

CS/DS 4433 - Big Data Management & Analytics Project 2 - Report

Data Set Preparation

The team used data from Project 1 to test Apache Pig solutions. We created a project folder and uploaded generated data into the Hadoop system using the following commands:

```
hdfs dfs -mkdir /project2
```

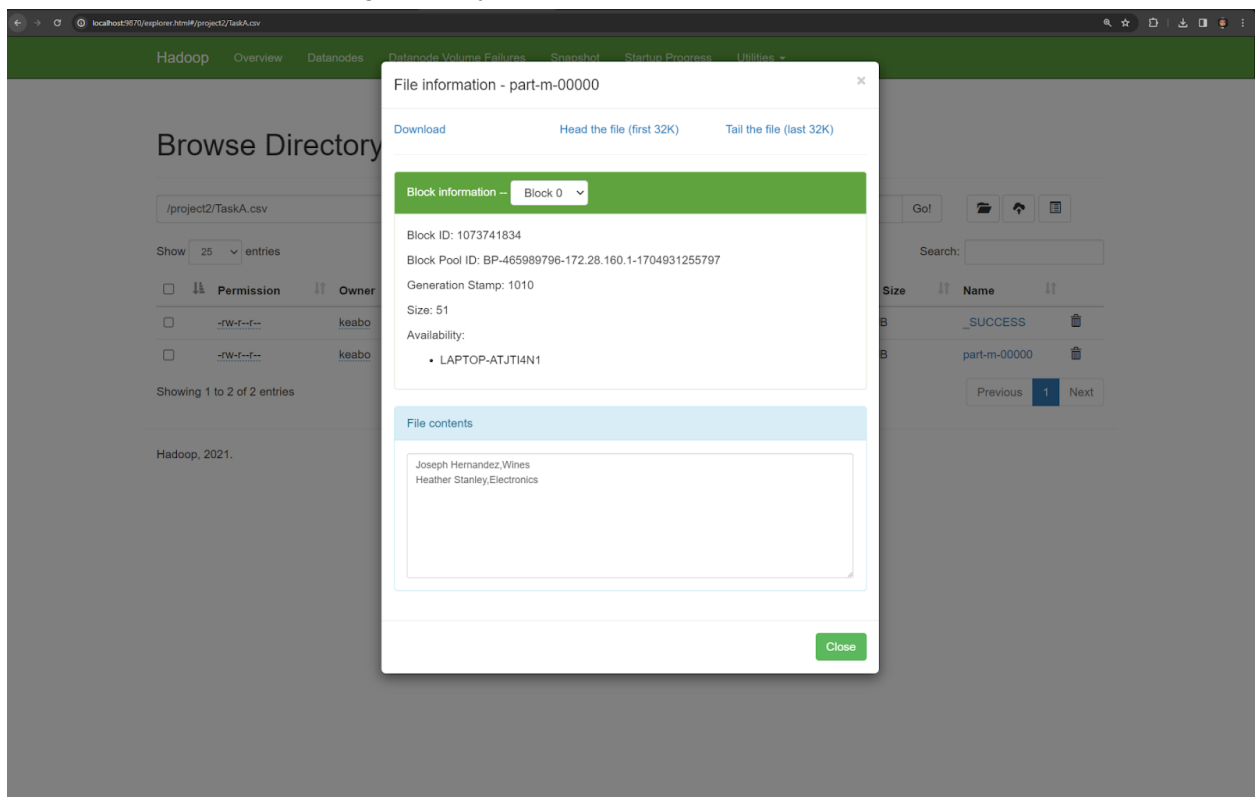
```
hdfs dfs -put access_logs.csv friends.csv pages.csv /project2/
```

Our team tested the solutions locally and then switched to HDFS paths using IntelliJ. Please make sure to put the files such that they are accessible on localhost:9000 on the hdfs.

Analytics Queries Logistics

1. Task A

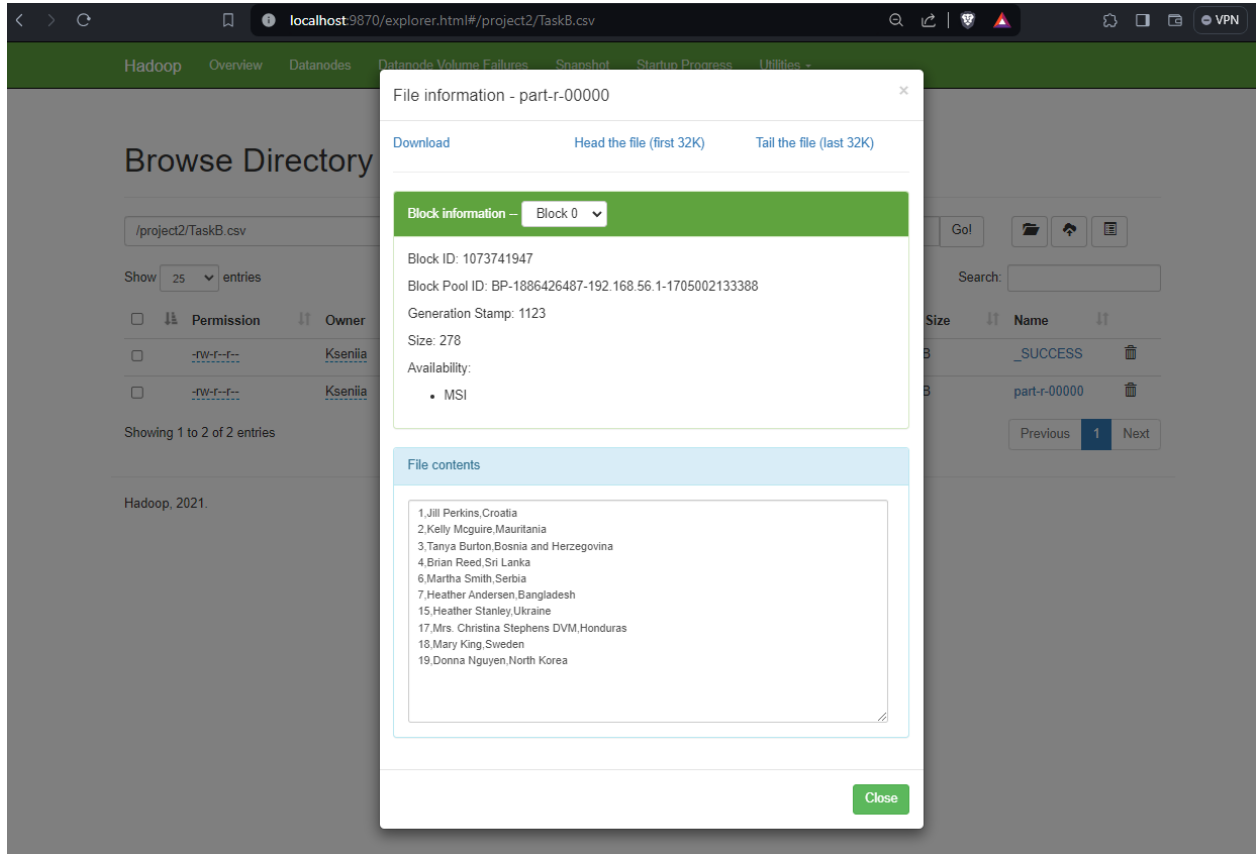
Task A was accomplished in Apache Pig by first loading the pages.csv file from HDFS. This data is then filtered so that only data loaded into the attribute “nationality” with the value “Ukraine” are returned. From this new filtered data set, we want to select (using foreach + generate) a new dataset that is the Name and Hobbies of each tuple in the nationality dataset. This is then stored back in HDFS, labeled TaskA.csv. Using pig -x taskA.pig, the file takes 3.070 to complete. Compared to the 2.372 seconds to run the Java map function, this is significantly slower.



