Performance Evaluation and Prediction of Parallel Big Data using MOA

Chanintorn Jittawiriyanukoon[#1], Vilasinee Srisarkun[*2]

[#] Graduate School of eLearning, Assumption University, Samutprakarn Province 10540, Thailand
[1] chanintornjtt@au.edu

[*] Martin de Tours School of Management and Economics, Assumption University,
Samutprakarn Province 10540, Thailand
[2] vilasineesrs@au.edu

***Abstract*— Currently, massive data has been compiled and examined at explosive volumes. The vast volume of data stream structured by current technological sources in up-to-the-minute society has amplified at an incredible pace, stimulating processing capacity and data curation. The traditional event-driven simulation models to investigate huge data can be no longer used but the paradigm to decision-making is now taken into all activities in the society, business applications, and scientific accounts. The overhead of data collection, redundant storage and processing improvement cost are on the rise for big data applications. In addition, there are technical aspects such as data inconsistency, redundancy, privacy, scalability, time-series and unusefulness [1]. Parallel processing is an essential method particularly for quantifying scale and time-series experiments. Simulation results for several datasets fit prediction results. Speedup and cost effective analysis will be considered as performance metrics as well.**

Keyword- Big Data Analytics, Massive Online Analysis (MOA), Parallel Processing, Performance Evaluation, Prediction

I. INTRODUCTION

The enormous volumes of data created by up-to-date high technologies in recent society introducing complications in processing cost and data classifications. On the other hand, analytical tools for cleansing and normalizing data streams are incapable to execute on real-time basis such explosive data size [5]. In addition, big data manipulation may exhibit volatile memory-paging, spatial locality and data flow control. On the other hand, the advent of parallel processing has disclosed the extraordinary processing power of PUs (processing units) to speed up data-explosive calculation much more as period elapses by. This research provides the evaluation of parallel processing for big data curation by using MOA (Massive Online Analysis) [6]. Another simulation tool, QuickSort, to evaluate genetic applications with parallelism has been proposed in [11]. Before executing experiments that implicate big data with parallelism, it is critical to split data sources into n subtasks. A previous research [5] exposes pros and cons of applying the graphic processors for computation. However, it is not resourceful unless the experiments are taking both partitioning and re-assembling time into consideration. Thus, in this research, analytical performance evaluation and prediction model are proposed by aiming the parallel processing architecture of MOA simulation as well as accounting both partitioning and re-assembling time. This paper is structured as follows. Firstly, section two outlines parallel big data description. Section three explains prediction method then section four describes simulation results and analysis. Lastly, section five draws conclusion and remarks the future work.

II. OVERVIEW OF PARALLEL BIG DATA

A deep learning is constructed up of machine learning's processors. A parallel processing [2] is made up for supporting several independent processing units accordingly. Clustering and distributed processing which covered the field of parallel processing are still on-the-fly to be scalable, cost-effective, uncomplicated to code and can be of off-the-shelf processing systems. High-speed processing system indulges elevated performance computing by splitting the heavy data into pieces then being processed among the parallel elements in the cluster. These elements (nodes) are scalable but cost sensitive and well interconnected. They are precisely designed to handle massive applications on parallel fashion [7],[8]. Traditional supercomputers are found on mentioned architecture as well.

The big data-parallel model is preventive structure per se. It is high level coding in which the processing intercommunication needs to be specified explicitly. It is preventive due to the specification of data-parallelism exhibition. For this situation, big data parallelism is a universal critical not only in programming paradigm but also in cost-effective performance analysis. Despite the difference between the programming paradigm and performance analytical model, the data-parallel design cited in previously is practical. Only that big data must be decomposed in order to convenient the processing capacity, intercommunication among nodes, load balancing, and reassembly. Data-parallel model will address the decomposition phase straightforwardly, by yielding explicit partitions which are relevant to a parallel computing grain, in which subtask will take on

service at individual processing unit. At this point it is to identify a partition which pinpoints situated concurrency. To route subtask to any processors indicates how they are to be transferred over parallel processing units based upon queuing discipline and thus checks load balancing in each directions. The latter can be inferred by the compiler.

