ESTUDO DA PLATAFORMA DE COMPUTAÇÃO EM *CLUSTER* APACHE SPARK VOLTADA AO *MACHINE LEARNING*

ARMINDO, Thiado de Almeida
Orientador: DEZANI, Henrique

e-mail:
thiago.armindo@fatec.sp.gov.br; henrique.dezani@fatec.sp.gov.br

Resumo: Neste projeto teve-se como proposta o estudo da ferramenta Apache Spark aplicada à computação em *cluster*, em especial na área de *Machine Learning,* em conjunto com o recurso de *streaming* de dados oferecido pelo Spark, a fim de realizar a extração, transformação e visualização dos dados em tempo real. Foram analisados os componentes da arquitetura do Apache Spark e seu funcionamento; em seguida, foi desenvolvida uma aplicação simples, aplicando na prática os conceitos estudados.

Palavras-chave: Spark. *Machine Learning*. Computação em *cluster*. Fluxo de dados.

Abstract: *In this project, the proposal was to study the Apache Spark tool applied to cluster computing, especially in the area of Machine Learning, together with the data stream feature offered by Spark, in order to perform the extraction, transformation and visualization of the data. real-time data. The components of the Apache Spark architecture and its functioning were analyzed, then a simple application was developed, applying the studied concepts in practice.*

Keywords: *Apache Spark, Machine Learning. Cluster computing. Data Streaming.*

1. Introdução

A computação em *cluster* surgiu como uma alternativa comercialmente viável para o paralelismo, tendo início na década de 1960 com a IBM. C*luster* é um termo em inglês que significa “aglomerar” ou “aglomeração” e pode ser aplicado em vários contextos. No caso da computação, o termo define uma arquitetura de sistema capaz combinar vários computadores para trabalharem em conjunto ou pode denominar o grupo em si de computadores combinados (GOMES, 2015).

Cada computador é denominado “nó” e, combinados, formam o *cluster*. Em alguns casos, é possível ver referências como “supercomputadores” ou “computação em cluster” para o mesmo cenário, representando o *hardware* usado ou o *software* especialmente desenvolvido para conseguir combinar esses equipamentos.

O Apache Spark se enquadra em um desses *softwares* para realização da clusterização. Ele é um *framework* de código aberto, desenvolvido pela universidade de Barkeley em 2009, cujo objetivo é processar grandes conjuntos de dados de forma paralela e distribuída, sendo algumas de suas funções o gerenciamento de *Big Data*, aplicações de *Machine Learning*, *Streaming* de Dados, SQL e processamento de Grafos.

2. Justificativa

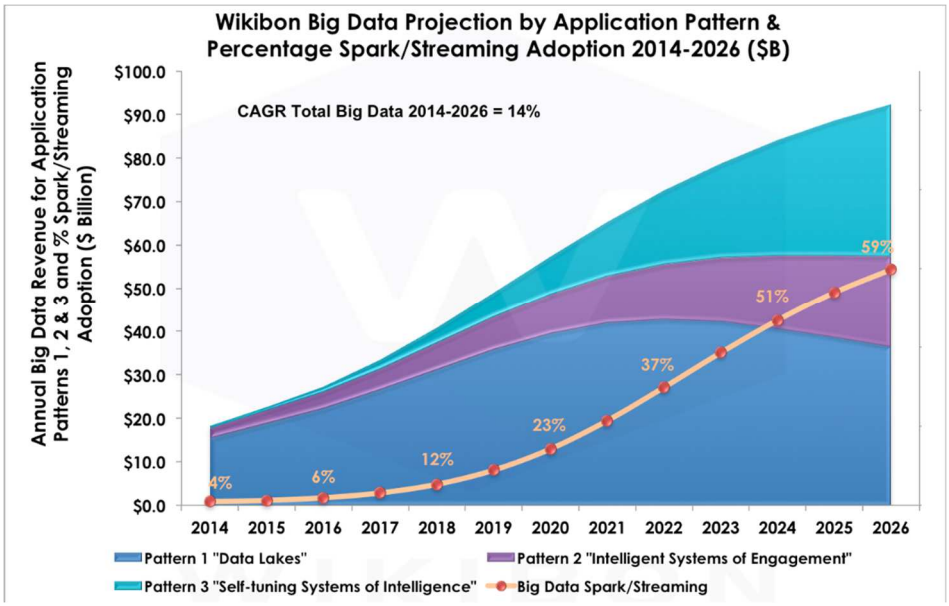
A popularização do uso de dispositivos eletrônicos no dia a dia, tais como *smartphones*, *notebooks* e, recentemente, dispositivos de IoT (*Internet of Things*), permitiu a geração de grande quantidade de dados. “De acordo com o Instituto Gartner, até 2020 é possível que haja um total de 40 trilhões de *gigabytes* de dados no mundo. Isso significa 2,2 milhões de *terabytes* de novos dados gerados todos os dias.” (NAVITA, 2019).

Com toda essa quantidade de dados, houve a possibilidade de usá-los para diversos fins, tais como, estudo do perfil de seus clientes, no setor de saúde, reconhecimento de padrões para identificar doenças, no setor ambiental, identificação dos sinais da natureza a fim de prever possíveis desastres naturais. Segundo o CEO da Mastercard em entrevista para a revista Época Negócios (2019), os dados são o novo petróleo.

Com um grande conjunto de dados, é necessário um grande poder computacional, ou seja, quanto mais dados para serem analisados, mais precisos serão os modelos de aprendizado de máquina criados. O Apache Spark possui inerente ao seu ecossistema o recurso de clusterização do processamento de modelos de *Machine Learning*, distribuindo o peso dessa tarefa em vários núcleos de processamento, que podem ser processadores físicos ou virtuais. “Potências da Internet como Netflix, Yahoo e eBay o implementaram em escala maciça, processando coletivamente múltiplos *petabytes* de dados em clusters de mais de 8.000 nós. Ele rapidamente se tornou a maior comunidade *open* source em *Big Data*, com mais de 1000 colaboradores de mais de 250 organizações”. (GARCIA, 2020)

No gráfico apresentado na Figura 1, pode-se observar a projeção de crescimento da adoção do Spark, e a equivalência em bilhões de dólares da receita de *Big Data* gerada por ele e seus concorrentes em padrões de aplicação. Nota-se que seu crescimento é exponencial, chegando a 59% em 2026 e uma receita de aproximadamente 50 bilhões de dólares (WHEATLEY, 2016).

