Using Data Variety for Efficient Progressive Big Data Processing in Warehouse-Scale Computers

Hossein Ahmadvand and
Maziar Goudarzi, *Senior Member, IEEE*

Abstract—Warehouse Scale Computers (WSC) are often used for various big data jobs where the big data under processing comes from a variety of sources. We show that different data portions, from the same or different sources, have different significances in determining the final outcome of the computation, and hence, by prioritizing them and assigning more resources to processing of more important data, the WSC can be used more efficiently in terms of time as well as cost. We provide a simple low-overhead mechanism to quickly assess the significance of each data portion, and show its effectiveness in finding the best ranking of data portions. We continue by demonstrating how this ranking is used in resource allocation to improve time and cost by up to 24 and 9 percent respectively, and also discuss other uses of this ranking information, e.g., in faster progressive approximation of the final outcome of big data job without processing entire data, and in more effective use of renewable energies in WSCs.

Index Terms—Big data, sampling, efficiency, resource allocation, order of processing

1 Introduction

Warehouse scale computing [1] views the entire computing resources in a data center as a large computer employed by internet-scale service providers for massive scales. Efficiency, in terms of performance, cost, and energy, is definitely a determining factor for user satisfaction as well as cost reduction in such expensive computing facilities. MapReduce [4] is a programming paradigm introduced by Google for such massive computers, and has since gained widespread use for other big data applications as well. Improving efficiency of such jobs in WSCs is thus of interest for both the user of the big data job as well as the owner of the facility. Prior art has explored efficient resource assignment [7], hardware acceleration [12], [14], approximate computations [6] as well as other techniques [10] for more efficient MapReduce processing on WSCs, but to the best of our knowledge data variety, in terms of the difference in significance of various chunks of data in determining the final output of the big data job, has not been previously explored for efficiency of big data jobs in WSCs. For example, imagine the vote-counting process in an election; if the votes in some states have higher influence on the election outcome, it makes sense to assign more resources to count them for faster approximation of the final result. Note that *efficiency* in this example is the *approximation accuracy after unit time*, but it can be defined otherwise in other examples.

We show that indeed various same-sized portions of data have different influences on the final outcome of computation. Thus, if a low-overhead technique can sort the data portions based on their significance, one can start processing from more important data portions, or assign more resources (by any available mechanism such as VM sizing) to more important data, or assign scarce renewable energy to process more important data, or other uses indeed depending on what efficiency metric she has in mind.

• *The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Azadi Avenue 11365-11155, Iran.
E-mail: ahmadvand@ce.sharif.edu, goudarzi@sharif.edu.*

*Manuscript received 9 July 2016; revised 10 Nov. 2016; accepted 29 Nov. 2016. Date of publication 0 . 0000; date of current version 0 . 0000.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2016.2636293*

Debatably, in some cases such as progressive approximation as above example, this may require to split the conventionally monolithic MapReduce job, e.g., Hadoop, into a number of smaller MapReduce sub-jobs each of which combines previously reduced (key, value) pairs from the reduce stage of previous sub-job—see the arrows in Fig. 1—with the pairs generated by its own map stage. In such case, at the end of each sub-job, an approximate outcome is generated which progressively converges to the final outcome. Such multi-stage computation introduces data and compute overheads in terms of (i) storing reduced (key, value) pairs of each sub-job until the shuffle phase of subsequent job, and (ii) changing the load of each shuffle and reduce phase to incorporate above additional pairs with the intermediate data of current sub-job. Obviously these overheads should also be considered, experimentally or analytically, in such cases.

To implement this multistage MapReduce, we used Apache Spark and Scala language: each input data block is converted to a Resilient Distributed Dataset (RDD). After the first stage of multi-stage MapReduce, the intermediate RDD is stored in the storage and is combined with other RDDs in later stages using Scala constructs such as *join* and *union*. Spark Streaming could have also been used with modifications for the same purpose, but we did not use it since it better suits fine-grained micro-batching whereas our technique deals with rather large blocks.

2 Related Work

Approximate computing [6], [8] explores the tradeoff between the accuracy of the results vs. efficiency. In such sense, our proposal can also be viewed as a progressive approximation of the output: at the end of each MapReduce sub-job, partial outputs are provided that approximate the final output. Processing more important data first, provides a faster approximation of the final output by processing unit volume of data.

