On the Fly Dynamic Graph Generation for Video Processing

Dominik Winecki
The Ohio State University

Introduction

Growth in Augmented Reality Devices and Internet connected cars is creating use-cases that conventional video processing systems are unable to handle. In many of these cases video is streamed to a remote server for processing[3].

Goals:
To facilitate the creation of distributed video processing pipelines.

- Real time video processing
- Arbitrarily high scaling
- Dynamically changing pipelines
- Transparent distribution over a server cluster
- Shared resources between pipelines

Current Approach:
In most video focused stream processing systems, like Scanner[1] and VideoStorm[5], tasks are represented with Directed Acyclic Graphs (DAGs). Since the graphs are static there is no way to dynamically change the workload. This can cause unnecessary work and limits on single video processing speed.

Proposed Solution:
Instead we take a user-defined fold function. This function is used to build the graph as new information becomes available.

Implementation

- Split between Input, Workers, and the Controller
- Connected through a Redis database
- Each task is performed by one or more workers
- Each controller defines a video's pipeline

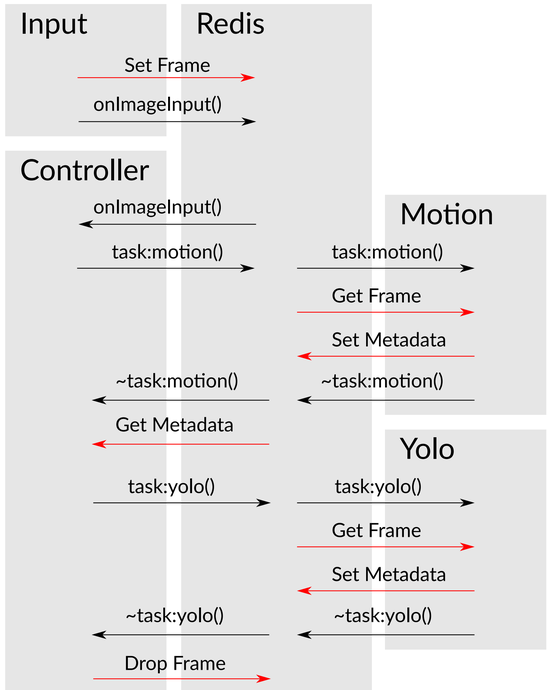

Figure 1: Per-frame data and control flow

Data Representation

- Frame data and metadata stored as binary string map
- Frames and tasks referenced by ID
- Task queues sorted by priority and user provided key

Controller Design

```
debt = 0.0

function onImageInput(frame)
    vt.taskQueue("motionDetect", frame, "", motionCallback, "")
end

function motionCallback(frame)
    local task = vt.taskGet(frame, "motionDetect")
    local frameChange = json.decode(task)["diff_ratio"]
    debt = debt + frameChange

    if debt > 0.10 then
     debt = 0.0
         vt.taskQueue("yolo", frame, "", yoloCallback, "")
    else
         vt.frameDelete(frame)
    end
end

function yoloCallback(frame)
    vt.frameDelete(frame)
    print("Finished with frame " .. frame)
end
```

Figure 2: Example control script

- Fold function defined as Lua script
- Context represented primarily as global variables
- Asynchronous programming model
- Per-video scalability limited only by controller script execution

Evaluation

Experiment 1: Object Detection

- 24,941 video frames from Berkeley DeepDrive[4] dataset
- OpenCV Motion Detection
- Darknet Yolo[2] Object Detection

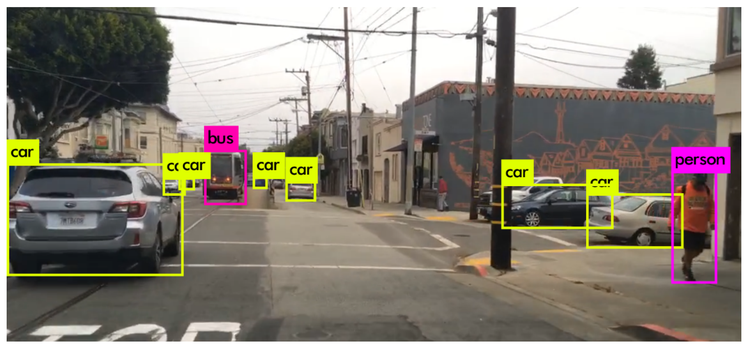

Figure 3: Annotated frame from evaluation

	Yolo	Motion	Full
CPU	8.9-9.3s	2ms	8.9-9.3s
GPU	70-74ms	2ms	85-100ms

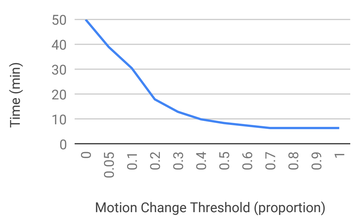

Figure 4: Motion detection experiment results

- Effectively reduced work through motion detection load shedding
- Single device performance consistent with 1, 2, 8, and 48 CPU cores

CPU: 2x Intel Xeon Gold 6126, **GPU:** Nvidia TITAN Xp

Experiment 2: Simulated Load at Scale

- Generated dataset of 100,000 frames
- Simulated neural network type workload
- Tests controller instead of frame store and tasks

Figure 5: Simulated Load at Scale Test

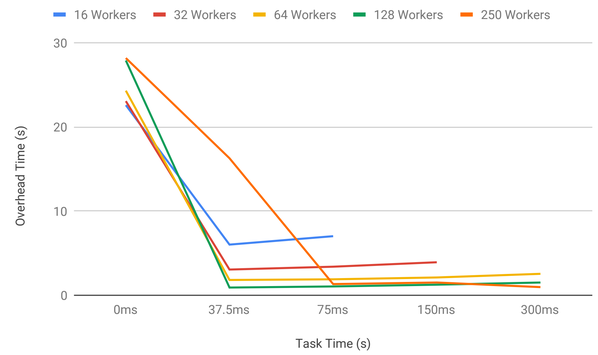

Figure 6: Pipeline delay overhead from theoretical compute times

- Upper end throughput of 4500-4900 fps with instant task execution
- 3000+ fps on normal workloads given sufficient hardware
- More efficient for longer tasks with many workers

HoloLens Video Streaming

- Demo built to overlay found objects
- Video stream consumed from device for processing
- <100ms processing time per frame
- Device video encoding overhead causes large majority of delay

Acknowledgements

This research was conducted with support from Faculty Mentor Arnab Nandi.

References

[1] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian.
Scanner: Efficient video analysis at scale.
CoRR, abs/1805.07339, 2018.

[2] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi.
You only look once: Unified, real-time object detection.
CoRR, abs/1506.02640, 2015.

[3] Cisco Systems.
Cisco Visual Networking Index: Forecast and Methodology, 2016-2021.
Technical report, 2017.

[4] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell.
BDD100K: A diverse driving video database with scalable annotation tooling.
CoRR, abs/1805.04687, 2018.

[5] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman.
Live video analytics at scale with approximation and delay-tolerance.
In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377--392, Boston, MA, 2017. USENIX Association.