Int J Parallel Prog (2018) 46:630–646
https://doi.org/10.1007/s10766-017-0513-2

Real-Time Big Data Stream Processing Using GPU with Spark Over Hadoop Ecosystem

M. Mazhar Rathore[1] · Hojae Son[1] · Awais Ahmad[2] · Anand Paul[1] · Gwanggil Jeon[3]

Received: 27 February 2017 / Accepted: 15 June 2017 / Published online: 27 June 2017

Abstract In this technological era, every person, authorities, entrepreneurs, businesses, and many things around us are connected to the internet, forming Internet of thing (IoT). This generates a massive amount of diverse data with very high-speed, termed as big data. However, this data is very useful that can be used as an asset for the businesses, organizations, and authorities to predict future in various aspects. However, efficiently processing Big Data while making real-time decisions is a quite challenging task. Some of the tools like Hadoop are used for Big Datasets processing. On the other hand, these tools could not perform well in the case of real-time high-speed stream processing. Therefore, in this paper, we proposed an efficient and real-time Big Data stream processing approach while mapping Hadoop MapReduce equivalent mechanism on graphics processing units (GPUs). We integrated a parallel and distributed environment of Hadoop ecosystem and a real-time streaming process-

✉ Anand Paul
paul.editor@gmail.com

M. Mazhar Rathore
rathoremazhar@gmail.com

Hojae Son
sonhj07@gmail.com

Awais Ahmad
aahmad.marwat@gmail.com

Gwanggil Jeon
gjeon@inu.ac.kr

[1] School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea

[2] Department of Information and Communication Engineering, Yeungnam University, Gyeongbuk, Korea

[3] Department of Embedded Systems Engineering, Incheon National University, Incheon, Korea

ing tool, i.e., Spark with GPU to make the system more powerful in order to handle the overwhelming amount of high-speed streaming. We designed a MapReduce equivalent algorithm for GPUs for a statistical parameter calculation by dividing overall Big Data files into fixed-size blocks. Finally, the system is evaluated while considering the efficiency aspect (processing time and throughput) using (1) large-size city traffic video data captured by static as well as moving vehicles' cameras while identifying vehicles and (2) large text-based files, like twitter data files, structural data, etc. Results show that the proposed system working with Spark on top and GPUs under the parallel and distributed environment of Hadoop ecosystem is more efficient and real-time as compared to existing standalone CPU-based MapReduce implementation.

Keywords Big Data · Hadoop · Spark · GPU · MapReduce

1 Introduction

Recently, an extensive interest has been seen in several areas of Information and Communication Technology (lCT) that contributed a substantial evolution in the data and the analytics [1,2]. One of the IBM reports in 2012 narrated that overall in the world, 90% of the data was generated in 2010–2011 [3]. Existing services (e.g., social websites, networks, web apps, and so forth) and various sensor technologies (e.g., health sensors, environmental sensors, road and vehicle sensors, ultrasonic sensors, environmental sensors, and so forth) are producing gigabytes of data within few milliseconds continuously. Furthermore, in this technological era, a number of devices and objects connected to the internet is more than humans, which forms an Internet of things (IoT) and this number would reach to 50 billion in 2020 [4]. With this number, one can imagine the amount of high-speed data generation. This massive amount of high-speed data, termed as Big Data, is acknowledged by 3 V's, as volume (data with huge size), velocity (data generating with high speed), and variety (data with diverse nature). Similarly, these days, most of the countries are using a network of video cameras to monitor streets and roads. Vehicles have cameras and black box to track the record in case of any accident. Moreover, the vehicle cameras are also being used to track and detect vehicles on the road while moving [5]. These thousands of cameras throughout the city are generating a vast quantity of high-speed Big video Data.

This great volume of high-speed diverse data brings the issues of aggregation, storage, and processing. On the other hand, it can be used in establishing smart parkings and smart transportation systems [6], health systems [7,8], Smart city and urban plannings [9], remote sensing [10], and in many other applications. In addition, Big Data analytics can be very useful in order to predict future, recommend systems based on user trends, predict future needs, improve businesses and infrastructure, and much more. In a nutshell, it is an asset for the organizations, authorities, and businessmen if properly handled, and efficiently processed and analyzed. Furthermore, people are moving to cities, and in 2050, 70% of the overall population will live in urban areas [11]. This transformation will have the disastrous effect the city traffic. Authorities have a lack of human resources to monitor and control such amount of traffic continuously. Also, there is a possibility of maladministration by the authorities to the public.

Therefore, these days, authorities are moving towards computer-based systems to continuously monitor and control city traffic without human's intervention by real-time analytics of Big video data generated by road and vehicles' cameras.

Even though, any data, either it is video or text, is very useful for analytics and decision-making, but, processing a big amount of data is a challenging task, which needs special hardware tools, computing models, software tools, and advanced communication infrastructure. Some systems exist that handles the IoT-generated data for various purposes and decision making [6–9]. However, these systems are specific to some sensors and IoT applications; They do not ruminate the real-time aspect of big traffic video processing. Many companies are using a distributed data processing tool called Hadoop [12], which is best to handle a large amount of stored data. Hadoop performs well for batch processing. On the other hand, it is not suitable to use for real-time applications, especially video processing. Apache Spark [13] provides the real-time data processing. Still, this tool is not efficient enough to analyze a large amount of high-speed text and video data while working standalone. Hence, there is a need for an efficient and real-time system that uses advanced technology to process a large amount of high-speed data (textual and video) in order to perform some analytics to make decisions.

