CS-6240 Section 1 HOMEWORK 4 NAOMI JOSHI

Design Discussion

Description of Spark Program

For reading input from the file I use .textFile() and then .map() which takes each line from the input bz2 parser and applies the Parser function (java) to it and returns an rdd with parsed line. Then I filter the null values from the rdd using a .filter() and also persist() it to keep it in the memory. I also specify the minimum no. of partitions in textFile so distribute the processing to multiple nodes.
To calculate the initial pagerank I use .count() which counts the no. of input records in the RDD. 1/countOfNodes gives the initial pagerank.
The input format returned from the parser is a string of nodes:[adjc list]
Then I transform input RDD to another RDD using .map() to take each string line and applying .split(":") function to and convert it to RDD of array<string> and then one more .map() to pick values based on the index of the array and convert it to RDD of structure (node=string, (adjList=string, initialPR=double)). This is a tuple with key as node.
After the input processing is done, I calculate dangling node by filtering all records in the RDD by adjcList = [] using .filter() then use .map() to pick only value part(adjlist, pagerank) then do another .map() which takes (adjlist, pagerank) and performs pagerank/countOfNodes function on all the records with blank adjacency list. Then .map() takes only new pagerank value and calls .reduce() on it and returns accumulated single value of sum of all the pagerank values for dangling nodes, same type as held in RDD

Next, I calculate pagerank contributions by filtering all records in the RDD by adjcList != [] using .filter() then use .map() to pick only value part(adjlist, pagerank). Then .flatMap() on this value to split the adjc list by "," which gives an RDD array(string) and then another .map() on this RDD to make a new structure (node, pagerank/no.adjancenyList) and return this RDD structure and the .reduceByKey accumulates single values of pagerank contributions by key. The Then used mapValues() which applies the pagerank calculation function on the values part of the RDD tuple(string,double).
Now I have pagerank new pagerank calculated for each node and I join these values to existing input values using a .join(), it joins by key of the RDD tuple which is the node and replace the old pagerank value with the new rank calculated using .map() function.
Run this for 10 iterations and keep replacing input records with new pagerank values.

After the iterations, pick up 2 values(pagerank, node) from the last RDD using .map() function and pick top(100) records which picks up the 100 highest records based on key which is pagerank. Then I use repartition(1) which basically sends the output to one partition then use .sortByKey() in descending order to sort those 100 records globally, then .map() and apply swap function of RDD(pagerank, node) which swaps key and value and then write it to output file

Comparison with Hadoop Job

```
val parser: Bz2WikiParser = new Bz2WikiParser
val input = sc.textFile(args(0),100).map(l =>
parser.parse(l)).filter(line => line != null).persist()
```

For the above line of code I had one job as pre-processing in MR which was running the Bz2parser as a Map Reduce job and creating a new file with parsed output. It had a Mapper class and an Identity reducer.

```
val initalRank = 1.toDouble./(input.count().toDouble)
```

To calculate initial rank I had a global counter which kept a record of num of nodes to calculate the initial page rank

```
var value = input.map(line => line.split(":")).map(fields =>
(fields.apply(0), (fields.apply(1), initalRank)))
```

I had a class called Node to store Adjacency list and pagerank of nodes. After reading the input line and splitting it by ":" I stored the list and outlinks and pagerank in different variables and depending on the size (if the size after splitting is 2), I assigned pagerank = initialRank

```
for (i <- 1 to 10) {
```

Had a while iterator which created 10 MR jobs and passed the output of one job as the input of next job.

```
val dangling = value.filter(f => "[]".equals(f._2._1))
  .map(fields => fields._2)
  .flatMap(t => t._1.substring(1, t._1.size - 1).split(",")
    .map(l => (l.trim, t._2 / nodes)))
  .map(l => l._2).reduce((pr, sum) => pr + sum);
```