Authors in [13] deal with the portions of code and variables that more significantly affect the final outcome of processing. They provide techniques and tools to automatically analyze the source code of the computation and assess the influence of variables and code portions in output quality so as to tradeoff quality for energy efficiency or time. In contrast, we deal with the input of the program, and aim to determine the portions of data that more significantly influence the final output.

Conventional MapReduce architecture is changed in [3] to produce progressive outputs to obtain "early returns" and approximations. They provide a pipelined version of Hadoop, named Hadoop Online Prototype (HOP), and discuss opportunities and advantages of such framework especially in large jobs, as well as challenges introduced by such pipelining. In case of a progressive computation as in [3], when multiple jobs are to be run one after the other in the pipeline, the choice of data splits to feed the pipeline at each round affects rationality of the changes observed in the progressively generated outputs. Mechanisms are proposed in [2] for data sampling and for defining progress intervals so that progressive computation becomes scalable to multiple back to back jobs. The sampling here refers to selection of data blocks whereas we sample from within each data block to assess relative significance of the blocks. None of the above works rank the data splits for more effective progressive computation, nor they use this information for better resource assignment as we do.

Other literature exists on algorithms for progressive computation [9], [11], and [16]] where large data space is pruned progressively to find the answer. Frameworks are also provided for progressive analytics [5] to help programmers focus on their analysis algorithms and to streamline implementation of progressive analytics pipelines. These are complementary to our techniques and can be combined with it for wider use cases in more complex analytics.

1556-6056 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

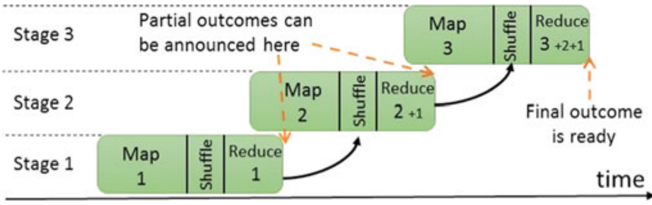

Fig. 1. Overall view of multistage MapReduce processing we propose. Reduced (key, value) pairs are passed to later shuffle stages.

3 Motivation

We ran three publicly available benchmarks, namely WordCount, InvertedIndex, and Grep, from BigDataBench benchmark suite [15] on 3 public datasets, namely IMDB, Gutenberg, and Quotes, used intact, but divided into 0.5 GB portions. We generated all possible permutations of these portions, then ran the benchmark on each sequence of portions while taking note of the partial outcome incrementally produced after processing each portion, and compared them against the final outcome. Obviously, the partial outcomes gradually converge to the final one, but the speed of this convergence differs depending on the sequence employed. Fig. 2 shows the results for WordCount; horizontal axis shows the percentage of data processed, and the vertical axis represents normalized *distance between the partial and final outcomes;* the definition of this *distance* obviously depends on the big data application. We used below definitions for our three benchmarks:

Grep: difference between the number of lines in which the match was found.
InvertedIndex: difference between the size of the output index files (as a measure of the number of indexes created.)
WordCount: difference between total number of words counted.

As Fig. 2 shows, there is a large gap between the worst and the best sequence of processing the data portions; e.g., after half of data is processed in WordCount, the worst sequence is still 71.9 percent far from the final outcome whereas the best one is only 28 percent far. Similar behavior is observed for Grep and InvertedIndex benchmarks respectively showing 62.3 and 73.9 percent difference between the best and the worst sequences at 50 percent of total data. An mHealth workload (chest movements sensed by sensors on human body) and a financial workload (investments of companies in different states) also showed 48 and 43 percent differences respectively. As an intuitive analogue, imagine vote-counting in an election: the sequence of counting the votes collected in different regions determines the speed of approaching the election final outcome.

We argue that the above observation is due to the *difference in significance of each data portion*, and it can be used in a number of ways for higher efficiency in big data processing on WSCs. We define *significance measure* for each data portion in each application similar to above definitions of distance; i.e., the number of matching lines, created indexes, and counted words in the data portion for Grep, InvertedIndex, and WordCount respectively. Fig. 3 shows the distribution of significance measures of data portions for each application which obviously highly differs even among data portions from the same source.