In recent years, GPUs have become a powerful coprocessor for general purpose computing and video processing. GPUs can be regarded as massively parallel processors with an order of magnitude higher computation power (in terms of some floating point operations per second) and memory bandwidth than CPUs [14]. Moreover, the computational performance of GPUs is improving at a rate higher than that of CPUs. GPUs are traditionally designed as special-purpose coprocessors for dedicated graphics rendering. As such, GPU cores are single-instruction-multiple-data (SIMD), which discourages complex control flows. Furthermore, GPU cores are virtualized, and the hardware manages threads. GPUs manage their onboard device memory and require programmers to explicitly transfer data between the GPU memory and the main memory. All these factors make it desirable to perform general purpose computation on GPUs (GPGPUs) framework on which users can develop correct and efficient GPU programs easily. However, it is still a challenging task of developing efficient GPU programs for complex applications and high-speed videos coming from thousands of cameras in the city. However, with the integration of GPUs and Big Data processing technologies, the real-time traffic monitoring and high amount of real-time Big Data processing could be possible. Few systems have been designed that use parallel and distributed processing system, i.e., Hadoop [12] with its programming paradigm called Hadoop MapReduce that will automatically distribute and execute tasks on multiple machines [15] or multiple CPUs in a single machine [16]. Thus, this paradigm reduces the programming complexity so that developers can easily exploit the parallelism in the underlying computing resources for complex tasks. On the other hand, the MapReduce is only suitable for batch processing. Therefore, in the real-time Big Data processing, MapReduce would not be a good option. Therefore, in this paper, we are proposing an efficient and real-time system that uses GPU and Spark with Hadoop ecosystem to process a large amount of high-speed data (textual and video) in order to perform some analytics to make decisions.

The contribution of the paper is as follows:

- An efficient and real-time Big Data processing system, including its architecture and implementation model, is proposed that integrates the advanced processing technology of Hadoop for parallel and distributed processing, Apache Spark for achieving real-time environment, and GPUs for fast and efficient processing.
- In order to extract features and compute various generic statistical parameter for Big Data analytics and classification problems, a Map Reduce equivalent mechanism is proposed to work with GPUs (especially, working with video data) while dividing texture and multimedia data into fixed-size blocks.
- Finally, the whole system is implemented and tested on texture data and real-time vehicular traffic videos captured from road and vehicle cameras. The evaluation is performed by considering efficiency aspect of the system with respect to processing time and system throughput in various circumstances.
- The proposed system with GPUs working under the Hadoop and Spark working over the Hadoop ecosystem is more proficient than the traditional MapReduce implementation.

The rest of the paper is organized as follows. The next section, Sect. 2, presents the state of the art work related to the proposed system. Section 3 provides the overall details of the proposed system including the proposed architecture to process high-speed real-time Big Data and the proposed Map Reduce equivalent data processing algorithm using GPUs. Later section i.e. Sect. 4, provides the information of datasets used, the system's implementation environment, and the system's evaluation. Finally, the last section concluded the article.

2 Related Work

May works have been done in the performance improvement of CPU and GPU architectures. Various ways of analyzing multicore CPUs systems are available in the literature [17–20]. For instance, multiple parametric performance models are mentioned by the authors [17], which aim to run multiple class applications. Also, it seeks to supply a comprehensive modeling that supports multifaceted data centers. In most of the cases, importance is given to the performance of a single component of the CPUs. A low-power overhead scheme is presented based on a portion of shared cache among different application [18]. Moreover, the focus is also given to L2 cache sharing in which the performance of on-chip cache came up with a novel architecture for configuration of sharing of synchronous dynamic access memory between multiple CPU functions [19]. Similarly, to understand the partitioning effects of the system performance, a scheme [20] is presented that is based on the partitioning of cache and bandwidth partitioning interaction. Also, the parallel software implementation of 3D/4D-variational (3D VAR/4D-VAR) data assimilation is done whose aim is to reduce the overall computational cost of the OceanVar code [21]. To achieve better results of truncated SVD, OceanVar is analyzed regarding condition number which describes the benefits obtained from using Cholesky factorization.

As a matter of the fact that a significant amount of work is investigated regarding GPUs for general purpose computing in which the effectiveness of GPUs is mentioned for multiple applications [22,23]. In both cases, the performance of GPUs is compared with multicore CPU performance. Apparently, it is also shown that the performance of GPU as compared with the CPUs mainly depends on the system data and overhead, which need to move the data where it can be used again [24]. To the best of our knowledge, a framework for high-performance computing application is lacking two or more programs running in a sharing manner [25]. Therefore, the usage of hardware acceleration provided by the GPU to overcome the said constraint. Similarly, the problem above is also solved by combining compute unified device architecture (CUDA) from various virtual machines by executing them concomitantly. Thus, to support sharing of GPU among virtual machines. A newly proposed stencil-reduce is proposed, which is a high parallel filter for visual data restoration based on a skeletal approach [26]. The proposed scheme is implemented by way of FastFlow parallel programming library on a multicore machine, which is on multi GPPUSs, or even both. The major drawback of the proposed scheme is that it does not consider the real-time aspect of video and Big Data processing. Also, the GPU, standalone, does not consider the larger amount of data because of a shortage of memory. One of the broad techniques that use dynamic scheduling takes benefits of the GPUs and CPUs architecture [27]. In the given method, GPUs are optimized in a way that throughput requires a significant number of inputs size to achieve high performance. Temporarily, it takes smaller tasks that are more suitable for CPU cores. Apparently, other techniques, i.e., linear algebra are deeply optimized so that to achieve high performance on the following attributes, i.e., different clusters with GPUs, conventional clusters without GPUs, and shared-memory multiple GPUs, and multicore computers, respectively [28,29]. Similarly, using both multicore processors and standard cluster environment are discussed that consider advantage of hybrid architecture [30]. Also, a heuristic technique is proposed based on the problem knowledge [31]. While discussing shared-memory multicore machine, many alternative codes are available, i.e., LAPACK [32], PLASMA [33], PetsC, Intel MKL, AMD and ACML libraries. However, distributed-memory CPU-based machines ScaLAPACK and TBLAS represent optimal choice [34,35], whereas the collection of LAPACK subroutines are provided by MAGMA [36]. Finally, a fine-to-coarse parallelization is proposed that uses a parallel hybrid architecture by the consideration of Optical Flow numerical problem while implementing parallel multilevel software. The proposed scheme is based on a smart combination of codes on GPUs and standard scientific parallel computing libraries on a cluster. The evaluation is made on a real satellites image sequence coming from large datasets while considering large data sets using multi-GPUs and CPUs.

