LARGE SCALE IMAGE PROCESSING IN REAL-TIME ENVIRONMENTS WITH KAFKA

Yoon-Ki Kim[1] and Chang-Sung Jeong[2]

[1,2]Department of Electrical Engineering, Korea University, Seoul, South Korea
[1]vardin@korea.ac.kr
[2]csjeong@korea.ac.kr

ABSTRACT

Recently, real-time image data generated is increasing not only in resolution but also in amount. This large-scale image originates from a large number of camera channels. There is a way to use GPU for high-speed processing of images, but it cannot be done efficiently by using single GPU for large-scale image processing. In this paper, we provide a new method for constructing a distributed environment using open source called Apache Kafka for real-time processing of large-scale images. This method provides an opportunity to gather related data into single node for high-speed processing using GPGPU or Xeon-Phi processing.

KEYWORDS

Real Time Image Processing, Distributed Processing, Real-Time Processing, Apache Kafka

1. INTRODUCTION

Recently, the development of video equipment such as CCTV, satellite, and drone has increased the volume of real-time image data. Large Scale of real-time image data coming from many camera channels is difficult to process at high speeds with single resources such as CPU and GPU. GPU has been developed for high-speed image processing, however it is not enough for large-scale image processing because of the limit of memory shortage. There is also an approach of HIPI [1] to large-scale image processing using the HDFS, but this is also inadequate for real-time image processing due to the limitations of the batch processing. In addition, the HDFS [2] uses a random access approach to the disk. It causes HDD I/O bottleneck. For this reason, HIPI is insufficient to process the real-time data continuously.

In order to process a large-scale image occurring in real-time, it is necessary to have a processor and memory capable of accommodating it. Large-scale image processing using multicore and GPU in a single node cannot accommodate data due to insufficient memory. To compensate for this, it is essential to distribute processing for large-scale images using cluster composed of several nodes. There are two major issues in approaching this way. The first is the memory acceptance issue at a single resource such as main memory and GPU. As the amount of data that can be accommodated in a node increases and reaches the limit of the memory capacity, the processing speed is rapidly reduced [3]. Since the data processing speed is slower than the data occurring speed, it is important not to lose the occurring data while the data is being processed. Second issue is the communication overhead between divided tasks. When a large task is divided into small tasks and distributed processing is performed, communication occurs between the divided tasks. The more associations between divided tasks, the more communication will occur. Communication overhead is caused by dividing data between each task into more than necessary pieces [4]. This problem becomes more severe when the task is divided into highly correlated tasks. For this reason, it is important to divide tasks appropriately in order to prevent a lot of communication between divided tasks.

Apache Kafka provides features that are suitable for addressing the issues mentioned previously. Kafka is distributed messaging system for log processing. Kafka uses Zookeeper [5] to organize several distributed nodes for storing data in real time stably. It also stores data in several messaging topics. It provides a messaging queue that allows nodes which needs a message to access and process it.

In this paper, we propose a new method to process large scale image data in real-time using Kafka. This method enables large-scale image data in real time to be processed quickly. In this method, large-scale images can be processed in real time with a resilient scale-out of computing resources.

The outline of our paper is as follows: In Section 2, we describe related works for introducing Apache Kafka and some approaches to handling large-scale images in real time. Section 3, we explain a new method to process large scale image data in real-time using Kafka. Section 4 explains implementation of proposed method and shows its experimental results. Lastly Section 5 summarizes the conclusion of our research.

2. RELATED WORKS

Apache Kafka [6] can be used to collect various types of data in real time. Kafka is an advanced open source project, and its ability to handle high-volume streams such as Internet of things and log data. It performs better than existing messaging system [7-9] with specialized architecture for large-scale real-time log processing. It is suitable for both offline and online message usage. And it is based on publish-subscribe model and consists of a producer, a consumer, and a broker. Producer generates a message and publishes it to a specific topic. And the consumer takes it and processes it. A broker is a server cluster that manages messages so that producers and consumers can meet, and categorizes messages which received from producer. It can be configured as multiple broker Clusters, and each node is monitored by Zookeeper.

