Homework 6
Introduction to Big Data Systems

N Y Hendrikx < 2019403077 >

April 21, 2019

1 Intro

This report contains the details of the homework 6. The task is to implement k-means in spark.

2 Implementation

2.1 Pre-processing

Part of the preprocessing i did in a separate python file 'pre_processing.py'. This was only to find the number of possibilities fot the features 1,2,3 and 42. The results for this was:

$[[tcp, udp, icmp], [http, smtp, domain_u, auth, finger, telnet, eco_i, ftp,$
$ntp_u, ecr_i, other,$
$urp_i, private, pop_3, ftp_data, netstat, daytime, ssh, echo, time, name, whois, domain, mtp,$
$gopher, remote_job, rje, ctf,$
$supdup, link, systat, discard, X11, shell, login, imap4, nntp, uucp, pm_dump, IRC, Z39_50,$
$netbios_dgm, ldap, sunrpc, courier, exec, bgp, csnet_ns, http_443, klogin, printer, netbios_ssn,$
$pop_2, nnsp, efs, hostnames, uucp_path,$
$sql_net, vmnet, iso_tsap,$
$netbios_ns, kshell, urh_i, http_2784, harvest, aol, tftp_u, http_8001, tim_i, red_i]$
$, [SF, S2, S1, S3, OTH, REJ, RSTO, S0, RSTR, RSTOS0, SH]$
$, [normal., buffer_overflow., loadmodule., perl., neptune., smurf.,$
$guess_passwd., pod., teardrop., portsweep., ipsweep., land., ftp_write., back., imap., satan.,$
$phf., nmap., multihop., warezmaster., warezclient., spy., rootkit.]].$

So every feature is transformed to a number. tcp = 0, udp = 1, icmp = 2. and htpp = 0 etc.

```
word_index = [1, 2, 3, 41]
categories = [[], [], [], []]
```

```
for i in l:
    for w in word_index:
        if(i[w] in categories[get_idx(w)]):
            continue
        else:
            categories[get_idx(w)].append(i[w])
```

2.2 Algorithm

The are multiple ways to find the k-means. I choose to use the 'Lloyd's algorithm' which is considered default. For details https://en.wikipedia.org/wiki/Lloyd%27s_algorithm.

From using the search engine well I found out that map and reduce functions will be parallelized in spark. The main implementation of this program is just 3 lines.

```
    #1st get closest center for each point
    closest = data.map( lambda pnt: (get_closest_pnt(pnt, random_pnts),
        (pnt, 1)) )
    #2nd reduce, output = 23 [center_i, (total pnt, #pnts)],
    reduce = closest.reduceByKey( lambda value1, value2: (value1[0] +
        value2[0], value1[1] + value2[1]))
    #3th calulate avg pnt of cluster, returns (center, avg)
    avg_pnts = reduce.map( lambda tup: (tup[0], tup[1][0] / tup[1][1])
        ).collect()
```

1. Variable pnt is an array of vectors which are actually the 43 features. get_closest point returns the center which is the closest to the current point. Map iterates over all the points. So it returns a list of tuples: (center, (pnt, 1)).
2. closest.reduceByKey takes this list and concatenates the values for where the key is the same. In this lambda expression it simply adds the values of the features together and does the same for the amount of point. The total results is a summation of the values of all the features of points for some center x with the amount of points it contains.
3. avg_point just takes the average value of the output closest.reduceByKey.

After this the new random points/ centers are calculated and this process keeps repeating until there is enough convergence.

```
    #calculate total distance between new and old center..
    tmp_dist = 0
    for t in avg_pnts:
        tmp_dist = tmp_dist + (np.sum((random_pnts[t[0]] - t[1]) ** 2))
            #new distance
        random_pnts[t[0]] = t[1] #new random points/centers)
```

2.3 Experiments

I used once the big data set with (8 gb 12 Cores, with an convergence constant of 0.5.). But this took to long so I changed the other experiments for comparison to the 10 percent dataset. The convergence constant is a measure of how close old and new centers are to each other. If close enough the search is stopped. Whit a high convergence constant it will stop searching more early.

The following setups where used: Memory: 1,2,4,8. Cores: 1,2,4,8,16,32. Where a convergence constant of 0.5 is used. The only problem is that I couldn't monitor the time elapsed of the experiments, because the internet connection wasn't stable and the i couldn't read the output. I did not manage to access the logs of the servers. And I implemented a python function but it seemed to be very inaccurate.

3 Results

See attached text files for results of centers, distance per center and details. In the name you can find the amount of gigabytes and cores. But in the tables and in this report display the most important data. I couldn't fit all the data readable on a page without losing much data. Per experiment there is a separate file. Example: results_10_4g_2c means: dataset 10 percent, 4 gb memory and 2 cores.

Figure 1: Performance 4gb vs 8Gb

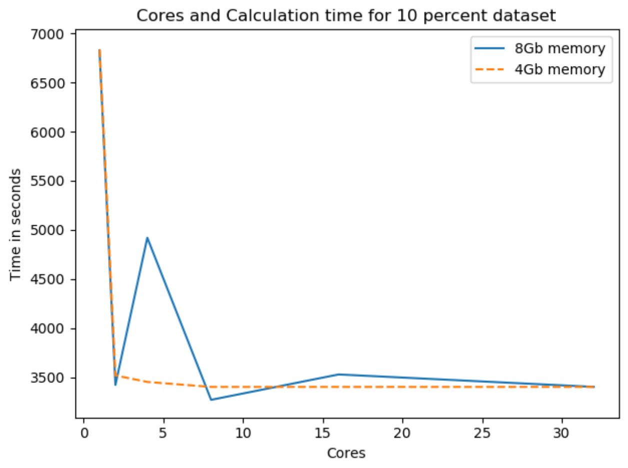

Table 1: results in seconds

Memory/ Cores	1	2	4	8	16	32
1gb	6661.39	-				
2gb	6691.39	3523.11	-			
4gb	6860.88	3516.74	3450.78	-	-	-
8gb	6829.035	3420.21	4919.61	3268.42	3527.10	3400.51

The Distance squared for each center is:

1. 6458172817507.836
2. 2117099264.698754
3. 4299204760.126798
4. 76778595565.05301
5. 4370365731.875955
6. 0.0
7. 75550243750.14536
8. 670574088174.7589
9. 25856257016.352932
10. 485719032467.1763
11. 93390050799.25665
12. 25161088005.804234
13. 12876555202.762737
14. 32226759944.750675
15. 2852795736585.315
16. 20014092202.98051
17. 74546493.92595416
18. 262611832364.3508
19. 141803458967338.88
20. 24910516279.909824
21. 42613362.67971128
22. 54594989067.12513
23. 4791925367.372628

4 Conclusion

I was surprised how bad this application/implementation scales. There seems to be no additional performance when adding more cores after 2 cores. Where 2 cores perform twice as good as 1 core. Also 1Gb memory seems to be enough. The implementation does not seem to use more then that. This I conclude from the fact that the 1Gb 1 core is as fast as the 4Gb or 8Gb 1 core memory experiments.