Soumya Mishra
Experimental Design – Junior Research
10/13/2017

Title: Effect of using statistical analysis to sort numbers in the most efficient way possible

Problem: Currently, though some sorting algorithms like bucket sort and merge sort have a respectably efficient methodology as to how they sort data sets, these algorithms run into many heap errors or take an unreasonably amount of time to sort the numbers. The only difference between many of these algorithms is just the method of comparing the numbers, which forces the algorithm to pass through the data set too many times; if there are *n* elements in a data set, the algorithm will have to pass through the data set at least *n* times to compare each number to every other element in the data set. The worst cases are usually n log(n) or n^2 – sorting is just brute force at this point. Some of these sorts, like bucket sort, are non-comparison based sorting algorithms and although these algorithms are relatively efficient, their programming leads them to use up unnecessary space, thus causing heap overload errors. Furthermore, these algorithms when This needs to be solved if we want to make sorting more efficient. If sorting can become more efficient, a multitude of algorithms will be much more efficient and allow applications to work faster while still saving space.

Hypothesis: If the analysis of the distribution of the data set is used to create the parameters of each of the buckets, then the sorting of large data sets will become more efficient for the time complexity of this algorithm will be approximately 6n (the number of operations taken to complete sorting using this algorithm will be 6n). This is because instead of comparing each data element with every other in the data set, or wasting space due predetermined bucket parameters, each data element will have to be compared to one value to place them in their respective buckets, and the number of buckets will be created on the spot as the algorithm is running to make sure there is no unnecessary empty space being wasted.

Soumya Mishra
Experimental Design – Junior Research
10/13/2017

<u>Independent variable</u> – using statistical analysis to set the parameters of the bucket

<u>Dependent variable</u> – time complexity using Big-O notation; time it takes for algorithm to sort *n* elements

<u>Constants</u> –

- Memory available
- Laptop and processor being used
- Bucket sort used for comparison
- Merge sort used for comparison
- Array.parallelSort used for comparison
- Multiple processing units (6 node network)

<u>Control</u> – run time for bucket sort and merge sort for the same number of elements that I am testing

<u>Repeated Trials</u> – For each data set I use (around 9 per distribution, and a total of 7 distributions), I will perform three runs per sorting algorithm (merge sort, bucket sort, and the new algorithm) during each step of my procedure in which I've made a significant change to the algorithm.

<u>Materials</u> -

- Java, Anaconda applications on laptop
- Excel spreadsheet to store data, Mathematica to examine any patterns
- Data sets that need to be sorted – 9 data sets per type of statistical distribution (around 7 total) ~ 63 data sets
- Multiple processing units (6 node server)

<u>Procedure</u>

1. Write preliminary program in Java; this will serve as the backbone structure for both the multithreaded java program and python program. This will include the basic function to separate a normally distributed data set into equally sized buckets with parameters dependent on the variation of the data set.
    a. This algorithm will then be compared with a sorting algorithm (Merge sort/Parallel Sort and bucket sort) that I write myself in Java (this allows for consistency and accurate comparisons for both of these algorithms will be unoptimized) and some preliminary data will be collected. The data set fed into the algorithm will just be a randomized array created in the beginning of the program.
    b. If my algorithm is faster than this unoptimized version the sorts I decide to compare it to, I will compare it to optimized version of these sorts and collect data.
    c. If it proves to be significantly slower than these sorts, I will then try to write my algorithm in a different manner to make it more efficient.

2. Because Java does not have built in function to analyze the variation of a given data set, the preliminary program will have to be modified to be implemented in Python so that Panda can be used for statistical analysis.
3. Then, I will learn how to use Panda to accomplish my goal of figuring out the variation of the data set and find a more universal way to set the parameters for the buckets. Because of future problems that may occur in using Python, I will examine how Panda operates and use similar strategies in Java to figure out the variations and parameters for the buckets.
    a. I will start feeding the algorithm the data sets (those with around 10000 elements) from each type of distribution to see if the data can actually be sorted.
    b. If the data sets can be sorted perfectly, I will then start comparing my algorithm with merge sort/parallel sort and bucket sort in Python and Java and collect data. I will collect this data in two separate data sets (one for Python and one for Java).
4. I will learn how to multithread in Python to speed up my algorithm. However, multithreading in Python has only been currently implemented and is much slower than Java and its multithreading capabilities. Therefore, I will learn multithreading in Java as well and compare which program runs my algorithm faster. I will also collect data during this step and record them in two separate data sets (one for Python and one for Java) as well.
5. Based on the data collected from steps 3 and 4, I will determine which language I should proceed with to complete the project. I will optimize my algorithm by

consulting with professionals (I have contacted Computer Science and Statistics Professors at Duke University who can guide me through the process).

a. I will then collect data by feeding the data sets into my optimized algorithm and then use the mean run time for each set of data sets to determine the Big O Notation for my algorithm. I will plot the points on excel or Mathematica (whichever can allow me to create the most accurate and precise formula) and then figure out the regression that best fits the run time. It is important to keep in mind that in this step, I will feed my data into the 6-node server and modify my program to use Hadoop-MapReduce (only the map function) to mimic companies that need to process huge amounts of data (hundred billions).

