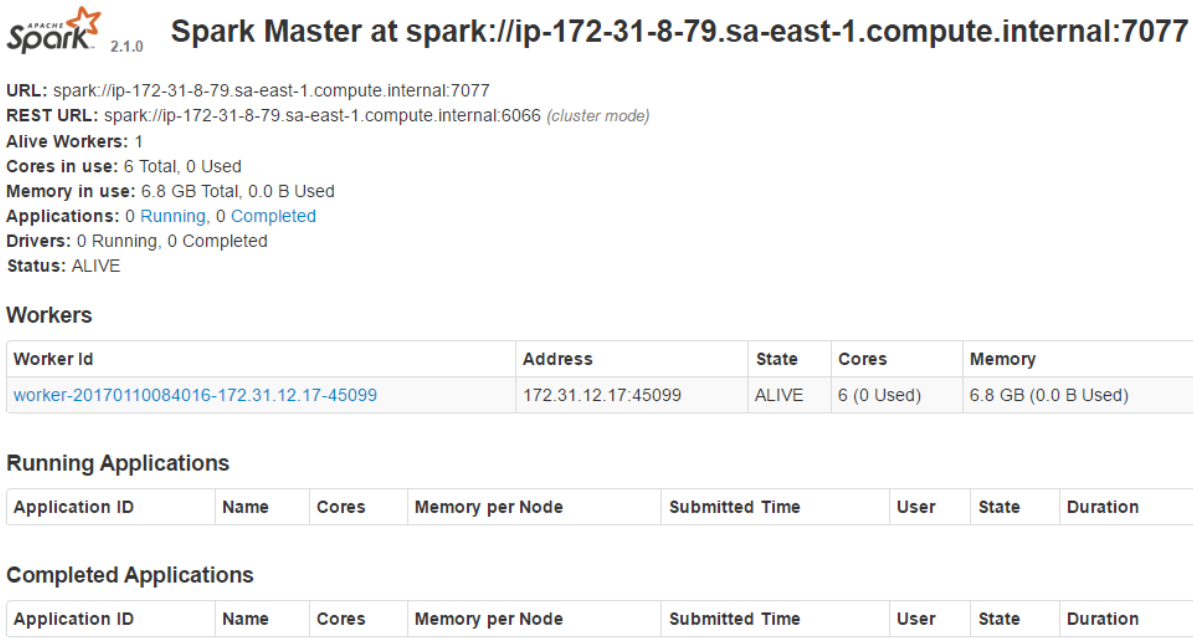
- $SPARK_HOME/conf/slaves: lista de DNS de todas as máquinas escravas do *cluster*.

Assim, o *cluster* pode ser iniciado através do seguinte comando:

$SPARK_HOME/sbin/start-all.sh

Similar ao Hadoop, um resumo do estado atual do *cluster* pode ser acessado através de um navegador no DNS da máquina mestre na porta 8080, como mostra a Figura 14.

Figura 14: Resumo do cluster Spark na porta 8080



Fonte: elaborado pelo autor

4.1.3 Problemas enfrentados

Os servidores EC2 usados neste trabalho são instâncias do tipo *spot*, o que permite que o usuário ofereça o preço que desejar para utilizar instâncias que não estejam em uso, e então os lances mais altos são atendidos, o que permite que os custos com o servidor sejam diminuídos significantemente. Porém, instâncias do tipo *spot* podem apenas ser destruídas (*terminate*), o que também destrói completamente todas as instalações e configurações feitas na máquina.

A instalação e configuração do Spark é trabalhosa e demorada, portanto seria inviável instalar e configurar todas as dependências sempre que fosse necessário ligar o servidor para testes.

A primeira tentativa para solucionar o problema foi alterar as configurações de cada instância para terem o HD salvo como um Elastic Block Store (EBS) antes de serem destruídas, e então, ao criar novas instâncias, ligá-las a estes HD onde o Spark já foi instalado e configurado. Porém, ao mesclar o HD das novas instâncias com os das antigas, muitas configurações do sistema operacional acabavam se perdendo e era necessário reconfigurar grande parte dos servidores.