III. PREDICTION METHOD

A queuing network involves several servers which offer services of some mechanism to incoming customers. Customers who experience all servers busy will have to wait for their turn at queue lines attached to servers. Therefore, these waiting lines are called queuing systems. There are numerous examples which can be defined as queuing systems, such as communication networks, banking service, computer systems, production systems and etc. A queuing model can be characterized by six components. Firstly, the customer's arrival rate will be described by interarrival time and the distribution function. In general practice customer will land regarding to a Poisson fashion (exponential interarrival time). Customer may arrive individually or substantially. An example of bulky arrivals is the immigration office at the border where tourist's passports must individually be controlled. Secondly, the behavior of customers may be patiently waiting in line or impatiently leaving after a certain while. Especially, call centers may face the hang-up customers when they cannot wait until next operator avails, whether or not they will try calling again. Thirdly, it is the service time which is independent of, in general, exponential distribution. Service time may depend on queue size. For example, the processing hours of an individual customer at the bank can be amplified if the number of waiting customers is too outsized. Fourthly, the service discipline explains how each customer will receive a service, one on one or bulk. Many patterns are assumed based upon the order in which customers come in, such as first come first serve, last come first serve, random sequence, priorities, or time-sharing. Fifthly, the service capacity can be either a single server or several servers handling the customers. Lastly, the waiting place can be limited to number of waiting customers in queue. For example, in computer network, only limited packets can be buffered at a router. The proper buffer space is an essential in the network design.

Analytical model used for prediction in this research will be brought up. First, the big data can be decomposed by n autonomous data sets in which are called tasks. The parallel processing system comprises of $n+2$ servers, which are a server for partitioning (*PS*), a server for merging (*MS*) and n parallel processing devices. Big data arrives at *PS* with an exponential service time. At *PS*, big data will spawn n tasks evenly. Spitted tasks are referred to sibling. All siblings will proceed directly at zero delay time to parallel processing facility as shown in Fig 1.

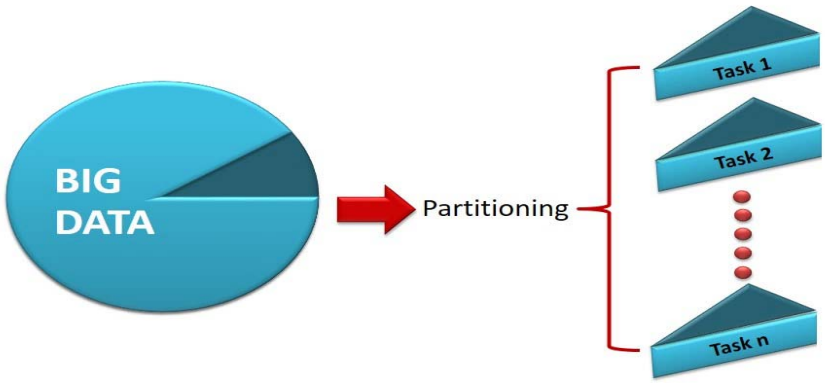

Figure 1. Big Data Splitter

Siblings are autonomously processed. Tasks with priority can help lower the processing time on parallel system as presented in [10] but subsequently it will reflect longer reassembly time. As sibling's processing is finished from parallel network, it leads to *MS* at once. The sibling must wait in the buffer for all siblings' completion before merging is active. Once all sibling completion is achieved, they are reassembled into big data. At this point, the execution process may keep iterating. The period of time from big data entering *PS* to big data leaving *MS* is referred to residual time (*RT*) as listed in Eq 1. Prediction method introduced in [4] is applied for collecting the performance measurements such as processing time and residual. These results will be used for comparing in subsequent section of the paper

The *M/M/n* system depicted in Fig 2. is applicable for the employment of MOA. Big data arrives with an exponential interarrival time (λ) and the service time distribution follows exponential function (mean = μ). Big data splitter based on [3] will be measured and assumed to be an average *B* and merger based on the same is assumed to be an average *M*. Results from MOA simulation will be compared to those from prediction.

$$RT = \max(S_1, S_2, ..., S_n) + B + M \quad (1)$$

Where B is average time for partitioning, S_n is time spent at the n^{th} server of the n^{th} task and M is average time for reassembly after the accomplishment of all tasks.

