

Tema_103_RDD

November 24, 2022

Creado por:

Isabel Maniega

0.1 Datos Distribuidos Resistentes (RDD)

```
[2]: from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("RDD") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

```
[4]: # Creamos el contexto de Spark para trabajar con los RDDs

sc = spark.sparkContext
```

Coger una muestra de 5 datos entre 100 datos

```
[6]: rdd = sc.parallelize(range(1000))
rdd
```

```
[6]: PythonRDD[1] at RDD at PythonRDD.scala:53
```

```
[7]: rdd.takeSample(False, 5)
```

```
[7]: [523, 750, 303, 914, 610]
```

Numero de particiones:

```
[8]: print("Initial partition count:" + str(rdd.getNumPartitions()))
```

```
Initial partition count:10
```

Si queremos trabajar con un número distinto de particiones, debemos asignarle el número para ello:

```
[9]: reparRDD = rdd.repartition(4)
print("Initial partition count:" + str(reparRDD.getNumPartitions()))
```

```
Initial partition count:4
```

```
[10]: # Mostrar la información en el RDD:
```

```
rdd.collect()
```

```
[10]: [0,  
      1,  
      2,  
      3,  
      4,  
      5,  
      6,  
      7,  
      8,  
      9,  
      10,  
      11,  
      12,  
      13,  
      14,  
      15,  
      16,  
      17,  
      18,  
      19,  
      20,  
      21,  
      22,  
      23,  
      24,  
      25,  
      26,  
      27,  
      28,  
      29,  
      30,  
      31,  
      32,  
      33,  
      34,  
      35,  
      36,  
      37,  
      38,  
      39,  
      40,  
      41,  
      42,
```

43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89,

90,
91,
92,
93,
94,
95,
96,
97,
98,
99,
100,
101,
102,
103,
104,
105,
106,
107,
108,
109,
110,
111,
112,
113,
114,
115,
116,
117,
118,
119,
120,
121,
122,
123,
124,
125,
126,
127,
128,
129,
130,
131,
132,
133,
134,
135,
136,

137,
138,
139,
140,
141,
142,
143,
144,
145,
146,
147,
148,
149,
150,
151,
152,
153,
154,
155,
156,
157,
158,
159,
160,
161,
162,
163,
164,
165,
166,
167,
168,
169,
170,
171,
172,
173,
174,
175,
176,
177,
178,
179,
180,
181,
182,
183,

184,
185,
186,
187,
188,
189,
190,
191,
192,
193,
194,
195,
196,
197,
198,
199,
200,
201,
202,
203,
204,
205,
206,
207,
208,
209,
210,
211,
212,
213,
214,
215,
216,
217,
218,
219,
220,
221,
222,
223,
224,
225,
226,
227,
228,
229,
230,

231,
232,
233,
234,
235,
236,
237,
238,
239,
240,
241,
242,
243,
244,
245,
246,
247,
248,
249,
250,
251,
252,
253,
254,
255,
256,
257,
258,
259,
260,
261,
262,
263,
264,
265,
266,
267,
268,
269,
270,
271,
272,
273,
274,
275,
276,
277,

278,
279,
280,
281,
282,
283,
284,
285,
286,
287,
288,
289,
290,
291,
292,
293,
294,
295,
296,
297,
298,
299,
300,
301,
302,
303,
304,
305,
306,
307,
308,
309,
310,
311,
312,
313,
314,
315,
316,
317,
318,
319,
320,
321,
322,
323,
324,

325,
326,
327,
328,
329,
330,
331,
332,
333,
334,
335,
336,
337,
338,
339,
340,
341,
342,
343,
344,
345,
346,
347,
348,
349,
350,
351,
352,
353,
354,
355,
356,
357,
358,
359,
360,
361,
362,
363,
364,
365,
366,
367,
368,
369,
370,
371,

372,
373,
374,
375,
376,
377,
378,
379,
380,
381,
382,
383,
384,
385,
386,
387,
388,
389,
390,
391,
392,
393,
394,
395,
396,
397,
398,
399,
400,
401,
402,
403,
404,
405,
406,
407,
408,
409,
410,
411,
412,
413,
414,
415,
416,
417,
418,

419,
420,
421,
422,
423,
424,
425,
426,
427,
428,
429,
430,
431,
432,
433,
434,
435,
436,
437,
438,
439,
440,
441,
442,
443,
444,
445,
446,
447,
448,
449,
450,
451,
452,
453,
454,
455,
456,
457,
458,
459,
460,
461,
462,
463,
464,
465,

466,
467,
468,
469,
470,
471,
472,
473,
474,
475,
476,
477,
478,
479,
480,
481,
482,
483,
484,
485,
486,
487,
488,
489,
490,
491,
492,
493,
494,
495,
496,
497,
498,
499,
500,
501,
502,
503,
504,
505,
506,
507,
508,
509,
510,
511,
512,

513,
514,
515,
516,
517,
518,
519,
520,
521,
522,
523,
524,
525,
526,
527,
528,
529,
530,
531,
532,
533,
534,
535,
536,
537,
538,
539,
540,
541,
542,
543,
544,
545,
546,
547,
548,
549,
550,
551,
552,
553,
554,
555,
556,
557,
558,
559,

560,
561,
562,
563,
564,
565,
566,
567,
568,
569,
570,
571,
572,
573,
574,
575,
576,
577,
578,
579,
580,
581,
582,
583,
584,
585,
586,
587,
588,
589,
590,
591,
592,
593,
594,
595,
596,
597,
598,
599,
600,
601,
602,
603,
604,
605,
606,

607,
608,
609,
610,
611,
612,
613,
614,
615,
616,
617,
618,
619,
620,
621,
622,
623,
624,
625,
626,
627,
628,
629,
630,
631,
632,
633,
634,
635,
636,
637,
638,
639,
640,
641,
642,
643,
644,
645,
646,
647,
648,
649,
650,
651,
652,
653,

654,
655,
656,
657,
658,
659,
660,
661,
662,
663,
664,
665,
666,
667,
668,
669,
670,
671,
672,
673,
674,
675,
676,
677,
678,
679,
680,
681,
682,
683,
684,
685,
686,
687,
688,
689,
690,
691,
692,
693,
694,
695,
696,
697,
698,
699,
700,

701,
702,
703,
704,
705,
706,
707,
708,
709,
710,
711,
712,
713,
714,
715,
716,
717,
718,
719,
720,
721,
722,
723,
724,
725,
726,
727,
728,
729,
730,
731,
732,
733,
734,
735,
736,
737,
738,
739,
740,
741,
742,
743,
744,
745,
746,
747,

748,
749,
750,
751,
752,
753,
754,
755,
756,
757,
758,
759,
760,
761,
762,
763,
764,
765,
766,
767,
768,
769,
770,
771,
772,
773,
774,
775,
776,
777,
778,
779,
780,
781,
782,
783,
784,
785,
786,
787,
788,
789,
790,
791,
792,
793,
794,

795,
796,
797,
798,
799,
800,
801,
802,
803,
804,
805,
806,
807,
808,
809,
810,
811,
812,
813,
814,
815,
816,
817,
818,
819,
820,
821,
822,
823,
824,
825,
826,
827,
828,
829,
830,
831,
832,
833,
834,
835,
836,
837,
838,
839,
840,
841,

842,
843,
844,
845,
846,
847,
848,
849,
850,
851,
852,
853,
854,
855,
856,
857,
858,
859,
860,
861,
862,
863,
864,
865,
866,
867,
868,
869,
870,
871,
872,
873,
874,
875,
876,
877,
878,
879,
880,
881,
882,
883,
884,
885,
886,
887,
888,

889,
890,
891,
892,
893,
894,
895,
896,
897,
898,
899,
900,
901,
902,
903,
904,
905,
906,
907,
908,
909,
910,
911,
912,
913,
914,
915,
916,
917,
918,
919,
920,
921,
922,
923,
924,
925,
926,
927,
928,
929,
930,
931,
932,
933,
934,
935,

936,
937,
938,
939,
940,
941,
942,
943,
944,
945,
946,
947,
948,
949,
950,
951,
952,
953,
954,
955,
956,
957,
958,
959,
960,
961,
962,
963,
964,
965,
966,
967,
968,
969,
970,
971,
972,
973,
974,
975,
976,
977,
978,
979,
980,
981,
982,

