Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Bacharelado em Ciência da Computação

Processamento de dados com Apache Spark e Storm: Uma visão sobre e-Participação nas capitais dos estados brasileiros

Felipe Cordeiro Alves Dias

Natal-RN
Junho, 2016

Felipe Cordeiro Alves Dias

Processamento de dados com Apache Spark e Storm: Uma visão sobre e-Participação nas capitais dos estados brasileiros

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Engenharia de Software.

Orientador

Nélio Alessandro Azevedo Cacho, doutor

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Informática e Matemática Aplicada – DIMAp

Natal-RN

Junho, 2016

Monografia de Graduação sob o título *Processamento de dados com Apache Spark e Storm: Uma visão sobre e-Participação nas capitais dos estados brasileiros* apresentada por Felipe Cordeiro Alves Dias e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Titulação e nome do(a) orientador(a)
Orientador(a)
Departamento
Universidade

Titulação e nome do membro da banca examinadora
Co-orientador(a), se houver
Departamento
Universidade

Titulação e nome do membro da banca examinadora
Departamento
Universidade

Titulação e nome do membro da banca examinadora
Departamento
Universidade

Natal-RN, data de aprovação (por extenso).

Agradeço a Deus por todas as oportunidades, pelo apoio da minha amada esposa, Laísa Dias Brito Alves; família e amigos. Todos especialmente importantes para mim.

Agradecimentos

Agradeço a toda a minha família pelo incentivo e apoio recebidos; aos professores do curso de Engenharia de Software com os quais tive aprendizados importantes para a minha vida acadêmica, profissional e pessoal. Também, ao professor Nélio Alessandro Azevedo Cacho, pela orientação, apoio e confiança. No demais, a todos que direta ou indiretamente fizeram parte da minha formação.

Queremos ter certezas e não dúvidas, resultados e não experiências, mas nem mesmo percebemos que as certezas só podem surgir através das dúvidas e os resultados somente através das experiências.

Carl Gustav Jung

Processamento de dados com Apache Spark e Storm: Uma visão sobre e-Participação nas capitais dos estados brasileiros

Autor: Felipe Cordeiro Alves Dias
Orientador: Doutor Nélio Alessandro Azevedo Cacho

Resumo

No contexto de Cidades Inteligentes, um dos desafios é o processamento de grande volumes de dados, em contínua expansão dado o aumento constante de pessoas e objetos conectados à Internet. Nesse cenário, é possível que os cidadãos participem, virtualmente, das questões abordadas por seu respectivo governo local, algo essencial para o desenvolvimento das Cidades Inteligentes e conhecido como e-Participação. Devido a isso, essa monografia analisa o nível de e-Participação existente nas capitais dos estados brasileiros, questionando as já ranqueadas como Inteligentes, utilizando para isso duas das principais ferramentas para processamento de dados: Apache Spark e Apache Storm.

Palavras-chave: Cidades Inteligentes, e-Participação, Processamento de dados.

Data Processing with Apache Spark and Storm: A vision on e-participation in Brazilian State Capitals

Author: Felipe Cordeiro Alves Dias
Advisor: Nélio Alessandro Azevedo Cacho, Doctor

Abstract

In the context of Smart Cities, one of the challenges is the processing of large volumes of data in continuous expansion given the steady increase of people and objects connected to the Internet. In this scenario, it is possible for citizens to participate virtually of issues addressed by its respective local government, which is essential for the development of Smart Cities and known as e-Participation. Because of this, this monograph analyzes the existing level of e-Participation in Brazilian state capitals, questioning already ranked as Smart, using for that two of the main tools for data processing: Spark Apache and Apache Storm.

Keywords: e-Participation, Data Processing, Smart Cities, .

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

UFRN – Universidade Federal do Rio Grande do Norte

DIMAp – Departamento de Informática e Matemática Aplicada

IoT – Internet of Things

ONU – Organização das Nações Unidas

TICs – Tecnologias da Informação e Comunicação

RTC – Real Time Computing

ESP – Event Stream Processing

CEP – Complex Event Processing

POSET – Partially ordered set of events

FIFO – First In, First Out

DStream – Discretized Stream

RDD – Resilient Distributed Dataset

DAG – Directed Acyclic Graph

SNR – Social Network Ratio

Unidade Federativa – UF

YARN – Yet Another Resource Negotiator

Sumário

1 Introdução

Hoje em dia, o crescimento populacional tem sido uma das fontes de stress sob a infraestrutura e recursos de uma cidade (CLARKE, 2013), fenômeno conhecido também como Tragédia dos comuns (HARDIN, 1968), no qual há um cenário de alta demanda por determinados recursos finitos, os quais explorados nessa escala se tornam escassos. Levando isso em consideração, Cidades Inteligentes, são aquelas que, principalmente, usam novas tecnologias para transformar seus sistemas e otimizar o uso de recursos finitos, melhorando a eficiência da economia, possibilitando desenvolvimento político, social, cultural e urbano (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014).

No contexto de Cidades Inteligentes, alguns objetos do nosso cotidiano têm a capacidade de serem conectados à Internet através de eletrônicos embarcados, sensores e software, formando a Internet das Coisas IoT (Internet of Things) . Em 2013, menos de 1% desses dispositivos estavam conectados, sendo a previsão para 2020 de 212 bilhões. Quanto a pessoas conectadas à Internet, prevê-se 3.5 bilhões em 2017, sendo 64% via dispositivos móveis (CLARKE, 2013).

Tal volume de dados, dobrará a cada dois anos até 2020 (EMC, 2014), principalmente devido ao crescimento do uso da Internet, smartphones e Redes Sociais; a queda do custo de equipamento para criar, capturar, administrar, proteger e armazenar informação; migração da TV analógica para a digital; crescimento dos dados gerados por máquinas, incluindo imagens de câmeras de segurança e da informação sobre informação (CLARKE, 2013).

Sendo assim, de acordo com essa expansão, chegar-se-á um patamar em que tudo o que é possível estar, estará conectado, ampliando o conceito de Internet das Coisas para o de Internet de Todas as Coisas, segundo a Cisco (CISCO, 2016). Com essa quantidade de pessoas e dispositivos conectados, cerca de 44 trilhões de gigabytes serão gerados (EMC, 2014), os quais processados por sistemas inteligentes, ajudarão no surgimento de serviços de grande impacto no cotidiano de uma cidade (CLARKE, 2013).

Portanto, umas das principais problemáticas abordadas por Cidades Inteligentes é a de processamento de grande volume dados, provenientes dos sensores instalados em tubulações de água, avenidas (para controle de congestionamentos), iluminações públicas; de câmeras de segurança; da análise de Redes Sociais, como o processamento de tweets (mensagem publicada ou trocada pelos utilizadores da rede social Twitter (INFOPEDIA, 2016)), etc.

Analisando o conteúdo das Redes Socias, por exemplo, é possível explorar o nível de interação entre cidadãos e governos locais no âmbito digital (e-Participação). Sendo essa exploração algo importante por a população ser um componente fundamental das Cidades Inteligentes, que tem como função principal serví-la. Devido a isso, ambas as partes precisam ter uma comunicação ativa (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014).

Apesar dessa importância, poucos trabalhos tem sido direcionados a exploração do pontencial das Redes Sociais para as Cidades Inteligentes (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014), principalmente no que diz respeito a e-Participação (MACIEL; ROQUE; GARCIA, 2009). Além disso, os rankings de Cidades Inteligentes, normalmente, adotam critérios relacionados a transparência e existência de serviços eletrônicos, ignorando o uso de outras ferramentas para Governo Eletrônico, como as Redes Sociais (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014).

1.1 Objetivo

Tendo o que foi exposto anteriormente em consideração, essa monografia tem como objetivo desenvolver uma aplicação para realizar processamento de tweets, coletando métricas relacionadas a e-Participação, das cidades brasileiras ranqueadas, recentemente, como Cidades Inteligentes (Connected Smartcities, 2015), exibindo essas informações numa aplicação web contendo um mapa, buscando obter uma visão do nível de participação, no âmbito digital, entre o governo local e os cidadãos, comparando-a com o ranking mencionado.

O processamento desses tweets, assim como de outros dados, pode ser feito usando, por exemplo, Processamento de Fluxo de Eventos (ESP - Event Stream Processing), processando enventos (acontecimentos do mundo real ou digital) na ordem em que eles chegam, ou, via Processamento Complexo de Eventos (CEP - Complex Event Processing), entendendo como esses eventos se relacionam, dentre outros aspectos, construindo padrões para identificá-los num fluxo de eventos (LUCKHAM, 2006).

A escolha entre essas abordagens depende dos requisitos da aplicação. E, para o objetivo dessa monografia, a abordagem de ESP se mostrou mais adequada, pois os tweets serão processados em ordem. Então, a partir disso, foi desenvolvida uma metodologia para selecionar duas plataformas para ESP, sendo o Apache Spark e Storm escolhidas, principalmente, devido ao número de citações que ambas possuem em artigos e na comunidade de software. Ainda, esse trabalho também tem como proposta comparar as ferramentas citadas, tendo como estudo de caso a aplicação a ser desenvolvida.

1.2 Organização do trabalho

Nesta seção deve ser apresentado como está organizado o trabalho, sendo descrito, portanto, do que trata cada capítulo.

2 Fundamentação Teórica

Nesse capítulo, são apresentados os conceitos relacionados a essa monografia, os quais estão divididos nas seguintes seções: 2.1, sobre a temática de Cidades Inteligentes; 2.2, a respeito de Plataformas de processamento em tempo real; 2.3 e 2.4, sobre o Apache Spark e Storm, respectivamente.

2.1 Cidades Inteligentes

Projeções da Organização das Nações Unidas (ONU) , indicam que a população mundial urbana atual passará de 3.9 bilhões de pessoas para 6.3 bilhões em 2050, representando 66% do total de habitantes, 9.54 bilhões (ONU, 2014). Tal crescimento populacional tem tido consequências graves como a miséria; sendo pertencente a classe de problemas de solução não técnica, de acordo com o fenômeno conhecido por Tragédia dos Comuns, no qual uma alta demanda de recursos finitos (água, energia, alimentos, estradas, etc.) os tornam escassos (HARDIN, 1968).

A ONU, por sua vez, indicam algumas estratégias políticas para lidar com esse problema, tais como: incentivo a migração interna; redistribuição espacial da população; expansão da infraestrutura; garantia de acesso a serviços e oportunidades de trabalho; uso de Tecnologias da Informação e Comunicação (TICs) , para melhoraria na entrega de serviços (ONU, 2014), etc. Podendo nesse último ponto, por exemplo, haver exploração das TICs visando a implantação de iniciativas relacionadas a Cidades Inteligentes.

As Cidades Inteligentes são locais nos quais, principalmente, busca-se transformar os sistemas da cidade e otimizar o uso de seus recursos finitos, melhorando assim a eficiência da economia, possibilitando desenvolvimento político, social, cultural e urbano (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014). Apesar dessas possíveis transformações e otimizações, é importante salientar, novamente, que nenhuma abordagem técnica resolve por si só as problemáticas decorrentes do crescimento populacional (HARDIN, 1968).

Uma das problemáticas que as Cidades Inteligentes ajudam a resolver é a de demanda por eficiência energética (decorrente das mudanças climáticas), que requer das cidades menos emissão de gás carbono, podendo isso ser obtido utilizando iluminação pública, com investimentos em prédios sustentáveis (energeticamente sustentável e com uso de áreas verdes), transportes públicos e ciclovias. Outro possibilidade, é o de desenvolvimento de serviços digitais capazes de ajudar os cidadãos com questões cotidianas, como achar um táxi, saber a localização de um ônibus, encontrar a rota com menos congestionamento de carros, reportar crimes, comunicar-se com o governo, etc.

O problema relacionado a Cidades Inteligentes tratado por essa monografia, como já mencionado, insere-se no contexto de processamento de grande volume de dados, o qual é devido, principalmente a conexão de objetos conectados à Internet (contexto de IoT, conceito explicado em 2.1.1), com previsão de existirem 212 bilhões até 2020. E, também, ao aumento do número de pessoas conectas (o que com o crescimento populacional tende a aumentar), prevendo-se 3.5 bilhões em 2017, sendo 64% via dispositivos móveis (CLARKE, 2013).

Ainda, mais especificamente, espera-se que a quantidade de dados existentes dobre a cada dois anos até 2020 (EMC, 2014). Outros fatores que tem contribuído para isso são: o crescimento do uso da Internet, smartphones e Redes Sociais; a queda do custo de equipamento para criar, capturar, administrar, proteger e armazenar informação; migração da TV analógica para a digital; crescimento dos dados gerados por máquinas, incluindo imagens de câmeras de segurança e da informação sobre informação (CLARKE, 2013).

2.1.1 Internet das Coisas

Considerando o exposto pela seção 2.1, a expansão de pessoas e coisas conectadas à Internet, chegar-se-á um patamar em que tudo o que é possível estar, estará conectado, ampliando o conceito de Internet das Coisas para o de Internet de Todas as Coisas, segundo a Cisco (CISCO, 2016). Em tal cenário, cerca de 44 trilhões de gigabytes serão gerados (EMC, 2014), os quais processados por sistemas inteligentes, ajudarão no surgimento de serviços de grande impacto no cotidiano de uma cidade (CLARKE, 2013).

Principalmente, aqueles que monitoram e reportam continuadamente qualquer modificação que ocorra em um determinado cenário. Por exemplo, no contexto de saúde pública, é possível monitorar as condições do organismo de um paciente, notificando os médicos responsáveis caso a pessoa esteja em risco, através de uma rápida comunicação e resposta a tal acontecimento. Alguns dos sistemas semelhantes a esse são exemplificados na Fig. 1.