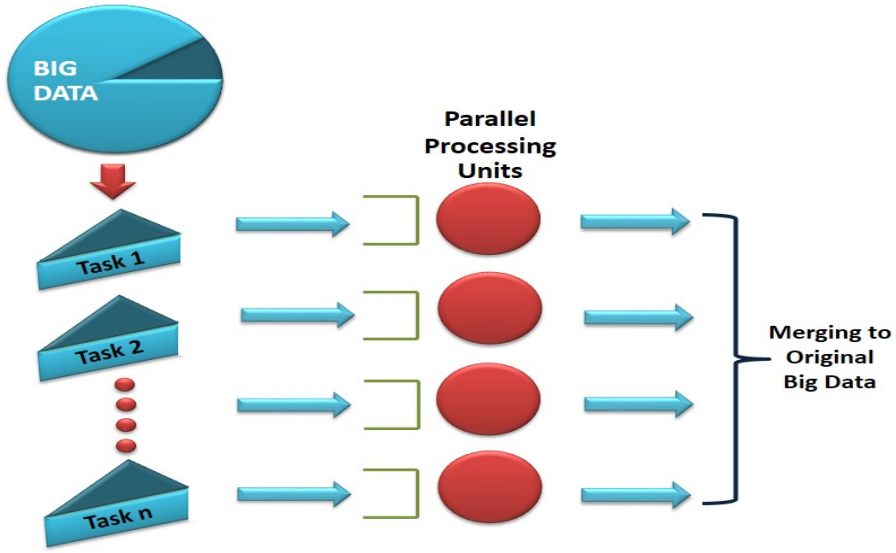

Figure 2. Parallel Processing Units

IV. Simulation Results and Analysis

Configuration setting in MOA simulation follows experimental parameters such as, *Task Evaluator*: EvaluatePrequentialRegression, *Learner*: VarianceReductionSplitCriterion, *Stream*: ArffDataStream, *Performance Evaluator*: WindowRegressionEvaluator, *GracePeriod*: 200, *AlternateTreeTime*: 1,500 and *AlternateTreeFading*: 0.995. The investigation of five synthetic datasets with size of (406, 432, 436, 438 and 505 MBytes) is taken into the experiment account. The *M/M/n* network system (*n* is ranging between 1 to 20) depicted in Fig 3 was configured and used by MOA simulation. Five datasets will be simulated on n parallel servers. Each dataset will be divided initially into n tasks regarding to targeted parallel processors. The residual time calculation will take splitting time and reassembly time into the account of both simulation and prediction. Splitting is computed based upon GSplitter developed by [3]. As reference, residual time can be presented by Eq 1. The comparison between simulation and prediction results executing on single up to twenty processing units (PUs) is shown in table 1. The same table lists residual time (*RT*) from simulation against those *RT* results from the prediction method. Prediction results are neighboring to those simulation results. Prediction may help simulate here-and-now results; nonetheless, involve less energy, time-consumption and budget than former methods.

