Simple Parallel Sorting Model for Embarrassingly Parallelism Environment

Yustinus Eko Soelistio and Pujianto Yugopuspito

***Abstract*—One challenge in data mining come from data sorting problem which needs enormous computation power when it comes to big data. While many machines and methods have been developed to tackle this problem, most solutions are either too expensive or too complex to be implemented in typical companies, non-profit organization, and individual researchers. One technique to tackle these difficulties is using volunteer computing model. This paper suggests one adaptable sorting model that can be used in such volunteer computing environment. The model is generic so it can be adapted directly in the existing application without the need to install additional agent or server. The model was tested using 750,000 random integer data. The tests were run 300 times each employing four nodes and three nodes with different number of buckets and number of data in a bucket, apparently those are faster than sequential merge sort.**

***Index Terms*—Data mining, parallel, sorting, volunteer computing.**

I. Introduction

Sorting is still one of the functions in data mining which use a great deal of computation resources, especially when it comes to big data. Many machines and methods were designed to tackle with this problem with some great success. Unfortunately, most of those machines and methods are relatively expensive or complex to be implemented in typical company, social organization, and individual researcher hence hindering them to make the most of available big data.

This paper explores an alternative model for implementing sorting algorithm using volunteer computing model. Volunteer computing has been suggested quite some times ago by [1]-[6] as an alternative to solve time consuming applications. This type of computing works by breaking data into smaller bits and distributes them into network of volunteers. These volunteers process the data with their own unused computer resources thus potentially make it more efficient in terms of cost and computer utilization.

In this paper, we elaborate one of the characteristic of volunteer computing is that it runs on embarrassingly parallel connectivity. Though this trait is proven to be an advantage for some applications, it can turn problematical for sorting algorithm to solve some arbitrary live data. Therefore this paper focuses on solving this problem. The model proposed in this paper is a generic sorting algorithm model that can be used in an embarrassingly parallel connectivity situation.

Manuscript received June 14, 2013, revised August 12, 2013.

Y. E. Soelistio is with the Faculty of ICT, Universitas Multimedia Nusantara, Tangerang, Indonesia (e-mail: yustinus.eko@ umn.ac.id).

P. Yugopuspito is with Research and Development Computer Lab., Universitas Pelita Harapan (e-mail: pujianto.yugopuspito@uph.edu).

II. Related Works

Previous works have been conducted by the author [7], [8].Those papers suggest divide and conquer model to solve travelling salesman problem (TSP) and geographic simulation involving large number of data in volunteer computing environment. The techniques used in those models are using pivot point to split and merge the data, and categorizing data by clients' specifications.

In TSP problems, map is segregated into several mini-maps. These mini-maps connected to each other by assigning one or more points in map as pivot point. The numbers of points in mini-map are adjusted with clients' specifications. Clients with higher specification are given bigger maps with more points than lower-end clients.

This paper uses merge-sort algorithm as groundwork for building the model. Many parallel sorting algorithms had been proposed using similar divide and conquer method [9]-[11]. The model proposes in this paper modify divide and conquer technique used in the previous researches so it can be applied in volunteer computing environment.

In this paper we introduce a new sorting algorithm design, by means of data segregation which can be run on embarrassingly parallel connectionism, and create a proof of concept for the design.

III. Algorithm Design

The algorithm can be run on multiple core or multiple computers. This paper will use the term nodes to indicate cores and computers. The main problem in sorting some arbitrary live data in embarrassingly parallel environment is the non-consecutive finishing time on each node. For example, there is no guarantee that node P_x will finish earlier than node P_{x+1}. Therefore master node P_M will not know whether data in $P_x < P_{x+1}$ or not. Moreover if the data given from P_M to volunteer's nodes P_x are random then data in each P_x will only be sorted locally in P_x.

The algorithm in this model approaches the problem just like merge-sort where it divides the data into atomic member and sorts them by merging each part sequentially. However, this model does not split database into single data. It splits the database into some set of data so each set will have a maximum m members to be sorted individually in each node P_x then merge them back into one. Value of m can be a single value thus applied in all P_x, or vary according to specification of P_x like proposed in [7], [8]. This paper used single value of m to test the model.