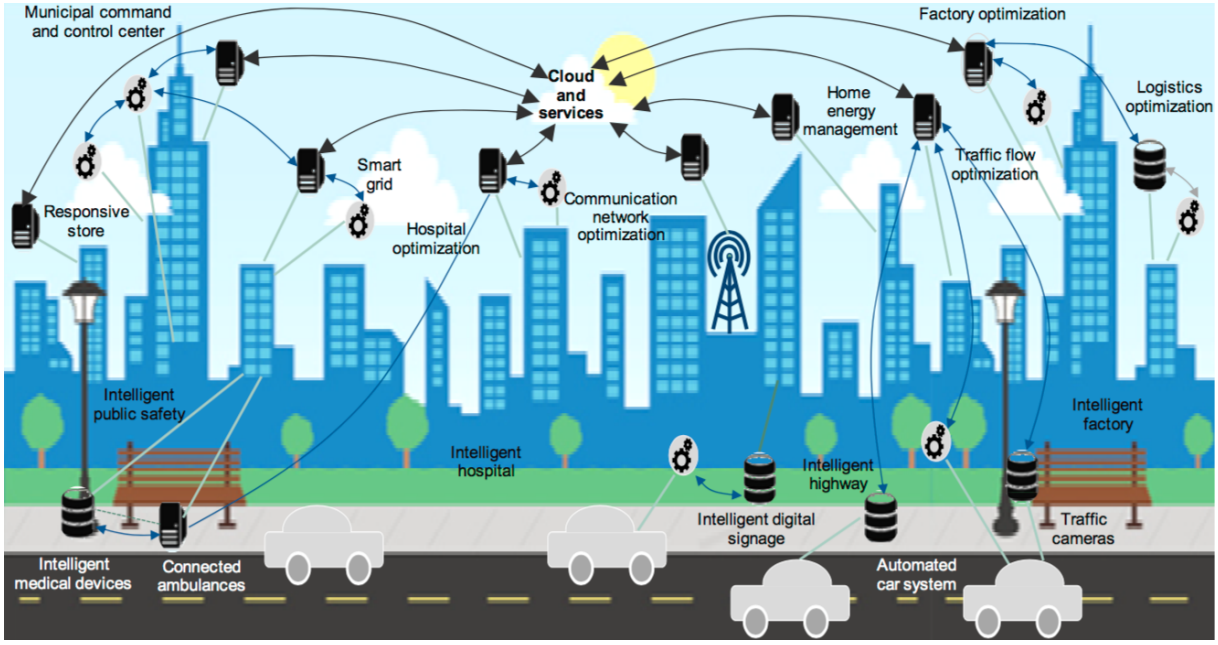

Figura 1: Cidade num invólucro digital
Fonte: (CLARKE, 2013)

Na Fig. 1, organizações, pessoas e objetos estão conectados, criando um invólucro digital, composto por representações de serviços conectados a nuvem, tais como: o de um (i) Hospital Inteligente, conectado a dispositivos médicos inteligentes e a ambulâncias; (ii) Centro Digital de Controle e Comando Municipal, integrado a um sistema inteligente de segurança pública e armazenamento de dados; (iii) Sistema de Otimização de Fluxo de Tráfico, conectado a um sistema de estradas inteligentes agregadas a câmeras de controle de tráfico; (iv) Fábrica Inteligente, integrada a um sistema de otimização de logística e fábrica; (v) Administração de Energia Doméstica e (vi) Smart Grid (um novo modelo de Redes Elétricas (H. et al., 2016)) e (vii) Otimização de Rede de Comunicação.

Tais serviços não serão estudados nesse trabalho, foram apenas referenciados visando demonstrar uma pequena parte da gama de possibilidades existentes em Cidades Inteligentes, Internet das Coisas, e a problemática relacionada ao processamento de dados.

2.1.2 Requisitos de uma aplicação de Cidade Inteligente

Ante a variedade de fontes de dados disponíveis, conforme mencionado na seção 2.1.1, a capacidade de processar grandes volumes de dados pode ser um dos requisitos necessários a uma aplicação de Cidade Inteligente. Normalmente, duas métricas são utilizadas para analisar esse requisito: os valores de (i) throughput e (ii) latência. O primeiro termo se refere a taxa de processamento (Techopedia, 2016) e, o segundo a variação do tempo entre

um estímulo e resposta (Techopedia, 2016).

Dependendo do contexto da aplicação, o valor da latência tem maior relevância sob o de throughput. Por exemplo, a latência talvez seja mais interessante de ser priorizada quando respostas a determinados eventos precisam ser no menor intervalo de tempo possível (menos de um segundo). Por outro lado, as aplicações que priorizam throughput, podem apenas necessitarem processar grandes volumes de dados dentro de um valor de latência aceitável (no máximo alguns segundos).

Outro requisito importante, é o de tolerância a falhas (necessário quando a aplicação está inserida em ambientes distribuídos, ou, de incertezas), o qual permite o funcionamento do sistema mesmo que uma falha ocorra, o que pode ser obtido, por exemplo, por meio de replicação (COULOURIS et al., 2013). Além disso, pode existir a necessidade de antender o requisito de escalabilidade, através do qual é possível oferecer um serviço de alta qualidade conforme o aumento da demanda, adicionando novos recursos (escalabilidade horizontal), ou, melhorando os existentes (escalabilidade vertical) (SOMMERVILLE, 2011).

Além disso, a aplicação pode precisar do requisito de processamento em tempo real, no qual há um limite de tempo pré-determinado para as respostas aos eventos do sistema (SOMMERVILLE, 2011). Por fim, é importante também avaliar as ferramentas que serão utilizadas no desenvolvimento da aplicação quanto ao suporte aos requisitos citados e as abstrações que elas dispõem.

2.1.3 e-Participação

Uma das temáticas abordadas em Cidades Inteligentes, é a de promover novos meios para a participação do cidadão com questões relaciondas a gestão da cidade (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014). Essa interação pode ser feita através de Redes Sociais, que são compostas por um conjunto de pessoas (ou, organizações), representando nós de uma rede, sendo a conexão entre eles conhecida como relacionamento (MACIEL; ROQUE; GARCIA, 2009).

Sendo assim, tal ambiente virtual, tem proporcionado um novo espaço para que os cidadãos possam participar de processos de consulta e deliberação (exame e dicussão de um assunto (PRIBERAM, 2016)), atuando com os governos como atores de processos de tomadas de decisão (MACIEL; ROQUE; GARCIA, 2009). Além das Redes Sociais, a participação pode ocorrer em outros ambientes virtuais, tais como aplicativos, wiki, fórum, blogs, dentre outros (MAGALHÃES, 2015), resumindo-se em e-participação, conceito inserido no

de Governo Eletrônico.

O Governo Eletrônico é caracterizado pelo uso de TICs pelo governo público, buscando prover melhores serviços, informações e conhecimento ao cidadão; facitando o acesso ao processo político e incentivando a participação (MAGALHÃES, 2015). Ele pode ser dividido, principalmente, nos três campos seguintes: e-Administração, a respeito do funcionamento interno do poder público; e-Governo, no tocante a entrega e fornecimento de serviços e informações qualificadas aos cidadãos; e-Democracia, relacionada a ampliação da participação da sociedade na tomada de decisão, sendo a e-Participação uma subárea desse último (MAGALHÃES, 2015).

A intenção da e-Participação é reforçar e renovar as interações entre o setor público, os políticos e cidadãos; tanto quanto possível no processo democrático (MAGALHÃES, 2015). Além disso, a e-Participação não pode ser avaliada somente por seus aspectos técnicos, mas também quanto a capacidade de incrementar a democracia (MAGALHÃES, 2015).

Um dos desafios nessa área é avaliar as ferramentas apoiadas pelas TICs, quanto as formas de engajamento existentes, como informação, consulta, ou, participação ativa (MAGALHÃES, 2015). Ainda, segundo citação contida em (MAGALHÃES, 2015), a e-Participação é um conjunto de várias tecnologias, medidas sociais e políticas, havendo a necessidade de melhorar a compreensão das relações entre esses componentes e como suas respectivas práticas de avaliação podem ser aplicadas a e-Particip̧ão como um todo.

Com isso, para que os processos que a envolvem sejam eficazes é importante considerar os ambientes online e off-line, ou seja, incrementando os métodos tradicionais (conferências, fóruns, conselhos, ouvidorias, audiências, consultas, reuniões, comitês, grupos de trabalho e mesas de negociação) com as possibilidades da e-Partição, estendendo-a ainda aos grupos desfavorecidos e desconectados (MAGALHÃES, 2015). Por fim, no parágrafo seguinte, são referenciados alguns dos diferentes níveis de e-Participação.

Níveis de e-Participação. No nível e-Informação, há um canal unidirecional, visando apenas fornecer informações de interesse cívico; no de e-Consulta, a comunicação é bidirecional, via coleta de opiniões e alternativas; no de e-Envolvimento busca-se garantir a compreensão e levar em consideração os anseios do cidadão; em e-Colaboração, a comunicação é bidirecional, e o cidadão participa ativamente no desenvolvimento de alternativas e indentificação de melhores soluções; por fim, no de e-Empoderamento, a influência, controle e elaboração de políticas pelo e para o público é viabilizada (MAGALHÃES, 2015).

2.2 Plataformas de processamento em tempo real

As plataformas de processamento em tempo real (RTC, Real Time Computing) , compostas por hardware ou software, são sistemas que tem como um dos principais requisitos emitir respostas a eventos de acordo com um determinado deadline (limite de tempo). Sendo assim, a corretude da computação não depende apenas da corretude lógica, mas também dos resultados serem produzidos de acordo com o deadline especificado (SHIN; RAMANATHAN, 1994), (SOMMERVILLE, 2011).

Normalmente, os resultados são obtidos através de processamentos realizados por um conjunto de tasks (tarefas) cooperativas, iniciadas por eventos do sistema. Os eventos são dependentes do ambiente no qual estão inseridos, podendo ocorrer periodicamente (de forma regular e previsível), ou, aperiodicamente (irregulares e imprevisíveis) (SHIN; RAMANATHAN, 1994), (SOMMERVILLE, 2011).

As tasks podem ter uma relação de interdependência e ao mesmo tempo nem todos os eventos que as originaram necessitarem de ser processados dentro de um deadline. Apesar disso, nenhuma task pode vir a comprometer o processamento de outra (SHIN; RAMANATHAN, 1994).

Com isso, os deadlines podem ser classificados em hard, firm, ou, soft. No primeiro caso, respectivamente, as respostas a todos os eventos devem necessariamente ocorrer dentro do deadline definido; no segundo, deadlines esporadicamente não atendidos são aceitos; no terceiro, deadlines não alcançados são permitidos frequentemente (SHIN; RAMANATHAN, 1994). Além desses categorias, há os sistemas com delay (atraso) introduzido entre o intervalo de tempo de estímulo e resposta, os quais são classificados como near real time (NRT), ou seja, são sistemas de processamento em "quase tempo real".

Sendo assim, na categoria hard, é considerado como falha se o tempo estimado para um processamento não for atendido, e nas demais, a ocorrência disso resulta numa degradação (contínua de acordo com a quantidade de deadlines não atendidos) (SHIN; RAMANATHAN, 1994), (SOMMERVILLE, 2011).

Entende-se por falha quando o usuário não recebe algo esperado (por exemplo, deadline não atendido) devido a um erro de sistema. Sendo esse erro um estado errôneo resultante de um defeito (característica do sistema que pode resultar em erro). E, degradação, como decréscimo da qualidade (conceito subjetivo, de acordo com os requisitos da aplicação) do sistema (SOMMERVILLE, 2011).

Por fim, é importante mencionar também que uma falha de sistema pode comprometer o requisito de Confiança e prejudicar um grande número de pessoas (SHIN; RAMANATHAN, 1994), (SOMMERVILLE, 2011). Na Fig. 2 a seguir, são ilustradas as principais propriedades desse requisito:

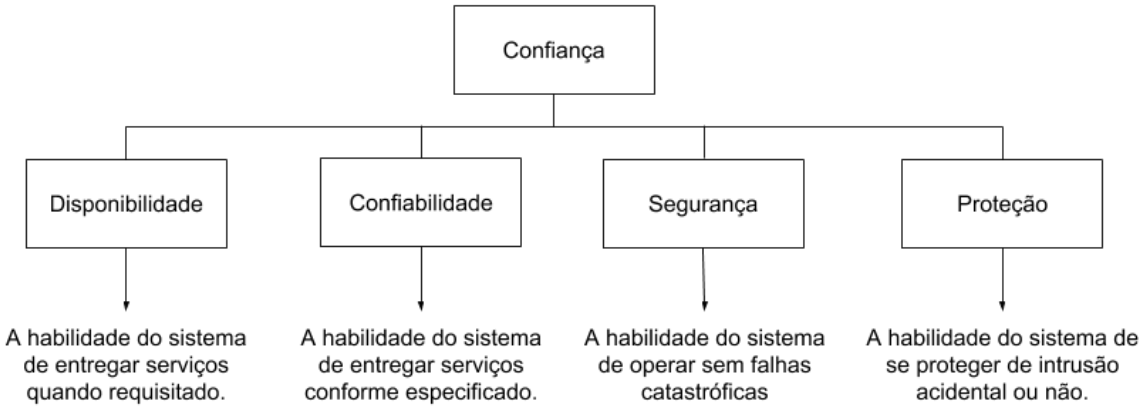

Figura 2: Principais propriedades de Confiança
Fonte: (SOMMERVILLE, 2011)

2.2.1 Processamento de fluxo de eventos em tempo real

Fluxo (ou stream) de eventos, basicamente, pode ser entendido como uma série de eventos em função do tempo (NARSUDE, 2015), (LUCKHAM, 2006). Dentro desse escopo, duas abordagens de processamento são importantes diferenciar: (i) ESP (Event Stream Processing, ou, Processamento de Fluxo de Eventos) e (ii) CEP (Complex Event Processing, ou, Processamento Complexo de Eventos) . A primeira, costuma-se focar principalmente com questões de baixo nível, relacionadas a como processar eventos em tempo real atendendo requisitos de escalabilidade, tolerância a falha, confiabilidade, etc (KLEPPMANN, 2015).

