Optimization of Position Finding Step of PCM-oMaRS Algorithm with Statistical Information

Ammar Balouch
Department of Computer Science, University of Rostock, Germany

Abstract: *The PCM-oMaRS algorithm guarantees the maximal reduction steps of the computation of the exact median in distributed datasets and proved that we can compute the exact median effectively with reduction of blocking time and without needing the usage of recursive or iterative methods anymore. This algorithm provided more efficient execution not only in distributed datasets even in local datasets with enormous data. We cannot reduce the steps of PCM-oMaRS algorithm any more but we have found an idea to optimize one step of it. The most important step of this algorithm is the step in which the position of exact median will be determinate. For this step, we have development a strategy to achieve more efficiency in determination of position of exact median. Our aim in this paper to maximize the best cases of our algorithm and this was achieved through dividing the calculation of number of all value that smaller than or equal to temporary median in two groups: The first one contains only the values that smaller than the temporary median and the second group contains the values that equal to the temporary median. In this dividing we achieve other best cases of PCM-oMaRS algorithm and reducing the number of values that are required to compute the exact median. The complexity cost of this algorithm will be discussed more in this article. In addition some statistical information depending on our implementation tests of this algorithm will be given in this paper.*

Keywords: *Median, parallel computation, algorithm, optimization, big data, evaluation, analysis, complexity costs*

Received June 11, 2015; *accepted October 18, 2015; published on line January 28, 2016*

1. Introduction

The goal of distributed aggregation is to compute an aggregation function on a set of distributed values. Typical aggregation functions are max, sum, count, average, median, variance, k^{th} smallest, or largest value, or combinations thereof. The most problematic aggregation is in the distributed computation of holistic aggregation function, in especially of distributed enormous data sets. Such data is like streaming data from network-sensors. The database community classifies aggregation functions into three categories: distributive, algebraic and holistic. Combinations of these functions are believed to support a wide range of reasonable aggregation queries.

PCM-oMaRS algorithm [2] shed a new light on the problem of distributed computation of exact median for general n distributed datasets. PCM-oMaRS solved the problem of exact median computation without using recursion or iteration steps and blocks determinate data only in one time by one step, actually by the last step if it is necessary. This algorithm consists of three major phases like as MapRaduce principle and depends on mathematical definition of median.

The reduction of blocking time of streaming data and of complexing cost is a recent growing interest in distributed aggregation, thanks to emerging application areas such as, e.g. data mining or sensor networks. Therefore we focus us on the optimization facilities of one step of PCM-oMaRS algorithm. With this optimization strategy

We can simply see that this strategy makes this step more efficient. Then, we show that the complexity of this algorithm is in worst case the worst case of a quick sort algorithm.

This article is organized as follows: In section 2 we list related works with a short summary. A short introduction of PCM-oMaRS algorithm is to find in section 3. Optimization strategy of PCM-oMaRs algorithm is presented in section 4. The complexity of PCM-oMaRS algorithm will be shown in section 5, in addition we discuss about some statistical information concerning implantation of PCM-oMaRS and at the end we summarize the most important points of the article in last section.

2. Related Works

Actually the research of distributed algorithm to determine the median is for more than 40 years active. This problem attracted many researchers. Blum *et al.* [3] presented a new selection algorithm named PICK. This algorithm proved a new lower bound for the cost of selection, and it was the first linear algorithm, Floyd and Rivest [7] presented a new selection algorithm which is shown to be very efficient on the average, both theoretically and practically, and Schönhage *et al.* [19] presented another algorithm which performs fewer comparisons on the worst case. Rodeh [16] considered the problem of computing the median of a bag of 2n numbers by using communicating processes, each having some of the numbers in its local memory. This algorithm described the distributive median problem as series of transformations. Marberg and Gafi [13] considered the problem of selecting the kth largest element in a set of n elements distributed arbitrarily among the processors of a Shout-Echo network. Chin and Ting [5] developed an improved algorithm for finding the median distributive. He embedded in the first part of its algorithm the Rodeh's algorithm, in the second part of its algorithm reduced the problem size by one quarter with three messages instead of reducing the problem size by half with two messages. With the third part of its algorithm resolved the problem of choosing the initiator. Santoro *et al.* [18] minimized with its algorithm the communication activities among the processors and considered the distributed k-selection problem. In [9] the existence of small core-sets for the problems of computing k-median and k-means clustering for points in low dimension and get an (1+ε)-approximation was shown. Kuhn *et al.* [12] presented a k-selection algorithm and proved that distributed selection indeed requires more work than other aggregation function. In this article has shown that the kth smallest element can be computed efficiently by providing both a randomized and a deterministic k-selection algorithm. And they considered the k-median clustering on stream data arriving at distributed sites which communicate through a routing tree. They proposed a suite of algorithms for computing (1+ε)-approximation k-median clustering over distributed data streams. The algorithms are able to reduce the data transmission to a small fraction of the original data. Feldman and Langberg [6] show a unified framework for constructing core-sets and approximate clustering for such general sets of functions.