With respect to the Big Data processing aspects, only couple of the aforementioned techniques might be able to handle it. On the other hand, few other architectures and implementations also exist that use Hadoop and Spark tools for various Big Data analytical applications. For instance, intelligent transportation system [6] and smart city [9] are established using Big Data analytics at central city building equipped with advanced computing models on Hadoop and Spark. Similarly, researchers also worked on healthcare [7,8] and remote sensing [10] Big Data using data fusion on clusters of Hadoop ecosystem. A parallel algorithm for multilevel data processing while working in a high-performance computing environment is proposed by Ahmad et al. [37] using divide and conquer strategy for YARN and Lustre client application. Rathore et al.

[38] deeply analyzed real-time traffic data for voice over Internet Protocol (VoIP) calls detection using Big Data approach. Even though there exist many Big Data processing approaches have been conceived, but most of them are specific to some application. Few others have a lack of efficient processing of real-time video data. GPU-based processing techniques in the literature also do not support real-time video monitoring and processing. Also, the use of standalone GPU does not consider the larger amount of data because of the shortage of memory. Thus, it is essential to integrate the advanced technologies with distributed and parallel processing mechanism to analyze Big Data, especially video data, in a real-time environment without producing any delay.

3 Proposed System for Real-Time High-Speed Big Data Processing

We know that there are lots of IoT devices and systems that generate the Big Data such as, sensors, smart systems like smart home, smart parking, smart city, and social networks like facebook, twitter, etc. The processing such overwhelming amount of high-speed data requires an efficient and powerful system to be analyzed and makes real-time decisions. Besides, it is very hard for the authorities to place the traffic police at each and every place to control and monitor the traffic. Therefore, they might require a system that can automatically monitor the traffic and generate an alert in case of any traffic violation through video capturing. For them, there are two options to monitors the city traffic, i.e., (1) by running vehicles' cameras and (2) by a network of static road camera. The vehicle camera monitors all the vehicles going in front of the car in all lanes, whereas, the static camera is normally mounted on a top of the pole to monitor all the vehicles under its coverage area. Consequently, this will make a network of thousands of cameras in the city, which are continuously generating the video data of huge size with high speed, term as Big video Data. Such Big video Data can be generated from any big camera networks monitoring any area. Thus, to process above-mentioned types of Big Data, we have proposed an architecture having the ability to process a huge volume of high-speed real-time Big Data. We also proposed GPU-based MapReduce equivalent algorithm to process Big Data matrix (images/frames) that are more powerful as compared to tradition Hadoop MapReduce programming platform.

The proposed system consists of two major components. The first one is the data generation and traffic monitoring that we have already discussed in the last paragraph. The second component is the key part of the system, called central analysis building (CAB), which consists of various processing layers. The proposed system architecture is depicted in Fig. 1 that has the ability to process high-speed real-time data from thousands of devices. By real-time analyzing the traffic videos, we can decide about the suspected illegal actions from the vehicles, such as wrong U-turn, wrong overtake, drunken driving, wrong parking, or other wrong driving actions. Moreover, the driver also has the option to alert the central system and to transmit the videos in case he saw any traffic event or possible suspected activity, such as an accident or drunken driver, etc.

The central system, which is the main processing building called CAB, has five processing layers. The main responsibility of the system is to investigate the incoming

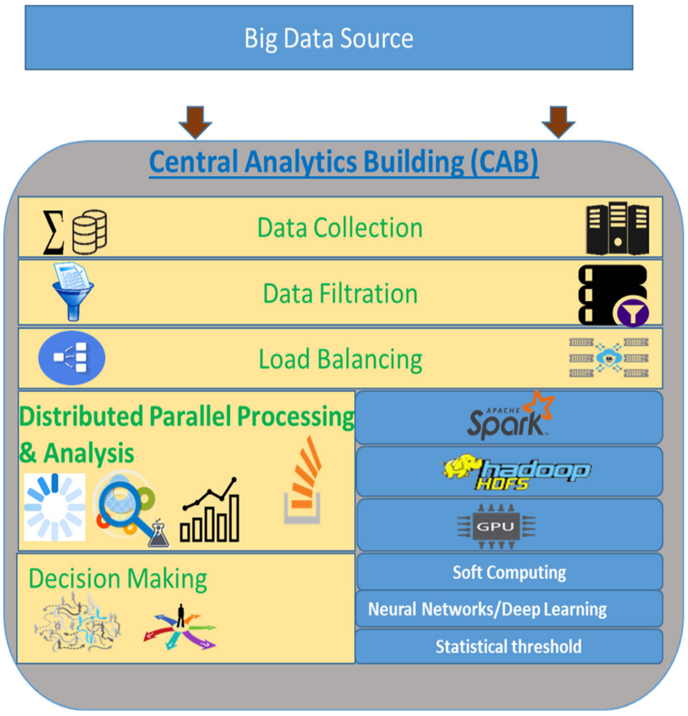

Fig. 1 Proposed real-time video processing architecture

videos/data and perform initial processing on the incoming data to extract the features and hidden parameters to perform various decision depending upon the data, application, and the needs. At the initial stage, the system collects all the videos and data from the sources, such as cameras, through its collection unit. The collection unit integrates all the data, checks for any possible intermediate alteration and noise for integrity, and transmits them for filtration. Since all the data/videos are not important to be processed as they do not have any important data (such as don't have suspected activity or traffic event), hence, the designed system does not need to process all the data generated from thousands of sources. For that reason, the filtration process is applied to make the process more efficient by discarding irrelevant data by reducing the overall data to be processed. For example, the filtration server filters all the unnecessary metadata and other video contents based on the video capturing area as well as time. Next, the set of media data is sent to the load balancing unit, which is the master server that controls and manages the data processing on a network of many data nodes (computers) working together. It has all the information of each node including its specification, its capability, its algorithms, and its current state. Based on this information, the load balancer sends each dada chunk of specific duration and the corresponding job to specific data nodes. Data nodes are a network of high-performance computers working together, equipped with GPUs, Hadoop distributed file system [12], and Apache Spark [13]. Hadoop has very powerful and efficient distributed parallel processing environment. On the other hand, the traditional Hadoop ecosystem with MapReduce programming paradigm is only suitable for batch processing. Therefore, we applied Apache Spark for real-time processing while working with powerful Hadoop environment. Spark has two main component (1) Spark streaming, that is responsible for taking real-time data in chunks (2) Spark engine, that immediately processes each data chunks at its generation by Spark streaming. In the proposed system, the Spark engine collaborates for processing with GPU that consists of thousands of specific purpose multiprocessors, which makes the system more efficient. In a nutshell, we are gaining the benefits of Hadoop parallel and distributed environment, real-time processing with Spark, and fast and efficient processing capabilities of GPUs. Finally, the decision making is performed based on the results generated by data nodes. For Instance, the decision-making process might identify as well as verify the suspected illegal traffic activities and events, and takes necessary actions, such as calling emergency services, alerting police and imposing fine on the vehicle in case of traffic control scenario.