Na segunda abordagem, os streams são utilizados para criar uma nuvem de eventos, parcialmente ordenados por tempo e causalidade, conceito conhecido como POSET (Partially ordered set of events) (LUCKHAM, 2006). Sendo assim, normalmente, visa-se trabalhar questões de alto nível, utilizando posets para a criação de padrões de eventos, envolvendo seus relacionamentos, estrutura interna, etc. Com esse conjunto de informações é possível compor eventos complexos, os quais podem ser consultados continuadamente (KLEPPMANN, 2015).

Portanto, nessa monografia, será utilizada a primeira abordagem (ESP) para proces-

samento de eventos, por ser mais adequada ao objetivo proposto no Cap. 1. Em ESPs, algumas plataformas processam os streams continuadamente conforme são recebidos pelo sistema; paradigma conhecido como One at time. Quando um evento é processado com sucesso, há a possibilidade de emitir uma notificação sobre isso, a qual é custosa por ser necessário rastreá-lo até o término de seu processamento (NARSUDE, 2015).

Outra alternativa, é a de realizar o processamento em micro-batchs (pequenos lotes) de eventos, tendo com uma das vantagens poder realizar operações em pequenos blocos de dados, em vez de individualmente, ao custo de introduzir um pequeno atraso no processo (NARSUDE, 2015).

2.3 Apache Spark

Apache Spark é uma plataforma de computação em cluster (unidade lógica composta por um conjunto de computadores conectados entre si através da Rede, permitindo compartilhamento de recursos (Techopedia, 2016)), com suporte a consultas a banco de dados, processamento de streaming (em near real time), grafos, e aprendizado de máquina. Tendo Java, Python e Scala como linguagens de programação suportadas (Apache Software Foundation, 2016g).

A arquitetura do Spark, conforme a Fig. 3, é composta por uma pilha integrando os seguintes componentes, a serem explicados posteriormente: Spark SQL, Spark Streaming, MLib, GraphX, Spark Core e Administradores de Cluster (Yarn (Apache Software Foundation, 2016c) e Apache Mesos (Apache Software Foundation, 2016d)). Tal estrutura visa ser de fácil manutenção, teste, deploy, e permitir aumento de perfomance do núcleo impactando seus demais componentes.

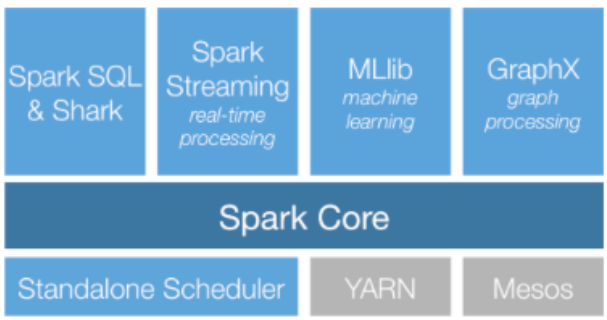

Figura 3: Ilustração da arquitetura em pilha do Apache Spark
Fonte: (KARAU et al., 2015)

2.3.1 Módulos do Apache Spark

O Spark Core é o principal módulo do Apache Spark, responsável principalmente pelo gerenciamento de memória, tasks (conceito explicado em 2.3.2.4) e tolerância a falha. Ainda nele, a abstração conhecida como RDD é definida, cujo papel é o de representar uma coleção de dados distribuídos e manipulados em paralelo. Os RDDs em Java são representados pela classe JavaRDD, criados através da classe JavaSparkContext (contexto da aplicação), que é instanciada recebendo como parâmetro um objeto SparkConf, contendo informações relacionadas ao nome da aplicação e o endereço em que ela será executada, pondendo ser local, ou, em cluster.

Outro módulo é o Spark Streaming, o qual realiza o processamento de dados em stream. Os streams são representados por uma sequência de RDDs, conhecida como DStream. O contexto de um streaming (JavaStreamingContext) é criado usando as configurações da aplicação e o intervalo no qual cada DStream será definido. Após isso, tais streams são atribuídos a um objeto da classe JavaReceiverInputDStream.

Além dos módulos detalhados nos parágrafos anteriores, é relevante mencionar o (i) Spark SQL, responsável por trabalhar com dados estruturados; o (ii) MLib, relacionado ao aprendizado de máquina, contendo algorítimos tais como o de classificação, regressão, "clusterização", filtragem colaborativa, avaliação de modelo e importação de dados; e o (ii) GraphX, que é composto por uma biblioteca para manipulação de grafos e computações paralelas. Por fim, o Spark também possui um administrador de cluster padrão, conhecido como Standalone Scheduler (Apache Software Foundation, 2016g), tendo também suporte ao YARN e Apache Mesos.

2.3.2 Composição Interna do Apache Spark

Nesta seção é explicada a composição interna do Apache Spark, expondo os componentes que permitem a execução distribuída de uma aplicação spark, tais como o Driver Program, Spark Context, Worker Node e Executor. Também são explicados os conceitos sobre as operações de Transformação e Ação, e os relativos a Job, Stage, Task e Partição RDD.

2.3.2.1 Modelo de execução das aplicações Spark

O modelo de execução das aplicações Spark é definido pela relação composta por um driver, SparkContext, executors e tasks, conforme ilustrado na Fig. 4. Nessa interação, as aplicações Spark funcionam num Worker Node (qualquer nó capaz de de executar código de aplicação localmente ou num cluster) como uma espécie de Driver, o qual executa operações paralelas e define dados distribuídos sobre um cluster. Além disso, o driver é responsável por encapsular a função principal da aplicação e prover acesso ao Spark, utilizando o objeto da classe SparkContext. Tal objeto é usado também para representar uma conexão com um cluster e construir RDDs (KARAU et al., 2015).

Após a construção de um RDD, é possível executar operações sobre ele. Essas operações são realizadas por executors, processos administrados por uma aplicação Spark (cada aplicação tem os seus próprios executors); nos quais há espaços reservados para execução de tasks (conceito explicado na seção 2.3.2.4). Há dois tipos de operações: (i) Ações (actions) e (ii) Transformações (transformations) (Apache Software Foundation, 2016g). O primeiro tipo retorna um resultado ao driver após computações sobre um conjunto de dados de acordo com uma função. Sendo o segundo, responsável por gerar novos conjuntos de dados (RDDs) através, também, de funções especificadas (KARAU et al., 2015).

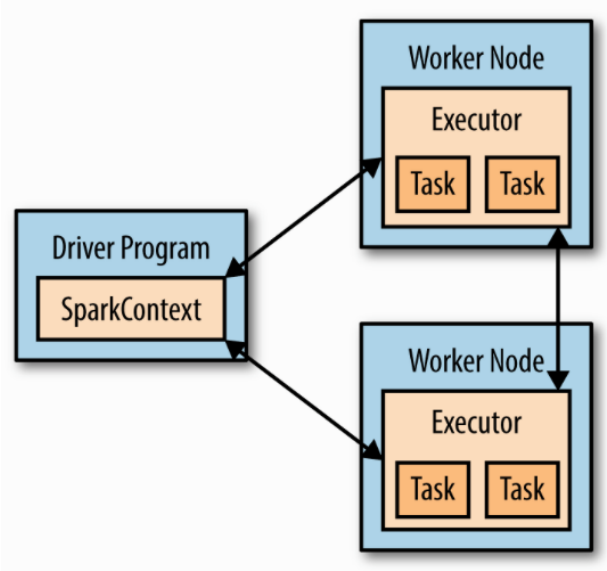

Figura 4: Componentes do Spark para execução distribuída
Fonte: (KARAU et al., 2015)

2.3.2.2 Job

A abstração do conceito de job está relacionada com o de action. Mais especificamente, um job é o conjunto de estágios (stages) resultantes de uma determinada action. Por exemplo, quando o método count() é invocado (para realizar a contagem de elementos dentro de um RDD), cria-se um job composto por um ou mais estágios. Os jobs, por padrão, são escalonados para serem executados em ordem FIFO (First In, First Out) . Além disso, o scheduler do Spark é responsável por criar um plano de execução física, o qual é utilizado para calcular as RDDs necessárias para executar a ação invocada (KARAU et al., 2015), (XU; JU; REN, 2015).

Em tal plano de execução física, defini-se basicamente um grafo acíclico dirigido (DAG), relacionando as dependências existentes entre os RDDs, numa espécie de linhagem de execução (lineage). Após isso, a partir da última RDD do último estágio, cada dependência é calculada, de trás para frente, para que posteriormente os RDDs dependentes possam ser calculados, até que todas as actions necessárias tenham terminado (KARAU et al., 2015), (XU; JU; REN, 2015).

2.3.2.3 Stage

A divisão de um job resulta no conceito de stage, ou seja, jobs são compostos por um ou mais stages, os quais têm tasks a serem executadas. Nem sempre a relação entre stages e RDDs será de 1:1. No mais simples dos casos, para cada RDD há um stage, sendo possível nos mais complexos haver mais de um stage gerado por um único RDD (KARAU et al., 2015), (XU; JU; REN, 2015).

Por exemplo, havendo três RDDs no grafo RDD, não necessariamente haverão três stages. Um dos motivos para isso acontecer é quando ocorre pipelining, situação na qual é possível reduzir o número de stages, calculando os valores de alguns RDDs utilizando unicamente informac?o?es de seus antecessores, sem movimentação de dados de outros RDDs (KARAU et al., 2015), (XU; JU; REN, 2015).

Além da possibilidade do número de stages ser reduzido, o contrário acontece quando fronteiras entre stages são definidas. Tal situação ocorre porque algumas transformações, como a reduceByKey (função de agregação por chaves), podem resultar em reparticionamento do conjunto de dados para a computação de alguma saída, exigindo dados de outros RDDs para a formação de um único RDD. Assim, em cada fronteira entre os stages, tasks do estágio antecessor são responsáveis por escrever dados para o disco e buscar

tasks no estágio sucessor, através da rede, resultando em um processo custoso, conhecido como shuffling (KARAU et al., 2015), (XU; JU; REN, 2015).

O número de partições entre o estágio predecessor e sucessor podem ser diferentes, sendo possível customizá-lo, embora isso não seja recomendável. Tal aconselhamento é devido ao fato de que se o número for modificado para uma pequena quantidade de partições, tais podem sofrer com tasks overloaded (sobrecarregadas), do contrário, havendo partições em excesso, resultará em um alto número de shuffling entre elas (KARAU et al., 2015), (XU; JU; REN, 2015).

Em resumo, shuffling sempre acontecerá quando for necessário obter dados de outras partições para computar um resultado, sendo esse um procedimento custoso devido ao número de operações de entrada e saída envolvendo: leitura/escrita em disco e transferência de dados através da rede. No entanto, se necessário, o Spark provê uma forma eficiente para reparticionamento, conhecida como coalesce(), a qual reduz o número de partições sem causar movimentação de dados (KARAU et al., 2015).

2.3.2.4 Task

A abstração do conceito task está relacionada principalmente com stage, pois, uma vez que os estágios são definidos, tasks são criadas e enviadas para um scheduler interno. Além disso, tasks estão muito próximas da definição de partition, pelo fato de ser necessária a existe?ncia de uma task para cada partition, para executar as computações necessárias sobre ela (KARAU et al., 2015).

Portanto, o número de tasks em um stage é definido pelo número de partições existentes no RDD antecessor. E, por outro lado, o número de partições em um determinado RDD é igual ao número de partições existentes no RDD do qual ele depende. No entanto, há exceções em caso de (i) coalescing, processo o qual permite criar um RDD com menos partições que seu antecessor; (ii) união, sendo o número de partições resultado da soma do número de partições predecessoras; (iii) produto cartesiano, produto dos predecessores (KARAU et al., 2015).

Cada task internamente é divida em algumas etapas, sendo elas: (i) carregamento dos dados de entrada, sejam eles provenientes de unidades de armazenamento (caso o RDD seja um RDD de entrada de dados), de um RDD existente (armazenado em memória cache), ou, de saídas de outro RDD (processo de shuffling), (ii) processamento da operação necessária para computação do RDD representado por ela, (iii) escrever a saída para o

processo de shuffling, para um armazenamento externo ou para o driver (caso seja o RDD final de uma ação). Em resumo, uma task é responsável por coletar os dados de entrada, processá-los e, por fim, gerar uma saída (KARAU et al., 2015).

2.4 Apache Storm

Apache Storm (Apache Software Foundation, 2016h) é uma plataforma de computação em cluster, com suporte a processamento de stream de dados em tempo real. O Storm requer pouca manutenção, é de fácil de administração e tem suporte a qualquer linguagem de programação (que leia e escreva fluxos de dados padronizados), posto que em seu núcleo é utilizado o Apache Thrift (Apache Software Foundation, 2016i) (framework que permite o desenvolvimento de serviços multilinguagem) para definir e submeter topologias (conceito definido em 2.4.1.4) (Apache Software Foundation, 2016h).

A arquitetura do Storm, conforme a Fig. 5, é composta pelos seguintes componentes, a serem explicados posteriormente: Nimbus, ZooKeeper e Supervisor Node.

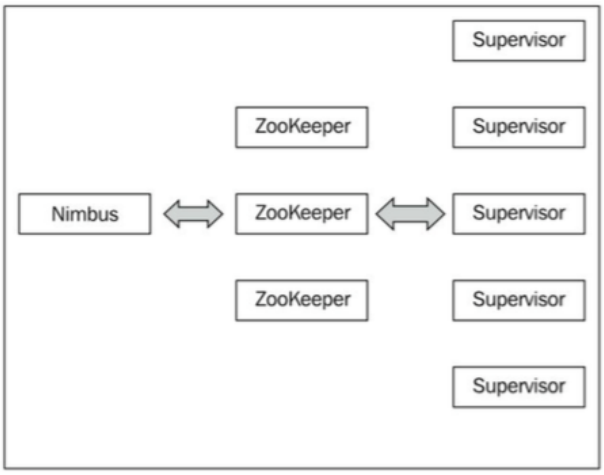

Figura 5: Ilustração da arquitetura do Apache Storm
Fonte: (JAIN; NALYA, 2014)

2.4.1 Composição interna do Apache Storm

Nesta seção é explicada a composição interna do Apache Storm, expondo os principais componentes que permitem a execução distribuída de uma topologia Storm, tais como o Nimbus, ZooKeeper e Supervisor nodes.