In the Mapper I was checking if the outlink is empty then store the pagerank/nodes value in a global counter so that the sum can be retrieved in the reducer. The global counter was adding all the values from every map task which the .reduce() method is doing here.

```
val rank = value.filter(f => !"[]".equals(f._2._1))
  .map(fields => fields._2)
  .flatMap(t => t._1.substring(1, t._1.size.-(1)).split(",")
    .map(l => (l.trim, t._2./(t._1.substring(1, t._1.size -
1).split(",").size))))
  .reduceByKey((pr, sum) => pr + sum)
  .mapValues(sum => 0.15 / nodes + 0.85 * (sum + dangling))
  .map(fields => (fields._1.trim, fields._2))
```

CS-6240 Section 1 HOMEWORK 4 NAOMI JOSHI

In the Mapper I was checking if the outlinks are not empty then emit pagerank/outlinks.length for every outlink of the node (.map() takes care of this part here). Then I was adding all the pagerank contributions of same node in the Reducer using the property that reducer will send all records with same key to same reducer(.reduceByKey() takes care of this part here). Then I was calculating the total pagerank by the formula in Reducer(.mapValues() here) and also add the dangling node contribution after taking it from the global counter.

```
value = value.join(rank).map(l => (l._1, (l._2._1._1, l._2._2)))
```

In MR program I was passing the entire Node class (having adjacency list and pagerank for all nodes) from Mapper to Reducer to keep track of outlinks for a node. Here I am replacing the same input structure with new pagerank and also I am not creating output file for every job for 10 iterations.

```
sc.parallelize(value.map(line => (line._2._2, line._1))
  .top(100)).repartition(1).sortByKey(false).map(l =>
l.swap)saveAsTextFile (args(1))
```

In MR version I had a job for it which was calling a Mapper to emit(pagerank, node) then a Key Comparator to sort it by Key, then a Partitioner to send all data to 1 reducer and emit first 100 records from the reducer to output file.

Advantages and shortcomings for different approaches

Batch processing jobs are faster in Spark than the Hadoop MapReduce framework as MapReduce does not leverage the memory of the Hadoop cluster to the maximum and leads to number of reads and writes to the disc. Spark the concept of RDDs (Resilient Distributed Datasets) lets us save data on memory and preserve it to the disc. Hence iterative algorithms like PageRank involve I/O bottlenecks in the MapReduce implementations. Spark caches the intermediate dataset after each iteration and runs multiple iterations on this cached dataset.

Hadoop MapReduce has a very strict API that doesn't allow for as much versatility. MapReduce is written in Java and is very difficult to program. Spark has comfortable APIs for Java, Scala and Python. However, Spark documentation is not that readily available on internet as we got for MapReduce. It is difficult to understand the underlying functionality of Spark but for Map Reduce it was fairly easy and lots of resources available everywhere.

In my experience of working on assignment 4 I realized that working on machines with large memory (increasing machine to xlarge) is more advantageous in Spark than increasing number of machines as it processes data in memory so we need machines with higher memory.

Performance Comparison

- 6 m4.large machines (1 master and 5 workers)

Spark Application - 2 hours 42 minutes
Map Reduce program – 1 hour 15 minutes

- 11 m4.large machines (1 master and 10 workers)

  Spark Application - 1 hours 15 minutes
  Map Reduce program – 38 minutes 30 seconds

In my case Map Reduce program turns out to be faster as the pre-processing of parser is taking a long time in Spark. I tried splitting the textFile into partitions by specifying partitions in .textFile(args(0),minPartitions) and also using mapPartitions but it did not have any effect in the running time. Tried many versions of the program after optimizing the RDDs but couldn't reduce the runtime. I ran the program after commenting the entire code and just ran the parser and realized it was parser which was taking a very long time. It maybe because the input data is not getting transferred to multiple nodes and parallel job is not executing.

