PÓS-GRADUAÇÃO

Linguagens de programação para ciência de dados (*Python* com *Spark*)

PÓS-GRADUAÇÃO

Real Time Analytics com *Python* e *Spark*

Bloco 1

Marcelo Tavares de Lima

Objetivos

- Apresentar conceitos fundamentais de *real time analytics*.
- Descrever como se utiliza a API *Apache Spark Streaming* para analisar dados de diferentes fontes.
- Descrever como fazer integração da API *Apache Spark* com outras fontes de dados *Apache Kafka* e *Elasticsearch*.

Real Time Analytics com *Python* e *Spark*

- Análise em tempo real (*Real Time Analytics*) é um termo usado para se referir a análises de dados, que podem ser acessadas e processadas quando entram em um sistema.
- Em geral, o termo *análise* é usado para definir padrões de dados que fornecem significado (valor), onde os analistas coletam informações valiosas a partir dos dados gerados em tempo real.

Real Time Analytics com *Python* e *Spark*

- Para exemplificar, podemos citar rastreamento de estatísticas sobre visualizações e comentário de página *web*, monitoramento de estradas e rodovias, ou detecção automática de fraudes em transação de cartão de créditos, segundo Karau (2015).
- Essas análises de dados podem ser extremamente importantes para as empresas que desejam realizar relatórios dinâmicos para responder rapidamente às tendências do comportamento do usuário.

Real Time Analytics com *Python* e *Spark*

- O processamento em lote (*batch*) ocorre se é realizada análise dos dados que foram armazenados por um período de tempo.
- O processamento em lote é usado em situações em que não precisa de resultados de análises em tempo real e quando é mais importante processar grandes volumes de dados, a fim de obter informações mais detalhadas do que para obter resultados rápidos.
- O *Hadoop MapReduce*, segundo Dittrich (2012) é a melhor estrutura para o processamento de dados em lotes.

Real Time Analytics com *Python* e *Spark*

- Um fluxo de dados (*data stream*) é uma sequência ilimitada de dados que chega continuamente por diferentes fontes de dados.
- O processamento de fluxo (*streaming*) permite processar dados em tempo real à medida que chegam.
- Ajudam a detectar rapidamente as condições em um pequeno período a partir do ponto de recebimento dos dados.

Real Time Analytics com *Python* e *Spark*

- O processamento de *streaming* permite alimentar dados em ferramentas de análise, assim que são geradas, e a obter resultados instantâneos de análise.
- Existem várias plataformas de processamento de *streaming open-source*, como: Apache Kafka, segundo Garg (2013); Apache Flink, segundo Carbone (2015); Apache Storm, segundo Iqbal (2015); Apache Samza, segundo Noghabi (2015) etc.

Real Time Analytics com *Python* e *Spark*

- As desvantagens no processamento em *batch* é que o processamento ocorre sobre todos (ou a maioria) dos dados armazenados.
- No processamento em *streaming*, a análise é feita sobre os dados atuais ou mais recentes.
- O processamento em *batch* lida com um grande lote de dados, enquanto o processamento em *streaming* lida com registros individuais ou micro lotes de arquivos com poucos registros de dados.

Real Time Analytics com *Python* e *Spark*

- No ponto de desempenho, a latência do processamento em *batch* deve ocorrer em minutos ou horas, enquanto a latência do processamento de *streaming* ocorre em segundos ou milissegundos.

Apache Spark Streaming API

- Em 2012, o *Apache Spark* incorporou a biblioteca *Spark Streaming* e sua *API DStreams*, uma das primeiras APIs a ativar processamento de *streaming* usando operadores funcionais de alto nível, como mapear (*map*) e reduzir (*reduce*), e fornecer uma API em Scala, Java e *Python*.

Apache Spark Streaming API

- Muitas empresam usam *DStreams* na produção para aplicativos em tempo real, geralmente, processando *terabytes* de dados por hora.
- Muito parecido com a biblioteca de conjuntos de dados resilientes (*Resilient Distributed Dataset* - RDD). No entanto, a API do *DStreams* é baseada em operações de nível relativamente baixo em objetos Java e *Python*.

Apache Spark Streaming API

- Em 2016, o projeto *Apache Spark* adicionou o *Structured Streaming*, uma nova API de *streaming* criada diretamente em *DataFrames*, que suporta tanto otimizações e integração significativamente mais simples com outros códigos *DataFrame* e *Dataset*, segundo Chambers (2018).

Apache Spark Streaming API

- A biblioteca *Spark Streaming* pode ser usada para processar dados de *streaming* em tempo real de diferentes fontes, como sensores, dispositivos de IoT (*Internet Of Things*), redes sociais e transações on-line e os resultados gerados podem ser armazenados em softwares como Kafka, HDFS, Cassandra e Elasticsearch.

