Benchmark Tests on Improved Merge for Big Data Processing

Zbigniew Marszałek∗, Marcin Woźniak∗, Grzegorz Borowik‡, Raniyah Wazirali§, Christian Napoli†, Giuseppe Pappalardo† and Emiliano Tramontana†
∗Institute of Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice, Poland
Email: Zbigniew.Marszalek@polsl.pl, Marcin.Wozniak@polsl.pl
‡Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland
Email: G.Borowik@tele.pw.edu.pl
§Faculty of Engineering and Information Technology, University of Technology Sydney, Broadway Bld 11, Ultimo 2007, NSW, Australia
Email: raniyah.a.wazirali@student.uts.edu.au
†Department of Mathematics and Informatics, University of Catania, Viale A. Doria 6, 95125 Catania, Italy
Email: Napoli@dmi.unict.it, Pappalardo@dmi.unict.it, Tramontana@dmi.unict.it

Abstract—Computer scientists and engineers work with increasing amounts of information. These data are used for knowledge retrieval, data management decision support and so on. Sorting algorithms are important procedures that if efficiently composed and implemented can increase speed of data processing and decision correctness. Many various sorting algorithms and their modifications are applicable in large computer systems. However, as the computer architectures are more efficient with each release and the software is more complex with each version there is need to improve sorting methods applied for big data computation.

Index Terms—Software, Dependability, Workflow, Algorithms, Big Data, Merge Sort

I. INTRODUCTION

Increasing amounts of information we get each day are causing problems for common information systems. There are some super computers like these listed in top 500 best machines announced, i.e., in 2014 in New Orleans, USA. However, these types of computers are available not for common or business purposes but rather for scientific tasks. Therefore a typical company needs methods tailored for machines they use every day. This makes research on efficient methods and algorithms still important and useful. For this reasons examinations of novel or improved techniques for data acquisition, work flow optimization or system management can help to build efficient computer systems composed of machines that are widely used in various companies in the globe.

It is possible to give many examples of knowledge retrieval methods, that applied in processed data can help in decision making [1], [2], [3], [4], [5], [6]. Paralleled to this, in the common computer systems we need to have optimal traffic on the server that is managing our business network [7], [8], [9], [10]. This positioning and optimization can be efficiently solved by computational intelligence methods [11], [12], [13], [14], [15]. Finally, in the data base system, we need methods that are crucial for fastest possible ordering, sorting, feature extraction, data discretization, data retrieval or reconstruction [16], [17], [18], [19]. The new, improved algorithms for sorting large data sets were reported in [20], [21]. These examination were helpful to optimize computer systems managing for large data sets, also by refactoring [22], however, without application of any sophisticated machines or sophisticated systems. In this article we are about to present further improvement.

II. RELATED WORKS

Sorting is one of widely used methods in business, statistics and research too. Experimental examinations for fast sorting in various structures of the input data were reported in [23], [24], [25]. In [26] was proved that commonly applied methods are inefficient in many real life situations. These helped to extend possible solutions and improve sorting by composing dedicated algorithms with lowest possible complexity tailored for various sorting systems. Fully applicable methods for large data sets were described in details in [27], [28]. In this article we are presenting improved approach to merge sorting.

III. BIG DATA COMPUTATION

Computer systems that operate on big data have distributed structure. This means that each client machine that requests an information from the system firstly contacts managing machine. This unit is to organize requests queue and manage responses. It is also the unit that has contact with processing units that are operating on the data structure. Sample system is presented in Fig. 1. Big data processing units need most
efficient algorithms to improve system stability and speed. Improving algorithms features like time complexity and stability can to increase Quality of Service. To show which methods are most appropriate we test and compare their measures like CPU (Central Processing Unit) clock cycles (clock rate) and timing. CPU clock rate can be measured as cycles rate per second for performing basic operations. It is useful to estimate performance.

