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

Título do trabalho

Felipe Cordeiro Alves Dias

Natal-RN
Mês (por extenso) e ano

Felipe Cordeiro Alves Dias

Título do trabalho

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 Ciência da Computação.

Orientador(a)

Nome e titulação do(a) professor(a) orientador(a)

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

Natal-RN

Mês (por extenso) e ano

Monografia de Graduação sob o título *Título da monografia* apresentada por Nome do aluno 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 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

Título do trabalho

Autor: Felipe Cordeiro Alves Dias
Orientador(a): Titulação e nome do(a) orientador(a)

Resumo

O resumo deve apresentar de forma concisa os pontos relevantes de um texto, fornecendo uma visão rápida e clara do conteúdo e das conclusões do trabalho. O texto, redigido na forma impessoal do verbo, é constituído de uma seqüência de frases concisas e objetivas e não de uma simples enumeração de tópicos, não ultrapassando 500 palavras, seguido, logo abaixo, das palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores. Por fim, deve-se evitar, na redação do resumo, o uso de parágrafos (em geral resumos são escritos em parágrafo único), bem como de fórmulas, equações, diagramas e símbolos, optando-se, quando necessário, pela transcrição na forma extensa, além de não incluir citações bibliográficas.

Palavras-chave: Palavra-chave 1, Palavra-chave 2, Palavra-chave 3.

Título do trabalho (em língua estrangeira)

Author: Nome do aluno
Advisor: Titulação e nome do(a) orientador(a)

Abstract

O resumo em língua estrangeira (em inglês *Abstract*, em espanhol *Resumen*, em francês *Résumé*) é uma versão do resumo escrito na língua vernácula para idioma de divulgação internacional. Ele deve apresentar as mesmas características do anterior (incluindo as mesmas palavras, isto é, seu conteúdo não deve diferir do resumo anterior), bem como ser seguido das palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores, na língua estrangeira. Embora a especificação abaixo considere o inglês como língua estrangeira (o mais comum), não fica impedido a adoção de outras linguas (a exemplo de espanhol ou francês) para redação do resumo em língua estrangeira.

Keywords: Keyword 1, Keyword 2, Keyword 3.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

RDD – Resilient Distributed Dataset

DStream – Discretized Stream

FIFO – First In, First Out

DAG – Directed Acyclic Graph

DAG – Directed Acyclic Graph

UFRN – Universidade Federal do Rio Grande do Norte

DIMAp – Departamento de Informática e Matemática Aplicada

YARN – Yet Another Resource Negotiator

Lista de símbolos

λ (algum símbolo)

Sumário

1 Introdução

A introdução é a parte inicial do texto e que possibilita uma visão geral de todo o trabalho, devendo constar a delimitação do assunto tratado, objetivos da pesquisa, motivação para o desenvolvimento da mesma e outros elementos necessários para situar o tema do trabalho.

1.1 Organização do trabalho

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

2 Fundamentação Teórica

Este é o primeiro capítulo da parte central do trabalho, isto é, o desenvolvimento, a parte mais extensa de todo o trabalho. Geralmente o desenvolvimento é dividido em capítulos, cada um com subseções e subseções, cujo tamanho e número de divisões variam em função da natureza do conteúdo do trabalho.

Em geral, a parte de desenvolvimento é subdividida em quatro subpartes:

- *contextualização ou definição do problema* – consiste em descrever a situação ou o contexto geral referente ao assunto em questão, devem constar informações atualizadas visando a proporcionar maior consistência ao trabalho;
- *referencial ou embasamento teórico* – texto no qual se deve apresentar os aspectos teóricos, isto é, os conceitos utilizados e a definição dos mesmos; nesta parte faz-se a revisão de literatura sobre o assunto, resumindo-se os resultados de estudos feitos por outros autores, cujas obras citadas e consultadas devem constar nas referências;
- *metodologia do trabalho ou procedimentos metodológicos* – deve constar o instrumental, os métodos e as técnicas aplicados para a elaboração do trabalho;
- *resultados* – devem ser apresentados, de forma objetiva, precisa e clara, tanto os resultados positivos quanto os negativos que foram obtidos com o desenvolvimento do trabalho, sendo feita uma discussão que consiste na avaliação circunstanciada, na qual se estabelecem relações, deduções e generalizações.

É recomendável que o número total de páginas referente à parte de desenvolvimento não ultrapasse 60 (sessenta) páginas.

