Explorando aplicações distribuídas em clusters

Programação p/ Sistemas Paralelos e Distribuídos

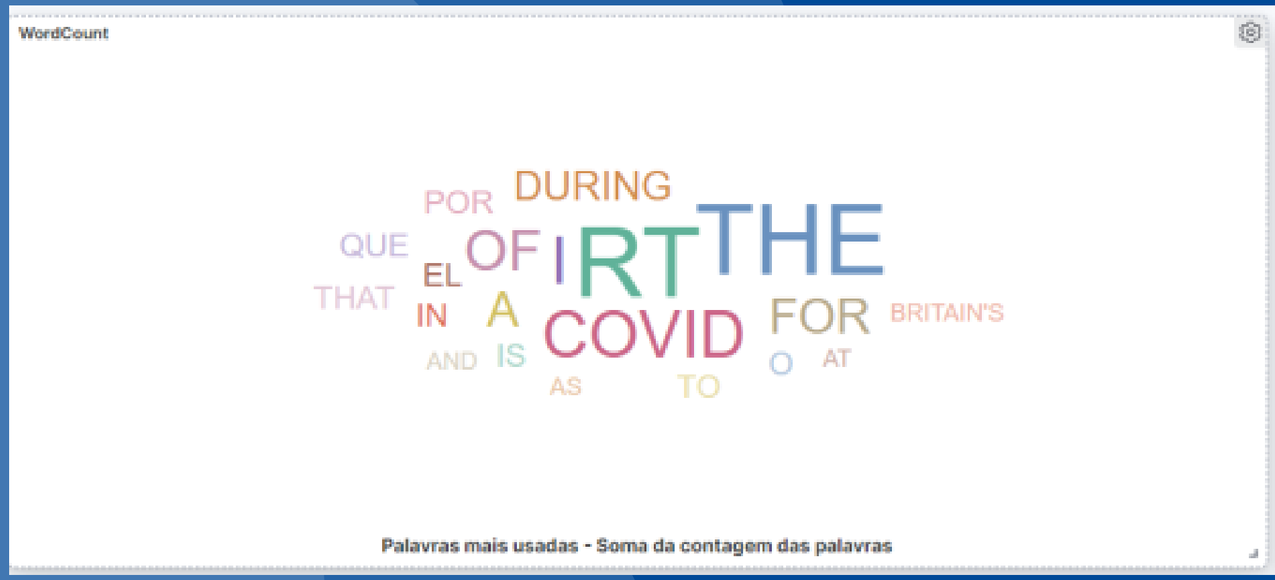
Equipe

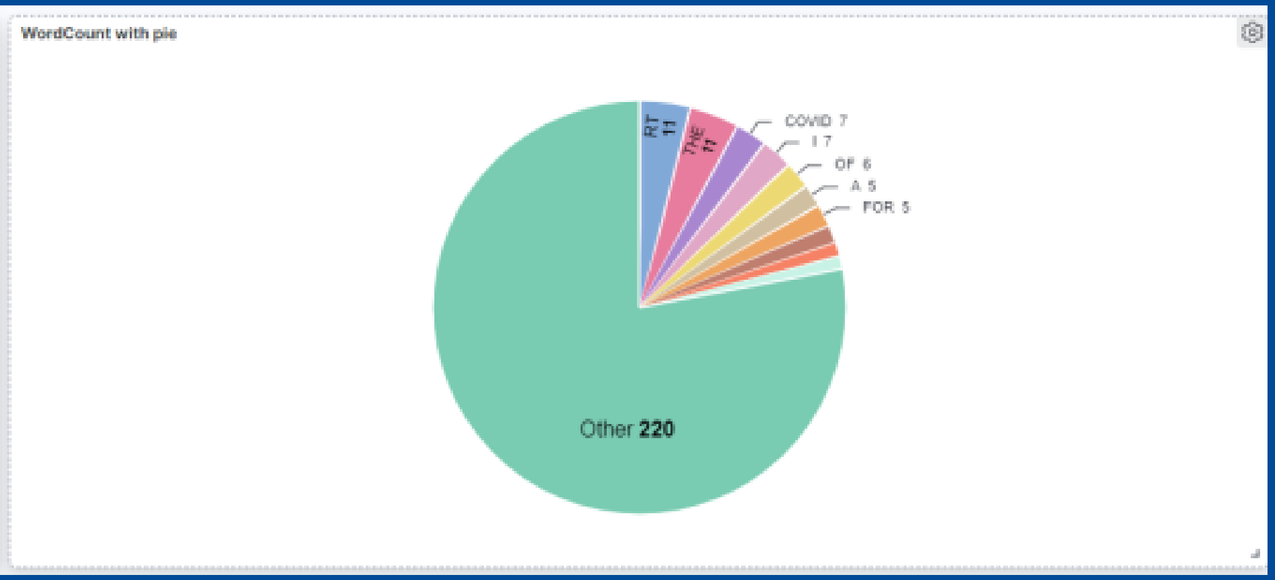
Eduardo Afonso
19/0012307

Thiago Paiva
19/0020377

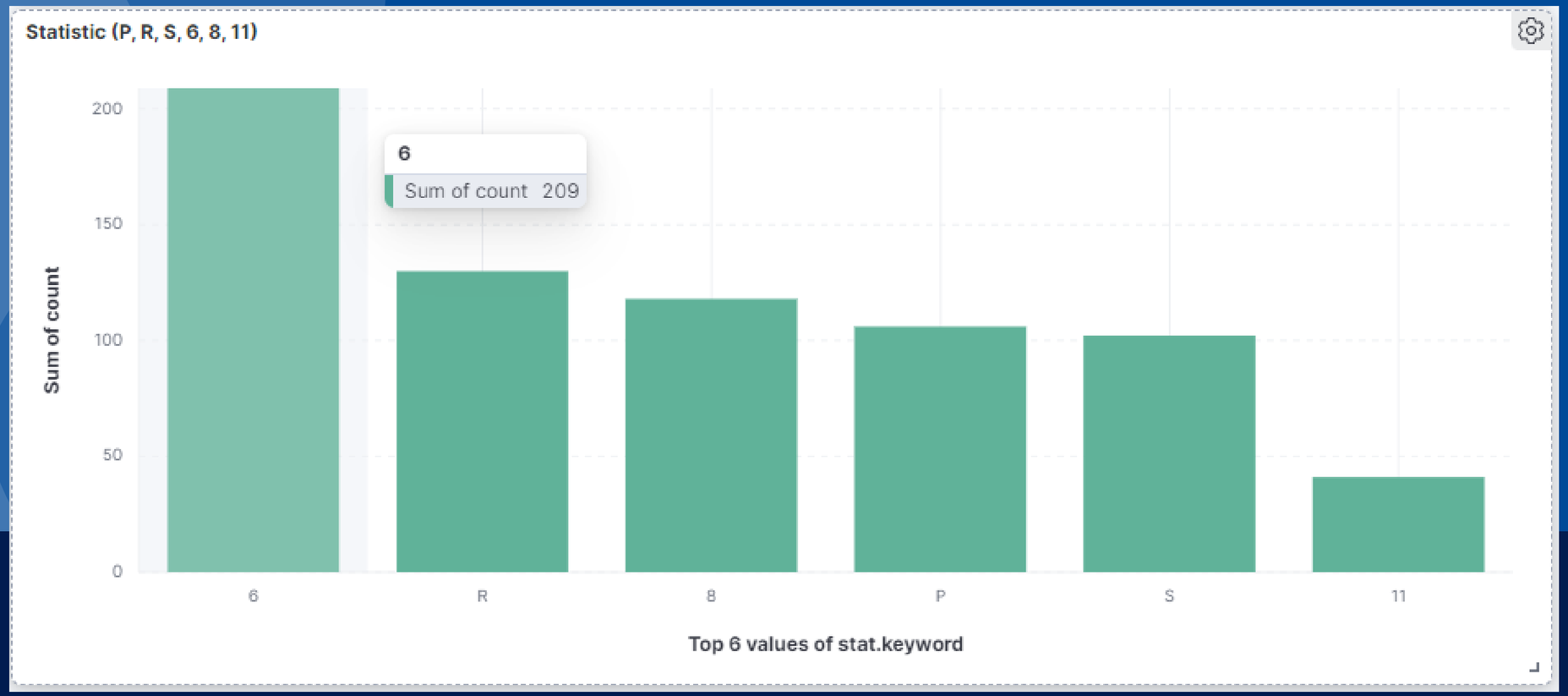
Rafael Cleydson
19/0019085

Gráficos Contagem de Palavras





Gráficos Contagem de Palavras com Certas Características



Contagem Total de Palavras

Implementação Contagem De Palavras

```
INTERVAL = os.getenv("INTERVAL", "10 seconds")
SPARK_MASTER = os.getenv("SPARK_MASTER", "spark://gpu3.esw:7077")
KAFKA_SERVER = os.getenv("KAFKA_SERVER", 'localhost:9092')

SPARK_APP_NAME = "Final - PSPD"

WORDS_TOPIC = os.getenv("WORDS_TOPIC", 'wc')
STATS_TOPIC = os.getenv("STATS_TOPIC", 'test-elasticsearch-sink')

SPARK_CORES_MAX = os.getenv("SPARK_CORES_MAX", "2")
```

```
conf = SparkConf() \
    .setMaster(SPARK_MASTER) \
    .setAppName(SPARK_APP_NAME) \
    .set("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0") \
    .set("spark.cores.max", "2")

context = SparkContext(conf=conf)
context.setLogLevel("ERROR")
```

Implementação Contagem De Palavras

```
lines = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
