INTERNATIONAL JOURNAL OF ADVANCED STUDIES
IN COMPUTER SCIENCE AND ENGINEERING
IJASCSE VOULME 5 ISSUE 11 2016

11/30/2016

Big Data Analysis Techniques using Multi-GPUs MapReduce Implementations

Belal Zaqaibeh
Department of Computer Science,
Applied Science University (ASU),
Al-Ekir, Bahrain
e-mail: zaqaibeh@asu.edu.bh

Islam Obaidat and Wegdan Hussien
Department of Computer Science,
Jordan University of Science and Technology (JUST),
Irbid, Jordan
e-mail: iamobaidat@gmail.com, wegdan200@gmail.com

***Abstract*—Big Data brought a lot of challenges that need to be solved. Such issues were targeted by many researchers and still an interesting field in the meantime. One important issue is to analyze big data frameworks and compromise this data and make it a tool for decision making. The big data needs more sophisticated analysis paradigm rather than those used to analyze traditional data due to the big data characteristics. Several approaches to big data analysis were proposed in the literature, those should be spotted are the approaches that take advantage of parallel processing due to its capability, promising, and remarkable results. A various implementations for MapReduce has been proven as a successful implementation. This research aims to discuss some of the current multi-GPU MapReduce frameworks for big data analysis considering the point of parallel data processing. In addition, various and comprehensive comparisons of those frameworks.**

Keywords- Big Data; MapReduce; GPU; Mapper; Reducer; multi-GPU frameworks

I. INTRODUCTION

The recent few years have led to a new and solid definition of databases from traditional textual throughout object oriented databases to what so called big data. Big data term refers to datasets which cannot be acquired, perceived, processed and managed by traditional software, hardware and information technology tools within an acceptable amount of time [1]. Big data exhibit a variety of characteristics as volume, velocity, variety, variability and complexity [2]. With the dawn of the big data era, many technical challenges emerges [3] which are incompleteness, heterogeneity, scale, and timeliness. To handle such characteristics, efficient techniques emerge to analyze such data.

The huge amounts of data generated as big data yields to new challenges and opportunities for data analysis. Valid big data analysis tends to be crucially important process due to the impact of data analysis in supporting decision in any field. There are two main goals for big data analysis [4] to determine the relationship between the response and features of scientific purposes to predict observations accurately about the future using effective methods.

To address big data analytics, parallel architectures have been established lately. One of the most important parallel techniques to handle big data is MapReduce [5] developed by GOOGLE and due to the successful fact about its processing paradigm, many MapReduce frameworks have been proposed on parallel platforms such as multi-core systems [6]. Moreover, many researchers try to take advantage of the well-known graphical processing unit (GPU) due to its astonishing performance in parallel processing. GPU is used with big data analysis to increase the performance by decreasing execution time of MapReduce operations. Some of that work presented as a single-GPU frameworks [7, 8] and others present multi-GPU frameworks [9].

II. PRELIMINARY

MapReduce [5] is a massive data sets parallel processing framework. A task undergoing MapReduce framework has two main phases which are:

- Map: In this phase the mapper takes key/value pairs and performs computation, the output is an intermediate result represented as pairs of key/value.
- Reduce: It takes the intermediate results and processes a reduce function to it.

The mapper results are shuffled before they are reduced. Data as shown in Figure 1 is being processed using MapReduce through six functions [10] which are input reader, mapping, combiner, partition, reduce, and output writer. Basically, for a precise task, a map function is just needed strictly, despite the fact that most tasks also use a reduce function. The data source and destination determine the need for an input reader and output writer. The data distribution is responsible for determining the need to provide a partition and combiner functions.

Parallelism is the computing trend nowadays since its success has been proven in real life regardless of the challenges associated with this concept. The modern effort in microprocessor development concentrates on the addition of

extra cores rather than focusing on enhancing the performance of a single-thread. The Graphics Processing Units (GPU) is a machine with many cores that is able to execute thousands of threads simultaneously. GPU is a highly parallel processing unit that is intended to perform a massive amount of computation that fulfills the computational needs for the demanding applications. Basically, GPU was designed for a specific types of applications with the thirst of computing resource characteristic.