2. Task B

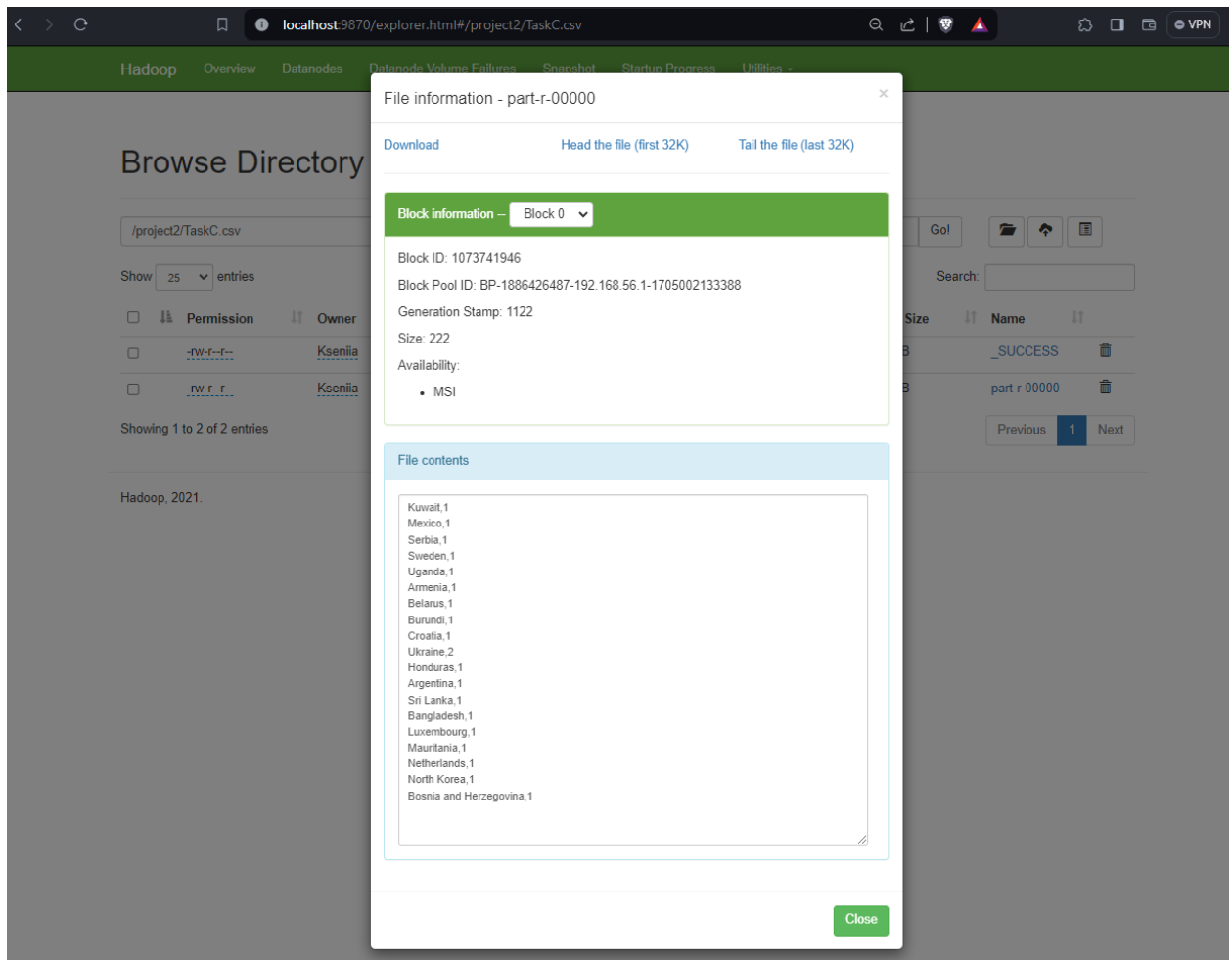
Task B was accomplished by first loading in the access_logs.csv & pages.csv file from HDFS into variables. Pages are cleaned to only be the personid, nationality, and name of a page. The access logs are grouped on What Page was accessed. Then we

count the amount of times a page is accessed. After that we order this list in descending order, limit the length to only top 10, and join this data on the page information to output the appropriate page information we need. This task takes 6.414 seconds to complete. The output is stored in HDFS. Compared to the 2.842 seconds to run the java mapreduce function, this is significantly slower.



3. Task C

Task C was implemented by first loading in pages.csv from HDFS. This data is then shrunk to the personID and Nationality of each tuple. Then, the first "headers" line is omitted using a filter. This data is then grouped by Nationalities of tuples, and then for each tuple we generate the Nationality and count of pages that have this Nationality. This is then stored in HDFS. The job took 3.398 seconds to complete. Compared to the mapreduce java code execution time of 2.684 seconds, this pig execution time is significantly slower.



4. Task D

For task D, both friends and pages data sets were loaded into the system. We then cleaned the unnecessary columns and removed header tuples. After that friends data was grouped by MyFriend column and then a count was calculated for how many followers each person has. Then a left outer join between pages and counts was made, so every page would have a count, even if it's 0. The final output tuples are in the form: person's id, count of followers. The entire solution took 4.481 seconds, which is longer than Java's map-reduce solution (3.064 seconds).

The screenshot shows a Hadoop web interface with a modal window titled "File information - part-r-00000". The modal contains the following information:

- Block information:** Block 0
 - Block ID: 1073741863
 - Block Pool ID: BP-1886426487-192.168.56.1-1705002133388
 - Generation Stamp: 1039
 - Size: 338
 - Availability:
 - MSI
- File contents:**

```

William Reynolds,1
Jill Perkins,3
Kelly McGuire,3
Tanya Burton,3
Brian Reed,2
Kevin Castillo,2
Martha Smith,3
Heather Andersen,0
Julia Hoover,4
Richard Tucker,1
Kristen Clark,0
Joseph Hernandez,2
Jennifer Zavala,0
Lisa Cardenas,4
David Conley,4
Heather Stanley,1
Julia Gutierrez,2
Mrs. Christina Stephens DVM,0
Mary King,3
Donna Nguyen,2
          
```