Here in this section, we are providing the details of using GPU and how it is working with Hadoop.

3.1 Use of Hadoop with GPU for Real-Time Video Processing

3.1.1 GPU Processing with CUDA

GPU consists of a grid of thousands of multicore processors (SMs) that performs a specific task in parallel. CUDA [39] is an application programming interface (API) and parallel computing platform created by Nvidia to work with GPUs. Basically, the GPU processes the data using a grid of SMs. The grid is decomposed into blocks and further into threads by using CUDA, as shown in Fig. 2. GPU allowed a maximum of 1024 number of blocks. The overall problem is divided into blocks and assigned to each of the SMs. GPU_Kernal function provided by the CUDA algorithm assigns each of the thread, i.e., the repetitive instruction that can be performed in parallel, to each of the SMs. All the threads (even thousands) are processed in parallel, reducing the overall time. Each block has a small amount of memory that is shared only among the SMs belonging to that block. GPU also has a shared memory that is shared among all the blocks or all the SMs. GPU processing is quite efficient, but it requires a task that can be processed in parallel and treated as a thread. Moreover, it could not perform analysis on the very larger amount of Big Data because of its memory problem and its inability to collaborate on a network of computers. Therefore, we overcome this drawback by using Hadoop ecosystem with GPU processing.

3.1.2 Hadoop and MapReduce

Hadoop processes data by its user-defined Map and Reduce job using its parallel and distributed MapReduce programming paradigm. The analysis job is given using Map and Reduce function that is performed on HDFS data in the multi-node environment. MapReduce used by Hadoop is an open source implementation that is originally proposed by Google for working with clusters [40]. Later it is used by different developers and companies to process and analyze Big Datasets. It works with Hadoop Distributed

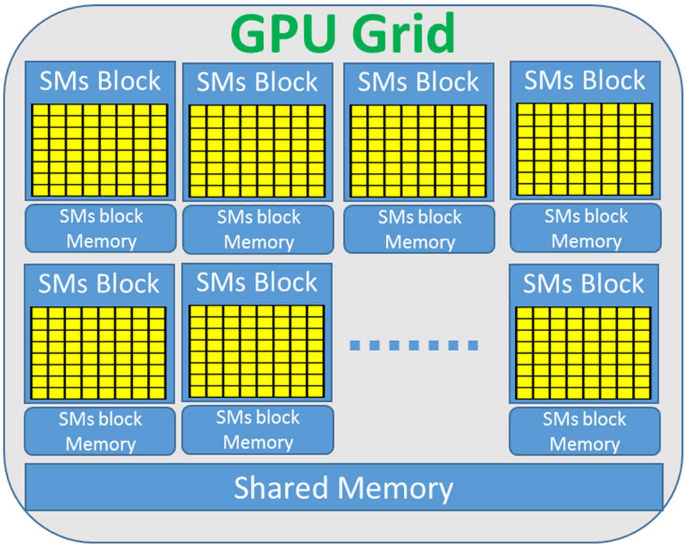

Fig. 2 Image frames processing with GPU

File System (HDFS) which is also an open source distributed file system used to store large datasets and files across distributed nodes in chunks.

MapReduce job is defined by giving the address of input and output files on HDFS in which Map function takes a set of inputs, processes it, and generates intermediate outputs in terms of Key and Value pairs for each chunk of the input file. The Reducer Job takes the input from Map function in (key, value) pair. Its responsibility is to sort, combine, and gathers all the intermediate outputs based on the Key. Though, MapReduce paradigm performs efficiently for batch processing and large datasets. But, it does not perform well for real-time processing. Therefore, to remove this drawback, we used Apache Spark with Hadoop ecosystem. We proposed an equivalent algorithm to process matrix type of data (image/video) using GPUs in the same fashion as MapReduce does. The proposed algorithm is suitable for features and parameters calculations from various raw data files and performs real-time analysis for decision making.

3.1.3 Video Processing Using Hadoop with GPU

Figure 3 shows the complete working model of GPU with Hadoop and Spark. Data is collected from remote sources and divided into small chunks using Apache Streaming that can be processed using in-Memory Database. Apache streaming captures the real-time data from the remote online source, from HDFS, and any TCP Server in chunks of a particular given duration or size. Later, the load balancer and distributed network data processing environment are implemented using Hadoop ecosystem. Load balancer works as Hadoop master node, and the distributed processing system works as HDFS data nodes. The data nodes are equipped with and GPUs. The data processing is done on distributed parallel data nodes as well as by parallel processing of GPU SMs on each node. The Spark engine implements those instructions which cannot be performed parallelly using GPU, such as CPU code in CPU-GPU programming environment. GPU runs parallel instructions (independent threads) by implementing them in GPU kernel function. The MapReduce equivalent mechanism is mapped on GPU in such a way that each matrix file (image/frame) from the data/video is divided into blocks. Parameters' calculations are performed in parallel on each block using GPUs (as MAPPER function does) and later are combined using CPU code (as REDUCER function does) based on Spark engine. There is also one global Reducer is implemented using Spark engine that combines all the results from the same key from multiple distributed nodes. The algorithm 1 presents the parameter calculations using MapReduce mechanism while algorithm 2 is the equivalent form of the algorithm using GPU. Table 1 describes all the symbols and parameters used in Algorithm 1 and Algorithm 2.