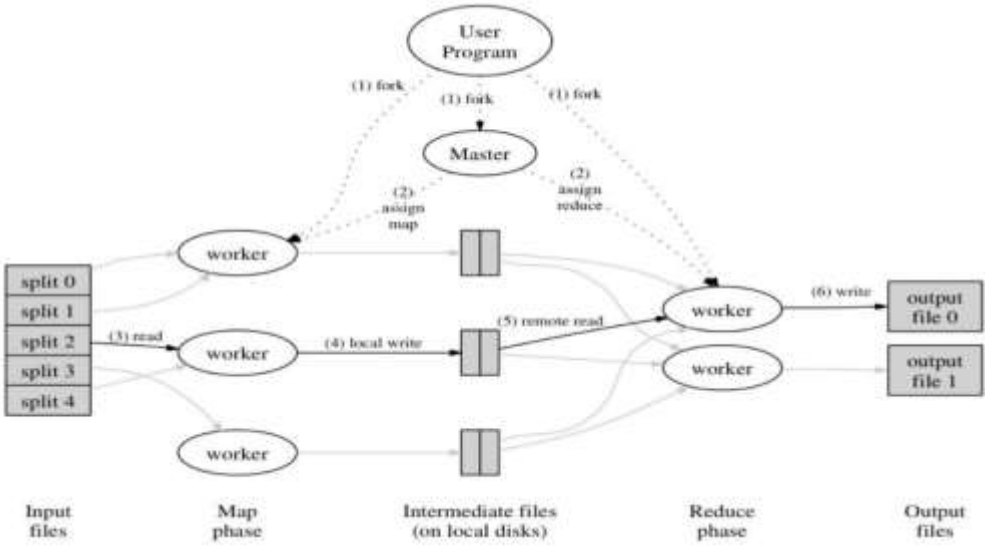

Figure 1. MapReduce algorithm execution

III. Related Work

A design and an implementation of a parallel framework on GPU clusters is proposed [11]. This framework provides a set of APIs to help programmers on their own application such that each pipeline is programmed by developers, nevertheless a default implementations is provided. The framework uses a distributed file system (GlusterFS) in order to store data in a distributed manner and it was implemented using thrust, C++ and CUDA and it also considers a dynamic load balancing. The framework was implemented on Prestack Kirchhoff Time Migration (PKTM) for seismic to test the framework, which is made up from three main parts: Master, Mapper, and Reducer as shown in Figure 2.