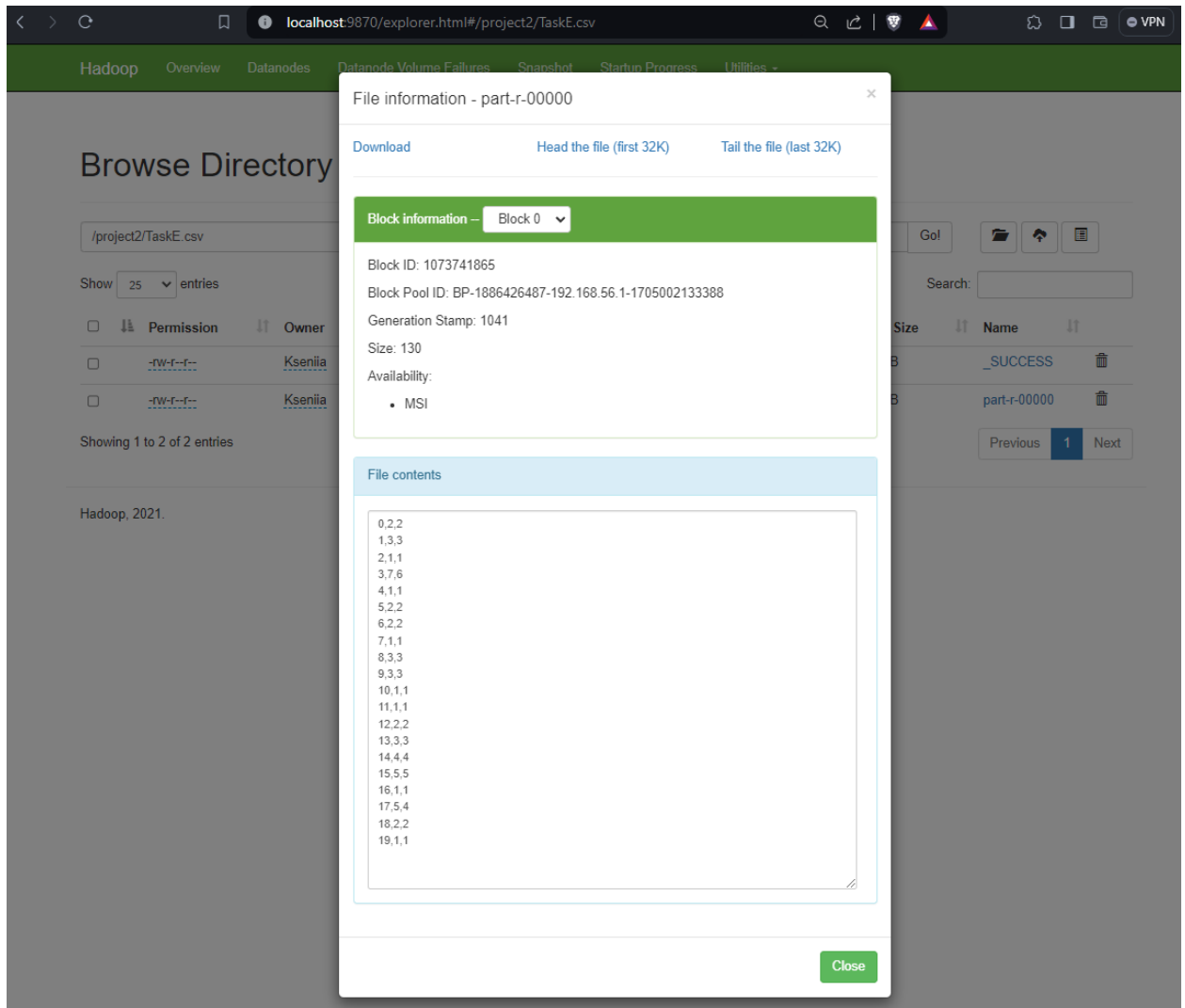
The background interface shows a "Browse Directory" view for "/project2/TaskD.csv" with a table of entries. The table has columns for "Permission", "Owner", and "Size". The entries are:

Permission	Owner	Size
-rw-r--r--	Ksenia	
-rw-r--r--	Ksenia	

The table shows "Showing 1 to 2 of 2 entries". The "Hadoop, 2021." logo is visible at the bottom left of the background interface.

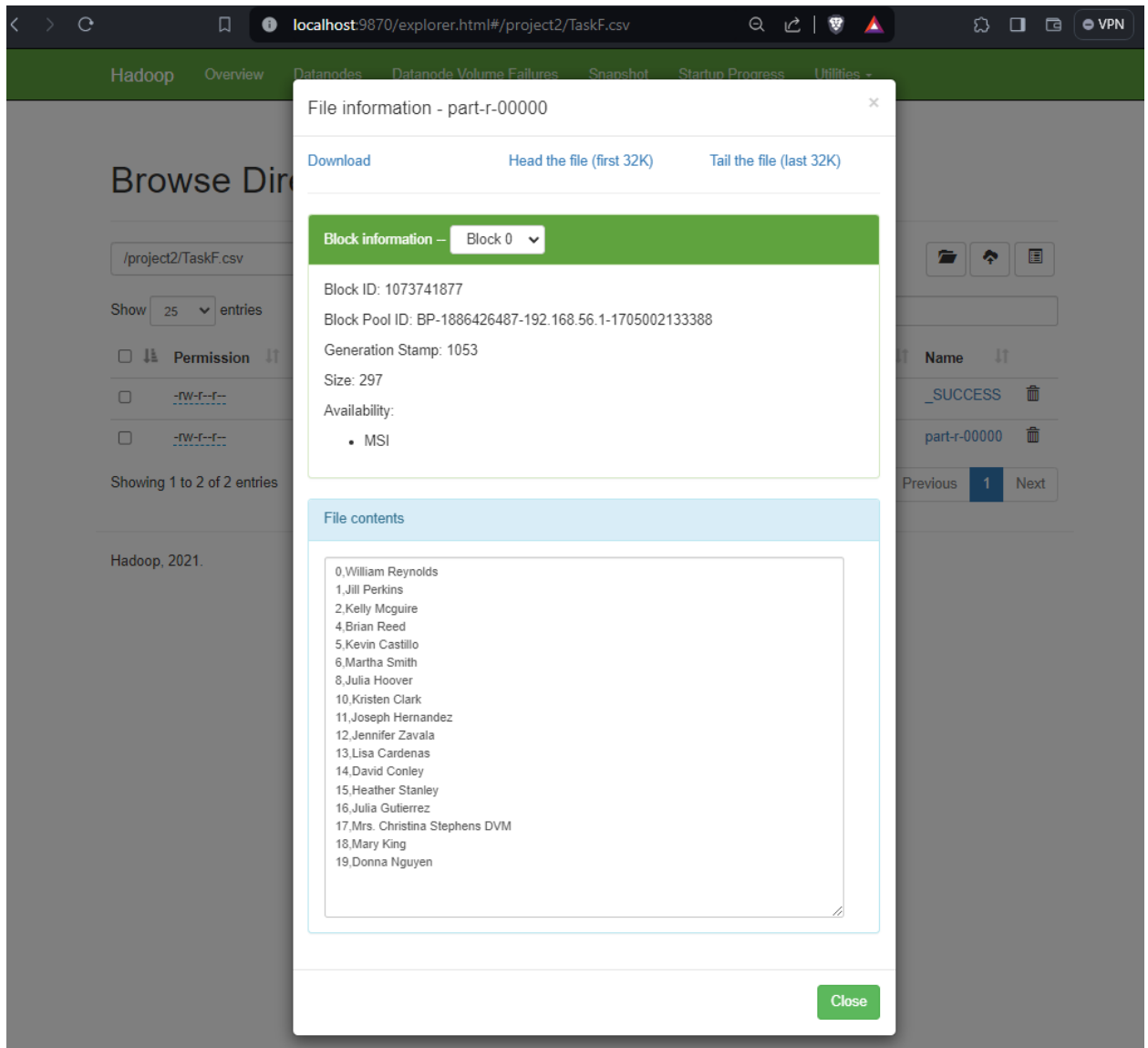
5. Task E

For task E we assume that Facebook page owners who are involved in this case should have at least one visit to a page. In this case, we do not need to take care of pages who never accessed anyone's page. Access logs data set was loaded into the system and unnecessary columns and header tuples were removed. Access logs were then grouped by ByWho column and the result was used to compute the final output. We calculated the total count of access by using COUNT, and the distinct number of access by using a combination of DISTINCT and COUNT. The final output is in the form: person's id, total access, unique accesses. The solution took 3.686 seconds, which is longer than Java's map-reduce solution (2.704 seconds).