If all the data are distributed in a manner such that $|P_{1..x-1}| < m$ and $|P_x| \leq m$ then at least $x-1$ nodes will have equal load among them.

To guarantee that all members of P_x is in the right sequence to all members of P_{x+1}, this model classify all data as buckets. These buckets are some set of array or list which have index 1 to k. Each bucket has members b range from $low_k \leq b \leq high_k$ where $(low_k, high_k)$<$(low_{k+1}, high_{k+1})$ for ascending sort and $(low_k, high_k)$>$(low_{k+1}, high_{k+1})$ for descending sort. This paper use interval method to determine k , though any other statistical method such standard deviation can also be used.

All buckets have maximum m number of members. Each of these buckets will be sent to P_x to be sorted. If all arbitrary members b in B_{k-1} is smaller than b in B_k then it is guaranteed that

$$\forall(b)_{sorted} \ni B_{k-1} < \forall(b)_{sorted} \ni B_k \qquad (1)$$

Hence by mapping P_x with corresponded B_k, then P_M will be able arrange B_k back in the right sequence.

A. Model and Complexity

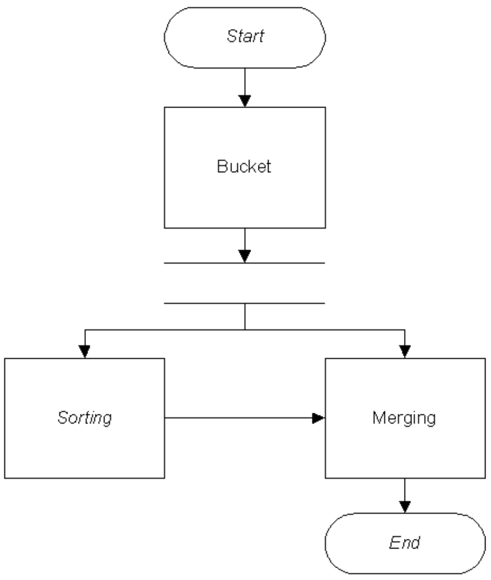

Fig. 1. Three steps sorting model.

The model uses three steps modules to run this sorting algorithm like illustrated in Fig. 1. First step, P_M surveys the data to check whether it is sorted or not. When it is not then P_M creates buckets by calculating the optimal value of m and k. The complexity of this step is $O(n)$, where n is the number of data. To put the data into buckets, P_M send parts of the data to P_x which categorize it into buckets. These buckets then sent to P_M to be finalized and distributed.

Second step is the actual sorting process. P_M will send and map B_k to P_x to be sorted. If the algorithm sort database A by distributing arbitrary data $\forall(a_i) \in A$ to nodes P_x , where $1 < x \leq y$ and y is the maximum available nodes, then the complexity will be

$$\begin{cases} 1 \leftarrow y \geq k \\ \frac{k}{y} \leftarrow y < k \end{cases} \qquad (2)$$

Complexity shown in (2) is the complexity of the parallel model, not the sorting process itself. The sorting process happened in P_x has its own complexity, depends on what algorithm it uses. For example, if all P_x use merge-sort then the complexity of the second step will be (2) multiply by $m \log m$. If $m \log m$ is substituted by q then in general the complexity will be

$$\begin{cases} q \leftarrow y \geq k \\ \frac{kq}{y} \leftarrow y < k \end{cases} \qquad (3)$$

The third step is when P_M received bucket B_k from P_x and save it in appropriate order in database A'. Every B_k put in A' will be in the ordered sequence like in (1), thus finishing the sorting process. The complexity in this step varied between the best case $O(1)$ and worst case $O(n)$, mostly depends on data structure of A'. Indexed data structure like array will have $O(1)$, and non-indexed like one-linked list will have $O(n)$.

The first step must be run sequentially since it has to classify all data into buckets. However, the second and the third step in this model can run as parallel thus the total complexity become

$$\begin{cases} n + q \leftarrow y \geq k \\ n + \frac{k}{y}(1 + q) \leftarrow y < k \end{cases} \qquad (4)$$

Mathematically, (4) is less complex compared to original merge-sort when $n > 55$ with $k, y \geq 2$.

