ELEN E6889 – Homework 2

Tong Wu, tw2906

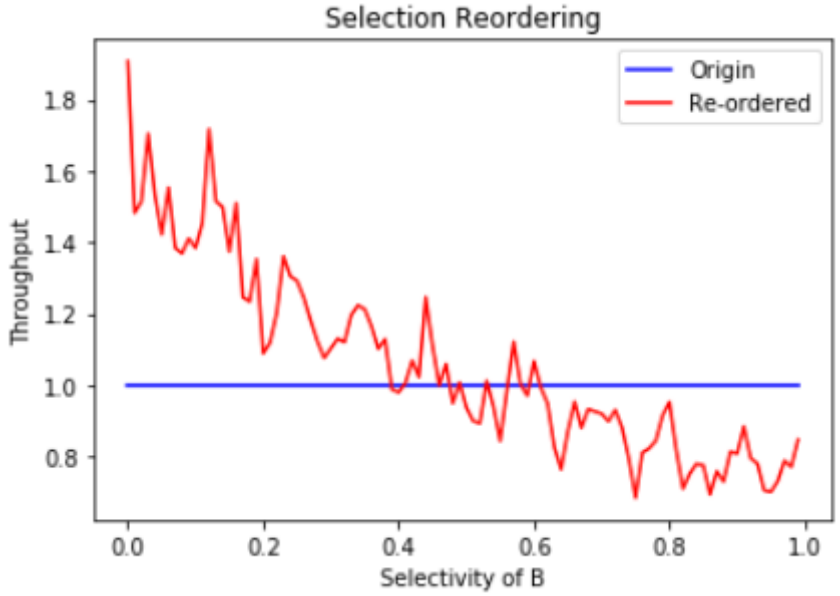
Introduction

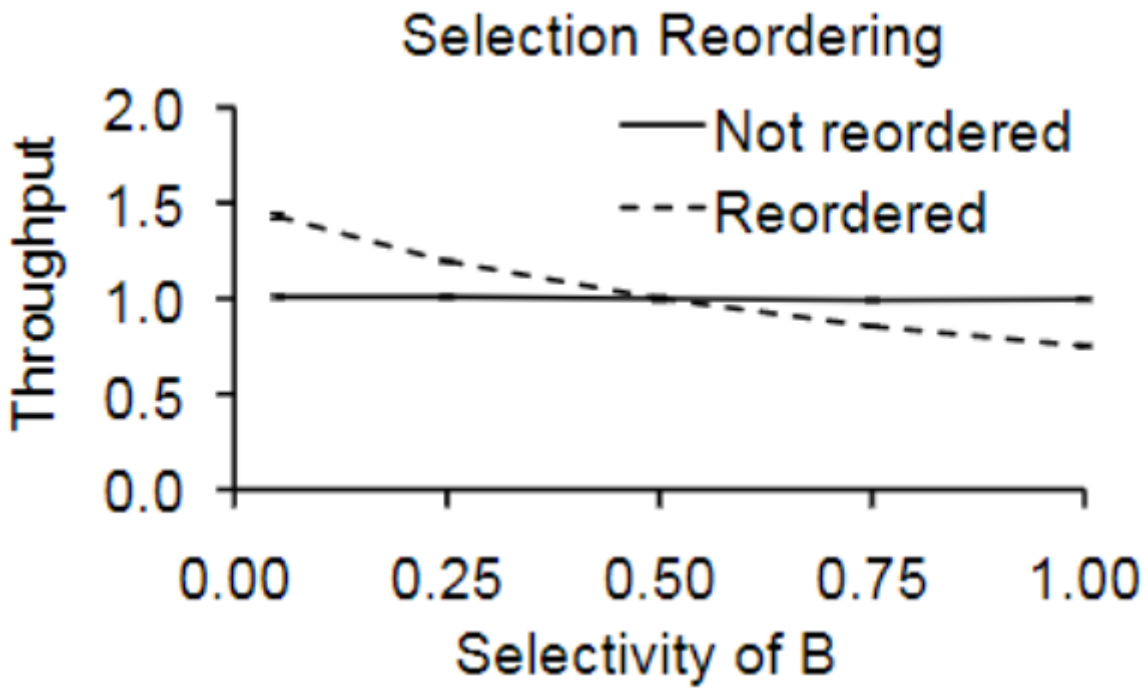
This homework uses python and pyspark to re-produce some results from the paper “A Catalog of Stream Processing Optimizations”. In my work, three optimizations’ results has been re-produced, which are ‘operator reordering’, ‘load shedding’ and bonus part I choose ‘algorithm selection’.