2.4.1.1 Nimbus

Nimbus é o processo master executado num cluster Storm, tendo ele uma única instância e sendo responsável por distribuir o código da aplicação (job) para os nós workers (computadores que compõem o cluster), aos quais são atribuídas tasks (parte de um job). As tasks são monitoradas pelo Supervisor node, sendo reiniciadas em caso de falha e quando requisitado. Ainda, caso um Supervisor node falhe continuadamente, o job é atribuído para outro nó worker (JAIN; NALYA, 2014), (HART; BHATNAGAR, 2015).

O Nimbus utiliza um protocolo de comunicação sem estado (Stateless protocol), ou seja, cada requisição é tratada individualmente, sem relação com a anterior, não armazenando dados ou estado (Techopedia, 2016), sendo assim todos os seus dados são armazenados no ZooKeeper (definido em 2.4.1.3). Além disso ele é projetado para ser fail-fast, ou seja, em caso de falha é rapidamente reiniciado, sem afetar as tasks em execução nos nós workers (JAIN; NALYA, 2014), (HART; BHATNAGAR, 2015).

2.4.1.2 Supervisor node

Cada Supervisor node é responsável pela administração de um determinado nó do cluster Storm; gerenciando o ciclo de vida de processos workers relacionados a execução das tasks (partes de uma topologia) atribuídas a ele próprio. Os workers quando em execução emitem "sinais de vida"(heartbeats), possibilitando o supervisor detectar e reiniciar caso não estejam respondendo. E, assim como o Nimbus, um Supervisor node é projetado para ser fail-fast (JAIN; NALYA, 2014), (HART; BHATNAGAR, 2015).

2.4.1.3 ZooKeeper

O ZooKeeper Cluster (Apache Software Foundation, 2016j) é responsável por coordenar processos, informações compartilhadas, tasks submetidas ao Storm e estados associados ao cluster, num contexto distribuído e de forma confiável, sendo possível aumentar o nível de confiabilidade tendo mais de um servidor ZooKeeper. O ZooKeeper atua também como o intermediário da comunicação entre Nimbus e os Supervisor nodes, pois eles não se comunicam diretamente entre si, ambos também podem ser finalizados sem afetar o cluster, visto que todos os seus dados, são armazenados no ZooKeeper, como já mencionado anteriormente (JAIN; NALYA, 2014), (HART; BHATNAGAR, 2015).

2.4.1.4 Topologia

A topologia Storm, ilustrada na Fig. 6, é uma abstração que define um grafo acíclico dirigido (DAG), utilizado para computações de streams. Cada nó desse grafo é responsável por executar algum processamento, enviando seu resultado para o próximo nó do fluxo. Após definida a topologia, ela pode ser executada localmente ou submetida a um cluster (JAIN; NALYA, 2014), (HART; BHATNAGAR, 2015).

Stream é um dos componentes da topologia Storm, definido como uma sequência de tuplas independentes e imutáveis, fornecidas por um ou mais spout e processadas por um ou mais bolt. Cada stream tem o seu respectivo ID, que é utilizado pelos bolts para consumirem e produzirem as tuplas. As tuplas compõem uma unidade básica de dados, suportando os tipos de dados (serializáveis) no qual a topologia está sendo desenvolvida.

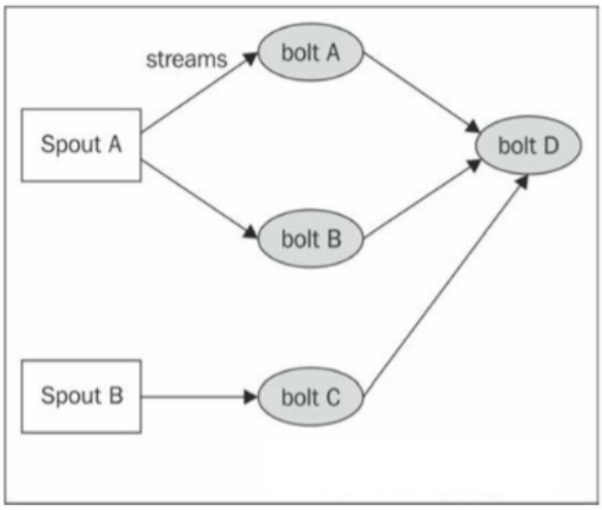

Figura 6: Ilustração de uma topologia do Storm
Fonte: (JAIN; NALYA, 2014)

Bolt. Os bolts são unidades de processamento de dados (streams), sendo eles responsáveis por executar transformações simples sobre as tuplas, as quais são combinadas com outras formando complexas transformações. Também, eles podem ser inscritos para receberem informações tanto de outros bolts, quanto dos spouts, além de produzirem streams como saída (JAIN; NALYA, 2014).

Tais streams são declarados, emitidos para outro bolt e processados, respectivamente, pelos métodos declareStream, emit e execute. As tuplas não precisam ser processadas imediatamente quando chegam a um bolt, pois, talvez haja a necessidade de aguardar para executar alguma operação de junção (join) com outra tupla, por exemplo. Também,

inúmeros métodos definidos pela interface Tuple podem recuperar metadados associados com a tupla recebida via execute (JAIN; NALYA, 2014).

Por exemplo, se um ID de mensagem está associado com uma tupla, o método execute deverá publicar um evento ack, ou, fail, para o bolt, caso contrário o Storm não tem como saber se uma tupla foi processada. Sendo assim, o bolt envia automaticamente uma mensagem de confirmação após o término da execução do método execute e, em caso de um evento de falha, é lançada uma exceção (JAIN; NALYA, 2014).

Além disso, num contexto distribuído, a topologia pode ser serializada e submetida, via Nimbus, para um cluster. No qual, será processada por worker nodes. Nesse contexto, o método prepare pode ser utilizado para assegurar que o bolt está, após a desserialização, configurado corretamente para executar as tuplas (JAIN; NALYA, 2014), (HART; BHATNAGAR, 2015).

Spout. Na topologia Storm um Spout é o responsável pelo fornecimento de tuplas, as quais são lidas e escritas por ele utilizando uma fonte externa de dados (JAIN; NALYA, 2014).

As tuplas emitidas por um spout são rastreadas pelo Storm até terminarem o processamento, sendo emitida uma confirmação ao término dele. Tal procedimento somente ocorre se um ID de mensagem foi gerado na emissão da tupla e, caso esse ID tenha sido definido como nulo, o rastreamento não irá acontecer (JAIN; NALYA, 2014).

Outra opção é definir um timeout a topologia, sendo dessa forma enviada uma mensagem de falha para o spout, caso a tupla não seja processada dentro do tempo estipulado. Sendo necessário, novamente, definir um ID de mensagem. O custo dessa confirmação do processamento ter sido executado com sucesso é a pequena perda de performance, que pode ser relevante em determinados contextos; sendo assim é possível ignorar a emissão de IDs de mensagens (JAIN; NALYA, 2014).

Os principais métodos de um spout são: o open(), executado quando o spout é inicializado, sendo nele definida a lógica para acesso a dados de fontes externas; o de declaração de stream (declareStream) e o de processamento de próxima tupla (nextTuple), que ocorre se houver confirmação de sucesso da tupla anterior (ack), sendo o método fail chamado pelo Storm caso isso não ocorra (JAIN; NALYA, 2014).

Por fim, nenhum dos métodos utilizados para a construção de um spout devem ser bloqueante, pois o Storm executa todos os métodos numa mesma thread. Além disso, todo

spout tem um buffer interno para o rastreamento dos status das tuplas emitidas, as quais são mantidas nele até uma confirmação de processamento concluído com sucesso (ack) ou falha (fail); não sendo inseridas novas tuplas (via nextTuple) no buffer quando ele esá cheio (JAIN; NALYA, 2014).

3 Aplicações desenvolvidas e estudo comparativo

Nesse capítulo, a seção 3.1 se refere a metodologia de escolha das plataformas para processamento de fluxo de eventos em tempo real, utilizadas pelas aplicações desenvolvidas, explicadas na seção 3.2. Por último, na seção 3.3, é realizado um estudo comparativo das ferramentas ESPs escolhidas anteriormente.

3.1 Metodologia de escolha das plataformas para processamento de fluxo de eventos em tempo real

Inicialmente, foi realizada uma pesquisa com a palavra chave "stream processing"e analisados os primeiros artigos (na ordem do resultado e que citavam mais de uma ferramenta) em busca de citações a ferramentas (open source) de processamento de streams em tempo real. Com base nesse resultado, os nomes das plataformas encontradas foram utilizados para buscar a quantidade de referências no Google Scholar (site para publicação de artigos acadêmicos) e Stack Over Flow (site para discussão de assuntos relacionados a desenvolvimento de software). Na Tab. 1, constam os resultados desse procedimento.

A plataforma Esper mencionada em (WÄHNER, 2014) e (KLEPPMANN, 2015). foi descartada do processo de escolha por ser um componente para CEP, normalmente integrado a ferramentas ESPs, fugindo do escopo desse trabalho (ESPERTECH, 2016). Sendo assim, continuando a análise, foram também consideradas as características das opções restantes, estudadas superficialmente nos parágrafos seguintes.

Apache Spark Streaming. Apache Spark Streaming é uma extensão do Spark para processamento de stream, em near real time. Além disso, o Spark possui outros módulos para consultas a banco de dados, grafos, e aprendizado de máquina (Apache Software Foundation, 2016g). O processamento de stream é realizado em micro-batching pelo Spark

Streaming (um de seus módulos), utilizando uma abstração conhecida como DStream , a qual é composta por uma sequências de RDDs (Resilient Distributed Dataset) , ilustrada na Fig. 7. RDDs abstraem uma coleção de dados distribuídos e manipulados em paralelo. No Spark, também é possível utilizar processamento em batch, independente do módulo que está sendo utilizado (KARAU et al., 2015).

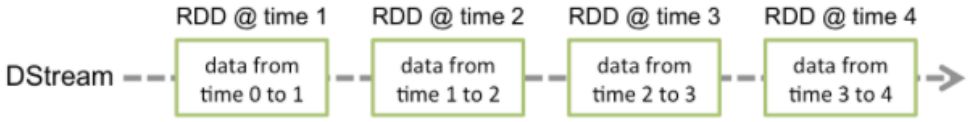

Figura 7: Ilustração de um DStream
Fonte: (Apache Software Foundation, 2016g)

Apache Flink. Apache Flink é um plataforma muito semelhante ao Spark. Com suporte a processamento de streams, os quais são definidos por uma abstração conhecida como DataStream, ilustrada na Fig. 8. O processamento em batch é realizado em cima de DataSets, semalhantes as RDDs do Spak. Da mesma forma como o DStream é definido como uma série de RDDS, o DataStream é composto por uma sequência de DataSets. A abstração Sink é utilizada para armazenar e retornar DataSets. Há suporte também para CEP, processamento em grafos, aprendizado de máquina e consulta a banco de dados (Apache Software Foundation, 2016b).

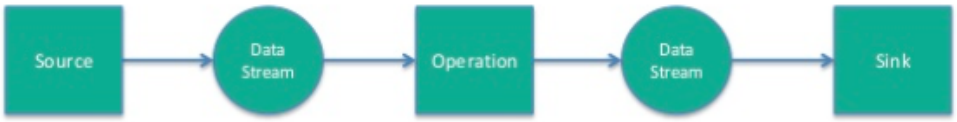

Figura 8: Ilustração de um fluxo de processamento no Flink
Fonte: (DATAARTISANS, 2015)

Apache Storm. Apache Storm é um plataforma para processamento de stream em tempo real. Ao contrário do Spark, tem "one at time"como modelo de processamento de dados. As streams são compostas por tuplas (unidade básica de dados, as quais processadas numa abstração conhecida como Topologia, ilustrada na Fig. 6, composta por um DAG (Directed Acyclic Graph (RUOHONEN, 2013)) de spouts e bolts, respectivamente, resposáveis por emitir streams de dados e processá-los (JAIN; NALYA, 2014).

Apache Samza. Apache Samza é outra plataforma para processamento de stream em tempo real, utilizando como abstração base o conceito de mensagem (identificadas por off-

sets, ids), em vez de tupla, ou, DStream. Os streams são separados em partições contendo mensagens ordenadas (acessadas apenas no modo de leitura), sendo processados por jobs responsáveis por ler e emir fluxos. Assim como o Spark, há suporte para processamento em batch, o qual é realizado processando sequências de streams.

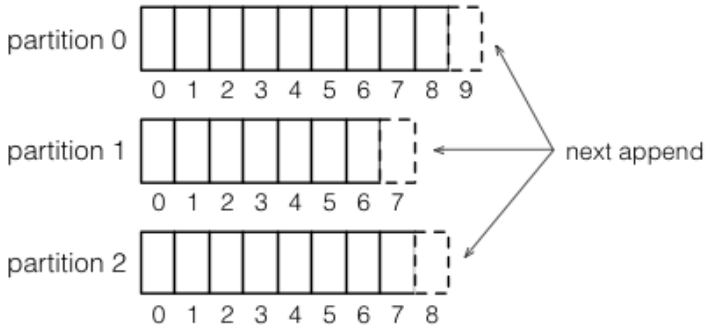

Figura 9: Ilustração de um stream particionado no Samza
Fonte: (Apache Software Foundation, 2016f)

Após essas breves descrições das plataformas para processamento de eventos em tempo real e da análise realizada do número de citações, o Spark Streaming e Storm foram escolhidas como ferramentas para o trabalho desenvolvido no Cap. 3. Isso, porque ambas possuem maior número de citações nas fontes pesquisadas, e consequentemente, comunidades maiores, o que pode favorecer melhor documentação, suporte e continuidade desses projetos.