RESULTS OF TOP 100 FOR EMR RUN:

SPARK:
(0.0024988912884009637,United_States_09d4)
(0.0023578638364487414,2006)
(0.0013040180951661,United_Kingdom_5ad7)
(0.0010957587518200592,2005)
(9.125294927816241E-4,Biography)
(8.27688885098006E-4,England)
(7.837429065214866E-4,Canada)
(7.679692720690917E-4,France)
(7.67154443586047E-4,2004)
(6.980340909621359E-4,Germany)
(6.537691693775912E-4,Australia)
(6.136860292268109E-4,India)
(5.962941805863736E-4,2003)
(5.643816503752807E-4,Japan)
(5.446145581113944E-4,Internet_Movie_Database_7ea7)
(5.096751953170442E-4,Italy)
(4.99835064791384E-4,2002)
(4.8178058154963236E-4,2001)
(4.78501954679947E-4,Record_label)
(4.635280541045675E-4,London)
(4.62871389620945E-4,Europe)
(4.418434103359069E-4,World_War_II_d045)
(4.3356222857879843E-4,2000)
(4.163939852770871E-4,1999)

(4.136250420883285E-4,English_language)
(4.00049829259276E-4,Music_genre)
(3.9000036341397165E-4,Geographic_coordinate_system)
(3.877686944236457E-4,Wiktionary)
(3.8247329660229414E-4,Russia)
(3.7025404684561255E-4,Spain)
(3.653307441271891E-4,1998)
(3.5224261869926793E-4,Wikimedia_Commons_7b57)
(3.487992836234409E-4,1997)
(3.386808095558928E-4,Television)
(3.284830098195253E-4,Football_(soccer))
(3.271256766980505E-4,Scotland)
(3.2426218894820227E-4,1996)
(3.193056073610916E-4,New_York_City_1428)
(3.09788625366672E-4,1995)
(3.012368219841305E-4,Scientific_classification)
(2.9684357911321347E-4,China)
(2.9490160285036265E-4,1994)
(2.888360326412013E-4,Netherlands)
(2.8836055575076587E-4,Sweden)
(2.8659101801702775E-4,New_Zealand_2311)
(2.8163162135639745E-4,Actor)
(2.810774976259009E-4,Film)
(2.7809526813213034E-4,1991)
(2.778867568650249E-4,1993)
(2.698751574381405E-4,California)
(2.681166729428088E-4,Public_domain)
(2.6768731906135645E-4,1990)
(2.6623014640054886E-4,1992)
(2.549103440944037E-4,Album)
(2.524975182358995E-4,Record_producer)
(2.48140115134499E-4,1989)
(2.439400471320350 4E-4,Ireland)
(2.3930761466736206E-4,Latin)
(2.3831918144631445E-4,1980)
(2.3540533551862723E-4,1986)
(2.309087444034323E-4,1985)
(2.309009408524758E-4,Animal)
(2.2923831961815854E-4,1982)
(2.2854572220235562E-4,New_York_3da4)
(2.272738451869666E-4,1981)
(2.2726099749422246E-4,1979)
(2.2591976004428433E-4,1984)
(2.2589614258767773E-4,January_1)
(2.2542524307645815E-4,1987)
(2.2451010379839387E-4,1983)

(2.2448801474368924E-4,Studio_album)
(2.2158029869895145E-4,Norway)
(2.2074909103507874E-4,Poland)
(2.205464612355439E-4,1974)
(2.2020817430527066E-4,1988)
(2.1900623569673987E-4,Politician)
(2.16315501594064E-4,French_language)
(2.144966781882906E-4,South_Africa_1287)
(2.1443169963359073E-4,1976)
(2.137902211457383E-4,1970)
(2.1265228399062924E-4,1975)
(2.1237721305808398E-4,Paris)
(2.1106492154845038E-4,Mexico)
(2.087819855736749E-4,Brazil)
(2.0858051761803775E-4,1969)
(2.084558600710638E-4,1977)
(2.0810902248400714E-4,1972)
(2.0732899342606513E-4,1978)
(2.0559645428273825E-4,Soviet_Union_ad1f)
(2.0517843681622802E-4,Personal_name)
(2.0469083909906036E-4,1945)
(2.0354560461654947E-4,Switzerland)
(2.0318402109306702E-4,Greece)
(2.0130354517521973E-4,1973)
(1.985637124454361E-4,Pakistan)
(1.9846474626736575E-4,1971)
(1.982197372173447E-4,Iran)
(1.968482459931893E-4,1968)
(1.9640827512144748E-4,1967)
(1.9397321645722352E-4,World_War_I_9429)

