PRÀCTICA DE SCALA

Jan Puig Martin
u1973798

Girona, Desembre de 2024

Abast de la solució.

Primera Part:

Les diverses funcions estan comentats de forma adient al codi, tot això es farà una breu explicació d'aquestes mateixes:

- **freq(x: String, n: Int = 1): List[(String, Int)]**: Aquesta funció rep un valor x, el qual és una cadena de strings i pot rebre opcionalment un valor n que de forma per defecte esta establert a 1, aquest segon valor és la quantitat de paraules que s'han d'agrupar per tal de formar, o no, n-grames. Aquesta funció també aplana el text deixant-lo només amb paraules minuscules i traient qualsevol caràcter que no formi part d'una paraula.
- **quitstop(x: String, y: List[String]): String**: Donat un String x s'hi treuen les paraules de la llista y, havent previament aplanat el text igual que a la funció freq.
- **nonstopfreq(x: String, y: List[String]): List[(String, Int)]**: Aquesta funció aplica les dues anteriors juntament
- **paraulafreqfreq(x: String): Unit**: Utilitzant la funció freq es mostren les 10 majors freqüències i les 10 menors.
- **showfreq(x: List[(String, Int)]): Unit**: Una funció simple per tal de mostrar per pantalla la sortida de les funcions **freq** i **nonstopfreq**.
- **cosinesim(x: String, y: String): Double**: Aquesta última funció retorna un valor entre 0 i 1 el qual descriu la similitud entre dos fitxers. Els fitxers que entren han de ser aplanats i sense *stop-words*.

La sortida de les diferents funcions sembla correcte, tot i això no s'ha assolit un resultat idèntic amb les funcions freq, nonstopfreq, i paraulafreqfreq, amb comparació amb les proves que el professorat ens ha donat.

Segona Part:

Per a aquesta segona part de la pràctica no s'ha assolit un resultat satisfactible. No s'han pogut assolir els punts 2 i 3.
Per a la segona part les modificacions de codi usades han estat:

- **MR**: Al parametritzar l'entrada de mappers i reducers s'ha modificat el codi per tal de crear el nombre entrat d'aquests, tot i això també s'ha modificat la creació de reducers per tal de crear el mínim entre el llistat de keys que es passen de mappers a reducers o els reducers entrats per paràmetre ja que altrament el sistema queda penjat.

```
116         reducersPendents = Math.min(nreducers, dict.size) // actualitzem els reducers pendents
117         val reducers = for (i <- 0 ≤ until < Math.min(nreducers, dict.size)) yield //es creen tants reducers com entrades al diccionari
118           context.actorOf(Props(new Reducer(reducing)), s"reducer$i")
```

- **timeMeasurement[A](function: => A): (A, Double)**: la funció per tal de mesurar el temps funciona correctament, esta definida de tal manera per tal d'executar la funció entrada per paràmetre dins del codi i així poder mesurar el temps que es triga en executar-la.
- **numPromigRefPagines(numMapers: Int = 0, numReducers: Int = 0): Double:** Aquest codi agafa el total de pàgines que es té, s'envia a un mapReduce el qual conta les referències totals que hi ha i el resultat de tornada, es divideix entre el nombre total de pàgines que es té per saber la mitjana de referències per pàgina.

MapReduce's

Degut al curt abast de la solució només s'ha creat un sol MapReduce. Aquest s'ha creat de la següent manera per al numPromigRefPagines:

```
def numPromigRefPagines(numMapers: Int = 0, numReducers: Int = 0): Double = {  Jan
  val actorSystem = ActorSystem("reducer-system")
  val files = getListOfFiles("viqui_files") // nombres de archivos
  val numfiles = files.length // cantidad de archivos
  val nFilesMapper = Math.ceil(numfiles.toDouble / numMapers).toInt // funcio mes complexa per asegurar una bona divisio


  val input = files.grouped(nFilesMapper).zipWithIndex.map {
    case (fileGroup, index) => (index, fileGroup)
  }.toList

  val mapping = (key: Int, fileList: List[java.io.File]) => {
    val totalRefs = fileList.map(file => ViquipediaParse.parseViquipediaFile(file.toString).refs.length).sum
    List(("total", totalRefs))
  }

  val reducing = (key: String, values: List[Int]) => {
    val totalReferences = values.sum
    (key, totalReferences)
  }

  val mapReduce = actorSystem.actorOf(Props(new MR(input, mapping, reducing, numMapers, numReducers)), "mapReduce")

  implicit val timeout: Timeout = Timeout(10000.seconds)
  val futureResult = mapReduce ? MapReduceCompute()
  Await.result(futureResult, timeout.duration)
  val resultat = futureResult.value.get.get.asInstanceOf[Map[String, Int]].get("total").get.toDouble / numfiles
  resultat
}
```

El funcionament d'aquest map reduce radica en agafar els fitxers disponibles en conjunts, sumar les referències de cada conjunt, i enviar a un reducer final les sumes d'aquests conjunts per tal de que ho sumi tot. El input pren per a key el index del grup i un grup de fitxers a analitzar. El mapper rep lo anterior i de cada fitxer conta les referències que hi han dins d'aquest i les suma amb la resta de referències, la clau de sortida esta hardcoded siguent aquesta "total" per tal d'enviar tots els resultats a un sol reducer. El reducer pren les sumes i fa una última suma total.

Actors

Sortida al executar el codi amb diferents actors per al número promig de referències per a les pàgines, per a diverses execucions:

NumMappers: 1, NumReducers: 1, Time: 2.44993975
NumMappers: 4, NumReducers: 1, Time: 0.294735625
NumMappers: 10, NumReducers: 1, Time: 0.166837083
NumMappers: 20, NumReducers: 1, Time: 0.175572875
NumMappers: 50, NumReducers: 1, Time: 0.13228525
NumMappers: 100, NumReducers: 1, Time: 0.144903167
NumMappers: 4693, NumReducers: 1, Time: 0.194770834

NumMappers: 1, NumReducers: 1, Time: 2.406563542
NumMappers: 4, NumReducers: 1, Time: 0.296701125
NumMappers: 10, NumReducers: 1, Time: 0.235465542
NumMappers: 20, NumReducers: 1, Time: 0.160882167
NumMappers: 50, NumReducers: 1, Time: 0.159689417
NumMappers: 100, NumReducers: 1, Time: 0.125552709
NumMappers: 4693, NumReducers: 1, Time: 0.204970083

NumMappers: 1, NumReducers: 1, Time: 2.278750916
NumMappers: 4, NumReducers: 1, Time: 0.2570915
NumMappers: 10, NumReducers: 1, Time: 0.167229542
NumMappers: 20, NumReducers: 1, Time: 0.157708584
NumMappers: 50, NumReducers: 1, Time: 0.12905775
NumMappers: 100, NumReducers: 1, Time: 0.129361459
NumMappers: 4693, NumReducers: 1, Time: 0.210836916

Sembla ser que al voltant dels 50 mappers s'assoleix la eficiencia màxima, ja que al augmentar aquest nombre sembla ser que augmenta el temps d'execució. Provar de crear tants mappers con fitxers existeixen no prova ser eficaç ja que tarda de mitja més que amb menys mappers que es divideixin la feina.
Per a totes les anteriors proves s'ha utilitzat la mateixa màquina siguent aquesta un Macbook Pro M2 de 16Gb de RAM.