CSC443 Assignment 2017: Research Report
CDF: hioefeli,
CDF: Habiberf, 1000668208

PART ONE

3.1 Experiment 1: Optimal Block Size

What is the optimal block size according to your experiment?
The optimal block size is 16384 according to the experiment.

Does it correspond to the system disk block size?
No, it does not correspond to the system block size.

Is there a block size when further increase does not contribute to better performance?
Yes, after and on block size 32768 the performance stays relatively the same at around 12 MBPS.

Is there a difference?
Yes, there is a difference between writing lines and writing blocks sequentially. Writing lines takes an even rate of approximately 14 MBPS to write to the file, while the rate with writing blocks sequentially varies with the input block size.

What is more efficient - writing in blocks or writing in lines? Why?
If we use the optimal blocksize for write_blocks_seq, then writing blocks would be more efficient. However, if we do not use the optimal blocksize, then writing lines would be more efficient. This is because of the I/Os used for writing. In write lines, each line needs an I/O call. However for write blocks, an I/O is called per block instead, which generally makes for a better performance since a lower amount of calls are being made. Though after a certain threshold, write blocks can have a worse performance than write lines, due to the buffer amount.

3.2 Experiment 2: Sequential vs. Random Read rate

What is the ratio of sequential read rate for secondary storage and for RAM?
For secondary storage:
In regards to a block size of 16384,
Sequential Read Rate for Primary: 0.019 MBPS
Sequential Read Rate for Secondary : 0.923 MBPS

Ratio = Secondary/Primary = 0.923 MBPS/0.019 MBPS = 48.58

For RAM:
In regards to a block size of 4194304,
Sequential Read Rate for Primary: 3999.992 MBPS
Sequential Read Rate for Secondary :3999.992 MBPS

Ratio = Secondary/Primary = 3999.992 MBPS / 3999.992 MBPS = 1

Does it correspond to the ratio discussed in class? If not, what do you think is the reason?
The ratio discussed in class indicates the ratio should be significantly larger, so no, it doesn't correspond to the ratio in class. It should be noted that there are problems with the timing function and the source code for ram, so the data above is inaccurate. Though, if the data was accurate, the ratio would still not correspond to the ones discussed in class due to Moore's Law.

Discuss differences in speed and make a conclusion about reading rates (sequential and random reads) for different memories.
Random ram access seems to be significantly faster in SSD than an Optical Hard Drive. While sequential access is also faster in SSD than an Optical Hard Drive. Reading sequentially generally has a better performance than reading randomly. This could be due to the random functions having to read the entire file first, then processing it, rather than reading and processing on the fly.

3.3. Experiment 3: Sequential vs. Random Write Rate

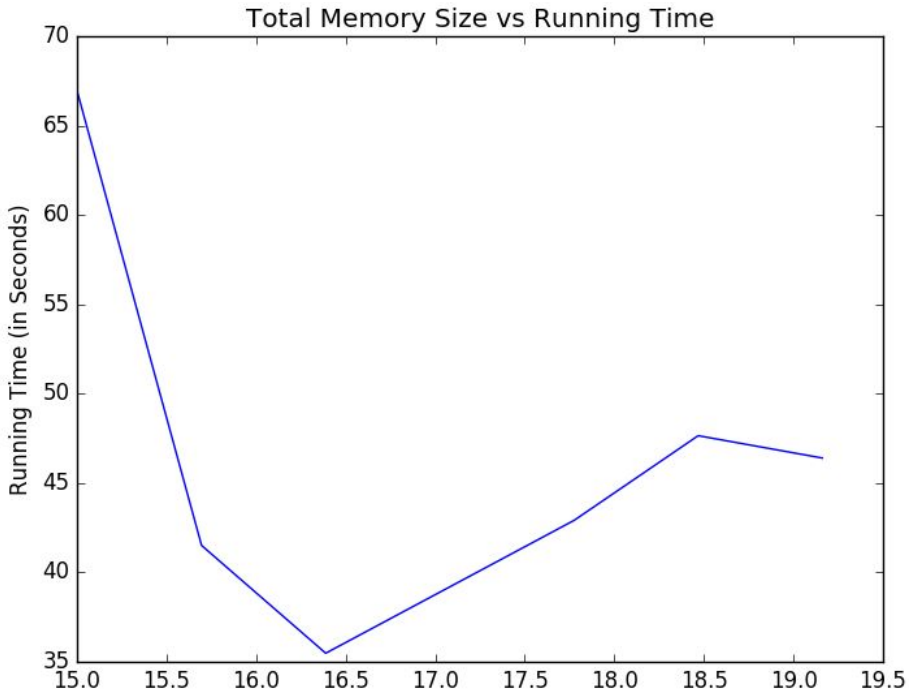
Finally, write summary for your report, discuss what have you learned about access patterns for different memory types. Did these experiment persuade you that we need to design different algorithms for primary and for secondary storage?

