Lab1:Batch job in cloud

Part A

- **Hadoop wordcount, Spark connected component试验结果如下图所示：**

```
hadoop@master:~$ hadoop fs -cat /output/part-r-00000
aabb    1
abc     1
count   1
hadoop  2
hdfs    1
hello   3
mapreduce       1
test    1
word    1
world   1
```

```
2023-12-03 14:11:05,231 INFO scheduler
main.scala:32, took 0.112008 s
(justinbieber,1)
(matei_zaharia,3)
(ladygaga,1)
(BarackObama,1)
(jeresig,3)
(odersky,3)
```

- **华为云ECS应该是一种 IaaS**，因为能够在root中部署底层计算资源，搭建应用环境，且拥有管理权限；此外，我们使用的ECS具有弹性，用户可以根据负载的变化自动扩展或缩减计算资源，这是 IaaS 的一个特性。而 PaaS 则更抽象，其提供的是一个完整的应用程序开发和部署平台，用户不需要关心底层的基础设施。

- **Hadoop 和 Spark 两种框架的区别有：**
  - **处理模型**：Hadoop: 基于MapReduce处理模型。而Spark: 提供了更丰富的处理模型，使其更适合迭代式算法和复杂的分析任务。
  - **性能**：Hadoop: 由于采用磁盘存储中间数据，MapReduce的性能可能受到磁盘IO的限制，尤其在迭代算法中可能效率较低。Spark使用内存存储中间数据，因此在迭代和复杂计算方面通常比MapReduce更高效。
  - **数据处理方式**：Hadoop: 主要处理静态数据，适用于离线数据分析和批处理。而Spark: 提供了更多实时和交互式处理能力，支持流处理和内存中迭代。
  - **编程接口**：Hadoop: 使用Java编程语言。而Spark: 提供了更多的编程接口，包括Java、Scala、Python和R。
  - **易用性**：Hadoop: 相对较复杂，需要开发者编写更多的代码来表达数据处理逻辑。而Spark: 提供更简洁的API，更容易使用。

附：配置 hadoop 以及 spark 后 Jps

```
hadoop@master:/usr/local/hadoop$ jps
10752 NameNode
11046 SecondaryNameNode
11613 Jps
11293 ResourceManager
```

```
hadoop@slave01:~$ jps
26003 DataNode
26180 NodeManager
26309 Jps
```

```
hadoop@master:/usr/local/spark$ jps
4146 SecondaryNameNode
4395 ResourceManager
4829 Jps
3838 NameNode
4766 Master
```

```
hadoop@slave01:~$ jps
31152 NodeManager
2386 Worker
30972 DataNode
2573 Jps
```

Part B

- **给出20名候选人名单：**

```
(4037,10.159729097543362)    (2654,4.26078111629283)
(15,6.780033943145)          (6634,4.257679186579282)
(2470,6.7482254419278425)    (5254,4.220791083775596)
(2237,6.323071398182568)     (214,4.078102410735729)
(1186,5.590634085967654)     (2285,4.026841138580162)
(2625,5.330932366106915)     (2328,3.955737740463529)
(665,4.669622977312882)      (2398,3.939081751904102)
(6774,4.409225692409475)     (28,3.8631717276144193)
(8293,4.312848725457975)     (4875,3.8619825467701725)
(4191,4.30575420842798)      (7620,3.6021458088825864)
```

