Massage Text Data and Word Occurrence using Mapreduce

Aravind Kumar[1], CH.Niveditha[2], T.Sandilya Kumar[3], A.Rajesh[4]
[1]Asst.prof, [2,3,4]Students
MLRIT,Dundigal

Abstract - These days most of the MNC's recruit the students by providing placements. Many MNC's are facing the drawback of vocabulary from employees working after recruiting them. So in order to overcome this limitation we came up with an idea to overcome this limitation facing by MNC's. We add a new round to the placements called essay writing or paragraph writing with the help of Big data analysis as the background filter to sort out the best desired student.

Keywords - BigData, MapReduce

I. INTRODUCTION

It manages the checking the general event of the word utilizing map decrease it goes under enormous data massaging it expels the superfluous one from it. In mapreduce it takes as key information and yield esteems. The word tally is useful at situation time to get a coveted understudy from the paper round For instance: if a writer needs to compose a base or most extreme measure of words for an article, exposition, report, story, book, paper, and so on. Word Counter will make the most of beyond any doubt its statement achieves a particular stage

STEP1: The word includes operation happens two phases a mapper stage and a reducer stage. In mapper stage first the test is tokenized into words then we frame a key esteem match with these words where the key being simply the word and esteem '1'.
For instance think about the sentence "tring the telephone rings"
<tring,1>
<tring,1>
<the,1>
<phone,1>
<rings,1>

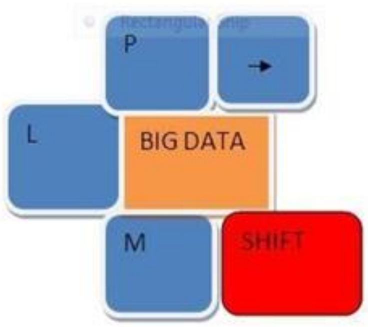


Figure 1

STEP2: In the diminish stage the keys are assembled together and the esteems for comparable keys are included.

- So here there are just a single match of comparative keys 'tring' the values for these keys would be included so the output key esteem sets would be

<tring,2>
<the,1>
<phone,1>
<rings,1>
In outline the sentence would be part as words and frame the underlying key esteem combine as this would give the quantity of event of each word in the info. Along these lines lessen frames a total stage for keys.

- It includes: Big Data
- It is a problem statement and it has: Hadoop.

It is an open source framework [1]

1. HDFS
2. Mapreduce

Challenges -

- Velocity
- Variety
- Volume
- Complexity

HDFS: Hadoop distributed file system it is used for storage
Mapreduce: It is used for processing the data. It takes the words in the form of key value in value out it first divides the message into words first it applies mapper phase and after it applies reduce phase.

II. METHODOLOGY

A. Technologies used -

i) HDFS: HDFS is an Apache Software Foundation venture and a subproject of the Apache Hadoop venture. Hadoop is perfect for putting away a lot of information, similar to terabytes and petabytes, and utilizations HDFS as its stockpiling framework. HDFS gives you a chance to interface hubs contained inside bunches over which information records are conveyed. You would then be able to access and store the information documents as one consistent record framework. Access to information documents is taken care of in a spilling way, implying that applications or orders are executed straightforwardly utilizing the MapReduce preparing model.
HDFS is blame tolerant and gives high-throughput access to substantial informational collections. This article

investigates the essential highlights of HDFS and gives an abnormal state perspective of the HDFS design.

Outline of HDFS - HDFS has numerous likenesses with other appropriated document frameworks, yet is distinctive in a few regards. One recognizable distinction is HDFS's compose once-perused many models that unwinds simultaneousness control necessities, improves information coherency, and empowers high-throughput get to.

Another one of a kind characteristic of HDFS is the perspective that it is typically better to find handling rationale close to the information instead of moving the information to the application space.

HDFS thoroughly confines information keeping in touch with one essayist at any given moment. Bytes are constantly added to the finish of a stream, and byte streams are ensured to be put away in the request composed.