A segunda tentativa foi através do uso de Amazon Machine Images (AMIs) personalizadas. A AMI representa uma imagem do sistema operacional que será

instalado nas instâncias, e, ao criar uma nova instância, é possível selecionar uma das AMIs padrões da AWS, ou então selecionar AMIs personalizadas que podem ser geradas através de antigas instâncias. Assim, foi salva uma AMI para cada tipo de instância do *cluster*, após elas serem configuradas, possibilitando que fossem destruídas e então criadas novamente com o sistema operacional e todos os seus programas e configurações iguais ao de quando a AMI foi salva.

Porém, grande parte das configurações do Hadoop e Spark são dependentes do endereço DNS das máquinas mestre e escravas, e, ao criar uma nova instância, um novo endereço DNS é atribuído à cada instância, necessitando refazer a maior parte das configurações.

Para solucionar este problema, foram utilizados Elastic IPs, um serviço da AWS que permite o aluguel de endereços IPs e DNS por uma taxa muito menor que o de uma instância EC2. Assim, foi possível manter um Elastic IP para cada instância do *cluster*, e então, ao ser criada uma nova instância com a AMI personalizada, alocar estes Elastic IPs para cada instância, garantindo que o DNS das máquinas fosse sempre o mesmo.

No entanto, quando a máquina mestre se conecta à cada uma das máquinas escravas pela primeira vez, é criado uma forma de impressão digital (*fingerprint*) nas máquinas escravas, possibilitando assim que a máquina mestre se conecte novamente com maior facilidade. Portanto, mesmo mantendo o mesmo endereço DNS, a conexão entre a máquina mestre e as máquinas escravas ainda precisava ser manualmente desfeita e feita novamente.

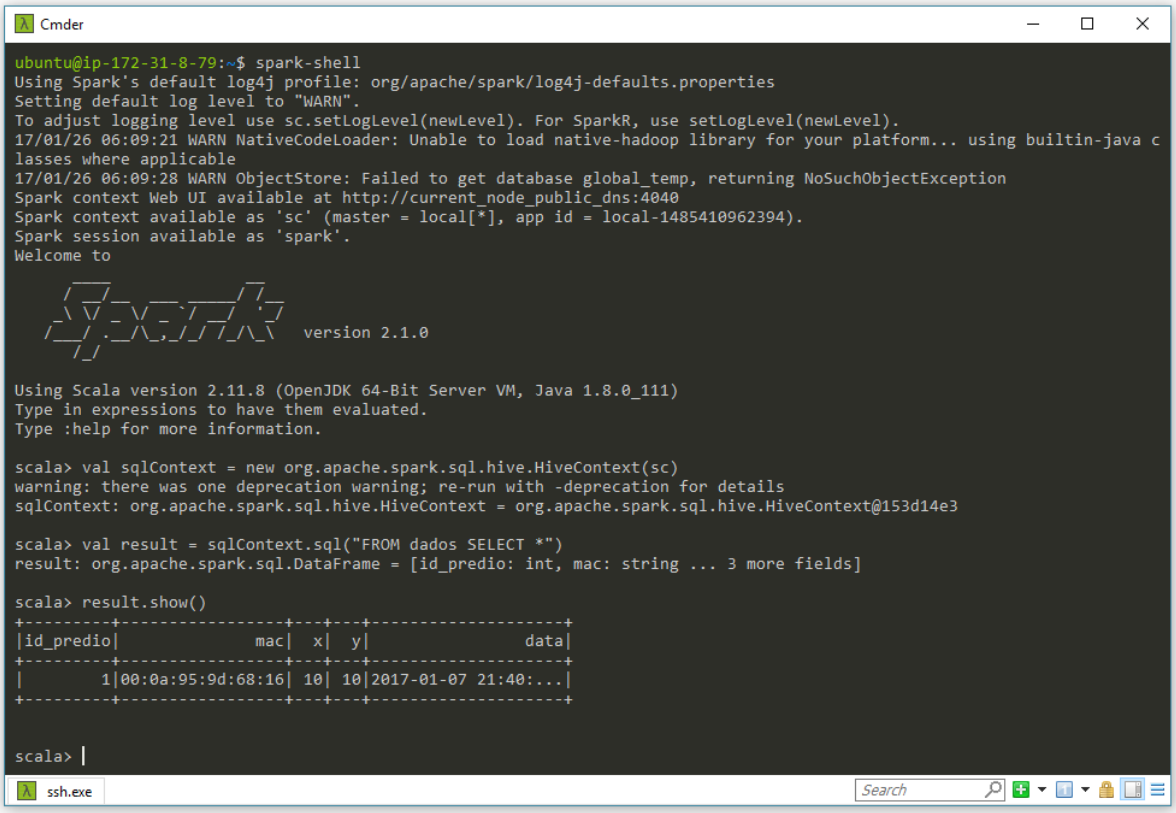
Em vista disso, mesmo a um custo maior que o esperado, foi decidido manter todas as instâncias do *cluster* funcionando ininterruptamente, uma vez que se mostrou inviável ter de reconfigurar o Hadoop e Spark sempre que as instâncias novas fossem criadas.