Sendo assim, como Flink e Samza são semelhantes ao Spark o único critério de diferenciação, superfecialmente, é o tamanho da comunidade e a popularidade que o Spark tem. No demais, o Storm se diferencia com o conceito de topologia, interessante de ser estudado. Devido a isso, o Spark e Storm foram aprofundados no Cap. 2 e utilizados no trabalho a ser apresentado nesse capítulo.

3.2 Aplicações desenvolvidas para estudo de caso

Após a escolha das plataformas para processamento de eventos em tempo real, quatro aplicações foram desenvolvidas. A explicada na subseção 3.2.1, exibe numa aplicação web um mapa contendo informações sobre as métricas de e-Participação. A subseção 3.2.2, refere-se a aplicação responsável por processar tweets e coletar essas métricas; a da subseção 3.2.3, por sua vez, usa o Spark Streaming para realizar análise de sentimento de streams de tweets; a referente a subseção 3.2.4, realiza o mesmo procedimento, mas com o Storm.

Tabela 1: Quantidade de citações a plataformas ESPs por referência

Referência	Spark Streaming	Storm	Flink	Samza
(WÄHNER, 2014)	1	1	1	1
(KLEPPMANN, 2015)	1	0	0	1
(Siciliane, T., 2015)	1	1	0	1
(NARSUDE, 2015)	1	1	0	0
(ZAPLETAL, 2015)	1	1	0	0
(SANDU, 2014)	1	1	0	0
(TREAT, 2015)	1	0	0	1
(Google Scholar, 2016)	806	766	171	283
(Stack Over Flow, 2016)	1541	538	567	87
Total de citações	2354	1311	739	374

Fonte: Elaboração própria

3.2.1 Aplicação web para visualização das métricas relacionadas a e-Participação

A aplicação web foi desenvolvida utilizando o framework web Django (Django Software Foundation, 2016), devido a sua simplicidade. Há somente uma página, contruída usando HTML, JavaScript e CSS, na qual o mapa para a visualização das métricas é exibido. O mapa, por sua vez, é obtido com o suporte da API MapsJavaScript (Google, 2016a) sendo nele sobrepostas camadas do Google Fusion Tables (aplicação web experimental para visualização de dados, coleta e compartilhamento de tabelas (Google, 2016b)).

As métricas podem ser escolhidas através de uma caixa de seleção (ComboBox), sendo assim, para cada Estado, se sua respectiva média for maior que a nacional a região dele no mapa é sobreposta com uma camada azul, se menor, vermelha. Ainda, clicando num Estado é possível visualizar um painel informativo com os valores das médias, medianas, mínimo, máximo, variância e desvio padrão, assim como a quantidade de tweets, seguidores, e o tempo de existência da conta.

Além disso, há conexão via API com a aplicação responsável por realizar a coleta de tweets e processamento das métricas. O tempo necessário para o levantamento de todas essas informações é em torno de 1h30m, devido aos limites de requisições à API do Twitter, explicado na subseção 3.2.2, e do Fusion Tables, limitado em 30 chamadas por minuto (Google, 2016a). Devido a isso, essa opção somente é habilitada no código quando necessário.

O código dessa aplicação está disponível no repositório localizado em (DIAS, 2016b), sendo "brazilian_smart_cities_map"o nome do projeto.

3.2.2 Aplicação para processamento de tweets e coleta de métricas relacionadas a e-Participação

O primeiro passo no desenvolvimento dessa aplicação, foi decidir as contas das quais os tweets seriam processados. Como, normalmente, as capitais dos estados tem maior concentração de pessoas, optou-se por fazer um levantamento dos perfis oficiais de suas respectivas prefeituras, para então posteriormente realizar o processamento dos tweets. Sendo assim, a Tab. 2 lista as contas relacionadas as prefeituras municipais das capitais dos vinte e sete estados brasileiros.

Em seguida, foram escolhidas quais métricas dessas contas seriam possíveis e importantes de coletar, tendo como referência (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014). Sendo assim, selecionou-se as seguintes, respectivas ao Twitter: média do número de tweets, seguidores, retweets (compartilhamento de um determinado tweet), comentários realizados por usuários, réplicas a tweets e tempo de resposta. As métricas referentes ao número de usuários acompanhando as listas (junções de timelines) do perfil e o total delas existentes foram desconsideras, pois são relacionadas a contas diferentes das em questão.

De acorco com (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014), através dessas métricas é possível obter indicadores relacionados ao nível e-Participação. Alguns dos indicadores propostos pela SNR (Social Network Ratio) para Redes Sociais são: Atividade, ou, audiência estimada; Tamanho, ou, esforço realizado pelo perfil para se comunicar; Visibilidade, ou, número total de menções ao perfil; Interação, ou, capacidade de impacto (viralização) da comunicação (SAÉZ-MARTÍN; ROSSARIO; CABA-PEREZ, 2014).

Portando, pode-se mapear a métrica média de seguidores ao indicador Atividade, e a de menções para o de Visibilidade; da mesma forma, as médias sobre tweets, réplicas por dia e tempo de resposta ao indicador Tamanho; por último, as médias de retweets e favoritos ao de Interação. A Fig. 10 exibe o diagrama de classes da aplicação desenvolvida para o processamento de tweets e coleta dessas métricas.

A aplicação exibida no diagrama de classes da Fig. 10 foi desenvolvida na linguagem de programação Java, devido a sua praticidade de uso, utilizando o framework para Web services, Apache CXF (Apache Software Foundation, 2016a), para expor dois de seus serviços através de uma REST API. O primeiro deles é exposto pela interface ITweetService, e o segundo pela ISparkService, pelos seguintes endpoints, respectivamente: "\tweets"e "\metrics", permitindo intregrá-la a aplicação web descrita na subseção 3.2.1.

A classe que implementa a interface ITweetService, é responsável por realizar a co-

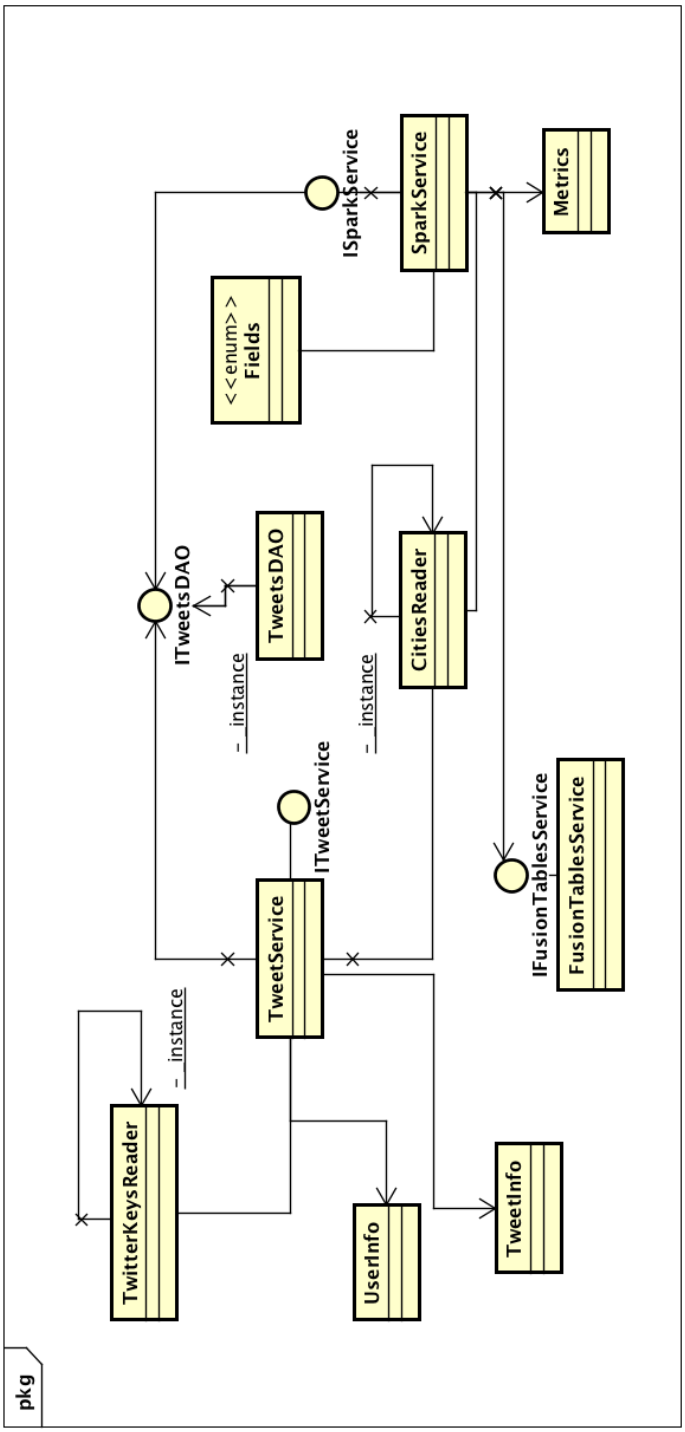

Figura 10: Diagrama de classes da aplicação desenvolvida para processamento e coleta de métricas relacionadas a e-Participação

Fonte: Elaboração própria

leta dos 3.200 tweets mais recentes (se disponíveis) de cada conta, através do endpoint "statuses/user_timeline". Tal quantidade é limitada pela API do Twitter, a qual pode ser

alcançada por no máximo 180 requisições, num intervalo de 15 minutos, com autenticação via conta de usuário (TWITTER, 2016).

Outro endpoint utilizado foi "search/tweets", para pesquisar as menções realizadas pelos cidadãos as contas das prefeituras. O limite de coleta é de 100 tweets coletados para cada requisição, sendo possível realizar 180 a cada 15 minutos. Os endpoints da API do Twitter foram acessados com o suporte da biblioteca Twitter4J (TWITTER4J, 2016).

Durante a coleta dos tweets, eles são mapeados para as seguintes classes do modelo da aplicação: UserInfo, que contém as informações respectivas ao usuário da conta (número de seguidores, tweets, localização, username e data de criação da conta), e TweetInfo, contendo as relacionadas ao tweet (data de criação, id do tweet de réplica e a qual usuário ele se refere, quantidade de retweets, favoritos, se é menção ou não, e o tempo de resposta calculado). Em seguida, os modelos são persistidos via a interface ITweetsDAO, a qual se comunica com o banco de dados não relacional MongoDB (MongoDB, 2016).

Os tweets coletados por essa aplicação são os mais recentes e anteriores a data 18/06/2016 (incluindo-a). Como essa coleta não foi realizada sob um stream de tweets, a interface ISparkService expõe o serviço responsável por realizar o processamento em batch desses tweets, coletando as métricas relacionadas a e-Participação. Sendo assim, cada métrica é recuperada do banco de dados e mepeadas para um RDD de doubles, quando números, ou, de strings, no caso da data.

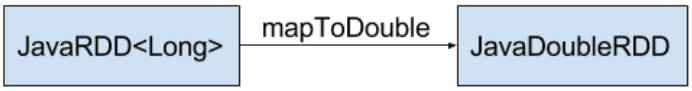

Figura 11: Diagrama do mapeamento entre um Resilient Distributed Dataset de Long para Double

Fonte: Elaboração própria

Sendo assim, após recuperar as métricas, é possível mapeá-las para um RDD de doubles, por meio do qual são obtidos os valores relacionados as suas respectivas médias, medianas, variâncias, máximos, mínimos e desvios padrões. As informações sobre datas, como strings, são mapeadas para o valor 1, representando a ocorrência de um tweet nesse dia; compondo uma sequência de pares, que permite obter e mapear as quantidades de tweets por dia a um RDD de doubles. Tais processos de mapeamentos são ilustrados nas Fig. 11 e Fig. 12.

Por fim, a interface IFusionTable é utilizada para submeter os valores das métricas relacionadas e-Participação a uma Fusion Table (DIAS, 2016a), via a API do Google Fusion Tables. Os valores dessa tabela são utilizados para criar o mapa contido na aplicação web.

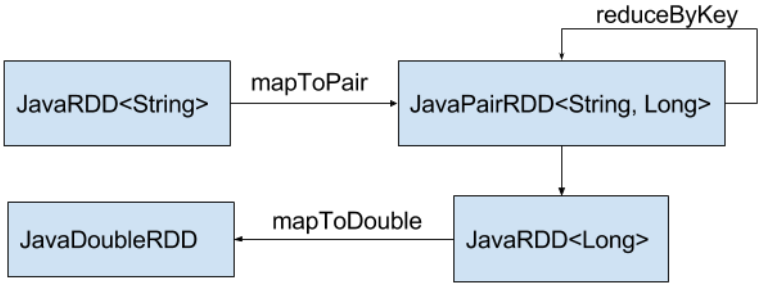

Figura 12: Diagrama do mapeamento entre um Resilient Distributed Dataset de Datas para suas respectivas frequências em Double

Fonte: Elaboração própria

O código dessa aplicação está disponível no repositório localizado em (DIAS, 2016b), sendo "twitter-data-analysis"o nome do projeto.

3.2.3 Aplicação para processamento de stream de tweets, utilizando Spark Streaming

A aplicação que utiliza o Spark Streaming, foi desenvolvida na linguagem de programação Java. No início de sua execução, é criado um contexto de stream no qual é definindo o cluster onde ela será executada e o intervalo de criação de cada RDD. Tais Resilient Distributed Datasets, são compostos ordenadamente em sequência, formando a abstração conhecida como DStream. No nosso caso, cada RDD é criado após 30000ms; sendo compostos pelos tweets coletados filtrando os nomes das conta das prefeituras no Twitter, enumerados na classe KeyWords, ilustrada no diagrama da Fig. 14.