HDFS Objectives - HDFS has numerous objectives. Here are probably the most outstanding:

- Fault resilience by recognizing issues and applying speedy, programmed recuperation
- Data get to by means of MapReduce spilling
- Simple and strong coherency display
- Processing rationale near the information, as opposed to the information near the preparing rationale
- Portability crosswise over heterogeneous product equipment and working frameworks
- Scalability to dependably store and process a lot of information
- Economy by circulating information and preparing crosswise over groups of product PCs
- Efficiency by circulating information and rationale to process it in parallel on hubs where information is found
- Reliability via naturally keeping up different duplicates of information and consequently redeploying handling rationale in case of disappointments

HDFS gives interfaces to applications to draw them nearer to where the information is situated, as portrayed in the accompanying area.

ii) MapReduce: MapReduce is a structure for preparing parallelizable issues crosswise over expansive datasets utilizing countless (hubs), on the whole alluded to as a bunch (if all hubs are on a similar nearby system and utilize comparative equipment) or a network (if the hubs are shared crosswise over topographically and authoritatively conveyed frameworks, and utilize more heterogeneous equipment). Handling can happen on information put away either in a file system (unstructured) or in a database (organized). MapReduce can exploit the region of information, preparing it close to the place it is put away with a specific end goal to limit correspondence overhead.

"Map" step: Each specialist hub applies the "guide()" capacity to the neighborhood information, and composes the yield to a brief stockpiling. An ace hub guarantees that just a single duplicate of excess information is prepared.

"Shuffle" step: Worker hubs redistribute information in view of the yield keys (delivered by the "guide()" work), with the end goal that all information having a place with one key is situated on a similar laborer hub.

"Reduce" step: Worker hubs now process each gathering of yield information, per key, in parallel.

MapReduce takes into account conveyed preparing of the guide and diminishment operations. Given that each mapping operation is free of the others, all maps can be performed in parallel – however practically speaking this is constrained by the quantity of autonomous information sources or potentially the quantity of CPUs close to each source. So also, an arrangement of 'reducers' can play out the decrease stage, gave that all yields of the guide operation that offer a similar key are introduced to a similar reducer in the meantime, or that the lessening capacity is cooperative. While this procedure can frequently seem wasteful contrasted with calculations that are more consecutive (on the grounds that various instead of one occasion of the decrease procedure must be run), MapReduce can be connected to essentially bigger datasets than "ware" servers can deal with – a vast server homestead can utilize MapReduce to sort a petabyte of information in just a couple of hours. The parallelism likewise offers some plausibility of recuperating from fractional disappointment of servers or capacity amid the operation: on the off chance that one mapper or reducer comes up short, the work can be rescheduled – expecting the info information is as yet accessible.

Another approach to take a gander at MapReduce is as a 5-step parallel and circulated calculation:

1. Prepare the Map() input - the "MapReduce framework" assigns Map processors, allocates the information key esteem K1 that every processor would chip away at, and furnishes that processor with all the information related with that key esteem.

2. Run the client gave Map() code - Map() is run precisely once for each K1 key esteem, producing yield composed by key esteems K2.

3. "Shuffle" the Map yield to the Reduce processors - The MapReduce framework assigns Reduce processors, doles out the K2 key esteem every processor should take a shot at, and gives that processor all the Map-produced information related with that key esteem.

4. Run the client gave Reduce() code - Reduce() is run precisely once for every K2 key esteem delivered by the Map step.

5. Produce the last yield - The MapReduce framework gathers all the Reduce yield, and sorts it by K2 to create the ultimate result.

These five stages can be consistently thought of as running in arrangement – each progression begins simply after the past advance is finished – in spite of the fact that by and by they can be interleaved as long as the last outcome isn't influenced. As a rule, the info information may as of now be disseminated among a wide range of servers, in which case stage 1 could once in a while be incredibly improved by doling out Map servers that would procedure the locally show input information. Likewise, stage 3 could some of the time be accelerated by allotting Reduce processors that are as close as conceivable to the Map-produced information they have to process.

Data Flow: The solidified piece of the MapReduce structure is a substantial disseminated sort. The problem areas, which the application characterizes, are:

- An input reader
- A Map function
- A segment function
- A think about function
- A Reduce function
- An output writer Input reader

The info peruser separates the contribution to fitting size 'parts' (practically speaking commonly 64 MB to 128 MB) and the system relegates one split to each Map work. The information reader reads information from stable stockpiling (regularly an appropriated document framework) and produces key/esteem sets.

A typical illustration will read a catalog loaded with content documents and restore each line as a record. Map function The Map work takes a progression of key/esteem sets, forms each, and creates at least zero yield key/esteem sets. The info and yield kinds of the guide can be (and regularly are) unique in relation to each other. On the off chance that the application is completing a word check, the guide capacity would break the line into words and yield a key/esteem combine for each word. Each yield combine would contain the word as the key and the quantity of occurrences of that word in the line as the esteem.

