SVIS: Large Scale Video Data Ingestion into Big Data Platform

Xiaoyan Guo(✉), Yu Cao, and Jun Tao

EMC Labs China, Beijing, China
{xiaoyan.guo,yu.cao,simon.tao}@emc.com

Abstract. Utilizing big data processing platform to analyze and extract insights from unstructured video streams becomes emerging trend in video surveillance area. As the first step, how to efficiently ingest video sources into big data platform is most demanding but challenging problem. However, existing data loading or ingesting tools either lack of video ingestion capability or cannot handle such huge volume of video data. In this paper, we present *SVIS*, a highly scalable and extendable video data ingestion system which can fast ingest different kinds of video source into centralized big data stores. *SVIS* embeds rich video content processing functionalities, e.g. video transcoding and object detection. As a result, the ingested data will have desired formats (i.e. structured data, well-encoded video sequence files) and hence can be analyzed directly. With a highly scalable architecture and an intelligent schedule engine, *SVIS* can be dynamically scaled out to handle large scale online camera streams and intensive ingestion jobs. *SVIS* is also highly extendable. It defines various interfaces to enable embedding user-defined modules to support new types of video source and data sink. Experimental results show that *SVIS* system has high efficiency and good scalability.

1 Introduction

With rapid development of big data technologies, more and more vertical industries, such as Internet, Finance, Healthcare, are placed into big data processing platform to extract hidden values from their data by utilizing state-of-the-art big data analytics [1]. Video surveillance analytics is another emerging trend to extract insights from unstructured video stream data [2]. Surveillance videos are in nature of great magnitude, and hence require the powerful processing capability of big data platform.

However, loading or ingesting video surveillance data, the first step before data can be analyzed by big data platform, is particularly challenging [3]. Firstly, unstructured video data is of diverse types of sources (e.g. online camera streams, historical video files) and encoding formats (e.g. H.264, MJPEG, MPEG4). How to automatically and efficiently transcode and transform video data to be tackleable by big data platform remains as a challenging problem. Secondly, video data is of huge volume. The resolution of IP cameras is increasing rapidly. There are extremely huge number of online cameras in city-wide surveillance systems.

A. Liu et al. (Eds.): DASFAA 2015 Workshops, LNCS 9052, pp. 300–306, 2015.
DOI: 10.1007/978-3-319-22324-7_31

As such, ingesting systems must be scalable and efficient enough to handle huge volume of video data and intensive ingestion workloads. Thirdly, certain video content analytics work need to be conducted within ingestion systems in order to obtain timely analytics results, e.g. real-time determining potential incidents. As a result, it is desirable or even necessary that the ingestion system has real-time video processing functionalities.

There are many existing systems in big data ecosystem to realize data loading or data ingestion work. Flume [4], Sqoop [5], Kafka [6], Scribe [7] and Chukwa [8] are all distributed, reliable and available systems to efficiently ingest large amounts of data, especially log data, from different data sources into centralized big data stores, such as HDFS, HBase, relational SQL database, etc.

However, all these systems are not perfectly suitable to address video ingestion jobs. They lack video processing capability, and hence it is difficult to ingest video data especially video streaming data directly from IP cameras. They do not have good flexibility and extendability so that it is difficult to extend them to support new types of data source and data sink. These systems usually utilize batch loading, emphasizing more on ingestion throughput rather than latency. While in video surveillance area, real-time ingestion is a desired feature.

In this paper, we thereby present *SVIS*, a scalable and extendable video data ingestion system which can fast ingest diverse video sources into big data stores. *SVIS* integrates rich video processing functionalities. It is able to transcode and transform video data of different source types, and then directly pipeline them to video analytics applications. *SVIS* can easily scale out to support large scale video surveillance systems. It can be conveniently extended to support new video sources and data sinks.

Our principal contributions in this paper are summarized as follows.

1. We design and build a highly scalable video ingestion framework, which is able to handle huge volume of video data and extremely intensive ingestion jobs.
2. We integrate rich video content processing functionalities into our ingestion system, e.g. transcoding video into frame sequence files and detecting objects within video frames. As a result, data ingested into big data store is exactly of the desired format that can be analyzed in the straight-forward way. Real-time analytics at edge nodes can also be easily realized.
3. We define various extendable interfaces to support user-defined modules. As a result, the users can develop their own modules and embed them into the system to support specific video sources and sinks.
4. We define a video ingestion DSL (Domain-Specific Language) to allow users to define their own ingestion jobs by declarative interfaces. We build an intelligent schedule engine to guarantee efficiency and fault-tolerance of the ingestion jobs.
5. We conduct comprehensive experiments to demonstrate the efficiency and scalability of our video ingestion system in two scenarios of ingesting large historical video files and intensive online camera streams.

