SVE: Distributed Video Processing at Facebook Scale
IBD Report by Fabrice Ouedraogo

Master of Cloud Computing & Services CCS, *Université de Rennes 1*.

INTRODUCTION

Videos are viewed more than 8 billion times on an average day at facebook. At some point, the system built by facebook wasn't scalable. So they needed a new system meeting some requirements such as: to provide Low Latency, to be Flexible enough to support many applications, and to be robust to Faults and Overload.
This paper describes the evolution from monolithic encoding script (MES) prior system to the current Streaming Video Engine (SVE) that overcomes each of the challenges.

Video At Facebook

There are more than 15 different applications at Facebook that integrate video, which collectively ingest tens of millions of uploads and generate billions of video processing tasks. Facebook video posts have a complicated set of Directed Acyclic Graph (DAG) averaging 153 video processing tasks per upload, Messenger videos which have the simplest processing pipeline averaging just over 18 task per upload, Instagram stories generate over 22 tasks per upload and 360 videos generates thousands of tasks per upload due to high parallelism.

Reading through the paper, it becomes clear that the one metric that matters at Facebook when it comes to video is time to share. How long does it take from when a person uploads (starts uploading) a video, to when it is available for sharing?

This leads to three major requirements for the video processing pipeline at Facebook:Low latency, Flexibility to support a range of different applications (with custom processing pipelines), Handle system overload and faults

How Facebook used to process video

Prior to SVE, Facebook initial processing system was a Monolithic Encoding Script (MES). It was hard to maintain and monitor in the presence of multiple different and evolving applications requirements for encoding and transcoding. But most importantly, it used a batch-oriented sequential processing pipeline. A video is uploaded and stored (via a front-end server), then triggers processing by an encoder within MES, and when processing is complete the results are written to storage again and become available for sharing. As we can see, it doesn't sound too different to what we might put together with Amazon S3, Lambda, and Elastic Transcoder.

The major issue with MES is it has a poor time-to-share. Just uploading can take minutes (with 3-10MB videos, over 50% of uploads take more than 10 seconds, and with larger video sizes we are easily into the minutes and even 10 of minutes). The time required to durably store a video makes a meaningful contribution to the overall latency. Batch is the enemy of latency.

Why not just use an existing solution?

Before designing SVE, they examined existing parallel processing frameworks including batch processing systems like MapReduce, Dryad that assume the data to be processed already exists and is accessible. (not optimise for time-to-sharing), and Streaming processing systems like Storm, Spark Streaming, that overlap uploading and processing, but are designed for processing continuous queries instead of discrete events.(don't support the overload control policies and custom DAG per task model). So they had no choice left than build they own streaming video engine.

Introducing STREAMING VIDEO ENGINE (SVE)

The central idea in SVE is to process video data in a streaming fashion, in parallel, as videos move through the pipeline.
The net result is 2.3x-9.3x reduction in time-to-share compared to MES.

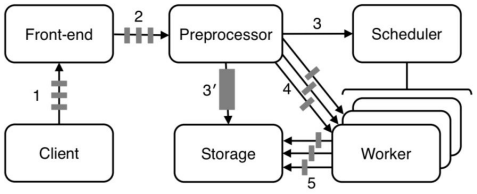
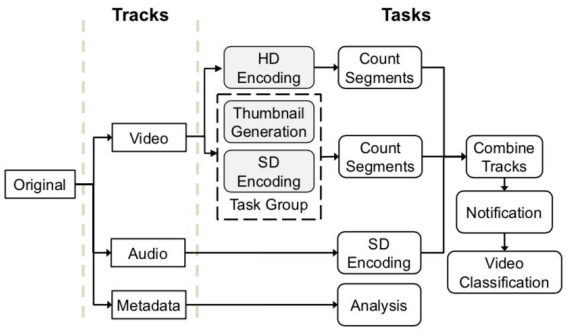


Figure 2: SVE architecture. Clients stream GOP split segments of the video that can be reencoded independently (1–2). The preprocessor does lightweight validation of video segments, further splits segments if needed, caches the segments in memory, and notifies the scheduler (3). In parallel, the preprocessor also stores the original upload to disk for fault tolerance (3′). The scheduler then schedules processing tasks on the workers that pull data from the preprocessor tier (4) and then store processed data to disk (5).

