RDD 并行度&分区

最小计算单元RDD 内部有个分区的概念 为了更好的 并行计算

那么到底多少个分区呢？？？？

1 代码一

如果在函数makeRDD 不写第二个分区参数 那么会有多少分区呢 why

object RDD\_Memory\_Par {  
 def main(args: Array[String]): Unit = {  
 val sparkConf = new SparkConf().setMaster("local[\*]").setAppName("RDD\_Memory\_Par")  
 val sparkContext = new SparkContext(sparkConf)  
  
 val rdd = sparkContext.makeRDD(List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),3)  
  
 rdd.saveAsTextFile("output")  
  
 sparkContext.stop()  
  
 }  
  
}

makeRDD(List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),3)

第二个参数 可以不用传递

*/\*\* Distribute a local Scala collection to form an RDD.  
 \* This method is identical to `parallelize`.  
 \** @param seq *Scala collection to distribute  
 \** @param numSlices *number of partitions to divide the collection into  
 \** @return *RDD representing distributed collection  
 \*/*def makeRDD[T: ClassTag](  
 seq: Seq[T],  
 numSlices: Int = defaultParallelism): RDD[T] = withScope {  
 parallelize(seq, numSlices)  
}

defaultParallelism 是多少呢？？？

*/\*\* Default level of parallelism to use when not given by user (e.g. parallelize and makeRDD). \*/*def defaultParallelism: Int = {  
 assertNotStopped()  
 taskScheduler.defaultParallelism  
}

*// Get the default level of parallelism to use in the cluster, as a hint for sizing jobs.*def defaultParallelism(): Int

在 private[spark] trait TaskScheduler 中

需要找到实现类 private[spark] class TaskSchedulerImpl

的

override def defaultParallelism(): Int = backend.defaultParallelism()

到了 private[spark] trait SchedulerBackend

中的

def defaultParallelism(): Int

到实现类private[spark] class LocalSchedulerBackend的

override def defaultParallelism(): Int =  
 scheduler.conf.getInt("spark.default.parallelism", totalCores)

点击这个conf

val conf = sc.conf

到

private[spark] def conf: SparkConf = \_conf

其实就是我们代码自己写的

val sparkConf = new SparkConf().setMaster("local[\*]").setAppName("RDD\_Memory\_Par")

嘿嘿 我们可以自己配置下这个参数spark.default.parallelism

2 代码二

分区数设置好了 那么数据跑到哪个分区 什么数据去哪个分区 是怎么决定的 怎么分配数据的

即

如果数据很少 分区很多那么怎么分配呢？

Note:是从集合中读取数据

object RDD\_Memory\_Data\_To\_Pars extends App with Serializable {  
  
 val sparkConf: SparkConf = new SparkConf().setMaster("local[\*]").setAppName("RDD\_Memory\_Data\_To\_Pars")  
  
  
 val sparkContext = new SparkContext(sparkConf)  
  
 val rdd: RDD[Int] = sparkContext.makeRDD(List(1, 2, 3, 4),2)  
  
 rdd.saveAsTextFile("output")  
  
 sparkContext.stop()  
  
  
}

2个分区

是怎么存呢？

[1,2] [3,4]

Or

[1,3] [2,4]

3个分区呢

val rdd: RDD[Int] = sparkContext.makeRDD(List(1, 2, 3, 4,5),3)

是

0-1

1-2 3

2-4 5

源码

def parallelize[T: ClassTag](  
 seq: Seq[T],  
 numSlices: Int = defaultParallelism): RDD[T] = withScope {  
 assertNotStopped()  
 new ParallelCollectionRDD[T](this, seq, numSlices, Map[Int, Seq[String]]())  
}

override def getPartitions: Array[Partition] = {  
 val slices = ParallelCollectionRDD.slice(data, numSlices).toArray  
 slices.indices.map(i => new ParallelCollectionPartition(id, i, slices(i))).toArray  
}

case \_ =>  
 val array = seq.toArray *// To prevent O(n^2) operations for List etc* positions(array.length, numSlices).map { case (start, end) =>  
 array.slice(start, end).toSeq  
 }.toSeq

array.length=5

numSlices=3

*// Sequences need to be sliced at the same set of index positions for operations  
// like RDD.zip() to behave as expected*def positions(length: Long, numSlices: Int): Iterator[(Int, Int)] = {  
 (0 until numSlices).iterator.map { i =>  
 val start = ((i \* length) / numSlices).toInt  
 val end = (((i + 1) \* length) / numSlices).toInt  
 (start, end)  
 }  
}

(0 until numSlices).iterator==0,1,2==i

i =>  
 val start = ((i \* length) / numSlices).toInt  
 val end = (((i + 1) \* length) / numSlices).toInt  
 (start, end)

