CS6240 HW4- Anvita Surapaneni

Design Discussion:

```
val p: BZ2Parser = new BZ2Parser
var input = sc.textFile(args(0)).map(l => p.parse123(l)).persist()
```

- p is an instanceof BZ2Parser
  parse123 is a method of BZ2parse that parses one line of input and gives the op as URL followed by its outlinks separated by tab. If the input does not match any pattern, it gives "dummy"
- It is taken as var because input will be updated in the next step to filter out "dummy" values
- This RDD will contain Array[String] because the map returns one line for each line in the RDD sc.textFile(args(0)) which has Array of each line from the input file
- sc.textFile(args(0)) is Resilient Distributed Dataset (RDD): Array[String] of lines from input file
- map performs a function on each item of input. Here, we call parse123 function
- persist Store RDD as deserialized Java objects in the JVM and reuses them in other actions on that dataset

```
input = input.filter(line => line != "dummy")
```

- Input is reassigned to only valid array items by filtering only lines that do not have "dummy"
- filter removes items from input which match the condition.

```
val pagecount = input.count().toDouble
```

- Pagecount is taken as val because pagecount does not change.
- It is the count of all pages in the graph.count gives Count of all items of input RDD.

```
val initialPageRank = 1.0/pagecount
```

- InitialPageRank is 1/no of pages in graph
- It is taken as val because its value is not a variable, it is a constant

```
var pagerank = input.map(line => line.split("\t")).
keyBy(line => line(0)).
mapValues(line =>  (initialPageRank, line(1).trim().substring(1, line(1).length() - 1).trim().split(",")))
```

- Pagerank is taken as var because it has to be re-initialised in each iteration of the pagerank
- RDD of form: Array[(String-URL, (Double-PageRank, Array[String]-OutLinks List))]
- For each line in input, we do a map by applying a function split("\t") on each item in list
- This will give us 2 strings, one-the URL, two-The outlinks [ol1, ol2, …]
- keyBy sets the key to the RDD. The key for pagerank is chosen to be the URL, mapValues sets the values for RDD. Value is chosen to be a multivalued value which contains the pagerank(for initial iteration, it will be the initial pagerank) and the Array of outlinks which is achieved by removing the [and] from the line(2) which is the string of outlinks and splitting on ","

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

- Iterate 10 times

```
var DandlingNodesPRsum = pagerank.filter(line => line._2._2.forall(_.isEmpty)).map(_._2._1).sum()
```

- Handling Dangling Nodes. DandlingNodesPRsum is the sum of pageranks for dangling nodes
- It is var because its value has to be recomputed for each iteration
- The filter gives only items of pagerank that do not have outlinks, i.e. the Array of out links at position

_2._2 is empty.

- *Map creates RDD using the pagerank value which is at position _2._1, the sum sums up all the items of RDD, i.e.pageranks*

```
var DanglingNodePRdist = DandlingNodesPRsum/pagecount
```

- Handling Dangling Nodes. DanglingNodePRdist is the pagerank distribution that has to be added to each node's pagerank to handle dangling nodes
- Divide DandlingNodesPRsum by pagecount
- It is var because its value has to be recomputed for each iteration

```
var paagerank1 = pagerank.flatMap{
         l => var PRdist = l._2._1/l._2._2.length
        l._2._2.map(line => (line.trim(), PRdist))}
```

- paagerank1 is a temporary RDD and is var because it needs to be reinitialized in every iteration
- flatMap Applies a function that returns an iterator to each value of a pair RDD.
- Here, we use flatMap to return a pair RDD of form (URL, Pagerank distribution) for every outlink of a URL in pagerank RDD
- PRdist will store the pagerank distribution for all the outlinks of a particular URL(claicultaed by dividing the pagerank of the URL(l._2._1) by the length of the outlinks(l._2._2.length))
- l._2._2.map(line => (line.trim(), PRdist)) : for every outlink of a URL in pagerank, return (outlink, caliculated pagerank distribution for that)

```
var paagerank2 = paagerank1.reduceByKey((x, y) => x + y).keyBy(l => l._1)mapValues(l => l._2)
```

- reduceByKey combines values with the same key. Here, we add all the values (pagerank distribution) with the same key i.e. same URl. keyBy sets key to URL and mapValues sets value to sum of pagerank distribution.

```
var prjoin1 = paagerank2.join(pagerank)
```