There are four fundamental changes in the SVE architecture compared to the MES. Firstly, the client breaks the video up into segments (group of pictures GOP) that are separately encoded, and thus reduce the latency by enabling processing to start earlier. Secondly, the front-end forwards video chunks straight to a preprocessor (rather than to storage as in the old system).Thirdly, the MES is replaced with the preprocessor, scheduler, and workers of SVE. The preprocessor submits encoding jobs to a distributed worker farm via a scheduler, in parallel with writing the original video to storage, while dynamically generating the DAG of processing tasks to be used for the video. Worker processes pull tasks from queues according to a simple high-priority and low-priority queue mechanism. When cluster utilisation is low worker pull from both queues, under heavier load they pull only from the high-priority queue. And finally, the SVE exposes pipeline construction in a DAG interface that allows application developers to quickly iterate on the pipeline logic.

Here's a simplified view of the Facebook app video processing DAG



The initial video is split into video, audio, and metadata tracks. The video and audio tracks are then duplicated n times, once for each encoding bitrate, and the encoding tasks operate in parallel over these. The output segments across tracks are then joined for storage.

Azure Media Services has a "**premium encoding**" option that also gives us the ability to define our own encoding workflows. AWS Elastic Transcoder supports '**transcoding pipelines**.' Now let's see how SVE manages fault tolerance and overload Control.

Fault tolerance and overload control

A worker detects task failure either through an exception or a non-zero exit value. For non-recoverable exceptions, the worker retries the tasks up to 2 times locally on the same worker, then up to 6 more times on another worker – leading to up to 21 execution attempts before finally giving up and the SVE terminates the DAG execution job by marking it as canceled. A large number of retries does increase end-to-end reliability

SVE provides robustness to overload through a progressing set of reactions to increasing load. There are three primary sources of overload: organic (that occurs when social events result in many people uploading videos at the same time at a rate much higher than the usual weekly peak eg:New Years Eve, ice bucket challenge), load-testing that occurs when Kraken(load-testing framework) tests the entire Facebook by running a disaster tolerance test that emulates a datacenter going offline, Bug-induced overload occurs when a memory leak in the preprocessor tier caused the memory utilization to max out..These reactions escalate from delaying non-latency-sensitive tasks, to rerouting tasks across datacenters, to time-shifting load by storing some uploaded videos on disk for later processing.

One question we might ask ourselves is "Can we use this video engine for the live streaming as well?"

Well it turns out that the livestream video application's requirements are a mismatch for what SVE provides.

The primary mismatch between SVE and livestreaming stems is that for the live streaming, each second only 1 second of video needs to be uploaded and only 1 second of video needs to be processed. A parallel processing is unnecessary because there is only a small segment of video to process at a given time.
Another mismatch is that the flexibility afforded through dynamic generation of a DAG for each video is unnecessary for live streaming. A third mismatch is that SVE recovers from failures and processes all segments of video, while once a segment is no longer “live” it is unnecessary to process it. Each of these mechanisms adds some latency, which is at odds with livestreaming primary requirement.
As a result, live streaming at Facebook is handled by a separate system.

RELATED WORK

This section reviews three categories of related work: video processing at scale, batch processing systems, and stream processing systems. SVE occupies a unique point in the intersection of these areas because it is a production system that specializes data ingestion, parallel processing, its programming abstraction, fault tolerance, and overload control for videos at massive scale.
Video Processing at Scale. ExCamera is a system for parallel video encoding that achieves very low latency through massively parallel processing of tiny segments of video that can still achieve high compression through state-passing. SVE is a much broader system than ExCamera.
There has been a tremendous boom in batch processing systems and related research. MapReduce is a parallel programming model which insights were to build the hard parts of distributed computation fault tolerance, moving data, scheduling into the framework so application programmers do not need to worry about them and to have the programmer explicitly specify parallelism through their map and reduce tasks. SVE, and many other distributed processing systems, exploit these same insights.
Spark is a batch processing system that uses resilient distributed datasets to keep data in memory and share it across computations, which results in much lower latency for iterative and interactive processing.

CONCLUSION

SVE provides low latency by harnessing parallelism in three ways that MES did not. Firstly, SVE overlaps the uploading and processing of videos. Secondly, it parallelizes the processing of videos by chunking them into smaller videos and processing each chunk separately in a large cluster of machines. Thirdly, SVE parallelizes the storing of uploaded videos, with replication for fault tolerance, with processing. Taken together, these improvements enable SVE to reduce the time between an upload complete and video share by 2.3X–9.3X over MES.