Algorithm 1: Parameter Calculation for data blocks using MapReduce

```
INPUT: Dataset_File F (or image/frame)
OUTPUT: Block with Calculated Parameters params
STEPS:
1.  blks [blk_id, blk_vals] := Divide_data (Data_File F, blk_size)
2.  FUNC MAPPER(Key blk_id, Value blk_vals):   START.
3.  |     curr_val :=First_val                    //initialized with first value
4.  |     LOOP WHILE (curr_val.HasMoreValues())      DO.
5.  |     |     EMIT (blk_id, curr_val)
6.  |     END LOOP
7.  END MAPPER FUNC
8.  FUNC REDUCER(KEY blk_id, IterableValues val) START.
9.  |     sum, sum^2, Max_val, Min_val, … :=0. //initialize all parameters with zero
10. |     LOOP FOREACH IterableValues val DO.
11. |     |     UPDATE sum,  sum^2, Max_val, Min_val, …    //update all parameters
12. |     |     Find Max_val, Min_val, ……
13. |     END LOOP FOREACH
14. |     Calculate sum, sum^2, MEAN, SD, Max_val, Min_val, ……
15. |     Return.set(blk_id, params)
16. |     CONTEXT.WRITE (blk_id, result)
17. END REDUCER FUNC.
```

Algorithm 2: Mapping of MapReduce Mechanism of Parameter Calculation for data blocks with CUDA

```
INPUT: Dataset_File F (or image/frame)
OUTPUT: Block with Calculated Parameters params
STEPS:
1.  blks [blk_id, blk_vals] := Divide_data (Data_File F, blk_size)
2.  CPUToGPU(blks[])
3.  FUNC GPU_KERNAL(INPUT blks, OUTPUT result, params, blk_size)   START.
4.  |     LOOP FOR idx_id=1 to blk_size.   DO.
5.  |     |     CALCULATE sum, sum^2, MEAN, SD, Max_val, Min_val, params
6.  |     END  LOOP FOR
7.  |     GPUtoCPU (blk_id, params)
8.  END GPU_KERNAL FUNC
```

4 System Implementation and Evaluation

4.1 Datasets Description

Both Video and text data are taken to test and evaluate the proposed system. The video's data is captured from YouTube, covering various traffic and road scenarios,

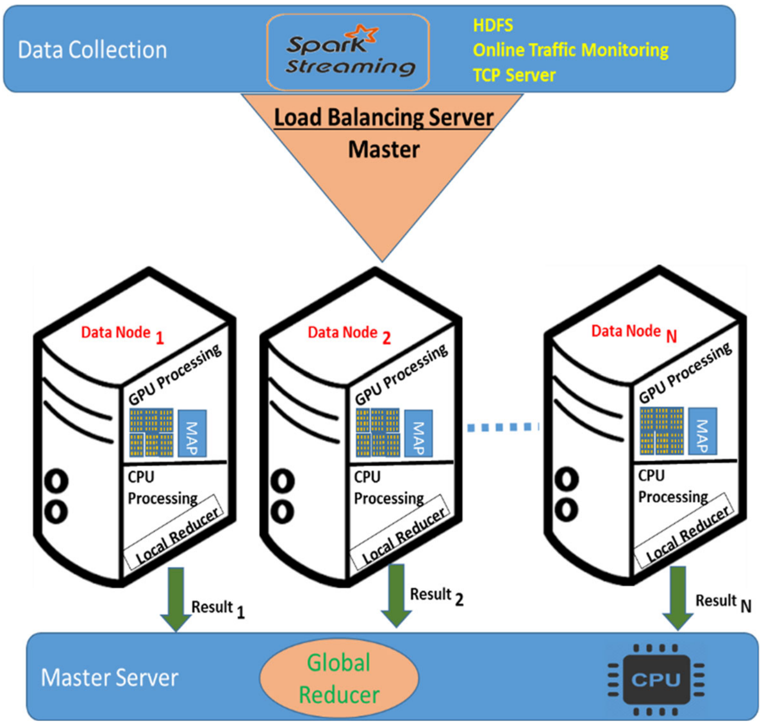

Fig. 3 Data processing in distributed environment using GPU

Table 1 Symbols used in algorithms

Algorithm symbols	Description	Algorithm symbols	Description
blk(s)	Data/image block(s)	Curr_val	Current processing value (pixel value)
blk_id	Block ID	Strat_val	Start value in the block
blk_val(s)	Block Value(s)	CPU to GPU	Send data from CPU to GPU
val(s)	Value(s)	GPU to CPU	Send data from GPU to CPU
blk_size	Block size	SD	Standard deviation
		Divide_data()	Function to divide data into fixed size matrix block

e.g., single direction and multi-direction traffic videos, Wrong U-turn and drunken driving videos captured from following vehicle cameras as well as static road cameras. For real-time system testing, online videos are taken continuously from road cameras of Arlingtonva.us [41] and Earth Cam [42]. For the texture data, we have generated more than 6 Gbs of data from sensors and from social networks through Apache Spark streaming.

4.2 Implementation Environment

For evaluation purpose, we implemented the system using GTX 750 Ti GPU Engine having 640CUDA Cores processors with 1020Base Clock (MHz), 1085 Boost Clock (MHz), 5.4Gbps Memory Clock, and 2048MB Standard Memory with GDDR5Memory Interface. GPU is used under a single node Hadoop cluster 2.7.2 with Intel(R) Core(TM) I5-6600 3.30GHz CPU and 16 GB Memory and Windows 10 OS. Also, the Apache Spark 2.0.1 with Spark streaming and Spark engine modules is implemented over the Hadoop server in order to achieve real-time processing.

4.3 System Evaluation

Since we are more focusing on the real-time processing of the Big Data, therefore, we evaluated the proposed system by considering the system efficiency in terms of processing time and throughput. Moreover, we have mapped the MapReduce Hadoop programming mechanism in GPU using CUDA for image processing by dividing the whole datasets into a number of independent blocks. Hence we compared our approach to the implementation of traditional MapReduce implementation by performing various parameter calculation by considering both textual as well as media data.