    .option("subscribe", WORDS_TOPIC) \
    .option('includeTimestamp', 'true') \
    .load()
```

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

Implementação Contagem De Palavras

```
wordCounts = words.withWatermark("timestamp", INTERVAL) \
                .groupBy(
                    window(words.timestamp, INTERVAL, INTERVAL),
                    "timestamp",
                    "word"
                ) \
                .count() \
                .select(
                    lit('1').alias("key"),
                    format_string("{\"word\": \"%s\", \"count\": %d, \"timestamp\": %d}", col("word"), col("count"), col("timestamp")).al
                )
```

Sinks

```
qWc = wordCounts \
    .writeStream \
    .outputMode("update") \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
    .option('topic', STATS_TOPIC) \
    .option('checkpointLocation', '/tmp/spark/wc-stats') \
    .trigger(processingTime=INTERVAL) \
    .start()
```

Implementação Contagem De Palavras

```
# Count the words that has length 6, 8 and 11
lengths = words \
    .filter(length(words.word).isin([6, 8, 11])) \
    .withWatermark("timestamp", INTERVAL) \
    .groupBy(
        window(words.timestamp, INTERVAL, INTERVAL),
        "timestamp",
        length(words.word).alias("length")
    ) \
    .count() \
    .select(
        lit('1').alias("key"),
        format_string("{\"stat\": \"%s\", \"count\": %d, \"timestamp\": %d}", col("length"), col("count"), col("timestamp")).alias("value
    )
```

```
qLen = lengths \
    .writeStream \
    .outputMode("update") \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
    .option('topic', STATS_TOPIC) \
    .option('checkpointLocation', '/tmp/spark/len-stats') \
    .trigger(processingTime=INTERVAL) \
    .start()
```

Implementação Contagem De Palavras

```
# Count the words that startswith S, P and R
letters = words \
    .filter(upper(substring(words.word, 0, 1)).isin(["S", "P", "R"])) \
    .withWatermark("timestamp", INTERVAL) \
    .groupBy(
        window(words.timestamp, INTERVAL, INTERVAL),
        "timestamp",
        upper(substring(words.word, 0, 1)).alias("stat"),
    ) \
    .count() \
    .select(
            lit('1').alias("key"),
            format_string("{\"stat\": \"%s\", \"count\": %d, \"timestamp\": %d}", col("stat"), col("count"), col("timestamp")).alias("val
        )
```