Operator Reordering

In this part, the key idea is to reorder the process order of operator in order to find the different of the performance. In the paper, the operator A has fixed selectivity 50%, and the operator B has different selectivity which should be the x-axis in the result graph, which should varies from 0% to 100%. The optimization discover the throughput of two combinations, which are operator A prior to the operator B, and operator B prior to the operator A. Since two operators has different selectivity, so the following operator can process less or more data if the order and the selectivity of operator B changes.

The code uses `random` package to generate a list containing random integer with range `1-100`, the number of integer is controlled by global variable `NUMBER_INPUT`, which is `100000` by default. Then, in order to maintain the 50% of selectivity for operator A, filter out all even number and keep odd number is a suitable way, by using `lf_list_RDD.filter(lambda x: x % 2 == 1)`. Then, use command `list_a.filter(lambda x: x < selectivity)` to filter out the number which is less than the selectivity from the list of operator A, in order to create the specific selectivity of operator B. After that, change the order of two operators. Finally, collect the data of selectivity and throughput and plot the result diagram. In the paper, the throughput of origin pipeline is fixed at `1`, so when implementing, each operator’s throughput is calculated by `end_time - start_time`, and calculate the ratio of the throughput B from throughput A. The re-produced result diagram and paper result are as shown below.



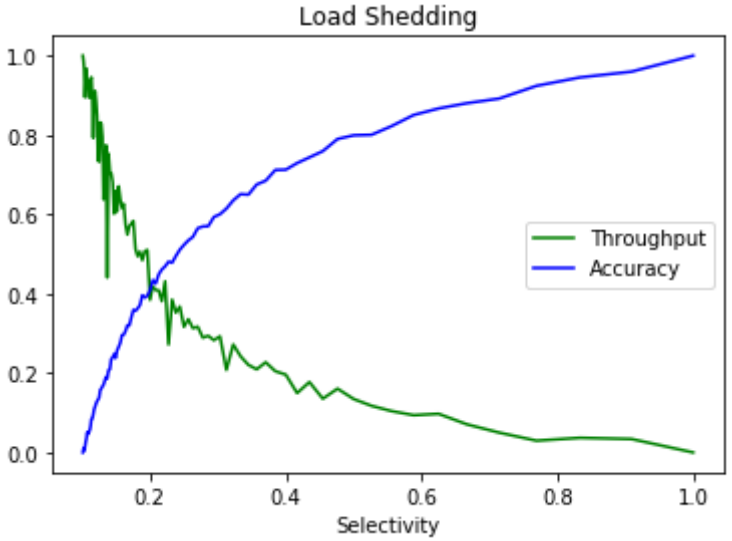


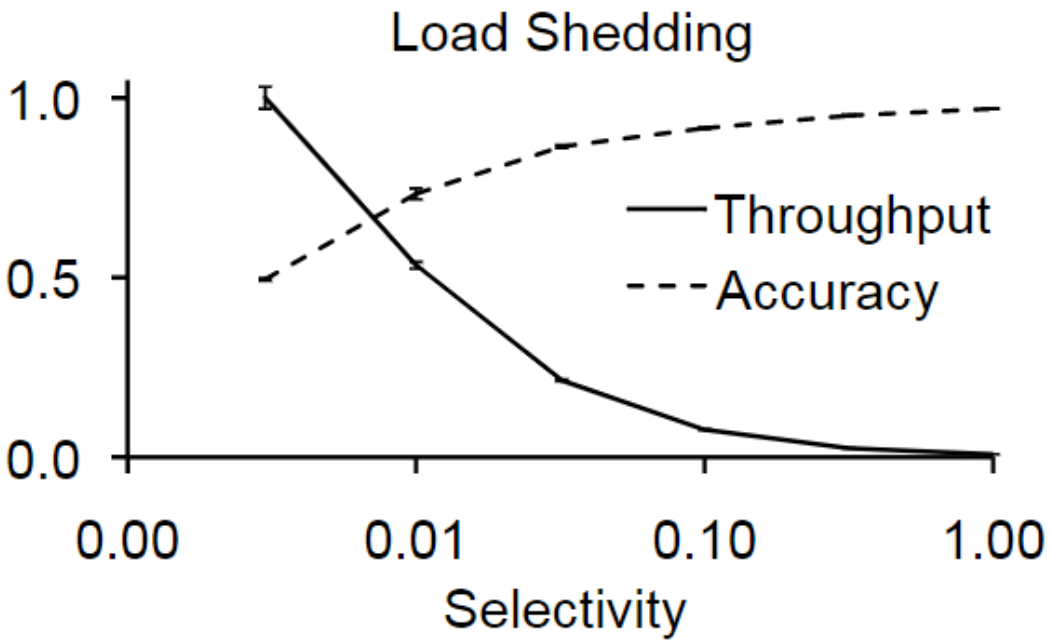
In general, the re-produced result is similar to the paper's, while the re-produced diagram has large jitter, which may caused by the server performance fluctuations. In order to smooth the line, can increase the running time manually to shrink the difference of running time between different selectivity in order to eliminate the influence from performance fluctuations.

Load Shedding

The main idea for load shedding is to give up some accuracy in order to follow up the large throughput given by the "disaster". The main idea of the code is similar to the operator reordering. In the code, the "disaster" can be created by using command `loaded_list = normal_list.map(lambda x: x * 0.02)` . The accuracy can be defined as the remain part of the error, which can be calculated by computing the Euclidean distance from the point to the approximate curve [1], which in code is

`np.linalg.norm(np.array(map_line_list) - np.array(load_factor_list))` . The idea of calculating the throughput is same as previous part. Both throughput and accuracy is normalized in order to fit the scale of the diagram [1], by using `[(error[i] - min(error)) / (max(error) - min(error)) for i in range(len(error))]` and `[(throughput_arr[i] - min(throughput_arr)) / (max(throughput_arr) - min(throughput_arr)) for i in range(len(throughput_arr))]` . The result diagram of re-produced and original are shown below.

