**SEC 02**

**HW 4**

**Darshan Patel**

[**patel.dars@husky.neu.edu**](mailto:PATEL.DARS@HUSKY.NEU.EDU)

**Design Discussion:**

A spark drive program is the first point of the contact for executing the spark program. It handles the complete flow of spark program. Driver program initialize the executor processes across various nodes on the cluster and assign them the work which is known as task in spark.

**Step 1:** Read the input file.

**sc.textFile(input, nbrOfPartitioner)**

By this line, spark program knows where the input source data is but it doesn’t read it as textFile command is transformation so it would be lazily evaluated.

Based upon the source data or provided number of partition value, spark creates that many partition of the input data.

**Step 2:** Parse input data

**val** allPages = *sc*.textFile(*input*, *nbrOfPartitioner*).mapPartitions(*lines* => **parseInputAndConvertDeadIntoDangling**(*lines*)).keyBy(*line* => *line*.*\_1*)

Once the input source is given to spark, mapPartitions will apply the **parseInputAndConvertDeadIntoDangling** scala function to each input of all the partition. The mapPartitions call will generate page name and its adjacency list from the partition of the RDD which is generated by textFile.

mapPartitions is also a transformation so spark won’t evaluate it until any action is called. Despite mapPartitions is not executed on the fly but Spark creates the execution plan on how to execute mapPartitions.

keyBy will convert the RDD into pairwise RDD.

**Step 2a:** mapPartition function.

**def** parseInputAndConvertDeadIntoDangling(*pages*: *Iterator*[*String*]) : *Iterator*[(*String*, util.**List**[*String*])] ={  
  
 **val** pageWithAdj = *pages*.filter(*line* => **isGoodName**(*line*)).  
 map( *line* => {  
 **val** node = **bz2WikiParser**(*line*)  
 (node.*pageName*, node.*adjPages*)  
 })  
  
 **val** adjPageWithDummyAdj = pageWithAdj.flatMap(*line* => **makeAdjPageWithDummyAdj**(*line*))  
 pageWithAdj ++ adjPageWithDummyAdj  
  
}

The above code first parses the input and converts the data of each partition into tuple (page name, adjacency pages). It also converts dead node into dangling node using **makeAdjPageWithDummyAdj** scala function using flatMap.

**Step 2b:** dead node to dangling node

**def** makeAdjPageWithDummyAdj(*page* : (*String*, util.**List**[*String*])): *Iterator*[(*String*, util.**List**[*String*])] ={  
  
 **val** adjPages = *page*.*\_2* **val** newArray = **for** (value <- adjPages) **yield** {  
 **val** emptyAdjPages = **List**[*String*]().asJava  
 (value, emptyAdjPages)  
 }  
 newArray.iterator  
  
}

The above code is used for the flatMap execution in the parseInputAndConvertDeadIntoDangling scala function call. It loops over each adjacency page and creates a tuple with adjacency page name and empty adjacency page list.

**Step 3:** Reduce the input data

**val** reducedAllPages = allPages.reduceByKey({(*a*, *b*) =>  
 **if**(*a*.*\_2*.length == **0**){  
 *b* }**else**{  
 *a* }  
})

As I am converting the dead nodes into dangling nodes by emitting/mapping each adjacency node/page with empty adjacency page list. I need a reducer to make sure there is no duplicate page name and each page should contain either non zero adjacency page list or empty adjacency list.

reduceByKey is transformation so it would be evaluated lazily. Once is executed, spark will shuffle the data as the various data belong to a key should be on same node/machine to reduce them by the key.

**Step 4:** Persist the data

reducedAllPages.persist()

The persist command tells each cluster node to stores the any partition data of the reducedAllPages RDD in its memory and reuse it for the future tasks.

**Step 5:** Accumulator

**val** nbrOfPages = sc.longAccumulator(**"Total Number of pages"**)  
allPages.foreach(*x* => nbrOfPages.add(**1**))

Above code creates the accumulator and counts the number of pages in the allPages RDD.

Foreach is action in the spark so at Each node will do the sum and then send it result to master node and at master node final sum will happen.

**Step 5:** Add Initial Page Rank

**var** allPagesWithPR = allPages.mapValues(*value* => (*value*.*\_2*, **1.0**/pageCount))

mapValues will add the page rank value for each value of the pair wise RDD. As mapValues is transformation the evaluation would be lazy.

