

PySpark Issue 16391

Scala and Python generate inconsistent results

Grupo

Nome (RA):

Lucas Alves Racoci (156331)

Luiz Fernando Rodrigues da Fonseca (156475)

Rafael Zane (157079)

Rodrigo Noronha Máximo (157209)

Informações Gerais do JIRA

Type: Bug 

Status: OPEN

Priority: Major 

Resolution: Unresolved

Affects Version/s: 1.4.1, 1.5.2, 1.6.0

Fix Version/s: None

Component/s: PySpark

Labels: None

People

Assignee: Unassigned

Reporter: Shixiong Zhu

Votes: 0 Vote for this issue

Watchers: 8 Start watching this issue

Dates

Created: 04/Mar/16 23:11

Updated: 16/Mar/16 17:46

Resultados Diferentes

Python:

```
>>> i = 0

>>> rdd = sc.parallelize(range(1, 11)).map(lambda x: x+i)
>>> rdd.collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> i += 1
>>> rdd.collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Scala:

```
scala> var i = 0
i: Int = 0
scala> val rdd = sc.parallelize(1 to 10).map(_ + i)
scala> rdd.collect()
res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
scala> i += 1
scala> rdd.collect()
res2: Array[Int] = Array(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
```

Mais Testes Realizados

Python:

```
>>> def func():
...     i = 0
...     rdd = sc.parallelize(range(1, 11)).map(lambda x: x+i)
...     print rdd.collect()
...     i += 1
...     print rdd.collect()
...
>>> func()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Scala:


```
scala> def func() : Unit = {
  | var i = 0
  | val rdd = sc.parallelize(1 to 10).map(_ + i)
  | print(rdd.collect().mkString(", "))
  | print("\n")
  | i += 1
  | print(rdd.collect().mkString(", "))
  | }
func: ()Unit

scala> func()
1,2,3,4,5,6,7,8,9,10
2,3,4,5,6,7,8,9,10,11
```

Mais Testes Realizados

Python:


```
>>> def func_0():
...     i = 0
...     rdd = sc.parallelize(range(1, 11)).map(lambda x: x+i)
...     a = rdd.collect()
...     i += 1
...     b = rdd.collect()
...     print a
...     print b
...
>>> func_0()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```



Scala:

```
scala> def func_0 : Unit = {
  | var i = 0
  | val rdd = sc.parallelize(1 to 10).map(_ + i)
  | val a = rdd.collect().mkString(",")
  | i += 1
  | val b = rdd.collect().mkString(",")
  | println(a)
  | println(b)
  | }
func_0: ()Unit

scala> func_0()
1,2,3,4,5,6,7,8,9,10
2,3,4,5,6,7,8,9,10,11
```



Mais Testes Realizados

Python:

```
>>> def func__():
...     i = 0
...     rdd = sc.parallelize(range(1, 11)).map(lambda x: x+i)
...     a = rdd.collect()
...     i += 1
...     b = rdd.collect()
...     print b
...     print a
...
>>> func__()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Scala:

```
scala> def func__(): Unit = {
|   var i = 0
|   val rdd = sc.parallelize(1 to 10).map(_ + i)
|   val a = rdd.collect().mkString(",")
|   i += 1
|   val b = rdd.collect().mkString(",")
|   println(b)
|   println(a)
| }
func__: ()Unit

scala> func__()
2,3,4,5,6,7,8,9,10,11
1,2,3,4,5,6,7,8,9,10
```

Mais Testes Realizados

Python:

```
>>> i = 0
>>> rdd = sc.parallelize(range(1, 11)).map(lambda x: x+i)
>>> i += 1
>>> rdd.collect()
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Python:

```
>>> i = 0
>>> rdd = sc.parallelize(range(1, 11)).map(lambda x: x+i)
>>> rdd.collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> i += 1
>>> rdd.collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```


Issues Relacionadas

SPARK-10086: Flaky StreamingKMeans test in PySpark

- Issue relacionada a biblioteca de machine learning do PySpark.
- Problema na utilização de um algoritmo de clusterização.
- Após meses explorando o problema, os usuários descobriram que a raiz do problema é o comportamento presente na issue 16391 (a issue estudada).

SPARK-10784: Flaky Streaming ML test umbrella

- Issue umbrella criada para que fossem relatados todos os problemas de inconsistência de testes com Streaming ML em Python ou Scala.

Funcionamento do Spark

Método de lazy evaluation

- As transformações em um RDD não são mandadas imediatamente para os workers e são utilizadas na criação de um DAG.
- **DAG: Directed Acyclic Graph**
 - **Vértices:** Representam os RDD's
 - **Arestas:** Representam as operações a serem aplicadas em um RDD
 - Arestas são direcionadas de um RDD mais antigo a um RDD posterior
- Quando o driver executa uma ação, o DAG é submetido ao **DAG Scheduler**, que o divide em **tasks** que são passadas para o **Task Scheduler**.

