Lab 4: Storage Benchmarking

Part 1: Benchmarking queries

csv_avg_income

```
import numpy as np
def min_med_max(times):
    return np.min(times), np.median(times), np.max(times)
```

```
# small
# times = bench.benchmark(spark, 25, queries.csv_avg_income,
'hdfs:/user/bm106/pub/people_small.csv')
times = [10.72643256187439, 2.1249630451202393, 1.7598192691802979,
4.8306567668914795, 1.1475980281829834,
         1.0005671977996826, 1.4488351345062256, 1.485851526260376,
1.3573002815246582, 1.3066086769104004,
         1.2712101936340332, 4.235631942749023, 1.3511958122253418,
1.9832358360290527, 4.922960996627808,
         2.0486748218536377, 2.897700548171997, 4.176742792129517,
4.824861526489258, 1.8207581043243408,
         1.841547966003418, 1.8550760746002197, 5.188734531402588,
2.417158842086792, 1.9613821506500244]
print('small: ' + str(min_med_max(times)))

# medium
# times = bench.benchmark(spark, 25, queries.csv_avg_income,
'hdfs:/user/bm106/pub/people_medium.csv')
times = [13.178252458572388, 4.829137802124023, 3.8072474002838135,
2.444997549057007, 2.221158742904663,
         2.237497568130493, 2.3422865867614746, 2.5664727687835693,
2.4378461837768555, 2.6552765369415283,
         3.2900893688201904, 3.543091297149658, 3.9364070892333984,
2.4299283027648926, 2.336242437362671,
         1.9344165325164795, 1.9004936218261719, 3.6448276042938232,
1.7675440311431885, 2.1850666999816895,
         2.8652048110961914, 2.0000977516174316, 3.086674213409424,
2.8874733448028564, 2.6132166385650635]
print('medium: ' + str(min_med_max(times)))

# large
```

```
# times = bench.benchmark(spark, 25, queries.csv_avg_income,
'hdfs:/user/bm106/pub/people_large.csv')
times = [20.702715635299683, 12.01996111869812, 11.616424083709717,
12.531810522079468, 11.59034514427185,
         11.511853218078613, 11.087104558944702, 11.70566701889038,
11.704535245895386, 11.277033805847168,
         12.700540781021118, 11.71760630607605, 12.453936338424683,
12.517395734786987, 11.801110029220581,
         13.210962533950806, 11.768583536148071, 11.630352020263672,
11.11518669128418, 10.827039241790771,
         11.346670866012573, 12.429071426391602, 11.337019681930542,
13.156404972076416, 11.966140508651733]
print('large: ' + str(min_med_max(times)))
```

```
small: (1.0005671977996826, 1.9613821506500244, 10.72643256187439)
medium: (1.7675440311431885, 2.5664727687835693, 13.178252458572388)
large: (10.827039241790771, 11.71760630607605, 20.702715635299683)
```

csv_max_income

```
# small
# times = bench.benchmark(spark, 25, queries.csv_max_income,
'hdfs:/user/bm106/pub/people_small.csv')
times = [8.272036075592041, 1.8393032550811768, 4.851282119750977,
2.180140256881714, 2.1292786598205566,
         1.8506710529327393, 1.821014165878296, 4.629326820373535,
1.9966862201690674, 5.087916135787964,
         1.8305814266204834, 5.6670918464660645, 3.4956483840942383,
4.939809083938599, 2.4026286602020264,
         2.178626298904419, 2.701977491378784, 3.547092914581299,
3.3236334323883057, 2.177008867263794,
         8.756099462509155, 2.0749940872192383, 4.177703380584717,
4.939222812652588, 2.7570571899414062]
print('small: ' + str(min_med_max(times)))

# medium
# times = bench.benchmark(spark, 25, queries.csv_max_income,
'hdfs:/user/bm106/pub/people_medium.csv')
times = [10.65914273262024, 6.745312690734863, 9.543211221694946,
3.6586802005767822, 8.249380350112915,
         3.369917392730713, 3.279481887817383, 2.720396041870117,
3.0080530643463135, 2.313899517059326,
         4.572814464569092, 3.640688419342041, 5.807548999786377,
10.191624879837036, 7.692674398422241,
```

```
        3.5240519046783447, 10.42319130897522, 1.6863858699798584,
4.151952505111694, 3.7211339473724365,
        8.01727843284607, 6.829142332077026, 1.035529613494873,
1.6681253910064697, 1.7069575786590576]
print('medium: ' + str(min_med_max(times)))

# large
# times = bench.benchmark(spark, 25, queries.csv_max_income,
'hdfs:/user/bm106/pub/people_large.csv')
times = [18.9820818901062, 15.00329852104187, 15.486320495605469,
10.64888072013855, 11.528424263000488,
        9.30210018157959, 9.276326894760132, 8.789887428283691,
9.940051317214966, 13.798205375671387,
        9.656174659729004, 11.459694147109985, 8.721364259719849,
8.936478853225708, 9.968897581100464,
        10.533168077468872, 8.924276113510132, 9.056111097335815,
8.762069702148438, 10.241969585418701,
        10.45299482345581, 10.24373197555542, 9.29390025138855,
8.600894689559937, 13.308589696884155]
print('large: ' + str(min_med_max(times)))
```

```
small: (1.821014165878296, 2.7570571899414062, 8.756099462509155)
medium: (1.035529613494873, 3.7211339473724365, 10.65914273262024)
large: (8.600894689559937, 9.968897581100464, 18.9820818901062)
```