Sequential access patterns prove to be faster in the memory types tested, while random access tends to be significantly slower depending on the memory type. Yes, different algorithms should be designed for primary and secondary storage primarily because each is faster in one aspect but slower in another. For example, reading from RAM is significantly faster than reading from secondary storage, but storage on RAM is significantly smaller than secondary storage.

PART TWO

2.2. Buffer size

Is there any difference in performance in your experiments? Explain why there is a difference or why there is no difference.

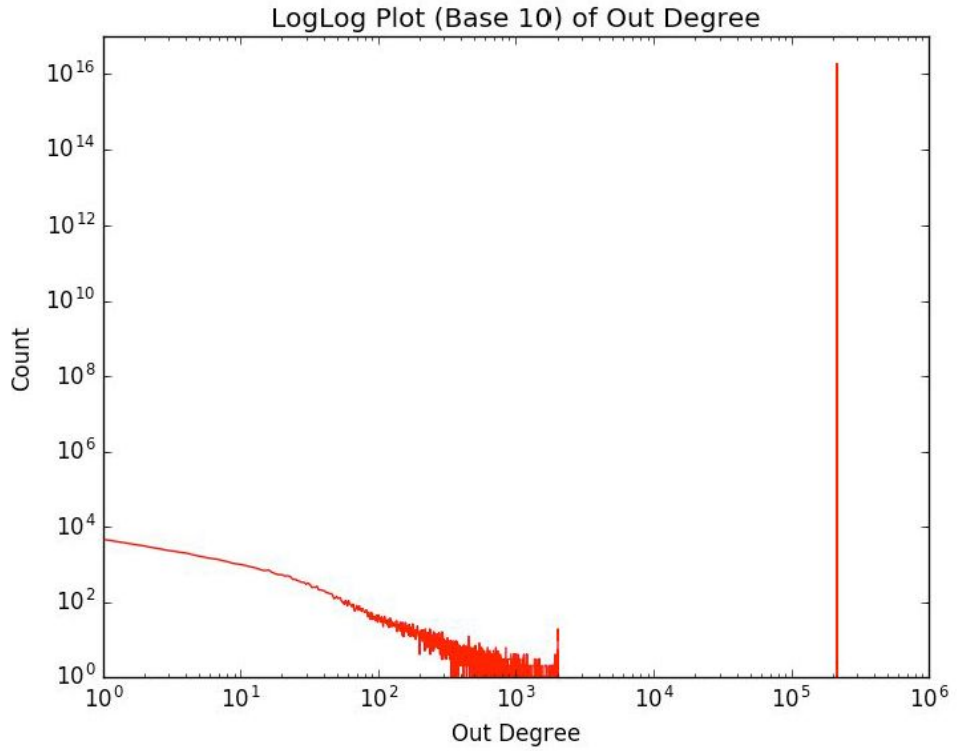


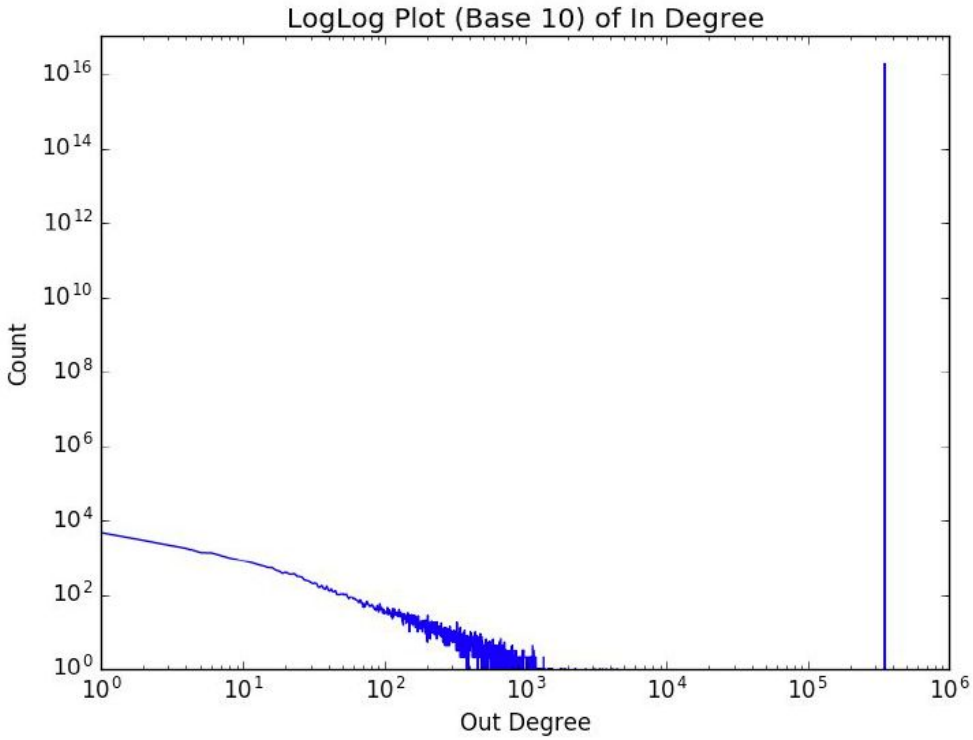
Yes, certain sizes of memory outperform others as visible in the graph. This can be accounted to the difference in memory size. For example, with a large memory size, it may take longer to initially sort all the sublists as they may be larger, but merging them may be quick. Whereas for a smaller memory size, it may take less time to sort the individual sublists as they are smaller. We also see that a larger memory size does not necessarily mean better performance as smaller memory sizes than 200MB outperform it.