In [21] an algorithm that produces (1+ε)α-approximation, using any α-approximation non-distributed algorithm as a subroutine, with total communication cost has been shown. There are many others works in this field [1, 4, 11, 14, 15, 17, 20, 22]. All of these researches have used the iteration, recursion or approximation in their steps. Garrigues and Manzanera [8] present a new grained distributed median randomized algorithm. This paper show that this algorithm is efficient on networks containing *O*(*N*) (up to *N*/4-1) processing units running in parallel. This algorithm focuses to reduce the number of elements to process after each iteration. Ivan *et al.* [10], present a structure of the median filter device. Median filters are widely used for smoothing operations in signal, speech, and image processing. The conveyer device with parallel implementation of algorithms is widely used for quick median filtering. Median calculation on the basis of completely parallel devices is redundant. Using streaming-conveyor devices reduce this redundancy. The calculation of the median in this devices is reduced to the execution pair wise sequence comparison and permutation of numbers.

3. PCM-oMaRS Algorithm

In this section we will give a short introduction of PCM-oMaRS algorithm [2]. At first we represent the general mechanism of this algorithm through the following sub section.

3.1. Illustration of PCM-oMaRs Algorithm Mechanism

Figure 1 shows an abstract of the mechanism of our algorithm cleared by one sequence. In this figure we can see that the finding of position of exact median depending on the value of *sclenD* and of *scgnD* in which will be known to which direction must the position be moved from the position of the temporary median to achieve the position of exact median.

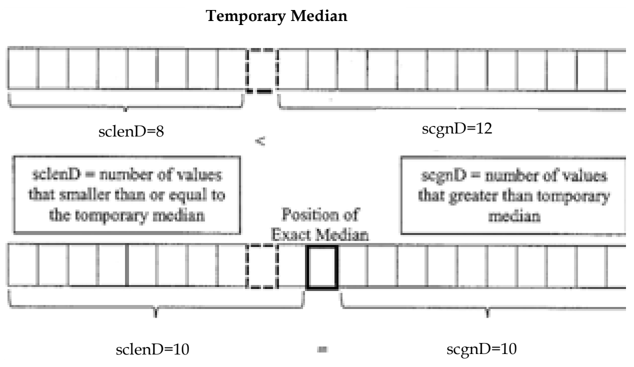
Temporary Median
sclenD=8
<
scgnD=12
sclenD = number of values that smaller than or equal to the temporary median
Position of Exact Median
scgnD = number of values that greater than temporary median
sclenD=10
=
scgnD=10

Figure 1. Abstract of PCM- oMaRS algorithm mechanism.

3.2. PCM-oMaRS Algorithm Steps

The PCM-oMaRS algorithm consists of two sub algorithms. The steps are represented in Algorithm 1:

```
Algorithm 1: PCM-oMaRS (Di)

#Execute Subalgorithm 1
CandidateFinding (Di)
#Execute Subalgoithm 2
PositionFinding(Di, MedT)
return MedE

Subalgorithm1 CandidateFinding(Di)
    #Get minimum, maximum and median of Di (Step 1)
   foreach (Di)
   {
     Get MinDi, MaxDi, MedDi;
   }
   #construct multiset Ord (Step 2)
   Ord = {MinDi, MaxDi, MedDi; i =1,..,n}
   #sorting Ord
    Sort(Ord)
    #compute Median of Ord (Step 3)
    MedT=Med(Ord)
   return MedT
```

If the size of the sequence *Ord* is EVEN then the temporary median value is the first/ second value of the two middle values of *Ord* instead to compute the average of both middle values. Med_T must stay one of the existing values not new calculated values.