There are several advantages to adopting Kafka in this paper. Firstly, since messages are stored in the file system, the durability of the data is ensured without any special configuration. Kafka uses sequential access to the HDD, resulting in faster performance than memory random access methods. Performance comparison of HDD sequential access method and memory random access is compared in [10]. In the existing messaging system, the performance of the system decreases drastically as the number of messages left unprocessed increases. However, since Kafka stores the message in the file system, the performance is not greatly reduced even if a large amount of messages are accumulated. It can also be used to accumulate data for periodic batching as well as real-time processing, since many messages can be stacked. Secondly, Kafka provides fault tolerance by storing messages in the file system and replicating them to the broker. Unlike an existing messaging system that immediately deletes an acknowledged message from a consumer, it does not delete the processed message but leaves it in the file system and deletes it after a set lifetime. Since the processed message is not deleted for a certain period of time, the consumer can rewind the message from the beginning if a problem occurs during processing of the message or if the processing logic is changed. Thirdly, Kafka provides storage capacity for large-scale data using pulling method. In traditional messaging systems, brokers push messages to consumers, while Kafka acts as a pull consumer, taking messages directly from the broker. Therefore, a consumer can get optimal performance by fetching only the messages of his processing capacity from the broker. Push-based messaging system, the broker directly calculates which messages each consumer should process, and tracks which messages are being processed. In Kafka, the consumer pulls the necessary messages directly from the broker. By pulling messages from broker, it is possible to build a batch consumer that stacks and periodically processes messages. For these reasons, we decided to adopt Kafka in the process of handling large-scale image processing on distributed environments.

3. METHODOLOGY

Real-time stream data is constantly generated so as not be accommodated in the memory continuously. Parallel processing of such a large amount of stream data in a single node is limited by the memory capacity limit. Also, communication overhead is caused by frequent data transmission between the main processor and the coprocessor. Performing only distributed processing on multiple nodes also causes communication overhead due to large-scale parameter exchange between nodes in the cluster. In order to solve this problem, we present a new architecture and several methods for large scale image processing using apache Kafka.

The model for real-time processing of large-scale images consists of three parts. The first is the part that detects the frame from the stream channel. The stream channel may be a channel directly connected to the camera or a remote channel transmitted through the RTSP protocol. This part is responsible for producing messages and consists of several nodes. One node can sense data coming from one or several channels. Depending on the resource capability of each node, it is determined how many channels can be detected in one node. In the part detecting the channel, the frame of the video is transmitted to the Kafka broker.

The second part is a Kafka broker. In this part, a large amount of stream data transmitted from the previous part is stored in a buffered queue. Generally, the processing speed of the stream is slower than generation rate. For this reason, in order to process a large amount of video data without loss, it is necessary to temporarily store the detected stream data. We used Kafka for a stream buffer. There are several reasons why we used Kafka to store large amounts of video in temporary buffers. The reasons we adopted Kafka in buffering large image processing is as follows.

- Since the video message is stored in the file system, the durability of the data is guaranteed. This advantage allows data to be permanently stored in the buffer, even if none of the nodes can process the video stream. The video stream is deleted from the file system at the time set by the user. In addition, since the video stream generated in real time has a large capacity, it can overcome the disadvantage that it is difficult to be loaded in the main memory.

- Instead of distributing messages to the distributed nodes in the master node, each node ready to process takes a message from the buffer and processes it. In this condition, even if the performance of each node is different, it is possible to perform high-speed processing without bottleneck of stream data because the node ready for processing processes the data.

The last part is that each node has an image processing application, which takes and processes stream data from the Kafka broker. The image processing application at each node is intended to perform the same operation. Since each node has one frame at a time, one frame can be seen as a set of related data. In the text processing system, data is transmitted or processed in units of tuples. In the proposed system, the image processing unit is regarded as a frame. It is a feature of image processing that many repetitions of matrix or vector operations are included. It can be processed quickly by using GPU accelerator considering frame as basic unit of processing. In this part, we use the asynchronous method of processing data regardless of the order of image generation.

We construct the model of this system with the three parts mentioned above, and this model is expressed as the following figure 1.