Which program is faster: your implementation or Unix sort? Which one uses less memory? Explain the difference (or the lack of difference) in performance. If there is a difference - what in your opinion could explain it?

Our implementation is faster than Unix sort. Ours also uses less memory (Maximum resident size is smaller for our implementation.) One explanation is that unix sort tries to sort the entire file, instead of splitting the file into smaller chunks, sorting those chunks, and then merging them.

3. Research: Twitter graph degree distributions





Both graphs show a power-law distribution.

Summary

In summary, we've seen that 2PMMS is quite an effective algorithm for sorting incredibly large files, however it requires that the memory size and buffer size be reasonable. In our experiments, we used a buffer size of 16384. We were able to sort edges.csv quite fast with this buffer size. Our experiments showed that having a larger memory size at our disposal was actually not that best for speed. For example, using a total memory size of 200MB actually took the longer than some smaller memory sizes. So in general, having more memory does not necessarily positively impact the speed. We also saw that at 1/128 of the original 200MB of memory, we were unable to run 2PMMS.

PART THREE

Step 1: Expressing queries in SQL

Producing the list and the count of user ID pairs which represent "true friends".

Description

Select the tuples where tuple A's uid1 is equal to tuple B's uid2, and tuple A's uid2 is equal to tuple B's uid1 and exclude extra tuples that were already included in the list/count.

Example:

(1, 3)		(3, 1)
(3, 1)		(2, 1)
(2, 1)	will produce	
(1, 2)		
(2, 3)		

Relational Algebra

t1 = twitter_database

t2 = twitter_database

π t1.uid1, t1.uid2 (σ t1.uid1 = t2.uid2 and t1.uid2 = t2.uid1 and t1.uid1 > t2.uid1 (t1 □ t2))

SQL

Create dummy database for ease of use and explanation

```
CREATE TABLE t1
   (`uid1` int, `uid2` int);
INSERT INTO t1
   (`uid1`, `uid2`)
VALUES
   (1, 3),
   (3, 1),
   (2, 1),
   (1, 2),
   (2, 3);
```

```
CREATE TABLE t2
   (`uid1` int, `uid2` int);
INSERT INTO t2
   (`uid1`, `uid2`)
VALUES
   (1, 3),
   (3, 1),
   (2, 1),
   (1, 2),
   (2, 3);
```

SQL Query

```
SELECT
  t1.uid1, t1.uid2
FROM
  t1 CROSS JOIN t2
WHERE
  t1.uid1 = t2.uid2
AND
  t2.uid1 = t1.uid2
AND
  t1.uid1 > t2.uid1;
```

Producing the list of top 10 "celebrities", sorted by (in-degree - out-degree)-difference in descending order (from more famous to less famous).

Description

Get the uid and outdegree of each user, get the uid and indegree of each user. Find the top ten users with the largest difference of indegree and outdegree, then sort by descending order.

Relational Algebra

t = twitter_database

outdegree = π t1.uid1 -> uid, count(uid2)-> count (t)

Indegree = π t1.uid2 -> uid, count(uid1)-> count (t)

Π indegree.uid, indegree.count - outdegree.count (σ indegree.uid = outdegree.uid (outdegree X indegree))

where the above relation is sorted by descending order and is limited to 10.

SQL

```
CREATE VIEW outdegree as SELECT uid1 as uid, count(uid2) as count FROM followers
GROUP BY uid1;
CREATE VIEW indegree as SELECT uid2 as uid, count(uid1) as count FROM followers
GROUP BY uid2;

SELECT indegree.uid, (indegree.count - outdegree.count) as diff
  FROM indegree, outdegree
  WHERE indegree.uid = outdegree.uid
  ORDER BY diff DESC
  LIMIT 10;
```

Step 3: Reports

Report the total number of friends vs. total number of distinct users, and the list of top 10 celebrities with the corresponding in- and out- degrees.

