



**Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Curso de Engenharia de Software**

**Programação para Sistemas Paralelos e
Distribuídos [FGA0244]**

**Projeto de pesquisa - Programação de Streaming em
clusters Spark/Kafka**

Professor: Fernando W. Cruz

**Autores:
Eduardo Afonso Dutra Silva
Rafael Cleydson da Silva Ramos
Thiago Sampaio de Paiva**

Brasília, DF



Sumário

Sumário	2
Introdução	3
Metodologia	3
Descrição da solução streaming	4
Parte 1 - Spark Streaming contabilizando palavras de entrada via socket	4
Parte 2 - Spark Streaming contabilizando palavras via Apache Kafka	5
Conclusão	8
Opinião Geral, Dificuldades e Aprendizado	8
Referências	9

1. Introdução

Ler e processar informações geradas em tempo real é uma ferramenta muito poderosa que permite o desenvolvimento de diversos tipos de aplicações. Com o objetivo de aprimorar os conhecimentos sobre este assunto, o presente relatório aborda os experimentos de classificação, contagem e apresentação de palavras lidas em tempo real utilizando o processador de *streaming Apache Spark* e o servidor de mensageria *Apache Kafka*.

2. Metodologia

Para viabilizar a realização do presente projeto, foram realizados alguns encontros síncronos entre os membros. Estes encontros síncronos foram necessários para alinhar o conhecimento dos membros referente ao conteúdo do projeto, e também discutir e aplicar as melhores abordagens e/ou soluções para cada desafio proposto, a ideia inicial foi começar a trabalhar com os códigos configurados em um nó local dos computadores de cada um dos membros, para posteriormente realizar as configurações via cluster para aproveitar os benefícios dessa abordagem, bem como comparar a velocidade de cada solução. Os encontros se deram da seguinte maneira:

18/08/2022: Foi alinhado o conhecimento dos membros referente às tecnologias utilizadas e também realizada a parte 1 do projeto e a configuração do *Apache Kafka* para utilização na parte 2.

19/08/2022: Desenvolveu-se o código capaz de receber no *Apache Spark* as palavras escritas no tópico criado no *Apache Kafka*, processa-las, separá-las em *DataFrames* e então apresentar no terminal cada uma das contagens separadamente (palavras iniciadas com a letra “S”, palavras iniciadas com a letra “P”, etc...).

20/08/2022: Foi criado um novo tópico chamado “statistics” no *Apache Kafka* para receber as contagens e também um novo *script python* para ler esse novo tópico e apresentar todos os dados unificados em um único *DataFrame*. Além disso, também foi desenvolvida a lógica para apresentar algumas contagens referentes às palavras inseridas durante todo o tempo de execução do programa e outras para apresentar somente as contagens das palavras inseridas nos últimos 3 segundos.

21/08/2022: Esta reunião foi dedicada a tentar executar em *cluster* utilizando os computadores de todos os membros da equipe e também para elaborar o presente relatório.

3. Descrição da solução streaming

Todas as soluções foram criadas utilizando o *Apache Spark* na versão 3.3.0, com a biblioteca *Spark Streaming* e a interface *PySpark*, que permite a utilização do processador de *streaming* por meio da linguagem de programação *Python*.

3.1. Parte 1 - Spark Streaming contabilizando palavras de entrada via socket

Para o desenvolvimento da primeira parte do projeto foi utilizada a ferramenta *Netcat* para abrir uma porta na rede e permitir a conexão do *Apache Spark Streaming* para leitura das palavras via socket.

A solução para o problema foi basicamente ler o conteúdo escrito na porta aberta utilizando o *Netcat*, conteúdo esse que vinha separado por linhas. Separar o conteúdo lido nas ocorrências de espaçamentos utilizando as funções “*select*”, “*explode*” e “*split*” do *PySpark*, assim tendo acesso às palavras.

Após separar todas as palavras em um *DataFrame*, foi possível utilizar a função “*count*” para contar todas as palavras, representadas por linhas no *DataFrame*, e também utilizar a função “*groupBy*” e “*sum*” para agrupar palavras iguais e contar o número de ocorrências de cada palavra. Os resultados foram apresentados no console utilizando a função “*writeStream*” dos *DataFrames* criados. A solução é ilustrada na Figura 1.