Apache Spark Streaming API

- O módulo *Apache Spark Streaming* fornece uma API para manipular fluxos de dados (*streaming*) que correspondem à API RDD ou também conhecido como PySpark RDD. Existem três fases do *Apache Spark Streaming*:
  - Coleta de dados (*gathering*).
    - Fontes básicas (*basic sources*).
    - Fontes avançadas (*advanced sources*).
  - Processamento (*processing*).
  - Armazenamento (*data storage*).

PÓS-GRADUAÇÃO

Real Time Analytics com *Python* e *Spark*

Bloco 2

Marcelo Tavares de Lima

Apache Spark Streaming API

- Desde o *Apache Spark* 2.0, os *DataFrames* e os *Datasets* podem representar dados estáticos e limitados, bem como dados ilimitados de *streaming*.
- Para criar *DataSet* e *DataFrames* de *streaming*, primeiramente, é necessário iniciar uma sessão de trabalho em *Python* usando a classe *SparkSession*.

Apache Spark Streaming API

- Em *Python*, os *DataFrames* de *streaming* podem ser criados por meio da interface *DataStreamReader* retornada por *SparkSession*.readStream().
- Semelhante à interface para criar *DataFrame* estático, pode-se especificar a fonte de dados de entrada (*datasources*).
- Na *API Spark Streaming* existem algumas funções para leitura de dados de diferentes fontes: arquivo semiestruturado (CSV, JSON, TXT, XML, ORC, PARQUET), socket e Apache Kafka.

Apache Spark Streaming API

- Código fonte:

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import
SparkSession
from pyspark.sql.types import StructType

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)
```

Continua...

Apache Spark Streaming API

- Código fonte – Continuação:

```
# Lendo textos de um socket
socketDF =
spark.readStream.format("<socket>").
option("host",
"<nome_host>").option("port",
<valor_porta>).load()
# Lendo todos os arquivos .csv no diretorio
userSchema = StructType().add("name",
"string").add("age", "integer")
csvDF = spark.readStream.option("sep",
";").schema(userSchema).
 .csv("/path/to/directory")
# Equivalent to
format("csv").load("/path/to/directory")
```

Por que utilizar RDD?

- O *Resilient Distributed Dataset* (RDD) é considerada a estrutura de dados mais importantes no *PySpark*.
- É um dos pioneiros na estrutura de dados sem esquema fundamental, que pode manipular dados estruturados e não estruturados.

Por que utilizar RDD?

- O compartilhamento de dados na memória torna os RDDs de 10 a 100 vezes mais rápidos que o compartilhamento de rede e disco.
- Uma característica importante do RDD é ser uma estrutura de dados imutável, ou seja, um objeto cujo estado não pode ser modificado após a criação, mas certamente pode ser transformado.

Por que utilizar RDD?

- A maiorias dos programas desenvolvidos em Apache Spark consistem em manipular RDDs, portanto, é importante conhecer as funcionalidades dos RDD:
  - Computações em memória (*In-Memory* Computations).
  - Avaliação preguiçosa (*Lazy Evaluation*).
  - Tolerante a falhas (*Fault Tolerant*).
  - Imutabilidade (*Immutability*).
  - Particionamento (*Partitioning*).
  - Persistência (*Persistence*).
  - Operações de granulação (*Coarse-Grained Operations*).

Apache Kafka

- É uma plataforma distribuída de código fonte, livre (*open-source*) de processamento de mensagens e *streams*, desenvolvida pela Apache *Software Foundation*, escrita na linguagem de programação Java e Scala.
- O principal objetivo do projeto Kafka é fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados em tempo real, segundo Garg (2013).

Apache Kafka

Exemplo de código *Python* para ler dados em tópicos e salvar o resultado na estrutura de dados (*DataFrame*).

```
import pyspark

from pyspark.context import SparkContext

from pyspark.sql.session import SparkSession

sc = SparkContext.getOrCreate()

spark = SparkSession(sc)

# Conectando em um tópico especifico

df1 = spark.readStream.format("kafka").

option("kafka.bootstrap.servers", "<host1:port1>,

<host2:port2>").option("subscribe", "topicoA").load()
```

Continua...

Apache Kafka

Exemplo de código *Python* para ler dados em tópicos e salvar o resultado na estrutura de dados (*DataFrame*).

Continuação...

