Universidade de Brasília Faculdade UnB Gama

2º. Projeto de pesquisa - Programação de Streaming em clusters Spark/Kafka

Fundamentos de Redes de Computadores

Prof. : Fernando William Cruz

Alunos:

Davi Matheus da Rocha de Oliveira- 19/0042419

1. Introdução

O objetivo desse projeto de pesquisa é que eu possa aprimorar meus conhecimentos de arquitetura Clusters de processamento de fluxo e programação de aplicativos para consumo e Processamento de eventos, em tempo real. Esse conhecimento deve passar Configuração de processadores de fluxo combinados com servidores de mensagens Classificação do evento.

Para cumprir esses objetivos utilizaremos dois scripts, divididos em duas partes, no qual a primeira parte, é um **Spark Streaming contabilizando palavras de entrada via socket,** com isso para construção dessa aplicação, utilizei da API Apache Spark Streaming e programação usando linguagem Python e para a implementação do socket de leitura das palavras fiz o uso da biblioteca Netcat. Com isso dito, a segunda parte irá ser um **Spark Streaming contabilizando palavras via Apache Kafka.**

Apenas relembrando um pouco os conceitos da primeira pesquisa, em que abordamos um pouco o Apache Spark, em que ele é um mecanismo multilíngue para executar engenharia de dados, ciência de dados e aprendizado de máquina em máquinas ou clusters de nó único. Foi desenvolvido no AMPLab da Universidade da Califórnia e posteriormente repassado para a Apache Software Foundation que o mantém desde então. Spark provê uma interface para programação de clusters com paralelismo e tolerância a falhas.

2. Metodologia utilizada

Bom primeiramente, na primeira semana foquei mais na Primeira parte, que seria: **Spark Streaming contabilizando palavras de entrada via socket,** com isso aprimorar os meus conhecimentos sobre o Apache Spark, e tentando utilizar e aprender mais utilizando o Python, além disso dei uma olhada em ambas as formas de implementação de socket indicadas pelo professor, como o uso da biblioteca Netcat (http://netcat.sourceforge.net), ou outras opções de websocket (https://pt.wikipedia.org/wiki/WebSocket) como o pywebsocket[https://pypi.org/project/pywebsocket/]. Contudo, busquei estudar mais o NetCat, já que foi o que o professor sugeriu. Com isso concluindo a primeira semana, consegui focar mais na primeira parte na tarefa

Entrando na segunda semana foquei mais no entendimento do próprio Apache Kafka, buscando em livros e documentações(https://kafka.apache.org), na qual eu achei mais difícil do que o normal, já que era a primeira, vez utilizando o kafka.

(Como realizei o projeto sozinho, não utilizei nenhuma técnica ou gerenciamento de software para a concepção do projeto, também não apliquei nenhum método de desenvolvimento, apenas um Trello para que eu possa anotar e organizar as etapas mais eficientes. Focando mais na parte teórica para aprofundar mais no conteúdo do que na parte prática em si, também utilizando alguns vídeos no youtube para reforçar o conteúdo).

3. Soluções do Problema

3.1 **Spark Streaming contabilizando palavras de entrada via socket,**

Como pré-requisitos para essa solução é necessário a instalação do Apache Spark[https://spark.apache.org/downloads.html], e do Netcat[http://netcat.sourceforge.net/]. Uma das vantagens do Apache Spark é o uso de API's para facilitar o trabalho, em que nesse projeto utilizei a biblioteca PySpark. O PySpark é uma interface de alto nível que permite acessar e usar o Spark por meio da linguagem Python. Com o PySpark, você pode escrever todo o seu código usando apenas nosso estilo de codificação Python. Então o **PySpark** é uma API Python para Apache **SPARK** denominado como o mecanismo de processamento analítico para aplicações de processamento de dados distribuídos em larga escala e aprendizado de máquina, ou seja, para grandes volumes de dados.

Em comparação, o Netcat é um utilitário de rede especializado que usa o protocolo TCP/IP para ler e gravar dados em uma conexão de rede. Ele foi projetado para ser uma ferramenta "backend" confiável que pode ser usada direta ou facilmente por outros programas e scripts. Ao mesmo tempo, é uma ferramenta de depuração e exploração de rede rica em recursos, pois pode criar praticamente qualquer tipo de conexão que você precisar e possui muitos recursos internos interessantes.

Com isso exemplificado o problema dizia que a aplicação deveria ser desenvolvido em um cluster (envolvendo mais de um host) e fazer uma aplicação que (i) consiga ler palavras enviadas a esse servidor à partir de um socket TCP ou UDP, (ii) contabilize o total de palavras recebidas pelo socket e o número de ocorrências de cada palavra, durante o tempo de atividade do servidor e (iii) apresente o resultado dessa contabilização em arquivo ou console, de modo que seja possível perceber a dinâmica de leitura/contabilização das entradas.