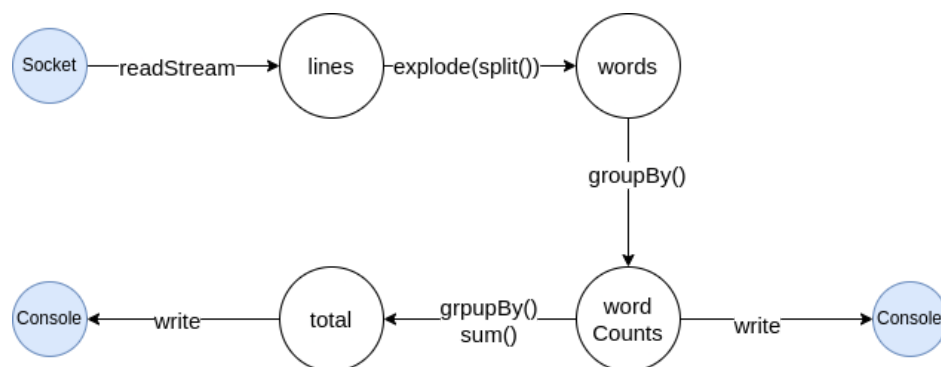


Figura 1: Diagrama da Solução com Kafka e Netcat

Fonte: autores

A solução foi executada em um cluster com 2 nós, tendo um total de 8 cores e 17GB de memória RAM, como é possível observar na Figura 2.

▼ Workers (2)					
Worker Id	Address	State	Cores	Memory	
worker-20220821164730-192.168.10.4-37581	192.168.10.4:37581	ALIVE	4 (4 Used)	2.8 GiB (2.0 GiB Used)	
worker-20220821164819-127.0.0.1-36405	127.0.0.1:36405	ALIVE	4 (4 Used)	14.5 GiB (2.0 GiB Used)	
▼ Running Applications (1)					
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
app-20220821174643-0011	(kill) P2 - PSPD - Socket	8	2.0 GiB		2022/08/21 17:46:43

Figura 2: Execução Primeira Etapa em Cluster

Fonte: autores

A saída em console da execução em cluster da solução da primeira etapa pode ser observada na Figura 3.

```
+-----+-----+
|  key|value|
+-----+-----+
|TOTAL|71247|
+-----+-----+

+-----+-----+
|               word|count|
+-----+-----+
|               USED| 446|
|             DURATION| 533|
|          APPLICATION| 533|
|    ACSOKASCOKOCSAK|   1|
|             ASCJASC|   1|
|        2022/08/21| 3013|
|WORKER-2022082116...|  223|
|              USED)|  892|
|            STATUS:|  223|
|SPARK://192.168.1...|  446|
|               GIB|  892|
|              7,4|  310|
|              NAME|  533|
|            18:44:26|  310|
|             WORKER|  223|
|    CSCASOACSSACJCSAJ|   1|
|APP-2022082118440...|  310|
|            19:05:36|  310|
|              8| 3149|
|            18:44:00|  310|
+-----+-----+
only showing top 20 rows
```

Figura 3: Saída em Console da Execução da Segunda Etapa em Cluster

Fonte: autores

3.2. Parte 2 - Spark Streaming contabilizando palavras via Apache Kafka

Para o desenvolvimento da parte 2 do projeto, a contabilização de palavras se deu por meio da inscrição de um tópico no *Apache Kafka*. A solução do *Apache Spark* se inscreve em um tópico do *Kafka* e fica esperando um fluxo de palavras para realizar o processamento.

Quando um bloco de palavras é processado a contagem das palavras é mostrada no *console* e as estatísticas são enviadas para um outro tópico do *Apache Kafka*, chamado *statistics*, para que as estatísticas fiquem agrupadas. Assim um

outro *job* do *Apache Spark* se inscreve nesse tópico de estatísticas e apresenta sua visualização no console. A solução é ilustrada na Figura 4.