MAP REDUCE:

2006:0.005129149449041743
United_States_09d4:0.004492913048100462
United_Kingdom_5ad7:0.002518389237467937
2005:0.0022707837004375146
France:0.0018790856977073558
2004:0.001624343156990327
Germany:0.0015111180070295563
England:0.00147740430672144
Italy:0.00142744191736116
Canada:0.001333442820097365
2003:0.001242007682310821
Australia:0.0011528916758065868
Japan:0.001132273370032307

index:0.0011179708189963317
English_language:0.001087251938677357
India:0.0010807092676739548
Europe:0.0010184917985445016
World_War_II_d045:9.979047327229335E-4
2002:9.933718905106507E-4
Wikimedia_Commons_7b57:9.57885690493096 8E-4
2001:9.470229973092762E-4
Russia:9.4193670036599E-4
London:9.415947679397907E-4
Wiktionary:9.340206105601811E-4
Spain:9.322447750533802E-4
Biography:8.528673661294494E-4
2000:8.436482257689477E-4
1999:8.372676240584725E-4
Internet_Movie_Database_7ea7:7.385166283045563E-4
1998:7.147292341051591E-4
1997:6.969437116915292E-4
Latin:6.849769491685031E-4
Sexagenary_cycle:6.739044859623275E-4
January_1:6.720499510744478E-4
Netherlands:6.602002739536533E-4
China:6.56519087490454E-4
New_York_City_1428:6.494292875183218E-4
1996:6.45134993898468E-4
Scotland:6.354585022042599E-4
French_language:6.278248807458337E-4
1995:6.214962567718055E-4
Geographic_coordinate_system:6.137256908925661E-4
Sweden:6.114641967777905E-4
1991:6.047570171259578E-4
Gregorian_calendar:5.9885430177269E-4
1994:5.983139546354373E-4
Soviet_Union_ad1f:5.87199221659451E-4
1990:5.741665610645349E-4
1993:5.638555436874276E-4
1992:5.502804642970164E-4
Egypt:5.465092619963279E-4
1945:5.418940185793729E-4
International_Phonetic_Alphabet_96f8:5.398502298942869E-4
Greek_language:5.349913456227379E-4
1980:5.341064900117803E-4
1989:5.304447533131564E-4
Public_domain:5.297973987792992E-4
New_Zealand_2311:5.204552183295226E-4
1979:5.187426918130323E-4

Poland:5.165131290618411E-4
1974:5.148772970554488E-4
Television:5.148403204539094E-4
1986:5.14811560739523E-4
Paris:5.14122154711963E-4
1970:5.133354288759246E-4
1981:5.047591183641088E-4
1976:5.045023036319759E-4
European_Union_e368:5.033218268196827E-4
1969:5.005275839081883E-4
1975:5.004875507592081E-4
1982:4.986493708048685E-4
1985:4.940912247572052E-4
Greece:4.906916913891982E-4
1972:4.888869125985893E-4
Portugal:4.8683615611325436E-4
Austria:4.8605438723816876E-4
German_language:4.8470195003976147E-4
Switzerland:4.8448502853132617E-4
1984:4.8110635941921544E-4
Ireland:4.7840266995451895E-4
1971:4.779233883504469E-4
1973:4.7783779512257265E-4
1983:4.766190530592744E-4
1977:4.74645548461477E-4
1968:4.6936937990713453E-4
1987:4.684503010801367E-4
19th_century:4.680616296357845E-4
1967:4.660088277689139E-4
1978:4.649199223288549E-4
People's_Republic_of_China_82bf:4.642805359038263E-4
World_War_I_9429:4.626416308239336E-4
1988:4.6012133590514205E-4
Turkey:4.594435033814591E-4
Israel:4.580987216579595E-4
Belgium:4.574951344082336E-4
Mexico:4.5694478205556097E-4
Norway:4.5599934796782687E-4
Denmark:4.532754907724953E-4
South_Africa_1287:4.523927537854544E-4
Football_(soccer):4.51420881632047E-4