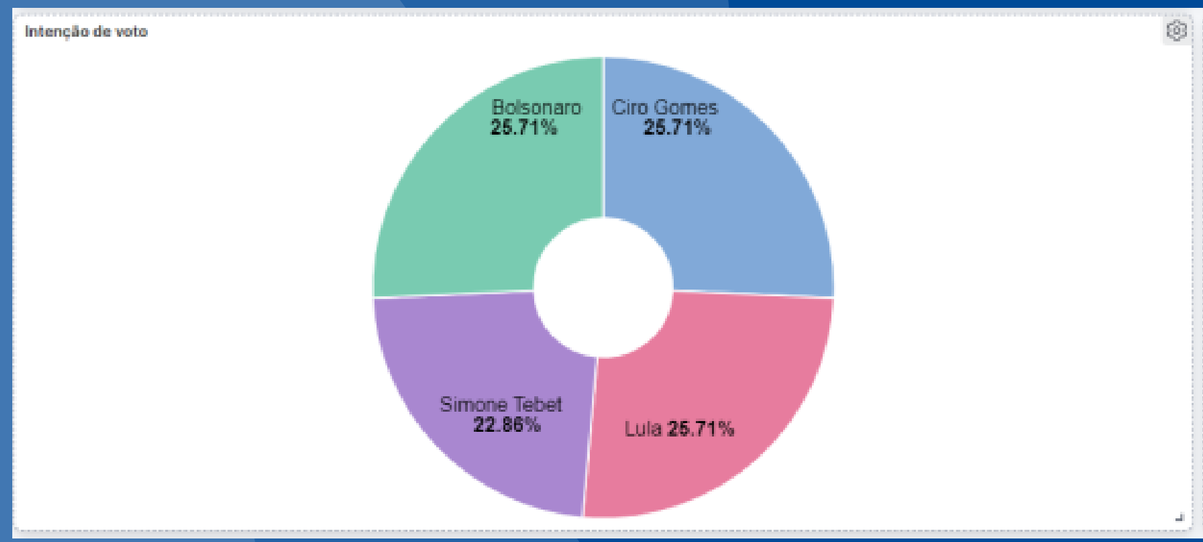
```
qLet = letters \
    .writeStream \
    .outputMode("update") \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
    .option('topic', STATS_TOPIC) \
    .option('checkpointLocation', '/tmp/spark/let-stats') \
    .trigger(processingTime=INTERVAL) \
    .start()
```

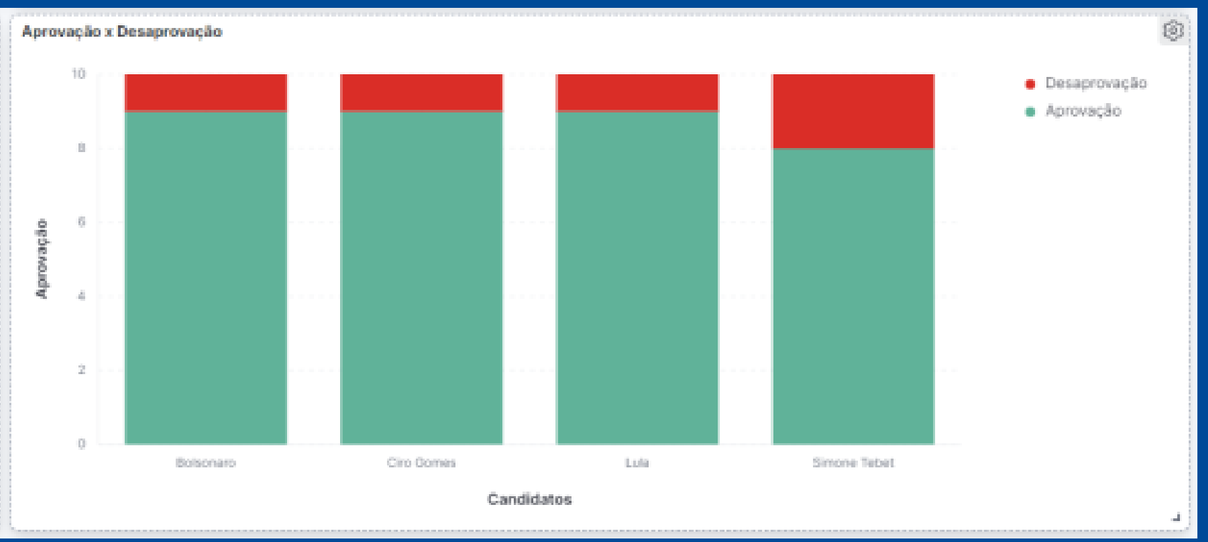
Implementação Contagem De Palavras

```
# Count the total of words readed
total = words \
    .groupBy() \
    .count() \
    .select(
            lit('1').alias("key"),
            format_string("{\"stat\": \"total\", \"count\": %d}", col("count")).alias("value")
        )
```

```
qT = total \
    .writeStream \
    .outputMode("complete") \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
    .option('topic', STATS_TOPIC) \
    .option('checkpointLocation', '/tmp/spark/total-stats') \
    .start()
```