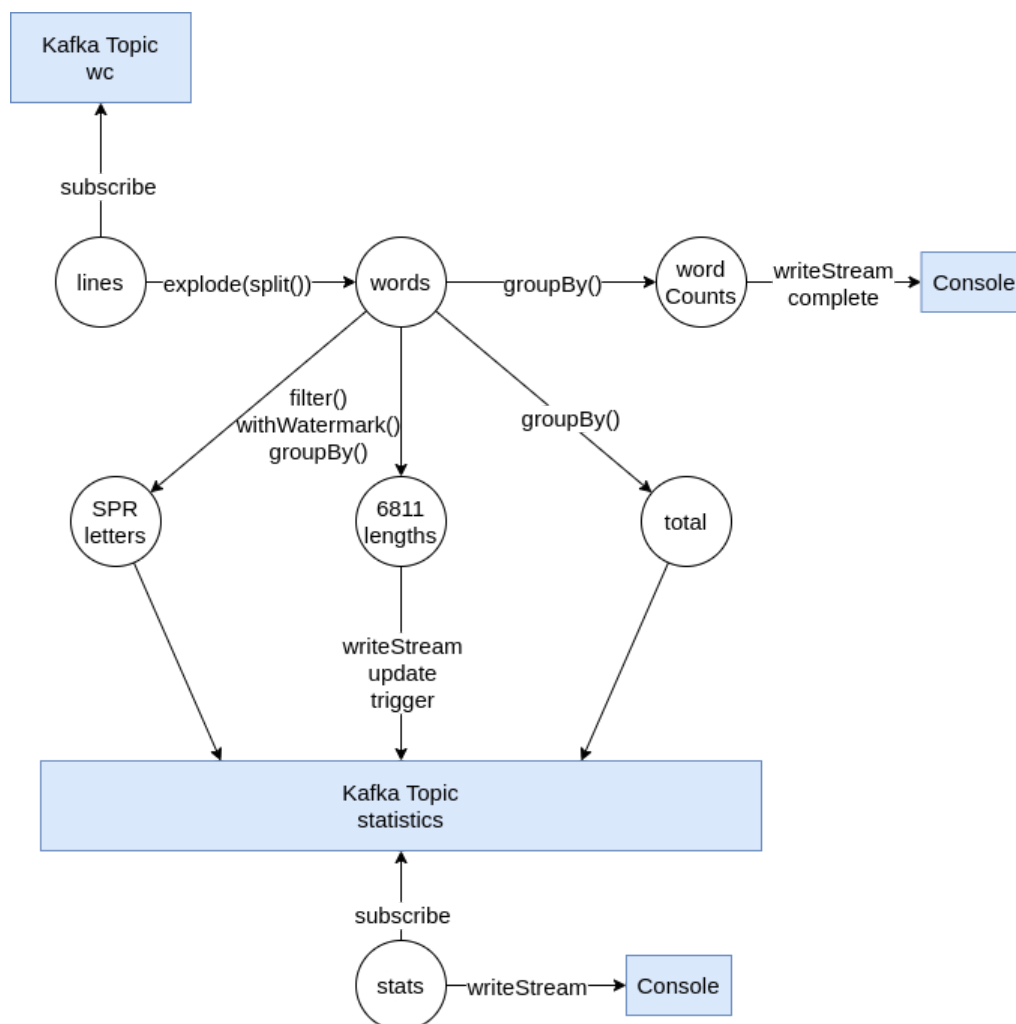


Figura 4: Diagrama da Solução com Kafka e Spark
Fonte: autores

Como é possível observar, acontece uma inscrição no tópico *wc* que gera um *Streaming DataFrame*, chamado de *lines*, que vai nos permitir interagir com o fluxo de dados em tempo real. O *lines* sofre diversas transformações para que seja possível obter os dados necessários. A primeira delas é separar as palavras que estão em linhas em palavras, isso acontece durante o `'explode(split())'`. Por conseguinte, essas palavras são agrupadas no *wordCounts* e são escritas no *console* no modo *complete*, este modo garante que todas as linhas sejam imprimidas.

