A Comparison of Sorting Methods for Sorting Data on Lower End Hardware and Software

Dan Cojocaru
Department of Computer Science,
West University of Timișoara
email: dan.cojocaru00@e-uvt.ro

Abstract

As smartphones become more affordable, people in poor countries start to get connected to the internet. The speed of the connection and the performance of the devices, however, can negatively impact the experience.

In this paper, I analysed the performance of sorting algorithms when applied to streams of data coming through slow connections and determined that splitting the data into smaller chunks and handling those can significantly improve the performance.

Contents

List of Figures

1 Introduction

1.1 Motivation

More of the world is getting connected to the internet, especially in developing countries, but also in remote areas of developed nations. This provides a challenge, as both the hardware and the software of these new internet users are significantly lower in performance than what most of the already connected world uses.

Since most of the world is used to potential internet speeds up to (and beyond) 100 Mbps, developers tend to assume that the bottleneck of an algorithm would be how quick a processor can process their data, rather than obtaining the data in the first place.

Given this (and recent internet connections), I decided to write this paper analysing how a different approach focused on optimising algorithms based on the speed of data throughput rather than data processing can be an important point of focus.

1.2 Short description of findings

Although initially I thought about exploiting the only commonly known online sorting algorithm[1], insertion sort (as described in [Aigner and West, 1987]), the results were disappointingly uninteresting.

I then turned to another approach. I started looking into chunking the incoming input and sorting little chunks at a time. Using this approach, I observed significant improvements for these following algorithms I analysed:

- Quicksort [Hoare, 1962]
- Heap sort
- Merge sort [Pardo, 1977]
- Bubble sort [Astrachan, 2003]
- Selection sort

1.3 Contributions

Instead of taking the usual approach of comparing sorting algorithms in a perfect scenario with no external influence, in this paper I compare sorting algorithms in real world situations where things like slow internet connection have a bigger impact than which algorithm is faster.

1.4 Reading instructions

Give an overview of the organization of the paper. 1-2 paragraphs outlining the contents of the paper.

[1] online algorithm = an algorithm which can process data as it is available rather than waiting for the whole dataset before starting to process it

2 Approach

In order to gather the data for this paper, I decided to write a server that emulated bad internet connections and a client which communicates to it.

The client will establish a connection to the server and ask it to start a *sorting session*. A sorting session is characterised by the following steps:

- The server will record the time at which the session is started.
- The server will communicate one by one each item in the list to be sorted. The communication of each item will have a delay that the client asked for between them.
- The server will ask the client to sort the received list.
- The client will send the sorted list back.
- The server will record the time at which the session is over, will check the received list to be correctly sorted and then will reply to the client informing it of the time it took for the list to be sorted.

This approach will impose several delays in the communication of the list. Even though the client may ask for a 0 millisecond delay, the TCP communication socket[2] will ensure that the communication is *near*-instantaneous, but **not** instantaneous. Furthermore, in my implementation of the client, a GUI[3] is maintained in order to communicate progress of sorting, the update of said GUI also taking time. Both the non-instantaneous communication and communicating progress are expected real world behaviours of user-facing applications and are therefore justifiably included in the time needed for a sorting session to complete.

3 Results

Initially, I expected insertion sort, which is an online algorithms, to perform better in an online context. Even so, my expectations were not met when I put them to text, insertion sort performing worse than other algorithms on average. This is despite other algorithms having to wait for the data to fully arrive before starting to process it.

I then tried different methods of splitting the received data into smaller chunks that can be processed as they are ready, instead of using all the data as one big chunk. This approach yielded surprising results.

The attempted methods of dividing the work into chunks were:

- Equal Chunks – after processing a chunk, wait until the amount of unprocessed data equals the amount of processed data before reprocessing

[2] (in networking context) socket = a bidirectional channel with which a client and a server can send each other data

[3] GUI = graphical user interface

- Constant Chunks – a constant number of unprocessed items will be taken as a chunk and processed each time
- Exponential Chunks – a counter starting at 1 is kept, and each time the amount of unprocessed items is greater or equal than the counter, the items are processed as a chunk, and then the counter is multiplied by a specified value

The processing of a chunk implies sorting it using a given sorting algorithm and then merging it with the existing processed data.

Although in my experimentation, which included constant chunks of size 500 and 2000 as well as exponential chunks with a factor of 3 and 5, a conclusive result was not reached on what values work better, further experimentation being needed for this, results constantly show that dividing the received data into chunks yields significantly better results.