2 Video Data Ingestion System

The *SVIS* video data ingestion system moves huge volume of video data from different sources into a centralized repository. Figure 1 shows the deployment topology of *SVIS* system in a real video surveillance system. Data sources of *SVIS* system are not only historical video files but also online video streams, e.g. online IP cameras. *SVIS* system ingests video data not only within campus local network but also from wide area network. It is able to ingest video data into diverse data sinks, such as HDFS, structured database, and NoSQL data store. It is also capable to directly ingest and pipeline video data to online video analytics applications.

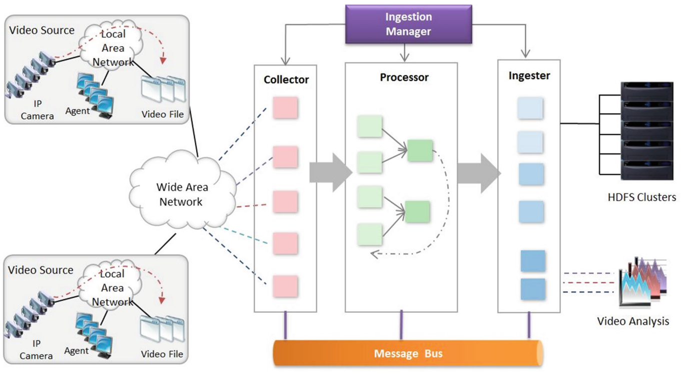

Fig. 1. System architecture

Ingestion Job Abstraction. To enhance system scalability and provide flexible and extendable interfaces, we abstract the ingestion job into three sub-tasks, which encapsulates the same kind of work unit into reusable modules. The three abstractive ingestion modules are *collector*, *processor* and *ingester*.

The collectors implement drivers to collect data from different types of data sources. They poll data from external video sources and then pipeline them to other modules. The first module in an ingestion job must be a collector.

The processors include rich built-in video processing functionalities, such as video transcoding, object detection and feature extraction. They receive video data from the collectors and perform video processing and video analytics at the edge node. The analytics results of processors are delivered to the ingesters or other processors. The processor is optional in an ingestion job. There are also multiple processors in a job to conduct a series of video processing tasks.

The ingesters implement drivers to write data into different data stores. They receive video data or analytics results from collectors or processors, and then ingest them into desired data sinks. The ingester is always the last module in an ingestion job.

In our video ingestion system, a video ingestion job is represented as a DAG, where the vertices are the instances of abstractive ingestion modules (i.e. collectors, processors, and ingesters), the edges are the established message queues to transfer data between two instances. An ingestion job is first scheduled into a DAG by the schedule engine and then executed accordingly. As such, the video data flows from video sources to collectors, processors and ingesters and finally arrives in data sinks.

System Overall Architecture. The middle of Fig. 1 depicts the high-level system architecture of *SVIS* video ingesting system, which consists of the following three major functional components: *ingestion manager*, *ingestion workers* and *message bus*.

Ingestion manager is responsible for scheduling job execution, launching ingestion jobs, monitoring runtime status of ingestion jobs and managing the whole ingestion cluster. By effectively scheduling and distributing an ingestion job to multiple ingestion workers and monitoring their status, it guarantees high scalability, load balance and fault-tolerance of the ingestion system.

Ingestion workers are responsible for executing the ingestion jobs actually. An ingestion worker is a node to execute one of the three ingestion modules mentioned in above section, i.e. collector, processor and ingester. The ingestion workers are scheduled by the ingestion manager. An ingestion job is represented as a DAG. For each vertex, the ingestion manager launches a corresponding ingestion worker to do the task. The ingestion workers can be dynamically added or removed according to the ingestion workload. It guarantees good flexibility and scalability of the ingestion system.

The ingestion workers share and transfer data via a configured messaging queue, the message bus shown in Fig. 1. The message bus provides multiple messaging patterns according to different ingestion topologies. It guarantees great reliability and high availability of the ingestion data.