Funcionamento do Spark

Método de lazy evaluation

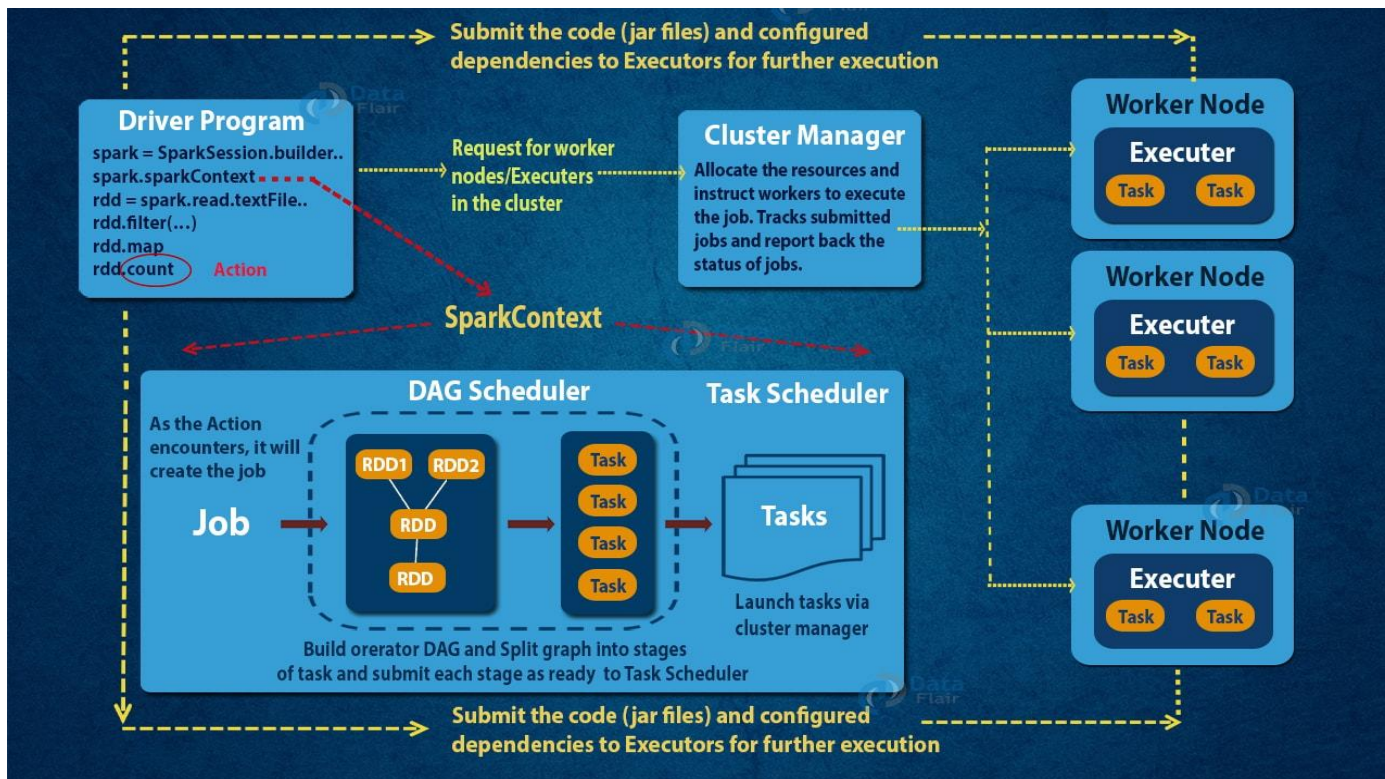
- O **Task Scheduler** lança as tasks para os workers através do **Cluster Manager**
 - As variáveis necessárias são copiadas para os **workers** e eles **não têm conhecimento das mudanças** sobre elas no driver.
 - Cada worker executa em uma parte do RDD
-
- **Portanto**, as transformações somente são computadas quando uma ação requer um resultado