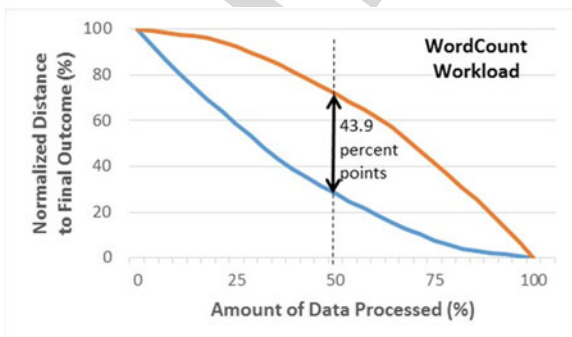

Fig. 2. Results of processing the data portions in various orders for WordCount benchmark for 0.5GB as size of data portions.

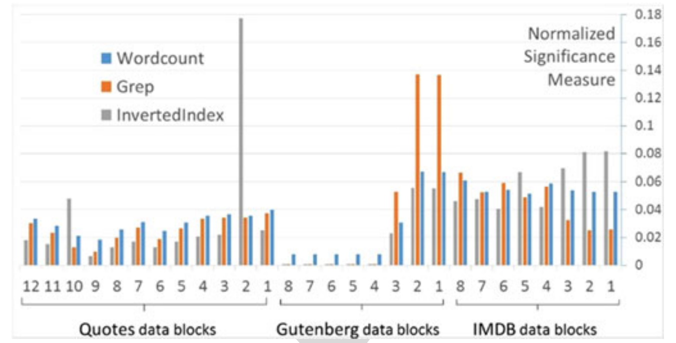

Fig. 3. Distribution of significance measures vs. data portions.

4 Proposed Approach

Our proposal, Fig. 4, consists of two steps: a fast *ranking step* where data portions are quickly evaluated and ranked based on their estimated significance, and a *multi-stage MapReduce step* (see Fig. 1) where the actual resource allocation and big data processing is performed on these data portions. Note that the per-stage operations depend on the optimization in mind—see Section 6—but it would typically consist of processing the data portions in descending order of significance.

The ranking step requires an efficient light-weight mechanism to avoid high overhead. We employed a simple sampling mechanism and a simple measure of significance: for a given size PS (Portion Size) for data portions, we take SS (Sample Size) number of 1 MB samples from each data portion uniformly distributed over the portion, and process it for the job in question, then we compute the significance measure defined for the job, and sort the data portions accordingly. We also examined randomly distributed sampling, but the results were only marginally different. As the *significance measure*, we used the definitions given in Section 3 above. The choices of PS and SS values are discussed in Section 5.

There are three parameters to tune here: (i) SS: the number of samples per data portion, represents a tradeoff between the per-portion overhead in the ranking step versus the accuracy of the ranking; (ii) PS: data portion size, determines the granularity of the ranking; and (iii) NumStages: the number of computation stages in the multi-stage MapReduce computation, determines the granularity of progression toward the final outcome.

Since a MapReduce job is originally defined as a monolithic computation, a key point here is whether it is practically possible to run multistage MapReduce jobs or not, and what overheads it

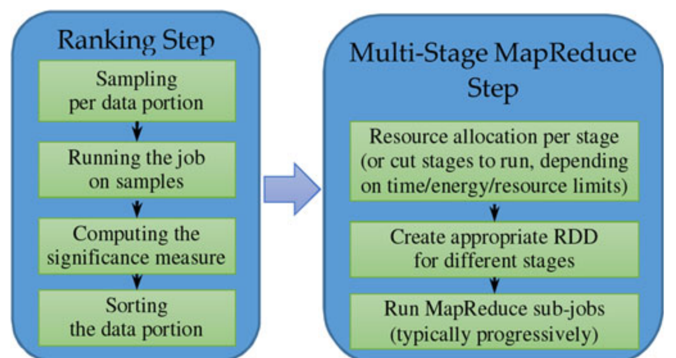

Fig. 4. Overall flow of our proposal

IEEE Proof

TABLE 1
MSE of Sequences Compared to the Best Possible One

Mean Square Error (MSE)	Ours vs. Best	Random vs. Best	Worst vs. Best
Grep	3.696	160.082	263.285
InvertedIndex	6.037	136.302	288.655
WordCount	3.641	74.751	172.575

