Apache Spark advanced

Использование Apache Spark

RDD, Сериализация

- “Из коробки” используется встроенная java сериализация
- Стандартно поддерживается библиотека сериализации Kryo, которая не требует от программиста дополнительного кода (в отличие от стандартного для hadoop api)

Инициализация

- Для управления настройками spark используется класс

  org.apache.spark.SparkConf

- Взаимодействие с системой spark осуществляется через экземпляр класса

  org.apache.spark.api.java.JavaSparkContext

- Пример

```
SparkConf conf = new SparkConf().setAppName("sample");
JavaSparkContext sc = new JavaSparkContext(conf);
```

Загрузка данных в RDD

- Для разработки можно применять метод parallelize

```
JavaRDD<String> lines = sc.parallelize(Arrays.asList("pandas",
"i like pandas"));
```

- Загрузка файла из HDFS

```
JavaRDD<String> lines = sc.textFile("/path/to/README.md");
```

- Загрузка данных из Hadoop InputFormat

```
JavaPairRDD<LongWritable, Text> data =
    sc.hadoopFile("war-and-peace-1.txt",
                  TextInputFormat.class,
                   LongWritable.class,
                   Text.class);
```

Функции преобразования

- Преобразование RDD производится с помощью лямбда-функций scala (и java8) или обычных функций для java7
- Пример на Scala

```
val errorsRDD=inputRDD.filter(line=>line.contains("error"))
```

- Пример java 8

```
JavaRDD<String> errorsRDD=inputRDD.filter(
                                    x -> x.contains("error")
                                );
```

Операции над RDD
Преобразования

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic

Операции над двумя RDD

Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training data).	`rdd.subtract(other)`	`{1, 2}`
`cartesian()`	Cartesian product with the other RDD.	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

Actions

- Загрузка из HDFS

```
JavaRDD<String> distFile = sc.textFile("war-and-peace-1.txt");
```

- Сохранить в HDFS

```
res.saveAsTextFile("result");
```

- Скачать данные

```
res.collect()
```

- Количество записей

```
res.count()
```

Работа с парами KeyValue

- Пары KeyValue(Tuple2) применяются для операций работающих с “группами” значений:

  reduceByKey – применяет функцию ко всем значениям имеющим одинаковый ключ во всем RDD

  Join – генерирует пары значения для заданного ключа

  ...

- Создать RDD вида KeyValue

```
JavaPairRDD<String, Long> wordsWithCount = splitted.mapToPair(
                                        s -> new Tuple2<>(s, 1l)
                                    );
```

KeyValue transformations

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey( (x, y) => x + y)`	`{(1, 2), (3, 10)}`
`groupByKey()`	Group values with the same key.	`rdd.groupByKey()`	`{(1, [2]), (3, [4, 6])}`
`combineBy Key(createCombiner, mergeValue, mergeCombiners, partitioner)`	Combine values with the same key using a different result type.	See Examples 4-12 through 4-14.	

KeyValue transformations

`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`
`flatMapValues(func)`	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	`rdd.flatMapValues(x => (x to 5)`	`{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)}`
`keys()`	Return an RDD of just the keys.	`rdd.keys()`	`{1, 3, 3}`

KeyValue transformations

`values()`	Return an RDD of just the values.	`rdd.values()`	`{2, 4, 6}`
`sortByKey()`	Return an RDD sorted by the key.	`rdd.sortByKey()`	`{(1, 2), (3, 4), (3, 6)}`

Пример работы с функцией
combineByKey

```
public class AvgCount implements
Serializable {
    private long total;
    private long counter;
    public long getTotal() {
        return total;
    }
    public long getCounter() {
        return counter;
    }
    public AvgCount(long total, long counter) {
        this.total = total;
        this.counter = counter;
    }
```

```
    public static AvgCount addValue
            (AvgCount a, long value) {
        return new AvgCount(
                a.getTotal() + value,
                a.getCounter() + 1l   );
    }
    public static AvgCount add
                    ( AvgCount a, AvgCount b) {
        return new AvgCount(
                a.getTotal() + b.getTotal(),
               a.getCounter() +b.getCounter()
        );
    }
    public float avg() {
        return total / (double) counter;
    }
}
```

Вызов combine by key

```
JavaPairRDD<String, AvgCount> avgCounts =
        nums.combineByKey(
                p -> new AvgCount(p.getValue(), 1l),
               (avgCount, p) → AvgCount.addValue(
                                       avgCount,
                                       p.getValue()),
                AvgCount::add
        );
```