Utilizei bastante da documentação para conseguir realizar o problema principalmente a aba Structured Streaming Programming

Guide[https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html], assim vamos indo para a solução proposta:

Primeiro, temos que importar as classes necessárias e criar uma SparkSession local, o ponto de partida de todas as funcionalidades relacionadas ao Spark:

FIGURA 1: Imports necessários
Fonte: Autor

Em seguida, vamos criar um DataFrame de streaming que representa os dados de texto recebidos de um servidor escutando em localhost:9999 e transformar o DataFrame para calcular a contagem de palavras:

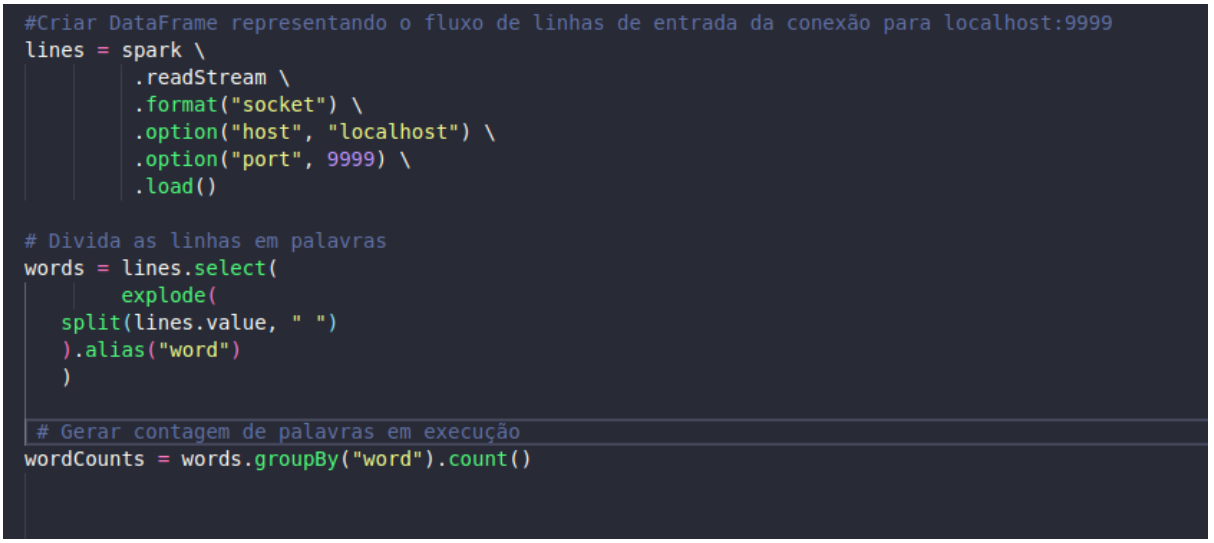

FIGURA 2: Conexão e Contagem de Palavras
Fonte: Autor

Esta linha DataFrame representa uma tabela infinita contendo dados de texto de streaming. A tabela contém uma coluna de strings chamada "valores", e cada linha nos dados de texto de streaming se torna uma linha na tabela. Como estamos apenas configurando a transição, ela ainda não começou. Em seguida, usamos duas funções SQL integradas - dividir e explodir, para dividir cada linha em várias linhas, uma palavra por linha.

O alias da função nomeia a nova coluna como "palavra". Por fim, definimos o WordCounts DataFrame agrupando pelos valores únicos no Dataset e contando-os. Observe que este é um dataframe de fluxo que representa a contagem de palavras em execução do fluxo.

Agora configuramos consultas em dados de streaming. Tudo o que resta é realmente começar a receber os dados e contar as palavras. Para fazer isso, nós o configuramos para imprimir o conjunto completo de contagens (especificado por outputMode("complete")) no console a cada atualização.

```
 # Comece a executar a consulta que imprime as contagens em execução no console
query = wordCounts \
          .writeStream \
          .outputMode("complete") \
          .format("console") \
          .start()

query.awaitTermination()
```

FIGURA 3: Consulta(Querry)

Fonte: Autor

Depois que esse código for executado, a computação de streaming será iniciada em segundo plano. O objeto de consulta é um identificador para essa consulta de streaming ativa e decidimos aguardar o término da consulta usando awaitTermination() para impedir que o processo seja encerrado enquanto a consulta estiver ativa.

3.2 **Spark Streaming contabilizando palavras via Apache Kafka.**

Apache Kafka é uma plataforma open-source de processamento de streams desenvolvida pela Apache Software Foundation, escrita em Scala e Java. O projeto tem como objetivo fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados em tempo real.