On a test dataset of 10,000 randomly generated numbers with a 20 ms delay between sending each item, the results were as follows:

- Insertion sort: 3 minutes, 31 seconds, 210 milliseconds
- Quicksort: 3 minutes, 29 seconds, 736 milliseconds
- Heap sort: 3 minutes, 31 seconds, 234 milliseconds
- Constant Chunks(500) Quicksort: 3 minutes, 28 seconds, 739 milliseconds
- Constant Chunks(500) Heap sort: 3 minutes, 30 seconds, 656 milliseconds
- Constant Chunks(2000) Quicksort: 3 minutes, 29 seconds, 130 milliseconds
- Constant Chunks(2000) Heap sort: 3 minutes, 30 seconds, 739 milliseconds
- Exponential Chunks(3) Quicksort: 3 minutes, 28 seconds, 960 milliseconds
- Exponential Chunks(3) Heap sort: 3 minutes, 30 seconds, 729 milliseconds
- **Exponential Chunks(5) Quicksort**: 3 minutes, 26 seconds, 772 milliseconds
- Exponential Chunks(5) Heap sort: 3 minutes, 30 seconds, 479 milliseconds

Significant improvement start to appear beginning with the 25,000 items to sort mark, where a difference of up to 20 seconds can be observed:

Chunking method	Sorting algorithm	Time for sorting 20,000 items
No chunking	Insertion sort	38 seconds 423 milliseconds
No chunking	Quicksort	41 seconds 172 milliseconds
Equal chunking	Quicksort	42 seconds 659 milliseconds
Constant chunking 500	Quicksort	27 seconds 962 milliseconds
Constant chunking 2000	Quicksort	23 seconds 843 milliseconds
Exponential chunking 3	Quicksort	24 seconds 484 milliseconds
Exponential chunking 5	Quicksort	25 seconds 559 milliseconds

4 Related Work

There are many works related to comparing sorting algorithms, each with it's own specifics and advantages. Some of them are:

- Adhikari, Pooja., "Review On Sorting Algorithms A comparative study on two sorting algorithms." Mississippi State, Mississippi (2007)
- Raza, Muhammad Ali, et al. "Comparison of Bubble Sort and Selection Sort with their Enhanced Versions."
- Grover, Deepti, and Sonal Beniwal. "Performance Analysis of Merge Sort and Performance Analysis of Merge Sort and Quick Sort: MQSORT."
- Alkharabsheh, Khalid & Alturani, Ibrahim & AlTurani, Abdallah & Zanoon, Dr.Nabeel. (2013). Review on Sorting Algorithms A Comparative Study. International Journal of Computer Science and Security (IJCSS)
- Blelloch, G. E., Leiserson, C. E., Maggs, B. M., Plaxton, C. G., Smith, S. J., & Zagha, M. (1991, June). A comparison of sorting algorithms for the connection machine CM-2. In Proceedings of the third annual ACM symposium on Parallel algorithms and architectures (pp. 3-16).
- Bharadwaj, A., & Mishra, S. (2013). Comparison of Sorting Algorithms based on Input Sequences. International Journal of Computer Applications, 78(14).

5 Conclusions and Future Work

To summarise, in this paper I analysed different approaches to sorting data on devices with low internet speed and processing power by imposing a delay on the receiving of data as well as sending each item to be sorted one by one, enabling optimisations based on having the input partially available.

I discovered that, despite being an *online algorithm*, insertion sort does not perform to expectations.

Instead, a combination of traditionally fast sorting algorithms combined with splitting the input into smaller chunks and processing them as they are available proved to be a better solution.

The findings of this paper can be further improved by testing for more possible methods of chunking the input, which could yield different results - better or worse.

Another area in which this paper can be expanded upon is by considering negative effects by using this approach without delays, assuming a very quick internet connection and very fast processing time, seeing if performance is greatly reduced compared to the traditional approach of sorting the entire list or stream at once.

References

[Aigner and West, 1987] Aigner, M. and West, D. B. (1987). Sorting by insertion of leading elements. *Journal of Combinatorial Theory, Series A*, 45(2):306–309.

[Astrachan, 2003] Astrachan, O. (2003). Bubble sort: an archaeological algorithmic analysis. *ACM Sigcse Bulletin*, 35(1):1–5.

[Hoare, 1962] Hoare, C. A. (1962). Quicksort. *The Computer Journal*, 5(1):10–16.

[Pardo, 1977] Pardo, L. T. (1977). Stable sorting and merging with optimal space and time bounds. *SIAM Journal on Computing*, 6(2):351–372.