O processo mencionado anteriormente, executado pela classe SparkStreaming, começa após a inicialização do contexto de stream, sendo os tweets coletados pela classe TwitterUtils (do próprio Spark Streaming). Durante a coleta dos streams de tweets (eventos), cada RDD é mapeado para a classe TweetInfo (do modelo da aplicação), através de uma transformação map. Na sequência, as ações forEachRDD e collect são executadas para inserir

Tabela 2: Contas do Twitter relacionadas as prefeituras municipais das capitais dos vinte e sete estados brasileiros

Estado	Capital	Conta no Twitter
Acre	Rio Branco	PrefRioBranco
Alagoas	Maceió	PrefMaceio
Amapá	Macapá	PMMacapa
Amazonas	Manaus	PrefManaus
Bahia	Salvador	agecomsalvador
Distrito Federal	Brasília	Gov_DF
Ceará	Fortaleza	prefeiturapmf
Espírito Santo	Vitória	VitoriaOnLine
Goiás	Goiánia	PrefeituraGy
Maranhão	São Luís	PrefeituraSL
Mato Grosso	Cuiabá	prefeitura_CBA
Mato Grosso do Sul	Campo Grande	cgnoticias
Minas Gerais	Belo Horizonte	prefeiturabh
Paraná	Curitiba	Curitiba_PMC
Paraíba	João Pessoa	pmjponline
Pará	Belém	prefeiturabelem
Pernambuco	Recife	prefrecife
Piauí	Teresina	prefeitura_the
Rio Grande do Norte	Natal	NatalPrefeitura
Rio Grande do Sul	Porto Alegre	Prefeitura_POA
Rio de Janeiro	Rio de Janeiro	Prefeitura_Rio
Rondônia	Porto Velho	prefeitura_pvh
Roraima	Boa Vista	PrefeituraBV
Santa Catarina	Florianópolis	scflorianopolis
Sergipe	Aracaju	PrefeituraAracaju
São Paulo	São Paulo	prefsp
Tocantins	Palmas	cidadepalmasy

Fonte: Elaboração própria

os tweets processados no banco de dados não relacional MongoDB, conforme ilustrado na Fig. 13.

A análise de sentimentos do conteúdo contido nos tweets ocorre durante a última parte do mapeamento, sendo antes disso realizados alguns processamentos para facilitar e viabilizar esse procedimento. Portanto, primeiramente, os textos contidos nos tweets são extraídos e formatados para minúsculo. Em seguida, todas as menções são identificadas pelo padrão em que ocorrem no Twitter (@displayname, nome exibido para os demais usuários da Rede Social) e removidas, assim como as referências a endereços de sites.

No Twitter, quando um tweet é compartilhado ele é marcado com a notação "RT",

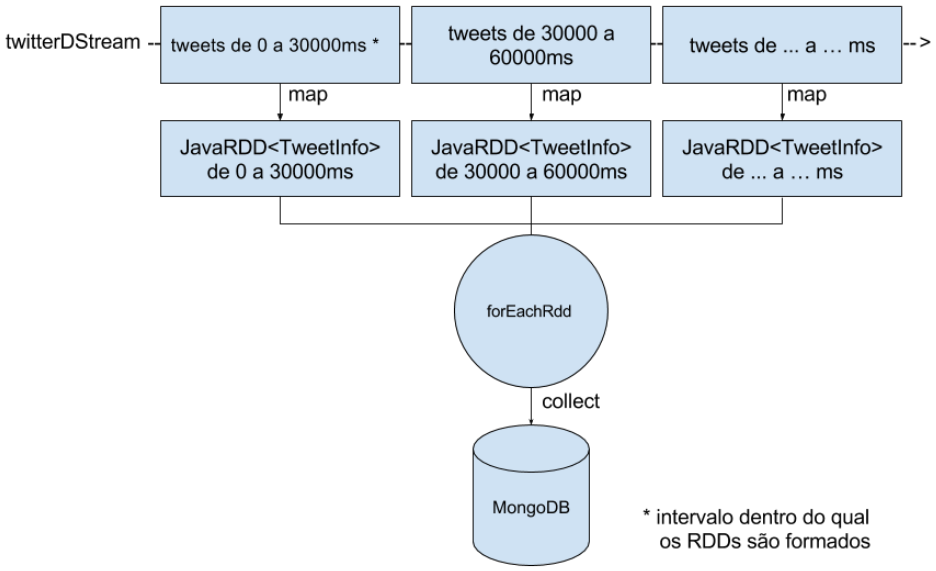

Figura 13: Fluxo do processamento de dados da aplicação utilizando o Spark Streaming

Fonte: Elaboração própria

abreviação de retweet, a qual também é removida do texto em processamento. Além disso, alguns símbolos são removidos, tais como: ., ., , %, #, etc. Sendo esse o conjunto de processamentos realizados para "limpar"inicialmente o texto, após o qual se inicia o processamento de linguagem natural.

Com esse prósito, usou-se a biblioteca OpenNLP (Apache Software Foundation, 2016e) para a tokenização dos tweets. Após a obtenção dos tokens, outros processamentos foram necesssários para melhorar o desempenho da fase seguinte, como a substituição das palavras normalmente abreviadas (vc - você, msm - mesmo, pq - porque, q - que, n - não, etc) e de expressões (sqn - só que não, kkk, hahaha, rsrsrs para situaões comumente engraçadas) por seus formatos formais, utilizando dicionários previamente construídos. Além disso, foram removidos os tokens contendo "stopwords"(palavras vazias), termo utilizado para as palavras comuns de um certo idioma (LESKOVEC; RAJARAMAN; ULLMAN, 2016).

Após a obtenção dos tokens, foram atribuídas a eles tags referentes as suas respectivas classes gramáticais, e, por fim, associadas uma part-of speech para cada tweet, usando os tokens e tags obtidos, finalizando a parte do processamento de linguagem natural. Indo então para a última etapa, que é a da análise de sentimentos, tendo como base para isso os adjetivos presentes em cada tweet.

A análise de sentimentos foi realizada com o suporte do Sentilex (versão 1), que é

um léxico de sentimentos para o Português, constituído de 6.321 lemas adjectivais (por convenção, na forma masculina singular) e 25.406 formas flexionadas, contendo como um de seus atributos a polaridade do adjectivo. As polaridade são classificadas em positivo (67% de precisão), negativo (82% de precisão), ou, neutro (45% de precisão), possibilitando estimar o sentimento expresso por um determinado texto (SILVA et al., 2016).

Os sentimentos são estimados contabilizando as polaridades presentes em cada tweet e o sentimento expresso pelos emotions (se houver), assim sendo, por exemplo, o emotion "(:"incrementa 1 para a polaridade postiva, e 1 para a negativa caso seja ":(". Também, considera-se a presença de advérbios de negação, os quais modificam o significado do verbo, adjetivo e de outros advérbios (PRIBERAM, 2016), alterando consequentemente a polaridade. Por fim, é realizada uma simples normatização com os somatórios das polaridades positivas e negativas, seguindo o seguinte modelo:

$$\text{score} = (\Sigma \text{ positivo - } \Sigma \text{ negativo}) \div (\Sigma \text{ positivo} + \Sigma \text{ negativo})$$

Caso o score seja menor do que zero, o tweet é classificado com polaridade negativa, se o seu complemento for maior do que 0.5, tem-se polaridade positiva e se igual a zero, neutra. Sendo assim, as informações sobre a polaridade (sentimento) do tweet, seu id e suas respectivas menções (recuperadas do tweet original) são armazenadas no banco de dados não relacional MongoDB. As polaridades são exibidas no mapa da aplicação web.

O código dessa aplicação está disponível no repositório localizado em (DIAS, 2016b), sendo "spark-data-analysis"o nome do projeto, e seu respectivo diagrama de classe é ilustrado na Fig. 14

3.2.4 Aplicação para processamento de stream de tweets, utilizando Storm

A aplicação que utiliza o Storm para o processamento de stream de tweets, foi desenvolvida utilizando a linguagem de programação Java. Nela é construída uma topologia, ilustrada na Fig. 15, composta por um Spout (classe Twitter), responsável pela conexão ao Twitter e coleta dos tweets, tendo como filtro o nome das contas das prefeituras no Twitter, utilizando a biblioteca Twitter4J.

Em sequência, há seis bolts, responsáveis pelo processamento (o mesmo realizado pela aplicação Spark) dos tweets coletados. O primeiro deles, classe TweetCleanerBolt, ilustrada na Fig. 16, remove as menções e urls contidas no tweet, após isso os simbolos existentes são removidos pelo bolt Symbols Cleaner.

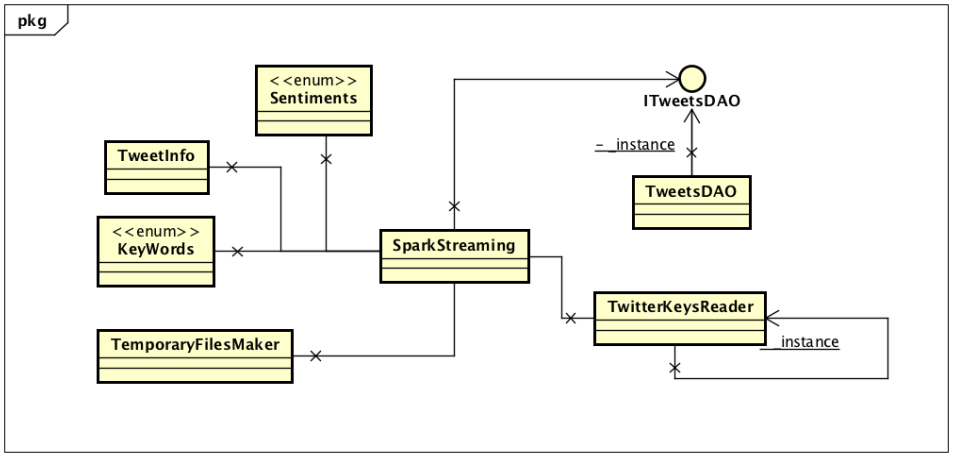

Figura 14: Diagrama de classes da aplicação utilizando o Spark Streaming

Fonte: Elaboração própria

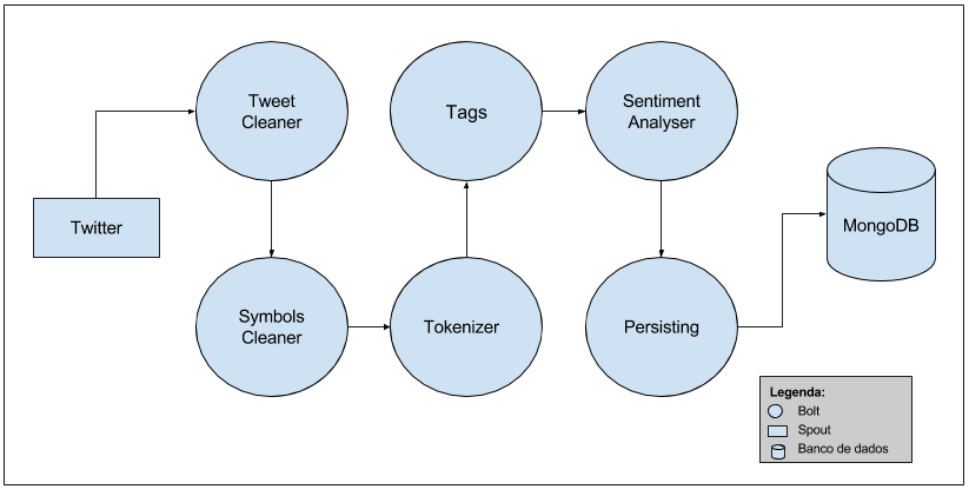

Figura 15: Topologia da aplicação utilizando Storm

Fonte: Elaboração própria

Após isso, a parte do processamento de linguagem natural é realizado pelos bolts Tokenizer e Tag. Na finalização do processo, o bol SentimentAnalyser calcula a polarização de cada tweet, emitindo-os para o bolt Persisting, responsável pelo armazená-los no banco de dados não relacional MongoDB. Vale ainda mencionar que os resultados de cada bolt são setados na classe TweetStream, do modelo da aplicação, o que foi necessário por nem todos os tipos de dados serem serializáveis, como o caso do UserMentionEntity, da biblioteca Twitter4J.