Criando o banco de dados e escrevendo o input:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split
from pyspark.sql.functions import substring
from pyspark.sql.functions import window, upper

spark = SparkSession \
            .builder \
            .appName("KAFKA") \
            .getOrCreate()

# Create DataFrame representing the stream of input lines and write it in kafka topic
lines = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9090") \
    .option("write", "contador_palavras") \
    .option('includeTimestamp', 'true') \
    .load()
```

FIGURA 4: Conexão Kafka
Fonte: Autor

O alias da função nomeia a nova coluna como "palavra". Por fim, definimos o WordCounts DataFrame agrupando pelos valores únicos no Dataset e contando-os. Observe que este é um dataframe de fluxo que representa a contagem de palavras em execução do fluxo

:

```
# Split the lines into words
words = lines.select(
    explode(
        split(lines.value, "\s+")).alias("word"),
        lines.timestamp
    )
words = words.select(upper(words.word).alias('word'), words.timestamp)

# Group words
wordCounts = words.groupBy("word").count()

# Count the total of words readed
total = words \
    .groupBy() \
    .count() \
    .selectExpr("'TOTAL' as key", "CAST(count AS STRING) as value")
```

FIGURA 5: Contagem de Palavras
Fonte: Autor

```
# Contar  6, 8 and 11
lengths = words \
    .filter(length(words.word).isin([6, 8, 11])) \
    .withWatermark("timestamp", "3 seconds") \
    .groupBy(
        window(words.timestamp, "3 seconds", "3 seconds"),
        length(words.word).alias("key")
    ) \
    .count() \
    .selectExpr("CAST(key AS STRING)", "CAST(count AS STRING) as value")



# Contar as palavras S, P and R
letters = words \
    .filter(upper(substring(words.word, 0, 1)).isin(["S", "P", "R"])) \
    .withWatermark("timestamp", "3 seconds") \
    .groupBy(
        window(words.timestamp, "3 seconds", "3 seconds"),
        upper(substring(words.word, 0, 1)).alias("key"),
    ) \
    .count() \
    .selectExpr("key", "CAST(count AS STRING) as value")
```

Para a conclusão parte 2 do projeto, a contagem de palavras é feita através do registro de um tópico no Apache Kafka. A solução Apache Spark se inscreve em um tópico Kafka e aguarda o processamento de um fluxo de palavras. À medida que um bloco de palavras é processado, a contagem de palavras é exibida no console e os tópicos são enviados para outro arquivo do Apache Kafka chamado

complemento para agrupar esses tópicos. Portanto, outro trabalho do Apache Spark se escreve neste tópico de estatísticas e exibe sua exibição no console

4. Conclusão

4.1 Resultado do Projeto

Neste projeto foi apresentado um estudo em torno da arquitetura e funcionamento do projeto da Apache Spark com socket e do Apache Kafka, as duas soluções do projeto foram concluídas com sucesso, atendendo a todos os requisitos descritos na especificação do projeto. No entanto, foram encontrados alguns problemas, não consegui fazer a segunda parte do projeto em cluster e também não obtive sucesso na utilização de gráficos utilizando a biblioteca

Por meio deste trabalho, pode-se perceber como é o verdadeiro funcionamento do Streaming e programação de aplicações para consumo e tratamento de eventos, em tempo real e diferença em que duas Apaches diferentes resultam.

5.2 Considerações Finais

Gostei bastante do projeto, ainda mais devido a facilidade de uso do Kafka e do Spark que foi dado devido a facilidade para instalar e executar tanto o Apache Spark quanto o Apache Kafka, em alguns minutos tudo já estava pronto para o colocar a mão na massa.

. Participei ativamente do desenvolvimento do projeto, em todas as etapas, inclusive na elaboração de funcionalidades centrais da aplicação e da construção de um relatorio bem elaborado, além disso, o trabalho possibilitou entender os conteúdos abordados em sala de aula, na prática, e uma visão ampla do conhecimento de streaming e clusters. em tempo real Acredito que uma nota 8,5 seria condizente com minha participação nesse projeto, desenvolvendo e produzindo para melhorar o trabalho.

REFERENCIAS

[1]https://phoenixnap.com/kb/install-spark-on-ubuntu
[2] Documentação Oficial - Apacha Spark (https://spark.apache.org/streaming/);

[3] http://netcat.sourceforge.net/

[4] https://pt.wikipedia.org/wiki/WebSocket

[5] https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

[6]https://docs.confluent.io/5.5.1/streams/quickstart.html

[7]Hadoop MapReduce vs Spark | Hadoop Tutorial For Beginners | Hadoop & Spark Tutorial | Edureka(https://www.youtube.com/watch?v=ivgQtdB-BS4)