The Nephele Livescale Toolkit: Real-Time Video Stream Processing At Scale

Björn Lohrmann
Technische Universität Berlin
Email: bjoern.lohrmann@tu-berlin.de

Odej Kao
Technische Universität Berlin
Email: odej.kao@tu-berlin.de

***Abstract*—With the growing number of commodity devices capable of producing continuous video and audio data streams, the resulting overall data volumes raise challenges with respect to the infrastructure used to to process and store them. The future availability of services to automatically aggregate, analyze or relate streams from different sources is desirable. We outline components of a possible architecture for a large-scale and real-time video processing system. Furthermore, we introduce the Nephele Livescale Toolkit, which can be used to build real-time video/audio processing data flows for the scalable, data-parallel execution engine Nephele. We demonstrate the Livescale Toolkit using a data flow, that groups and ranks live video streams by geographic origin and video quality in real-time and redistributes them to subscribed clients.**

I. Introduction

As a growing number of commodity devices, such as mobile phones and tablets, is capable of producing continuous video and audio data streams, the sheer amount of these devices and the resulting overall data volumes raise challenges with respect to the infrastructure used to to process, store and potentially distribute them. The ongoing deployment of WebRTC-enabled (Web Real-Time Communication) browsers and smartphone apps can be expected to give an additional boost to this development. Over the past years, a lot of use cases for inexpensive recording devices that address a large audience of viewers have emerged. Websites like Livestream[1] or Ustream[2], offer their users means to produce and broadcast live media content to a large audience in a way that has been reserved to major television networks before. More and more academic institutions and conferences not only offer recorded lectures and talks but also broadcast them as livestreams. CCTV (Closed-Circuit Television) cameras in public places produce large amounts of video material, most of which is currently either recorded to facilitate post-incident investigations or needs to be sighted by humans, which is both an expensive and tedious task. Automated video analysis could be a great cost-saver here and is also ongoing area of research and development.

However, at the moment, the capabilities of available large-scale services are limited to media transcoding or simple picture overlays. In contrast to that, future services might also offer to automatically aggregate, analyze or relate streams from different sources. For example one could group video streams by keyword tags, geographic closeness or more sophisticated classification algorithms and find the best quality streams from that group. As these operations are computationally expensive and in most scenarios must be applied in real-time, we propose to use existing highly scalable stream processing engines. We will outline components of a possible architecture for a real-time video processing system, where video is captured on large numbers of IP-enabled devices, processed in parallel inside a shared-nothing cluster and distributed back to subscribed client devices. Furthermore, we introduce the Livescale Toolkit, which can be used to build real-time video/audio processing data flows for Stratosphere's[3] scalable, data-parallel execution engine Nephele and demonstrate its capabilities with a sample application.

II. Background

In recent years various engines for scalable stream data processing have emerged [1], [2], [3], many of which can be leveraged to process video and audio stream data in real-time. The engines, while different in many aspects, typically follow a master-worker pattern. The master node receives a processing job, splits it into sets of individual tasks, and schedules those tasks to run on the available worker nodes. These frameworks are intended to run on existing shared-nothing compute clusters (both virtualized and bare-metal) or integrate with Infrastructure-as-a-Services (IaaSs) clouds [3] to allocate virtual machines on demand.

These frameworks are intended to be general purpose and can process arbitrary types of data. They take care of three major aspects of the distributed computation. First, they provide a programming model, that allows programmers to continue writing sequential code as User-Defined Functions (UDFs) and not worry about parallelization. The programming model is usually a data flow graph, where vertices represent the UDFs and the edges denote communication channels between them. Some frameworks support arbitrary Directed Acyclic Graphs (DAGs) shaped data flow graphs [3], some allow graph structures containing loops [2], [1]. Second, these engines execute the data flow in a data-parallel fashion, organize the data transfers between tasks and balance the load between the worker nodes. The way both vertices and edges translate to system resources at runtime is similar among all of the mentioned systems. Each task vertex of the job typically translates to either a separate process or thread within a process. Third, some frameworks provide fault tolerance mechanisms, so that when a task fails due to hardware or software failures, the system can ideally recover the lost computation and data.

[1] http://www.livestream.com/
[2] http://www.ustream.tv/
[3] http://www.stratosphere.eu

