

Текстовая расшифровка видео:

ПОТОКОВАЯ ОБРАБОТКА

План:

- Discretized Streams;
- Источники данных;
- Простой пример;
- Трансформации;
- Structured Streaming;
- Minibatch WordCount.

Discretized Streams

В Spark существует два варианта подхода к работе с потоковыми данными:

DStreams (Discretized Stream) – это подход, основанный на RDD.

Spark Structured Streaming – это подход, реализующийся поверх DataFrame. Данный подход считается более новым. **Официальная рекомендация:** любой новый стриминг следует писать с помощью Structured Streams.

Как работает DStreams?

Когда мы создаем RDD, мы можем создать целую пачку. Также, когда прилетают данные, мы можем из каждого фрагмента данных создавать новый RDD и каждый из этих RDD независимо процессить в параллель. Эта идея похожа на partitions, только в стриминговой парадигме:

- Каждый кусочек данных на кластере в RDD «размазан», данные прилетают в виде RDD.
- Мы применяем к ним стриминговую обработку.



▶ Запустить стенд



Дедлайн 07 июля, 23:59 Мск



- Пишем результат дальше в приемник.

Источники данных

Источников для работы с потоками два (условное разделение):

Простые источники – источники, базово поддерживаемые Spark'ом из коробки (чтение из файлов, чтение из сетевых сокетов).

Сложные источники – источники с более сложной стриминговой системой (Apache Kafka, AWS Kinesis и т.д.).

Простой пример

Рассмотрим простой пример того, как может выглядеть работа в парадигме DStreams:

```
1. from pyspark import SparkContext, SparkConf
2.
3. conf = SparkConf().setAppName("my-app").setMaster("local[4]")
4. sc = SparkContext(conf=conf)
5. ssc = StreamingContext(sc, 10)
6.
7. rdd_q = [
8.     sc.parallelize([j for j in range(1, 1001)], 10)
9. ]
10. input_stream = ssc.queueStream(rdd_q)
11. by_10_stream = input_stream.map(lambda x: (x % 10, 1))
12. sum_stream = by_10_stream.reduceByKey(lambda a, b: a + b)
13. sum_stream.pprint()
14.
15. ssc.start()
16. time.sleep(6)
17. ssc.stop(stopSparkContext=True, stopGraceFully=True)
```

Шаги:

- Создаем пачку RDD;
- Параллелизуем и разбиваем их на 10 кусков;
- В каждом из этих кусков есть числа от 1 до 1000. Для каждого из этих чисел считаем пару, где ключом будет остаток от деления числа на 10;
- В конце делаем reduceByKey, то есть суммируем, сколько раз каждый из результатов встречался в исходных данных.

Формально мы можем добавлять дополнительные RDD, они также будут обсчитывать результаты, а результат также будет вводиться.

В примере мы вводим результаты, запускаем, ждем выведения результата и полностью останавливаем с закрытием и корректным завершением всех промежуточных механизмов.

Посмотрим, как это может выглядеть.

У нас есть DStreams. Достанем SparkContext.

Обратите внимание: помимо стандартного SparkContext, SparkSession появился StreamingContext:

```

skew.ipynb x explain.ipynb x groupby_join.ipynb x StructuredStreaming.ipynb x DStreams.ipynb
Code
[1]: from pyspark import SparkContext, SparkConf
    from pyspark.streaming import StreamingContext

[ ]: import time

[ ]: conf = SparkConf().setAppName("my-app").setMaster("local[4]")
    sc = SparkContext(conf=conf)
    ssc = StreamingContext(sc, 10)
    socket_stream = ssc.socketTextStream("127.0.0.1", 5555)

[ ]: def simple_queue(ssc):
    ssc.queueStream([range(5), ['a', 'b'], ['c']], oneAtATime=True).pprint()

[ ]: rdd_q = [
    sc.parallelize([j for j in range(1, 1001)], 10)
]
input_stream = ssc.queueStream(rdd_q)
by_10_stream = input_stream.map(lambda x: (x % 10, 1))
sum_stream = by_10_stream.reduceByKey(lambda a, b: a + b)
sum_stream.pprint()

[ ]: ssc.start()
    time.sleep(6)

```

Поскольку в примере использована новая версия Spark, Spark «ругается» (DStreams deprecated) и просит взять новый Structured Streaming:

```

/usr/local/spark/python/pyspark/streaming/context.py:72: FutureWarning: DStream is deprecated as of Spark 3.4.0. Migrate to Structured Streaming.
  warnings.warn(