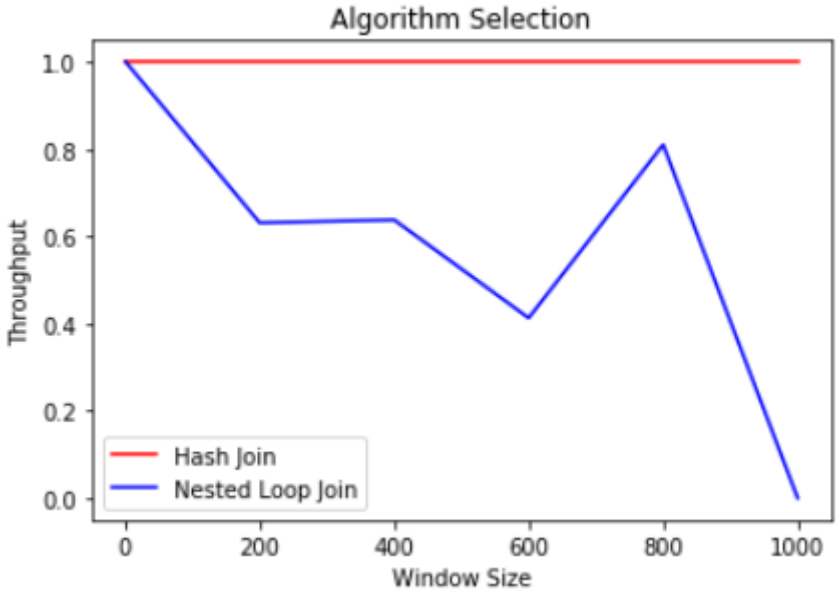


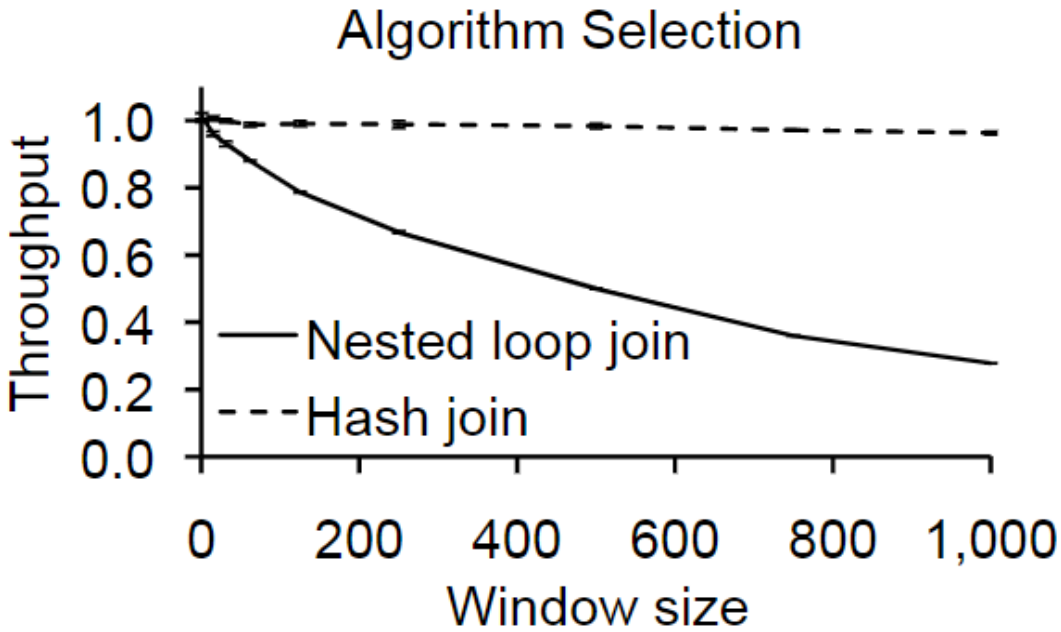


In general, the re-produced result is similar to the paper result. There is also exist the jitter which can be eliminate by the same way as stated in previous part.

Algorithm Selection

As the bonus part, the algorithm selection has been chosen. The main idea of this optimization is to compare the throughput of two different algorithm, hash join and nested loop join at different window size. The has join can be implemented with `list1.join(list2).reduceByKey(lambda x, y: x + y)`, and the nested loop join can be implemented as `list1.cartesian(list2).filter(lambda x: x[0][0] == x[1][0]).reduceByKey(lambda x, y: x + y)`. In order to enlarge the running time, some additional processes are added such as `map` and `reduce`. Since hash join and nested loop join are so CPU-consuming, so the global `NUMBER_INPUT` has set to 1000 in order to reduce the execution time. Then, use command `filter(lambda x: x[0][0] < window_size)` to filter the data to the window size. The re-produced result diagram and paper result are as shown below.





In general, the re-produced result is similar to the paper result.

References

[1] A DYNAMIC ATTRIBUTE-BASED LOAD SHEDDING AND DATA RECOVERY SCHEME FOR DATA STREAM MANAGEMENT SYSTEMS. (2006, July). Amit Ahuj. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=40f63f6df469245c8e75feb9993ce2a8b8fe76cf