- **PageRank算法实现：**
  - 首先仿照 Graphx 示例中的 PageRank 进行runPageRank编写。函数参数包含输入图；容忍度，用于判断算法是否收敛的阈值；重置概率，表示在每次迭代中，以一定的概率重置节点的PageRank值；最大迭代次数，用于控制算法运行的最大迭代次数。

```scala
def runPageRank(graph: Graph[Int, Int], tol: Double, resetProb: Double, maxIter: Int): Graph[Double, Int] = {
    var ranks = graph
      .outerJoinVertices(graph.outDegrees) { (_, _, deg) => deg.getOrElse(0) }
      .mapTriplets(e => 1.0 / e.srcAttr, TripletFields.Src)
      .mapVertices { (_, _) => 1.0 }

    for (_ <- 1 to maxIter) {
      val prevRanks = ranks
      val updates = ranks.aggregateMessages[Double](ctx => ctx.sendToDst(ctx.srcAttr * ctx.attr), _ + _, TripletFields.Src)
      ranks = ranks.outerJoinVertices(updates) { (_, oldRank, msgSumOpt) =>
        resetProb + (1.0 - resetProb) * msgSumOpt.getOrElse(0.0)
      }

      // 计算误差并检查是否收敛
      val diff = prevRanks.vertices.join(ranks.vertices).map {
        case (_, (oldRank, newRank)) => math.abs(oldRank - newRank)
      }
      val totalDiff = diff.sum()
      if (totalDiff < tol) {
        println(s"Converged after ${_} iterations.")
        return ranks
      }
    }
```

  - 在该函数中，进行以下操作：
    - 初始化节点的PageRank值，将每个节点的PageRank初始化为1.0。
    - 进行迭代，每次迭代包括以下步骤：
      - 为每条边计算权重，在每个节点上聚合邻居节点的PageRank值乘以对应边的权重。
      - 根据公式更新每个节点的PageRank值，考虑重置概率。
      - 计算前后两次迭代之间每个节点PageRank值的差异，检查是否达到收敛条件。
      - 如果总的差异小于容忍度 tol，则认为算法已经收敛，输出结果并结束。

```scala
def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder.appName(s"${this.getClass.getSimpleName}").getOrCreate()
    val sc = spark.sparkContext
    // 从文件加载边列表创建图
    val link: Graph[Int, Int] = GraphLoader.edgeListFile(sc, "data/pagerank/Wiki-Vote.txt")
    // 运行 PageRank 算法
    val ranks = runPageRank(link, 0.0001, 0.85, 100)
    // 输出排名最高的 20 个节点
    println(ranks.vertices.collect().sortBy(-_._2).take(20).mkString("\n"))
    // 关闭 SparkSession
    spark.stop()
  }
```

- 在主函数中
  - 利用 Graph 进行数据读入，这里用到的数据。
  - 调用 runPageRank 函数，设定迭代100次。
  - 输出前20的候选人
  - 最后关闭Spark

- **算法部署及运行步骤：**
  - 先在 hadoop 用户中启动 hadoop 和 spark 集群
    ```
    > cd /usr/local/hadoop/
    > sbin/start-all.sh
    > cd /usr/local/spark/
    > sbin/start-master.sh
    ```
  - 将 wiki-Vote.txt 数据集发到 hadoop fs 中
    ```
    > hadoop fs -put wiki-Vote.txt /user/hadoop/data/pagerank/
    ```
  - 编写 simple.sbt 文件

```
name := "Simple Project"
version := "1.9.7"
scalaVersion := "2.12.15"
libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "3.2.4",
  "org.apache.spark" %% "spark-graphx" % "3.2.4",
  "org.apache.spark" %% "spark-sql" % "3.2.4"
)
```

  - 打包并发送到 Spark 中运行
    ```
    > /usr/local/sbt/sbt package
    > /usr/local/spark/bin/spark-submit --class "PageRank" /usr/local/spark/mycode/pagerank/target/scala-2.12/simple-project_2.12-1.9.7.jar
    ```
- 运行结果

```
2023-12-03 14:27:31,654 INFO storage.
ce0 on master:32957 in memory (size:
(4037,13.687824661001775)
(15,10.9328059520621)
(6634,10.656469713599291)
(2625,9.755679770957746)
(2398,7.750205920765446)
(2470,7.498077775768758)
(2237,7.417430386169762)
(4191,6.7377444603752865)
(7553,6.44622789746703)
(5254,6.38790776195012)
(2328,6.058602112229776)
(1186,6.0475330647209775)
(1297,5.78105575667395)
(4335,5.754084280496569)
(7620,5.740174509100523)
(5412,5.70106209414574)
(7632,5.667866508823103)
(4875,5.567064113522038)
(6946,5.372790269370961)
(3352,5.300100134345485)
```