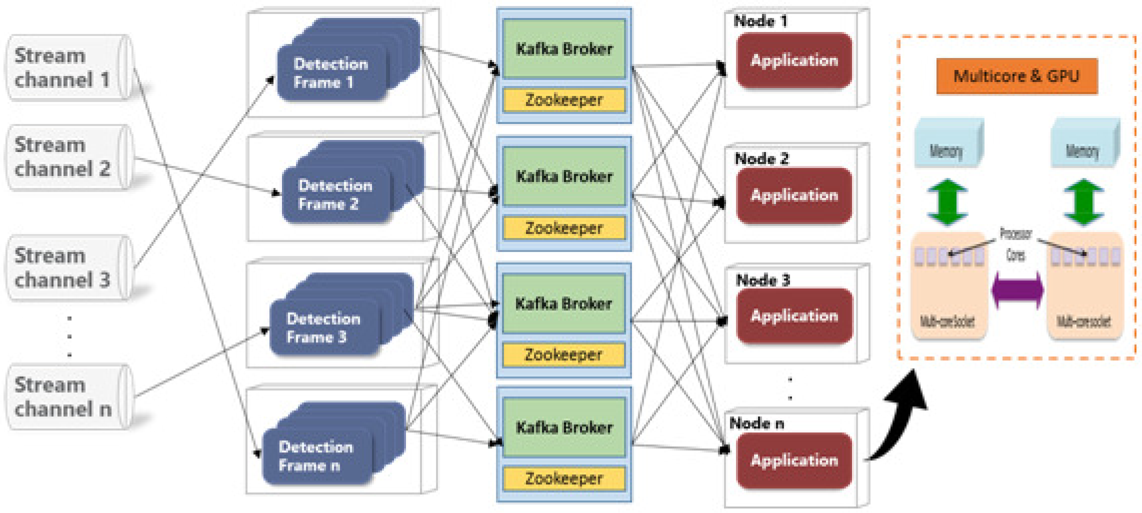

Figure 1. The overall model of large scale image processing using Kafka

3.2 Strategies

To process large-scale images in real time, we present several strategies as follows.

(1) Since the same operation is performed at the application level, the resolution of the image must be synchronized. When an image is distributed to multiple distributed resources, the image is shuffled. Therefore, when images of different resolutions are transmitted to a distributed environment, the application may require excessive workload, which may degrade performance. For this reason, the size information of the image must be transmitted to the application level in the part detecting the frame.

(2) The unit transmitted at one time is the frame of the image. In general, one frame transmitted from a camera channel has multiple channels. One frame is serialized into a byte array for transmitting to the Kafka broker. When the byte array is transmitted from the Kafka broker, the original image is obtained by re-encoding based on information such as the resolution of the image and the number of channels.

(3) Transfer image data asynchronously to Kafka Broker. This is a strategy to increase image processing speed in distributed resource environment composed of multiple nodes. This results in faster image processing, while the order of the images can be shuffled. It can be used for the purpose of identifying an object in a video generated in a large camera channel and storing the value permanently by taking only the information of the identified object.

(4) Take advantage of multiple distributed nodes to retrieve data from topics and parallelize them. A frame from one channel can be sent to multiple topics to allow distributed nodes to retrieve and process data from each topic. Alternatively, multiple nodes can access and process data with the same topic. The larger the number of channels, the greater the memory requirement for processing video frames, which increases the size of the node and maximizes memory capacity.

3.2.1 Transmission of frame

As shown in Figure 2, it is necessary to serialize the data to transfer the frame to the Kafka broker. The data coming from the channel has three channels in matrix form. In order to transmit this, a matrix composed of three channels of RBG is converted into a byte array. When this process is performed, the height and width information of the frame is lost as to how many channels are formed by one frame. This information is needed to re-encode the image at the application level, so once the channel is detected, it should be sent only once at the moment of connection with the Kafka broker.