- join performs an inner join between two RDDs.
- Prjoin will contain an RDD of innerjoin of paagerank2 and pagerank
- Resultant RDD will be of the form: Array[(String, (Double-new total pagerank from paagerank2, (Double, Array[String])-pagerank's value(old pagerank, outlink list)))]

```
pagerank = prjoin1.map(l => l).keyBy(l => l._1).
mapValues(l => ((0.15 * 1/ pagecount) + (0.85 * l._2._1) + DanglingNodePRdist, l._2._2._2))
```

- Reinitialize pagerank by computing new pagerank values.
- (0.15 * 1/ pagecount) + (0.85 * l._2._1) is the new pagerank computation usng formula (alpha * 1/|v|) + (1-alpha) * total pagerank distribution for that URL
- we add the dangling node's pagerank distribution to each page's pagerank
- l._2._2._2 is the position of outlinks in the prjoin1

} – end of loop

```
val output = pagerank.map(l => (l._1, l._2._1))
```

- output: map on final pagerank which returns the (URL: l._1, pagerank: l._2._1)
- it is taken as val since its value is not a variable

```
val out = output.sortBy(_._2, false).take(100)
```

- sortBy sorts by the field _._2 of output RDD ie the pagerank, false makes the sort order to be descending. Take(100) takes the top 100 items of RDD.

```
sc.parallelize(out).repartition(1).sortBy(_._2, false).saveAsTextFile(args(1))
```

- upon parallelize, the distributed dataset can be operated in parallel.
- Repartition aways shuffles the data in RDD randomly to create given no of partitions and balance it across them. Here we have one because we want all 100 RDD items to be on the same op file
- Again call sortBy on the pagerank field because repartition reshuffles the order.
- The saveAsTextFile saves the RDD to an output file.

Compare the Hadoop Map reduce and Spark implementation of Page Rank.

```
val p: BZ2Parser = new BZ2Parser
var input = sc.textFile(args(0)).map(l => p.parse123(l)).persist()
```

- It is similar to the parse-mapper which would parse the line of the input file and return the url followed by list of outlinks separated by tab in the from of a string.

```
input = input.filter(line => line != "dummy")
```

- This task is performed in parsemapper where the invalied input lines are discarded

```
val pagecount = input.count().toDouble
```

- This task is done by counters ”NodesCount” in Hadoop which increment by one in parse mapper for each valid line
- This task is better performed in spark because one operation gives us the count, reduces overhead in incrementing count multiple times

```
val initialPageRank = 1.0/pagecount
```

- Initial pagerank is caliculated in input-mapper by doing 1/NodesCount
-

```
var pagerank = input.map(line => line.split("\t")).
keyBy(line => line(0)).
mapValues(line =>  (initialPageRank, line(1).trim().substring(1, line(1).length() - 1).trim().split(",")))
```

- This task is also done in input mapper using context.write which emits URL, class of pagerank, name, outlinks, in spark we just need to store pagerank and outlinks as values.
- This task is better performed in spark because, we don’t need to store other variables in value other than the outlinks and the pagerank

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

- Iterate 10 times
- Uses PRmapper and PR reducer for iteration

```
var DandlingNodesPRsum = pagerank.filter(line => line._2._2.forall(_.isEmpty)).map(_._2._1).sum()
```

- This value is caliculated in PRreducer by extracting the counter value of
- DanglingTotalPR counter of previous iteration. Its value is also updated in PRreducer by incrementing the counter by the pgerank of dangling nodes(whose key is set to 1)
- Sum operation of spark is easier to implement than the counters in spark

```
var DanglingNodePRdist = DandlingNodesPRsum/pagecount
```

- Caliculated in PRreducer by dividing DanglingTotalPR counter by NodesCount counter

```
var paagerank1 = pagerank.flatMap{
         l => var PRdist = l._2._1/l._2._2.length
         l._2._2.map(line => (line.trim(), PRdist))}
```

- This task is done by PRmapper which caliculates pagerank distribution for each out link and emits outlink URL and object with its pagerank distribution, name and empty outlinks value

```
var paagerank2 = paagerank1.reduceByKey((x, y) => x + y).keyBy(l => l._1)mapValues(l => l._2)
var prjoin1 = paagerank2.join(pagerank)
pagerank = prjoin1.map(l => l).keyBy(l => l._1).
mapValues(l => ((0.15 * 1/ pagecount) + (0.85 * l._2._1) + DanglingNodePRdist, l._2._2._2))
```