The following steps belong to the second subalgorihtm "position finding". These steps will be optimized in the following section of this paper.

```
Subalgorithm 2: PositionFinding (Di, MedT)
 # Sum numbers according MedT (Step 4)
 foreach (Di) {
 # counting numbers that are greater than MedT
 cgnDi=count ({Ni | Ni >MedT})
 # counting numbers that are smaller than or equal to
 MedT
 clenDi = count ({Mi | Mi ≤ MedT})
   # sum all cgnDi and all slenDi
   scgnD = sum (cgnDi)
   sclenD = sum (clenDi)-1
 }
 # Calculate the exact median MedE (Step 5)
 if (scgnD < sclenD)
   {
   MedLP = (sclenD - scgnD)/ 2;
 # Construct multiset LtD and sorting it descending
   LtD= {maximum MedLP largest {Ni | Ni ≤ MedT}}
   MedE =LtD[MedLP]
 } else if (scgnD > sclenD)
 {
 MedRP = (scgnD - sclenD)/ 2;
 # Construct multiset GtD and sorting it ascending
 GtD= {maximum MedRP smallest {Ni | Ni >MedT}}
   MedE =GtD[MedRP]
 } else MedE = MedT
 return MedE
```

For more detailed information about PCM-oMaRS algorithm see [2].

3.3. Example of Application

Let us have the following Datasets after ordering:

$D1= (3, 5, 11, 27, 30)$

$D2= (1, 7, 27, 27, 29)$

$D3= (10, 18, 27, 30, 32)$

In the first step we have to get Med_T a temporary median. Med_T is the median of *Ord*-set in which contains ordering all Minimal, Maximal and Median values of each Dataset.

The minimal, maximal and median values for each dataset are as following:

$MinMaxMedD1=\{3, 11, 30\}$

$MinMaxMedD2=\{1, 27, 29\}$

$MinMaxMedD3=\{10 ,27, 32\}$

Then the ordered set of them is:

$Ord=(1, 3, 10, 11, 27, 27, 29, 30, 32)$

And the median of *Ord* is the temporary median, *MedT* and equal to 11.

$Med_T=27$

Now we start with step 4. We calculate *cgnD*1, *cgnD*2 and *cgnD*1 (number of all values that greater than Med_T of each dataset respectively) as following:

$cgnD1=|\{30\}|, cgnD2=|\{29\}|, cgnD3=|\{30, 32\}|$

Now the number of all values that greater than Med_T in all datasets is *scgnD*:

$scgnD = 1+1+2=4$

On the other hand, we calculate now the number of all values that smaller than or equal to Med_T:

$clenD1=|\{3, 5, 11, 27\}|$

$clenD2=|\{1, 7, 27, 27\}|$

$clenD3=|\{10, 18, 27\}|$

That means, the number of all values that smaller than or equal to is *sclenD*:

$sclenD = (4+4+3)-1=10$

(-1) is because we do not need to take the temporary median itself into consideration. That means, actually instead of counting *clenD*2 as:

$clenD2 = |\{1, 7, 27, 27\}|$

We count *clenD*2 as:

$clenD2 = |\{1, 7, 27\}|$

Now we have the case:

$sclenD > scgnD$

We start now with step 5 of PCM-oMaRS algorithm:

$MedLP = ((sclenD-scgnD)/ 2)$

$MedLP = (10–4)/2 = 3$

Now we know that the position of exact median is to find in the left side of temporary median in 3 positions. We get now *LtD*, the sequence that contains maximum 3 largest numbers from each dataset smaller than or equal to Med_T outer Med_T self.

Max 3 *greatest Nrs* of $D1\leq 27(Med_T)$are 27, 11, 5

Max 3 *greatest Nrs* of $D2\leq 27(Med_T)$are 27, 7, 1

Max 3 *greatest Nrs* of $D3\leq 27(Med_T)$ are 27, 18, 10

Then, *LtD* is as following:

$LtD=(27, 27, 27, 18, 11, 10, 7, 5, 1)$

The exact median is now:

$MedE=LtD[3]= 27$

The value 27 is really the exact median because the number of all values that greater than 27 is equal to the number of all values that smaller than or equal to 27 minus 1 (the median itself).