Intenções de Voto





Implementação Intenções de Voto

Constants

```
TRAINING_FILE = os.getenv("TRAINING_FILE","dataset/dataset.csv")
SPARK_MASTER = os.getenv("SPARK_MASTER", "spark://gpu3.esw:7077")
KAFKA_SERVER = os.getenv("KAFKA_SERVER", 'localhost:9092')

SPARK_APP_NAME = "Final - PSPD - Predict"
INTERVAL = os.getenv("INTERVAL", "10 seconds")

PREDICT_TOPIC = os.getenv("PREDICT_TOPIC", 'election')
STATS_TOPIC = os.getenv("STATS_TOPIC", 'test-elasticsearch-sink')

PACKAGES = "org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0"

PRETRAINED_MODEL_PATH = os.getenv("PRETRAINED_MODEL_PATH", "model/trained.model")
STOPWORDS_PATH = os.getenv("STOPWORDS_PATH", "dataset/stopwords.txt")

SPARK_CORES_MAX = os.getenv("SPARK_CORES_MAX", "2")
```

Startup

```
conf = SparkConf() \
    .setMaster(SPARK_MASTER) \
    .setAppName(SPARK_APP_NAME) \
    .set("spark.jars.packages", PACKAGES) \
    .set("spark.cores.max", "2")

context = SparkContext(conf=conf)
context.setLogLevel("ERROR")
```

Implementação Intenções de Voto

Cleaner

```
import re

CLEAN_REGEX = r"[.,/\\\[\]\{\}`~^\d&!@#$%*\)\(\'\"<>=+-:;?]"

stopwords = set()

with open(STOPWORDS_PATH, "r") as stop_file:
    for w in stop_file:
        stopwords.add(w.strip().lower())

def cleaner(sentence):
    print(sentence)
    sentence = " ".join(
        filter(
            lambda x: x not in stopwords,
            re.sub(CLEAN_REGEX, '', sentence).split()
        )
    )
    return sentence

cleaner_col = udf(lambda s: cleaner(s), StringType())
```

Load Pre-trained Model

```
model = PipelineModel.load(PRETRAINED_MODEL_PATH)
```

Implementação Intenções de Voto

Prediction

```
def foreach_batch_func(df: DataFrame, _):
    # Preparations - split into candidate and message and clean
    candidateMessage = split(df.value, ",", 2)
    sentences = df \
                .withColumn("candidate", candidateMessage.getItem(0)) \
                .withColumn("sentence", cleaner_col(lower(candidateMessage.getItem(1))))

    # Predict
    prediction = model.transform(sentences) \
                .select(
                    "candidate",
                    "sentence",
                    "probability",
                    when(col("prediction") == 1.0, "positive").otherwise("negative").alias("prediction")
                ) \

    # Write in console
    prediction \
        .write \
        .format("console") \
        .save()

    # Prepare prediction to elasticsearch format
    # Group by candidate and prediction and format to json
    predictionElastic = prediction \
                        .groupBy(
                            "candidate",
                            "prediction"
                        ).count() \
                        .select(
                            lit('1').alias("key"),
                            format_string(
                                "{\"candidate\": \"%s\", \"%s\": %d}",
                                col("candidate"), col("prediction"), col("count")
                            ).alias("value")
                        )

    # Write to kafka elasticsearch topic
    predictionElastic.write \
                    .format("kafka") \
                    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
                    .option('topic', STATS_TOPIC) \
                    .save()
```

Implementação Intenções de Voto

Sink

```
lines = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
    .option("subscribe", PREDICT_TOPIC) \
    .option("failOnDataLoss", "false") \
    .load() \
    .writeStream \
    .foreachBatch(foreach_batch_func) \
    .option("checkpointLocation", "/tmp/spark/mllib-predict") \
    .trigger(processingTime=INTERVAL) \
    .start()
```

Muito Obrigado!!