Figura 1: Projeção de *Big Data* por padrão de aplicativo e porcentagem de adoção do Spark/*Streaming* de 2014-2026 em Bilhão de dólares



Fonte: Wheatley (2016)

Portanto, seu estudo se torna relevante para entender seu funcionamento, e as possibilidades que a ferramenta fornece, sendo também um conhecimento importante no âmbito profissional, tendo em vista sua força e presença nas empresas de diversos setores.

3. Objetivos

3.1 Objetivo Geral

- Estudar e analisar os recursos e arquitetura da ferramenta Apache Spark para computação em *cluster,* aplicado ao processamento de *Machine Learning*, com utilização de *streaming* de dados.

3.2. Objetivos específicos

- Estudar os conceitos teóricos de computação em *cluster*;
- Estudar e analisar a ferramenta Apache Spark de maneira geral;
- Estudar a arquitetura de processamento do Apache Spark;
- Estudar o recurso de *streaming* de dados do Spark Streaming;
- Implantar um *cluster* do Apache Spark;
- Desenvolver um modelo de *machine learning,* para processamento de linguagem natural, utilizando o Spark Streaming para obtenção dos dados;
- Processar o modelo de *machine learning* através do *cluster* Spark;
- Identificar qualidades e problemas enfrentados durante o desenvolvimento.

4. Fundamentação Teórica

4.1 Computação em *cluster*

Um *cluster* de computadores é definido quando existem duas ou mais máquinas trabalhando em conjunto para o processamento de requisições, a palavra vem do inglês e significa agrupamento, cada computador ou processador presente no cluster é chamado “nodo”.

Apesar de seu surgimento na década de 1960 com a IBM, a tecnologia só começou a ganhar espaço e importância a partir dos anos 80, de acordo com Pitanga (2008), o crescimento do uso da tecnologia foi motivado por três fatores: a construção de processadores de alto desempenho, o surgimento de redes de comunicação de baixa latência e a padronização de ferramentas para computação paralela e distribuída.

Os *clusters* podem ser divididos em duas categorias básicas: Alta Disponibilidade e Alto Desempenho de Computação. Sendo que o primeiro é projetado para prover disponibilidade a uma aplicação pelo maior tempo e da forma mais segura possível, já o segundo, tem a finalidade de propiciar um alto poder computacional, com sua programação conciliando a capacidade de processamento de cada computador, criando um super computador.

O *cluster* de Alto Desempenho, é o tipo necessário para o processamento do algoritmo de *Machine Learning* estudado nesse trabalho, com ele é possível executar aplicações e processamentos que requerem alto desempenho, utilizando equipamentos comuns, e distribuindo o processamento entre os nodos.

4.2 Aprendizado de Máquina (*Machine Learning*)

Aprendizado de máquina ou *machine learning*, é a ciência da programação de computadores para que eles possam aprender com os dados. De acordo com Géron (2019), nos anos 1959, Arthur Samuel definiu o *machine learning* como "O campo de estudo que dá aos computadores a habilidade de aprender sem ser explicitamente programado", posteriormente em 1997, Tom Mitchell definiu que é a técnica em que "Um programa de computador aprende pela experiência E em relação a algum tipo de tarefa T e alguma medida de desempenho P se o seu desempenho em T, conforme medido por P, melhora com a experiência E".

Portanto, ela consiste em elaborar algoritmos capazes de definir padrões que possam ser seguidos para se alcançar esse objetivo, porém, diferentemente dos algoritmos tradicionais, em que esses padrões são fixos e definidos no momento da codificação pelos desenvolvedores, nos de *machine learning* são definidos parâmetros e inseridos dados para que a máquina treine e analise os mesmos, encontre os padrões, e defina a importância de cada um deles para prever ou identificar a informação desejada.

4.3 Apache Spark

De acordo com a documentação do Apache Spark (2021), ele é um mecanismo de análise unificado para processamento de dados em grande escala, fornecendo APIs de alto nível em Java, Scala, Python e R e um mecanismo otimizado que oferece suporte a gráficos de execução geral. Ele também oferece suporte a um rico conjunto de ferramentas de nível superior, incluindo Spark SQL para interpretação e execução de SQL e processamento de dados estruturados, MLlib para aprendizado de máquina, GraphX para processamento gráfico e Streams estruturados para computação incremental.

Segundo Damji *et al.* (2020), o Apache Spark provê armazenamento em memória para intermediar o processamento, fazendo com que ele seja muito mais rápido que o seu antecessor Hadoop MapReduce. O Apache Spark teve sua filosofia de design centrado em quatro características chave, quais sejam, velocidade, facilidade de uso, modularidade e extensibilidade.

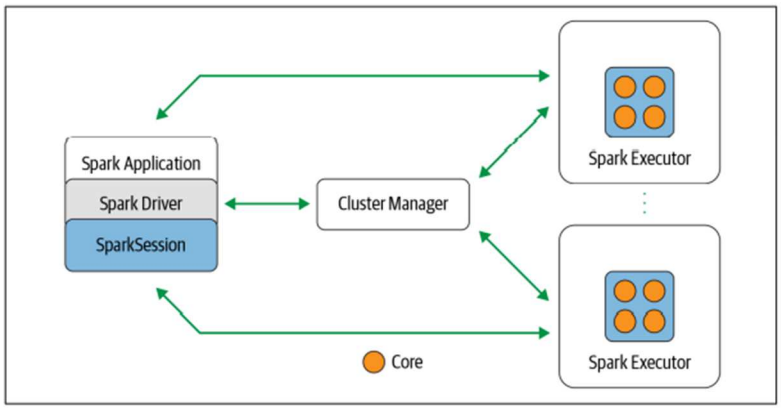
4.3.1 Arquitetura do Apache Spark

O funcionamento do Spark é baseado no processamento distribuído dos dados, através dos componentes funcionando cooperativamente em um *cluster*, como é definido por Damji *et al.* (2020). Uma aplicação Spark consiste em um programa *Driver* que é responsável por orquestrar as operações paralelas no cluster Spark. O *Driver* acessa os componentes do *cluster*,

sendo eles o *Cluster Manager* e o Spark *Executors*, através de uma *SparkSession*, e distribui as operações que estão sendo requisitadas pela aplicação.