The complete workflow of an ingestion job is described as follows. The ingestion manager accepts the ingestion request and ingestion job definition from the user. It then schedules and breaks down the ingestion job into a DAG ingestion topology, i.e. independent ingestion tasks, which can be directly executed by the ingestion workers. According to the ingestion topology, the ingestion manager launches a series of ingestion workers and distributes the ingestion tasks to them. The ingestion workers understand the task definitions and execute them in coordination of the ingestion manager. The ingestion workers transfer data via message bus and ingest video data into desired data sinks. During the ingestion, ingestion manager periodically collects tuntime status of each ingestion worker and monitor the ingestion progress. Once a task is failed, it restarts the corresponding ingestion worker and re-execute the task.

Video Ingestion Extendable API. There are varied video sources and video sinks in different video surveillance systems. It is impossible to implement a one-size-fits-all ingestion system to support all the data sources and sinks. As such, we abstract the video ingestion job into three fundamental modules, i.e. collector, processor and ingester. For each module, we define abstractive interfaces for the

```yaml
collector:
    - name: collector1
      sourceType: videoFile
      moduleType: VideoFileFrameCollector
      parallelism: 1
      outputQueue: FaceDetectionQ1
      property:
          host:
              - "10.62.98.123"
          filePath:
              - "/root/videoclips/7fgate.mp4"

processor:
    - name: processor1
      moduleType: FaceDetectionProcessor
      parallelism: 1
      inputQueue: FaceDetectionQ1
      outputQueue: FaceDetectionQ2
```

```yaml
ingester:
    - name: ingester1
      sinkType: gfxd
      moduleType: FaceGFXDIngester
      parallelism: 1
      inputQueue: FaceDetectionQ2
      property:
          jdbcURL:
              - "jdbc:gemfirexd://phd10:1527/"
          tableName:
              - "FaceDetectionCam4"
          persistProperty:
              - "HDFS"
queuepipe:
    - name: FaceDetectionQ1
      type: rabbitMQ

    - name: FaceDetectionQ2
      type: rabbitMQ
```

Fig. 2. DSL example of the ingestion job

users to implement user-defined modules to extend the system to support new specific types of data source and data sink.

For collector module, the users need to implement the interface to obtain data from data source. For ingester module, the users need to implement the interface to write data into corresponding data sink. For processor module, the users need to implement the interfaces to process and analyze the video data in order to gain intermediate results. After defining a module, the user can embed it into the ingestion system. The module will then be available and can be included in an ingestion job.

Video Ingestion DSL. We define a video ingestion DSL (Domain-Specific Language) to allow users to define their own ingestion jobs by declarative interfaces. With the declarative language, the users can define an ingestion job in a quite straight-forward way. The users can also be able to dynamically submit or stop an ingestion job easily.

With the declarative language, the users need to define the properties of collectors, ingesters and desired processors, desired parallelism degree, and queue pipe topology among these modules. All the definitions can be configured in a YAML or JSON configuration file. Figure 2 shows an example of YAML configuration file of ingesting a video file into HDFS and conducting face detection processing in the meantime.

3 Experimental Study

Experiment Setup. We setup a 10-node video ingestion cluster on 10 physical servers with 16 GB RAM and Xeon E5-2640@2.00 GHz 4 core CPU. They were connected by 1 Gbps network. We deployed 1 ingestion manager node, 8 ingestion worker nodes, and 1 RabbitMQ server node serving as the message bus.

There were two types of video sources: video files and online camera streams. We generated video files of different encodings and sizes. We also setup 20 IP

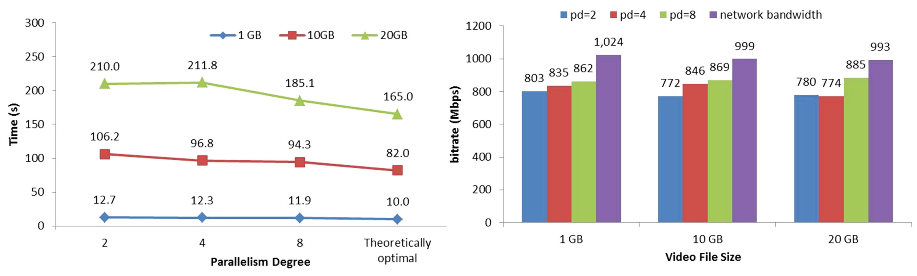

Fig. 3. Experimental results of ingesting video files into HDFS

cameras to provide online RTSP stream data. These cameras were configured with FPS of 20 and frame resolution of 800*600.