2.1 Cidades Inteligentes

2.2 Plataformas de processamento em tempo real

2.3 Apache Spark

Apache Spark (Apache Software Foundation, 2016c) 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 stream e aprendizado de máquina. Tendo Java, Python e Scala como linguagens de programação suportadas.

A arquitetura do Spark, conforme a Fig. 1, 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, 2016a) e Apache Mesos (Apache Software Foundation, 2016b)). Tal estrutura visa ser de fácil manutenção, teste, deploy, e permitir aumento de perfomance do núcleo impactando seus demais componentes.

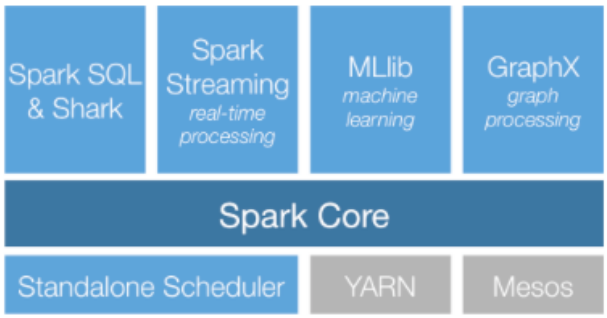

Figura 1: Ilustração da arquitetura em pilha do Apache Spark
Fonte: (KARAU et al., 2015)

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 (Resilient Distributed Dataset) 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 parâme-

tro 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, 2016c), 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. 2. 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 ope-

rações 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, 2016c). 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).

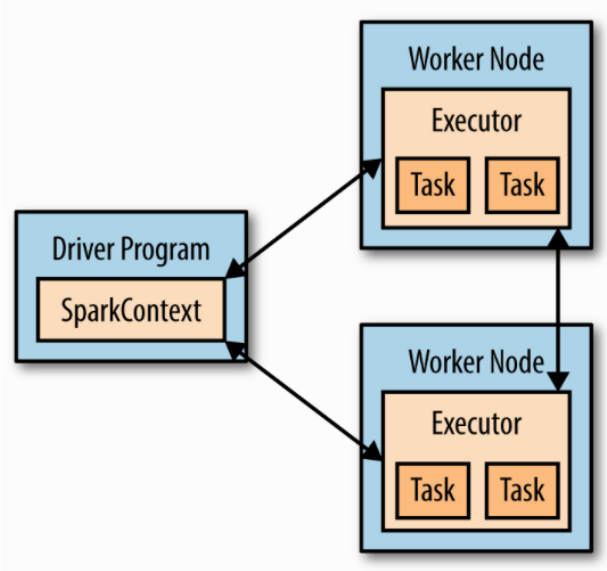

Figura 2: Componentes do Spark para execução distribuída
Fonte: (KARAU et al., 2015)

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 (Directed Acyclic Graph (RUOHONEN, 2013)) , 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, 2016d) 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, 2016e) (framework que permite o desenvolvimento de serviços multilinguagem) para definir e submeter topologias (conceito definido em 2.4.1.4) (Apache Software Foundation, 2016d).

A arquitetura do Storm, conforme a Fig. 3, é composta pelos seguintes componentes, a serem explicados posteriormente: Nimbus, ZooKeeper e Supervisor Node.

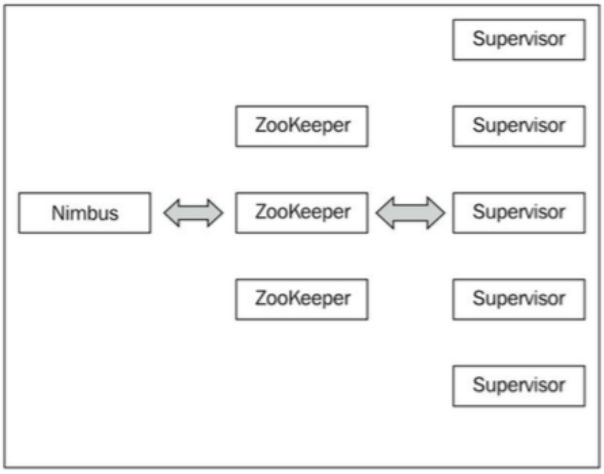

Figura 3: Ilustração da arquitetura do Apache Storm
Fonte: (JAIN; NALYA, 2014)

2.4.1 Composição interna do Apache Storm

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, 2016f) 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. 4, é uma abstração que define um grafo acíclico dirigido DAG (Directed Acyclic Graph (RUOHONEN, 2013)) , 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, 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.

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).