4.2 Implementação da Base de Dados

A base de dados deste trabalho foi inicialmente projetada para funcionar através da ferramenta Hive on Spark, que permite o uso da infraestrutura de dados do Hive, feita para o Hadoop, no Spark, que pode então ser acessada e manipulada com a linguagem SQL através do Spark SQL.

Para testar as funcionalidades e conectividade da base de dados, através do Spark Shell, uma ferramenta que permite de modo fácil executar algumas ações do Spark sem a necessidade de desenvolver um programa completo, foi criada uma simples tabela em Hive, como mostra a Figura 15.

Figura 15: Tabela em Hive no Spark Shell



Fonte: elaborado pelo autor

4.2.1 Problemas enfrentados

O Spark é uma ferramenta criada para manusear quantidades massivas de dados, e, apesar de ser otimizado para funcionar muito mais rapidamente que ferramentas similares, nativamente não é capaz de trabalhar em situações interativas que necessitam de respostas variadas em tempo real, como em um aplicativo Web.

Para solucionar este problema, é possível instalar o MySQL, um sistema de gerenciamento de banco de dados relacional, no mesmo *cluster* onde o Spark está instalado e, então, manusear a base de dados do MySQL utilizando comandos do Spark, o que permite que o MySQL use a arquitetura altamente paralela do Spark, o deixando 10 vezes mais rápido (RUBIN, 2016).

Desta forma, dentro do servidor a base de dados funciona em cima da arquitetura paralela do Spark, mas, por se tratar de uma base de dados MySQL, pode ser acessada facilmente através do aplicativo Web.

4.2.2 Instalação e configuração do MySQL

O MySQL foi instalado normalmente na máquina mestre do *cluster* de servidores da AWS, através dos seguintes comandos:

```
sudo apt-get install mysql-server
sudo apt-get install mysql-client
```

A única configuração necessária durante a instalação é a definição de uma senha para o *root* do MySQL, utilizada para acessar a base de dados posteriormente.

A base de dados foi criada através do MySQL Shell, seguindo a estrutura ilustrada na Figura 16.

Figura 16: Estrutura da base de dados

Fonte: elaborado pelo autor

A base de dados foi estruturada de modo que o sistema suporte a inclusão de vários prédios, que são identificados na tabela “predios”, juntamente com a largura e comprimento, em metros, do mapa do prédio que será salvo no sistema. Para cada prédio incluído no sistema, automaticamente são criadas duas novas tabelas, “dispositivos_*id*” e “paredes_*id*”, onde id representa o id do novo prédio na tabela “predios”.

Na tabela “dispositivos_*id*” serão gravadas as informações de geolocalização dos dispositivos presentes no prédio, que podem ser capturadas e calculadas através de uma rede de sensores no prédio. A localização deve ser gravada na forma de coordenadas x e y em relação ao mapa do prédio.

A tabela “paredes_id” indica as paredes presentes no prédio na forma de retas com um ponto de início e término. Através destas tabelas, o mapa do prédio pode ser construído dinamicamente para visualização no aplicativo Web.

Foi escolhido esta estrutura pois, de acordo com Tocker (2013), separar os dados em tabelas menores é uma das formas de se otimizar o funcionamento do MySQL ao trabalhar com grandes quantidades de dados.