RESULTS OF TOP 100 FOR LOCAL RUN ON SIMPLE FILE:

SPARK:
(United_States_09d4,0.005760933936782571)

(Wikimedia_Commons_7b57,0.004832053811141776)
(Country,0.0038553276326007404)
(Europe,0.0026127897899594498)
(England,0.002603054906353909)
(United_Kingdom_5ad7,0.0025788091592809145)
(Water,0.0025636578006176833)
(France,0.0025507865130214233)
(Germany,0.002537950508215886)
(Animal,0.0024784941451436375)
(Earth,0.0024285970327204013)
(City,0.0021801066435404334)
(Week,0.0020635105312479187)
(Sunday,0.001936616097228122)
(Asia,0.001928471852633296)
(Monday,0.0019067452421453542)
(Wednesday,0.001888892422240173)
(Friday,0.001844576331728899)
(Wiktionary,0.0018381993823523455)
(Saturday,0.0018240851927825966)
(Money,0.001818855666964163)
(Thursday,0.0018000485554921309)
(Tuesday,0.0017874105610209995)
(Plant,0.0017863598893336286)
(Computer,0.001737284233848977)
(Italy,0.0017333612544653191)
(English_language,0.0017170358655830538)
(India,0.0016721803983134004)
(Number,0.0016320279617125474)
(Government,0.0016308033370710595)
(Spain,0.001549779167035176)
(Day,0.0015082513247293816)
(Japan,0.0014521527422957058)
(Human,0.0014386950277017353)
(People,0.0014335849706506203)
(Wikimedia_Foundation_83d9,0.001379632081102132)
(Canada,0.0013614561802000119)
(China,0.0013500537875433447)
(Energy,0.0013426009485409882)
(Food,0.0013134344483575683)
(Sun,0.00130267678858231)
(Science,0.0013011682635951385)
(Australia,0.0012855359955304627)
(Mathematics,0.0012847043795535826)
(index,0.0012743430838339214)
(Television,0.0011990567617252185)
(Russia,0.0011861002969024773)

(Year,0.001178166080831231)
(Music,0.0011579985675446373)
(Language,0.0011247589700059438)
(Capital_(city),0.0011035248273079676)
(Metal,0.0010912709218789116)
(2004,0.001073696372803391)
(Wikipedia,0.0010732595809068634)
(State,0.0010696270889741075)
(Greek_language,0.0010587616705181057)
(Religion,0.0010558210414714547)
(Sound,0.0010380570298355644)
(London,0.0010379957046549911)
(Planet,0.001033018046118854)
(Scotland,0.0010251770181726552)
(Africa,0.0010090440387671036)
(Greece,0.0010040671368184127)
(20th_century,9.812819413470295E-4)
(19th_century,9.53086881118386E-4)
(Law,9.430916173701821E-4)
(Geography,9.387565762019182E-4)
(Liquid,9.353453133558265E-4)
(World,9.25518870394235E-4)
(Poland,9.228639439831613E-4)
(Scientist,9.144472302253993E-4)
(Society,9.070169903208096E-4)
(Atom,8.942109592672027E-4)
(Latin,8.729547859549844E-4)
(Light,8.689268462075285E-4)
(War,8.688582415673335E-4)
(History,8.675943275791358E-4)
(God,8.566188278826124E-4)
(Netherlands,8.52812220935503E-4)
(Culture,8.456839508583859E-4)
(Centuries,8.399368301278492E-4)
(Building,8.338265527992222E-4)
(Chemical_element,8.3002928805687E-4)
(Turkey,8.28964684110964E-4)
(Sweden,8.224068771112319E-4)
(Plural,8.184385843755101E-4)
(Information,8.15547347475843E-4)
(Portugal,7.988215519598837E-4)
(Austria,7.792228825521131E-4)
(Disease,7.78106327221732E-4)
(Denmark,7.778805099323484E-4)
(Species,7.745882900103362E-4)
(Cyprus,7.579921589910843E-4)