**Step 6:** PageRank Calculation

**Step 6a:** Distribute Page Rank

**def** distributePageRank(*pageInfo* : **RDD**[(*String*, (util.**List**[*String*], Double))]): **RDD**[(*String*, (*String*, Double))]={  
  
 **val** prDistribution = *pageInfo*.flatMap( *value* => {  
 **val** adjPages = *value*.*\_2*.*\_1* **val** pageRank = *value*.*\_2*.*\_2* **val** adjPageCount = adjPages.length  
 **val** pageRankDist = **for** (adjPage <- adjPages) **yield** {  
 (adjPage, pageRank/adjPageCount)  
 }  
 pageRankDist  
 }).keyBy{*line* => *line*.*\_1*}  
  
 **val** reducedPRDistribution = prDistribution.reduceByKey({(*a*, *b*) => (*a*.*\_1*, *a*.*\_2*+*b*.*\_2*)})  
 reducedPRDistribution  
}

The above method distributes the page rank to the adjacency pages.

flatMap command will generate values for each entry in the pageInfo RDD. This command would ask each node to perform flatMap on the partition it has.

keyBy will convert the resulting RDD into pairwise RDD. flatMap and keyBy will be evaluated lazily.

reduceByKey is used to sum the page rank contribution of each adjacency page. Again lazy evaluation.

**Step 6a:** Calculate Page Rank and Sum of dangling node

**def** calculateNewPageRank(*distAdjPagePR* : **RDD**[(*String*, (*String*, Double))],  
 *allPages* : **RDD**[(*String*, (util.**List**[*String*], Double))],  
 *pageCount* : Long,  
 *delta*: Double,  
 *sc* : **SparkContext**,  
 *dangPRSum* : **DoubleAccumulator**,  
 *allPRSum* : **DoubleAccumulator**) : **RDD**[(*String*, (util.**List**[*String*], Double))] ={  
  
 **val** pagesWithContributionPR = *allPages*.leftOuterJoin(*distAdjPagePR*)  
 **val** alpha = **0.15d  
  
 val** newPagePR = pagesWithContributionPR.mapValues( *value* => {  
  
 **val** adjPages = *value*.*\_1*.*\_1* **val** curretnPageRank = *value*.*\_1*.*\_2* **val** prContribSum = *value*.*\_2* **var** newPageRank = **0.0  
 if**(prContribSum != **None**){  
 newPageRank = (alpha/*pageCount*) + (**1**-alpha)\*(prContribSum.get.*\_2* + (*delta*/*pageCount*))  
 }**else**{  
 newPageRank = (alpha/*pageCount*) + (**1**-alpha)\*(*delta*/*pageCount*)  
 }  
 **if**(adjPages.size() == **0**){  
 *dangPRSum*.add(newPageRank)  
 }  
 *allPRSum*.add(newPageRank)  
 (adjPages, newPageRank)  
 })  
 newPagePR  
}

The above code first performs the left outer join between the main RDD and page rank contribution RDD. The result RDD is used to calculate new page rank.

leftOuterJoin is transformation which join the page information with its page rank contribution from the other pages. The spark will do shuffling to do the join so it is memory consuming operation.

After the join, map is applied to each record in rdd to calculate the new page rank and accumulator is incremented to calculate sum of dangling node page rank.

**Step 7:** Force execution

**def** evaluate[*T*](*rdd*:**RDD**[*T*]) = {  
 *rdd*.sparkContext.runJob(*rdd*,(*iter*: *Iterator*[*T*]) => {  
 **while**(*iter*.hasNext) *iter*.next()  
 })  
}

**evaluate**(allPagesWithPR)

As most of the operation were transformation, spark doesn’t execute them and we need sum of dangling node page rank for the next page rank iteration so we need to perform action to force the spark do the execution.

Above code calls an action of the RDD due to which spark will perform all the operation to generate the RDD and accumulator would be calculated.

**Step 8:** Top 100

**val** top100 = allPagesWithPR.mapPartitions(*lines* => **sortPageLocally**(*lines*), *preservesPartitioning* = **true**).  
 takeOrdered(**100**)(**Ordering**[Double].reverse.on(*line* => *line*.*\_2*))  
sc.parallelize(top100, **1**).saveAsTextFile(output)

The above code finds the top 100 in each partition using mapPartitions and then give that records to master node and its sort them and gives the final 100.

mapPartitions performs the given function on each partition and generates the RDD which contains only top 100 pages with the highest page ranks.

takeOrdered forces each partition to send its data to master node and then master node selects the top 100 for all of them.