B. Model for Live Data

For sorting live data, this model employs marking. First it marks the beginning and the end of the data and called it database A. This database will be sorted using the three steps modules described above and produces database A'.

Any new data will be sorted only after A is fully sorted. Since this paper uses interval method to define k then the model will calculate new value k' when the new data are breaching the lowest and the highest value of A. The flow of algorithm for this model can be seen in Fig. 2.

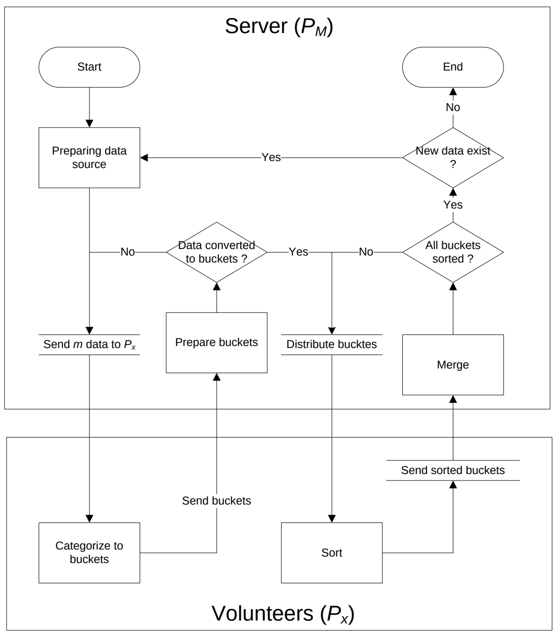

Fig. 2. Algorithm flow.

IV. Testing

The model was tested using 750,000 random integer data. The tests were run 300 times each employing four nodes and three nodes with different m and k values. First test was using four nodes with $m = 2{,}500$ and $k = 10$. Second test used four nodes with $m = 250$ and $k = 10$. Third test used three nodes with $m = 2{,}500$ and $k = 10$. All nodes used merge sort to sort the data and connected in a LAN environment, as shown in Fig. 3. Tabel I shows the average results.

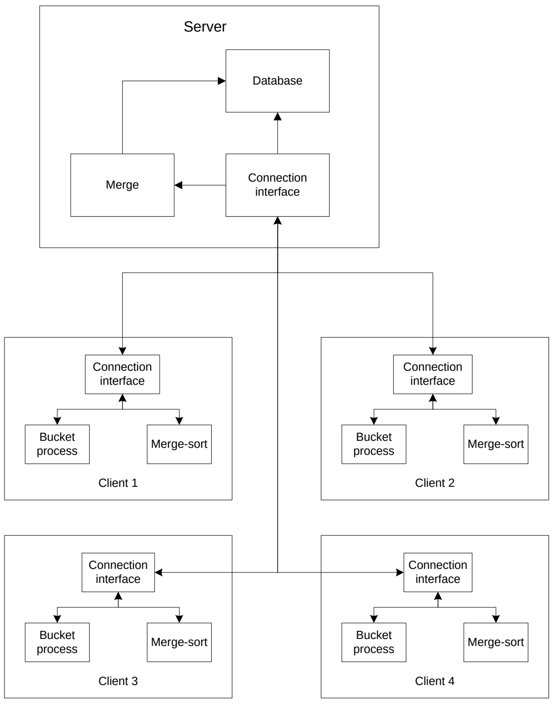

Fig. 3. Volunteer computing arrangement.

TABLE I: AVERAGE TEST RESULTS

Test 1	
Bucket:	0.0080747
Sorting & Merging:	0.1362573
Total:	**0.144332**
Test 2	
Bucket:	0.0095788
Sorting & Merging:	0.1024624
Total:	**0.1120412**
Test 3	
Bucket:	0.0109169
Sorting & Merging:	0.1552226
Total:	**0.1661395**
Sequential Merge-sort:	**0.2086206**

Although this was only a trivial test, there are some patterns shown. First, the more nodes used, the less time required to finish the first step. Second, the time needed depends on value of m. All tests are also faster then sequential merge-sort. However, this test was conducted on LAN environment thus the system did not experience lost connection or glitch. Further test on internet connectionism is required to verify its performance over sequential system.