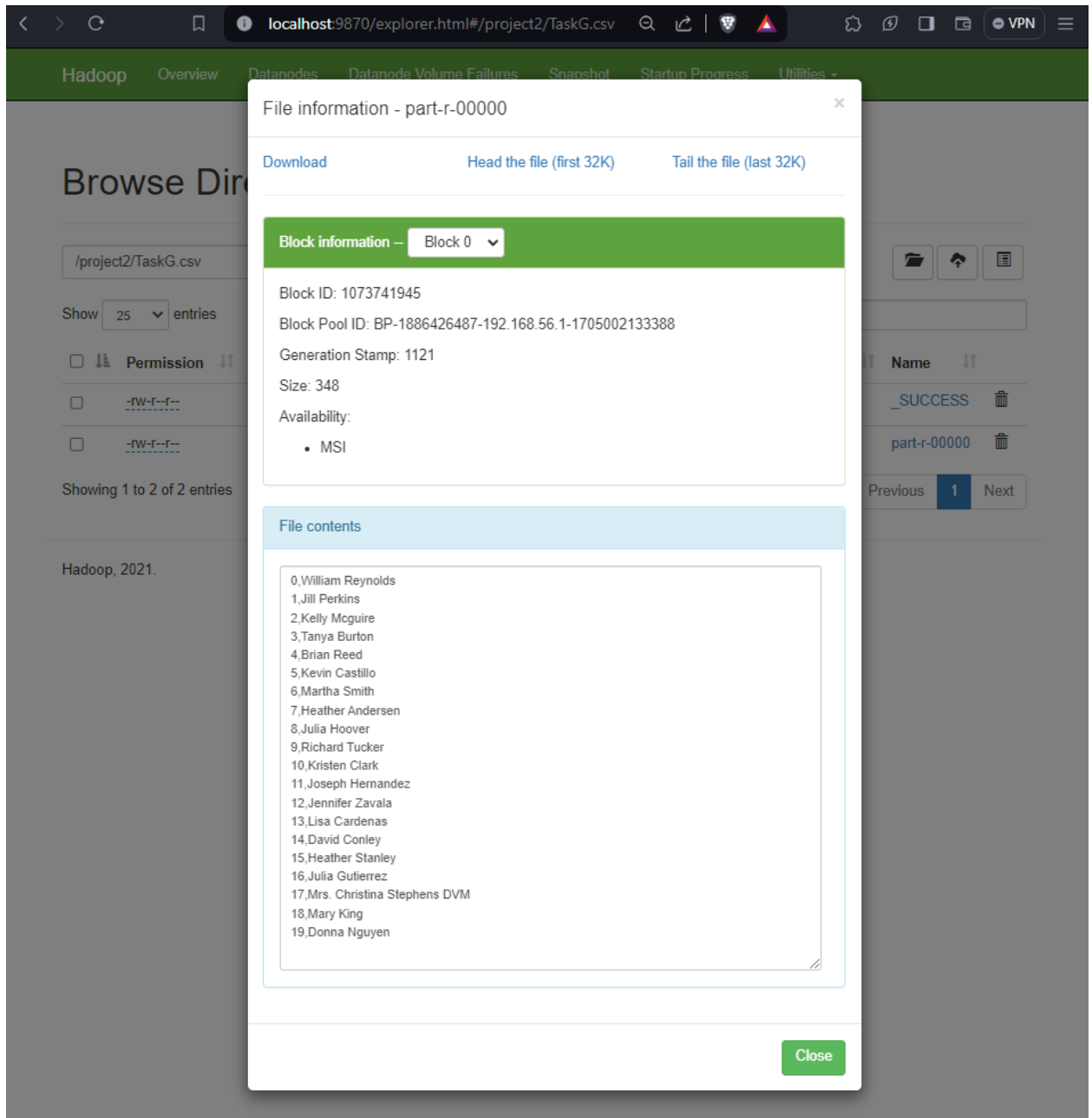
6. Task F

For task F, we loaded all three data sets into the system: access_logs, friends, and pages. We filter out all header tuples as well as unnecessary columns. After that we made a left outer join on friends and access logs data set to find people who declared someone a friend but never accessed their page, meaning that PersonID,MyFriend combination is in friends data set but not in access logs data set (ByWho,WhatPage). We then used filters to remove not null ids and grouped our results by person's id. Then we performed a join on the previous result using pages, so we could get the name of each person. The final output is in the form: person's id, name. The solution took 5.340 seconds to complete, which is longer than Java's map-reduce solution (3.428 seconds).