O *DataFrame words* é transformado para que seja possível obter as estatísticas do problema. Para isso, as palavras com iniciais S, P e R e com comprimento de 6, 8 e 11 são filtradas e agrupadas em dois *DataFrame*, um das letras e o outro do comprimento.

Para que a saída seja escrita em um intervalo de tempo, foi utilizado *watermark* e . Quando se recebe um fluxo de dados do spark, esse dado vem com um *timestamp*. Os filtros são agrupados pelo *timestamp* e pela característica, letra ou comprimento, e enviados para o tópico de estatística ativados por um trigger a cada 3 segundos.

A solução foi executada em um cluster com 2 nós, tendo um total de 8 cores e 17GB de memória RAM, como é possível observar na Figura 5.

▼ Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20220821164730-192.168.10.4-37581	192.168.10.4:37581	ALIVE	4 (4 Used)	2.8 GiB (2.0 GiB Used)
worker-20220821184246-127.0.0.1-36517	127.0.0.1:36517	ALIVE	4 (4 Used)	14.5 GiB (1024.0 MiB Used)

▼ Running Applications (2)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
app-20220821184426-0002	(kill) P2 - PSPD - Transformer	7	1024.0 MiB		2022/08/21 18:44:26
app-20220821184400-0001	(kill) P2 - PSPD - Stats Consumer	1	1024.0 MiB		2022/08/21 18:44:00

Figura 5: Execução Segunda Etapa em Cluster

Fonte: autores

A saída em console da execução em cluster da solução da segunda etapa pode ser observada na Figura 6.

<pre> +-----+-----+ word count +-----+-----+ YTSGWVEV 1 DZ 1 VDXZW 1 WNZPUDOPF 1 FEKZUVF 1 LT 1 EVVHRWKETXG 1 DAAPTIXRX 1 GFSZXJVEE 1 VWEXCUXOQW 1 LEB 1 NBKP 1 K 17 ATPPYCWGX 1 PZKVVNI 1 FMXBZKE 1 MM 1 KOTWSZIBIDT 1 RDJY 1 BRKCICSACE 1 +-----+-----+ only showing top 20 rows </pre>	<pre> +-----+-----+ key value +-----+-----+ TOTAL 19462001 S 170323 R 169696 P 170119 +-----+-----+ Batch: 8 +-----+-----+ key value +-----+-----+ 8 368409 11 367776 6 368207 +-----+-----+ Batch: 9 +-----+-----+ key value +-----+-----+ TOTAL 100000001 +-----+-----+ </pre>
---	---

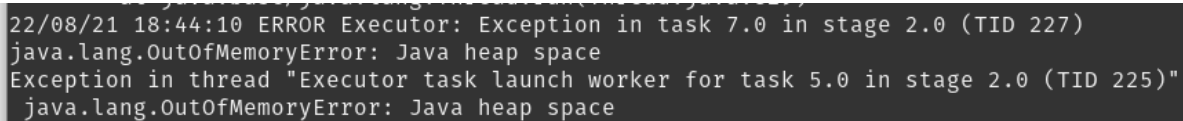
Figura 6: Saída em Console da Execução da Segunda Etapa em Cluster

Fonte: autores

4. Conclusão

Ambas as etapas do projeto foram concluídas com êxito, atendendo a todos os requisitos descritos na especificação do projeto. Porém foram encontradas algumas limitações.

Na primeira parte do projeto não foi possível executar arquivos com mais de 70 MB. Mesmo utilizando *tags* de execução para limitar o uso de memória sempre é disparado o erro “Java heap space”, conforme figura 3, portanto a solução não é capaz de comportar arquivos de grandes dimensões.



```
22/08/21 18:44:10 ERROR Executor: Exception in task 7.0 in stage 2.0 (TID 227)
java.lang.OutOfMemoryError: Java heap space
Exception in thread "Executor task launch worker for task 5.0 in stage 2.0 (TID 225)"
java.lang.OutOfMemoryError: Java heap space
```

Figura 7: Erro “Java heap space”

Fonte: autores

Outra limitação foi a execução da solução em cluster. A equipe não conseguiu executar a segunda parte do projeto em cluster pois existem dois *jobs* no *Apache Spark*, um para fazer a contagem das palavras e outro para apresentar os resultados, porém utilizando cluster não foi possível executar tarefas paralelamente. Portanto, a solução encontrada foi utilizar o cluster para executar o *job* de contagem e executar o *job* de apresentação localmente.