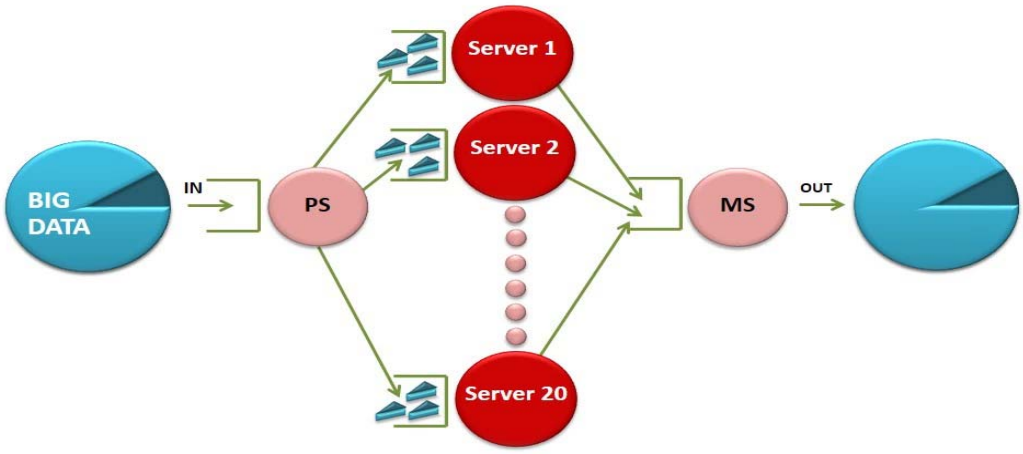

Figure 3. Experimental Model

TABLE I. Residual time from simulation versus results from the prediction method

Residual Time (sec)		Data Set				
		1	2	3	4	5
Single	SIM	246	107	163	110	306
	APP	245	106.19	163	109.89	306.07
2 PUs	SIM	138	57.39	90	57	163.26
	APP	136	56.83	90.81	56.9	162.84
4 PUs	SIM	64.7	27.35	50.08	29.31	87.14
	APP	64.6	27.06	50.05	29.26	87
6 PUs	SIM	43.4	18.34	28.72	20.19	55
	APP	43.3	18.04	28.64	20	54.8
8 PUs	SIM	32.95	14.18	21.45	16.87	47
	APP	32.9	14.04	21.41	16.84	47
10 PUs	SIM	27.24	11	17.4	11.5	34.86
	APP	27.2	10.89	17.6	11.42	34.47
12 PUs	SIM	21.68	9.24	13.6	9.3	28.27
	APP	21.65	9.15	13.5	9.2	28.24
14 PUs	SIM	19.42	8.67	11.8	8.6	24.07
	APP	19.39	8.46	11.7	8.5	24
16 PUs	SIM	16.83	7.93	10.67	7.41	20.62
	APP	16.8	7.82	10.63	7.4	20.63
18 PUs	SIM	13.8	6.2	9	6.16	18.14
	APP	13.8	6.1	9	6.1	18.12
20 PUs	SIM	12.72	5.58	8.25	5.5	16.3
	APP	12.7	5.5	8.1	5.4	16.2

Speedup efficiency is analyzed using metrics from simulation. The speedup improvement is achieved by pre-configuration set for the experiment. Simulations are performed on a parallel system which has n processors. Let RT_n represent the residual time which expires from the beginning of the partitioning to the reassembly. Speedup analysis of the calculation's residual time emphasizes on these folds. The experiment run by n processors is the total number of processing devices. The intercommunication overhead from synchronization is identical to the run-time for the calculation on a single processor, symbolized by RT_1. *Speedup* is the speed achievement done by parallel processing comparable to single processor: $Speedup_n = RT_1 / RT_n$. If the speedup is *speedup(n)* for input n parallelism, the speedup metric follows a regression. Simulation results conclude head-to-head line to prediction ones accordingly. The speedup efficiency is remarkable for parallel processing benefits. Cost-effectiveness figure is next step of calculation to further analyze for price-performance, which is a cost of processing units over achieved benefits. Comparison results between simulation versus prediction are summarized in table 2. Both simulation and prediction results are close.

Speedup metrics have been visualized for the sake of healthier comprehension as shown in Fig 4. Speedup increases linearly regarding to the higher number of parallel processing units using MOA simulation. Further analysis on optimization can be achieved by employing these fundamental collective data. Optimization for instance an image using microwave technique had been obtainable in [9].

TABLE III. Speedup from simulation versus results from the prediction method

Speedup		Data Set				
		1	2	3	4	5
Single	SIM	1.0	1.0	1.0	1.0	1.0
	APP	1.0	1.0	1.0	1.0	1.0
2 PUs	SIM	1.8	1.9	1.8	1.9	1.9
	APP	1.8	1.9	1.8	1.9	1.9
4 PUs	SIM	3.8	3.9	3.3	3.8	3.5
	APP	3.8	4.0	3.3	3.8	3.5
6 PUs	SIM	5.7	5.8	5.7	5.4	5.6
	APP	5.7	5.9	5.7	5.5	5.6
8 PUs	SIM	7.5	7.5	7.6	6.5	6.5
	APP	7.5	7.6	7.6	6.5	6.5
10 PUs	SIM	9.0	9.7	9.4	9.6	8.8
	APP	9.0	9.8	9.3	9.6	8.9
12 PUs	SIM	11.3	11.6	12.0	11.8	10.8
	APP	11.4	11.7	12.1	12.0	10.8
14 PUs	SIM	12.7	12.3	13.8	12.8	12.7
	APP	12.7	12.6	13.9	12.9	12.8
16 PUs	SIM	14.6	13.5	15.3	14.8	14.8
	APP	14.6	13.7	15.3	14.9	14.8
18 PUs	SIM	17.8	17.3	18.1	17.9	16.9
	APP	17.8	17.5	18.1	18.0	16.9
20 PUs	SIM	19.3	19.2	19.8	20.0	18.8
	APP	19.4	19.5	20.1	20.4	18.9