Partition function: Each Map work yield is dispensed to a specific reducer by the application's parcel work for sharing purposes. The segment work is given the key and the quantity of reducers and returns the file of the coveted reducer.

A run of the mill default is to hash the key and utilize the hash esteem modulo the quantity of reducers. It is critical to pick a segment work that gives a roughly uniform conveyance of information per share for stack adjusting purposes, generally the MapReduce operation can be held up sitting tight for ease back reducers to complete (i.e. the reducers appointed the bigger offers of the non-consistently divided information).

Between the guide and diminish stages, the information are rearranged (parallel-arranged/traded between hubs) so as to move the information from the guide hub that created them to the shard in which they will be lessened. The rearrange can now and again take longer than the calculation time contingent upon arrange transfer speed, CPU speeds, information created and time taken by outline decrease calculations.

Comparison function: The contribution for each Reduce is pulled from the machine where the Map ran and arranged utilizing the application's examination work. Reduce function:

The system calls the application's Reduce work once for every novel key in the arranged request. The Reduce can repeat through the qualities that are related with that key and create at least zero yields.

In the word check case, the Reduce work takes the info esteems, entireties them and produces a solitary yield of the word and the last entirety.

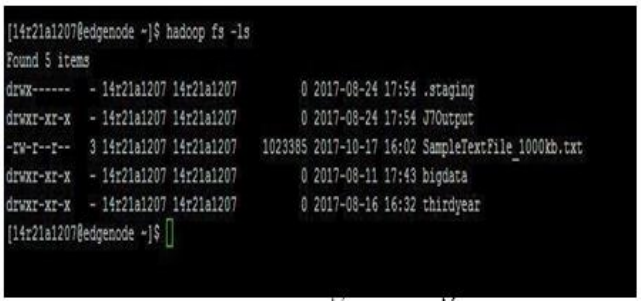
Output writer: The Output Writer composes the output of the Reduce to the steady stockpiling.

Dataset have to be loaded into the cluster: Here we took a sample data set of 1GB we worked on this by using the commands we have saved it on the cluster

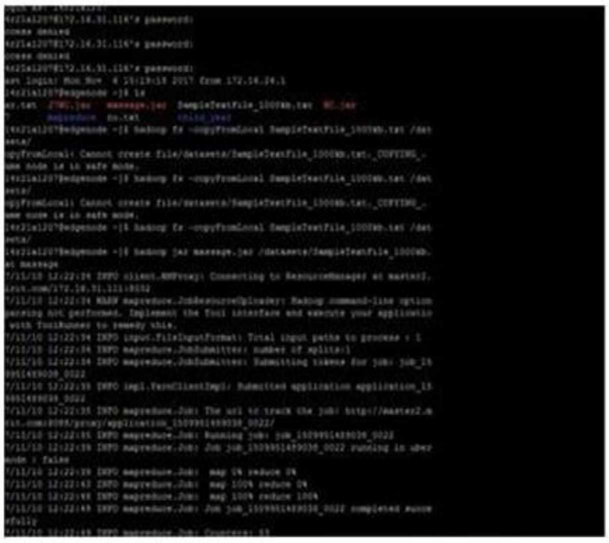
III. IMPLEMENTATION

Step1: Dataset loaded on to the cluster.
Step2: create jar file Word.jar third_year massage_data.txt
Step3: move jar file on to the cluster Hadoop fs_copy from local massage_data.txt
Step4: excute jar file Hadoop jar word.jar massagedata.txt word_op/
Step 4: Code has been implemented and it is saved

IV. RESULT

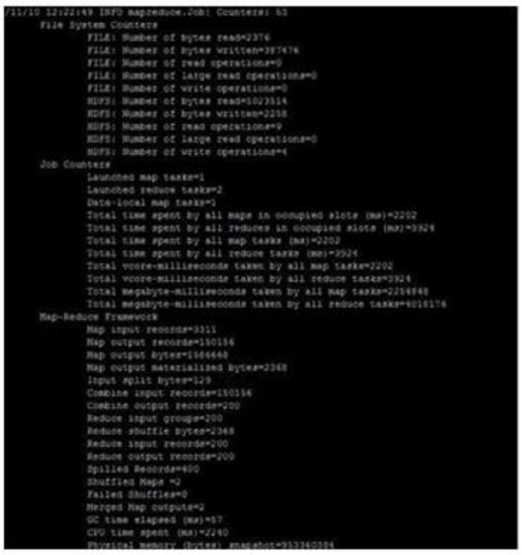


```
[14r21a1207@edgenode ~]$ hadoop fs -ls
Found 5 items
drwx------   - 14r21a1207 14r21a1207          0 2017-08-24 17:54 .staging
drwxr-xr-x   - 14r21a1207 14r21a1207          0 2017-08-24 17:54 J7Output
-rw-r--r--   3 14r21a1207 14r21a1207    1023385 2017-10-17 16:02 SampleTextFile_1000kb.txt
drwxr-xr-x   - 14r21a1207 14r21a1207          0 2017-08-11 17:43 bigdata
drwxr-xr-x   - 14r21a1207 14r21a1207          0 2017-08-16 16:32 thirdyear
[14r21a1207@edgenode ~]$
```