imposes. It is important to note that the notion of (key, value) pairs makes it inherently possible to break a larger MapReduce job into smaller ones: each small MapReduce job by definition produces a list of (key, value) pairs each with a unique key during its reduce phase; these pairs can be added to the pairs produced by the map phase of the subsequent small MapReduce sub-job, and passed to its reduce phase by the same shuffle operation to obtain new progressively more complete (key, value) pairs. This is repeated until the end of the original large MapReduce job. At the end of each stage, however, one might need to run a Merge function to produce the partial outputs corresponding to the computations completed till then. Consequently, there are three overheads to analyze here: (i) passing reduce outputs of each stage to the next one, (ii) reduction of additional (key, value) pairs from previous MapReduce stage, and (iii) the merge possibly required at the end of each stage.

5 Experimental Results

We evaluated 3 benchmarks from BigDataBench suite [515] as in Section 3. The datasets comprised of 14 GB of data from 3 different sources. Experiments were run on an intel Core-i7 4-core CPU at 2.8 GHz with 4 GB of RAM.

Effectiveness of our ranking: We set $\mathrm{PS} = 0.5$ GB and $\mathrm{SS} = 5$ in this set of experiments. Note also that we have set NumStages = 28 equal to the number of data portions in these experiments to achieve the highest resolution; thus, at each stage, only one data portion is MapReduced. The sequence obtained by our ranking step is compared to the best, the worst, and conventional random sequence in Table 1 regarding distances to the final outcome—see vertical axis of Fig. 2. Our ranking step reduces Mean Square Error (MSE) by over 20x compared to the random sequence. Table 2 gives three major data points of the compared sequences.

Sensitivity to SS and PS: We tried other data portion sizes (PS = 0.5 GB, 1 GB, 2 GB) and sample sizes (SS = 1, 5, and 10 samples per data portion) and compared the results in terms of computation overhead as well as the accuracy of the ranking they produce. Since each sample is a 1MB data, the overhead is simple to compute in terms of data amount to process in the ranking step; this overhead spans from 0.05 percent (SS = 1, PS = 2 GB) to 2 percent (SS = 10, PS = 0.5 GB). The time it takes to process this data is however higher than that since it incurs the overhead of setting up MapReduce sessions for small sizes of data (one run per aggregate samples of each data portion). Table 3 shows this overhead. In return for this overhead, the ranking step returns a sorted list of data portions. We defined the RD (Ranking Difference) measure in Eq. (1) to compare the lists that different rankings produce:

$$RD = \left\{ \sum_{i=1}^{N=Number\ of\ portions} \left((N-i) \times |X_i - Y_i| \right) \right\} \Bigg/ \left(N * \sum_{i=1}^{N} X_i \right) \quad (1)$$

Where X_i and Y_i are the significance measures of the ith element of the sorted list of data portions obtained respectively by processing whole data portions (i.e., the golden list), and by our sampling and ranking method described above. The ideal case is $\mathrm{RD} = 0$. The RD values are also reported in Table 3 which shows that for this dataset, $\mathrm{SS} = 5$ is enough even for $\mathrm{PS} = 2$ GB. Note that choice of SS and PS depends on the dataset. For a case where large variation of significance measure across data portions is observed in the dataset, smaller PS and larger SS is needed to obtain more uniformity within a data portion as well as better differentiation among different portions. Thus, intra- as well as inter-portion variations should be sampled for appropriate decision.

Table 3 shows the ranking overhead. Compared to the best total processing time by our approach as reported in Table 4 for WordCount, these overheads are very small: 0.1 to 5.9 percent depending on the sample size, and specifically 0.8 for the $\mathrm{PS} = 2$ GB and $\mathrm{SS} = 5$ case above. For InvertedIndex and Grep, the ranking overheads were respectively 0.55% and 0.85% of total processing time in our experiments for $\mathrm{PS} = 2$ GB and $\mathrm{SS} = 5$ compared to the fastest processing time obtained by our technique, similar to WordCount case in Table 4.

Data-Variety-Aware Resource Allocation. With close-to-the-best ranking of data portions obtained by our light-weight method, resource allocation can be done proportional to the significance of the data portions to be processed at each stage to obtain higher efficiency. Other uses are described in Section 6.