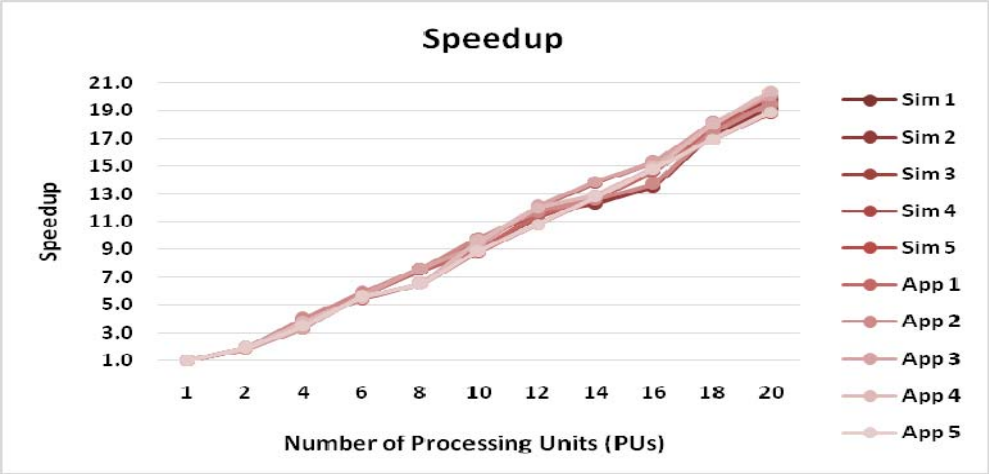

Figure 4. Speedup comparison results

Cost-effectiveness experiment is an analysis of economic point of view which associates the cost of investment with obtainable performance from different sequences of experiment. It is used in engineering field for the purpose of which experiment will worth the investment most. Normally it is equated in terms of a ratio of performance achievement done by parallel processing comparable to number of processing units: *Cost-effectiveness*$_n$ = *performance of n processors* / *n*. The *cost-effectiveness* can be realistic to the development and strategic planning of various organizations. It is also applied in several practices. In the purchase of computer networks, for instance, network architecture is associated not only for computer costs, but also for factors such as their speedups, transaction processing rate, and bandwidth. If a network performance of a supplier is inferior to the competitor, but considerably cheaper and friendlier to use, top management may

choose the inferior one due to cost effective analysis. On the other hand, if the difference of price scale is next to zero, but the higher bandwidth, more processing power and better speedup top management may select the other company as an alternative due to similar cost effectiveness concept. Cost effective visualization is presented in Fig 5.