Na Figura 2 é possível visualizar a arquitetura de um *cluster* Spark, em que a aplicação Spark inicia um Spark *Driver* através de uma *SparkSession*, o *Driver* então se comunica com o *Cluster Manager* que aciona os Spark *Executors*, e estes, por sua vez, realizam o processamento solicitado e retornam as saídas para aplicação.

Figura 2: Componentes do Apache Spark



Fonte: Damji *et al* (2020)

O Spark possuí suporte para uma variedade de modos de implantação, segundo Damji *et al*. (2020), ele pode ser executado em diferentes configurações e ambientes, contanto que o *cluster manager* possa acessar os recursos do sistema e os spark *executors*, os modos de implantação disponíveis atualmente são: Local, *Standalone*, YARN cliente, YARN *cluster* e Kubernetes.

4.3.2 Spark Driver

De acordo com Damji *et al* (2020), o *Driver* é responsável por instanciar o *SparkSession*. Ele exerce múltiplas funções, como a comunicação com o *cluster manager*, requerimento de recursos (CPU, memória, etc.) do *cluster manager* para os *executors*, e transforma as operações em processamento DAG, em seguida ele agenda a execução deste processamento, e distribui suas execuções em forma de tarefas para os *executors*.

4.3.3 SparkSession

Como é definido por Damji *et al* (2020), o *SparkSession* é um canal unificado para todas as operações e processamento de dados do Spark, essa função surgiu no Spark 2.0, que tornou o trabalho com o Spark mais simples e fácil, pois ele agrupou as funções de entradas existentes (*SparkContext*, *SQLContext*, *HiveContext*, *SparkConf* e *StreamingContext*) em uma única função. Aos instanciar o *SparkSession*, todas as funcionalidades das funções de entradas anteriores, ficam disponíveis para uso.

4.3.4 Cluster Manager

O *cluster manager*, conforme Damji *et al* (2020), é responsável por gerenciar e alocar os recursos para o *cluster* de cada aplicação Spark. Atualmente o Spark oferece suporte a quatro diferentes *clusters managers*, quais sejam, o interno *standalone*, Apache Hadoop YARN, Apache Mesos, e Kubernetes.

4.3.5 Spark Executors

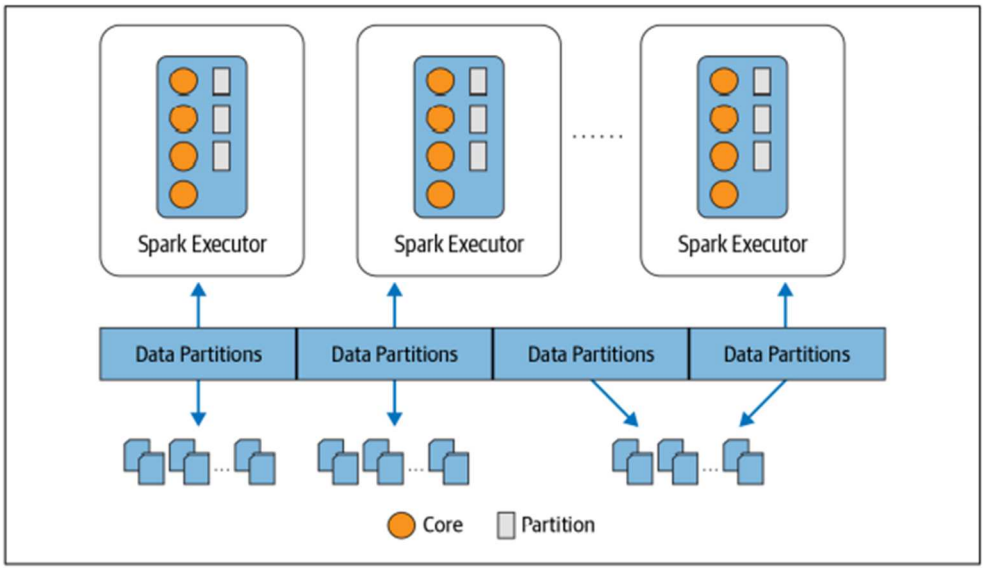
Segundo Damji *et al* (2020), o Spark *executor* é executado em cada nó de trabalho de um *cluster*, ele se comunica com o *driver*, sendo responsável por executar as tarefas que são atribuídas aos nós de trabalho, na maioria das implantações, um único executor é utilizado em cada nó.

4.3.6 Partições e dados distribuídos

Os dados físicos reais são distribuídos pelo armazenamento como partições em esquemas HDFS ou armazenamento em nuvem, de acordo com Damji *et al* (2020), enquanto os dados são distribuídos como partições ao longo do *cluster* físico, o Spark trata cada partição como uma abstração lógica de alto nível, como por exemplo um *DataFrame* na memória e, sempre que possível, cada executor do Spark é designado para realizar uma tarefa que exige a leitura da partição mais próxima a ele na rede de trabalho, observando a localidade dos dados.

Na Figura 3 é exemplificado como os *executors* interagem com as partições de dados, de acordo com Damji *et al* (2020), o particionamento é o que permite um paralelismo eficiente, distribuindo os dados em partições, os *executors* somente processam os dados que estão próximos deles, minimizando a largura de banda.

Figura 3: Atribuição dos *executors* de acordo com a proximidade da partição



Fonte: Damji *et al* (2020)

4.4 Spark Streaming

Com o advento dos sistemas de Big Data, a quantidade de dados emitida por sensores, sites, e dispositivos IoT aumentou exponencialmente e, com isso, a necessidade de se utilizar e analisar estes dados em tempo real se tornou um recurso chave. Segundo Aven (2017), por volta de 2008, com o surgimento das redes sociais, como Facebook e Twitter, a empresa BackType, especializada em *social media* e inteligência de mercado, iniciou um novo projeto para processamento distribuído de *streaming* de dados, chamado Storm, que era integrado ao ecossistema do Hadoop e que rapidamente se tornou líder no segmento *open source* de processamento de dados em tempo real.