TABLE 2
Data Points of Compared Sequences

	Amount of data processed (%)	Normalized distance to final outcome (%)			
		Best sequence	Our sequence	Random sequence	Worst Sequence
Grep	25	43.87	43.87	82.69	97.76
	50	18.83	19.85	58.12	81.16
	75	2.24	2.24	25.45	56.13
Inverted Index	25	41.32	41.32	77.16	98.09
	50	14.14	14.85	52.86	85.86
	75	1.91	1.91	12.63	58.68
Word Count	25	58.56	58.56	72.54	92.27
	50	28.07	29.56	49.61	71.93
	75	7.73	8.13	24.14	41.44

TABLE 3
Overheads and Ranking-Differences of Our Sampling

Benchmark	SS / PS	Time spent for ranking (s)			RD: Ranking Difference		
		1	5	10	1	5	10
WordCount	0.5GB	1.88	8.71	17.12	0.079	0.0034	0.0021
	1GB	0.96	4.33	8.63	0.158	0.0068	0.0014
	2GB	0.51	2.19	4.33	0.083	0	0
InvertedIndex	0.5GB	196.06	1084.79	2230.66	0.3423	0.0481	0.0673
	1GB	98.67	527.78	1084.79	0.6847	0.0961	0.0197
	2GB	56.05	269.40	527.78	7*10-7	4*10-7	1*10-7
Grep	0.5GB	0.54	2.66	5.54	0.2217	0.0397	0.0214
	1GB	0.28	1.33	2.66	0.4435	0.0793	0.1228
	2GB	0.17	0.67	1.33	0.0476	1*10-4	0

TABLE 4
Comparison of Variety-Oblivious and -Aware Resource Allocations

	Resource Allocation	Cost (relative)	Time (s)	Total time(s)	Total cost
Data-variety-oblivious	1st half on S1	0.3	205.2	320.2	119.06
	2nd half on S2	0.5	115		
	1st half on S2	0.5	116	317	118.3
	2nd half on S1	0.3	201		
Our data-variety-aware approach	Upper half on S1	0.3	247.5	348.5	124.75
	Lower half on S2	0.5	101		
	Upper half on S2	0.5	136	287.8	113.5
	Lower half on S1	0.3	151.8		
Naïve MapReduce	All data on S1	0.3	398.8	398.8	119.64
	All data on S2	0.5	245	245	122.5

IEEE Proof

Assuming two server configurations, S1 with 4 GB RAM and 4-core CPU, S2 with double those resources, and with normalized costs of respectively 0.3 and 0.5 as per Amazon EC2 cloud, we compare different choices of resource allocation in a case with PS = 0.5 GB and NumStages = 2. Data-variety-oblivious scenario chooses data portions randomly but our data-variety-aware one first sorts data portions by our ranking step above and then allocates resources to process upper and lower halves of the list.

As Table 4 shows for WordCount application, by assigning more resources to more significant data, both total time as well as cost are respectively reduced by 9.2 and 4.1 percent compared to the best alternative possible by the oblivious counterpart. Similarly, for Grep and InvertedIndex, our approach saves run-time from respectively 180.4s and 74334.0s by 6.3 and 23.6 percent, and reduces cost from 66.7 and 27269.9 by 2.8 and 9.0 percent; i.e., we save more at higher costs and time. Compared with naïve MapReduce, where entire data is processed in parallel on one type of machine, our technique takes 5.1 and 7.3 percent lower cost respectively for naïve case on S1 and S2 machines, and even finishes 27.8 percent faster than naïve case on S1. Although naïve MapReduce on S2 takes 14.8 percent less time than us, note that our progressive computation has already provided a partial output at time 136s (i.e., 44 percent earlier than naïve MapReduce) corresponding to 70.5 percent of the final output; further note that MapReduce is basically used for batch jobs where time is of less concern than cost.