At the start, we have chosen the numbers of raw data files of fixed length (65 MB). We have run the parameter calculation algorithm on these files using both MapReduce implementation as well as GPU-based proposed system implementation. We observed that for the textual raw file, the GPU implementation perform extraordinary than the traditional MapReduce implementation. For ten number of files, GPU just took almost 1 seconds, whereas CPU took 12 seconds. Moreover, when we increase the number of files, the processing time is also remarkably increased in the case of CPU implementation. For GPU-based implementation, the processing time starts increasing very gradually and the rise in processing time is very little as compared to the number of files increased. The processing time comparative analysis of the proposed GPU-based implementation and the traditional MapReduce implementation corresponding to the number of files processed is shown in Fig. 4. Similarly, we have also performed the efficiency analysis on text documents while considering the system throughput. The throughput almost remains constant for both of the cases with increasing data size. The throughput of the GPU-based implementation is quite higher, i.e., from 300 to 350 MBps. Whereas, the CPU-based MapReduce without GPU support only have a throughput of 50MBps, which is quite poor, as shown in Fig. 5. Therefore, we concluded that for text/number based big raw files, the GPU-based implementation is quite faster than the CPU MapReduce.

While evaluating the system efficiency on video related Big Data, the processing time is measured in seconds against the video duration, as shown in Fig. 6. It is obvious that with single CPU MapReduce-based implementation, the processing time is rapidly growing with the increase in the duration of the video. Whereas, with

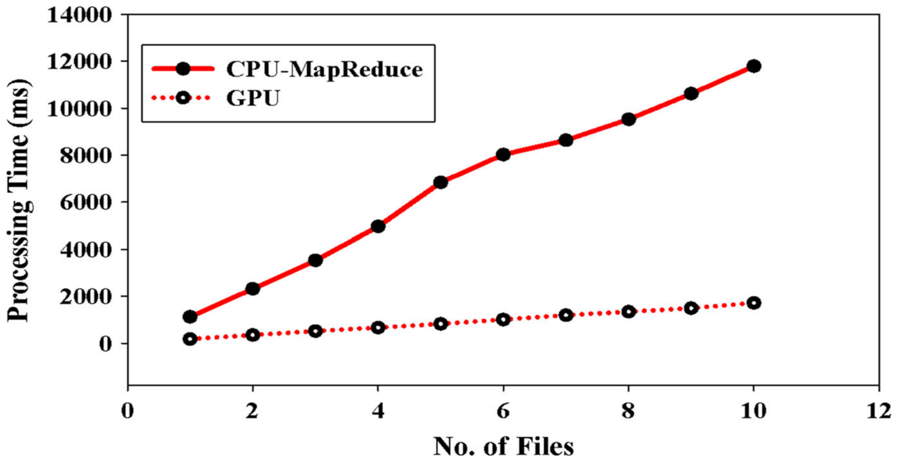

Fig. 4 Processing time with respect to number of large size text file

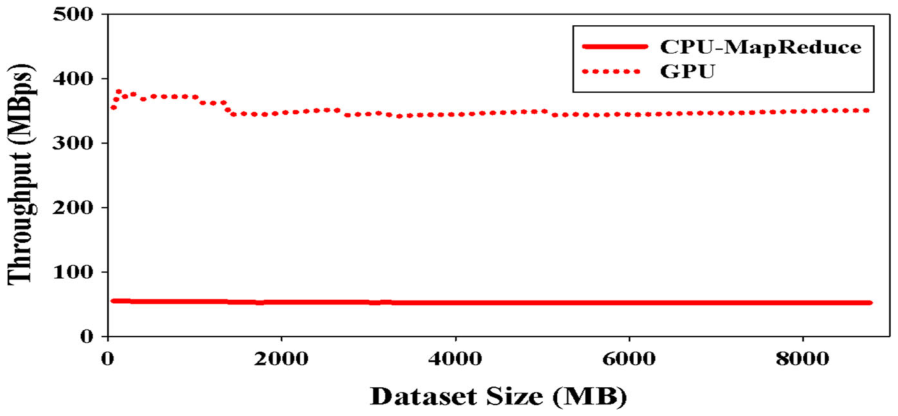

Fig. 5 System throughput with respect to data size

Spark and GPU based Hadoop system, there is a very short increase in the processing time when there is an increase of hundreds of seconds in the video duration. GPU processes the video almost seven times faster than the video duration. With these results, it is obvious that the proposed system is capable of processing real-time traffic videos.

Normally the videos are generated as 30 frames per seconds. The frame processing efficiency of the proposed system and the comparison with the MapReduce CPU implementation can be seen in Fig. 7. The proposed system takes almost four milliseconds (ms) to process one frame, which is quite low as compared to the time taken by the CPU MapReduce system to process one frame. The CPU MapReduce-based system take almost 18 ms to process one frame and almost more than half second to process one-second video (18 * 30 frames). This processing time is quite high in case of processing multiple videos at one time. So, standalone