csv_sue

```
# small
# times = bench.benchmark(spark, 25, queries.csv_sue,
'hdfs:/user/bm106/pub/people_small.csv')
times = [5.1861231327056885, 0.13321518898010254, 0.09746980667114258,
0.11011457443237305, 0.08919739723205566,
        0.09728646278381348, 0.08553767204284668, 0.08995628356933594,
0.08264780044555664, 0.11731433868408203,
        0.08501434326171875, 0.08080768585205078, 0.0901036262512207,
0.07599043846130371, 0.07678961753845215,
        0.07326912879943848, 0.08805012702941895, 0.07274150848388672,
0.07789969444274902, 0.07732343673706055,
        0.0777125358581543, 0.07209038734436035, 0.06302952766418457,
0.08720731735229492, 0.06086564064025879]
print('small: ' + str(min_med_max(times)))

# medium
```

```
# times = bench.benchmark(spark, 25, queries.csv_sue,
'hdfs:/user/bm106/pub/people_medium.csv')
times = [0.84224534034729, 0.7941586971282959, 0.7487056255340576,
0.7743487358093262, 0.8325152397155762,
         0.7639195919036865, 0.749953031539917, 0.7091042995452881,
0.6646599769592285, 0.6880800724029541,
         0.7214500904083252, 1.0501644611358643, 0.7413945198059082,
0.7277126312255859, 0.6454167366027832,
         0.6585845947265625, 0.6873371601104736, 0.662146806716919,
0.6719765663146973, 0.7171051502227783,
         0.702751874923706, 1.148108959197998, 0.6470708847045898,
1.6949477195739746, 0.7361054420471191]
print('medium: ' + str(min_med_max(times)))

# large
# times = bench.benchmark(spark, 25, queries.csv_sue,
'hdfs:/user/bm106/pub/people_large.csv')
times = [17.950129985809326, 9.296796560287476, 9.301931142807007,
10.017601728439331, 9.682102680206299,
         9.608166217803955, 9.687591314315796, 9.414314270019531,
9.268576860427856, 9.938567638397217,
         10.04891061782837, 10.306361436843872, 9.468688726425171,
10.081690311431885, 10.646968126296997,
         8.959693908691406, 9.90155029296875, 9.773750305175781,
9.730990648269653, 10.109845638275146,
         9.433907270431519, 8.978452444076538, 8.83516788482666,
8.785390853881836, 8.867632389068604]
print('large: ' + str(min_med_max(times)))
```

```
small: (0.06086564064025879, 0.08501434326171875, 5.1861231327056885)
medium: (0.6454167366027832, 0.7277126312255859, 1.6949477195739746)
large: (8.785390853881836, 9.682102680206299, 17.950129985809326)
```