In the following sections we focus on the optimization of PCM-oMaRS algorithm and its complexity cost, in addition we discus some statistical information of our implementation.

4. Optimization of PCM-oMaRS Algorithm

The single step in which we can implement an optimization is the step 5 of PCM-oMaRS algorithm. In this step we can apply other computation strategy. This strategy makes a brilliant optimization of this algorithm. This Optimizations step will illustrate in the following sub section.

4.1. Position Finding Optimization

In step 5 of PCM-oMaRS algorithm we find the relationship of position determination in tow cases. The first one is if *sclenD* greater than *scgnD*. The second case is if *sclenD* smaller than *scgnD*. For the second case we could not find any possibility of optimization but for the first case we have found that we can optimize this case with clever steps. *scgnD* is the number of all values that greater than the temporary median and *sclenD* is the number of all values that smaller than or equal to temporary median. Now, we make the change. Instead to calculate the number of all values that smaller than or equal to temporary median, we calculate the number of all values that smaller than the temporary median and the number of all values that equal to the temporary median in *sclnD* and *scenD* respectively. The second subalgorithm is optimized as following:

Algorithm 2: Optimized PCM-oMaRS (Di).

```
#Execute Subalgorithm 2
CandidateFinding(Di)
#Execute Subalgoithm 3
  OptimizedPositionFinding(Di, MedT)
return MedE
Subalgorithm 3OptimizedPositionFinding (Di,MedT)
  # Sum numbers according MedT (Step 4)
  foreach (Di)
 { #counting numbers that are greater than MedT
 cgnDi=count ({Ni | Ni >MedT})
 # counting numbers that are smaller than MedT
 clnDi = count ({Mi | Mi < MedT})
# counting numbers that are equal to MedT
 clnDi = count ({Mi | Mi = MedT})
# sum all cgnDi, all clnDi and cenG
scgnD = sum (cgnDi)
sclnD = sum (clnDi)
scenD = sum (cenDi)-1
}
# Calculate the exact median MedE (Step 5)
if (scgnD < (sclnD+scenD))
{
MedLP = ((sclnD+scenD)-scgnD)/2;
 if (MedLP ≤ scenD)
  MedE= MedT
else
{ # Construct multiset LtD and sorting it descending
 LtD= {maximum MedLP-scenD largest {Ni | Ni <
MedT}}
 MedE =LtD[MedLP-scenD]
 } else if (scgnD > (sclnD + scenD))
{
 MedRP=(scgnD-(sclnD+scenD))/2;
 # Construct multiset GtD and sorting it ascending
 GtD= {maximum MedRP smallest {Ni | Ni >MedT}}
 MedE =GtD[MedRP]
}
else MedE = MedT
return MedE
```

In this case we can see simply that if the position *MedLP* smaller than or equal to *scenD* then we do not need to make any other computations to get the exact median. In this optimization step we make this case belong to the best cases of PCM-oMaRS algorithm.

In this optimization strategy is to remark too that if the *MedLP* greater than *scenD* then instead of sending all sequences of datasets to send back maximum *MedLP* values smaller than or equal to temporary median, the algorithm sends all datasets to send back maximum (*MedLP–scenD*) values smaller than temporary median. This step of our optimization reduce too the number of values that will be sorted to get the exact median. The other cases of this step remain unchanged as following: The last case of this step is simplest one and presents one of the best cases of this algorithm.

In the following section we represent the new cases map after executing the presented optimization strategy.

4.2. Example of Application with the Optimization

Let us now apply our optimization of the previous example to clear the efficiency. Let us start with step 4 the calculation of *cgnD*1, *cgnD*2 and *cgnD*1 stay unchanged as following:

$$cgnD1=|\{30\}|,\ cgnD2=|\{29\}|,\ cgnD3=|\{30, 32\}|$$

Where the number of all values that greater than Med_T in all datasets is *scgnD*:

$$scgnD = 1+1+2=4$$

Now, we calculate now the number of all values that only smaller than Med_T:

$$clnD1 = |\{3, 5, 11\}|,\ clnD2 = |\{1, 7\}|,\ clnD3=|\{10, 18\}|$$

That means, the number of all values that smaller than is *sclnD*:

$$sclnD=(3+2+2)=7.$$

On the other hand, we compute the number of all values that only equal to Med_T:

$$cenD1= |\{27\}|,\ cenD2=|\{27, 27\}|,\ cenD3=|\{27\}|$$

That means, the number of all values that smaller than is *sclnD*: $scenD = (1+2+1)-1=3$.