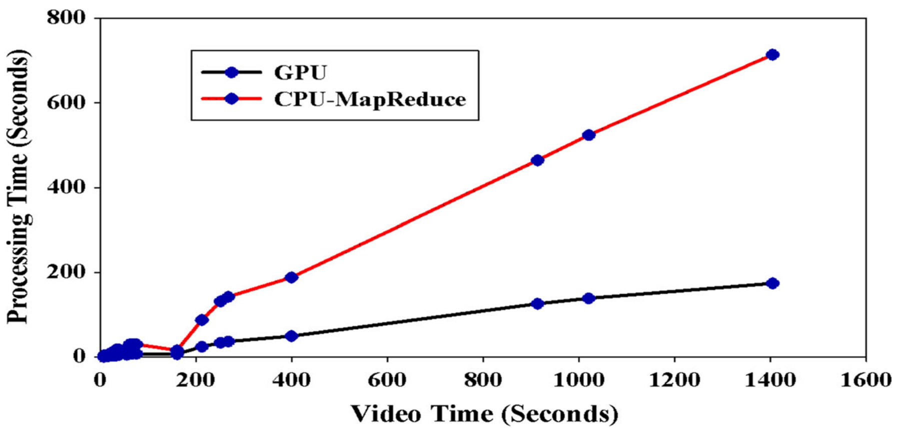

Fig. 6 Processing time with respect to video duration

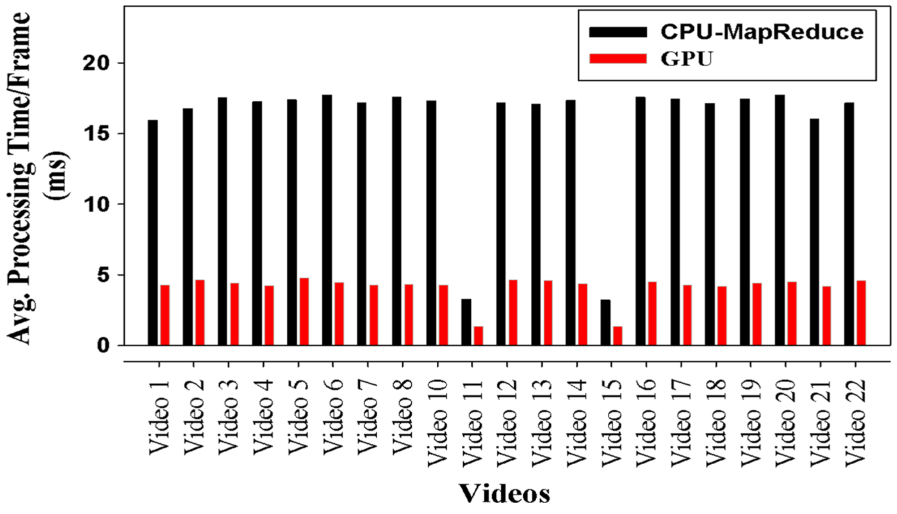

Fig. 7 Average processing time per frame with various videos

CPU MapReduce-based implementation would not be good enough for real-time video processing especially when videos are continuously coming from multiple sources.

Finally, we took the throughput of the system (frame processed per second) that is measured by dividing the total number of video frames divided by the total time taken to process the whole video. The throughput of both the GPU and CPU implementation is almost constant even with the increase in the number of frames, as depicted by the results in Fig. 8. The proposed system, with GPU-based implementation, have very high throughput as compared to the CPU implementation. The GPU processes 200–250 frames per seconds, which means it can process 7–8 live

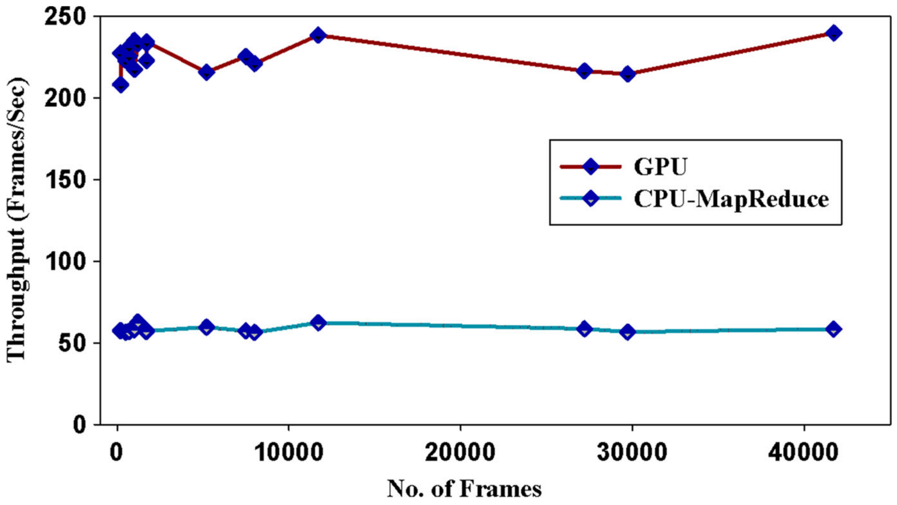

Fig. 8 Throughput with respect to number of frames processed per second

video at real-time. Whereas, The CPU has only the ability to process 1–2 video at a time.

5 Conclusion

In this paper, we proposed an efficient and real-time Big Data stream processing system to analyze the data and make immediate decisions. We proposed an implementation model by integrating parallel and distributed environment of Hadoop ecosystem with graphics processing unit (GPU) and Spark to make it more powerful and real-time in terms of processing. We also proposed MapReduce equivalent algorithm for efficient data processing using GPUs for parameters' calculation by dividing overall Big Data files into fixed-size blocks. Apache Spark streaming is employed to captured real-time data from remote locations and distribute it among various Hadoop data nodes using HDFS. Apache Spark engine is used to process that captured data in real-time. Each Data node is equipped with GPUs and corresponding algorithm to perform processing using iterative instructions and return back the results to Spark engine using the intermediate module. We evaluated our approach by taking the city traffic videos, i.e., captured by static as well as running vehicles' cameras, by identifying vehicles on the road and by taking large text-based files, like twitter data files, machine learning classification data, etc. Finally, we evaluated the proposed system on efficiency while considering the processing time and throughput with various aspects. The proposed system is proved more efficient as compared to the traditional standalone CPU based MapReduce.

Acknowledgements This study was supported by the Brain Korea 21 Plus project (SW Human Resource Development Program for Supporting Smart Life) funded by Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (21A20131600005).