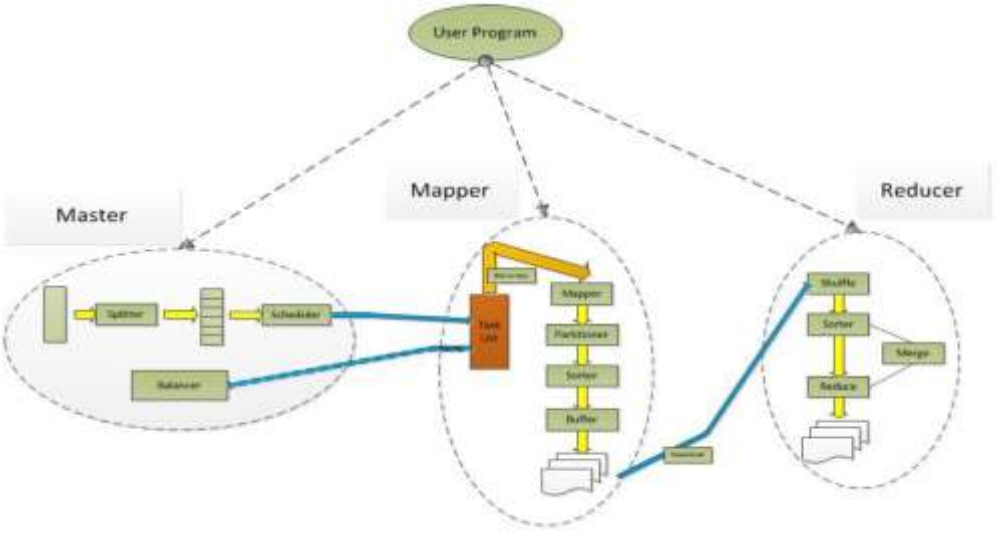

Figure 2. Framework work flow

Each node (Master, Mapper, and Reducer) gets its job based on a job controller. The user specifies the strategy of data split and submit a job to Master, thus the data is split into chunks by the master and the master goes on and schedules jobs to each computing node. The master node is responsible for load balancing. The Mapper node has multiple GPUs, each GPU runs a specific task unit and generates a key-value intermediate pair sets are combined and partitioned. The reducer has its own task list where it keeps the received job unit, Reducers GPU download the job data from Mappers and start running job units. Test result on PKTM of seismic data shows an acceleration in performance over CPUs and one GPU.

A multi-GPU MapReduce which is called MGMR [12] as Figure 3 shows is developed to utilize multiple GPUs and manage large-scale processing of data that cannot be handled by GPU memory. In MGMR framework, user can specify all phases and sub phases, and workers accomplish their job. The workers are hardware threads among GPUs in order to perform load balancing. In Map stage, output is shuffled between workers among GPUs without being passed to CPU memory. Workers with a single GPU are responsible for the communication. Workers with multiple GPUs uses GPUDirect which allow remote memory access for GPU with no need to be passed through CPU memory. Finally, the Reduce stage output of multiple GPUs is then copied to CPU memory.

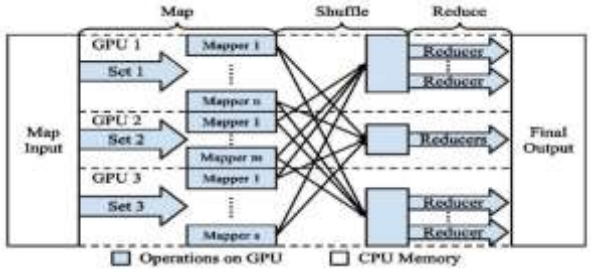

Figure 3. Single-round MGMR workflow

An upgrade version of MGMR which is called a Pipelined Multi-GPU MapReduce system (PMGMR) [13] is proposed. PMGMR compromises hard disk utilization and the new GPU features such as Hyper-Q and streams to enhance the performance. A GPU gets a lot of enhancements, powerful GPUs emerges as NVIDA Kepler. To seize of Kepler advanced features such as Hyper Q and Asynchronous Dual-channel Data Transfer, this work implements PMGMR for efficiency and to get more powerful multi GPUs framework.

Since the Big Data size is growing rapidly the performance enhancement is not enough. So to handle such sizes this work introduces the Memory Hierarchy in MGMR idea and improve the ability to store a huge data sizes as well as the data transfer between processing units and disks is more efficient. PMGMR has three functional units which are a GPU operation scheduler, a configuration optimizer, and a job scheduler.

MGMR++ is proposed [14] to eliminate the limitation of the pipelined and GPU memory. The MGMR++ uses flexible C++ templates as extension to MGMR and it is dedicated to NVIDIA Fermi family GPUs platform. It is implemented to maintain high utilization of multiple GPUs to be customizable and extensible. Load balancing among different GPUs is targeted at runtime based on job sizes and hardware performance.

The communication is done through global shared GPU memory for a single GPU workers. In Multi GPUs workers, to achieve a better performance GPUDirect is used to allow remote access of GPU memory without the need to go in the CPU memory. If data size exceeds the total memory of multiple GPU in MGMR++, this can be handled through iterative GPU activations in in multi-round mode.