Пример работы с функцией
combineByKey

Partition 1

coffee	1
coffee	2
panda	3

Partition 2

coffee	9

```
def createCombiner(value):
  (value, 1)

def mergeValue(acc, value):
  (acc[0] + value, acc[1] +1)

def mergeCombiners(acc1, acc2):
  (acc1[0] + acc2[0], acc1[1] + acc2[1])
```

Partition 1 trace:
(coffee, 1) -> new key
accumulators[coffee] = createCombiner(1)
(coffee, 2) -> existing key
accumulators[coffee] = merge Value(accumulators[coffee], 2)
(panda, 3) -> new key
accumulators[panda] = createCombiner(3)

Partition 2 trace:
(coffee, 9) -> new key
accumulators[coffee] = createCombiner(9)

Merge Partitions:
mergeCombiners(partition1.accumulators[coffee],
partition2.accumulators[coffee])

Работа с двумя RDD KeyValue

Table 4-2. Transformations on two pair RDDs (rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)})

Function name	Purpose	Example	Result
`subtractByKey`	Remove elements with a key present in the other RDD.	`rdd.subtractByKey(other)`	`{(1, 2)}`
`join`	Perform an inner join between two RDDs.	`rdd.join(other)`	`{(3, (4, 9)), (3, (6, 9))}`
`rightOuterJoin`	Perform a join between two RDDs where the key must be present in the first RDD.	`rdd.rightOuterJoin(other)`	`{(3,(Some(4),9)), (3,(Some(6),9))}`
`leftOuterJoin`	Perform a join between two RDDs where the key must be present in the other RDD.	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}`
`cogroup`	Group data from both RDDs sharing the same key.	`rdd.cogroup(other)`	`{(1,([2],[])), (3, ([4, 6],[9]))}`

Actions с парами KeyValue

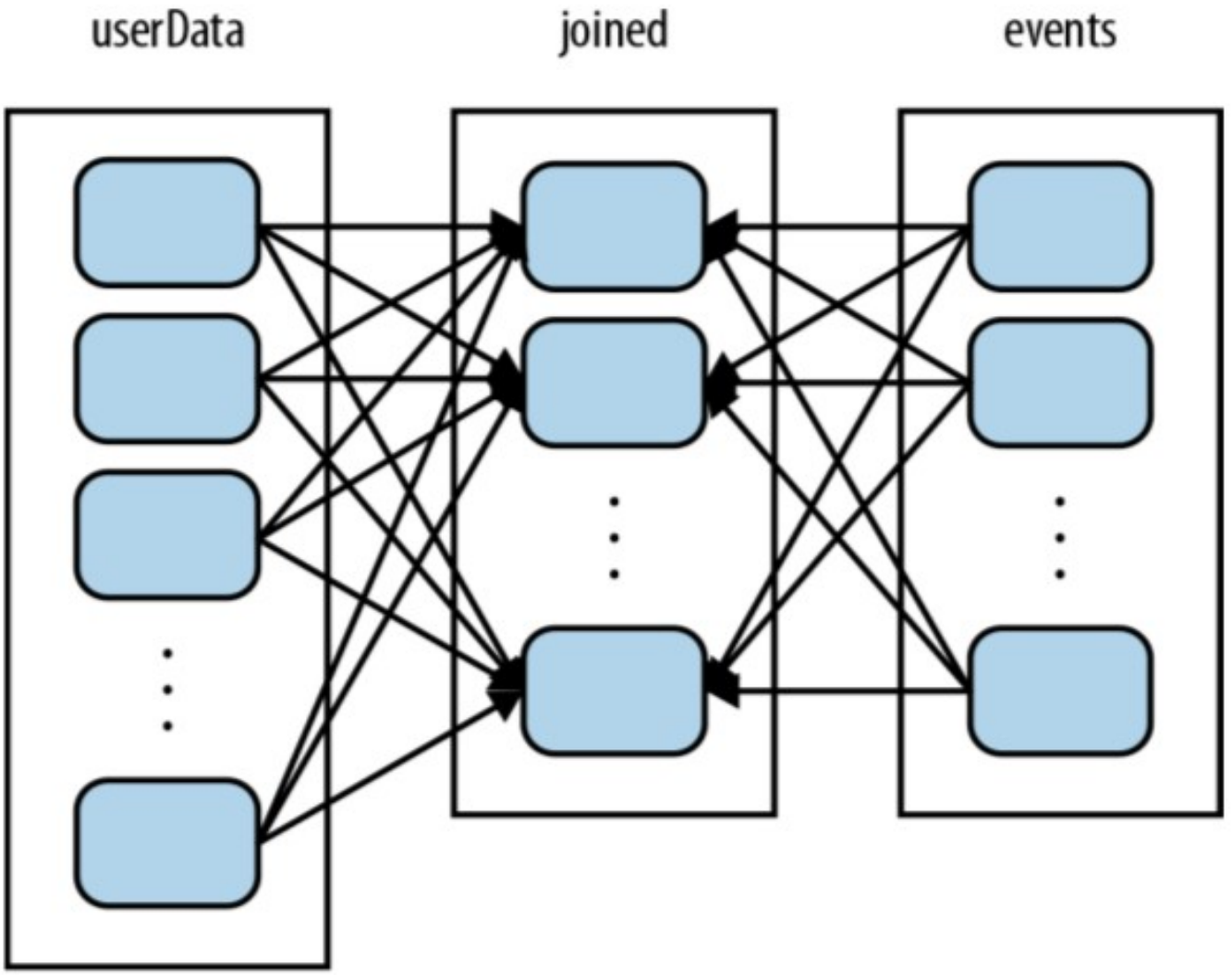
Table 4-3. Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key.	`rdd.countByKey()`	`{(1, 1), (3, 2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup.	`rdd.collectAsMap()`	`Map{(1, 2), (3, 4), (3, 6)}`
`lookup(key)`	Return all values associated with the provided key.	`rdd.lookup(3)`	`[4, 6]`