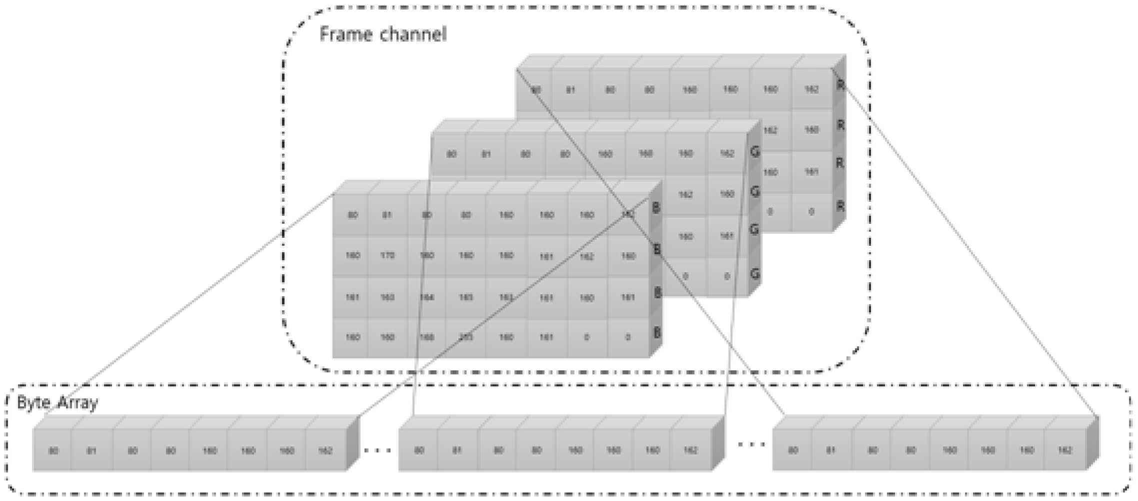

Figure 2. Conversion of frame to byte array

3.2.2 Distribution of image channel

Channels coming in on multiple channels will go into different queues through topics provided by Kafka's brokers. Figure 3 below shows how each channel is distributed across multiple topics. Multiple applications on distributed resources process data according to the resource situation and then take data by accessing the Kafka topic and bringing the message directly. As shown in the figure, the message can be retrieved by accessing multiple topics from one node, and the message stored in one topic can be distributed by allowing access by multiple nodes.