Part 2: CSV vs Parquet

pq_avg_income

```
# small
# times = bench.benchmark(spark, 25, queries.pq_avg_income,
'hdfs:/user/zg866/pub/people_small.parquet')
times = [1.773221492767334, 2.4023077487945557, 1.232593297958374,
1.4581794738769531, 1.4106040000915527,
         0.8896863460540771, 1.3065736293792725, 1.3712272644042969,
1.4223275184631348, 1.0509741306304932,
```

```
        1.2472922801971436, 1.0169014930725098, 6.66071629524231,
1.5358893871307373, 0.930931806564331,
        1.0240628719329834, 0.5250425338745117, 1.1743497848510742,
1.0184199810028076, 1.3818225860595703,
        2.717022180557251, 2.0580880641937256, 1.818345308303833,
0.9028000831604004, 1.169945240020752]
print('small: ' + str(min_med_max(times)))

# medium
# times = bench.benchmark(spark, 25, queries.pq_avg_income,
'hdfs:/user/zg866/pub/people_medium.parquet')
times = [7.507219552993774, 6.279432058334351, 5.331487417221069,
6.211604356765747, 5.163673162460327,
        5.863237380981445, 9.142545938491821, 5.6090099811553955,
2.731693744659424, 1.9259870052337646,
        3.7502567768096924, 0.623316764831543, 1.4893569946289062,
2.6103549003601074, 1.324047327041626,
        1.188188076019287, 0.6049540042877197, 4.616985082626343,
3.7935378551483154, 0.8563001155853271,
        0.6044309139251709, 0.7110710144042969, 3.027240753173828,
0.5894231796264648, 0.7186920642852783]
print('medium: ' + str(min_med_max(times)))

# large
# times = bench.benchmark(spark, 25, queries.pq_avg_income,
'hdfs:/user/zg866/pub/people_large.parquet')
times = [10.83827257156372, 11.81512188911438, 11.871103763580322,
9.251910924911499, 8.611916303634644,
        9.835714340209961, 8.841277837753296, 8.993723154067993,
8.43880295753479, 8.316175699234009,
        8.539896726608276, 9.707672357559204, 8.095171928405762,
8.870018482208252, 8.15876841545105,
        9.327128648757935, 8.970306158065796, 9.479353666305542,
11.452757120132446, 9.264886617660522,
        7.726163625717163, 9.364404201507568, 5.4440248012542725,
6.1871726512908936, 8.895921468734741]
print('large: ' + str(min_med_max(times)))
```

```
small: (0.5250425338745117, 1.3065736293792725, 6.66071629524231)
medium: (0.5894231796264648, 2.731693744659424, 9.142545938491821)
large: (5.4440248012542725, 8.970306158065796, 11.871103763580322)
```

Parquet format on avg income is faster than csv format

pq_max_income

```
# small
# times = bench.benchmark(spark, 25, queries.pq_max_income,
'hdfs:/user/zg866/pub/people_small.parquet')
times = [1.690657377243042, 1.676131010055542, 1.9504203796386719,
2.7357819080352783, 2.1257388591766357,
         1.6809003353118896, 1.2605915069580078, 3.6260974407196045,
1.9735352993011475, 3.1403143405914307,
         1.7707078456878662, 2.240137815475464, 1.6705660820007324,
1.706944227218628, 2.4685511589050293,
         1.7645673751831055, 1.6959528923034668, 1.2168545722961426,
1.7333264350891113, 1.250253438949585,
         1.878328800201416, 1.780709981918335, 2.32139253616333,
2.3764708042144775, 1.7436721324920654]
print('small: ' + str(min_med_max(times)))

# medium
# times = bench.benchmark(spark, 25, queries.pq_max_income,
'hdfs:/user/zg866/pub/people_medium.parquet')
times = [7.626177072525024, 6.064553499221802, 1.3282549381256104,
1.230391502380371, 1.1914067268371582,
         1.5204508304595947, 1.0812830924987793, 1.3148114681243896,
1.0954341888427734, 1.1427099704742432,
         1.1746442317962646, 0.9202256202697754, 1.0947084426879883,
1.1218397617340088, 1.385873556137085,
         1.1052277088165283, 0.9576749801635742, 0.8699929714202881,
1.0313410758972168, 1.3049359321594238,
         1.166604995727539, 1.042555332183838, 1.3507459163665771,
1.7322866916656494, 1.3503851890563965]
print('medium: ' + str(min_med_max(times)))

# large
# times = bench.benchmark(spark, 25, queries.pq_max_income,
'hdfs:/user/zg866/pub/people_large.parquet')
times = [11.177906513214111, 9.880311250686646, 11.356461524963379,
9.88586950302124, 14.026943922042847,
         13.290226221084595, 14.848320484161377, 9.47061538696289,
11.481203556060791, 5.7813661098480225,
         8.519686222076416, 11.605329751968384, 7.697016000747681,
5.376753330230713, 4.810025691986084,
         5.951918601989746, 4.757143497467041, 5.4744157791137695,
10.67808222770691, 7.17632794380188,
         6.188856363296509, 5.002212762832642, 5.7758948802948, 5.72064995765686,
5.160812854766846]
print('large: ' + str(min_med_max(times)))
```

```
small: (1.2168545722961426, 1.7707078456878662, 3.6260974407196045)
medium: (0.8699929714202881, 1.1746442317962646, 7.626177072525024)
large: (4.757143497467041, 7.697016000747681, 14.848320484161377)
```