0=> start=0

end=5/3=1

(start, end)==(0,1)[tuple]

0=>(0,1)

1=>(1,3)

2=>(3,5)

然后到

positions(array.length, numSlices).map { case (start, end) =>  
 array.slice(start, end).toSeq  
 }

即切分数组 点击slice

到

override def slice(from : scala.Int, until : scala.Int) : scala.Array[T] = { */\* compiled code \*/* } //util是不包含

所以

[1,2,3,4,5]

0=>(0,1)=>1

1=>(1,3)=>2 3

2=>(3,5)=>3 4

3 代码三

从文件读取数据

数据按 Hadoop 文件读取的规则进行切片分区，而切片规则和数据读取的规则有些差异。

到底会有多少个分区？？

object RDD\_File\_Par extends App {  
 val sparkConf: SparkConf = new SparkConf().setMaster("local[\*]").setAppName("RDD\_File\_Par")  
  
 val sparkContext = new SparkContext(sparkConf)  
  
 *// 3.创建RDD* val rdd = sparkContext.textFile("data/wc/wc01.txt")  
  
 *// 4.保存文件* rdd.saveAsTextFile("output")  
  
 *// 5.关闭连接* sparkContext.stop()  
}

*/\*\*  
 \* Read a text file from HDFS, a local file system (available on all nodes), or any  
 \* Hadoop-supported file system URI, and return it as an RDD of Strings.  
 \* The text files must be encoded as UTF-8.  
 \*  
 \** @param path *path to the text file on a supported file system  
 \** @param minPartitions *suggested minimum number of partitions for the resulting RDD  
 \** @return *RDD of lines of the text file  
 \*/*def textFile(  
 path: String,  
 minPartitions: Int = defaultMinPartitions): RDD[String] = withScope {  
 assertNotStopped()  
 hadoopFile(path, classOf[TextInputFormat], classOf[LongWritable], classOf[Text],  
 minPartitions).map(pair => pair.\_2.toString).setName(path)  
}

*/\*\*  
 \* Default min number of partitions for Hadoop RDDs when not given by user  
 \* Notice that we use math.min so the "defaultMinPartitions" cannot be higher than 2.  
 \* The reasons for this are discussed in https://github.com/mesos/spark/pull/718  
 \*/*def defaultMinPartitions: Int = math.min(defaultParallelism, 2)

应该是math.min(8, 2)==2

是对的

业务代码改一下

val rdd = sparkContext.textFile("data/wc/wc01.txt",3)

确实产生3个分区

我们把data/wc/wc01.txt数据改一下

1  
2  
3

代码改为

val rdd = sparkContext.textFile("data/wc/wc01.txt")

但是产生了3个分区

A screenshot of a computer

Description automatically generated

代码是val rdd = sparkContext.textFile("data/wc/wc01.txt",2)

也是3个分区

也就是

真正的分区可能比最小分区这个数[2]大

那么到底会有多少个分区？？

怎么算的呢？

数据按 Hadoop 文件读取的规则进行切片分区

1）public abstract class FileInputFormat<K, V> implements InputFormat<K, V>

2）public InputSplit[] getSplits(JobConf job, int numSplits)

3）拿到totalSize

所有读取的文件 统计字节数总和

这个文件有换行 其实不是3个字节 是5个字节

object FileByteCounter {  
 def getFileSize(filePath: String): Option[Long] = {  
 val file = new File(filePath)  
 if (file.exists()) {  
 Some(file.length())  
 } else {  
 None  
 }  
 }  
  
 def main(args: Array[String]): Unit = {  
 val filePath = "data/wc/wc01.txt"  
 val fileSize = getFileSize(filePath)  
 fileSize match {  
 case Some(size) => println(s"File $filePath size is $size")  
 case None => println(s"File $filePath not found")  
 }  
 }  
}

--5

long goalSize = totalSize / (numSplits == 0 ? 1 : numSplits);

long goalSize = 5 / (2 == 0 ? 1 : 2);--2 每个分区2个字节

5/2=2…1

会不会产生新的分区呢？

Hadoop有个1.1的概念 剩余的这个字节数占每个分区的字节数的百分之多少 现在50%

大于10%产生新分区

最后会有3个分区

那么分区怎么存数据的呢？

目前是

* 1. 2

1-3

2-null

1)数据按行读取和Hadoop一样和字节数无关系 偏移量不会被重复读

2)数据读取时以偏移量为单位

1@=>01

2@=>23

3=4

3)那么每个分区读多少个字节 即数据分区的偏移量范围如何计算

已经得到 goalSize=2

0=>[0,2]  
1=>[2,4]

2=>[4,6]

4)最后读到 一行行读

0=>[0,2]=>1@2@  
1=>[2,4]=>3

2=>[4,6]=>无