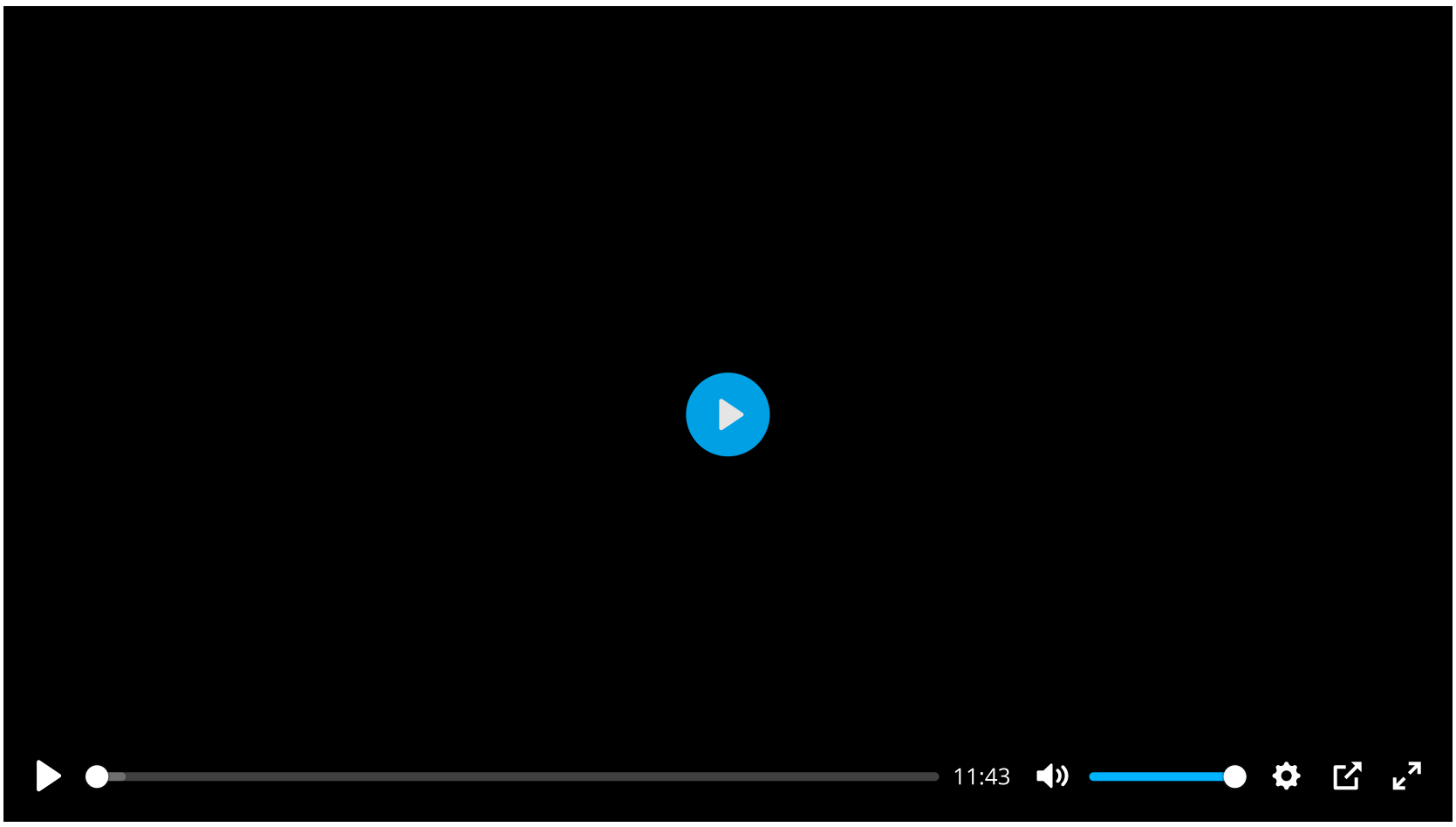
4.2 Что такое Spark и зачем он нужен DE. Введение в Resilient Distributed Datasets



Текстовая расшифровка видео:

КЛЮЧИ И ЗНАЧЕНИЯ

План:

- Старые-добрые ключи-значения;
- WordCount;
- Join’ы.