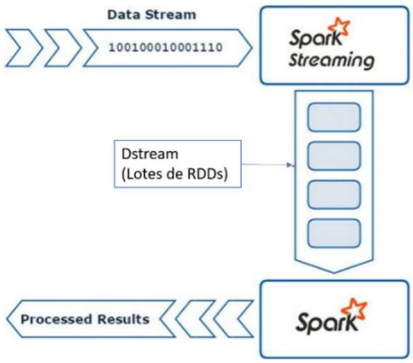
Com isso, os fundadores do Spark, enxergaram a necessidade de desenvolver uma solução para processamento de *stream* dentro do projeto Spark, porém, eles queriam um sistema tolerante a falhas e com garantia de integridade dos dados, fatores que eram limitantes no Storm. A abordagem do Spark foi de entregar um sistema tolerante a falhas e que garantia que cada evento seria processado exatamente uma vez, mesmo se um nó falhasse, além disso, o sistema deveria ser integrado ao seu *framework* baseado em *batch* RDD, e assim surgiu o Spark Streaming. Aven (2017) também complementa que, o design do Spark Streaming foi desenvolvido visando quatro objetivos: Baixa latência (escala de segundos), processamento de um evento somente uma vez, escalabilidade linear e integração com o Spark Core API.

4.4.1 Arquitetura do Spark Streaming

De acordo com Aven (2017), a arquitetura do Spark Streaming introduziu o conceito de *discretized streams* (DStreams), que são lotes de dados armazenados em múltiplos RDDs, cada lote representa uma janela de tempo, normalmente em segundos, em que os dados são coletados. Os RDDs resultantes podem ser processados utilizando as APIs do Apache Spark.

Na Figura 4 é possível ter uma visão geral do funcionamento do Spark Streaming, em que o módulo de Streaming recebe a *stream* de dados, em seguida armazena isso em lotes de RDDs divididos de acordo com a janela de tempo que foi definida, em seguida passa pelo processamento do Spark, e retorna os resultados processados.

Figura 4: Visão geral em alto nível da arquitetura do Spark Streaming



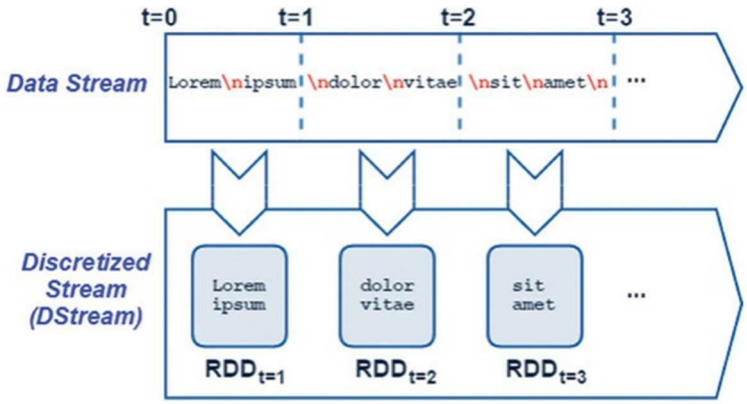
Fonte: Aven (2017)

4.4.2 Discretized Streams (DStreams)

Segundo Aven (2017), DStreams são os objetos básicos de programação do Spark Streaming API e representam uma sequência continua de RDDs, os quais são criados através do *stream* de dados, sendo cada RDD representante de uma janela de tempo. DStreams podem ser criadas a partir de diferentes fontes de *stream* de dados, como *sockets* TCP, sistemas de mensageria, e APIs de *streaming* (como a do Twitter). DStreams suportam dois tipos de operações: operações de transformação e operações de saída.

Na Figura 5 é apresentada a estrutura da DStream, nos quais os dados são recebidos através da fonte de *stream* de dados, de acordo com o intervalo escolhido (representado pela letra *t*), em seguida a DStream os tranforma em um lote RDD que armazena os dados equivalente a janela de tempo. Para cada janela de tempo haverá um RDD armazenando seus dados recebidos.

Figura 5: Estrutura da DStream



Fonte: Aven (2017)

4.5 Natural Language Toolkit (NLTK)

O Natural Language Toolkit (NLTK), de acordo com a sua documentação (2021), pode ser definido como uma plataforma para construção de programas Python para trabalhar com dados de linguagem humana. Ele fornece mais de 50 córpus e recursos lexicais, em conjunto com um pacote de bibliotecas de processamento de texto para classificação, tokenização, lematização, marcação, análise e raciocínio semântico e processamento de linguagem natural.

5. Trabalhos Similares

No trabalho de Carvalho (2018), intitulado "Uma análise comparativa de ambientes para Big Data: Apache Spark e HPAT", o autor realizou uma comparação de desempenho do Apache Spark e seu concorrente High Performance Analytic Toolkit (HPAT), a partir dos testes de dois algoritmos, um para soma dos elementos de um vetor unidimensional e o outro sendo o *K-means*, sendo este último utilizado para identificar diferentes grupos de dados, muito utilizado

na área de *machine learning*. Seu objetivo foi descobrir se o HPAT é uma alternativa, em desempenho, ao Apache Spark, levando em consideração dois ambientes distintos de trabalho. Ainda segundo o autor, HPAT obteve um desempenho melhor do que o Apache Spark em um ambiente sem falhas, porém, o Spark teve um desempenho melhor em um ambiente com falhas.

Este comportamento encontrado por Carvalho, reforça ainda mais os conceitos do Apache Spark apresentados neste trabalho, pois, no desenvolvimento do Spark, sempre foi uma grande preocupação a tolerância a falhas, o que pode tornar o seu processamento um pouco mais lento em certos cenários, porém, sempre será muito confiável.

6. Metodologia

A pesquisa contempla um estudo teórico sobre a arquitetura e funcionalidades da ferramenta Apache Spark, com foco no recurso de *streaming* de dados, integrado ao *machine learning*.

Os dados utilizados contêm informações sobre mensagens enviadas pelo aplicativo Twitter, os quais, após serem extraídos, passaram pelo algoritmo de análise de sentimento, que determinou se aquele *tweet* tinha um teor positivo ou negativo sobre o termo pesquisado.