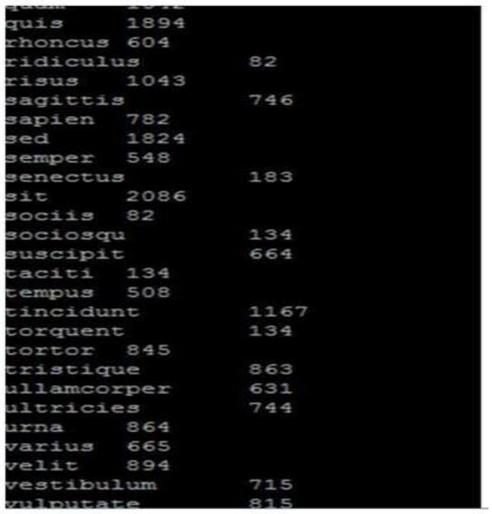
```
4r21a1207@172.16.31.116's password:
ccess denied
4r21a1207@172.16.31.116's password:
ccess denied
4r21a1207@172.16.31.116's password:
ast login: Mon Nov  6 15:19:18 2017 from 172.16.24.1
14r21a1207@edgenode ~]$ ls
ar.txt  J7WC.jar  massage.jar  SampleTextFile_1000kb.txt  WC.jar
?       Mapreduce  no.txt       third_year
14r21a1207@edgenode ~]$ hadoop fs -copyFromLocal SampleTextFile_1000kb.txt /dat
sets/
opyFromLocal: Cannot create file/datasets/SampleTextFile_1000kb.txt._COPYING_.
ame node is in safe mode.
14r21a1207@edgenode ~]$ hadoop fs -copyFromLocal SampleTextFile_1000kb.txt /dat
sets/
opyFromLocal: Cannot create file/datasets/SampleTextFile_1000kb.txt._COPYING_.
ame node is in safe mode.
14r21a1207@edgenode ~]$ hadoop fs -copyFromLocal SampleTextFile_1000kb.txt /dat
sets/
14r21a1207@edgenode ~]$ hadoop jar massage.jar /datasets/SampleTextFile_1000kb.
xt massage
7/11/10 12:22:34 INFO client.RMProxy: Connecting to ResourceManager at master2.
[illegible].com/172.16.31.111:8032
7/11/10 12:22:34 WARN mapreduce.JobResourceUploader: Hadoop command-line option
parsing not performed. Implement the Tool interface and execute your applicatio
 with ToolRunner to remedy this.
7/11/10 12:22:34 INFO input.FileInputFormat: Total input paths to process : 1
7/11/10 12:22:34 INFO mapreduce.JobSubmitter: number of splits:1
7/11/10 12:22:34 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_15
09951489038_0022
7/11/10 12:22:35 INFO impl.YarnClientImpl: Submitted application application_15
09951489038_0022
7/11/10 12:22:35 INFO mapreduce.Job: The url to track the job: http://master2.m
[illegible].com:8088/proxy/application_1509951489038_0022/
7/11/10 12:22:35 INFO mapreduce.Job: Running job: job_1509951489038_0022
7/11/10 12:22:39 INFO mapreduce.Job: Job job_1509951489038_0022 running in uber
mode : false
7/11/10 12:22:39 INFO mapreduce.Job:  map 0% reduce 0%
7/11/10 12:22:43 INFO mapreduce.Job:  map 100% reduce 0%
7/11/10 12:22:48 INFO mapreduce.Job:  map 100% reduce 100%
7/11/10 12:22:49 INFO mapreduce.Job: Job job_1509951489038_0022 completed succe
sfully
7/11/10 12:22:49 INFO mapreduce.Job: Counters: 49
```