Parquet format on max income is faster than csv format

pq_sue

```
# small
# times = bench.benchmark(spark, 25, queries.pq_sue,
'hdfs:/user/zg866/pub/people_small.parquet')
times = [0.19817686080932617, 0.14753460884094238, 0.09882020950317383,
0.22094082832336426, 0.09486150741577148,
         0.12266850471496582, 0.06716656684875488, 0.06661176681518555,
0.06406426429748535, 0.0685276985168457,
         0.14310145378112793, 0.0824131965637207, 0.13033843040466309,
0.0650484561920166, 0.06681370735168457,
         0.06559205055236816, 0.07020282745361328, 0.08170962333679199,
0.0683894157409668, 0.06344199180603027,
         0.06851744651794434, 0.06336426734924316, 0.0665283203125,
0.08109068870544434, 0.0655527114868164]
print('small: ' + str(min_med_max(times)))

# medium
# times = bench.benchmark(spark, 25, queries.pq_sue,
'hdfs:/user/zg866/pub/people_medium.parquet')
times = [5.591477155685425, 0.2612447738647461, 0.1329667568206787,
0.13568592071533203, 0.15938878059387207,
         0.18215680122375488, 0.11843514442443848, 0.12320876121520996,
0.11158370971679688, 0.1898651123046875,
         0.10389041900634766, 0.10503673553466797, 0.10054326057434082,
0.11231756210327148, 0.11691856384277344,
         0.10106039047241211, 0.1023111343383789, 0.09516096115112305,
0.10235428810119629, 0.11697912216186523,
         0.1018216609954834, 0.11709952354431152, 0.10643768310546875,
0.09747123718261719, 0.09461760520935059]
print('medium: ' + str(min_med_max(times)))

# large
# times = bench.benchmark(spark, 25, queries.pq_sue,
'hdfs:/user/zg866/pub/people_large.parquet')
times = [3.8279521465301514, 3.89027738571167, 3.9378042221069336,
3.9099621772766113, 3.845405101776123,
         4.0450050830841064, 3.8229308128356934, 3.8364272117614746,
3.937244415283203, 3.888990879058838,
```

```
        3.7519264221191406, 3.9135236740112305, 3.898261070251465,
3.9235680103302, 3.883985996246338,
        3.9536633491516113, 3.8644607067108154, 3.8966197967529297,
3.9793591499328613, 3.855433464050293,
        3.925356149673462, 3.8914384841918945, 3.935798168182373,
3.905827522277832, 3.8712334632873535]
print('large: ' + str(min_med_max(times)))
```

```
small: (0.06336426734924316, 0.0685276985168457, 0.22094082832336426)
medium: (0.09461760520935059, 0.11231756210327148, 5.591477155685425)
large: (3.7519264221191406, 3.8966197967529297, 4.0450050830841064)
```

Parquet format on sue is faster than csv format

Part 3: Optimizing Parquet

We'll only run analysis on large data because small data and medium data might not reflect the difference between the optimization due to the fact that the speed of processing is also affected by Dumbo business. I think large data should reflect difference more obviously even if the Dumbo speed changes over time.

```
# avg_income
# We sort by zipcode since we group by zipcode
# file = spark.read.csv("hdfs:/user/bm106/pub/people_large.csv", header=True,
schema='first_name STRING, last_name STRING, income FLOAT, zipcode INT')
# file = file.sort('zipcode')
# file.write.parquet('hdfs:/user/zg866/people_large_cp1.parquet')
# times = bench.benchmark(spark, 25, queries.pq_avg_income,
'hdfs:/user/zg866/people_large_cp1.parquet')
times = [11.93700098991394, 11.18663740158081, 10.333355188369751,
9.208278894424438, 6.949475526809692,
        6.2524425983428955, 6.847636938095093, 7.330341339111328,
8.222419500350952, 5.4638543128967285,
        9.367559432983398, 9.306601524353027, 8.917731761932373,
6.381899118423462, 7.2479822635650635,
        6.03526496887207, 6.226237058639526, 7.20980429649353,
6.568508863449097, 5.9570348262786865,
        8.334661960601807, 5.167510747909546, 6.555747747421265,
5.656611442565918, 4.975043296813965]
print('before: (4.757143497467041, 7.697016000747681, 14.848320484161377)')
print('after: ' + str(min_med_max(times)))
```

```
before: (4.757143497467041, 7.697016000747681, 14.848320484161377)
after: (4.975043296813965, 6.949475526809692, 11.93700098991394)
```