Inicialmente, foi analisado a arquitetura do cluster Spark, identificando seus componentes, e como eles foram desenvolvidos para otimizar o processamento distribuído dos dados. Após a compreensão de seu funcionamento, foi implantado e configurado um *cluster* local para processamento do modelo de *machine learning* que foi criado.

Com o *cluster* implantado, foi possível desenvolver a *pipeline* de tratamento de dados utilizando o Spark Streaming para obtenção e transformação dos dados, os quais foram obtidos através de requisições à API do Twitter.

Tendo como fonte esses dados obtidos e tratados, foi desenvolvido um modelo de *machine learning* utilizando algoritmo de processamento de linguagem natural, para análise de sentimento das mensagens enviadas pelo Twitter.

Após esse desenvolvimento, o modelo foi submetido ao *cluster* Spark para realizar o processamento de dados em tempo real, de forma distribuída, sendo que a métrica utilizada para a verificação da capacidade do Spark coletar, processar e retornar os dados em tempo quase real consiste na determinação do seu tempo de atraso com relação a cada lote de *stream* de dados.

Em seguida, os resultados obtidos da classificação das mensagens, foram armazenados e exibidos em forma de gráficos, também atualizados em tempo quase real.

Foi utilizado a linguagem de programação Python para codificação, um *notebook* com sistema operacional Linux para executar o *cluster* Spark, *framework* Apache Spark para processamento em *cluster* dos dados, API Apache Streaming para obtenção dos dados em tempo real, e realizar os tratamentos necessários, API do Twitter como fonte de dados, biblioteca NLTK para criação dos modelos de *machine learning*, e o Jupyter Notebook como ambiente de desenvolvimento.

7. Desenvolvimento

Para realizar o desenvolvimento da aplicação Spark foi utilizado o arquivo no formato *notebook* (.ipynb), através do Jupyter Notebook. O primeiro passo foi realizar a importação dos módulos e pacotes necessários para aplicação. Em seguida, como é exibido na Figura 6, foi criado um objeto *StreamingContext*, que recebe como argumento uma *SparkSession* e um intervalo de tempo para a coleta dos lotes de dados (*batch*). O *batch* foi definido como 5 segundos.

Figura 6: Definição do intervalo de tempo dos lotes de dados e inicialização do *StreamingContext*

```
# Frequencia de batch - 5 seg
INTERVALO_BATCH = 5

# Criando o StreamingContext
ssc = StreamingContext(sc, INTERVALO_BATCH)
```

Fonte: Autor (2021)

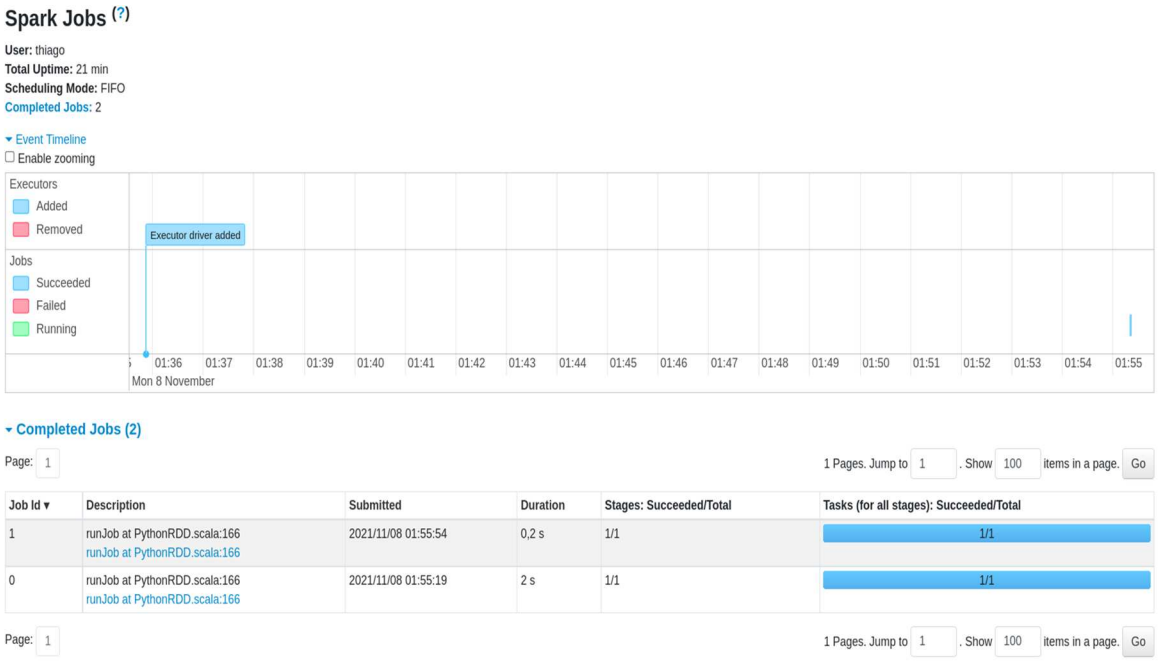
Com o *StreamingContext* inicializado, já foi possível utilizar o processamento do Spark, e, dessa forma, foi realizada a leitura de um conjunto de dados que continha *tweets* em inglês e suas classificações entre positivas e negativas. Este conjunto de dados foi utilizado para treinar o algoritmo de *machine learning*.

Após este procedimento, foi necessário instanciar o objeto *SentimentAnalyzer*, responsável por criar um modelo de *machine learning* para análise dos *tweets*. Em seguida, foi necessário realizar o *download* do pacote de *stopwords* em inglês da biblioteca NLTK, e criar uma lista com estas palavras.

Por fim, foram utilizados 10 mil *tweets* como base de treino para o modelo. A partir do momento em que a *SparkSession* foi iniciada, foi possível acessar uma interface gráfica para o Spark através da porta *localhost*:4040. Esta interface possui um menu para exibição dos *Jobs*, *Stages*, *Storage*, *Environment*, *Executors*, *SQL* e *Streaming*.

Na Figura 7 são apresentadas as informações dos *Jobs* executados, ou que já foram concluídos pelos executores. Esta aba exibe uma linha do tempo com os eventos registrados pelo *cluster*. Além disso, também foi possível ver a hora de envio do *job*, sua duração, os *stages*, e as tarefas envolvidas em cada *stage*.