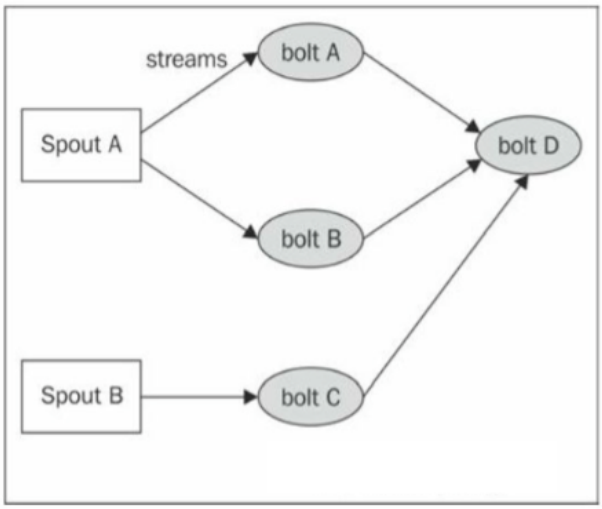

Figura 4: Exemplo de uma topologia do Storm
Fonte: (JAIN; NALYA, 2014)

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 pro-

cessamento, 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 Capítulo 3

Algumas regras devem ser observadas na redação da monografia:

1. ser claro, preciso, direto, objetivo e conciso, utilizando frases curtas e evitando ordens inversas desnecessárias;

2. construir períodos com no máximo duas ou três linhas, bem como parágrafos com cinco linhas cheias, em média, e no máximo oito (ou seja, não construir parágrafos e períodos muito longos, pois isso cansa o(s) leitor(es) e pode fazer com que ele(s) percam a linha de raciocínio desenvolvida);

3. a simplicidade deve ser condição essencial do texto; a simplicidade do texto não implica necessariamente repetição de formas e frases desgastadas, uso exagerado de voz passiva (como *será iniciado*, *será realizado*), pobreza vocabular etc. Com palavras conhecidas de todos, é possível escrever de maneira original e criativa e produzir frases elegantes, variadas, fluentes e bem alinhavadas;

4. adotar como norma a ordem direta, por ser aquela que conduz mais facilmente o leitor à essência do texto, dispensando detalhes irrelevantes e indo diretamente ao que interessa, sem rodeios (verborragias);

5. não começar períodos ou parágrafos seguidos com a mesma palavra, nem usar repetidamente a mesma estrutura de frase;

6. desprezar as longas descrições e relatar o fato no menor número possível de palavras;

7. recorrer aos termos técnicos somente quando absolutamente indispensáveis e nesse caso colocar o seu significado entre parênteses (ou seja, não se deve admitir que todos os que lerão o trabalho já dispõem de algum conhecimento desenvolvido no mesmo);

8. dispensar palavras e formas empoladas ou rebuscadas, que tentem transmitir ao leitor mera idéia de erudição;

9. não perder de vista o universo vocabular do leitor, adotando a seguinte regra prática: *nunca escrever o que não se diria*;

10. termos coloquiais ou de gíria devem ser usados com *extrema* parcimônia (ou mesmo nem serem utilizados) e apenas em casos muito especiais, para não darem ao leitor a idéia de vulgaridade e descaracterizar o trabalho;

11. ser rigoroso na escolha das palavras do texto, desconfiando dos sinônimos perfeitos ou de termos que sirvam para todas as ocasiões; em geral, há uma palavra para definir uma situação;

12. encadear o assunto de maneira suave e harmoniosa, evitando a criação de um texto onde os parágrafos se sucedem uns aos outros como compartimentos estanques, sem nenhuma fluência entre si;

13. ter um extremo cuidado durante a redação do texto, principalmente com relação às regras gramaticais e ortográficas da língua; geralmente todo o texto é escrito na forma impessoal do verbo, não se utilizando, portanto, de termos em primeira pessoa, seja do plural ou do singular.

3.1 Seção 1

Teste de uma tabela:

Tabela 1: Tabela sem sentido

Titulo Coluna 1	Título Coluna 2
X	Y
X	W

3.2 Seção 2

Seção 2

3.2.1 Subseção 2.1

Referência à tabela definida no início: 3.1

3.2.2 Subseção 2.2

Subsection 2.2

3.3 Seção 3

Seção 3

4 Capítulo 4

4.1 Seção 1

Teste para símbolo

λ

4.2 Seção 2

Teste para abreviatura

UFRN

DIMAp

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 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 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.

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

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.

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

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

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