```

Обратите внимание на pprint:

```

[4]: rdd_q = [
    sc.parallelize([j for j in range(1, 1001)], 10)
]
input_stream = ssc.queueStream(rdd_q)
by_10_stream = input_stream.map(lambda x: (x % 10, 1))
sum_stream = by_10_stream.reduceByKey(lambda a, b: a + b)
sum_stream.pprint()

```

Он ничего не вывел. Он добавился, как некая трансформация, как команда «Что делать потоку с каждым кусочком данных, который туда прилетает». Когда мы запустим поток и подождем, данные начнут писаться:

Сделаем start. Видим, что порция вывелась. Закончились 6 секунд, мы завершились:

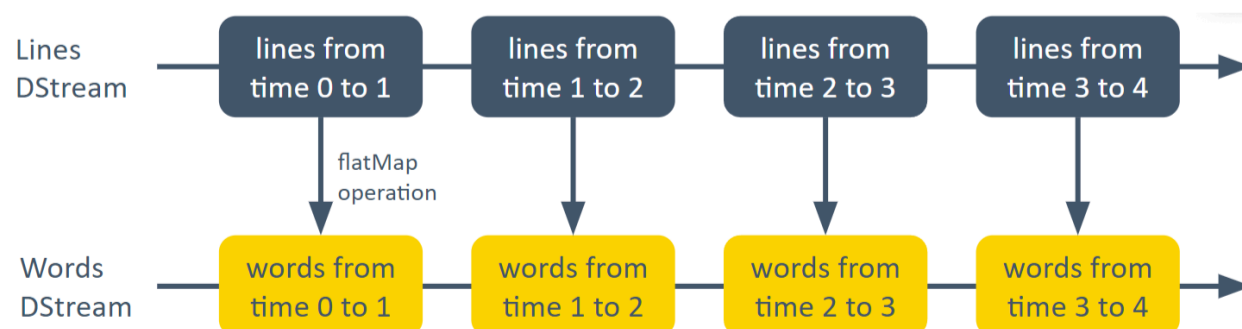
```

[*]: ssc.start()
    time.sleep(6)
    ssc.stop(stopSparkContext=True, stopGraceFully=True)

-----
Time: 2023-07-28 17:16:30
-----
(4, 100)
(8, 100)
(0, 100)
(1, 100)
(5, 100)
(9, 100)
(2, 100)
(6, 100)
(3, 100)
(7, 100)

```

Обратите внимание на схему:



Здесь показан пример, который называется «Разбиение текста на слова». Прилетает текст со строками, мы используем условный flatMap, нарезая текст по пробельному символу. Далее, в выходной поток летят RDD со словами, а не строками.

Многие из этих операций рассчитаны на применение чего-либо к разным рядам в RDD. Операции, например, `map()`, `filter()`, `flatMap()`, `repartition()` и т.д. работают для RDD так же, как и для DStreams. Отличие только в том, что они работают на потоке (для каждого из кусочков отдельно).

Отдельно стоит упомянуть [оконные функции](#). Поскольку мы работаем с потоком, стоит воспользоваться поддержкой «окошек», чтобы иметь «скользящее окно», которым мы сможем считать, например, скользящее среднее или среднее количество кликов в логах сайта.

Оконные функции:

- `window()`;
- `countByWindow()`;
- `reduceByKeyAndWindow()`;
- `countByValueAndWindow()`.

Претензия к Spark: это стриминг, а `minibatch`, потому что данные нарезаются на RDD, с каждым мы работаем, как с RDD и т.д.

Structured Streaming

Можно представить, что это виртуальная таблица, в конец которой постоянно записываются новые данные.

Structured Streaming позволяет использовать гораздо больше подкапотных оптимизаций благодаря Catalyst. На стриме к нему можно применять SQL-запросы, то есть можно использовать SQL даже для потоков.

Лучше интегрируется со сторонними плагинами, например, с MLLib.

Код во многом проще.