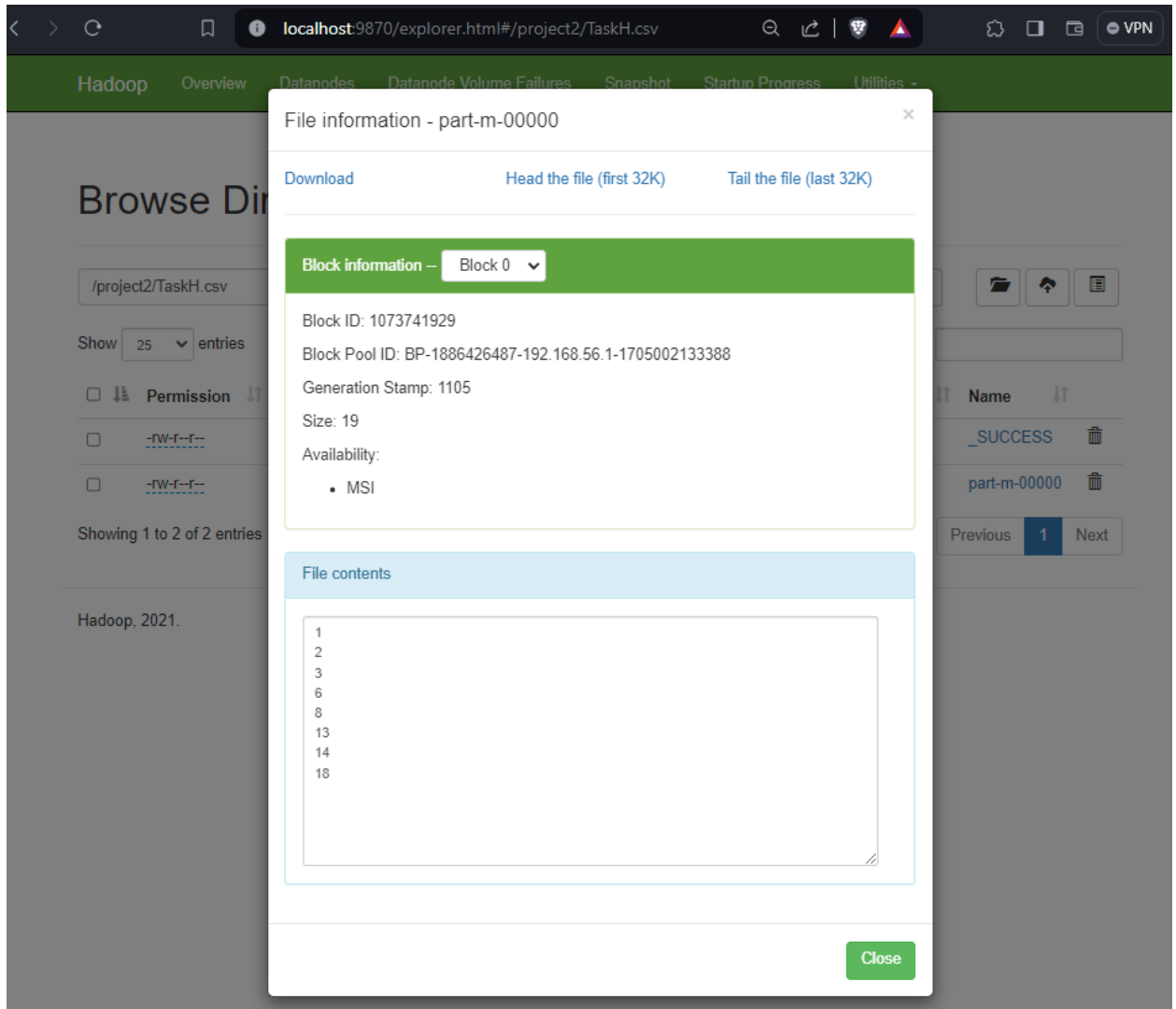
7. Task G

Task G starts with loading `access_logs` and `pages` into the system and removing unnecessary columns and header tuples. Then, we create a group of access logs for `PersonID`, so we can see all of the time logs for each person's id. After that, we generate new data to find the max date (the latest access date-time). This lets us filter tuples using days between function and current time function. Then we join the resulting data with `pages` to get people's names and output their id and name. The solution took 4.592 seconds, which is longer than java's map-reduce solution (2.868 seconds).



8. Task H

For task H we assume that every page has at least one follower, so we don't have to count pages which are not in the friends.csv. First, we load friends.csv data set into the system. Then we filter out the unnecessary columns and header. After that we group based on MyFriend, so we can count up all of the followers for this id. Then, we group by ALL to calculate the average. Finally, we filter out people who have below average follower count and output person's id. This solution took 4.963 seconds, which is longer than java's map-reduce solution which only took 2.716 seconds.



K-Means Clustering Algorithm

Assumptions:

- All coordinates are stored as int values (after centroid calculation double value is converted to int, example: $2065.032 = 2065$).
- If the cluster doesn't have any points near it then it will be removed and not outputted.
- Centroids might have duplicates and are generated from data points using dataGenerator class, duplicate centroids will be removed since all points will be assigned to the first duplicate.

To run the code dataset.csv, friends.csv, access_logs.csv, pages.csv need to be in hadoop server. Otherwise, change the paths to use local versions of the files.

A. Single-Iteration K-Means

We pass k number in args[0] or we can initialize it in the beginning of the main() function. Then, we generate random centroids and store them on hadoop. After that our

team wrote configuration for the map-reduce job and cached centroid file, since it is a small data set.

On the map side, we used the setup function to read cached centroids and store them in memory. In the map function, we read each coordinate stored in our big data set and calculate the distance from this coordinate point to all centroids and pick the shortest one. Then, we output the key as centroid coordinate and value as point coordinate.

On the reduce side, we iterate through all points for each centroid and calculate the mean for x-coordinate and y-coordinate. Then, we output a new centroid coordinate as a key and nullwritable as value.

B. Multi-Iteration K-Means

The logic for the multi-iteration k-means algorithm is pretty much the same. The only difference is that we set up a loop in the main() function that runs map-reduce jobs multiple times, in our case it's 6 iterations. After centroids.csv has been created and stored locally in the data folder, we cache this file to use during our first iteration. During consequent iterations we cache output from the previous job to update our centroids.

C. Convergent K-Means

This version of k-means builds on top of the multi-iteration k-means that we created before. It has a for loop that terminates after 20 iterations. It also has an if statement inside the loop that checks whether previous centroids are similar to the centroids calculated during current iteration. If they are similar, then the centroids converge and we can break out of the for loop and be done.

For this version we changed reducer's output value which was changed from NullWritable to Text. It now outputs the same key and the value is equal to "yes" if the centroid converged or "no" if it didn't converge. Meaning that if one centroid didn't converge, then we will need another iteration. The threshold for converging is set to 1 and can be changed.

D. Optimized K-Means

In terms of optimization, we can use combiner and singular reducer to optimize the solution in part C. We added combiner class which outputs key as key and values as sumX, sumY, and count, which is then passed down to reducer. We used a singular reducer since the amount of data passed to one reducer is small due to combiners. The logic is very similar to what we had previously, but with addition of the combiner that sums up x-coordinates, y-coordinates, and total count of points.

E. Different output for Part D

We already output cluster centers along with an indication if convergence has been reached for each center in Part D. We can do this directly in the reducer, which simplifies the process of identifying if another iteration is needed. Since we have our previous center available to us, we can compute the new center and see if they are equal or within threshold, in our case the threshold is equal to 1. If the new center is within threshold, we output the value of "yes", if it's not, then we output value of "no".

In order to output center as a key and value as points around this center, we needed to store all points in the combiner output value and then perform a split in the reducer using two delimiters. The first one is ";" to separate points and our calculations

for the means. And then use “,” to separate the calculations. The final output of the reducer will look something like this “1657,4018 627,3460 | 2687,4576”. In this example 1657,4018 is the center and 627,3460 and 2687,4576 are points around the center.

F. Comparison on large file (3000 points)

a. Single-Iteration

K = 2. The solution took 3.113 seconds.

kMeansSingle.java	centroids.csv	part-r-00000
1	989,3981	3747,3159
2	2501,4918	1597,2139
3		

K = 50. The solution took 3.156 seconds.

kMeansSingle.java ×

centroids.csv ×

part-r-00000 ×

16	999,2362
17	4399,3347
18	1320,3060
19	2154,4214
20	422,78
21	2481,1685
22	21,4730
23	481,4156
24	1297,4538
25	1217,2173
26	2058,571
27	241,4352
28	1878,3572
29	3559,1184
30	4595,3656
31	3219,96
32	2900,820
33	2976,4053
34	1931,1938
35	2841,361
36	510,1197
37	3402,4507
38	3395,2219
39	4485,1003
40	4929,2376
41	1291,4431
42	2701,3016
43	2320,1130
44	1430,4450
45	578,761
46	3209,383
47	779,2605
48	474,326
49	4218,4628
50	2775,90
51	

kMeansSingle.java ×

centroids.csv ×

part-r-00000 ×

15	2268,1173
16	274,4358
17	2382,356
18	2607,1720
19	2584,2621
20	2542,3117
21	2743,134
22	2868,420
23	2928,963
24	2872,4032
25	3199,3443
26	3378,457
27	3495,99
28	3150,4413
29	3564,2245
30	3456,4520
31	3598,1304
32	352,3039
33	3583,4845
34	67,332
35	4300,482
36	4344,4706
37	454,70
38	4383,4159
39	4202,3162
40	4623,1015
41	4737,3667
42	700,353
43	561,3937
44	4673,2485
45	4668,1788
46	438,1435
47	539,700
48	857,961
49	571,2472
50	856,2219
51	

Overall, k doesn't change the run time of this solution by much. However, since it's a single iteration k-means algorithm, the means that we get at the end might not be the actual centers of the data points that we were given.

b. Multi-Iteration

K = 2 and R = 6. The solution took 10.729 seconds.

1	2354, 3826
2	2612, 1322
3	

K = 50 and R = 6. The solution took 11.457 seconds.