O código dessa aplicação está disponível no repositório localizado em (DIAS, 2016b),

sendo "storm-data-analysis"o nome do projeto

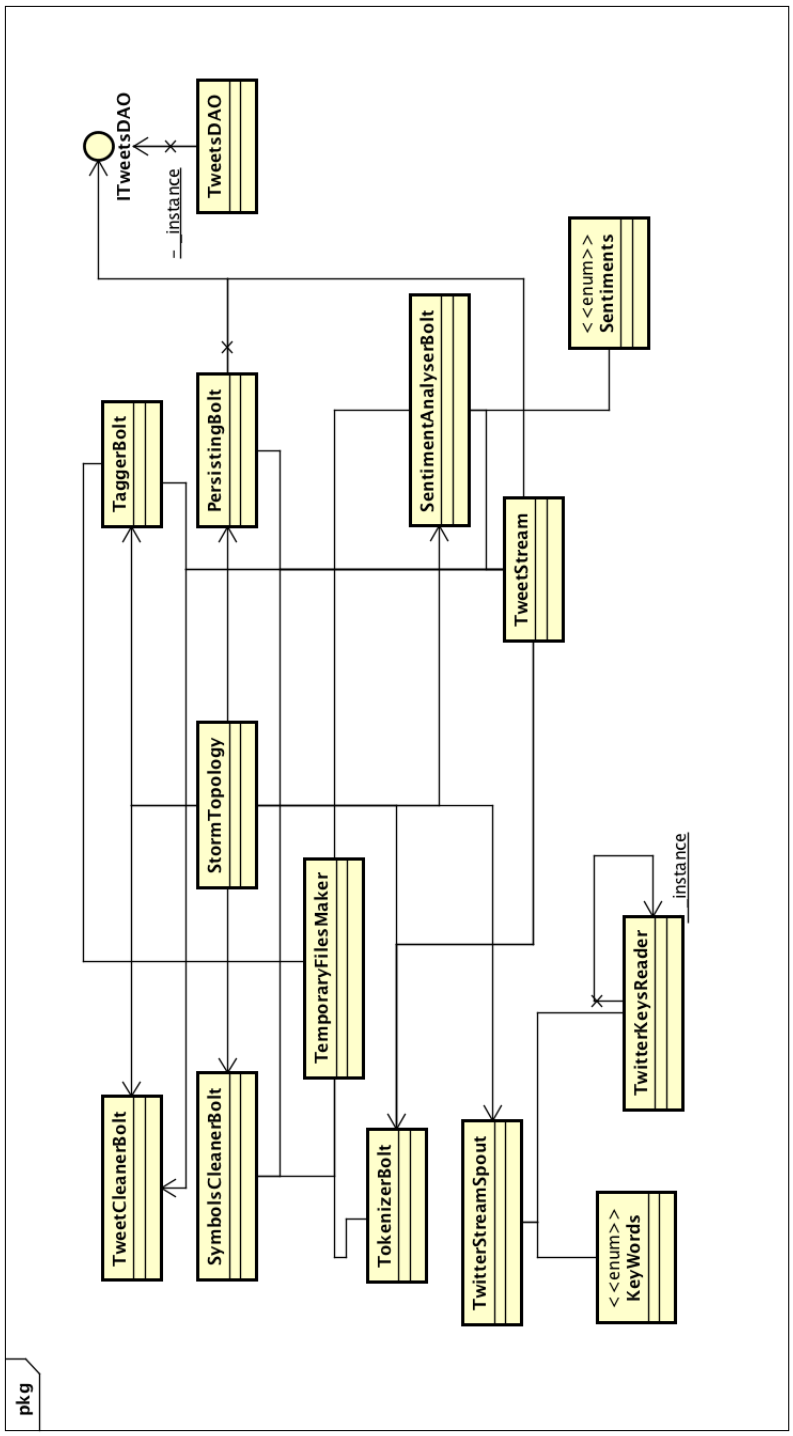

Figura 16: Diagrama de classes da aplicação utilizando o Storm

Fonte: Elaboração própria

3.3 Estudo comparativo

Nessa seção, é realizada uma análise comparativa entre o Apache Storm e Spark, sendo a subseção 3.3.1 sobre o requisito de Processamento de grande volume de dados em tempo real, a 3.3.2 a respeito de Tolerância a falhas, a 3.3.2.1 sobre Garantias de processamento, a 3.3.3 em relação a Escalabilidade e, por fim, a 3.3.4 sobre o modelo de programação de ambas as ferramentas.

3.3.1 Processamento de grande volume de dados em tempo real

Quanto ao requisito de processamento de grande volumes de dados, de acordo com a referência (DAS, 2016), o Apache Spark pode processar até 670k de registros por segundo e por nó, enquanto que o Storm 155k. Portanto, ambas as ferramentas são capazes de processar grande fluxo de dados.

No entanto, elas se diferenciam no requisito de processamento em tempo real. O Spark tem latência de 0.52s (ZAHARIA et al., 2012b), e devido a esse delay de até alguns segundos, é considerado uma ferramenta de Near real time. O Storm, por outro lado, tem como latência de 1100ms (ZAHARIA et al., 2013), (ZAHARIA et al., 2012b), sendo por isso um sistema com processamento em tempo real.

Como já mencionado na fundamentação tórica, quanto menor o valor de latência, mais rápido se obtém resposta a um dado evento. Além disso, no quesito de processamento em tempo real, também é importante avaliar o valor de throughput, o qual no Spark é maior em comparação ao do Storm (ZAHARIA et al., 2013), favorecendo para o tempo de processamento ser menor.

Como base no que foi dito anteriormente, nesse quesito o Spark é mais adequado as aplicações desenvolvidas, pois elas estão inseridas num contexto em que a propriedade de throughput tem maior relevância. Isso, devido a quantidade de tweets processados, tanto para análise de sentimento, como o módulo Spark Streaming, quanto para a obtenção das métricas relacionadas a e-Participação.

3.3.2 Tolerância a falhas

Discretized Streams (DStreams) é um modelo de programação para processamento de streams distribuídas, utilizado pelo Spark Streaming, capaz de fornecer tolerância a falhas, através do método parallel recovery (ZAHARIA et al., 2012c). Nesse modelo, a tolerância

a falhas é implementada através do conceito lineage, o que permite as informações serem recuperadas paralelamente (ZAHARIA et al., 2012a).

O mecanismo de recuperação via lineage é definido por um grafo acíclico dirigido (DAG), por meio do qual RDDs e DStreams rastreiam, ao nível das partições RDDs, suas respectivas dependências e operações realizadas sob elas (ZAHARIA et al., 2012a). Sendo assim, os RDDs e DStreams conseguem "saber"como foram construídos. Podendo, consequentemente, cada nó do cluster reconstruí-lo paralela e eficientemente em caso de falhas. Especificamente, o processo de recuperação é realizado computando novamente uma determinada partição RDD, reexecutando as tasks que a originaram (ZAHARIA et al., 2012a).

Visando prevenir infinitas recomputações, também são realizados checkpoints num determinado espaço de tempo, com replicações assíncronas de RDDs. Tal procedimento não é necessário para todo o conjunto de dados, pois, como já mencionado, a recuperação executada por nós em paralelo é realizada com demasiada eficiência (ZAHARIA et al., 2012a).

Todo o processo descrito até aqui é confiável quando a fonte dos dados pode ser lida novamente, no caso do Spark Streaming é necessário haver alguma fonte externa para replicação dos dados (Apache Software Foundation, 2016g). Além disso, se o Driver for finalizado, por manter o contexto da aplicação, todo o conteúdo em memória dos executors é perdido (Apache Software Foundation, 2016g).

No Storm, a tolerância a falhas é aplicada a cada um de seus componentes. Por exemplo, se o worker falha, ele é reinicializado pelo Supervisor no próprio nó (JAIN; NALYA, 2014). Caso o nó esteja indisposnível, ou, falhando continuadamente, suas tasks são então atribuídas a outro disponível no cluster, via Nimbus (HART; BHATNAGAR, 2015).

O Nimbus e os supervisors, armazenam seus estados no Zookeeper, podendo ser reinicializados sem perdê-los em caso de falha, se houver falha no Zookeeper, outro pode ser "eleito"para o seu lugar (JAIN; NALYA, 2014). Caso algum supervisor falhe, seus workers são reatribuídos pelo Nimbus a outro supervisor, no entanto, ficando impedido de receberem novas tuplas (HART; BHATNAGAR, 2015).

Por sua vez, o Nimbus, dentre suas atribuições, é responsável também por reinicializar as tasks caso uma delas venha a falhar, e se o mesmo acontecer com ele próprio, as tasks em execução não são afetadas, mas novas topologias são impedidas de serem submetidas ao cluster (JAIN; NALYA, 2014), assim como reatribuições (HART; BHATNAGAR, 2015).

O modelo de recuperação de falhas do Spark pode ser uma boa escolha caso a aplicação permita perda de dados, em prol de eficiência (ZAHARIA et al., 2012c). Caso contrário, é necessário configurar uma fonte externa para replicação dos dados que estão sendo processados. Em contraste, a arquitetura do Storm, busca possibilitar que seus componentes falhem havendo pouco prejuízo para a aplicação, ainda assim a replicação pode adicionar mais uma camada de confiabilidade.

Levando em consideração o exposto acima, a aplicação desenvolvida para a coleta de métricas realiza processamento em batch, podendo ter acesso a fonte de dados para coletá-los novamente, em caso de falha. Além disso, a eficiência foi priorizada, não havendo prejuízo ao utilizar o Spark. Para aplicação que realiza análise de sentimentos, processamendo stream de tweets, é mais interessante o suporte a tolerância a falhas do Storm, principalmente devido ao seu longo tempo de execução.

3.3.2.1 Garantia de processamento

Em adição a subseção anterior, o Spark Streaming e Storm possuem formas diferentes de garantir o processamento de um evento. No Spark Streaminghá a garantia de que todo evento será processado exatamente uma vez (Exactly Once), sem perdas, ou, duplicadas, via parallel recovery, levando em consideração as observações já mencionadas (Apache Software Foundation, 2016g).

No Storm, por outro lado, não há suporte ao modelo "Exactly Once", mas sim aos "At Least Once"e "At Most Once". Em ordem, a primeira opção permite que o processamento seja realizado no mínimo uma vez, rastreando se o (a) evento (tupla) foi processado(a) ou não, através de seus IDs e mensagens "ack"(tupla processada), podendo haver duplicatas; o oposto ocorre na segunda alternativa, em que perdas são aceitadas processando os eventos no máximo uma vez (Apache Software Foundation, 2016h).

A prosta do Spark Streaming, nesse aspecto, é mais interessante para o processamento de stream de tweets, por através do "Exactly Once"garantir que a informação será processada uma única vez.

3.3.3 Escalabilidade

Quanto ao requisito de escalabilidade, ambos suportam clusterização, portando são escaláveis horizontalmente, sendo o maior cluster Spark conhecido composto por 8.000 nós (Apache Software Foundation, 2016g). E, embora não tenha sido encontradas fontes confiáveis

divulgando informações sobre o maior cluster Storm existente, sabese que ele é capaz de processar um milhão de mensagens com tamanho de 100 byte, por segundo e por nó (Apache Software Foundation, 2016h).

3.3.4 Modelo de programação

Conforme exposto no Cap. 2, no Spark os RDDs são conjuntos de dados que podem ser manipulados de forma distribuída, sendo possível realizar operações sob eles, gerando novos RDDs, através de transformações, ou, computando-os por meio das ações. O módulo Streaming utiliza essa unidade como base da abstração DStream, conjunto de RDDs representando um stream. Tais conceitos do modelo de programação do Spark podem ser mais difíceis de abstrair, tendo uma classificação de baixo nível, se comparados ao do Storm.

Por outro lado, numa perspectiva alto nível, o Storm propõe o conceito de topologia, composta por Bolts e Spouts. Os Spouts são responsáveis pela entrada dos streams (conjunto de tuplas) da aplicação, processados posteriormente pelos Bolts. Devido a essas abstrações, consequentemente, pode haver maior facilidade em programar utilizando o Storm do que o Spark.

4 Resultados e Trabalhos Relacionados

4.1 Resultados

4.2 Trabalhos Relacionados

Tabela 3: Métricas relacionadas ao número de retweets dos perfis das prefeituras das capitais, referentes aos 3.200 tweets anteriores a 18/05/2016

UF	**Capital**	**Média**	**Med.**	**Mín.**	**Máx.**	**Variância**	**D. Padrão**
AC	Rio Branco	0.3885	0	0	33	1.5863	1.2595
AL	Maceió	0.4881	0	0	49	2.0948	1.4473
AP	Macapá	5.8987	5	0	37	21.9878	4.6891
AM	Manaus	0.4962	0	0	162	12.5106	3.5370
BA	Salvador	0.0837	0	0	2	0.0929	0.3049
DF	Brasília	1.6449	1	0	329	63.2714	1.9681
CE	Fortaleza	0.7656	0	0	32	1.5338	1.2384
ES	Vitória	0.3018	0	0	23	0.7576	0.8704
GO	Goiánia	1.6332	1	0	18	3.5193	1.8759
MA	São Luís	14.5868	15	0	47	147.1399	12.1301
MT	Cuiabá	0.0765	0	0	4	0.0919	0.3032
MS	Campo Grande	0.0784	0	0	3	0.0841	0.2901
MG	Belo Horizonte	2.5498	1	0	308	53.3316	7.3028
PR	Curitiba	7.9868	1	0	842	1044.1073	32.3126
PB	João Pessoa	0.6059	0	0	14	1.4487	1.2036
PA	Belém	0.3258	0	0	10	0.6245	0.7902
PE	Recife	1.3226	0	0	74	12.8881	3.5900
PI	Teresina	2.7886	0	0	2274	2822.6661	53.1287
RN	Natal	2.5653	2	0	45	13.9157	3.7303
RS	Porto Alegre	4.9975	2	0	209	106.0212	10.2966
RJ	Rio de Janeiro	2.6015	1	0	2868	2611.8290	51.1060
RO	Porto Velho	0.3559	0	0	5	1.0223	1.0111
RR	Boa Vista	0.2621	0	0	20	0.9140	0.9560
SC	Florianópolis	0.5096	0	0	27	1.1811	1.0868
SE	Aracaju	1.3246	1	0	14	1.8780	1.3704
SP	São Paulo	5.2272	3	0	237	104.7239	10.2334
TO	Palmas	1.3872	1	0	31	3.3871	1.8404

Fonte: Elaboração própria

Tabela 4: Métricas relacionadas ao número de favoritos dos perfis das prefeituras das capitais, referentes aos 3.200 tweets anteriores a 18/05/2016