Minibatch WordCount

Рассмотрим пример:

```
1. from pyspark.sql import SparkSession
2. from pyspark.sql.functions import explode, split
3.
4. spark = SparkSession.builder.appName("SlurmStructuredStreamingDemo") \
5.     .getOrCreate()
6. lines = spark.readStream.format("socket") \
7.     .option("host", "127.0.0.1").option("port", 5555).load()
8. words = lines.select(explode(split(lines.value, " ")).alias("word"))
9. word_counts = words.groupBy("word").count()
10. query = word_counts.writeStream.outputMode("complete").format("console").start()
11. query.awaitTermination()
```

Мы делаем простейший WordCount, как мы уже делали это при работе с DataFrame. В примере используются похожие функции: разбиваем DataFrame по словам и считаем, сколько раз каждое слово встречалось в исходных данных. Нюанс: мы это делаем на потоке, то есть периодически смотрим на момент прихода очередной порции данных. Также следует обратить внимание на API. Раньше мы вызывали в обычном режиме `spark.read` и `spark.write`. Здесь мы вызываем `spark.readStream` и `spark.writeStream`. Мы читаем данные из какого-то источника (например, из сокета), пишем данные в какой-то приемник (например, в вывод самого Spark) и запускаем, как стриминговый `query`.

Чтобы это работало, перезапускаем локальный Spark с указанием NetWare Host, так как мы хотим читать с локального сокета:

```
meow-nofer@clefairy ~/Experiments/slurm$ docker run --rm --name pyspark -v $(pwd):/home/jovyan/jupyter/pyspark-notebook
Entered start.sh with args: jupyter lab
/usr/local/bin/start.sh: running hooks in /usr/local/bin/before-notebook.d as uid / gid: 1000 /
/usr/local/bin/start.sh: running script /usr/local/bin/before-notebook.d/spark-config.sh
/usr/local/bin/start.sh: done running hooks in /usr/local/bin/before-notebook.d
Executing the command: jupyter lab
```

Мы используем стандартную утилиту «NC», которая позволит закинуть чуть больше данных в сокет:

```
[I 2023-07-28 17:33:02.301 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2023-07-28 17:33:02.307 ServerApp]

To access the server, open this file in a browser:
file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html
Or copy and paste one of these URLs:
http://clefairy:8888/lab?token=e87fc85bce6a6dd7a802c5de92cd9e26ebd71376ba3f5358
http://127.0.0.1:8888/lab?token=e87fc85bce6a6dd7a802c5de92cd9e26ebd71376ba3f5358
[I 2023-07-28 17:33:04.120 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server,
-nodesjs, javascript-typescript-langserver, jedi-language-server, julia-language-server, pyright, python-language-server,
python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-language-server, unified-language-server,
vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yamll-language-server
[W 2023-07-28 17:33:05.216 ServerApp] 404 GET /api/kernels/85f5c18e-5165-4c80-b894-1a323bd42ee7?1690565585209 (127.0.0.1)
[0] 0:[tmux]* 1:zsh-
```

Создаем SparkSession. Здесь нам не нужно создавать StreamingContext и т.д.:

```
[*]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("SlurmStructuredStreamingDemo").getOrCreate()

[3]: from pyspark.sql.functions import explode, split
Используем netcat: nc -lk 5555 (в случае Docker потребует --network host) и попечатает что-нибудь

[4]: lines = spark.readStream.format("socket").option("host", "127.0.0.1").option("port", 5555).load()
[5]: words = lines.select(explode(split(lines.value, " ")).alias("word"))
[6]: word_counts = words.groupBy("word").count()
[7]: query = word_counts.writeStream.outputMode("complete").format("console").start()
[ ]: query.awaitTermination()
[ ]:
```

Теперь мы можем запустить nc с записью «5555». Набираем текст:

```
meow-nofer@clefairy ~/Experiments/slurm$ nc -lk 5555
abc abc abc asdasfdg
aesrga wrgaew raerg dfb
dfb g asd awawee abc
```

Создаем читалку из потока. Для каждой новой порции будем разбивать по словам, группировать, считать word_count. У нас уже запущен query. Выполняем start и ждем завершения:

```
skew.ipynb explain.ipynb groupby_join.ipynb StructuredStreaming.ipynb DStreams.ipynb
[1]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("SlurmStructuredStreamingDemo").getOrCreate()
[2]: from pyspark.sql.functions import explode, split
Используем netcat: nc -lk 5555 (в случае Docker потребует --network host) и попечатает что-нибудь
[3]: lines = spark.readStream.format("socket").option("host", "127.0.0.1").option("port", 5555).load()
[4]: words = lines.select(explode(split(lines.value, " ")).alias("word"))
[5]: word_counts = words.groupBy("word").count()
[6]: query = word_counts.writeStream.outputMode("complete").format("console").start()
[*]: query.awaitTermination()
[ ]:
```