Project	kMeansMultiple.java	centroids5.csv\part-r-00000
TaskE.pig		
TaskF.pig		
data		
kMeanOutput		
centroids0.csv		
_SUCCESS.crc		
part-r-00000.crc		
_SUCCESS		
part-r-00000		
centroids1.csv		
_SUCCESS.crc		
part-r-00000.crc		
_SUCCESS		
part-r-00000		
centroids2.csv		
_SUCCESS.crc		
part-r-00000.crc		
_SUCCESS		
part-r-00000		
centroids3.csv		
_SUCCESS.crc		
part-r-00000.crc		
_SUCCESS		
part-r-00000		
centroids4.csv		
_SUCCESS.crc		
part-r-00000.crc		
_SUCCESS		
part-r-00000		
centroids5.csv		
_SUCCESS.crc		
part-r-00000.crc		
_SUCCESS		
part-r-00000		
.centroids.csv.crc		
access_logs.csv		
centroids.csv		
dataset.csv		
friends.csv		
pages.csv		
java		
dataGenerator		
kMeansConverge		
kMeansMultiple		
kMeansOptimized		
kMeansOptimized2		
10	2072, 2859	
11	240, 2892	
12	2219, 380	
13	2239, 1686	
14	2397, 4169	
15	2594, 2660	
16	2603, 4752	
17	2574, 1000	
18	2670, 3445	
19	2653, 1987	
20	276, 4282	
21	2824, 1584	
22	310, 1676	
23	3183, 2418	
24	3150, 1329	
25	3181, 4496	
26	3191, 305	
27	3309, 1864	
28	3363, 3728	
29	3442, 2977	
30	3494, 914	
31	3764, 1915	
32	3852, 1297	
33	3883, 4793	
34	3945, 2476	
35	3945, 4184	
36	372, 525	
37	4044, 719	
38	4170, 1780	
39	4146, 3395	
40	4491, 1326	
41	4488, 238	
42	4638, 4657	
43	4661, 2829	
44	4737, 1959	
45	4745, 3838	
46	4776, 850	
47	602, 3501	
48	614, 2457	
49	833, 1147	
50	968, 4692	
51		

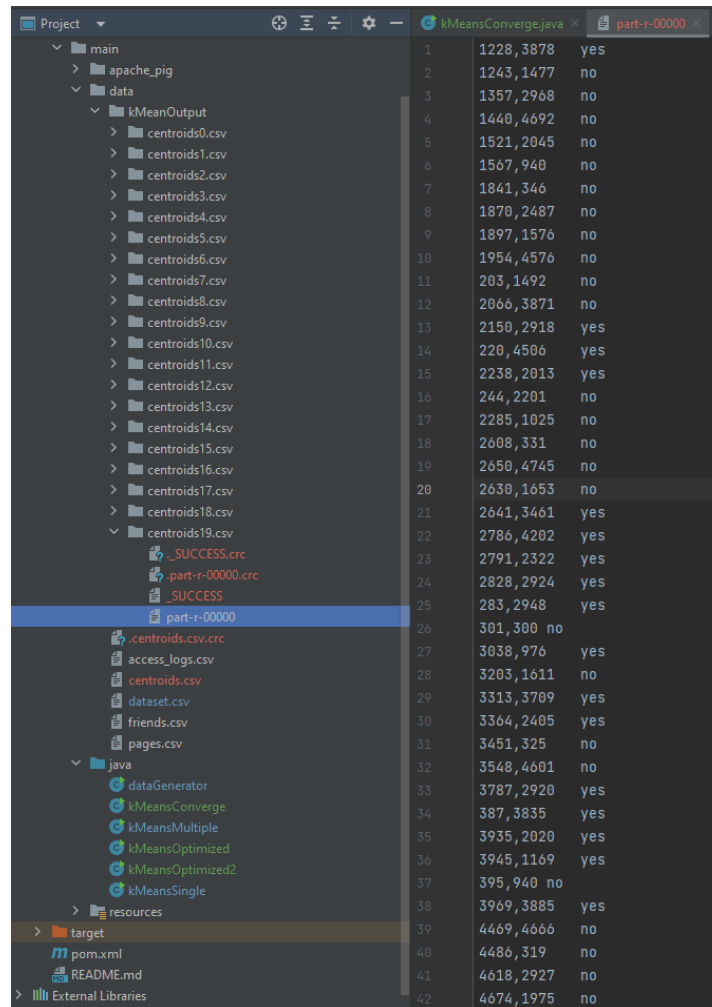
Since we have iterations set to 6, the solution takes longer (Larger R => Longer Time). Even if centers converge early, we will still have to run 6 full iterations. In the case of K = 2 and R = 6, the numbers almost converge, however, the difference between previous x,y and new x,y is still bigger than the acceptable threshold of 1 (can be changed).

c. Convergent Version

K = 2 and R = 20. The solution took 24.307 seconds and 14 iterations to converge.

kMeansConverge.java	centroids12.csv\part-r-00000	
1	1235, 2738	no
2	3702, 2387	no
3		

K = 50 and R = 20. The solution took 34.009 seconds and 20 iterations didn't converge.



From the provided image it's clear that some centroids did converge, but others did not. In this case more iterations are needed for better results. This solution also took much longer than the previous one, meaning that the bigger K we have, the more time it will take.

d. Optimized Version

K = 2 and R = 20. The solution took 22.615 seconds and 13 iterations to converge.

Iteration	Centroid	Converged
1	1230, 2725	no
2	3701, 2400	no
3		

Iteration	Centroid	Converged
1	1229, 2726	yes
2	3700, 2400	yes
3		

Considering this solution took almost the same amount of iterations to converge, it is faster than a simple convergent solution. This means that optimizations that were made are working and making our solution faster.

K = 50 and R = 20. The solution took 33.785 seconds and 20 iterations didn't converge.

The screenshot shows an IDE with a project named 'CS4433_Project2'. The project structure includes a 'main' directory with 'data' and 'kMeanOutput' subdirectories. The 'kMeanOutput' directory contains 20 centroid files (centroids0.csv to centroids19.csv) and two success files (.SUCCESS). The 'data' directory contains various CSV files (access_logs.csv, centroids.csv, dataset.csv, friends.csv, pages.csv) and a 'resources' directory. The 'resources' directory contains a 'target' directory. The table on the right shows the results of the k-means algorithm, with columns for iteration number, centroid coordinates, and a convergence status (yes/no).

Iteration	Centroid Coordinates	Convergence Status
1	1056, 2821	no
2	1195, 1121	yes
3	1221, 3419	no
4	1499, 4401	yes
5	1548, 460	yes
6	1586, 3925	yes
7	1622, 1898	no
8	1759, 4766	yes
9	1875, 2576	yes
10	2129, 3473	yes
11	2132, 1185	yes
12	226, 4427	yes
13	2315, 345	yes
14	2324, 3019	yes
15	2343, 4073	yes
16	2403, 1839	yes
17	2502, 4679	yes
18	248, 1745	no
19	2641, 2598	yes
20	289, 2704	no
21	2877, 752	yes
22	2909, 3366	yes
23	3026, 4000	no
24	3049, 1536	yes
25	3135, 2275	yes
26	3232, 4657	yes
27	3375, 272	yes
28	3509, 2777	yes
29	3618, 1019	yes
30	3779, 3609	yes
31	3850, 4205	no
32	3855, 1990	yes
33	397, 3581	yes
34	4011, 4745	no
35	402, 1066	yes
36	4071, 1279	yes
37	4143, 417	yes
38	4174, 2926	yes
39	4611, 4013	no
40	4637, 1834	yes
41	4675, 743	yes
42	4676, 4674	yes

This solution also took less time to execute than a simple convergent solution. In this case the centroids almost converged, however, there are still some centers that need more iterations to converge.

e. Different Output

K = 2 and R = 20. The solution took 11.376 seconds and 6 iterations to converge.