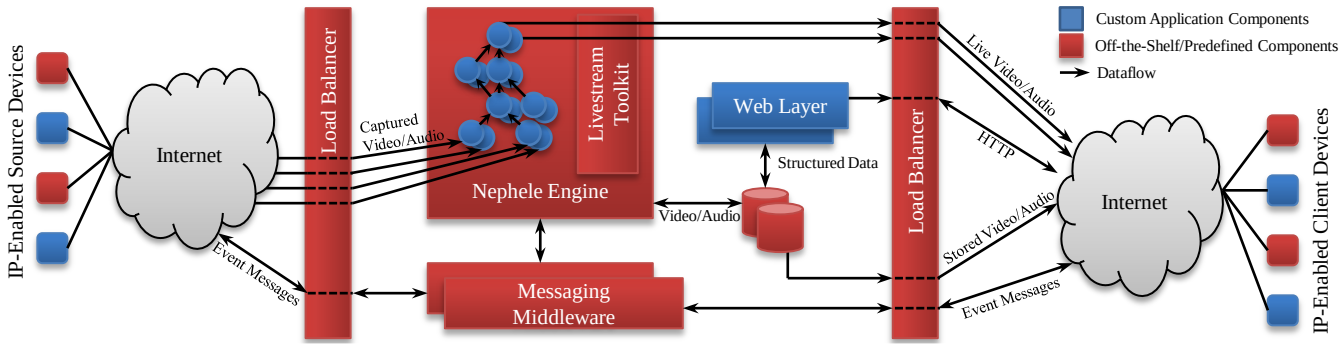

Fig. 1. Architecture of a large-scale real-time video analysis and broadcasting system

III. System Architecture

In this section we will outline the architecture of a large-scale real-time video processing system, shown in Figure 1. Video/audio data originates at IP-enabled source devices and is streamed through a transparent load-balancing layer into a stream processing engine. We assume usage of the Nephele [3] engine, that executes arbitrary DAGs and supports latency constraints [4], which can guarantee fast processing of incoming data. Other engines may however be equally suited to the task. The processed video/audio can then be shipped live to interested clients or stored in a scalable storage system. To signal additional or asynchronous events between the components of the architecture standard *messaging middleware* can optionally be employed, this is however outside the scope of the poster. The *load balancers* coordinate where stream sources send their data with the goal of evenly spreading the load. In their simplest form they can be realized with round-robin DNS for example. Another optional component is the *Web Layer*, that can offer additional application-dependent functionality, e.g. serve lists of available streams, manage user accounts etc. All components must however be able to scale to large numbers of clients.

IV. The Livescale Toolkit

To support the development of a system as outlined in the latter section, we have researched and developed the *Livescale Toolkit*, a set of reusable building blocks for real-time video and audio stream processing inside the Nephele engine. The source code and documentation are open source[4]. In the following we will briefly present the most important types of data flow building blocks.

a) Receivers and Broadcasters: A *receiver* is an arrival point for video and audio data streams as well as meta-data from outside the data flow and converts it into discrete records for shipping within the engine. It also assigns a unique identifier to each stream and attaches it to the records resulting from that stream. A broadcaster provides the opposite functionality and ships video/audio streams to connected or subscribed clients.

b) Decoder and Encoder: Decoders apply codecs to decompress video/audio streams into records with raw frames or audio samples. Analogous, encoders reverse this process. We have integrated the ffmpeg[5] library to support a wide variety of video and audio formats.

c) Frame Processors: A frame processor looks at the frames it receives and applies operations on the frame records. This can be a manipulation of the frame image, e.g. adding a graphic overlay, or analysis of the frame content, e.g. image quality assessment. All frame processors consume and emit frame records or meta-data records with analysis results. The toolkit contains a set of predefined frame processors, such as text and timer overlays and image quality assessors. Application-specific frame processors can be easily implemented.

d) Frame Hubs and Routers: Frame hubs receive frame records and emit a duplicate for each output, thus duplicating the videostream. This enables concurrent processing of the same video/audio stream by distinct branches of the Nephele data flow. A router receives frame and meta-data records of a stream and decides which parallel instance of the subsequent vertex the frame record shall be routed to. This allows for example to group streams by properties or prior analysis results.

V. Demonstration

We demonstrate a scalable application that groups and ranks live video streams by geographic origin and their quality so that clients can choose the "best quality" live broadcast by geographic location. We provide a setup of the major components with a number of simulated video sources and clients. The demonstration focuses on how the *Livescale Toolkit* enables the development of scalable, highly-parallel video processing applications within the Nephele execution engine. WLAN (or preferably wired LAN) access and Internet connectivity will be required. If unavailable, the demonstration can be also be scaled down to run locally on the presenter's laptop.

VI. Conclusion

We have outlined and demonstrated an architecture of a large-scale real-time video processing system where video/audio data originates from and can be delivered to IP-enabled source and client devices. Further, we have outlined the *Livescale Toolkit* that provides video/audio stream processing capabilities to the scalable execution engine Nephele.

[4] http://github.com/bjoernlohrmann/livescale-toolkit

[5] http://ffmpeg.org/

REFERENCES

[1] "nathanmarz/storm - GitHub," https://github.com/nathanmarz/storm, 2012.

[2] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *2010 IEEE International Conference on Data Mining Workshops*, ser. ICDMW '10. IEEE, 2010, pp. 170–177.

[3] D. Warneke and O. Kao, "Exploiting dynamic resource allocation for efficient parallel data processing in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 985–997, Jun. 2011.

[4] B. Lohrmann, D. Warneke, and O. Kao, "Nephele streaming: Stream processing under qos constraints at scale," *Cluster Computing*, 2013.