(Ocean,7.550078907792369E-4)
(Biology,7.53439561963392E-4)
(Capital_city,7.521188580874944E-4)
(Inhabitant,7.504536673340602E-4)
(Book,7.44166831046308E-4)
(University,7.440645610122928E-4)
(List_of_decades,7.431337260319948E-4)

MAP REDUCE:
United_States_09d4:0.0454852507732353
Week:0.03772705069781671
Sunday:0.029203165906032547
Monday:0.028553038474976013
Wednesday:0.02825025222869564
Friday:0.02741482898377306
Saturday:0.027087909272236307
Day:0.02680241129328086
Thursday:0.02663301041406329
Tuesday:0.02647985715913479
Country:0.025347654552460554
Wikimedia_Commons_7b57:0.025030209342995477
Europe:0.01831436627025558
United_Kingdom_5ad7:0.01726088026809777
Earth:0.016532332974908595
France:0.014063151455722319
Water:0.013990694074367606
Germany:0.012923231029418638
Asia:0.012715042483263852
England:0.012657010570046909
City:0.012343067541579591
Animal:0.011625011036475933
Sun:0.011349370183746108
Year:0.01115665332330853
English_language:0.010747789338969041
Money:0.010391974369820988
Government:0.010244875782399979
Italy:0.010184014406384702
Number:0.010162091446642676
index:0.01001340626273539
India:0.009831644364015686
Canada:0.008722623886963899
Wiktionary:0.008586675916110575
Spain:0.008551767401417122
Plant:0.008538664323993882
Planet:0.008314112717676667

People:0.008269425955768829
Computer:0.007942147936913066
Japan:0.007920412477290431
Wikimedia_Foundation_83d9:0.007767705377697868
China:0.0076336577985531935
Moon:0.007525919978151609
Australia:0.007486019926958532
Energy:0.007484332904204559
Russia:0.007260311288695717
Human:0.007199562564065275
Thor:0.007184595619813111
State:0.0070957074026162855
Science:0.006838893673541622
20th_century:0.006781715769856163
Capital_(city):0.006651064831413421
19th_century:0.006442635008879791
Geography:0.006399262623998236
God:0.0063775301228415696
Greece:0.006312461093133384
Africa:0.006276024113201375
Greek_language:0.006231107226943432
Religion:0.006200854234929282
Mathematics:0.006191403199456686
Scotland:0.006099852514683112
Food:0.006053776661276938
2004:0.005984866932524986
February:0.005840806964818388
Language:0.005811990915690682
Poland:0.005751616796439403
Wikipedia:0.005734922742142722
Society:0.005732066709176805
Sweden:0.005632091057134353
January:0.005608437803081466
World:0.0055824994171650285
Turkey:0.00555244722320288
History:0.0055453986994576454
Centuries:0.005530586076582796
Cyprus:0.0054937006824995775
Television:0.00546350491606917 1
Culture:0.005435489451664863
Law:0.005388481727693353
Odin:0.005381232839626957
Sound:0.005340248109379373
March:0.005314159609413709
Latin:0.005286269388816302
Month:0.005225667256954458

London:0.005204979562437963
Music:0.005189592129546086
War:0.005168338166881445
List_of_decades:0.005078875341472842
Denmark:0.005063828554266997
Portugal:0.00500231667778429
Greek_mythology:0.004976419222982018
Metal:0.004966963218853904
Plural:0.004916565541951172
Austria:0.004860750120543997
Scientist:0.004813227495006541
Liquid:0.004775088070909461
April:0.00476278120169472
Netherlands:0.004757545211132822
Light:0.004756346827541554
Norse_mythology:0.004715423222849549
Information:0.0047076801314134735
Atom:0.004584355253553932

The top 100 nodes for both EMR and local execution in Spark and Map Reduce are more or less the same but the values of pageranks are different. This maybe because in Map Reduce the nodes with no outlinks and inlinks were considered in all the 10 jobs so their dangling mass was added everytime. In spark, these nodes are removed in every iteration and hence in the interation we don't get the dangling mass for those nodes. And ideally I believe we should not consider dangling mass for nodes which do not have any outlinks or inlinks.