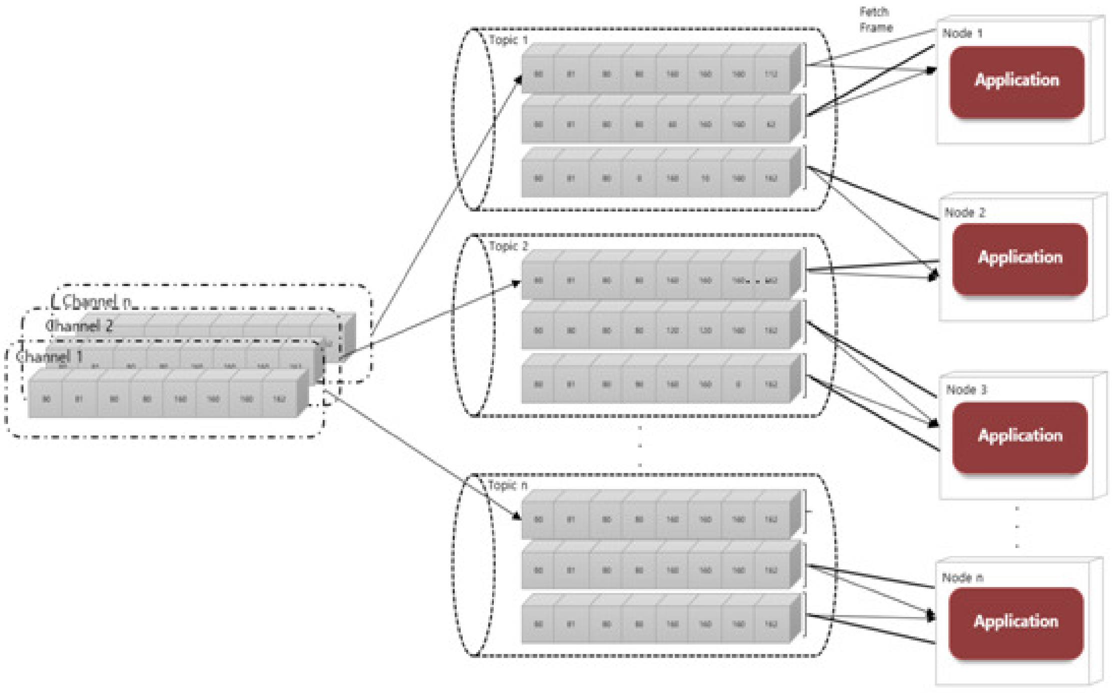

Figure 3. Distribution of each channel frame

4. Experimental Results

We configured the Kafka broker with three physical nodes which has Intel® core™ CPU E6550 2.33 GHz processors and 4GB memory. Nodes at the application level have been run in a virtual environment to make it easier to reshape the number of nodes. Each node in the application level has a quad core virtual CPU and 8GB memory.

In the experiment, the image is processed 60 seconds after the channel is generated. The following accumulate means the total number of frames generated. And consume is the number of frames taken from the Kafka broker. Finally, lag means the number of frames remaining in the buffer without being processed. As shown in Figures 4, 5, and 6 below, the lag drops sharply as the number of processing nodes increases. As the number of nodes increases, the difference between the number of frames accumulated in the buffer and the number of processed frames decreases.