MapReduce framework which is called MOIM [15] is an efficient utilization of parallelism in both multicore CPUs and GPUs, it minimizes delay by overlapping the computations of both CPU and GPU, and supports load balancing among mappers and reducers, also deals with data of both fixed and variable size.

A new MapReduce framework is called GCMR [16] is proposed to accelerate processing of large data on a GPU cluster. The GCMR is implemented using CUDA and MPI and it is based on two parallelization levels schema which are the inter node and intra node levels. A multi-threading pipeline is used to increase the efficiency by overlapping data communication and computation between the CPU and the GPU. GCMR framework consists of four main stages which are splitting input, mapping, shuffling, and reducing.

A GPU MapReduce framework which is called GMRF [17] is designed to be a solution for running MapReduce operations that can be parallelizable in the GPU environment. The aim of GMRF is to integrate the framework with MapReduce applications such Hadoop that can process a large amount of data efficiently using the high capabilities of GPU. GMRF is designed to be generic so the user can implement only the operations of MapReduce model and the framework will handle all operations related to GPU programming using OpenCL library. GMRF supports different types of data: text, vector, matrix and (key, value).

GMRF is consisting of one master node and several slave nodes. JopTracker in master node controls the cluster by scheduling many slave nodes whereas TaskTracker in slave node manages the input/output data and starts the GPU controller which establishes the connection between GPU MapReduce module and Hadoop. The input data is allocated to slave node and if the all input files can be stored in the GPU memory then the input files can be processed in a single step operation, otherwise the input files will be grouped by the GPU Controller and will be processed in several steps by GPU MapReduce module. The output is stored temporarily in a file and then all results will be grouped by the Combiner.

In reducing task, the intermediate data is stored in the GPU memory and processed in order to reduce the number of keys and resulting (key, value).

IV. Analysis and comparison

The MGMR does not support load balance especially when shuffling intermediate pairs to reducers because it takes only the advantage of multi-GPUs in a single machine where PMGMR, MGMR++, and MOIM frameworks support load balancing between cluster nodes. Table 1 illustrates a comparison among most important frameworks.

Data transfer between CPU and GPU is considered to be a bottleneck in many applications based on GPU, so overlapping is an important schema to increase execution time by minimizing the time of communication between CPU and GPU, in PMGMR and MGMR++ pipelined data path maximizes the GPU usage using CUDA streams, the MOIM supports pipelining by overlapping computations of CPU and GPU, GCMR also supports overlapping using multi-threading scheme, whereas MGMR does not support overlapping since it focuses on GPU utilization without taking into account CPU cores. The GMRF resolves the bottleneck of communication by proposing a model to use GPU shared memory efficiently. Moreover results show that PMGMR, MGMR++, schemes increase scalability of the system compared to MGMR and over performs with about 2.5-fold improvement in performance, and enable users to write a better big data code in MapReduce.

TABLE I: A comparison among multi-GPU MapReduce FRAMEWORKS

Framework	No. GPU	Benchmarks	Communication
MGMR	2 GPUs (Tesla C2070 and Quadro 6000)	K-Means Clustering (KMC) and Unique Phrase Pattern (UPP)	OpenMP
PMGMR	4 GPUs (2 Tesla C2050 and 2 Tesla K20Xm)	K-Means Clustering (KMC)	GPUDirect
MGMR++	2 GPUs (Tesla C2070 and Quadro 6000)	K-Means Clustering (KMC) and Unique Phrase Pattern (UPP)	OpenMP
MOIM	12 GPUs PNY GeForce GTX580 (3GPUs per node)	Matrix Multiplication (MM) & Word Count (WC)	MPI
GCMR	8 GPUs Tesla c1060 (2 GPUs per node)	Matrix Multiplication (MM), String Match (SM) & Word Count (WC)	MPI
GMRF	-	Matrix Multiplication (MM) & Word Count (WC)	-

V. Conclusion

The analysis of data with big data is one of the major issues and address trend for researchers. Moreover, the size of such data is growing rapidly, so there is a need for models and frameworks to handle this kind of data in an efficient manner. MapReduce is a successful technique used widely to analyze big data, MapReduce has several implementations to prove the soundness of this platform. According to the fact that GPU capabilities are more powerful than CPUs, this survey targets MapReduce algorithms that compromise multi-GPUs architecture in the state of art to give an insight to those algorithms.