4.3 Implementação do Aplicativo Web

O aplicativo Web desenvolvido neste trabalho é um protótipo para possibilitar a visualização dos dados armazenados no servidor. Portanto, apesar do servidor estar preparado para ser usado em sistemas maiores com vários prédios, o aplicativo apresenta apenas uma página com o mapa de um prédio, gerado dinamicamente de acordo com as informações da base de dados, e a posição dos dispositivos dentro dele.

4.3.1 Problemas enfrentados

O maior problema enfrentado na implementação do aplicativo Web foi a conexão com o servidor Spark.

A API EclairJS foi usada inicialmente, juntamente com o *framework* Express, na tentativa de se conectar à tabela Hive criada no Spark através do Spark Shell. Porém, o EclairJS, por ser uma API nova que ainda está em desenvolvimento, atualmente não possui suporte ao Hive on Spark, necessário para manipular tabelas Hive no Spark.

A segunda tentativa para solucionar este problema foi trocar o Express pelo Play Framework, um *framework* desenvolvido pela Lightbend e Zengularity para desenvolvimento de aplicativos Web em Java e Scala. Como os dois *frameworks* funcionam de forma bem diferentes, foi necessário reestruturar completamente o aplicativo Web. O objetivo com essa troca era desenvolver um *script* em Scala, linguagem nativa do Spark, que conseguisse desta forma manusear a tabela Hive existente no servidor Spark.

Através do Play Framework foi possível desenvolver um script em Scala que se conectava com sucesso ao servidor Spark, porém, o único meio de se acessar o servidor Spark é através do comando “spark-submit”, no qual é necessário fornecer um programa já compilado, que será então executado no Spark. Tratando-se de um aplicativo Web onde as consultas à base de dados precisam ser dinâmicas, a necessidade de compilar um programa inteiro para cada requisição à base de dados tornou essa alternativa inviável. Portanto, decidiu-se abandonar o Play Framework e retornar o aplicativo para o Express.

A terceira alternativa considerada foi acessar o terminal do servidor da máquina mestre onde o Spark está instalado através do NodeJS, criar um arquivo de texto com as instruções SQL desejadas, carregar esse arquivo para ser executado no Spark Shell, salvar todo o texto retornado pelo Spark Shell em outro arquivo de texto e, então, capturar as informações desejadas deste arquivo. Porém, alguns testes revelaram que o texto retornado pelo Spark Shell era mais complexo do que o esperado, o que requereria uma mineração de dados profunda para filtrar apenas os dados desejados, o que não era o foco deste trabalho. Portanto, esta alternativa também foi descartada.

Por fim, a solução encontrada foi o uso do MySQL dentro do *cluster* Spark, sendo assim possível manipular a base de dados através da API EclairJS.

5 CONCLUSÃO

Conforme definido na introdução, este trabalho teve como objetivo desenvolver um servidor e um aplicativo Web para armazenamento e visualização de dados de localização de dispositivos gerados a partir de uma rede de sensores de Wi-Fi ou bluetooth fixados em pontos pré-determinados dentro de um prédio, com o propósito de construir uma base de dados que possa ser utilizada em sistemas de geoposicionamento internos.

Para garantir que o sistema seja facilmente escalável e esteja preparado para lidar com grandes quantidades de dados, foram estudados diferentes conceitos e tecnologias, com o foco principal em ferramentas de *big data* com arquitetura altamente paralela.

Conforme justificado no capítulo 4, foi escolhido para este trabalho o uso de um *cluster* de servidores da Amazon Web Services com o Spark instalado e configurado para funcionar paralelamente em todas as máquinas. Porém, problemas na conexão do aplicativo Web com o servidor necessitaram mudanças na estrutura da base de dados.

Para solucionar este problema, foi instalado o MySQL no servidor, juntamente com o Spark, para que este possa usufruir da arquitetura paralela do Spark, mas ainda assim seja eficiente ao se comunicar com o aplicativo Web.

Portanto, é possível concluir que as ferramentas de *big data* atualmente disponíveis no mercado ainda não estão otimizadas para o uso em situações onde é necessária uma resposta dinâmica em tempo real.

Devido aos problemas com a implementação do servidor e da base de dados, foi possível desenvolver apenas um protótipo do aplicativo Web, permitindo de forma simples a visualização dos dados armazenados.

Conclui-se, portanto, que este trabalho atingiu todos os objetivos propostos em relação ao servidor e base de dados, porém, em relação ao aplicativo Web, foi possível o desenvolvimento de apenas um protótipo do aplicativo originalmente proposto.

