RDD 广播变量、累加器、持久化

2020 年 11 月 30 日

1 Broadcast 广播变量

共享变量（Shared variable）可用于节省内存与运行时间，提高并行处理的执行效率，共享变量包括 - Broadcast–广播变量 - accumulator–累加器

创建 kvFruit

[1]:
```
kvFruit = sc.parallelize([(1,"apple"),(2,"orange"),(3,"banana"),(4,"grape")])
```

创建 fruitMap 字典

[2]:
```
fruitMap=kvFruit.collectAsMap()
print("dict:"+str(fruitMap))
```

```
dict:{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

创建 fruitIds

[3]:
```
fruitIds=sc.parallelize([2,4,1,3])
print("fruitIds:"+str(fruitIds.collect()))
```

```
fruitIds:[2, 4, 1, 3]
```

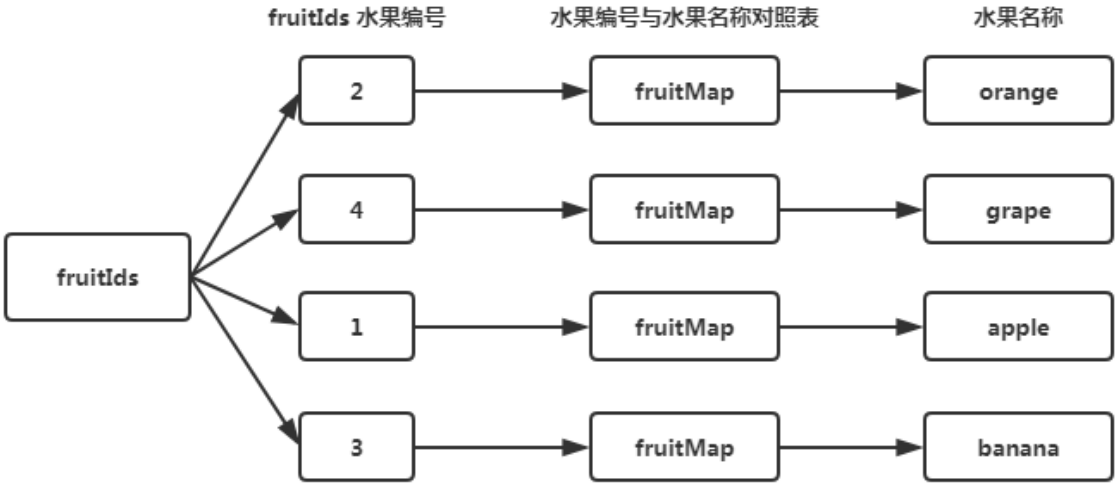
使用 fruitMap 字典进行转换

[4]:
```
fruitNames=fruitIds.map(lambda x:fruitMap[x]).collect()
print("fruitNames:"+str(fruitNames))
```

```
fruitNames:['orange', 'grape', 'apple', 'banana']
```

以上的例子执行起来虽然没问题，但是在并行处理中每执行一次转换都必须将 fruitIds 与 fruitMap 传到 Worker 节点，才能够执行转换。如果字典 fruitMap（对照表）很大，而且需要转换的 fruitIds

水果编号 RDD 也很大，就会消耗很多内存与时间



为了解决这个问题，可以使用 Broadcast 广播变量。

Broadcast 广播变量的使用规则如下：

- 可以使用 SparkContext.broadcast([初始值]) 创建。
- 使用.value 的方法来读取广播变量的值
- Broadcast 广播变量被创建后不能修改

下列使用 Broadcast 广播变量的例子与之前的例子类似，不同之处是使用 sc.broadcast 传入 fruitMap 作为参数，创建 bcFruitMap 广播变量，使用 bcFruitMap.value(x) 广播变量转换为 fruitNames 水果名称。

创建 kvFruit

[5]:
```
kvFruit = sc.parallelize([(1,"apple"),(2,"orange"),(3,"banana"),(4,"grape")])
```

创建 fruitMap 字典

使用 collectAsMap 创建 fruitMap 字典（水果编号与名称对照表）

[6]:
```
fruitMap=kvFruit.collectAsMap()
print("dict:"+str(fruitMap))
```