Figure 2



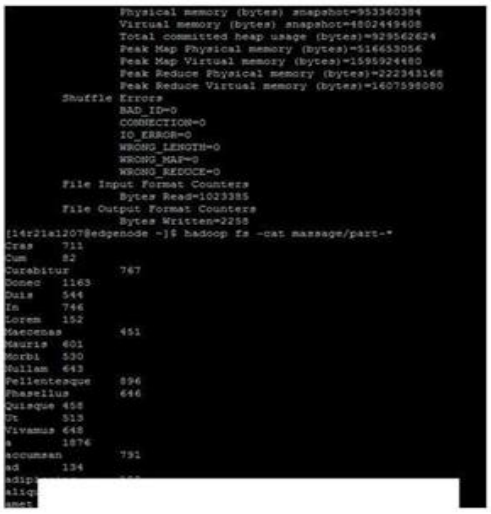
```
/11/10 12:22:49 INFO mapreduce.Job: Counters: 53
    File System Counters
            FILE: Number of bytes read=2376
            FILE: Number of bytes written=387476
            FILE: Number of read operations=0
            FILE: Number of large read operations=0
            FILE: Number of write operations=0
            HDFS: Number of bytes read=1023514
            HDFS: Number of bytes written=2258
            HDFS: Number of read operations=9
            HDFS: Number of large read operations=0
            HDFS: Number of write operations=4
    Job Counters
            Launched map tasks=1
            Launched reduce tasks=2
            Data-local map tasks=1
            Total time spent by all maps in occupied slots (ms)=2202
            Total time spent by all reduces in occupied slots (ms)=3924
            Total time spent by all map tasks (ms)=2202
            Total time spent by all reduce tasks (ms)=3924
            Total vcore-milliseconds taken by all map tasks=2202
            Total vcore-milliseconds taken by all reduce tasks=3924
            Total megabyte-milliseconds taken by all map tasks=2254848
            Total megabyte-milliseconds taken by all reduce tasks=4018176
    Map-Reduce Framework
            Map input records=3311
            Map output records=150156
            Map output bytes=1584648
            Map output materialized bytes=2368
            Input split bytes=129
            Combine input records=150156
            Combine output records=200
            Reduce input groups=200
            Reduce shuffle bytes=2368
            Reduce input records=200
            Reduce output records=200
            Spilled Records=400
            Shuffled Maps =2
            Failed Shuffles=0
            Merged Map outputs=2
            GC time elapsed (ms)=57
            CPU time spent (ms)=2240
            Physical memory (bytes) snapshot=953360384
```

Figure 3



```
quam      1842
quis      1894
rhoncus   604
ridiculus         82
risus     1043
sagittis          746
sapien    782
sed       1824
semper    548
senectus          183
sit       2086
sociis    82
sociosqu          134
suscipit          664
taciti    134
tempus    508
tincidunt         1167
torquent          134
tortor    845
tristique         863
ullamcorper       631
ultricies         744
urna      864
varius    665
velit     894
vestibulum        715
vulputate         815
```

Figure 4



```
                Physical memory (bytes) snapshot=953360384
                Virtual memory (bytes) snapshot=4802449408
                Total committed heap usage (bytes)=929562624
                Peak Map Physical memory (bytes)=516653056
                Peak Map Virtual memory (bytes)=1595924480
                Peak Reduce Physical memory (bytes)=222343168
                Peak Reduce Virtual memory (bytes)=1607598080
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1023385
        File Output Format Counters
                Bytes Written=2258
[14r21a1207@edgenode ~]$ hadoop fs -cat massage/part-*
Cras    711
Cum     82
Curabitur       767
Donec   1163
Duis    544
In      746
Lorem   152
Maecenas        451
Mauris  601
Morbi   530
Nullam  643
Pellentesque    896
Phasellus       646
Quisque 458
Ut      513
Vivamus 648
a       1876
accumsan        791
ad      134
adipi
aliq
amet
```

Figure 5

V. CONCLUSION

- Intended to encourage and improve the handling of immense measures of information in parallel on expansive groups of ware equipment in a dependable, fault- tolerant manner.
- Computational handling happens on both
- Unstructured data: document framework
- Organized data: database.

VI. REFERENCES

[1]. https://en.wikipedia.org/wiki/Bigdata

[2]. https://www.tutorialspoint.com/hadoop/hadoop_hadoop_overview.htm/

[3]. https://hadoop.apache.org/

[4]. https://en.wikipedia.org/wiki/MapReduce