5.1 Trabalhos futuros

Como trabalhos futuros propõem-se melhorias e novas funcionalidades para o aplicativo Web, tais como:

- Suporte à visualização e criação de vários prédios;
- Edição do mapa de prédios existentes;
- Novas formas de visualização dos dados, como mapas de calor;
- Análise dos dados armazenados, permitindo que seja inferida informações como o local mais frequentado dentro do prédio;
- Implementação de uma camada de segurança, para que o aplicativo possa ser usado em servidores de acesso público.

REFERÊNCIAS

AMAZON WEB SERVICES. AWS Global Infrastructure, 2016a. Disponivel em: <https://aws.amazon.com/about-aws/global-infrastructure/>. Acesso em: 10 Maio 2016.

AMAZON WEB SERVICES. Big Data Technology Fundamentals. **Amazon Web Services - Training and Certification**, 2016b. Disponivel em: <http://aws.amazon.com/training/course-descriptions/bigdata-fundamentals/>. Acesso em: 10 Maio 2016.

AMAZON WEB SERVICES. Cloud Products, 2016c. Disponivel em: <https://aws.amazon.com/products/>. Acesso em: 22 Dezembro 2016.

ASHTON, K. That 'Internet of Things' Thing. **RFID Journal**, 2009. Disponivel em: <http://www.rfidjournal.com/articles/view?4986>. Acesso em: 09 Março 2016.

BLECKY. SubPos Positioning System. **Hackaday.io**, 2016. Disponivel em: <https://hackaday.io/project/4872-subpos-positioning-system>. Acesso em: 21 Março 2016.

BOOTSTRAP. Bootstrap, 2016. Disponivel em: <http://getbootstrap.com/>. Acesso em: 10 Março 2016.

CHEN, G.; KOTZ, D. A Survey of Context-Aware Mobile Computing Research. **Dartmouth Computer Science Technical Report TR2000-381**, Dartmouth College, 2000. Disponivel em: <http://www.cs.dartmouth.edu/reports/TR2000-381.pdf>. Acesso em: 22 Março 2016.

DJUKNIC, G. M.; RICHTON, R. E. Geolocation and Assisted GPS. **IEEE Computer**, Bell Laboratories - Lucent Technologies, 2001. 123-125. Disponivel em: <http://www.cs.columbia.edu/~drexel/CandExam/Geolocation_assistedGPS.pdf>. Acesso em: 25 Março 2016.

DUMBILL, E. What is big data? **O'Reilly**, 2016. Disponivel em: <https://www.oreilly.com/ideas/what-is-big-data>. Acesso em: 21 Dezembro 2016.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE. The Scala Programming Language, 2016. Disponivel em: <https://www.scala-lang.org/>. Acesso em: 21 Dezembro 2016.

ESPOSITO, J. **D'Zones 2015 Guide to Big Data, Business Intelligence, and Analytics**, 2015. Disponivel em: <https://dzone.com/storage/assets/332483-dzone-guidetobigdata-2015.pdf>. Acesso em: 22 Março 2016.

FIGUEIREDO, A. Por que a Internet das Coisas será alavanca de Big Data e Analytics. **CIO**, 2015. Disponivel em: <http://cio.com.br/tecnologia/2015/10/09/por-que-a-internet-das-coisas-sera-alavanca-de-big-data-analytics/>. Acesso em: 21 Dezembro 2016.

GARTNER. Hype Cycle for Emerging Technologies, 2015. Disponivel em: <http://www.gartner.com/smarterwithgartner/whats-new-in-gartners-hype-cycle-for-emerging-technologies-2015/>. Acesso em: 01 Maio 2016.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of Things (IoT): A vision, architectural elements, and future directions. **Future Generation Computer Systems 29**, 2013. 1645-1660. Disponivel em: <http://www.sciencedirect.com/science/article/pii/S0167739X13000241>. Acesso em: 02 Maio 2016.

IBM CORPORATION. New expectations for a new era. **CHRO insights from the Global C-suite Study**, 2014. Disponivel em: <http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=XB&infotype=PM&appname=GBSE_GB_TI_USEN&htmlfid=GBE03592USEN&attachment=GBE03592USEN.PDF>. Acesso em: 21 Dezembro 2016.