Figura 7: Interface gráfica de exibição dos *Jobs* no Spark



Fonte: Autor (2021)

O próximo passo foi a autenticação na API do Twitter, através de credenciais obtidas ao utilizar uma conta de desenvolvedor no site do Twitter. Após realizar a autenticação, foi possível definir um termo de busca para requisitar via URL para API.

Na Figura 8 é exibida a definição do número de *tweets* coletados, e a função responsável por fazer a coleta, essa função requisita os *tweets* via método *get*, passando como parâmetro a URL com o termo escolhido para busca, as credenciais de autenticação, e atribuição verdadeira ao *stream*. Para este teste, foi definido uma quantidade de 10 tweets coletados por cada *batch*.

Figura 8: Definição da quantidade de *tweets* coletados e função para coleta das mensagens do Twitter

```
# Total de tweets por update
NUM_TWEETS = 9
```

```
# Essa função conecta ao Twitter e retorna um número específico de Tweets (NUM_TWEETS)
def tfunc(t, rdd):
  return rdd.flatMap(lambda x: stream_twitter_data())

def stream_twitter_data():
  response = requests.get(filter_url, auth = auth, stream = True)
  print(filter_url, response)
  count = 0
  for line in response.iter_lines():
    try:
      if count > NUM_TWEETS:
        break
      post = json.loads(line.decode('utf-8'))
      contents = [post['text']]
      count += 1
      yield str(contents)
    except:
      result = False
```

Fonte: Autor (2021)

Após coletar os *tweets*, foi utilizada uma função para pegar apenas o texto da mensagem e o classificar utilizando a função classifica_tweet(), como é exibido na figura 9.

Figura 9: Função que classifica a mensagem e coleta apenas o texto do *tweet* coletado

```
# Essa função classifica os tweets, aplicando as features do modelo criado anteriormente
def classifica_tweet(tweet):
  sentence = [(tweet, '')]
  test_set = sentiment_analyzer.apply_features(sentence)
  print(tweet, classifier.classify(test_set[0][0]))
  return(tweet, classifier.classify(test_set[0][0]))
```

```
# Essa função retorna o texto do Twitter
def get_tweet_text(rdd):
  for line in rdd:
    tweet = line.strip()
    translator = str.maketrans({key: None for key in string.punctuation})
    tweet = tweet.translate(translator)
    tweet = tweet.split(' ')
    tweet_lower = []
    for word in tweet:
      tweet_lower.append(word.lower())
    return(classifica_tweet(tweet_lower))
```

Fonte: Autor (2021)

A figura 10 exibe a função responsável por chamar a função que limpa o texto do Twitter, em seguida o classifica e armazena em uma lista, e em um arquivo .csv, mantendo o padrão de primeira coluna sendo horário da mensagem, segunda coluna a quantidade de mensagens classificadas como positiva, e a terceira coluna a quantidade de mensagens negativas.

Figura 10: Função responsável por armazenar os resultados das mensagens classificadas

```
# Cria uma lista vazia para os resultados
resultados = []
```

```
# Essa função salva o resultado dos batches de Tweets junto com o timestamp
def output_rdd(rdd):
  global resultados
  pairs = rdd.map(lambda x: (get_tweet_text(x)[1],1))
  counts = pairs.reduceByKey(add)
  output = []
  for count in counts.collect():
    output.append(count)
  result = [time.strftime("%I:%M:%S"), output]
  resultados.append(result)
  with open('resultados.csv','a') as file:
    if len(resultados[-1][1]) >= 1:
        if resultados[-1][1][0][0] == '0':
            positivo = resultados[-1][1][0][1]
            if len(resultados[-1][1]) == 2:
                negativo = resultados[-1][1][1][1]
            else:
                negativo = 0
            horas = resultados[-1][0]
        else:
            negativo = resultados[-1][1][0][1]
            if len(resultados[-1][1]) == 2:
                positivo = resultados[-1][1][1][1]
            else:
                positivo = 0
            horas = resultados[-1][0]

        file.write(str(horas)+',')
        file.write(str(positivo)+',')
        file.write(str(negativo)+'\n')
  print(result)
```

Fonte: Autor (2021)

Após a definição dessas principais funções, é possível iniciar o *streaming* de dados, a figura 11 mostra a função que irá aplicar a função criada anteriormente (figura 10) em cada lote de dados recebido, e a função responsável por iniciar o *streaming* de dados.

Figura 11: Função que aplica a função output_rdd() para cada *batch* e inicia e *streaming* de dados

```
# A função foreachRDD() aplica uma função a cada RDD to streaming de dados
coord_stream.foreachRDD(lambda t, rdd: output_rdd(rdd))

# Start streaming
ssc.start()
# ssc.awaitTermination()
```

Fonte: Autor (2021)

Em um dos testes realizados, se utilizou o termo "Powell" para busca, esse termo estava em alta no Twitter no dia 22/11/2021, pois o presidente dos Estados Unidos, Joe Biden, renomeou Jerome Powell nesta data como *chair* do Federal Reserve (Banco central norte-americano) (Reuters, 2021). Os resultados de saída da aplicação podem ser vistos na Figura 12.