Если посмотреть на логи Spark, мы увидим, что запустилась Streaming Job'a, пошел Stage, создалась query на потоке:

```
Alacrity meow-nofer - Thunar
[I 2023-07-28 17:33:30.607 ServerApp] Connecting to kernel 51750bfc-e9ba-4c87-b4fc-33bab23f24e3.
[I 2023-07-28 17:33:30.643 ServerApp] Starting buffering for c996e93b-6d8e-41de-94b2-d3435d49d408.
4c-53f0fa0f7c9f
[I 2023-07-28 17:33:30.660 ServerApp] Connecting to kernel 12c608d3-cfd3-4e10-9e13-af9e11a76673.
[I 2023-07-28 17:33:30.758 ServerApp] Connecting to kernel 61d24bf0-957d-4eba-a355-ee9ca3badce6.
[I 2023-07-28 17:33:30.838 ServerApp] Connecting to kernel c996e93b-6d8e-41de-94b2-d3435d49d408.
[I 2023-07-28 17:33:30.910 ServerApp] Connecting to kernel c996e93b-6d8e-41de-94b2-d3435d49d408.
23/07/28 17:33:36 WARN Utils: Your hostname, clefairy resolves to a loopback address: 127.0.1.1;
stead (on interface wlan0)
23/07/28 17:33:36 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/07/28 17:33:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java c
lasses where applicable
23/07/28 17:34:21 WARN TextSocketSourceProvider: The socket source should not be used for production applications! It d
oes not support recovery.
23/07/28 17:34:30 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the q
uery didn't fail: /tmp/temporary-71deeac9-3d2c-4152-9d39-c58dced8aa15. If it's required to delete it under any circumst
ances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkp
oint folder is best effort.
23/07/28 17:34:30 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datase
ts and will be disabled.
[Stage 1:=====> (98 + 8) / 200]
[0] 0:docker* 1:nc- "clefairy" 19:34 28-Jul-23
```

Когда мы будем писать новые данные на пс, они должны появляться здесь:

```
23/07/28 17:34:30 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datase
ts and will be disabled.
-----
Batch: 0
-----
+-----+
|word|count|
+-----+
+-----+

[Stage 3:=====> (189 + 9) / 200]
[0] 0:docker* 1:nc- "clefairy" 19:34 28-Jul-23
```

Видим, что он посчитал данные, которые мы закинули ранее:

```
Alacrity meow-nofer - Thunar
|word|count|
+-----+
+-----+

-----
Batch: 1
-----
+-----+
| word|count|
+-----+
| g| 1|
| wrgaew| 1|
| raerg| 1|
| dfb| 2|
| asd| 1|
| abc| 4|
| | 1|
| awawee| 1|
| asdasfsdg| 1|
| aesrga| 1|
+-----+
```

Можем написать что-либо еще:

```
Alacrity
meow-nofer@clefairy ~/Experiments/slurm nc -lk 5555
abc abc abc asdasfsdg
aesrga wrgaew raerg dfb
dfb g asd awawee abc
sadf dsgj asfg areg sfg srsa g
dhs df afh s hasdg adf
safg dh afs
```

```
+-----+
| word | count |
+-----+
| srsa | 1 |
| g | 2 |
| wrgaew | 1 |
| asfg | 1 |
| raerg | 1 |
| dfb | 2 |
| dsgj | 1 |
| sadf | 1 |
| areg | 1 |
| asd | 1 |
| sfg | 1 |
| abc | 4 |
| | 1 |
| awawee | 1 |
| asdasfsdg | 1 |
| aesrga | 1 |
+-----+

[Stage 7:===> (11 + 8) / 200]
[0] 0:docker* 1:nc-
```

Если вы скажете, что это minibatch, то будете правы. Мы действительно обрабатываем данные «батчами». Однако это Stream processing и то, как работает современный Spark Streaming.

Рекомендуем поэкспериментировать с этим самостоятельно.

ИТОГИ

Мы узнали:

- Что такое логический и физический план;
- Как бороться с неравномерным распределением данных по кластеру.
- Как кэшировать промежуточные результаты вычислений.
- Как с помощью spark можно решать задачи потоковой обработки.

Как вам урок?



Изучил, далее >>

Слёрм ©

[+7 \(495\) 248-05-80](tel:+7(495)248-05-80)

[Лицензия №ДЛ-1368 от 22.08.2019](#)

[Политика конфиденциальности](#)

[Публичная оферта](#)