KONSEK, H. IoT Gateways and Architecture. **D'Zones 2015 Guide to The Internet of Things**, 2015. Disponivel em: <https://dzone.com/storage/assets/162677-dzone-2015-iot-2.pdf>. Acesso em: 15 Março 2016.

KUNTAMUKKALA, A. DZone - Apache Spark - An Engine for Large-Scale Data Processing, 2014. Disponivel em: <https://dzone.com/refcardz/apache-spark>. Acesso em: 10 Maio 2016.

LEONG, L.; PETRI, G.; GILL, B.; DOROSH, M. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide, 2016. Disponivel em: <https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519>. Acesso em: 10 Maio 2016.

LIGHTBEND. Play Framework - Build Modern & Scalable Web Apps with Java and Scala, 2016. Disponivel em: <https://www.playframework.com/>. Acesso em: 21 Dezembro 2016.

NIXON, R. **Learning PHP, MySQL, JavaScript, CSS & HTML5**. 3. ed. Sebastopol, CA: O'Reilly Media, 2014.

NODE.JS FOUNDATION. Express - Node.js web application framework, 2016a. Disponivel em: <http://expressjs.com/>. Acesso em: 21 Dezembro 2016.

NODE.JS FOUNDATION. Node.js, 2016b. Disponivel em: <https://nodejs.org/>. Acesso em: 17 Março 2016.

ORACLE CORPORATION. MySQL, 2016. Disponivel em: <https://www.mysql.com/>. Acesso em: 22 Dezembro 2016.

RUBIN, A. How Apache Spark makes your slow MySQL queries 10x faster (or more). **Percona**, 2016. Disponivel em: <https://www.percona.com/blog/2016/08/17/apache-spark-makes-slow-mysql-queries-10x-faster/>. Acesso em: 03 January 2017.

SMITH, T. Executive Insights on The Internet of Things. **D'Zones 2015 Guide to The Internet of Things**, 2015. Disponivel em: <https://dzone.com/storage/assets/162677-dzone-2015-iot-2.pdf>. Acesso em: 15 Março 2016.

TAURION, C. Entre os Vs do Big data, velocidade cresce em importância. **CIO**, 2014. Disponivel em: <http://cio.com.br/tecnologia/2014/11/25/entre-os-vs-do-big-data-velocidade-cresce-em-importancia/>. Acesso em: 21 Dezembro 2016.

THE APACHE SOFTWARE FOUNDATION. Cassandra, 2016a. Disponivel em: <http://cassandra.apache.org/>. Acesso em: 24 Março 2016.

THE APACHE SOFTWARE FOUNDATION. Spark, 2016b. Disponivel em: <http://spark.apache.org/>. Acesso em: 24 Março 2016.

THE APACHE SOFTWARE FOUNDATION. Storm, 2016c. Disponivel em: <http://storm.apache.org/>. Acesso em: 24 Março 2016.

TOCKER, M. Ten ways to improve the performance of large tables in MySQL. **Master MySQL**, 2013. Disponivel em: <http://www.tocker.ca/2013/10/24/improving-the-performance-of-large-tables-in-mysql.html>. Acesso em: 03 Janeiro 2017.

TRAINING.COM. About Node.js, and why you should add Node.js to your skill set?, 2016. Disponivel em: <http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html>. Acesso em: 27 Dezembro 2016.

WEISER, M.; GOLD, R.; BROWN, S. J. The origins of ubiquitous computing research at PARC in the late 1980s. **IBM Systems Journal**, 38, n. 4, 1999. Disponivel em: <http://www.cs.cmu.edu/~jasonh/courses/ubicomp-sp2007/papers/03-weiser-origins.pdf>. Acesso em: 03 Maio 2016.

ZEIMPEKIS, V.; GIAGLIS, G. M.; LEKAKOS, G. A Taxonomy of Indoor and Outdoor Positioning. **Newsletter ACM SIGecom Exchanges**, Athens University of Economics and Business - Department of Management Science & Technology, Volume 3, n. Issue 4, 2003. 19-27. Disponivel em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.202.2253&rep=rep1&type=pdf>. Acesso em: 22 Março 2016.