Now we have the case: $(sclnD+scenD) > scgnD$, we start now with step 5 of PCM-oMaRS algorithm

$$MedLP=(((sclnD+scenD)-scgnD)/2)$$
$$MedLP=((7+3) - 4)/ 2 = 3$$

Now, we know that the position of exact median is to find in the left side of temporary median in 3 positions.

Now before we get *LtD*, the sequence that contains maximum 3 largest numbers from each dataset smaller than or equal to Med_T differencing Med_T self. We compare the *MedLP* with scenD. If $MedLP \leq scenD$ then we do not need to do any computation more because the temporary median has the same value as the exact median. That means:

$$MedE= MedT = 27$$

Based on this example we can see the important role of this optimization. This optimization provides best cases of our algorithm. The next section illustrates these cases with the optimization.

4.3. Optimized PCM-oMaRS Cases Map

In Figure 2, it was clarified that the best case of our algorithm is executable by many cases. The first two cases are applicable if the number of values that greater than the temporary median equal to the number of values that smaller than or equal to the temporary median for both cases of total size of all datasets.
The other cases is to find if the position number smaller than or equal to the number of values that equal to the temporary median for the case that the number of values that greater than the temporary median smaller than the number of values that smaller than or equal to temporary median in the both cases of total size of all datasets. If the total size of all multisets is an even number and the difference equal to 1 then this case is too a best case of PCM-oMaRS algorithm because in this case the temporary median is one of the two middle values of exact median.

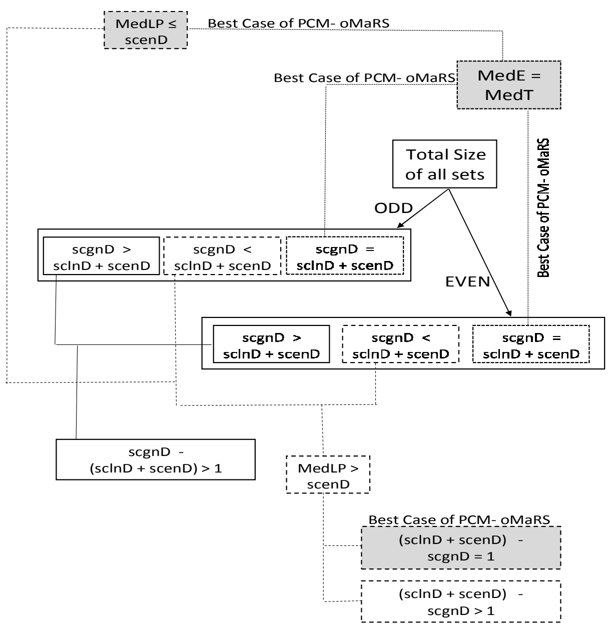

Figure 2. Abstract map of optimized PCM-oMaRS cases.

In these cases, the PCM-oMaRS algorithm does not need more to apply algorithm 2 completely, because the temporary median in this case is the required exact median. In other words that means, after computing the position of the exact median we do not need to apply any operation anymore to achieve the required result.

4.3. Complexity and Statistical Study of PCM-oMARS Algorithm

In this section we discuss the cost of complexity of our algorithm and give basic statistical information of our implementation experiments.

4.4. Complexity of PCM-oMaRS

The complexity of parallel algorithms depends on three major classes of costs. The first class is the communication cost because parallel algorithms access data storage in distributed connected nodes. The second class is the local execution cost. Generally the more expensive operations execute in local level the more efficient for the complexity of the parallel algorithms. The third one is the cost of the parallel algorithm in the global level.

In most cases the communication cost plays the most expensive role in the total complexity cost. Therefore by renouncing the use of iterations or recursions of communications the PCM-oMaRS algorithm and its optimized version achieved the maximal reduction of steps that play a crucial role with the communications costs. Excessively with our algorithm only basic operation and optimized sort algorithm will be required to apply in both local and global levels.

These classes of costs are cleared in the following table:

Table 1. Abstract of total cost of PCM- oMaRS.