References

[1] M. Chen, S. Mao, Y. Zhang, V. Leung. “Big data: related technologies, challenges and future prospects,” Cham: Springer, 2014.

[2] A. Katal, M. Wazid, RH Goudar. “Big data: Issues, challenges, tools and good practices,” In Contemporary Computing (IC3), 2013 Sixth International Conference on, pages 404-409. IEEE, 2013.

[3] N. Ammu, M. Irfanuddin. “Big Data Challenges,” International Journal of Advanced Trends in Computer Science and Engineering, Vol.2, No.1, Pages: 613 – 615, 2013.

[4] Bickel, P. Discussion on the paper ‘Sure independence screening for ultrahigh dimensional feature space’ by Fan and Lv. J Roy Stat Soc B 2008; 70: 883–4.

[5] J. Dean, S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of ACM, 51:107–113, 2008.

[6] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Phoenix++: Modular MapReduce for Shared-Memory Systems. In International Workshop on MapReduce and its Applications, 2011.

[7] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: a MapReduce Framework on Graphics Processors. In International Conference on Parallel Architectures and Compilation Techniques, 2008.

[8] C. Hong, D. Chen, W. Chen, W. Zheng, and H. Lin. MapCG: Writing Parallel Program Portable between CPU and GPU. In International Conference on Parallel Architectures and Compilation Techniques, 2010.

[9] J. A. Stuart and J. D. Owens. Multi-GPU MapReduce on GPU Clusters. In IEEE International Parallel and Distributed Processing Symposium, 2011.

[10] Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2004.

[11] Heng Gao, Jie Tang, Gangshan Wu, "A MapReduce Computing Framework Based on GPU Cluster," IEEE Conference, High Performance Computing and Communications & Embedded and Ubiquitous Computing, pages 1902-1907, 2013.

[12] Yi Chen, Zhi Qiao, Hai Jiang, Kuan-Ching Li, Won Woo Ro, “MGMR: Multi-GPU MapReduce on GPU Clusters,” Grid and Pervasive Computing. Lecture Notes in Computer Science, vol. 7861, pp. 433–442. Springer, Berlin, 2013.

[13] Yi Chen, Zhi Qiao, Spencer Davis, Hai Jiang, Kuan-Ching Li, “Pipelined Multi-GPU MapReduce for Big-Data Processing,” Computer and Information Science Studies in Computational Intelligence Volume 493, pp 231-246, 2013.

[14] Hai Jiang, Yi Chen, Zhi Qiao, Tien-Hsiung Weng, Kuan-Ching Li, “Scaling up MapReduce-based Big Data Processing on Multi-GPU systems,” Cluster Computing, Volume 18, Issue 1, pp 369-383, 2015.

[15] Mengjun Xie; Kyoung-Don Kang; Basaran, C., "Moim: A Multi-GPU MapReduce Framework," Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on , vol., no., pp.1279,1286, 3-5 Dec. 2013.

[16] Yiru Guo; Weiguo Liu; Gong, B.; Voss, G.; Muller-Wittig, W., "GCMR: A GPU Cluster-Based MapReduce Framework for Large-Scale Data Processing," High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, vol., no., pp.580,586, 13-15 Nov. 2013.

[17] Nitu, R.; Apostol, E.; Cristea, V., "An improved GPU MapReduce framework for data intensive applications," Intelligent Computer Communication and Processing (ICCP), 2014 IEEE International Conference on , vol., no., pp.355,362, 4-6 Sept. 2014.