6. To finally prove that my algorithm can increase the efficiency of existing algorithms with a near linear run time, I will take an R algorithm that relies heavily on sorting, implement my sorting algorithm into it, and then see whether it works faster - and if so, by how much - than the R algorithm with its initial sorting algorithm embedded into it.

References –

Wang, L., et al. (2012). G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*. doi:10/1016/j.future.2012.09.00

Monaknov, A. (2016). Composable multi-threading for python libraries.

Goel, N., Laxmi, V., Saxena, A. (2015). Handling multithreading approach using java. *International Journal of Computer Science Trends and Technology*.3.2.

Hai, H., Guang-hui, J., & Xiao-tian, Z. (2013). A fast numerical approach for Whipple shield ballistic limit analysis. *Acta Astronautica*. 93. 112-120. Retrieved from http://dx.doi.org/10.1016/j.actaastro.2013.06.014

Dehne, F., & Zaboli, H. (2016). Parallel sorting for GPUs. *Emergent Computation*. 293 – 302. doi:10.1007/978-3-319-46376-6_12

Statistics –

1) I am measuring the effect of using statistical analysis to set the parameters of the buckets on the number of operations (directly proportional to run time) it takes this algorithm to sort data sets. I will program the algorithm in Java and Python, see which one is more efficient (has the lesser run time). I will then apply multithreading, professional optimization, and multi-distribution across multiple servers to optimize the algorithm. I will gather the mean run times for sorting all the data sets and then create a regression that fits the data points collected. I will collect data on the run times of merge sort, bucket sort, and parallel sort when sorting the same data sets. I will then use ANOVA and Kruskal Wallis analysis tests to analyze the data.
2) I am statistically comparing statistical analysis of the data set and the run time of the sorting algorithm.
3) The controls for this experiment are the run times of merge sort, bucket sort, and parallel sort when given the task to sort the data sets for each distribution.

Soumya Mishra
Experimental Design – Junior Research
10/13/2017

4)

Table 1: Effect using Statistical Analysis of Normally Distributed Data Sets to set Parameters of Buckets on Run Time of "Stat" Sort (comparing to run times in three other sorts)

Elements	**Sort Run Times (seconds)**								
	Merge Sort (Parallel)			Bucket Sort			New Sort		
	Trial 1	Trial 2	Trial 1	Trial 2	Trial 3	Trial 3	Trial 1	Trial 2	Trial 3
1,000									
10,000									
100,000									
1,000,000									
10,000,000									
100,000,000									
1,000,000,000									
10,000,000,000									
100,000,000,000									

* 7 more tables, like the one above, will be used to compare customized parameters of the buckets on the run times based on the 7 other popular distributions (Binomial Distribution, T-distribution, Poisson distribution, Chi-squared distribution, Uniform distribution, Exponential distribution, Beta distribution) on Java for steps 3 and 4. 2 tables, total, like the one above will be used to collect data from my algorithm in Python in steps 3 and 4 in my procedure.

5) For each control as well as the "Stat" sorting algorithm, I will run three trials for each data set for every distribution. I will run a total of 1260 trials for the entire project. The time allotted for data collection and the space available on the nodes will limit me to these trials; I will only be able to test upto multiple billions of numbers.

6) I will be using ANOVA and Kruskal-Wallis tests to see if the mean run times of my algorithm, bucket sort, and merge sort vary by a significant amount for the different methods of sorting. I will first use a Kruskal Wallis test to compare the medians of the mean run times for each of the 7 distributions (7 total Kruskal-Wallis tests). I will then conduct another Kruskal Wallis non-parametric statistical analysis to compare the collective mean run times of the three sorts; I will average out the run times for all the distribution based on just the number of elements (not the distribution) and then compare these means to get a holistic analysis of the run times of these sorts. A Kruskal-Wallis non-parametric statistical analysis will be the primary statistical test as there are only 10 values in each data set. This test will examine any significant differences of the medians of the different data sets for the different sorts. Then I will perform the parametric equivalent to this test, the ANOVA test, which examines any significant differences of the means. Because of the small size of our data – less than 30 data points per sort per test – the means of the ANOVA will be greatly impacted by our outliers as they will affect our means by 10%. The medians will all stay the same as they are not greatly affected by any outliers.