Comparing to our previous results [7], [8], this model is more flexible in terms of distributing and handling data. In this model, each client can have different method to sort the data. Furthermore, unlike pivot point method in [7] and [8], bucket size and data size for clients in this model can be easily adjusted between each cycle.

V. CONCLUSION

This paper explores one model to process arbitrary live data in an embarrassingly parallel connectivity situation. The model is a three steps method which can be adapted by modifying its parameters. This model had been tested, although further validations are still needed. Further test should use non-ideal connection environment for better simulate volunteer computing environment such as the use of internet connection and potential malicious volunteers. A bigger and more varies data set are also needed for further confirmation.

The model proposed in this paper can be customized by modifying number of bucket, maximum data in a bucket, and sorting algorithm in clients (k, m, and q). k can be compute using statistic method such as interval and standard deviation. m will directly influence work load in q since m is the amount of data that will be sorted by an algorithm with q complexity. In the next attempt, it is possible to develop a machine learning algorithm to obtain the best combination of k, m, and q.

REFERENCES

[1] L. F. G. Sarmenta, “Volunteer comp,” Ph.D. Dissertation, Dept. Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2001.

[2] D. P. Anderson, “Boinc: A system for public-resource computation and storage,” in *Proc. 5th IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4-10.

[3] C. Christensen, T. Aina, and D. Stainforth, “The challenge of volunteer comp. with lengthy climate model simulation,” in *Proc. First International Conference on e-Science and Grid Computing*, 2005, pp. 8-15.

[4] D. P. Anderson, E. Korpela, and R. Walton, “High-performance task distribution for volunteer computing,” in *Proc. First International Conference on e-Science and Grid Computing*, 2005, pp. 196-203.

[5] N. Z. C. Fülöp, “A desktop grid approaches for scientific computation and visualization,” Ph.D. Dissertation, Dept. Comp. and Information Sci., Norwegian University of Science and Technology, 2008.

[6] S. Yi, E. Jeannot, D. Kondo, and D. Anderson, “Towards real-time volunteer distributed comp,” in *Proc. The 11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2011, pp. 154-163.

[7] Y. E. Soelistio and P. Yugopuspito, “Disain implementasi TSP pada lanscape Jakarta menggunakan volunteer computing,” *Seminar Nasional Teknologi Informasi*, Universitas Tarumanegara, 2012, pp. C2-10–C2-17.

[8] Y. E. Soelistio and P. Yugopuspito, “Penggunaan volunteer computing di perguruan Tinggi,” *Journal Ilmiah Ilmu Komputer*, vol. 9, no 2, March 2013.

[9] D. R. Helman and D. A. Bader, “A randomized parallel sorting algorithm with an experimental study,” *Journal of Parallel and Distributed Computing*, pp. 1-23, 1998.

[10] M. C. Albutiu, A. Kemper, and T. Neumann, “Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database System,” in *Proc. VLDB Endowment*, vol. 5, no. 10, 2012, pp. 1064-1075.

[11] S. Odeh, O. Green, Z. Mwassi, O. Shmueli, and Y. Birk, “Merge path–parallel merging made simple,” in *Proc. IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, 2012, pp. 1611–1618.

Yustinus Eko Soelistio received his bachelor degree in information engineering and master degree in management from Univesitas Pelita Harapan, in 2005 and 2008 respectively. Currently he teaches information system and engineering at Multimedia Nusantara University. His research interests are parallel computing, machine learning, and green computing.

Pujianto Yugopuspito received engineer degree in mechanical engineering in 1991 from Unversitas Gadjah Mada, Indonesia; Master degree in software techniques for computer aided engineering in 1996 from Cranfield University, United Kingdom; and Doctor of Engineering in computer science and communication engineering, specialization in software methodology in 2001 from Kyushu University, Japan. Since 2004 he is with Universitas Pelita Harapan. His research interest includes software engineering, formal methods, high performance computing, and mobile applications. He is a member of IEEE and IAENG.