There were two types of data sinks in our experiments: HDFS and Gemfire XD [9]. The write speed of HDFS was about 50 MB/s. Gemfire XD is a main-memory based, distributed low latency data store for structured data and key-value data. HDFS and Gemfire XD were both deployed in a 10-node cluster.

Efficiency of Ingesting Large Video Files. We conducted experiments to ingest large volume of video files into HDFS and studyed ingestion speed and efficiency. We generated three video files with size of 1 GB, 10 GB and 20 GB and parallel ingested them with different parallelism degrees. Greater parallelism degree indicates that more ingestion workers are employed by *SVIS* ingestion system. Figure 3 shows the experimental results: left figure shows the total ingestion time of each file, right figure shows the achieved ingestion bitrate of each file.

Due to write speed of HDFS and network bandwidth, the largest throughput is bounded at 1000Mbps. From the experimental results, we can find that with a proper parallelism degree, the ingestion system can reach particularly high ingestion throughput that is about 85 % of the optimal throughput.

Latency and Scalability of Ingesting Intensive Online Camera Streams. We conducted experiments to ingest online camera streams into Gemfire XD. We increased ingestion workload (by increasing the ingested camera number from 1 to 20) and evaluated the latency and throughput of our ingestion system. Figure 4 shows the experimental results: left figure shows the average ingestion latency and total FPS achieved for each workload, right figure shows the achieved detail FPS of each camera when ingesting the 20-camera workload.

The latency represents the time interval of a video frame between generated by the camera and ingested into Gemfire XD. While camera number and total workload are increasing, the average latency is slightly increasing. However, the largest latency is bounded at 40ms, which is less than the generation interval of frames (i.e. 1000 ms / 20fps = 50 ms). As such, our ingestion system can achieve real-time ingestion capability that is able to catch up with the generation speed of camera frames.

In addition, the total FPS is linear increasing along with the increasing of ingestion workload (i.e. camera number). It proves that there is no frame loss during the ingestion. We can hence conclude that the overall performance of our *SVIS* video ingestion system is linearly scalable.

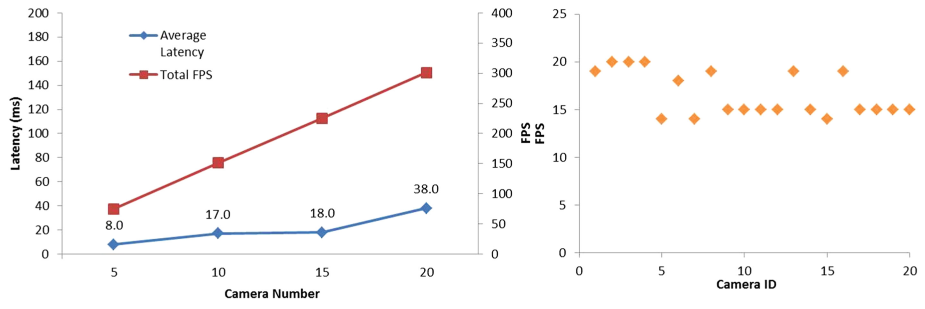

Fig. 4. Experimental results of ingesting online camera streams into gemfire XD

References

1. Han, Hu, Wen, Yonggang, Chua, Tat-Seng, Li, Xuelong: Toward scalable systems for big data analytics: a technology tutorial. IEEE Access **2**, 652–687 (2014)
2. Devasena, C.L., Revath, R., Hemalatha, M.: Video surveillance systems - A survey. IJCSI Int. J. Comput. Sci. Issues **8**(4), 1 (2011)
3. Intel: Extract, Transform, and Load Big Data with Apache Hadoop. White Paper (2013)
4. Apache Flume. http://flume.apache.org/
5. Apache Sqoop. http://sqoop.apache.org/
6. Apache Kafka: A high-throughput distributed messaging system. http://kafka.apache.org/
7. Scribe. http://sourceforge.net/projects/scribeserver/
8. Apache Chukwa. https://chukwa.apache.org/
9. Pivotal Gemfire XD. http://www.pivotal.io/big-data/pivotal-gemfire-xd