The screenshot shows a terminal window with the output of the k-means algorithm. The output is a table with 3 columns: iteration number, centroid coordinates, and a convergence status (yes/no). The table shows the results for iterations 1 through 3.

Iteration	Centroid Coordinates	Convergence Status
1	2155, 3768 1393, 3232 2079, 2921 4086, 3281 977, 2391 1948, 3035 857, 3313 1043, 4078 2540, 4159 2347, 3277 1990, 3671 27	✓
2	2814, 1351 4835, 3214 3693, 1840 1315, 1622 4485, 1003 618, 749 1896, 1346 4965, 1300 2005, 1092 1861, 1422 3991, 942 2277, 2	
3		

This solution has the same logic as the previous one, however, the output value is different. It lists all of the points which are around a particular centroid.

K = 50 and R = 20. This solution took 33.896 seconds and 20 iterations didn't converge.

1	1166,1599	1185,1638	1007,1407	1410,1835	987,1339	986,1348	1013,1348	1301,1704	1315,1622	1397,1793	1283,1531	1
2	1263,991	1504,867	1422,1051	1430,1005	1413,823	1425,1014	1064,1241	1040,880	1369,1102	1175,1082	1375,785	1458,
3	1445,4158	1018,4208	1190,4169	1312,3844	1894,4245	1343,4241	1389,4456	1291,4431	1038,4422	1496,4134	1777,3947	1
4	1488,2427	1770,2442	1267,2665	1294,2048	1675,2399	1370,2508	1358,2559	1547,2721	1551,2556	1598,2788	1676,2400	1
5	1494,3211	1486,3272	1624,2988	1436,3477	1236,3395	1359,3351	1574,3165	1428,2920	1607,3563	1286,3047	1245,3023	1
6	1649,515	1439,741	1530,671	1618,872	1841,619	1648,726	1913,642	1610,8	1676,750	1699,339	1723,333	1571,561
7	1659,4709	1705,4371	1580,4884	1908,4852	1886,4378	1336,4977	1490,4832	1549,4911	1367,4885	1775,4867	1957,4533	1
8	1688,1367	1648,1331	1609,1125	1419,1452	1596,1497	1575,1276	1535,1162	1930,1167	1543,1216	1704,1654	1726,1366	1
9	169,1709	60,1652	104,1898	376,1515	242,1731	291,1580	32,1890	82,1473	196,1990	108,1465	164,1950	275,1487
10	1757,1963	1836,1765	1706,2020	1875,2167	1692,1885	2048,2193	1588,1927	1419,1981	1899,1781	2017,2266	1836,2103	1
11	2118,2709	2344,2334	2380,2343	2140,2432	2024,2708	2140,2375	2052,2485	1925,2789	2079,2921	2374,2449	2304,2424	1
12	214,1023	457,963	130,668	335,1209	24,1073	252,733	214,1073	89,943	433,733	392,711	176,975	149,1186
13	2169,289	2306,524	2136,392	2239,26	2059,392	2220,49	2302,416	2311,566	2427,287	2405,528	2190,25	2168,216
14	2222,1868	2475,1992	2100,1481	2135,1730	2146,1575	2413,1831	2038,1889	2237,1989	2313,1954	2042,1632	2149,1490	2
15	2245,3888	2502,3571	2508,3567	2319,4163	2114,4196	2614,3637	2102,3437	2196,3669	2236,3890	2236,4271	2145,4262	1
16	2246,983	2272,1246	2314,916	2081,1217	2556,1236	2394,1259	2160,731	2350,1091	2169,873	2531,990	2163,772	2429,1
17	2611,1641	2858,1741	2357,1605	2642,1750	2563,1434	2431,1683	2656,1865	2453,1667	2417,1598	2505,1564	2566,1466	2
18	2658,4599	3002,4372	2765,4750	2250,4482	2458,4815	2956,4300	2927,4417	2665,4139	2871,4787	2284,4520	2918,4383	2
19	265,2854	343,2978	156,2668	261,2749	70,2612	231,3127	284,2894	393,3022	80,3052	93,2697	459,2831	182,2745
20	2682,3154	2379,3088	2701,3016	3023,2956	2829,2786	2525,2887	2351,3299	2763,3375	2670,3135	2774,2836	2367,3407	2
21	279,2320	329,2245	583,2336	459,2506	14,2299	403,2577	248,2246	73,2180	158,2294	546,2315	407,2456	142,2329
22	280,4083	90,4284	414,4224	461,3851	226,3964	214,4193	350,4360	84,3913	236,3865	80,4192	309,3913	67,4429
23	2864,382	2797,531	2978,428	3096,568	2650,130	2809,218	3072,316	2590,136	2910,416	2865,562	2528,379	3114,228
24	2967,2351	3012,2169	3215,2454	3239,2171	2883,2681	3077,2601	2829,2208	2817,2503	2589,2427	2986,2479	3244,2429	3
25	3036,1052	2649,938	2930,1300	2897,1018	2900,820	3425,988	3109,1085	2781,1126	2841,1116	3146,1163	2929,1069	3058
26	3055,3808	3438,4030	3096,3882	3272,3895	3240,3357	3257,4123	2803,3748	3286,3375	2763,3791	2812,3822	3149,4121	3
27	3201,1620	3301,1867	3124,1387	2961,1814	2998,1743	3361,1565	3513,1758	3215,1554	2981,1570	3342,1888	2965,1442	3
28	338,328	311,419	57,174	337,273	416,567	425,322	220,51	83,359	350,637	407,363	193,23	268,3
29	3545,321	3258,350	3381,239	3518,634	3590,2	3734,364	3285,15	4017,253	3264,394	3411,605	3694,558	3578,464
30	3561,2764	3432,2690	3430,3112	3434,2361	3262,2770	3907,2849	3320,3148	3765,3221	3327,3089	3363,3061	3280,3280	3
31	3598,4570	3981,4843	4027,4403	3600,4133	3317,4305	3860,4371	3137,4468	3274,4292	3732,4650	3625,4755	3756,4373	3
32	364,3491	504,3305	129,3419	361,3699	519,3640	569,3767	360,3397	82,3303	525,3776	203,3224	354,3412	524,3807
33	3875,3737	4134,3325	3683,3946	3936,3581	3818,3881	4158,3772	3527,3931	3995,3425	3765,4043	3866,3590	4234,3927	3
34	3936,1134	4102,662	4109,834	4051,1176	3789,859	3925,826	4043,896	4048,1053	3885,1174	4220,1428	4078,1214	3945,8
35	3946,1965	3780,2335	4075,1615	3563,1941	4252,2159	4221,2114	3810,2095	4003,2185	3910,2016	3785,2014	3861,2213	4
36	4472,2842	4366,2972	4360,2985	4649,2441	4116,2532	4976,2762	4807,3020	4230,3124	4246,2396	4502,3188	4155,3249	4
37	4501,4622	4932,4284	4792,4972	4870,4821	4692,4977	4421,4148	4297,4874	4446,4688	4370,4532	4281,4312	4604,4456	4
38	4524,336	4657,668	4461,47	4420,203	4707,297	4931,592	4807,185	4540,400	4333,203	4881,17	4487,427	4170,580
39	4694,1039	4959,1030	4485,1003	4965,1300	4639,1036	4844,934	4845,826	4595,1175	4851,982	4765,962	4728,1223	4846,
40	4711,1963	4810,1870	4647,1780	4980,1967	4426,1549	4418,2277	4545,1758	4837,1765	4694,1581	4681,2081	4415,1914	4
41	4725,3696	4774,3729	4396,3919	4477,4120	4334,3458	4579,3673	4668,3899	4454,3786	4727,3524	4553,6101	4951,4228	4
42	579,4656	869,4576	716,4255	690,4950	21,4710	349,4518	927,4703	985,4535	593,4820	62,4686	703,4896	649,4686