UF	**Capital**	**Média**	**Med.**	**Mín.**	**Máx.**	**Variância**	**D. Padrão**
AC	Rio Branco	0.1981	0	0	10	0.2872	0.5359
AL	Maceió	0.8884	0	0	32	2.9797	1.72618
AP	Macapá	3.3284	3	0	22	7.8843	2.8079
AM	Manaus	1.0487	1	0	40	3.5744	1.8906
BA	Salvador	0.0821	0	0	2	0.0916	0.3027
DF	Brasília	1.9681	1	0	504	160.0064	12.6493
CE	Fortaleza	1.8134	1	0	22	2.6636	1.6320
ES	Vitória	0.4421	0	0	15	0.6922	0.8320
GO	Goiánia	1.9177	1	0	20	3.7246	1.9299
MA	São Luís	16.3471	18	0	63	149.8610	12.2417
MT	Cuiabá	0.1206	0	0	8	0.1629	0.4036
MS	Campo Grande	0.0025	0	0	1	0.0024	0.0499
MG	Belo Horizonte	3.4410	2	0	257	37.6513	6.1360
PR	Curitiba	12.7833	1	0	443	698.3285	26.4259
PB	João Pessoa	1.1934	1	0	37	4.0453	2.0113
PA	Belém	0.5028	0	0	9	0.8356	0.9141
PE	Recife	2.1312	1	0	39	8.2941	2.8799
PI	Teresina	0.8487	0	0	12	1.5913	1.2614
RN	Natal	2.6521	2	0	25	7.8230	2.7969
RS	Porto Alegre	2.2115	0	0	117	37.0543	6.0872
RJ	Rio de Janeiro	3.5790	2	0	2216	1543.9374	39.2929
RO	Porto Velho	0.03245	0	0	9	0.09345	0.3057
RR	Boa Vista	0.3943	0	0	9	0.6482	0.8051
SC	Florianópolis	0.4428	0	0	22	1.1186	1.0576
SE	Aracaju	1.1550	1	0	10	1.9697	1.4034
SP	São Paulo	4.3504	0	0	213	101.2060	10.0601
TO	Palmas	2.2001	1	0	40	7.5964	2.7561

Fonte: Elaboração própria

Tabela 5: Métricas relacionadas ao número de réplicas dos perfis das prefeituras das capitais, referentes aos 3.200 tweets anteriores a 18/05/2016

UF	**Capital**	**Média**	**Med.**	**Mín.**	**Máx.**	**Variância**	**D. Padrão**
AC	Rio Branco	0.0046	0	0	1	0.0046	0.0683
AL	Maceió	0.0800	0	0	1	0.0736	0.2712
AP	Macapá	0.0884	0	0	1	0.0806	0.2839
AM	Manaus	0.0228	0	0	1	0.0222	0.1493
BA	Salvador	0	0	0	0	0	0
DF	Brasília	0.0218	0	0	1	0.0213	0.1462
CE	Fortaleza	0.0099	0	0	1	0.0098	0.0994
ES	Vitória	0.0356	0	0	1	0.0343	0.1853
GO	Goiánia	0.0253	0	0	1	0.0246	0.1571
MA	São Luís	0.0825	0	0	1	0.0756	0.2751
MT	Cuiabá	0.01187	0	0	1	0.0117	0.1083
MS	Campo Grande	0.0012	0	0	1	0.0012	0.0353
MG	Belo Horizonte	0.0375	0	0	1	0.0361	0.1900
PR	Curitiba	0.5395	1	0	1	0.2484	0.4984
PB	João Pessoa	0.2106	0	0	1	0.1662	0.4077
PA	Belém	0.0244	0	0	1	0.0238	0.1545
PE	Recife	0.2525	0	0	1	0.1887	0.4344
PI	Teresina	0.1337	0	0	1	0.1158	0.3404
RN	Natal	0.0118	0	0	1	0.0117	0.1083
RS	Porto Alegre	0.1231	0	0	1	0.1079	0.3285
RJ	Rio de Janeiro	0.0315	0	0	1	0.0305	0.1748
RO	Porto Velho	0	0	0	0	0	0
RR	Boa Vista	0.0403	0	0	1	0.0386	0.1966
SC	Florianópolis	0.0046	0	0	1	0.0046	0.0683
SE	Aracaju	0.0015625	0	0	1	0.0015	0.0394
SP	São Paulo	0.2719	0	0	1	0.1979	0.4449
TO	Palmas	0.0878	0	0	1	0.0801	0.2831

Fonte: Elaboração própria

5 Capítulo 5

5.1 Seção 1

Seção 1

5.2 Seção 2

5.2.1 Subseção 5.1

Subseção 5.1

5.2.2 Subseção 5.2

Subsection 5.2

5.3 Seção 3

Seção 3

6 Considerações finais

As considerações finais formam a parte final (fechamento) do texto, sendo dito de forma resumida (1) o que foi desenvolvido no presente trabalho e quais os resultados do mesmo, (2) o que se pôde concluir após o desenvolvimento bem como as principais contribuições do trabalho, e (3) perspectivas para o desenvolvimento de trabalhos futuros. O texto referente às considerações finais do autor deve salientar a extensão e os resultados da contribuição do trabalho e os argumentos utilizados estar baseados em dados comprovados e fundamentados nos resultados e na discussão do texto, contendo deduções lógicas correspondentes aos objetivos do trabalho, propostos inicialmente.

Referências

Apache Software Foundation. *Apache CXF*. 2016. Disponível em: <http://cxf.apache.org>. Acesso em Maio 20, 2016.

Apache Software Foundation. *Apache Flink*. 2016. Disponível em: <http://flink.apache.org>. Acesso em Maio 7, 2016.

Apache Software Foundation. *Apache Hadoop YARN* . 2016. Disponível em: <http://hadoop.apache.org>. Acesso em Abril 22, 2016.

Apache Software Foundation. *Apache Mesos*. 2016. Disponível em: <http://mesos.apache.org>. Acesso em Abril 23, 2016.

Apache Software Foundation. *Apache OpenNLP*. 2016. Disponível em: <https://opennlp.apache.org>. Acesso em Maio 21, 2016.

Apache Software Foundation. *Apache Samza*. 2016. Disponível em: <http://samza.apache.org>. Acesso em Maio 06, 2016.

Apache Software Foundation. *Apache Spark*. 2016. Disponível em: <http://spark.apache.org>. Acesso em Abril 22, 2016.

Apache Software Foundation. *Apache Storm*. 2016. Disponível em: <http://storm.apache.org>. Acesso em Abril 30, 2016.

Apache Software Foundation. *Apache Thrift*. 2016. Disponível em: <http://thrift.apache.org>. Acesso em Abril 30, 2016.

Apache Software Foundation. *Apache ZooKeeper*. 2016. Disponível em: <https://zookeeper.apache.org>. Acesso em Abril 30, 2016.

CISCO. *Internet of Everything*. 2016. Disponível em: <http://ioeassessment.cisco.com/pt-br>. Acesso em Maio 07, 2016.

CLARKE, R. *Smart Cities and the Internet of Everything: The Foundation for Delivering Next Generation CitizenServices*. 2013. Disponível em: <http://www.cisco.com>. Acesso em Maio 07, 2016.

Connected Smartcities. *Ranking Connected Smartcities*. 2015. Disponível em: <http://www.connectedsmartcities.com.br>. Acesso em Maio 08, 2016.

COULOURIS, G. et al. *Sistemas Distribuídos Conceitos e Projetos*. 5st. ed. [S.l.]: Bookman, 2013.

DAS, T. *Large-scale near-real-time stream processing*. 2016. Disponível em: <https://goo.gl/QSQ5uK>. Acesso em Maio 22, 2016.

DATAARTISANS. *Apache Flink Training - System Overview*. 2015. Disponível em: <http://pt.slideshare.net>. Acesso em Maio 7, 2016.

DIAS, F. *Brazilian Smart Cities e-Participation Metrics based on Twitter*. 2016. Disponível em: <https://goo.gl/0I3Vgj>. Acesso em Maio 20, 2016.

DIAS, F. *Repositório no Git Hub*. 2016. Disponível em: <https://github.com/fcas>. Acesso em Maio 21, 2016.

Django Software Foundation. *Django*. 2016. Disponível em: <https://www.djangoproject.com>. Acesso em Maio 22, 2016.

EMC. *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things*. 2014. Disponível em: <https://www.emc.com>. Acesso em Maio 11, 2016.

ESPERTECH. 2016. Disponível em: <http://www.espertech.com>. Acesso em Maio 7, 2016.

Google. *Google Developers*. 2016. Disponível em: <https://developers.google.com>. Acesso em Maio 22, 2016.

Google. *Google Fusion Tables*. 2016. Disponível em: <http://tables.googlelabs.com>. Acesso em Maio 20, 2016.

Google Scholar. 2016. Disponível em: <https://scholar.google.com>. Acesso em Maio 7, 2016.

H., M. et al. *Special section on smart grids: A hub of interdisciplinary research*. 2016.

HARDIN, G. A vision of social media in the spanish smartest cities. *Science*, v. 162, 1968. Disponível em: <http://science.sciencemag.org>. Acesso em Maio 08, 2016.

HART, B.; BHATNAGAR, K. *Building Python Real-Time Applications with Storm*. 1st. ed. [S.l.]: Packt Publishing, 2015.

INFOPEDIA. *Tweet*. 2016. Disponível em: <http://www.infopedia.pt>. Acesso em Maio 08, 2016.

JAIN, A.; NALYA, A. *Learning Storm*. 1st. ed. [S.l.]: Packt Publishing, 2014.

KARAU, H. et al. *Learning Spark*. 1st. ed. [S.l.]: O'Reilly Media, 2015.

KLEPPMANN, M. *Stream processing, Event sourcing, Reactive, CEP? and making sense of it all*. 2015. Disponível em: <http://www.confluent.io>. Acesso em Maio 7, 2016.

LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. *Mining of Massive Datasets*. 2016. Disponível em: <http://infolab.stanford.edu/~ullman>. Acesso em Maio 21, 2016.

LUCKHAM, D. *What's the Difference Between ESP and CEP?* 2006. Disponível em: <http://www.complexevents.com>. Acesso em Maio 7, 2016.

MACIEL, C.; ROQUE, L.; GARCIA, A. *Democratic Citizenship Community: a social network to promote e-Deliberative process*. 2009.

MAGALHÃES, L. *Instâncias e mecanismos de participação em ambientes virtuais: análise das experiências de participação política online em políticas públicas*. 2015. 39º Encontro Anual da Associação Nacional de Pós-Graduação e Pesquisa em Ciências Sociais.

MongoDB. 2016. Disponível em: <https://www.mongodb.com>. Acesso em Maio 20, 2016.

NARSUDE, C. *Real-Time Event Stream Processing ? What are your choices?* 2015. Disponível em: <https://www.datatorrent.com>. Acesso em Maio 5, 2016.

ONU. *World Urbanization Prospects The 2014 Revision*. 2014. Disponível em: <http://esa.un.org>. Acesso em Maio 10, 2016.

PRIBERAM. *Definição de deliberação*. 2016. Disponível em: <https://www.priberam.pt>. Acesso em Maio 14, 2016.

RUOHONEN, K. *Graph Teory*. 2013. Disponível em: <http://math.tut.fi/~ruohonen/GT_English.pdf>. Acesso em Maio 01, 2016.

SANDU, D. *Without stream processing, there?s no big data and no Internet of things*. 2014. Disponível em: <http://venturebeat.com>. Acesso em Maio 7, 2016.

SAÉZ-MARTÍN, A.; ROSSARIO, A. Haro-de; CABA-PEREZ, C. A vision of social media in the spanish smartest cities. *Transforming Government: People, Process and Policy*, v. 8, n. 4, 2014.

SHIN, K. G.; RAMANATHAN, P. Real-time computing: A new discipline of computer science and egineering. *Procedings og the IEEE*, v. 82, n. 1, 1994.

Siciliane, T. *Streaming Big Data: Storm, Spark and Samza*. 2015. Disponível em: <https://dzone.com>. Acesso em Maio 7, 2016.

SILVA, M. et al. *SentiLex-PT 01*. 2016. Disponível em: <http://dmir.inesc-id.pt/project/SentiLex-PT_01>. Acesso em Maio 21, 2016.

SOMMERVILLE, I. *Engenharia de Software*. 9th. ed. [S.l.]: Pearson, 2011.

Stack Over Flow. 2016. Disponível em: <https://http://stackoverflow.com>. Acesso em Maio 7, 2016.

Techopedia. 2016. Disponível em: <https://www.techopedia.com>. Acesso em Abril 23, 2016.

TREAT, T. *Stream Processing and Probabilistic Methods: Data at Scale*. 2015. Disponível em: <http://bravenewgeek.com>. Acesso em Maio 7, 2016.

TWITTER. *Twitter Developers*. 2016. Disponível em: <https://dev.twitter.com>. Acesso em Maio 20, 2016.

TWITTER4J. 2016. Disponível em: <http://twitter4j.org/>. Acesso em Maio 20, 2016.

WÄHNER, K. *Real-Time Stream Processing as Game Changer in a Big Data World with Hadoop and Data Warehouse*. 2014. Disponível em: <http://www.infoq.com>. Acesso em Maio 7, 2016.

XU, L.; JU, H.; REN, H. *Gitbook: SparkInternals*. 2015. Disponível em: <https://github.com/JerryLead/SparkInternals>. Acesso em Abril 30, 2016.

ZAHARIA, M. et al. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction forIn-Memory Cluster Computing*. 2012. Disponível em: <http://www-bcf.usc.edu/~minlanyu/teach/csci599-fall12/papers>. Acesso em Maio 23, 2016.

ZAHARIA, M. et al. *Discretized Streams: A Fault Tolerant Model for Scalable Stream Processing*. 2012. Disponível em: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2012>. Acesso em Maio 22, 2016.

ZAHARIA, M. et al. *Discretized Streams: An Efficient and Fault-Tolerant Model forStream Processing on Large Clusters*. 2012. Disponível em: <http://www.eecs.berkeley.edu/~haoyuan/papers>. Acesso em Maio 23, 2016.

ZAHARIA, M. et al. *Discretized Streams: Fault Tolerant Streaming Computation at Scale*. 2013. Disponível em: <https://people.csail.mit.edu/matei/papers/2013>. Acesso em Maio 22, 2016.

ZAPLETAL, P. *Introduction Into Distributed Real-Time Stream Processing*. 2015. Disponível em: <http://www.cakesolutions.net>. Acesso em Maio 7, 2016.

APÊNDICE A – Primeiro apêndice

Os apêndices são textos ou documentos elaborados pelo autor, a fim de complementar sua argumentação, sem prejuízo da unidade nuclear do trabalho.

ANEXO A – Primeiro anexo

Os anexos são textos ou documentos não elaborado pelo autor, que servem de fundamentação, comprovação e ilustração.