- These tasks are done in PRreducer where for a given key we add the pagerank distributions, extrach outlinks based on if it's a parent node and caliculate pagerank and return url, pagerank and outlinks for that key/url
- Spark implementation of this task is better because a great deal of computation is performed in a single step, and its functionality can be interpreted by looking at the statement

} – stop ierating on PRmapper and PRreducer jobs

```
val output = pagerank.map(l => (l._1, l._2._1))
```

- This is similar to the final PRreducer
- Simple manipulation of RDD does this job in spark

```
val out = output.sortBy(_._2, false).take(100)
sc.parallelize(out).repartition(1).sortBy(_._2, false).saveAsTextFile(args(1))
```

- This task is tome by the sortReducer ehich sorts the entries based on pagerank and emits only the top 100 pages
- In spark function like sortBy and take perform this task easily.

Performance Comparisons

Machine/Time	Spark Execution	Hadoop MR execution
6 M4.large	14:45:55 -16:04:13= 1:18:18	00:00:08 – 00:53:23= 53:26
11 M4.large	18:20:35-19:02:29= 41:54	00:01:32 - 00:36:14 = 35:27

The spark version is slightly slower thank the Hadoop execution.

The parse function is called on each and every line of the input. This takes up a majority of the time about 1hr 10 mins for the 6 machines and 27 mins for 11 machine execution which is higher than the input parse time for Hadoop execution.
It could be because the M4.large machine can not hold the entire data in the input file when we give persist(). This might lead to extra time in recompilation.

Hadoop simple

Spark Simple

URL-Hadoop-Simple	PageRank	URL-Spark-Simple	Pagerank
United_States_09d4	0.003911085	United_States_09d4	0.004656847
Wikimedia_Commons_7b57	0.003864404	Wikimedia_Commons_7b57	0.003865091
Country	0.002367578	Country	0.003157114
Animal	0.00215252	Europe	0.002126678
England	0.002041629	Water	0.002089962
Water	0.001985125	United_Kingdom_5ad7	0.002086062
Germany	0.001931787	England	0.002050873
City	0.001923488	France	0.002036448
Europe	0.001519932	Germany	0.002003035
France	0.001483443	Earth	0.001999935
United_Kingdom_5ad7	0.001473907	Animal	0.001993579
Earth	0.001433041	Week	0.001753298
India	0.001393748	City	0.001747556
English_language	0.001384157	Sunday	0.00161487
Computer	0.001376926	Monday	0.001588987
Human	0.001374173	Asia	0.001582022
Plant	0.001353901	Wednesday	0.001573679
Music	0.001187505	Friday	0.001536117
Money	0.001172039	Saturday	0.001518927
Television	0.001165258	Thursday	0.001499027
Food	0.001162194	Tuesday	0.001488303
Number	0.00116001	Money	0.001476236
Wiktionary	0.001146038	Wiktionary	0.001469604
Australia	0.001084805	Plant	0.001442022
Spain	0.001073383	Italy	0.001392702
People	0.001069536	Computer	0.0013908
Government	0.001053871	English_language	0.001389943
Wikimedia_Foundation_83d9	0.00105373	India	0.001350973
Japan	0.001030822	Number	0.001335897
Inhabitant	0.001020993	Government	0.001329986
Italy	0.00098478	Day	0.001278938
Metal	0.000974766	Spain	0.001236642
Mathematics	0.000965428	People	0.001161285
China	0.00096342	Human	0.001160644
State	0.000954979	Japan	0.001159393
Asia	0.000951143	Wikimedia_Foundation_83d9	0.001124492
Canada	0.000902867	Energy	0.001110455
Sound	0.000893355	index	0.001109715
Species	0.000840677	Canada	0.001099798
Language	0.000830162	China	0.001088
Religion	0.000793617	Sun	0.001080232
Capital_(city)	0.000792631	Science	0.001060564