```
983,  
984,  
985,  
986,  
987,  
988,  
989,  
990,  
991,  
992,  
993,  
994,  
995,  
996,  
997,  
998,  
999]
```

```
[11]: # Longitud del RDD:
```

```
rdd.count()
```

```
[11]: 1000
```

```
[32]: # declaramos la función con la cuál vamos a trabajar:
```

```
def function(x):  
    print(x)
```

```
fore = rdd.foreach(function)
```

```
[26]: # Filtrar la información
```

```
rdd_filter = rdd.filter(lambda x: x<=30)  
filtered = rdd_filter.collect()  
filtered
```

```
[26]: [0,  
1,  
2,  
3,  
4,  
5,  
6,  
7,  
8,  
9,
```

```
10,  
11,  
12,  
13,  
14,  
15,  
16,  
17,  
18,  
19,  
20,  
21,  
22,  
23,  
24,  
25,  
26,  
27,  
28,  
29,  
30]
```

```
[27]: # Primer valor del RDD  
  
firstRec = rdd.first()  
print("First record:" + str(firstRec))
```

First record:0

```
[29]: # Obtener el valor Máximo  
  
datMax = rdd.max()  
print("Max record:" + str(datMax))
```

Max record:999

```
[30]: # Mostrar los tres primeros valores  
  
data3 = rdd.take(3)  
data3
```

```
[30]: [0, 1, 2]
```


0.2 Caché y Memoria

```
[35]: # Guarda los datos automáticamente en el caché que se encuentra en:
```