Cost / Operation	Local Execution	Communication	Global Execution
Basic Operation	+	-	+
General Operation	+	-	-
Sorting	+	-	+!
Communication	-	+!	-
!: it is only one time and if it is required			

In other words, the global and local execution costs are in best case $O(1)$ and the worst case $O(n.logn)$ and the communication cost is in the worst case the usually applied network system communications costs and it is counted only once. In relation to this idea the blocking of data will be executed only one time and only if it is necessary.

4.5. Statistical Study

Depending on tests of implementation of PCM-oMaRS algorithm we have received some important results. We have carried out over 55000 tests with Eclipse-Parallel-Luna on intel(R) Core(TM)2 Duo CPU P9600 processor with 8GB RAM. These results are organized in a short statistical study as following.

We have classified the required data to getting the exact median in 4 classes depending on our results. We have found that in the worst case of applying PCM-oMaRS algorithm we need to receive only 21.31% of all values in all datasets and in the best case we do not need to receive any value (0% of all values). In 35.63% of all tests we do not need to receive any data (0% of all values) to get the exact median and in 43.47% of all tests we need 0.01-4.99 % all values. Belong to the sector 5-9.99% all values 14.76% of all tests and in 7.14% of all tests we need 10-21.31% all values.

This statistical information is represented in the following Table:

Table 2. Statistical information.

Required Data	% all tests
0%	35,63
(0,01-4,99) %	43,47
(5,00- 9,99) %	14,76
(10,00- 21,31)%	07,14

This table is represented in the following diagram:

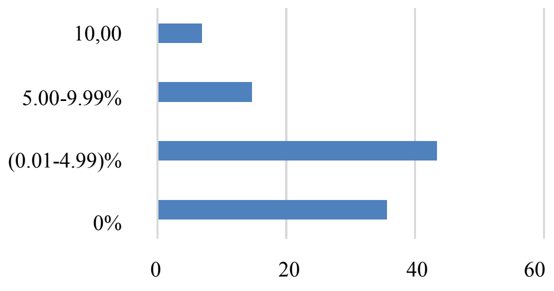

Figure 3. Statistical study of implmention of PCM- oMaRS.

5. Conclusions

The research points of this field "median computing of distributed datasets" divided into two main directions The first one cares on the approximation methods. The other one focuses on the computation of the exact median with usage of iterative or recursive steps. We have shown that we can compute the exact median with clever steps depending on the calculation of the position of the exact median without needing to apply iterations or recursions to get the value of the exact median. That means, PCM-oMaRS algorithm guarantees the maximum reduction of median computation steps. Too, instead applying blocking of the required data by the beginning an execution of an algorithm, the data may be blocked only in one non iterative or recursive step with the execution of our algorithm and if it is necessary.

In this article we have shown that the most computation of our algorithm is calculated in the local nodes (computers), basic operations and operation with efficient complexity will be executed in the master computer (global one). That means in other word, the costs of complexity of our algorithm is computed through the common communication costs and local execution costs like all other algorithms in addition only the cost of an efficient sort algorithm in step 5. In our experiments we have proved that the execution of our algorithm can be more effective in the local execution too, if we divided the local dataset that contains enormous values in many local datasets.

We have implemented this algorithm by Java with two different input possibilities. The first one is with manually targeted inputs to test extreme cases of values distributions and the other one is random inputs to be able to check all possible cases with the passage of time. We have tested the implementation of our algorithm with more than 40000 cases, some of these depended on the manually targeted inputs and the rest were in relation to the random inputs. In each case, the number of datasets is different, and each dataset includes many different values.

Acknowledgment

We would like to thank our colleague in Database and information systems department at university of Rostock for their notes. Moreover, we would like to express our gratitude IIE-SRF/ USA organization for its support scientists at risk and its scholarships.

References

[1] Anagreh M., Samsudin A., and Omar M., "Parallel Method for Computing Elliptic Curve Scalar Multiplication Based on MOF," *the International Arab Journal of Information Technology*, vol. 11, no. 6, pp. 521-525, 2014.

[2] Balouch A., "PCM-oMaRS Algorithm: Parallel Computation of Median-Omniscient Maximal Reduction Steps," *American Research Journal of Computer Science and Information Technology*, vol. 1, no. 1, pp. 1-11, 2015.