We can see that the speed after sorting the data with regarding to zipcode increases because parquet is more sensitive to the ordering of rows in a table.

```
# max_income
# We first perform partition on last_name and sort within each partition by
column
# file = spark.read.csv("hdfs:/user/bm106/pub/people_large.csv", header=True,
schema='first_name STRING, last_name STRING, income FLOAT, zipcode INT')
# file = file.repartition('last_name').sortWithinPartitions("income", ascending =
False)
# file.write.parquet('hdfs:/user/zg866/people_large_cp2.parquet')
# times = bench.benchmark(spark, 25, queries.pq_max_income,
'hdfs:/user/zg866/people_large_cp2.parquet')
times = [15.457244634628296, 12.124978303909302, 11.793726444244385,
10.48680067062378, 7.213249206542969,
         7.163550138473511, 6.065203428268433, 6.871981382369995,
7.634435176849365, 8.275434494018555,
         6.902074813842773, 7.976489543914795, 9.025429487228394,
7.3816633224487305, 12.747310400009155,
         8.236906290054321, 8.890035390853882, 8.35719919204712,
8.541898727416992, 6.866826772689819,
         7.562272548675537, 5.778368234634399, 5.286047458648682,
5.920332193374634, 6.979835510253906]
print('large: (4.757143497467041, 7.697016000747681, 14.848320484161377)')
print('after: ' + str(min_med_max(times)))
```

```
large: (4.757143497467041, 7.697016000747681, 14.848320484161377)
after: (5.286047458648682, 7.634435176849365, 15.457244634628296)
```

We can see that the speed with partition on last_name and then sorting by income within each partition is larger, because parquet deals with the block-wise data structure more efficiently. We also tried another transformations. We use parition but by explicitly setting the partition columns when writing out to parquet, but this is very slow, so we don't want to use it.

```
# sue
# We first perform partition on first_name and sort within each partition by column
# file = spark.read.csv("hdfs:/user/bm106/pub/people_large.csv", header=True,
schema='first_name STRING, last_name STRING, income FLOAT, zipcode INT')
# file = file.repartition('first_name').sortWithinPartitions("income", ascending =
False)
# file.write.parquet('hdfs:/user/zg866/people_large_cp3.parquet')
# times = bench.benchmark(spark, 25, queries.pq_max_income,
'hdfs:/user/zg866/people_large_cp3.parquet')
times = [16.8740336894989, 7.696384429931641, 9.147611618041992, 8.328000783920288,
7.761777400970459,
         7.5230138301849365, 17.87124729156494, 10.89341950416565,
12.242117881774902, 7.3309102058410645,
         6.380555629730225, 6.878279685974121, 9.344660758972168, 8.68332552909851,
7.1019532680511475,
         7.090843915939331, 8.707935094833374, 8.055660724639893,
7.919919967651367, 8.343156576156616,
         8.53393030166626, 6.428770303726196, 6.951282739639282, 8.430216550827026,
7.96998929977417]
print('before: (6.663837194442749, 9.07683515548706, 6.663837194442749)')
print('after: ' + str(min_med_max(times)))
```

```
before: (6.663837194442749, 9.07683515548706, 6.663837194442749)
after: (6.380555629730225, 8.055660724639893, 17.87124729156494)
```

The speed with partition on first_name and then sorting by income within each partition is quicker , because parquet deals with the block-wise data structure more efficiently. However, the max value increases, but this could be due to the fact that Dumbo speed varies from time to time and I didn't finish the homework in one setting.

```
# hfs –setrep 3 people_large_cp6.parquet # change replication factor into 2
# hfs –setrep 5 people_large_cp6.parquet # change replication factor into 5
```

I also tried changing the replication factor into 3 and 5, but both implmentation makes the running slower.