Figura 12: Resultados de saída da aplicação

```
https://stream.twitter.com/1.1/statuses/filter.json?track=Powell <Response [200]>
['rt', 'bbcworld', 'jerome', 'powell', 'nominated', 'to', 'stay', 'as', 'us', 'federal', 'reserve', 'chair', 'http
stcopzhk7ety8u'] 1
['rt', 'smartertrader', 'so', 'let', 'me', 'get', 'this', 'right', 'powell', 'says', 'no', 'hike', 'till', '2023',
'he', 'says', 'this', '50', 'times', 'and', 'every', 'day', 'on', 'cnbc', 'every', 'day', 'some', 'clo…'] 0
['rt', 'schuldensuehner', 'treasury', 'yields', 'rose', 'following', 'news', 'that', 'us', 'president', 'biden',
'picked', 'jerome', 'powell', 'for', '2nd', 'term', 'as', 'head', 'of', 'fed', '2y', 'amp', '5y…'] 0
['white', 'house', 'nominates', 'fed', 'chair', 'jerome', 'powell', 'for', '2nd', 'term', 'ending\xa0speculation',
'httpstcok2kmel51ag'] 0
['rt', 'tradestrey', 'powell', 'reelected', '\n\nguessing', 'biden', 'didn't', 'see', 'this', 'reference', 'on',
'his', 'resume', 'or', 'something', '\n\nall', 'the', 'cool', 'feds', 'insider', 'trade…'] 1
['rt', 'bitfurygeorge', 'buying', 'up', 'more', 'btc', 'at', '56k\nwhy', '', '\nthe', 'answer', 'is', 'simple',
'', 'powell', '20', 'will', 'bring', 'us', 'even', 'more', 'prrrrrrinting', 'there', 'is', 'no…'] 1
['rt', 'reuters', 'powell', 'tapped', 'for', 'second', 'term', 'as', 'fed', 'chair', 'brainard', 'as', 'vicechai
r', 'httpstcoxnifb1xqwp', 'httpstcoy5r1vijv3o'] 0
['rt', 'davidgura', 'as', 'president', 'biden', 'describes', 'how', 'federal', 'reserve', 'chair', 'jerome', 'powe
ll', 'stood', 'up', 'to', 'bidens', 'predecessor', 'recall', 'what', 'president…'] 0
['rt', 'formulascout', 'the', '2021', 'european', 'karting', 'calendar', 'concluded', 'last', 'weekend', 'with',
'italys', 'wsk', 'final', 'cup\n\nit', 'was', 'a', 'battle', 'between', 'f1', 'junior…'] 0
['bloomberg', 'us', 'chief', 'economist', 'anna', 'powells', 'reelection', 'nomination', 'means', 'the', 'continui
ty', 'of', 'fed', 'policy', 'because', 'he', 'w…', 'httpstcobec2qeyen0'] 1

['05:14:08', [('1', 4), ('0', 6)]]
```

Fonte: Autor (2021)

Na figura 12 pode-se observar a saída da aplicação, que é composta por uma lista com os *tweets* analisados, e um número na frente de cada um, o qual foi 0 quando o modelo identificou um sentimento negativo no *tweet*, ou 1 quando identificou sentimento positivo.

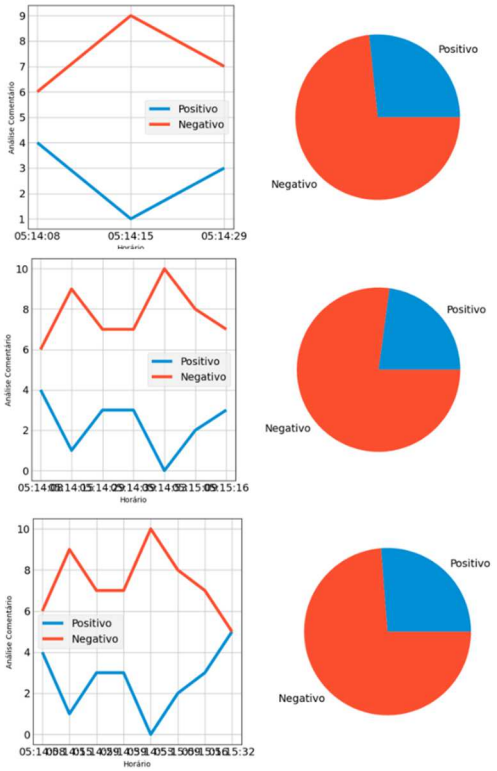
No menu de *Streaming*, é possível verificar informações estatísticas sobre o desempenho da execução da aplicação. Na Figura 13 a primeira estatística apresentada é a taxa de entrada (*Input Rate*), em seguida é o atraso de agendamento (*Scheduling Delay*). Essa métrica representa o tempo que o streaming leva para enviar o *job* de um *batch*. Após é exibido o tempo de processamento (*Processing Time*), que é o tempo necessário para realizar todos os processamentos em cada *batch*, e, por último, o atraso total (*Total Delay*), o qual contempla o tempo total para completar o processamento de cada *batch*.

Figura 13: Interface gráfica da estatística de desempenho do *streaming* no Spark

Fonte: Autor (2021)

Após realizar a extração, transformação e processamento, foi realizada a construção da visualização dos resultados obtidos, para isso, foi utilizado a biblioteca matplotlib, que permite a criação de gráficos atualizados em tempo real, a partir dos dados que estão sendo salvos. A figura 14 exibe a progressão do gráfico ao longo do tempo.

Figura 14: Gráfico gerado a partir dos resultados armazenados



Fonte: Autor (2021

O gráfico de linhas apresenta a variação da quantidade de comentários positivos e negativos (eixo y) pelo horário do lote de dados (eixo x), o gráfico de pizza representa a proporção entre a quantidade total de classificações positivas e negativas.

8. Resultados e Discussões

Ao longo da realização do projeto, e através das pesquisas realizadas, pode-se perceber a flexibilidade e utilidade do Apache Spark, ele conta com diversos recursos relacionados ao ecossistema de *Big Data*, sendo assim, a maioria dos processos de utilização de dados, terão funções que podem ser atendidas por alguma das APIs do Spark Core.

A possibilidade de escalonamento através da clusterização também é um ponto forte a se destacar, com ela é possível realizar um grande processamento, com um pequeno poder computacional, dividindo o processamento em etapas e distribuindo através dos nós do *cluster*, dessa forma, trazendo acessibilidade ao *Big Data*, para empresas de menor porte.

As maiores dificuldades encontradas ao longo do projeto, foram inicialmente realizar a instalação, devido algumas particularidades do sistema Linux, porém, o conteúdo sobre o tema é abundante na *web*, e após isso, a adaptação as funções próprias do Spark, foi crescente e exponencial, principalmente devido ao livro Learning Spark 2.0, que apresenta de forma clara e simples a utilização de suas funções.

Durante o desenvolvimento, a aplicação desenvolvida foi apenas um, dos muitos tipos de uso que o Spark Streaming pode fornecer, essa aplicação poderia por exemplo, verificar a popularidade de um politico em tempo real durante uma campanha, ou um *e-commerce* poderia utilizar para classificar os comentários sobre um produto assim que é lançada, ou ainda um *Call Center* poderia realizar o *streaming* de dados para monitorar os dados das ligações, enfim, as possibilidades são diversas.