Greek_language	0.000791297	Food	0.001055543
2004	0.000787772	Mathematics	0.001035222
Fish	0.00076547	Australia	0.001032383
Liquid	0.000764962	Year	9.60E-04
Science	0.000762227	Russia	9.58E-04
Atom	0.000757051	Television	9.50E-04
Law	0.000748992	Music	9.13E-04
Week	0.00071828	Language	9.06E-04
Plural	0.000712348	Capital_city	8.89E-04
Light	0.000707322	Metal	8.79E-04
Greece	0.000706374	Wikipedia	8.76E-04
Disease	0.000702193	State	8.76E-04
Wikispecies	0.000701813	Greek_language	8.62E-04
Africa	0.000697413	Planet	8.62E-04
Wikipedia	0.000695025	2004	8.59E-04
Greek_mythology	0.000692057	Religion	8.48E-04
Image	0.000690506	Sound	8.38E-04
Energy	0.000687612	Scotland	8.24E-04
Chemical_element	0.000687385	London	8.20E-04
Mammal	0.000687255	Africa	8.20E-04
Sun	0.000677184	Greece	8.11E-04
Biology	0.000671066	20th_century	7.87E-04
Society	0.000666427	Geography	7.73E-04
London	0.00064914	Liquid	7.64E-04
Child	0.000639533	Law	7.64E-04
Scotland	0.000632709	19th_century	7.63E-04
God	0.000631957	World	7.54E-04
Uniform_Resource_Locator_1b4e	0.000613379	Poland	7.46E-04
Russia	0.000612099	Society	7.44E-04
Building	0.000608926	Scientist	7.43E-04
Car	0.000608817	Atom	7.37E-04
Female	0.00059945	Light	7.10E-04
Male	0.00059859	History	7.08E-04
Website	0.000591476	Latin	7.07E-04
River	0.000584581	War	7.00E-04
Netherlands	0.000578584	Culture	6.92E-04
Book	0.000575272	God	6.87E-04
World	0.000571852	Netherlands	6.82E-04
Planet	0.000568603	Turkey	6.78E-04
Capital_city	0.000568423	Building	6.73E-04
Day	0.000565643	Chemical_element	6.70E-04
Brazil	0.000565329	Plural	6.70E-04
Latin	0.000562765	Centuries	6.70E-04
Electricity	0.000556591	Sweden	6.67E-04
North_America_e7c4	0.000554501	Information	6.62E-04

Ocean	0.000553972	Portugal	6.48E-04
Bird	0.000543583	Denmark	6.32E-04
Mineral	0.000540388	Austria	6.30E-04
Year	0.000536268	Cyprus	6.27E-04
Organism	0.00053488	Disease	6.21E-04
Movie	0.000529349	Ocean	6.17E-04
Gas	0.000526982	Species	6.16E-04
Video	0.000520031	Moon	6.15E-04
Solid	0.000513917	Biology	6.08E-04
Sunday	0.000511775	Capital_city	5.97E-04
U.S._state_5a68	0.00050864	List_of_decades	5.94E-04
Monday	0.000507583	North_America_e7c4	5.94E-04
University	0.000505668	Electricity	5.94E-04

The values and pages are almost the same, the variation could be because of loss of precession in pageranks of dangling nodes which have been caliculated by using counters in the map reduce approach

Hadoop full dataset **Spark Full dataset**

URL-Hadoop-Full	Pagerank	URL- spark-Full	page rank
United_States_09d4	0.001275723	United_States_094	0.001460519
Biography	0.000870649	2006	0.001407527
2006	0.000622371	United_Kingdom_5ad7	7.68E-04
United_Kingdom_5ad7	0.000578397	2005	6.50E-04
Geographic_coordinate_system	0.000468929	Biography	5.08E-04
England	0.000446513	England	4.83E-04
Canada	0.000428644	France	4.61E-04
France	0.000386809	Canada	4.55E-04
2005	0.000347386	2004	4.54E-04
Germany	0.000346009	Germany	4.15E-04
Australia	0.000344369	Australia	3.82E-04
India	0.000343791	India	3.58E-04
Football_(soccer)	0.00028204	2003	3.53E-04
2004	0.000276218	Japan	3.33E-04
Japan	0.000268112	Internet_Movie_Database_7ea7	3.12E-04
Personal_name	0.000267786	Italy	3.06E-04
Politician	0.000248188	2002	2.95E-04
2003	0.0002351	2001	2.84E-04
Europe	0.000228755	Europe	2.76E-04