Parallelize converts the resulted top 100 array into RDD and save it as text file.

**MapReduce and Spark Implementation**

How each spark program line is implemented in my Map reduce jobs functionality.

**Step 1:** Read the input file.

**Equivalent Map Reduce Job/Task:**

My Parser Hadoop Mapper (ParserMapper.java) was reading each line of the input file.

**Step 2:** Read input file and parse input data (convert dead node to dangling node)

**Step 2a:** mapPartition function

**Step 2b:** dead node to dangling node

**Equivalent Map Reduce Job/Task:**

My Parser Hadoop Mapper (ParserMapper.java) was reading each line of the input file and applying BZ2 parser on each line and emitting the page name as key and adjacency pages as value. I had identity Parser Hadoop reducer.

To convert the dead node into dangling node, I have created a new map reduce job (NumberMapper.java and NumberReducer.java) which turns out to be not needed as I could have done that task in parsing map reduce job.

In the dead node to dangling node conversation map reduce job mapper, I was emitting adjacency page as key and empty list as values.

**Step 3:** Reduce the input data

**Equivalent Map Reduce Job/Task:**

(NumberReducer.java) Dead node to dangling node map reduce job reducer was checking all the adjacency list for the given page name(key) and reduce them to a single adjacency page list.

**Step 4:** Persist the data

**Equivalent Map Reduce Job/Task:**

There was no way to persist data on memory as each map reduce job supposed to read from the disk and write to the disk

**Step 5:** Accumulator

**Equivalent Map Reduce Job/Task:**

I have used Global counter in reducer of the dead node to dangling node job (NumberReducer.java)

**Step 6a:** Distribute Page Rank

**Equivalent Map Reduce Job/Task:**

The Mapper of Page Rank Map Reduce job was distributing the page rank of the current page to its adjacency page list.

**Step 6a:** Calculate Page Rank

**Equivalent Map Reduce Job/Task:**

The Reducer of Page Rank Map Reduce job was getting the all the page rank contribution and adjacency page list of a page. Where I calculated the new page rank and incremented global counter to calculate sum of dangling node page rank

**Step 7:** Force execution

**Equivalent Map Reduce Job/Task:**

As there is not lazy evaluation in map reduce there is no equivalent component.

**Step 8:** Top 100

**Equivalent Map Reduce Job/Task:**

There was a separate map reduce job to calculate top 100. Where in the map I used to find the local top 100-page rank and reducer was finding global top 100 page with highest page rank.

**Advantage and short coming of the different approaches**

**Performance Comparison**

|  |  |  |
| --- | --- | --- |
|  | Spark  (Seconds) | Hadoop  (Seconds) |
| 6 Machines | 5802 | 3818 |
| 11 Machines | 2996 | 1944 |

The Hadoop performance is much faster than spark. The reason could be the spark shuffling operations as spark shuffle operation require high amount of data transfer. This high amount data transfer could create a bottleneck in the network communication.

**Output Results:**

**Full Dataset:**

Spark:

(United\_States\_09d4,0.0022823409704574448)

(2006,0.0010622188890400607)

(United\_Kingdom\_5ad7,0.0010423984820144736)

(Biography,8.766426371988046E-4)

(2005,7.925523069490351E-4)

(England,7.650153849964413E-4)

(Canada,7.443792471671949E-4)

(Geographic\_coordinate\_system,6.79304349292132E-4)

(France,6.240501044074982E-4)

(2004,6.223308866790076E-4)

(Australia,5.91284786692608E-4)

(Germany,5.651787175518646E-4)

(India,5.083933749487629E-4)

(2003,5.082693501997339E-4)

(Japan,5.04471056598319E-4)

(Internet\_Movie\_Database\_7ea7,4.679659588370048E-4)

(Europe,4.386487318207832E-4)

(Record\_label,4.367717747000707E-4)

(2001,4.217963292450441E-4)

(2002,4.175043725346054E-4)

(Music\_genre,4.152882522660081E-4)

(Population\_density,4.10268763768178E-4)

(World\_War\_II\_d045,4.101369961023399E-4)