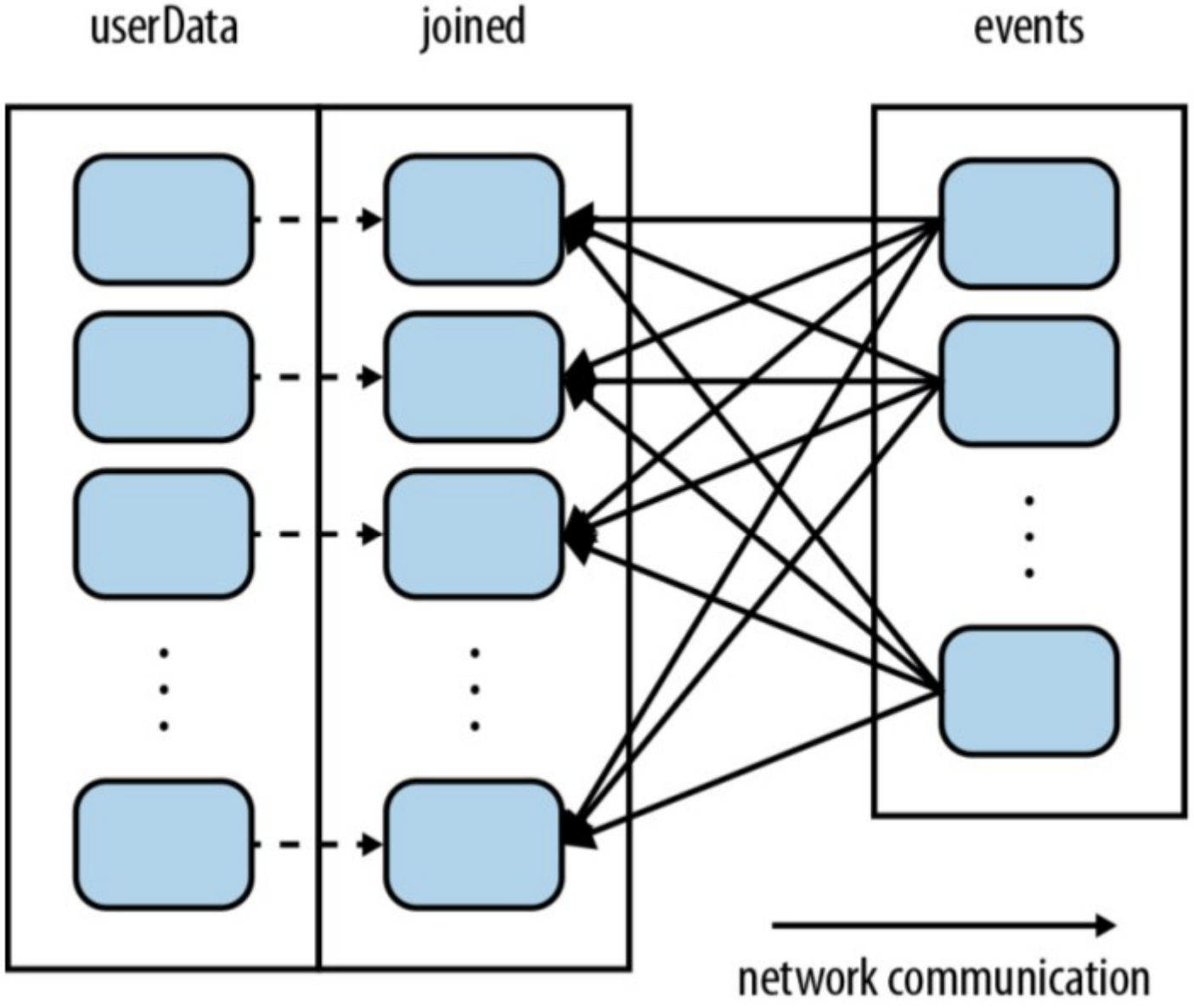
Data partitioning

- По умолчанию нельзя быть уверенным в том как распределены данные по кластеру
- В ряде операций это неэффективно
- partitionBy(partitioner) управляет распределением RDD по кластеру
- Операция repartition() генерирует новый RDD распределенный по кластеру – предназаначенна для равномерного распределения данных

Join непартиционированных данных



Join партиционированных данных



Persistance, Caching

- В случае если промежуточные данные требуется сохранить между вычислениями – их можно сохранить в памяти узлов
- persist(level) – сохраняет данные на узлах
- unpersist() - удяляет сохраненные данные

Аккумуляторы

- Предназначены для вычисления глобальных значений для всего вычисления
- Создаются в глобальном контексте и доступны в лямбда функциях
- Значение аккумулятора недоступно в функциях во время выполнениия
- Результирующее значение аккумулятора доступно после завершения вычислений

Использование аккумулятора

```
….
final LongAccumulator total =
jsc.sc().longAccumulator();
….
JavaPairRDD<String, Long> collectedWords =
wordsWithCount.reduceByKey((a, b) -> {
        total.add(1l);
        return a + b;
    });
System.out.println("accumulator value="+total.value());
```

Broadcast variables

- Механизм broadcast служит для рассылки всем узлам одного и того же набора данных
- Набор данных сериализируется,эффективно рассылается на все узлы и доступен внутри функций

```
final Broadcast<Map<String, AirportData>> airportsBroadcasted =
sc.broadcast(stringAirportDataMap);


JavaRDD<String> distFile =
sc.textFile("664600583_T_ONTIME_sample_cutted.csv");