To make the benchmark tests comparable with similar research, it all must be done for proper number of samples. For the presented improved method, a 100 test input files representing each of classes: random arrangements, reversely sorted and correct order were sorted in the benchmark tests. To compare improvements for analysis were used similar statistical methods as in [27]. Measurement of the statistical mean for \( n \) items in sample set of \( a_1, \ldots, a_n \) describes formula

\[
\bar{a} = \frac{a_1 + \cdots + a_n}{n}.
\]

(1)

Possible deviation from it is describing statistic variability from the measurements. This helps to estimate performance. Each sampling is a discrete variable in the statistic sense. Thus in benchmark, it was calculated using formula

\[
\sigma = \sqrt{\frac{\sum_{i=1}^{n} (a_i - \sum_{i=1}^{n} a_i \cdot p_i)^2}{n}},
\]

(2)

what means that random sampling \( X \) can have \( N \) values \( a_1, \ldots, a_n \) with corresponding probabilities \( p_i \). It has an approximation that depends on the information we have about entire benchmark

\[
\sigma = \sqrt{\frac{\sum_{i=1}^{n} (a_i - \bar{a})^2}{n - 1}},
\]

(3)

where \( n \) is the number of elements in sample, \( a_i \) are random variables in sample, \( \bar{a} \) is average of all measured results. It’s value has an interpretation that with differences in performance this is also increased, what helps to depict comparison to other benchmarks. Similarly with decreased deviation from standard values the benchmarks will be more stable.

We can also test statistical stability of examined algorithm. Therefore we calculate coefficient of variation for benchmark samplings defined by formula

\[
\sigma = \frac{\sigma}{\bar{a}},
\]

(4)

where \( \sigma \) is standard deviation of benchmark sampling (defined in (3)), \( \bar{a} \) is expected benchmark value (defined in (1)). This coefficient represents diversity in examinations. Based on benchmark analysis of these measurements we can estimate statistical stability. The higher are values of deviation and variation the greater is potential instability of examine method in statistic sense. In presented benchmark tests each sampling is operation of sorting input arrangement. For the presentation of the improvements random sets of 100, 1000, 10000, 100000, 1000000, 10000000 and 100000000 elements were sorted, for which the resulting charts are presented.

A. Improved Merge - Theory and Practice

Software engineers often use dedicated mechanisms to analyze information or try to implement efficient methods tailored for the given tasks like big data analysis. Sorting algorithms dedicated for these type of data are of special interest. Therefore we propose improved merge sorting in a special version for big data.

The inspiration for the design of the improved method presented in this work is to reduce the number of comparisons performed during big data sorting (the constant reduction of computational complexity). This method should not have the critical data inputs that effectively prevent sorting as it is for other commonly implemented methods. Improved method manages to reduce sorting time in relation to classic merging by replacing selection of the smallest element with double merging of three data stacks. Improved method merges first two stacks into one ordered structure, and than merges the other two stacks. This method is not only less complex, but it’s implementation is also much easier. The proposed merge is pictured in Fig. 2. Algorithms must be defined in theory but also these assumptions need practical verification.

**THEOREM 1:** Presented triple merge sort algorithm with applied double merge sort has time complexity

\[
\frac{5}{3} n \cdot \log_3 n.
\]

(5)

**PROOF 1:** At the beginning of the triple merge algorithm, we merge two one element strings to have one string. Next we merge two elements string with one element to have three elements string. These operation for \( n \)-elements in the sequence is done within \( \frac{5}{3} n - 2 \) comparisons.

After this merging, for the next step two three elements strings are merged into one six elements string. Next we merge this string with tree elements string. This operation needs for
\( n \)-elements input less than \( \frac{5}{3} n \) comparisons (similarly for three
\( m \) elements inputs we make \( \frac{5}{3} \cdot 3m - 2 \) comparisons). Each
time as a result we get sorted sequence. As a conclusion we
state that \( n \) input elements we merge in \( k \) steps and for each
of them less than \( \frac{5}{3} n \) comparisons are done.