References

1. Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J.M., Welton, C.: Mad skills: new analysis practices for Big Data. Proc. VLDB Endow. **2**(2), 1481–1492 (2009)
2. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
3. IBM, Armonk, NY, USA.: Four Vendor Views on Big Data and Big Data Analytics. IBM [Online]. http://www-Ol.ibm.comlsoftware/in/data/bigdata/ (2012)
4. CISCO.: The Internet of Things, Infographic. http://blogs.cisco.com/news/the-internet-of-things-infographic/ (2015)
5. Sivaraman, S., Trivedi, M.M.: Integrated lane and vehicle detection, localization, and tracking: a synergistic approach. IEEE Trans. Intell. Transp. Syst. **14**(2), 906–917 (2013)
6. Rathore, M.M., Ahmad, A., Paul, A., Jeon, G.: Efficient graph-oriented smart transportation using internet of things generated Big Data. In: 2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), pp. 512–519 (2015)
7. Ahmad, A., Paul, A., Rathore, M.M., Chang, H.: Smart cyber society: integration of capillary devices with high usability based on cyber-physical system. Future Gen. Comput. Syst. **56**, 493–503 (2016)
8. Rathore, M.M., Ahmad, A., Paul, A., Wan, J., Daqiang, Z.: Real-time medical emergency response system: exploiting IoT and Big Data for public health. J. Med. Syst. **40**(12), 283 (2016)
9. Rathore, M.M., Ahmad, A., Paul, A., Rho, S.: Urban planning and building smart cities based on the internet of things using Big Data analytics. Comput. Netw. **101**, 63–80 (2016)
10. Ahmad, A., Paul, A., Rathore, M.M.: An efficient divide-and-conquer approach for Big Data analytics in machine-to-machine communication. Neurocomputing **174**, 439–453 (2016)
11. Jin, J., Gubbi, J., Marusic, S., Palaniswami, M.: An information framework for creating a smart city through internet of things. IEEE Internet Things J. **1**(2), 112–121 (2014)
12. Apache Hadoop.: Welcome to Apache™ Hadoop®!. http://hadoop.apache.org/ (2016). Accessed 1 Nov 2016
13. Apache SPARK.: Apache Spark™. http://spark.apache.org/ (2016). Accessed 1 Nov 2016
14. Ailamaki, A., Govindaraju, N.K., Harizopoulos, S., Manocha, D.: Query co-processing on commodity processors. VLDB **6**, 1267–1267 (2006)
15. Hadoop.: http://ati.amd.com/technology/streamcomputing/ (2010). Accessed 1 Nov 2016
16. Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., Kozyrakis, C.: Evaluating mapreduce for multi-core and multiprocessor systems. In: IEEE 13th International Symposium on High Performance Computer Architecture 2007. HPCA 2007, pp. 13–24 (2007)
17. Cerotti, D., et al.: Modeling and analysis of performances for concurrent multithread applications on multicore and graphics processing unit systems. Concurr. Comput. Pract. Exp. **28**(2), 438–452 (2016)
18. Qureshi, M.K., Patt, Y.N.: Utility-based cache partitioning: a low-overhead, high-performance, runtime mechanism to partition shared caches. In: Microarchitecture. 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on IEEE (2006)
19. Kavadias, S.G. et al.: On-chip communication and synchronization mechanisms with cache-integrated network interfaces. In: Proceedings of the 7th ACM International Conference on Computing Frontiers. ACM (2010)
20. Liu, F., Xiaowei J., Solihin, Y.: Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance. In: High Performance Computer Architecture (HPCA). 2010 IEEE 16th International Symposium on IEEE (2010)
21. D'Amore, L., et al.: HPC computation issues of the incremental 3D variational data assimilation scheme in OceanVar software. J. Numer. Anal. Ind. Appl. Math. **7**(3–4), 91–105 (2012)
22. Che, S., et al.: A performance study of general-purpose applications on graphics processors using CUDA. J. Parallel Distrib. Comput. **68**(10), 1370–1380 (2008)
23. Owens, J.D., et al.: GPU computing. Proc. IEEE **96**(5), 879–899 (2008)
24. Gregg, C., Hazelwood K.: Where is the data? Why you cannot debate CPU versus GPU performance without the answer. In: Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on IEEE (2011)
25. Shi, L., et al.: vCUDA: GPU-accelerated high-performance computing in virtual machines. IEEE Trans. Comput. **61**(6), 804–816 (2012)
26. Aldinucci, M., et al.: Parallel visual data restoration on multi-GPGPUs using stencil-reduce pattern. Int. J. High Perform. Comput. Appl. **29**(4), 461–472 (2015)

27. Wu, W., et al.: Hierarchical dag scheduling for hybrid distributed systems. In: Parallel and Distributed Processing Symposium (IPDPS), 2015 International IEEE (2015)
28. Song, F., Dongarra, J.: A scalable approach to solving dense linear algebra problems on hybrid CPU-GPU systems. Concurr. Comput. Pract. Exp. **27**(14), 3702–3723 (2015)
29. Du, P., et al.: Soft error resilient QR factorization for hybrid system with GPGPU. J. Comput. Sci. **4**(6), 457–464 (2013)
30. Dongarra, J., et al.: Hpc programming on intel many-integrated-core hardware with magma port to xeon phi. Sci. Program. **2015**, 9 (2015)
31. Braun, T.D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
32. Anderson, E., et al.: LAPACK Users' guide. In: Society for Industrial and Applied Mathematics (1999)
33. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. SIAM, Philadelphia (1999)
34. Agullo, E., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Langou, J., Ltaief, H., Luszczek, P., YarKhan, A.: Plasma Users' Guide, Technical report. In: ICL, UTK (2014)
35. Blackford, L.S., Choi, J., Cleary, A., D'Azeuedo, E., Demmel, J., Dhillon, I., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK User's Guide. In: Society for Industrial and Applied Mathematics, Philadelphia (1997)
36. Song, F., YarKhan, A., Dongarra, J.: Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. pp. 1–11 (2009)
37. Ahmad, A., et al.: Multilevel data processing using parallel algorithms for analyzing Big Data in high-performance computing. Int. J. Parallel Program. doi:10.1007/s10766-017-0498-x (2017)
38. Rathore, M.M., et al.: Exploiting encrypted and tunneled multimedia calls in high-speed Big Data environment. Multimed. Tools Appl. doi:10.1007/s11042-017-4393-7 (2017)
39. NVIDIA ACCELERATED COMPUTING.: CUDA Toolkit 8.0. https://developer.nvidia.com/cuda-downloads (2016). Accessed 1 Nov 2016
40. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of Sixth Conference Symposium on Opearting Systems Design and Implementation (OSDI) (2004)
41. Arlingtonva.us.: Live traffic cameras. https://transportation.arlingtonva.us/live-traffic-cameras/ (2016). Accessed 1 Nov 2016
42. 43Earth Cam.: LIVE Webcam Network. http://www.earthcam.com/ (2016). Accessed 1 Nov 2016