JavaRDD<ParsedData> splitted = distFile.map(
        s -> new ParsedData(s, airportsBroadcasted.value())
);
```

Операции внутри Partitions

- Функции внутри partitions применяются для всех значений RDD внутри одного partition
- Применяются для эффективной работы с внешними ресурсами которые трудоемко создавать (например подключения к базе данных)

Операции внутри Partitions

Function name	We are called with	We return	Function signature on RDD[T]
`mapPartitions()`	Iterator of the elements in that partition	Iterator of our return elements	`f: (Iterator[T]) → Iterator[U]`
`mapPartitionsWithIndex()`	Integer of partition number, and Iterator of the elements in that partition	Iterator of our return elements	`f: (Int, Iterator[T]) → Iterator[U]`
`foreachPartition()`	Iterator of the elements	Nothing	`f: (Iterator[T]) → Unit`

Dataframe, Dataset

- Альтернативный подход spark к работе с данными
- Dataset – набо данных аналогично RDD “размазанный по кластеру”
- В отличие от RDD – Dataset имеет информацию о полях и может ее использовать для оптимизации запросов
- Строка Row – аналогична строкам в базах данных, имеет набор типизированных полей

Виды Dataset

- Typed – содержит набор java объектов. Требует Encoder который предоставит схему сериализации объекта как Row
- Untyped – содержит Row

На что это похоже

- Dataset<Row> people = spark.read().parquet("...");
- Dataset<Row> department = spark.read().parquet("...");
-
- people.filter(people.col("age").gt(30))
- .join(department, people.col("deptId").equalTo(department.col("id")))
- .groupBy(department.col("name"), people.col("gender"))
- .agg(avg(people.col("salary")), max(people.col("age")));