```
cacheRDD = rdd.cache()  
cacheRDD
```

```
[35]: PythonRDD[1] at RDD at PythonRDD.scala:53
```

```
[37]: import pyspark
```

```
[38]: dfPersist = rdd.persist(pyspark.StorageLevel.MEMORY_ONLY)  
dfPersist
```

```
[38]: PythonRDD[1] at RDD at PythonRDD.scala:53
```

Hay varios tipos de almacenamiento que se añadirá a `pyspark.StorageLevel` son:

- **MEMORY_ONLY:** esto es por defecto el comportamiento de RDD lo guarda en `cache()` y almacena el RDD como objetos deserializados en la memoria JVM. Cuando no hay suficiente memoria disponible, no se guardará en RDD de algunas particiones y estas se volverán a calcular cuando sea necesario. Esto requiere más almacenamiento, pero se ejecuta más rápido, ya que se necesitan pocos ciclos de CPU para leer de la memoria.
- **MEMORY_ONLY_SER:** Es lo mismo que `MEMORY_ONLY`, pero la diferencia es que almacena RDD como objetos serializados en la memoria JVM. Se necesita menos memoria (uso eficiente del espacio) que `MEMORY_ONLY`, ya que guarda los objetos como serializados y requiere algunos ciclos adicionales de CPU para deserializar.
- **MEMORY_ONLY_2:** Igual que el nivel de almacenamiento `MEMORY_ONLY` pero replica cada partición en dos nodos de clúster.
- **MEMORY_ONLY_SER_2:** Igual que el nivel de almacenamiento `MEMORY_ONLY_SER` pero replica cada partición en dos nodos de clúster.
- **MEMORY_AND_DISK:** En este nivel de almacenamiento, el RDD se almacenará en la memoria JVM como objetos deserializados. Cuando el almacenamiento requerido es mayor que la memoria disponible, almacena algunas de las particiones sobrantes en el disco y lee los datos del disco cuando es necesario. Es más lento porque hay E / S involucradas.
- **MEMORY_AND_DISK_SER:** Esto es lo mismo que la diferencia de nivel de almacenamiento `MEMORY_AND_DISK` ya que serializa los objetos RDD en la memoria y en el disco cuando no hay espacio disponible.
- **MEMORY_AND_DISK_2:** Igual que el nivel de almacenamiento `MEMORY_AND_DISK` pero replica cada partición en dos nodos de clúster.
- **MEMORY_AND_DISK_SER_2:** Igual que el nivel de almacenamiento `MEMORY_AND_DISK_SER` pero replica cada partición en dos nodos de clúster.
- **DISK_ONLY:** En este nivel de almacenamiento, RDD se almacena solo en el disco y el tiempo de cálculo de la CPU es alto en función de las E / S involucradas.

- **DISK_ONLY_2:** Igual que el nivel de almacenamiento DISK_ONLY pero replica cada partición en dos nodos de clúster.

```
[40]: # Podemos dejar de guardar en memoria esa información:
```

```
rddPersist2 = dfPersist.unpersist()
rddPersist2.is_cached
```

```
[40]: False
```

1 Broadcast

Son variables de sólo lectura que se almacenan en el caché y están disponibles en todos los nodos del clúster. Utiliza algoritmos de transmisión eficientes para reducir los costes de comunicación. Útil para la buqueda de datos.

```
[41]: import pyspark
      from pyspark.sql import SparkSession
```

```
[42]: spark = SparkSession.builder.appName("SparkByExample.com") \
      .getOrCreate()
```

```
[43]: # Creamos el Broadcast con la información relativa a los estados:
      states = {"ED1": "Edicion 1", "ED2": "Edicion 2", "ED3": "Edicion 3"}

      broadcastStates = spark.sparkContext.broadcast(states)
```

```
[44]: data = [
      ("James", "Smith", "SPAIN", "ED2"),
      ("Michel", "Rose", "BOLIVIA", "ED3"),
      ("Robert", "Williams", "ARGENTINA", "ED2"),
      ("Maria", "Jones", "SPAIN", "ED1"),
      ]
```

```
[45]: columns = ["firstname", "lastname", "country", "edition"]
```

```
[47]: df = spark.createDataFrame(data=data, schema=columns)
      df.printSchema()
```

```
root
 |-- firstname: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- country: string (nullable = true)
 |-- edition: string (nullable = true)
```

```
[48]: df.show(truncate=False)
```

firstname	lastname	country	edition
James	Smith	SPAIN	ED2
Michel	Rose	BOLIVIA	ED3
Robert	Williams	ARGENTINA	ED2
Maria	Jones	SPAIN	ED1

```
[49]: def state_convert(code):
      return broadcastStates.value[code]

result = df.rdd.map(lambda x: (x[0], x[1], x[2], state_convert(x[3]))).
    ↪toDF(columns)
result.show(truncate=False)
```

firstname	lastname	country	edition
James	Smith	SPAIN	Edicion 2
Michel	Rose	BOLIVIA	Edicion 3
Robert	Williams	ARGENTINA	Edicion 2
Maria	Jones	SPAIN	Edicion 1

Creado por:

Isabel Maniega