Старые-добрые ключи-значения

У нас есть датасет с городами:

```
1. pair_rdd = rdd.flatMap(lambda x: x.lower()).map(lambda x: (x, 1))
2. print(pair_rdd.take(4))
3. # [('m', 1), ('o', 1), ('s', 1), ('c', 1)]
4. counts: dict = pair_rdd.countByKey()
5. # {'m': 2, 'o': 5, 's': 2, 'c': 1, 'w': 2, 'p': 1, 'a': 2, ... }
```

Мы разравниваем его flatMap’ом и превращаем в пары ключ-значение. В контексте Python мы возвращаем кортежи (tuple), где первый элемент – ключ, второй – значение. Получаем RDD, где каждый элемент представляет собой tuple. У RDD с такими данными может сработать пара интересных методов, например, **countByKey()**.

Если мы хотим посчитать сколько раз буквы встречаются в исходном датасете, то мы можем выполнить данный код:

```
counts: dict = pair_rdd.countByKey()
```

Важно помнить, что **countByKey()** возвращает слова. Это action, который перешлет результат всех вычислений на мастер-ноду. Это не всегда может совпадать с нашими планами.

```
[13]: pair_rdd = rdd.flatMap(lambda x: x.lower()).map(lambda x: (x, 1))
      print(pair_rdd.take(4))
      # [('m', 1), ('o', 1), ('s', 1), ('c', 1)]
      counts: dict = pair_rdd.countByKey()

      [('m', 1), ('o', 1), ('s', 1), ('c', 1)]

[ ]:
```

У нас получился набор элементов. Посмотрим, что лежит в counts:

```
[14]: counts

[14]: defaultdict(int,
                  {'m': 2,
                   'o': 5,
                   's': 2,
                   'c': 1,
                   'w': 2,
                   'p': 1,
                   'a': 2,
                   'r': 3,
                   'i': 2,
                   'd': 3,
                   'l': 1,
                   'n': 3,
                   'e': 1,
                   ' ': 1,
                   'y': 1,
                   'k': 1})
```

Здесь лежит большой словарь с буквами и их количеством повторений в датасете.

Есть альтернативный вариант – **ReduceByKey**, который является не action, а transformation:

```
1. letter_count: RDD = pair_rdd.reduceByKey(lambda x, y: x + y)
2. print(letter_count.take(3))
3. # [('p', 1), ('d', 3), ('l', 1)]
```

Мы можем сказать ReduceByKey передать лямбду, как в случае обычного reduce'а, где на выходе получим RDD, с которым далее можем что-либо считать.

В целях гибкости по умолчанию стоит использовать **ReduceByKey** и задавать конкретный предикат свертки, с помощью которого мы будем один RDD преобразовывать в другой. Если вашей задачей является подсчет по ключам, то countByKey() – ваш друг.

Запомним:

- **countByKey()** – action;
- **ReduceByKey** – transformation.

Выполним код **ReduceByKey** и проверим:

```
[15]: letter_count: RDD = pair_rdd.reduceByKey(lambda x, y: x + y)
      print(letter_count.take(3))

      [('p', 1), ('d', 3), ('l', 1)]

[ ]:
```

В данном случае letter_count – RDD, и данные «размазаны» по кластеру. Пока мы не вызвали YAML take, никакие вычисления не запускались.

WordCount

WordCount может выглядеть примерно следующим образом:

```
1. text_rdd = sc.textFile("Experiments/slurm/76-0.txt")
2. text_rdd.flatMap(
3.   lambda l: l.split(" ")
4. ).map(lambda w: (w, 1)).reduceByKey(
5.   lambda x, y: x + y
```

Дедлайн 07 июля, 23:59 Мск

Мы читаем текст с помощью textFile, делаем flatMap (разбиваем по пробелам и схлопываем, чтобы была последовательность любых непробельных символов). Мы считаем ее за слово. Преобразуем все к ключу-значению с единичками, делаем Reduce и смотрим топ-10 самых встречаемых слов.