9. Conclusões

Neste projeto, foi estudada a ferramenta Apache Spark e seu módulo Spark Streaming integrada ao *Machine Learning*, para extração, processamento e visualização de dados obtidos através da API do Twitter, durante o estudo da ferramenta, verificou-se a arquitetura dos componentes principais que integram tanto o Apache Spark, quanto o Spark Streaming, assim como, a forma que eles interagem entre si.

Desenvolveu-se uma aplicação para analisar os *tweets* coletados, e classificá-los entre sentimento positivo ou negativo, com relação ao termo pesquisado, para isso, utilizou-se a biblioteca NLTK, que permitiu a criação de um modelo de algoritmo de analise de sentimento. Os dados foram obtidos através da API do Twitter em tempo quase real utilizando o Spark Streaming, após a coleta e classificação dos *tweets*, os resultados foram armazenados, e foi construída a visualização em forma de gráficos dos resultados obtidos, os gráficos foram gerados de forma dinâmica, em tempo quase real.

A capacidade de processamento em *cluster* contribuiu para a possibilidade do processamento das informações, mesmo em um equipamento com menor poder computacional.

Tendo em vista todas as possibilidades que se tornam possíveis, graças as funcionalidades do Apache Spark, conclui-se que é totalmente justificada sua presença em grande parte das empresas que trabalham com *Big Data*, seu desempenho, facilidade de uso e escalabilidade, fazem de seu ecossistema, uma solução completa para os trabalhos com dados e informações.

Referências

APACHE SPARK. **Spark Overview**. [S. l.], 2021. Disponível em: https://spark.apache.org/docs/3.1.2/. Acesso em: 18 jun. 2021.

AVEN, Jeffrey. **Sams Teach Yourself Apache Spark™ in 24 Hours**. [S. l.]: Sams, 2017.

CARVALHO, Rafael Aquino de. **Uma análise comparativa de ambientes para Big Data: Apache Spark e HPAT**. Orientador: Prof. Dr. Alfredo Goldman. 2018. Dissertação (Mestrado em Ciências) - Instituto de Matemática e Estatística da Universidade de São Paulo, [S. l.], 2018. Disponível em: https://teses.usp.br/teses/disponiveis/45/45134/tde-15062018-110116/publico/dissertacao_rafael_aquino.pdf. Acesso em: 7 nov. 2021.

DAMJI, Jules S. et al. **Learning Spark 2.0.** 2. ed. [S. l.]: O'Reilly Media, 2020.

DOCUMENTATION: **Natural Language Toolkit**. [S. l.], 2021. Disponível em: https://www.nltk.org/. Acesso em: 7 nov. 2021.

GARCIA, Marco. **Spark: Saiba mais sobre esse poderoso framework**. [S. l.], 8 ago. 2020. Disponível em: https://www.cetax.com.br/blog/conheca-mais-sobre-o-framework-apache-spark/. Acesso em: 7 jun. 2021

GÉRON, Aurélien. **Mãos a obra: Aprendizado de máquina com Scikit-Learn & TensorFlow**: Conceitos, ferramentas e técnicas para a construção de sistemas inteligentes. 1. ed. [S. l.]: Alta Books, 2019.

GOMES, Pedro César Tebaldi. **O QUE É E COMO FUNCIONA UM CLUSTER?.** [S. l.], 3 mar. 2015. Disponível em: https://www.opservices.com.br/o-que-e-um-cluster/. Acesso em: 8 jun. 2021.

GUTIERREZ, Daniel. **Why is Apache Spark So Hot?.** [S. l.], 17 nov. 2015. Disponível em: https://insidebigdata.com/2015/11/17/why-is-apache-spark-so-hot/. Acesso em: 8 jun. 2021.

JULIO, RENNAN. "Dados são o novo petróleo", diz CEO da Mastercard – exceto por um pequeno detalhe: Para Ajay Banga, internet das coisas é a mais impactante tecnologia da transformação digital. **Época Negócios**, [s. l.], 5 jul. 2019. Disponível em:

https://epocanegocios.globo.com/Empresa/noticia/2019/07/dados-sao-o-novo-petroleo-diz-ceo-da-mastercard.html. Acesso em: 8 nov. 2020

NAVITA. **Big Data: O que é? Conheça seu conceito e definição.** [S. l.], 2 ago. 2019. Disponível em: https://navita.com.br/blog/big-data-saiba-mais-sobre-o-conceito-e-definicao/#:~:text=De%20acordo%20com%20o%20Instituto,dados%20gerados%20todos%20os%20dias. Acesso em: 8 jun. 2021.

PITANGA, Marcos. **Construindo supercomputadores com linux**. 3. ed. Rio de janeiro: Brasport, 2008.

SALLOUM, Salman; DAUTOV, Ruslan; CHEN, Xiaojun; PENG, Patrick Xiaogang; HUANG, Joshua Zhexue. **Big data analytics on Apache Spark**. International Journal of Data Science and Analytics, [s. l.], 2016. Disponível em: https://link.springer.com/content/pdf/10.1007/s41060-016-0027-9.pdf. Acesso em: 7 nov. 2021.

REUTERS. Biden renomeia Jerome Powell como presidente do Fed para segundo mandato. **G1**, [s. l.], 22 nov. 2021. Disponível em: https://g1.globo.com/economia/noticia/2021/11/22/biden-nomeia-jerome-powell-como-presidente-do-fed-pelo-segundo-mandato.ghtml. Acesso em: 22 nov. 2021.

SCHOCH, Andréa. **O que é Machine Learning?** [S. l.], 2 jan. 2018. Disponível em: https://www.appai.org.br/desenrola-machine-learning-aprendizado-de-maquina/. Acesso em: 7 jun. 2020

WHEATLEY, Mike. **Apache Spark will dominate the Big Data landscape by 2022, Wikibon says**. [S. l.], 30 mar. 2016. Disponível em: https://siliconangle.com/2016/03/30/apache-spark-will-dominate-the-big-data-landscape-by-2022-wikibon-says/. Acesso em: 9 jun. 2021.