(2000,4.0175879140683064E-4)

(Italy,3.8147214231266957E-4)

(London,3.7452735250189496E-4)

(Wiktionary,3.7134042817402163E-4)

(Wikimedia\_Commons\_7b57,3.678542093549606E-4)

(English\_language,3.566564951786024E-4)

(1999,3.505864951953201E-4)

(Spain,3.1136804334278126E-4)

(1998,3.080723957031494E-4)

(Russia,2.940248856062858E-4)

(Television,2.93534243381881E-4)

(1997,2.914380076505924E-4)

(New\_York\_City\_1428,2.8775615914428094E-4)

(Football\_(soccer),2.8740617438993075E-4)

(Census,2.8369305677747905E-4)

(1996,2.7959341358994996E-4)

(Scotland,2.7832655762356664E-4)

(Scientific\_classification,2.6863667246625216E-4)

(1995,2.675432518103803E-4)

(China,2.6498879186775243E-4)

(Square\_mile,2.6472101499855073E-4)

(Population,2.6414547378184473E-4)

(California,2.619021519957263E-4)

(Record\_producer,2.526668128830162E-4)

(1994,2.508180860273167E-4)

(Film,2.495751201460018E-4)

(Public\_domain,2.4911096749213006E-4)

(Sweden,2.489415167526218E-4)

(New\_Zealand\_2311,2.4502428683150244E-4)

(New\_York\_3da4,2.4134670452198554E-4)

(Marriage,2.4122767871494137E-4)

(United\_States\_Census\_Bureau\_2c85,2.392097104605315E-4)

(Netherlands,2.3739901655432717E-4)

(1993,2.3719097913036915E-4)

(Politician,2.357191005558706E-4)

(1991,2.341884192837532E-4)

(Album,2.3264980544823946E-4)

(1990,2.312588373342116E-4)

(1992,2.2966233876375343E-4)

(Actor,2.2714323258775807E-4)

(Studio\_album,2.2552174961603806E-4)

(Per\_capita\_income,2.240635554390752E-4)

(Ireland,2.2271919395570284E-4)

(Poverty\_line,2.2020576458393602E-4)

(Latin,2.1969906033447186E-4)

(1989,2.126436997241068E-4)

(Norway,2.0947031210155227E-4)

(Website,2.0749942546507836E-4)

(1980,2.0264744012002143E-4)

(Animal,2.017799022393574E-4)

(Personal\_name,2.0167874134272034E-4)

(Area,1.97721019713018E-4)

(Brazil,1.9579665517932286E-4)

(1986,1.9559545677374682E-4)

(Poland,1.9543991566001132E-4)

(1985,1.9309871062279775E-4)

(1987,1.925601776195502E-4)

(1983,1.9118271682179103E-4)

(1982,1.9062756124643818E-4)

(1981,1.8897530673580705E-4)

(1984,1.887870758626447E-4)

(1979,1.8871008897038065E-4)

(1988,1.8852887274792624E-4)

(1974,1.871778112591281E-4)

(World\_War\_I\_9429,1.869087475249676E-4)

(Paris,1.8679114367267023E-4)

(Mexico,1.8596437403976148E-4)

(French\_language,1.8592991557653794E-4)

(USA\_f75d,1.837279850490253E-4)

(White\_(U.S.\_Census)\_c45a,1.8180989232848338E-4)

(1970,1.8134709201077036E-4)

(1975,1.791612510955626E-4)

(1976,1.7911988362026622E-4)

(19th\_century,1.789082528085707E-4)

(January\_1,1.7862979279173214E-4)

(South\_Africa\_1287,1.7847812793180632E-4)

(Africa,1.783625324031857E-4)

MapReduce:

1 United\_States\_09d4 0.0025096659140062855

2 2006 0.0011785851099251927

3 United\_Kingdom\_5ad7 0.0011487161194704374

4 Biography 9.992215951309072E-4

5 2005 8.809871915095769E-4

6 England 8.427043612430178E-4

7 Canada 8.226895820054495E-4

8 Geographic\_coordinate\_system 7.872634598367912E-4

9 France 7.140708411359073E-4

10 2004 6.969835358617605E-4

11 Australia 6.517292777479323E-4

12 Germany 6.408297799111921E-4

13 2003 5.680818542160823E-4

14 India 5.661760460350251E-4

15 Japan 5.601519822971705E-4

16 Internet\_Movie\_Database\_7ea7 5.18537565474506E-4

17 Europe 4.974873798410063E-4

18 Record\_label 4.8117764907760715E-4

19 2001 4.7102052562857316E-4

20 2002 4.6246156962486144E-4

21 World\_War\_II\_d045 4.6025459057193657E-4

22 Population\_density 4.5976657176419883E-4

23 Music\_genre 4.594408094492676E-4

24 2000 4.453238934353059E-4

25 Italy 4.325699542852E-4

26 Wikimedia\_Commons\_7b57 4.3009904914980193E-4

27 London 4.119627766504688E-4

28 Wiktionary 4.085318519197703E-4

29 English\_language 3.9744566144720095E-4

30 1999 3.911376758442965E-4

31 Spain 3.5702102098416835E-4

32 1998 3.416381199997776E-4

33 Russia 3.365076799211658E-4

34 Football\_(soccer) 3.2801672225369884E-4

35 Television 3.2394751054587743E-4

36 1997 3.21727554089625E-4

37 New\_York\_City\_1428 3.156950337506983E-4

38 Census 3.120685894659107E-4

39 1996 3.0982458919264286E-4

40 Scotland 3.0633359347667383E-4

41 Scientific\_classification 3.005168604761116E-4

42 Population 2.999884460651893E-4

43 1995 2.9574561897426543E-4

44 China 2.938095689265429E-4

45 Square\_mile 2.90409172499914E-4

46 California 2.873175428034615E-4

47 Sweden 2.802852825313939E-4

48 Public\_domain 2.7987055080668123E-4

49 1994 2.778534611857294E-4

50 Record\_producer 2.767898044523298E-4

51 Film 2.757826155116941E-4

52 New\_Zealand\_2311 2.689209061527103E-4

53 Politician 2.6741428682174355E-4

54 Netherlands 2.6548826312608123E-4

55 New\_York\_3da4 2.6497705408920047E-4

56 Marriage 2.64068674147525E-4

57 1993 2.631330725353927E-4

58 United\_States\_Census\_Bureau\_2c85 2.6126112302550834E-4

59 1991 2.591157727900411E-4

60 1990 2.561701235125258E-4

61 Album 2.55920901545778E-4

62 1992 2.5512799465044427E-4

63 Actor 2.5317269660773284E-4

64 Studio\_album 2.482453786025729E-4

65 Poland 2.4797876757512894E-4

66 Ireland 2.462651466657447E-4

67 Latin 2.459683690951089E-4

68 Per\_capita\_income 2.4513166928885177E-4

69 Norway 2.409407495306371E-4

70 Poverty\_line 2.409344440180673E-4

71 Km² 2.4002316447721724E-4

72 1989 2.3799407253437434E-4

73 Website 2.2983868860660292E-4

74 Animal 2.2664362885396202E-4

75 Area 2.2623379952340096E-4

76 Brazil 2.252994673856048E-4

77 Personal\_name 2.2407017500746229E-4

78 1980 2.2375942940022702E-4

79 1986 2.174827821456171E-4

80 Paris 2.1350339112740597E-4

81 1985 2.134335127869099E-4

82 1987 2.1270289004420793E-4

83 1982 2.124346258103653E-4

84 1983 2.1209917322222692E-4

85 Mexico 2.1150808749435126E-4

86 1984 2.0936999000721433E-4

87 1988 2.0909675454609372E-4

88 1981 2.0902949542015087E-4

89 1979 2.0841309043014345E-4

90 French\_language 2.076744646996704E-4

91 World\_War\_I\_9429 2.074910554155313E-4

92 1974 2.067843367126416E-4

93 Africa 2.0573930180696197E-4

94 South\_Africa\_1287 2.0447924789190967E-4

95 USA\_f75d 2.0297351146755498E-4

96 1970 2.0167466792105163E-4

97 1975 1.9924206478253367E-4

98 White\_(U.S.\_Census)\_c45a 1.9906382978926223E-4

99 January\_1 1.9874365160208474E-4

100 19th\_century 1.986711444118808E-4

**Sample Dataset:**

Spark:

(United\_States\_09d4,0.004849536758629031)

(Wikimedia\_Commons\_7b57,0.004519465659082289)

(Country,0.0036444498237329943)

(England,0.002599741451153389)