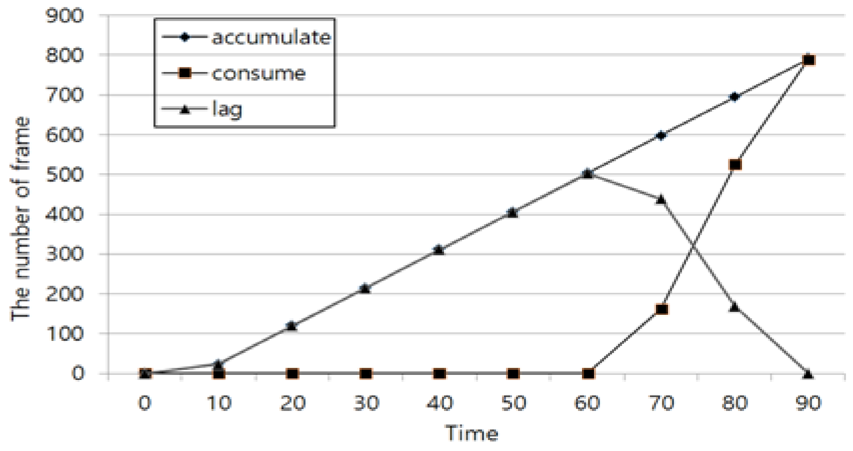

Figure 4. The result of one topic and one node

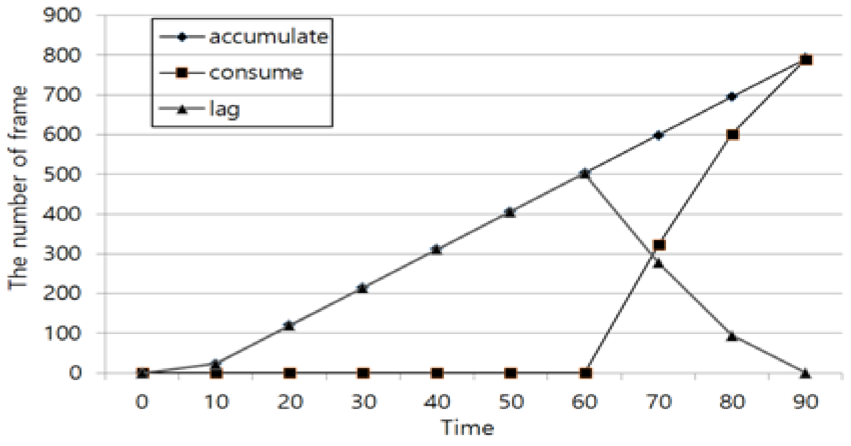

Figure 5. The result of two topics and two nodes

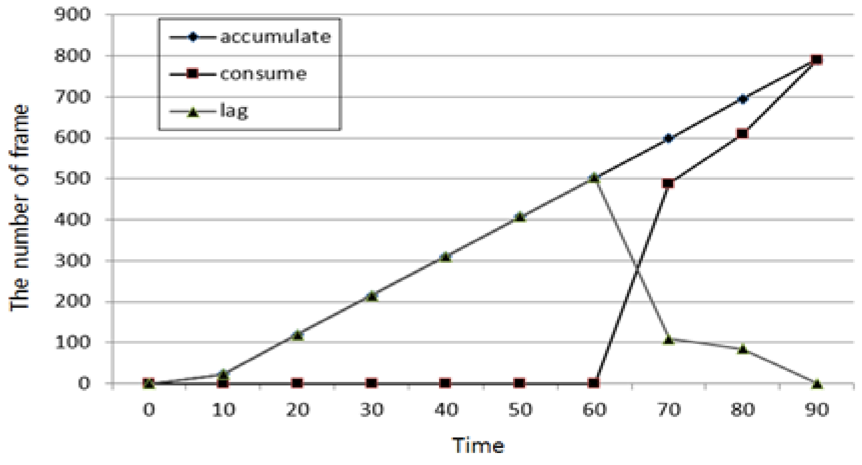

Figure 6. The results of four topics and four nodes

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a new large-scale image processing system with Kafka. Existing systems use GPGPU on a single node to process images for high performance in real-time environments. However, this method is not suitable for large-scale image processing. For large-scale image processing, memory capacity must be available to accommodate large-scale images. Although the GPU is well suited for high-speed processing of images, it still has limited memory capacity. We suggested new large-scale image processing model that use a Kafka to create a distributed environment. It allows overcoming the memory capacity that cannot be accommodated by one node. In particular, since image data can be stored in the file system, it is advantageous to handle large-scale images without data loss. In this paper, we have not been able to perform experiments that use GPUs on a single node to achieve even greater performance. Future research will focus on how to use GPUs or Xeon-phi for each node to achieve higher performance in large-scale image processing.

ACKNOWLEDGMENTS

This work was supported by the Brain Korea 21 Plus Project in 2016 and Institute for Information & communications Technology. Promotion (IITP) grant funded by the Korea government (MSIP) (No. R0190-16-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development).

REFERENCES

[1] Sweeney, Chris, et al. "HIPI: a Hadoop image processing interface for image-based mapreduce tasks." Chris. University of Virginia (2011).

[2] Shvachko, Konstantin, et al. "The hadoop distributed file system." 2010 IEEE 26th symposium on mass storage systems and technologies (MSST). IEEE, 2010.

[3] Gregg, Chris, and Kim Hazelwood. "Where is the data? Why you cannot debate CPU vs. GPU performance without the answer." Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on. IEEE, 2011.

[4] Xu, Zhiwei, and Kai Hwang. "Modeling communication overhead: MPI and MPL performance on the IBM SP2." IEEE Parallel & Distributed Technology: Systems & Applications 4.1 (1996): 9-24.

[5] Hunt, Patrick, et al. "ZooKeeper: Wait-free Coordination for Internet-scale Systems." USENIX Annual Technical Conference. Vol. 8. 2010

[6] Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." Proceedings of the NetDB. 2011.

[7] Snyder, Bruce, Dejan Bosnganac, and Rob Davies. ActiveMQ in action. Vol. 47. Manning, 2011.

[8] Rostanski, Maciej, Krzysztof Grochla, and Aleksander Seman. "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ." Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on. IEEE, 2014.

[9] Hintjens, Pieter. ZeroMQ: Messaging for Many Applications. " O'Reilly Media, Inc.", 2013.

[10] Jacobs, Adam. "The pathologies of big data." Communications of the ACM52.8 (2009): 36-44.

AUTHORS

Yoon-Ki Kim is currently working toward the ph.D degree in Electronic and Computer Engineering at the Korea University. His research interests include real-time distributed and parallel data processing, IoT, Sensor processing and computer vision.

Chang-Sung Jeong is a professor at the department of EE/CE at Korea University. He received his MS.(1985) and Ph.D.(1987) from Northwestern University, and B.S.(1981) from Seoul National University. Before joining Korea University, he was a professor at POSTECH during 1982-1992. He also worked as an associate researcher at UCSC during 1998-1999.