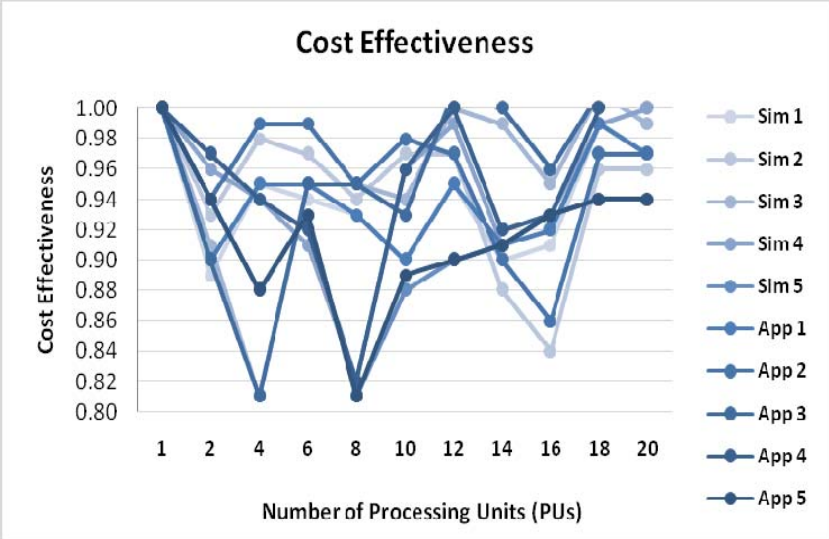

Figure 5. Cost-effectiveness results

As results, the higher number of processors will be more cost effective accordingly. However, in the environment of both four and eight processing units, it is found that cost-effectiveness drops to approximately 80%. The tendency of decreasing follows both simulation and prediction results. The impact may be caused by particular dataset characteristics which will not approve the partition of both four and eight tasks accordingly.

V. Conclusions

In this research, to conquer the computing load in big data curation is focused. In data curation for a big file, smaller tasks are recommended regarding to the parallel processing power as a huge dataset contains most statistical data. Thus the big data is firstly partitioned into independent tasks while the division time will be taken into account. The experiments succeed with increases from 1 up to 20 processing power with huge data volume. Next, parallel processing is applied accordingly before the reassembly of these parallel results. Similarly, merging time is considered. To obtain the performance of parallel processing system, simulation is used for five different datasets. Besides, in order to avoid computing burden, prediction method is proposed. Comparison results conclude prediction method contributes evenly for simulation. Lastly, prediction and simulation results are in our head-to-head comparison. Future works include optimization based on cost-effective metrics and simulation with different configuration settings for machine learning as well as deep learning.

References

[1] L. Breckels, S. Holden, D. Wojna, C. Malvey, A. Christoforou and A. Groen, “Learning from Heterogeneous Data Sources,” Plos Comput Biol, vol. 12, no. 5, 2016, pp.1-6.
[2] C. F. Tsai, W. C. Lin and S.W. Ke, “Big Data Mining with Parallel Computing: A Comparison of Distributed and MapReduce Methodologies,” Journal of Systems and Software, 2016, pp. 83-92.
[3] “G.D.G. Software SARL,” 29 December 2016. [Online]. Available: http://www.gdgsoft.com. [Accessed 15 February 2017].
[4] C. Jittawiriyanukoon, “Performance Evaluation of Reliable Data Scheduling for Erlang Multimedia in Cloud Computing,” IEEE Proceedings of the Ninth International Conference on Digital Information Management (ICDIM 2014), 2014, pp. 39-44.
[5] A. Rodriguez, O. Trelles and M. Ujaldon, “Using Graphics Processors for High Performance Normalization of Gene-Expression,” IEEE 13th International Conference on High Performance Computing and Communications (HPPC), 2011, pp. 599-604.
[6] A. Bifet, R. Kirkby, G. Holmes and B. Pfahringer, “MOA: Massive Online Analysis,” Journal of Machine Learning Research 11, 2010, pp. 1601-1604.
[7] M. Xu, P. Thulasiraman and S. Noghanian, “Microwave Tomography for Breast Cancer Detection on Cell Broadband Engine Processors,” Journal of Parallel and Distributed Computing, Elsevier, Vol.72, Issue 9, 2012, pp. 1106-1116.
[8] M. Xu and P. Thulasiraman, “Mapping Iterative Medical Imaging Algorithm on Cell Accelerator,” International Journal of Biomedical Imaging, no. 11, 2011.
[9] A. Sabouni, S. Noghanian and S. Pistorius, “A Global Optimization Technique for Microwave Imaging of the Inhomogeneous and Dispersive Breast,” Canadian Journal of Electrical and Computer Engineering, vol. 35, no. 1, 2011, pp.15-24.
[10] A. Ashtari and et. al., “Using a Priori Information for Regularization in Breast Microwave Image Reconstruction,” IEEE Transactions on Biomedical Engineering, vol. 57, no. 9, 2010, pp. 2197-2208.
[11] R. P. Souto and et. al, “Performance Evaluation of Quicksort with GPU Dynamic Parallism for Gene-Expression Quantile Normalization,” Journal of Communication and Computer, vol. 10, 2013, pp.1522-1528.