Total Number of Friends
21776094

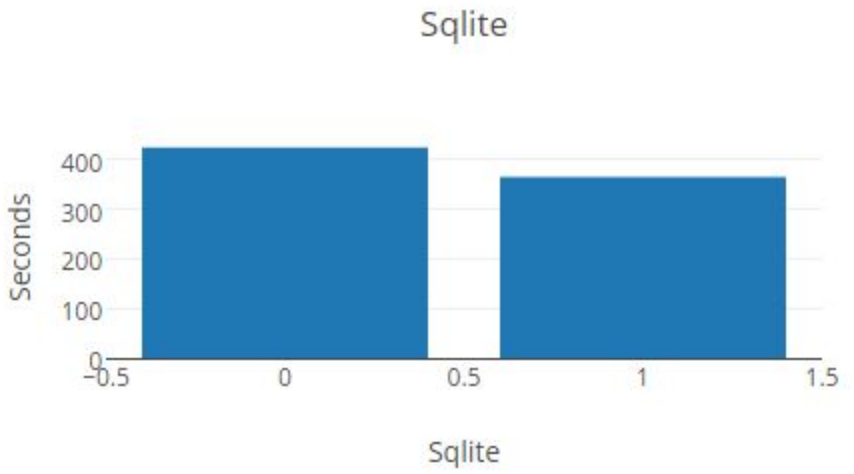
Total Number Distinct Users
26297226

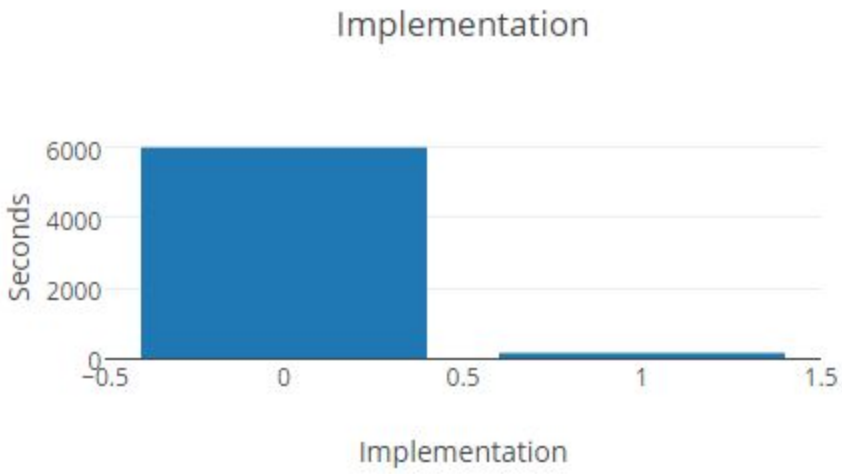
Top Ten Celebrities
UID=5994113, indegree=564238, outdegree=291, diff=563947
UID=7496, indegree=350712, outdegree=6032, diff=344680
UID=1349110, indegree=341790, outdegree=1471, diff=340319
UID=1629776, indegree=172153, outdegree=2119, diff=170034
UID=8121005, indegree=155894, outdegree=34, diff=155860
UID=2041453, indegree=152615, outdegree=620, diff=151995
UID=797152, indegree=118766, outdegree=74, diff=118692
UID=6623784, indegree=115946, outdegree=183, diff=115763
UID=645019, indegree=107858, outdegree=274, diff=107584
UID=3403, indegree=102822, outdegree=4943, diff=97879

Conclude your research report with a brief summary of what you have learned about the users of the social media during this assignment. Are there any other interesting questions you could ask about this data?

There are 26297226 unique users and 21776094 user with friends. User 5994113 has the most followers. With the given data a user must follow someone, or be followed by someone. An interesting question would be how this dataset would store new users in the Twitterverse that don't follow anyone, and have no followers. Currently, this user would not appear in the data.

Plot the comparative performance at the end of the performance report you started in A1.1. Is your implementation faster? If yes, why do you think it is faster? If not, why not?





where left is true friends and right is celebrites.

For celebrities, our implementation is faster. This is because the implemented join runs in linear time, which runs faster for big data. For true friends, our implementation is slower. This could be because of the nested loops used to execute the query, which runs at x^n time.

Conclude your performance report with a brief summary of all the performance results.

Sqlite True Friends
423.320 sec

Our True Friends
~6000 sec

Sqlite Celebrities

363.833 sec

Our Celebrities
166 sec

Reflect about and list everything you learned about working with large inputs.
Optimization is really important when working with large data. A bad implementation can run for days, while a good implementation can use a few seconds and achieve the correct results.