Um ponto importante é que não foram utilizadas bibliotecas extras do *Apache Spark*, além da *Spark Streaming*. A biblioteca *GraphX* não foi utilizada pois ela é feita para um contexto de grafos e não foi encontrada uma aplicação de grafos no escopo do projeto. Já a biblioteca *MLlib* despertou o interesse da equipe porém não obtivemos êxito em sua implementação.

4.1. Opinião Geral, Dificuldades e Aprendizado

Eduardo: Acredito que o grande ponto positivo deste projeto foi a facilidade para instalar e executar tanto o *Apache Spark* quanto o *Apache Kafka*, em alguns minutos tudo já estava pronto para o desenvolvimento. A grande dificuldade deste projeto foi utilizar a API do Apache Spark, muitos comandos não funcionam ou funcionam de forma diferente quando utilizados em contexto de *streaming* assim perdemos muito tempo para fazer coisas que teoricamente seriam simples como manipular os *DataFrames*. Minha participação foi contribuir com o debate sobre os problemas encontrados, buscando e testando soluções e auxiliando na construção do relatório. Em relação ao escopo do projeto eu daria uma nota 8/10, pois não conseguimos aplicar a biblioteca de machine learning.

Rafael: Fiquei muito animado a princípio com o projeto devido ao uso do spark streaming permitir observar as alterações em tempo real, além de verificar também o passo-a-passo para chegar até a solução final. Senti alguma dificuldade na utilização dos métodos do `pyspark.sql.functions` que trabalham com streaming dataframes devido ao fato dele não permitir algumas operações que são utilizadas em dataframes normais, então essa restrição devido a algumas interações de agregações impossibilitou a solução que estávamos a desenvolver, e desafiou a gente a tentar novas soluções.

Acho que podíamos ter alcançado um projeto mais robusto com a utilização de machine learning para identificar o sentimento geral das frases, se está mais para uma frase boa (elogio) ou mau (crítica maldosa), porém, acabei ficando muito doente e não consegui olhar essa parte de maneira mais atenciosa, mas já estava cheio de ideias em relação a isso, que seria uma simples interface web utilizando websockets na comunicação, e um modelo treinado com palavras boas e ruins em português que estivesse disponível na internet, pelo menos o database.

Devido ao fato de que fiquei doente e tive algumas outras responsabilidades e não consegui contribuir com o projeto minha nota é 7/10.

Thiago:

Gostei bastante do projeto, ainda mais devido a facilidade de uso do Kafka e do Spark. Achei no início a programação em Streaming bastante complicada porque tem que levar em consideração que agregações não podem interagir, além de precisar se preocupar com o tempo de vida, não poder usar sql no fluxo de dados, com o watermarking, com o sink. O socket por meio do NetCat só permite um cliente por vez, então conseguir realizar a contagem das palavras e do total foi um grande desafio.

Como tenho uma máquina virtual na Azure, tentei montar um super cluster juntando o cluster da FGA e dos integrantes, chegou a ter 108 cores e 150GB. Devido ao bloqueio da porta pelo firewall da FGA, tive que mudar as portas do Spark Master para 8080 e do Kafka para 13508. Como não consegui rodar o Kafka na máquina da Azure, tive que fazer um SSH Remote Forwarding para deixar meu Kafka local público. No final, não deu pra usar esse cluster porque necessitava de mais portas que não dava pra conectar devido o firewall e o disco do cluster estava cheio. Mesmo assim, aprendi bastante sobre redes e ssh nesse projeto.

Já no projeto eu contribuí em todas as etapas. Então por cumprir os requisitos **minha nota é 10/10.**

5. Referências

APACHE. Structured Streaming Programming Guide. Disponível em: <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>.

Acesso em: 21 ago. 2022

APACHE. Structured Streaming + Kafka Integration Guide. Disponível em: <https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html>.

Acesso em: 21 ago. 2022