```
dict:{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

将 fruitMap 字典转换为 bcFruitMap 广播变量

使用 sc.broadcast 传入 fruitMap 参数，创建 bcFruitMap 广播变量

[11]:
```
bcFruitMap=sc.broadcast(fruitMap)
print("dict:"+str(fruitMap))
```

```
dict:{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

创建 fruitIds

[8]:
```
fruitIds=sc.parallelize([2,4,1,3])
print("fruitIds:"+str(fruitIds.collect()))
```

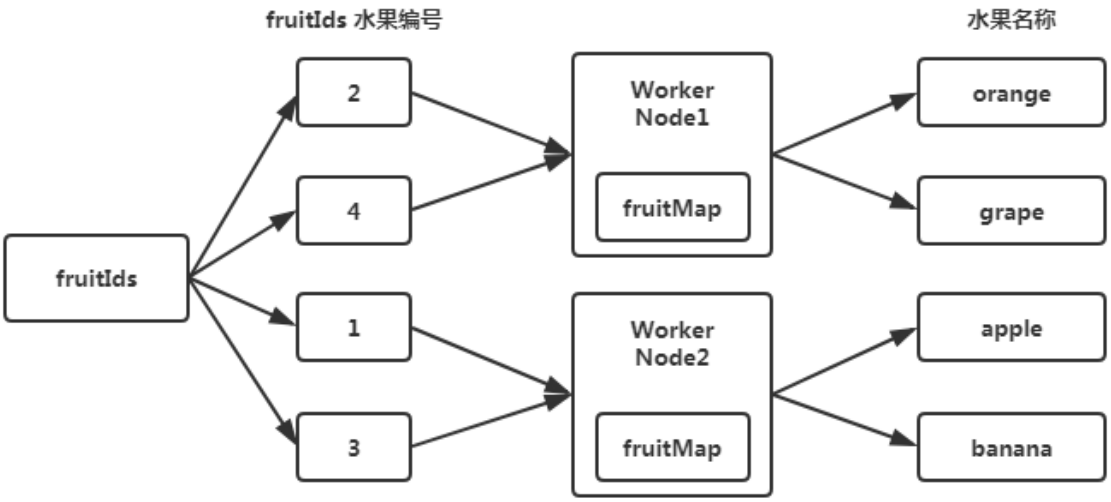
```
fruitIds:[2, 4, 1, 3]
```

使用 bcFruitMap.value[x] 广播变量将 fruitIds 转换为 fruitNames

[9]:
```
fruitNames=fruitIds.map(lambda x:bcFruitMap.value[x]).collect()
print("fruitNames:"+str(fruitNames))
```

```
fruitNames:['orange', 'grape', 'apple', 'banana']
```

执行结果与之前的方法相同，在并行处理中 bcFruitMap 广播变量会传送到 Worker Node 机器，并且存储在内存中，后续在此 Worker Node 都可以使用这个 bcFruitMap 广播变量执行转换，这样就可以节省很多内存与传送时间。



2 accumulator 累加器

计算总和是 MapReduce 常用的运算。为了方便并行处理，Spark 特别提供了 accumulator 累加器共享变量（Shared variable），使用规则如下：- accumulator 累加器可以使用 SparkContext.accumulator([初始值]) 来创建。- 使用.add() 进行累加 - 在 task 中，例如 foreach 循环中，

不能读取累加器的值 - 只有驱动程序，也就是循环外，才可以使用.value 来读取累加器的值

[12]:
```
intRDD = sc.parallelize([3,1,2,5,5])
```

创建 total 累加器，初始值使用 0.0，所以是 Double 的类型

[13]:
```
total = sc.accumulator(0.0)
```

创建 num 累加器，初始值使用 0，所以是 Int 类型

[14]:
```
num = sc.accumulator(0)
```

使用 foreach 传入参数 i，针对每一项数据执行，total 累加 intRDD 元素的值、num 累加 intRDD 元素的数量

[15]:
```
intRDD.foreach(lambda i : [total.add(i),num.add(1)])
```

计算平均 = 求和/计数，并显示总和、数量

[16]:
```
avg = total.value/num.value
print("total="+str(total.value)+"num="+str(num.value)+"avg="+str(avg))
```

```
total=16.0num=5avg=3.2
```

3 RDD Persistence 持久化

Spark RDD 持久化机制可以用于将需要重复运算的 RDD 存储在内存中，以便大幅提升运算效率。

Spark RDD 持久化使用方法如下：- RDD.persist(存储的等级)——可以指定存储等级，默认是 MEMORY_ONLY，也就是存储在内存中 - RDD.unpersist()——取消持久化

创建 intRddMemory

[17]:
```
intRddMemory = sc.parallelize([3,1,2,5,5])
```

使用 RDD.persist() 将 intRddMemory 进行持久化

[18]:
```
intRddMemory.persist()
```

[18]:
```
ParallelCollectionRDD[8] at parallelize at PythonRDD.scala:195
```

查看是否已经缓存

[19]:
```
intRddMemory.is_cached
```

[19]: True

取消持久化

[20]:
```
intRddMemory.unpersist()
```

[20]: ParallelCollectionRDD[8] at parallelize at PythonRDD.scala:195