Funcionamento do Spark

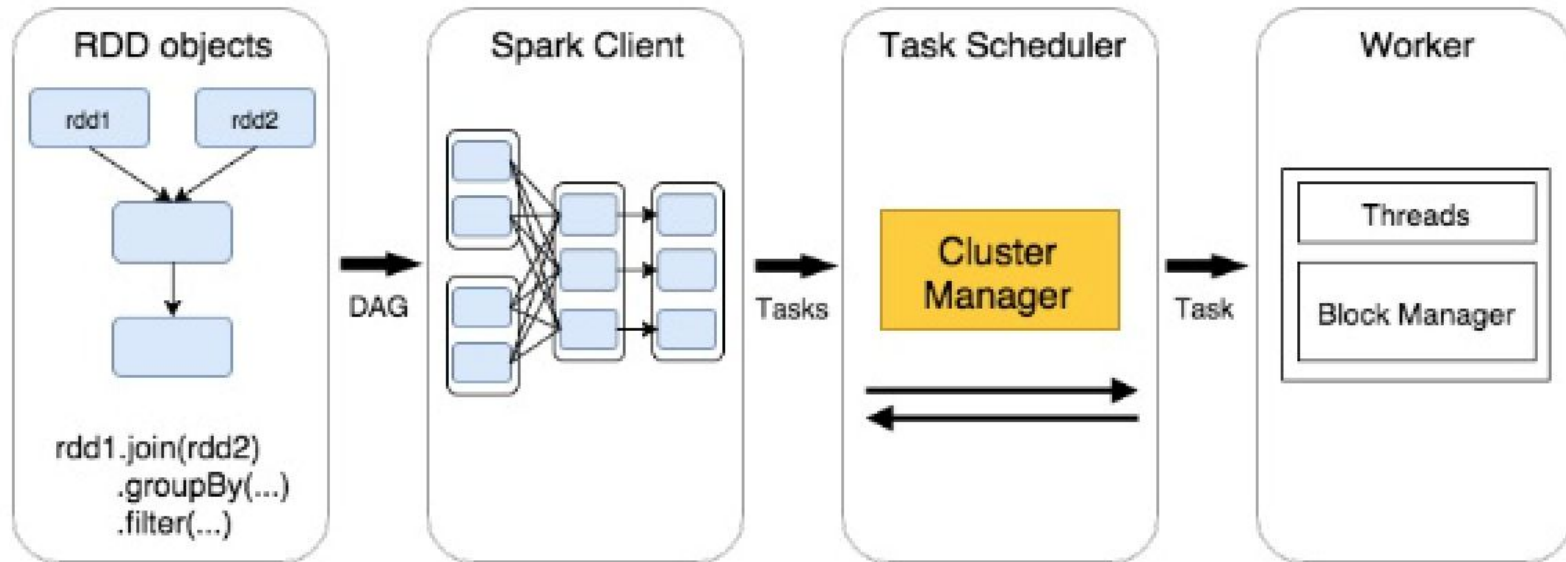
Método de lazy evaluation

- Esse método é **mais eficiente** que o Map Reduce do Hadoop
 - Dado lido do HDFS
 - Operações de MapReduce aplicadas
 - Resultado computado escrito de volta no HDFS
 - **Desperdício de memória** (maioria dos casos)
 - Na maioria das vezes não é necessário armazenar um resultado intermediário

Esquema do funcionamento do Spark



Esquema do funcionamento do Spark



Análise

Possíveis justificativas para o comportamento em **Python**:

- As transformações são adicionadas ao **DAG** no momento em que a ação é chamada.
- O valor mais atual da **variável é copiado** para o worker.
- Quando uma nova ação acontecer, o worker não recebe uma nova cópia da variável, assim o incremento da variável i posterior a criação da transformação não seria levado em consideração.

Possíveis justificativas para o comportamento em **Scala**:

- As transformações são adicionadas ao **DAG** no momento em que a ação é chamada.
- O valor mais atual da **variável é copiado** para o worker.
- Quando uma nova ação acontecer, após o incremento da variável i, o novo valor será repassado para o worker e portanto o valor mais atual será utilizado.

Work Around

Python

```
>>> i = 0

>>> rdd = sc.parallelize(range(1, 11)).map(lambda x: x+ i )
>>> rdd.collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> i += 1
>>> rdd.collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> rdd = rdd.map(lambda x: x)
>>> rdd.collect()
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Scala

```
scala> var i = 0
i: Int = 0
scala> val rdd = sc.parallelize(1 to 10).map(_ + i)
scala> rdd.collect()
res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
scala> i += 1
scala> rdd.collect()
res2: Array[Int] = Array(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
scala> val rdd1 = rdd.map(x => x)
scala> rdd1.collect()
res3: Array[Int] = Array(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
```


Conclusão

- Concluimos que o **bug** ocorre em **Python**
 - Toda **transformação** em um RDD é recomputada cada vez que uma **ação** é executada sobre ele
 - Quando uma nova ação acontece, o worker não recebe uma nova cópia da variável, assim o incremento da variável i posterior a criação da transformação não é levado em consideração, **o que está incorreto** de acordo com a afirmação acima contida na documentação do Spark