Выполним данный код:

```
[16]: text_rdd = sc.textFile("76-0.txt")
      text_rdd.flatMap(
        lambda l: l.split(" ")
      ).map(lambda w: (w, 1)).reduceByKey(
        lambda x, y: x + y
      ).top(10, key=lambda i: i[1])
```

```
[16]: [('and', 6048),
       ('the', 4705),
       ('a', 2935),
       ('to', 2903),
       ('', 2736),
       ('I', 2476),
       ('was', 1942),
       ('of', 1732),
       ('it', 1427),
       ('he', 1372)]
```

Мы быстро получили результат с встречающимися словами.

Join’ы

Когда данные представлены в формате ключ-значение, мы можем применить к ним нужный тип Join’а (left, right, auto и т.д.) и получить какой-либо результат.

Например, у нас есть текст, который мы прочитали в RDD:

```
1. favorite_letters = ['a', 'd', 'o']
2. fav_let_rdd = sc.parallelize(favorite_letters).map(lambda x: (x,1))
3. joined = letter_count.leftOuterJoin(fav_let_rdd)
```

Мы хотим посмотреть, в каких строчках встречаются любимые буквы. Мы разбиваем текст на буквы, Join’им датасет с исходным текстом (с буквами) и датасет с любимыми буквами. Как это сделать?

Мы создаем датасет с любимыми буквами (список), параллелизуем его в пары ключ-значение, Join’им одно с другим:

```
[17]: favorite_letters = ['a', 'd', 'o']
      fav_let_rdd = sc.parallelize(favorite_letters).map(lambda x: (x,1))
      joined = letter_count.leftOuterJoin(fav_let_rdd)
```

```
[ ]:
```

Пока что Join – «ленивая» операция, и мы ничего не запустили. У нас создался еще один RDD. Нам нужно вывести результат в каком-то виде. Посчитаем на основе результатов Join’а, где есть любимы буквы, а где – нет. Далее, составим строчки с результатом, а потом их выведем.

Результатом Join’а будет хитрая конструкция: кортеж из двух элементов, где первый – это ключ, по которому мы Join’или, то есть буква, а второй элемент – это еще один кортеж из двух элементов в значении из левой таблицы и правой:

```
[17]: favorite_letters = ['a', 'd', 'o']
      fav_let_rdd = sc.parallelize(favorite_letters).map(lambda x: (x,1))
      joined = letter_count.leftOuterJoin(fav_let_rdd)
```

```
[ ]: def get_favorite(letter_info):
       letter, (left_count, right_count) = letter_info
       match right_count:
         case None:
           return f"The letter {letter} is not my favorite"
         case _:
           return f"The letter {letter} is my favorite and appears in text {left_count} times"

     joined.map(get_favorite).collect()
```

Ситуации могут быть следующие: буква есть и слева, и справа, что значит, что она есть в датасете

Дедлайн 07 июля, 23:59 Мск

Также мы использовали питоновский паттерн «Match». Поскольку мы используем leftOuterJoin, none'ы могут быть в правом count'е, потому что Join'им левый letter_count с правым favorite_letters. Далее смотрим: если справа – none, то пишем, что буква нелюбимая; если none нет, то любимая буква присутствует и, соответственно, пишем, что в исходном тексте она встречается столько-то раз (информация есть в исходном датасете). В итоге получаем следующий результат:

```
    case _:
      return f"The letter {letter} is my favorite and appears in text {left_count} times"

joined.map(get_favorite).collect()
```

```
[18]: ['The letter p is not my favorite',
       'The letter d is my favorite and appears in text 3 times',
       'The letter   is not my favorite',
       'The letter e is not my favorite',
       'The letter y is not my favorite',
       'The letter c is not my favorite',
       'The letter m is not my favorite',
       'The letter w is not my favorite',
       'The letter a is my favorite and appears in text 2 times',
       'The letter l is not my favorite',
       'The letter s is not my favorite',
       'The letter n is not my favorite',
       'The letter k is not my favorite',
       'The letter o is my favorite and appears in text 5 times',
       'The letter r is not my favorite',
       'The letter i is not my favorite']
```

ИТОГИ

Мы рассмотрели следующее:

- Как пробовать Spark у себя;
- Как утроено низкоуровневое API в лице RDD;
- В чем отличие между трансформациями (transformations) и действиями (actions);
- Как делать распределенные операции с ключами и значениями.

Как вам урок? 😠 😐 🔥

Изучил, далее ›

Дедлайн 07 июля, 23:59 Мск