(Water,0.0024732590996438208)

(City,0.0023806983228107483)

(Animal,0.002366408586979564)

(Germany,0.0022861232316742494)

(France,0.002191548696469876)

(United\_Kingdom\_5ad7,0.002191000821937883)

(Earth,0.002130177273920159)

(Europe,0.0018997344882401763)

(Wiktionary,0.0016420672144561086)

(English\_language,0.0016268720550796235)

(Government,0.0016029114687094391)

(Computer,0.0015992226767032858)

(India,0.0015913662313415877)

(Money,0.0015415553200311789)

(Japan,0.0014560061595082642)

(Plant,0.0014129747500999496)

(Italy,0.0014121153529784013)

(Spain,0.0013952662774137441)

(Canada,0.00138236136156337)

(Food,0.0013205889208451967)

(Human,0.0013103201493322005)

(China,0.0012996142389554144)

(People,0.0012812295902786465)

(Australia,0.0012424336995870413)

(Capital\_(city),0.0011912224027143619)

(Asia,0.0011874227188683134)

(Television,0.0011870998738080854)

(State,0.0011479611116950385)

(Sun,0.0011474553356357487)

(Number,0.0011467577119406275)

(Sound,0.0011450417819045228)

(Mathematics,0.0011433262224262186)

(Science,0.001134176356060682)

(Metal,0.0011027815265556664)

(2004,0.0011020310880070013)

(Year,0.0010822942739963483)

(Language,0.001072944425122411)

(Russia,0.0010654855433022167)

(Wikipedia,0.0010385599252076502)

(Music,0.0010251810075558828)

(Religion,0.001023038400003153)

(19th\_century,0.0010221605558902436)

(Scotland,9.850661661006106E-4)

(Greece,9.825765424973025E-4)

(20th\_century,9.816536873545401E-4)

(London,9.645125171060077E-4)

(Latin,9.562559640751263E-4)

(Greek\_language,9.298640004685307E-4)

(Energy,9.137240455568098E-4)

(World,9.109207354527348E-4)

(Centuries,9.087067892987958E-4)

(Culture,8.709527838770002E-4)

(History,8.671668697643267E-4)

(Netherlands,8.495337270418076E-4)

(Liquid,8.390385143775117E-4)

(Inhabitant,8.281160770520924E-4)

(Society,8.268045413942235E-4)

(Light,8.256181041312925E-4)

(Wikimedia\_Foundation\_83d9,8.239009047086908E-4)

(Planet,8.231491964907471E-4)

(Image,8.21332957487155E-4)

(Law,8.212708864075959E-4)

(Scientist,8.198495133067633E-4)

(List\_of\_decades,8.168991883904769E-4)

(Atom,8.089400316143181E-4)

(Geography,8.088314698570061E-4)

(Africa,8.087285471902451E-4)

(Uniform\_Resource\_Locator\_1b4e,7.98873510732818E-4)

(Capital\_city,7.847146892685078E-4)

(Turkey,7.785877452959423E-4)

(Poland,7.740063546488542E-4)

(Plural,7.544056441483656E-4)

(Electricity,7.479443187876577E-4)

(Book,7.429387059691324E-4)

(Car,7.415384003370616E-4)

(Sweden,7.391581757838297E-4)

(Building,7.388277641559798E-4)

(Biology,7.272847554602692E-4)

(War,7.178825397486962E-4)

(God,7.166338857173402E-4)

(Chemical\_element,7.087534494008209E-4)

(North\_America\_e7c4,7.035665655446292E-4)

(September\_7,6.992322392263642E-4)

(2006,6.898971805754122E-4)

(Website,6.881019444902207E-4)

(Politics,6.863957253704088E-4)

(Species,6.824024630839779E-4)

(Nation,6.823819443286571E-4)

(Fish,6.813394581587644E-4)

(Switzerland,6.79443883464455E-4)

(Mammal,6.710719757557075E-4)

(River,6.70707492686828E-4)

(Portugal,6.701114195095531E-4)

(Island,6.658661406278748E-4)

(World\_War\_II\_d045,6.606476271768632E-4)

(Population,6.533076791290544E-4)

Hadoop:

1 United\_States\_09d4 0.004877659328716546

2 Wikimedia\_Commons\_7b57 0.004548605046026623

3 Country 0.0036664731979925377

4 England 0.002612821605337235

5 Water 0.002489009614855558

6 City 0.002393955886536013

7 Animal 0.0023867285000744695

8 Germany 0.0023065234450352027

9 United\_Kingdom\_5ad7 0.002205212151302254

10 France 0.002201176369359446

11 Earth 0.0021462691802845435

12 Europe 0.0019101020431913159

13 Wiktionary 0.0016531374206027313

14 English\_language 0.0016378556210463722

15 Government 0.001612162780236346

16 Computer 0.0016078804188372313

17 India 0.0015986553959675541

18 Money 0.001554646146935443

19 Japan 0.0014629740906092884

20 Plant 0.001427549631058239

21 Italy 0.0014184817592084835

22 Spain 0.0014055616975281748

23 Canada 0.0013890326905440999

24 Food 0.001329339303368923

25 Human 0.0013197189487116415

26 China 0.0013072419396927585

27 People 0.001294128559112034

28 Australia 0.0012479429412974093

29 Capital\_(city) 0.0011978128884340203

30 Television 0.0011959246331379601

31 Asia 0.0011940189323182758

32 Sun 0.001157739790591706

33 Number 0.0011548425386711276

34 State 0.0011539713078695577

35 Mathematics 0.0011528846193809034

36 Sound 0.0011514321020817905

37 Science 0.001141188424615729

38 2004 0.0011085933049208103

39 Metal 0.0011084421171927866

40 Year 0.0010880647155164136

41 Language 0.001082213154693999

42 Russia 0.0010713547719380738

43 Wikipedia 0.0010450864918915208

44 Religion 0.0010321692448856845

45 Music 0.0010307565446426865

46 19th\_century 0.0010275663342868458

47 Scotland 9.892541847845327E-4

48 Greece 9.870644544316615E-4

49 20th\_century 9.860164656035205E-4

50 London 9.740482503427417E-4

51 Latin 9.629096767805106E-4

52 Greek\_language 9.359512400327889E-4

53 Energy 9.207607736013204E-4

54 World 9.173722647261367E-4

55 Centuries 9.1419630417482E-4

56 Culture 8.75951041252504E-4

57 History 8.714336901480959E-4

58 Netherlands 8.588987608445802E-4

59 Liquid 8.437455752801664E-4

60 Inhabitant 8.381218195350012E-4

61 Society 8.334535378532852E-4

62 Light 8.327947308996996E-4

63 Planet 8.304916459047852E-4

64 Image 8.30250633461785E-4

65 Wikimedia\_Foundation\_83d9 8.289652702928677E-4

66 Law 8.266745510208703E-4

67 Scientist 8.255281111447872E-4

68 List\_of\_decades 8.218338159558719E-4

69 Atom 8.15902020461337E-4

70 Africa 8.12990764747871E-4

71 Geography 8.126170066529702E-4

72 Uniform\_Resource\_Locator\_1b4e 8.037279795283851E-4

73 Capital\_city 7.879212343142659E-4

74 Turkey 7.824086944597078E-4

75 Poland 7.783396139109577E-4

76 Plural 7.609862040282504E-4

77 Electricity 7.523689009350158E-4

78 Car 7.508454280164442E-4

79 Book 7.482916126820497E-4

80 Building 7.443733891720991E-4

81 Sweden 7.424127002023986E-4

82 Biology 7.319116855481665E-4

83 God 7.293376885819988E-4

84 War 7.227829512926038E-4

85 Chemical\_element 7.128927902391862E-4

86 North\_America\_e7c4 7.075020607433735E-4

87 September\_7 7.031269600216752E-4

88 2006 6.946274640579173E-4

89 Fish 6.932045692559188E-4

90 Website 6.924996074107898E-4

91 Politics 6.919862425884745E-4

92 Switzerland 6.873046043407955E-4

93 Nation 6.872640029815618E-4

94 Species 6.871872254036781E-4

95 River 6.769280893985056E-4

96 Mammal 6.752671197334419E-4

97 Portugal 6.732303015196399E-4

98 Island 6.68570102879043E-4

99 World\_War\_II\_d045 6.63763851007558E-4

100 Gas 6.568474440212067E-4

**Conclusion:**

Spark and Hadoop execution results are not exactly same but there are almost equal for the top 10 records.

The reason could be how sum of dangling node is handled in the spark program compared to global counter in the Hadoop job.