```
# Conectando em vários tópicos

df2 =
spark.readStream.format("kafka")..option("kafka.bootstrap.servers",
"<host1:port1>,<host2:port2>").option("subscribe",

 "topicoA,topicoB").load()

# Conectando no tópico padrão (pattern)

df3 = spark.readStream.format("kafka").

option("kafka.bootstrap.servers",
"<host1:port1>,<host2:port2>").option("subscribePattern",

 "topic.*").load()
```

PÓS-GRADUAÇÃO

Teoria em prática

Bloco 3

Marcelo Tavares de Lima

Real Time Analytics

- Análise de dados em tempo real (*Real Time Analytics*) é um termo usado para se referir a análises de dados que podem ser acessadas e processadas quando entram em um sistema ou aplicações.
- Em geral, o termo *análise* é usado para definir padrões de dados que fornecem significado (valor), onde os analistas coletam informações valiosas a partir dos dados gerados em tempo real.

Real Time Analytics

- Um exemplo clássico de análise de dados em tempo real são os *logs* gerados pelas aplicações.
- Os arquivos de *logs* contêm muitas informações que podem ajudar na análise de erros que correm durante a execução das aplicações.
- Diante desse cenário, você poderá utilizar a Apache Spark *Streaming* (API) para criar algoritmos que possam fazer a leituras de arquivos de *logs* de servidores de aplicações (JBoss, Wildfly, Tomcat) para filtrar todos os erros inesperados durante a execução dos programas.

PÓS-GRADUAÇÃO

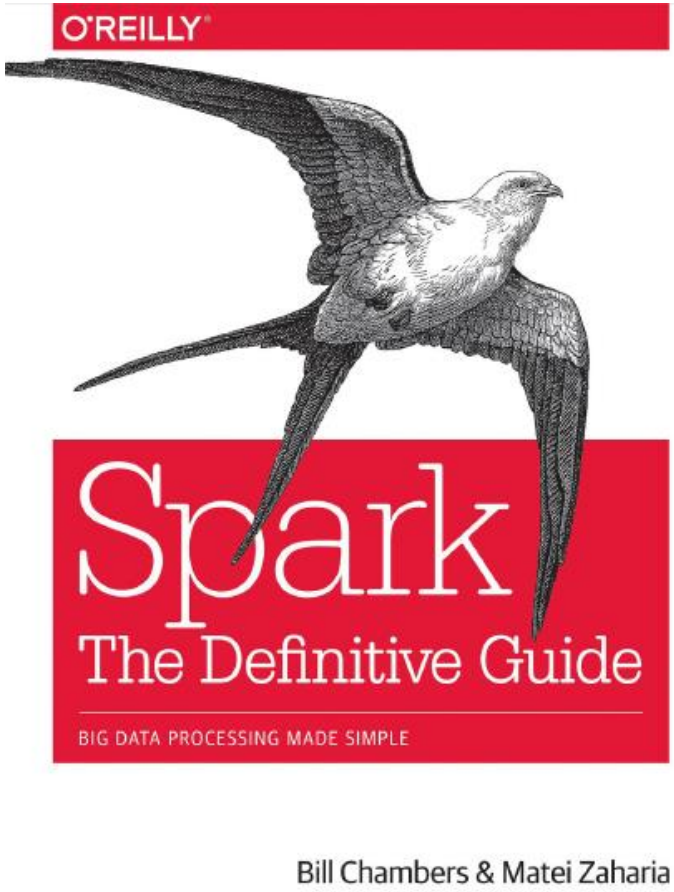
Dica do professor

Bloco 4

Marcelo Tavares de Lima

Dica de site

Figura 1 - Site



Título: *Spark – The definitive Guide*.

Autores: Bill Chambers e Matei Zaharia.

Editora: O'Reilly Media Inc.

Ano de publicação: 2018.

Fonte: https://www.amazon.com/Spark-Definitive-Guide-Processing-Simple-ebook/dp/B079P71JHY. Acesso em: 21 jan. 2020.

Referências

CARBONE, P.; KATSIFODIMOS, A.; EWEN, S. *et al.* Apache flink: stream and batch processing in a single engine. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, 2015.

CHAMBERS, B.; ZAHARIA, M. **Spark:** the definitive guide: Big Data processing made simple. San Francisco: O'Reilly Media, 2018.

DITTRICH, J.; QUIANÉ-RUIZ, J. Efficient Big Data processing in Hadoop MapReduce. **Proceedings of the VLDB Endowment**, v. 5, n. 12, 2012.

GARG, N. **Apache Kafka**. Packt Publishing Ltd, 2013.

Referências

IQBAL, M. H.; SOOMRO, T. R. Big Data analysis: Apache storm perspective. **International journal of computer trends and technology**, 2015.

KARAU, H.; KONWINSKI, A.; WENDELL, P. *et al.* **Learning spark**: lightning-fast Big Data analysis. O'Reilly Media, 2015.

NOGHABI, S. A.; PARAMASIVAM, K.; PAN, Y. *et al.* Samza: stateful scalable stream processing at LinkedIn. **Proceedings of the VLDB Endowment**, 2017.