Record_label	0.000223799	London	2.74E-04
Population_density	0.00022179	Record_label	2.69E-04
Italy	0.000206737	World_War_II_d045	2.64E-04
2001	0.00020063	2000	2.55E-04
Spain	0.000197726	English_language	2.52E-04
Internet_Movie_Database_7ea7	0.000195165	1999	2.46E-04
Television	0.000188279	Wiktionary	2.35E-04
Sweden	0.000181103	Russia	2.30E-04
Scientific_classification	0.000180911	Music_genre	2.24E-04
2002	0.000180341	Geographic_coordinate_system	2.23E-04
2000	0.000179618	Spain	2.23E-04
London	0.000179417	Wikimedia_Commons_7b57	2.16E-04
Music_genre	0.000177268	1998	2.15E-04
Scotland	0.000169289	1997	2.06E-04
World_War_II_d045	0.000168471	Television	1.96E-04
Wiktionary	0.000164447	Scotland	1.93E-04
Norway	0.000163458	1996	1.91E-04
Public_domain	0.000163171	New_York_City_1428	1.89E-04
Census	0.000162602	Football_soccer	1.87E-04
Actor	0.000161958	1995	1.83E-04
Russia	0.000160743	China	1.77E-04
1999	0.000154304	1994	1.74E-04
Race_(United_States_Census)_a07d	0.000151044	Netherlands	1.73E-04
Square_mile	0.000150222	Sweden	1.71E-04
Per_capita_income	0.000148572	Scientific_classification	1.70E-04
Album	0.000145515	New_Zealand_2311	1.68E-04
Poland	1.45E-04	1991	1.65E-04
New_Zealand_2311	1.45E-04	1993	1.64E-04
Marriage	1.42E-04	Film	1.62E-04
Brazil	1.41E-04	Actor	1.60E-04
Km^2	1.40E-04	1990	1.59E-04
United_States_Census_Bureau_2c85	1.37E-04	Public_domain	1.58E-04
Film	1.36E-04	1992	1.58E-04
Poverty_line	1.35E-04	California	1.57E-04
1998	1.33E-04	1989	1.47E-04
English_language	1.30E-04	Latin	1.47E-04
China	1.29E-04	Ireland	1.44E-04
California	1.29E-04	1980	1.42E-04
Ireland	1.26E-04	Album	1.42E-04
White_(U.S._Census)_c45a	1.25E-04	Record_producer	1.42E-04
Animal	1.24E-04	1986	1.40E-04

1997	1.22E-04	January_1	1.39E-04
Writer	1.22E-04	1985	1.37E-04
Mexico	1.18E-04	1982	1.36E-04
Studio_album	1.17E-04	1979	1.36E-04
Corporation	1.17E-04	1981	1.35E-04
Poet	1.16E-04	1984	1.34E-04
Netherlands	1.16E-04	New_York_3da4	1.34E-04
1996	1.15E-04	1987	1.34E-04
Record_producer	1.14E-04	French_language	1.33E-04
Population	1.14E-04	1983	1.33E-04
New_York_City_1428	1.13E-04	1974	1.32E-04
School	1.10E-04	Poland	1.32E-04
New_York_3da4	1.09E-04	Animal	1.32E-04
Romania	1.09E-04	Norway	1.30E-04
Building	1.08E-04	1988	1.30E-04
1995	1.05E-04	1976	1.28E-04
Hispanic_(U.S._Census)_1387	1.03E-04	1970	1.28E-04
Latino_(U.S._Census)_5f0e	1.03E-04	Paris	1.28E-04
1994	1.03E-04	1975	1.27E-04
South_Africa_1287	1.03E-04	South_Africa_1287	1.27E-04
1990	9.97E-05	Soviet_Union_ad1f	1.26E-04
1993	9.93E-05	1969	1.25E-04
Musician	9.88E-05	Mexico	1.25E-04
Switzerland	9.77E-05	1972	1.25E-04
British_Columbia_90ec	9.62E-05	Studio_album	1.25E-04
1992	9.61E-05	1977	1.24E-04
Company_(law)	9.54E-05	1945	1.24E-04
Greece	9.49E-05	Brazil	1.24E-04
1991	9.29E-05	1978	1.24E-04
Paris	9.25E-05	Politician	1.22E-04
Portugal	9.16E-05	Greece	1.22E-04
Pakistan	9.15E-05	Switzerland	1.22E-04
Finland	9.07E-05	1973	1.21E-04
Native_American_(U.S._Census)_1a7a	9.04E-05	1971	1.19E-04
Iran	9.02E-05	1968	1.18E-04
1980	8.87E-05	Iran	1.18E-04
Wikimedia_Commons_7b57	8.78E-05	1967	1.18E-04
Denmark	8.77E-05	Pakistan	1.17E-04
1982	8.71E-05	Egypt	1.17E-04
1989	8.69E-05	World_War_I_9429	1.16E-04

The values and pages are almost the same, the variation could be because of loss of precession in pageranks of dangling nodes which have been caliculated by using counters in the map reduce approach