By looking at this solution and previous ones for $K = 50$ and $R = 20$, it's clear that 20 iterations is usually not enough to get all convergent centroids. The run time of this solution is not much different from the one in part d.

f. Bonus K-means

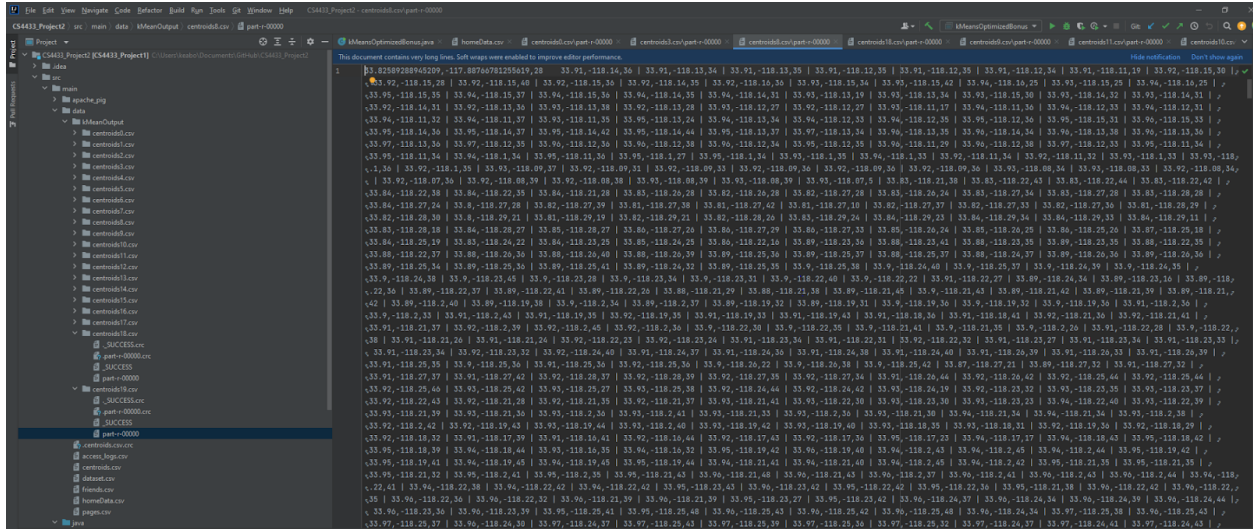
The dataset we chose to implement a new iteration of K-means clustering around was a very large dataset of about 20,000 tuples regarding housing information for houses in California. I found this dataset on Kaggle. Though there is a lot of information about the houses in the dataset, I wanted to take advantage of the house latitude (x), longitude (y), and the median age of the household – this file can be found as homeData.csv in our submission.

This data could be applied in the real world to analyze data based on clusters of housing in an area, calculating average ages, income, and more data based on these clusters for realtors to convey to future house buyers or other realty applications.

With the new implementation, I wanted centroids to also contain the average age of houses clustered and output this alongside the longitude and latitude averages of the house, though the age did not have any factor into how points are clustered, that solely based on Latitude and Longitude.

For this, we also created a new data generator to fit the latitude & longitude values (doubles), so that centroids work properly.

With this new implementation, running the clustering with $R = 20$ and $K = 5$, it takes 63.49 seconds, and converges by file centroids18.csv (19 iterations).



Contribution Statement

Skills

Kseniia Romanova:

Prior to working on this project, I had some knowledge about SQL queries from Database Systems I, as well as map-reduce jobs from project 1. After this project, I familiarized myself with the k-means algorithm and Apache Pig.

Keaton Mangone:

Jackson Lundberg:

Before working on this project, I was knowledgeable about SQL queries and getting familiar with working with hadoop and had just learned about the topics of apache pig and k-means algorithms that were covered in class. After this project, I am much more knowledgeable and comfortable using pig, hadoop, and working with k-means algorithms.

Contributions

Before starting to work on this project, our team decided on how to split work between all members equally. We came up with the solution that the tasks could be split between members of the team, and then the team assembled and checked the work that has been done.

- Kseniia Romanova:
 - Created report
 - Completed tasks D, E, F
 - Completed k-means algorithm
- Keaton Mangone:
 - Completed tasks A, B, C
 - Helped with k-means algorithm
 - Completed bonus k-means clustering adjustments
 - Assisted with tasks G, H
- Jackson Lundberg:

- Completed tasks G, H and reviewed all tasks
- Helped with k-means algorithm

Resource Usage Statement

Our team used resources provided by the professor and TAs on canvas, such as discussion boards, helpful links, and tutorials.

Remove file from hadoop: `hdfs dfs -rm -R /project2/TaskC.csv`

Run file locally for apache pig: `pig -x local`

`C:\Users\Kseniia\Documents\GitHub\CS4433_Project2\src\main\apache_pig\taskC.pig`

- Kseniia Romanova:
 - I used resources provided in the discussion board.
- Keaton Mangone:
 - I used resources provided in the discussion board.
- Jackson Lundberg:
 - I used the resources provided by the professor and TAs on canvas, such as discussion boards, helpful links, and tutorials to help me with this assignment. I used the pig documentation to help me learn more about working with apache pig on my computer as well.
 - <https://pig.apache.org/docs/latest/func.html>
 - <https://pig.apache.org/docs/latest/basic.html>

Credits

We provided screenshots for each task as well as the screenshot of our hadoop directory containing all tasks. Each member of the team was able to run all the tasks successfully.

The screenshot shows the Hadoop web interface at localhost:9870/explorer.html#/project2. The 'Browse Directory' section displays a list of files and directories. The table below represents the data shown in the screenshot:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 13 21:08	0	0 B	TaskA.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 16 13:35	0	0 B	TaskB.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 16 13:30	0	0 B	TaskC.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 13 12:47	0	0 B	TaskD.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 13 14:24	0	0 B	TaskE.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 13 14:51	0	0 B	TaskF.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 16 12:28	0	0 B	TaskG.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 15 23:11	0	0 B	TaskH.csv
-rw-r--r--	Kseniia	supergroup	2.13 KB	Feb 13 11:39	1	128 MB	access_logs.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 14 14:21	0	0 B	centroids.csv
-rw-r--r--	Kseniia	supergroup	30.94 KB	Feb 15 18:08	1	128 MB	dataset.csv
-rw-r--r--	Kseniia	supergroup	1.08 KB	Feb 13 11:39	1	128 MB	friends.csv
drwxr-xr-x	Kseniia	supergroup	0 B	Feb 14 13:32	0	0 B	new_centroids.csv
-rw-r--r--	Kseniia	supergroup	929 B	Feb 13 11:39	1	128 MB	pages.csv

Showing 1 to 14 of 14 entries

Hadoop, 2021.