Finally we assume that output sequence has \( 3^k \) elements,
what gives the following assumption

\[
\min_{k \in \mathbb{N}} 3^k \geq n. \tag{6}
\]

After logarithm (6) we have

\[
\min_{k \in \mathbb{N}} \log_3 3^k \geq \log_3 n. \tag{7}
\]

Because the logarithmic function we can write that

\[
\min_{k \in \mathbb{N}} k \cdot \log_3 3 \geq \log_3 n. \tag{8}
\]

Therefore (8) equals

\[
\min_{k \in \mathbb{N}} k \geq \log_3 n. \tag{9}
\]

This gives conclusion about the number of operations
performed for sorting

\[
k = \lfloor \log_3 n \rfloor. \tag{10}
\]

Therefore average time to process input data

\[
T_{avg} = \frac{5}{3} n \cdot k = \frac{5}{3} n \cdot \log_3 n \approx 1.05n \cdot \log_2 n. \tag{11}
\]

Presented method was implemented in C++, what is shown in
Fig. 3. Implementation was examined in benchmark tests.

IV. EXPERIMENTAL RESULTS

Algorithms must be defined in theory but also these assumptions need practical verification. In benchmark tests we verify
expected processing time defined in section III-A.

We have tested improved merge on randomly chosen inputs
of 100 series for each frequency where various positioning
was verified, in the benchmark tests a quad core amd opteron processor 8356 8p was used. Research results are plotted in
Fig. 4 and Fig. 5. Analyzing Fig. 4 one may compare efficiency measured in CPU usage and time consumption. There is
pictured benchmark tests result for sorting using merge version defined in [27] in comparison to the presented improved
merge. One may see that improved method from section III-A is much faster. Improved merge is using CPU only marginally
what helps to improve overall system performance. This
version is very efficient in processing, but also computationally
less complex than other methods. If we compare results for
big data processing, the system behaves much faster without
loss of stability for various inputs. From Fig. 5 we conclude
that statistical stability of the benchmark tests for improved
merge is higher in comparison to other versions. Other merge
methods have changes in stability for various inputs. The
improved merge behaves stable for all the inputs no matter
what is the number of input data. This is very important for
big data, where processing performance repeatability is crucial
for the entire system workflow management. The benchmark
tests confirmed efficiency of the improved merge.
public void merge(array<int> a)
{
    int i, pa, pb, p0, p1, q1, q2, q3, t0, t1;
    int n = a->Length;
    array<int> b = gcnew array<int>(n);
    t = 1;
    while (t < n)
    {
        i = 0;
        t0 = 2*t;
        t1 = 3*t;
        while (i < n - t)
        {
            p1 = i + t;
            pb = 0;
            p0 = i;
            q1 = p1;
            q2 = i + t0;
            if (q2 > n) q2 = n;
            while (p0 < q1 && p1 < q2)
            {
                if (a[p0] <= a[p1])
                    b[pb++] = a[p0++];
                else
                    b[pb++] = a[p1++];
            }
            while (p0 < q1)
                b[pb++] = a[p0++];
            while (p1 < q2)
                b[pb++] = a[p1++];
            pa = 0;
            p0 = i;
            q3 = i + t1;
            if (q3 > n) q3 = n;
            while (pa < pb && q2 < q3)
            {
                if (b[pa] <= a[q2])
                    a[p0++] = b[pa++];
                else
                    a[p0++] = a[q2++];
            }
            while (pa < pb)
                a[p0++] = b[pa++];
            i += t1;
        }
        t = t * 3;
    }
}

Fig. 3. Merge method implemented in the distributed system

V. FINAL REMARKS

Research on software engineering advances and algorithms performance for big data processing help to improve workflows and stability of computer systems. If the system is configured to work with big data the fast and stable methods can improve overall performance, what has a positive impact on efficiency in energy management.

Presented in section III-A method outperforms other processing while applied to common computer systems. The final step in the project on big data processing will be to present a complete analysis of all tested methods. This will lead to compose a catalog of efficient algorithms for big data processing with their performance analysis.

ACKNOWLEDGEMENTS

This work has been partially supported by project PRISMA PON04a2 A/F funded by the Italian Ministry of University and Research within PON 2007-2013 framework.

REFERENCES

Fig. 5. Projection of the stability of sorting statistically measured for CPU and time consumption.