6 Discussion of Usages

We showed that different data portions have different significances in determining the final outcome of a big data job. In addition to the resource allocation in WSC case shown above, this phenomenon can benefit the user in a number of ways: (i) if limitations in time or resources force us to process only part of entire data, two interesting problems can be defined: (a) "In a given time limit, or under energy limit such as a solar-equipped WSC, which data portions should be processed so as to get the closest result to the golden outcome achievable by processing entire data?", and (b) "Which data portions should be processed so as to get to a given distance from the golden final outcome in the minimum amount of time or resource?"; (ii) if entire data is to be ultimately processed, but we are interested in more rapidly approaching the golden final outcome (e.g., vote-counting example), the number of stages employed is of interest since it determines the amount of inter-stage overhead as well as the number of partial outcome announcements (c.f. election vote reports case above); (iii) stages can also overlap in time if enough resources are available.

7 Summary and Conclusion

We showed the significance and use of the differences among data portions in a MapReduce computation to improve time and cost efficiency in WSC. We presented a quick and effective mechanism to rank the portions before processing for above purpose. This opportunity in such big data computations in WSCs can be used to more time- and cost-efficiently process the entire data, or process partial data with optimized approximation in case of resource, time, or energy limitations.

To use our technique, one only needs to define a significance measure to compare data portions, use a sampling or other lightweight mechanism to assess data portions, and a multistage MapReduce paradigm, e.g., Spark, for progressive computation. We showed the value this approach can provide, and discussed a number of ways we envision as its use cases.

Many avenues exist to further this work including sampling methods, categorization of MapReduce applications, variety-aware resource allocation techniques as well as progressive approximate computation: in cases where the statistical distribution of data is known, other sampling methods such as importance sampling and Cochran sampling can be used to set the number of samples for a target confidence interval; Big data workloads can be more deeply examined to classify the workloads that can, or cannot, benefit from the technique proposed in this work; Efficient variety-aware resource allocation algorithms are needed to take best advantage of the observation introduced in this paper; Finally, approximate and progressive computations mentioned in Section 6 are also important further research to do.

Acknowledgments

This research is supported by grant number G930826 from Sharif University of Technology. We are grateful for their support.

References

[1] L. A. Barroso, J. Clidaras, and U. Hölzle, *Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd Ed. San Rafael, CA, USA: Morgan & Claypool Publishers, 2013.

[2] B. Chandramouli, J. Goldstein, and A. Quamar, "Scalable progressive analytics on big data in the cloud," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 1726–1737, 2013.

[3] T. Condie, N. Conway, P. Alvaro, and J. M. Hellerstein, "MapReduce online," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, May 2010, pp. 21–21.

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Operating Syst. Des. Implementation*, 2004, pp. 10–10.

[5] J.-D. Fekete and R. Primet, "Progressive analytics: A computation paradigm for exploratory data analysis," *CoRR*, vol. abs/1607.05162, 2016. [Online]. Available: http://arxiv.org/abs/1607.05162

[6] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, "ApproxHadoop: Bringing approximations to MapReduce frameworks," in *Proc. 20th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2015, pp. 383–397.

[7] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of MapReduce jobs for big data applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 10, pp. 2720–2733, Oct. 2015.

[8] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surveys*, vol. 48, 2016, Art. no. 62.

[9] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–42, 2005.

[10] I. Polato, R. Ré, A. Goldman, and F. Kon, "A comprehensive view of Hadoop research—A systematic literature review," *J. Netw. Comput. Appl.*, vol. 46, pp. 1–25, 2014.

[11] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. 27th Int. Conf. Very Large Data Bases*, 2001, pp. 301–310.

[12] K. O. Toshimori Honjo, "Hardware acceleration of hadoop MapReduce," in *Proc. Int. Conf. Big Data*, 2013, pp. 118–124.

[13] V. Vassiliadis, et al., "Towards automatic significance analysis for approximate computing," in *Proc. IEEE/ACM Int. Symp. Code Generation Optimization*, 2016, pp. 182–183.

[14] B. W. Yi Shan, Jing Yan, Yu Wang, Ningyi Xu, and Huazhong Yang, "FPMR: MapReduce framework on FPGA, A case study of RankBoost acceleration," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2010, pp. 93–102.

[15] L. Wang, et al., "BigDataBench: A big data benchmark suite from internet services," in *Proc. Int. Symp. High Performance Comput. Archit.* 2014, pp. 488–499.

[16] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progessive computation of the min-dist optimal-location query," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 643–654.

IEEE Proof