[3] Blum M., Floyd R., Pratt V., Rivest R., and Tarjan R., "Time Bounds for Selection," *Journal of Computer and System Sciences*, vol. 7, no. 4, pp. 448-461, 1973.

[4] Chen W., *Linear Networks and Systems*, Belmont, CA: Wadsworth, 1993.

[5] Chin F. and Ting H., "An Improved Algorithm for Finding the Median Distributively," *Algorithmic a,* vol. 2, no. 5, pp. 235-249, 1987.

[6] Feldman D. and Langberg M., "A Unified Framework for Approximating and Clustering Data," *in Proceedings of the Annual ACM Symposium on Theory of Computing*, New York, USA, pp. 569-578, 2011.

[7] Floyd R. and Rivest R., "Expected Time Bounds for Selection," *Communications of the ACM*, vol. 18, no. 3, pp. 165-172, 1975.

[8] Garrigues M. and Manzanera A., "Exact and Approximate Median Splitting on Distributed Memory Machines," *in Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2012.

[9] Har-Peled S. and Mazumdar S, "On Coresets for k-means and k-median Clustering," *in Proceedings of the Annual ACM Symposium on Theory of Computing*, New York, USA, pp. 291-300, 2004.

[10] Ivan T., Dmytro P., and Ivan I., "Parallel Algorithms and VLSI Structures for Median Filtering of Images in Real Time," *International Journal of Advanced Research in Computer Engineering and Technology*, vol. 3, no. 8, pp. 2643-2649, 2014.

[11] Jia L., Lin G., Noubir G., Rajaraman R., and

Sundaram R., "Universal Approximations for TSP, Steiner Tree and Set Cover," *in Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pp. 386-395, 2005.

[12] Kuhn F., Locher T., and Wattenhofer R., "Distributed Selection: A missing Piece of Data Aggregation," *Communications of the ACM*, vol. 51, no. 9, pp. 93-99. 2008.

[13] Marberg J. and Gafi E., "An Optimal Shout-Echo Algorithm for Selection in Distributed Sets," *Technical Report University of California, Los Angeles, Computer Science Department*, 1985.

[14] Negro A., Samtoro N., and Urrutia J., "Efficient distributed selection with bounded messages," *in Proceedings of IEEE Transactions of Parallel and Distributed Systems*, pp. 397-401, 1997.

[15] Peleg D., *Distributed Computing: A Locality-Sensitive Approach*, Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2000.

[16] Rodeh M., "Finding the Median Distributively," *Journal of Computer and System Sciences*, vol. 24, no. 2, pp. 162-166, 1982.

[17] Santoro N., Scheutzow M., and Sidney J., "On the Expected Complexity of Distributed Selection," *Journal of Parallel and Distributed Computing*, vol. 5, no. 2, pp. 194-203, 1988.

[18] Santoro N., Sidney J., and Sidney S., "A Distributed Selection Algorithm and its Expected Communication Complexity," *Theoretical Computer Science*, vol. 100, no. 1, pp. 185-204. 1992.

[19] Schönhage A., Paterson M., and Pippenger N., "Finding the median," *Journal of Computer and System Sciences*, vol. 13, no. 2, pp. 184-199, 1976.

[20] Yao Y. and Gehrke J., "The Cougar Approach to in-Network Query Processing in Sensor Networks," *ACM SIGMOD Record*, vol. 31, no. 3, pp. 9-18. 2002.

[21] Yingyu L., "Distributed k-Median/ k-Means Clustering on General Topologies," *in Proceedings of the Advances in Neural Information Processing Systems*, pp. 1995-2003, 2013.

[22] Zhang Q., Liu J., and Wang W., "Approximate Clustering on Distributed Data Streams," *in Proceedings of IEEE 24th International Conference on Data Engineering*, Cancun, Mexico, pp. 1131-1139, 2008.

Ammar Balouch is married and has three children. He received his PhD from University of Rostock in Germany in 2006. He is currently a researcher at the Chair of Database and Information Systems in the Institute of Computer Science at the University of Rostock. He is active in and concentrating on his research topics: Analysis and evaluation of big data, and parallel computation of statistical functions. He managed and supervised several IT projects. Focuses were on data mining and data warehousing technologies in the application areas Trading and Construction Equipment. He is a supporter and cofounder of organizations of human support and rights.