College of Engineering, Design and Physical Sciences,
Electronic and Computer Engineering

Distributed Computing Systems Engineering

Interim Report

Conception and realization of a distributed and automated computer vision pipeline

Michael Watzko
1841795

Supervisor:
Dr. Paul Kyberd

Date of Submission: August 15, 2019

Contents

Chapter 1

Introduction

A research project that analyses traffic flow uses camera drones to capture aerial videos of vehicular traffic. The distribution and execution of the process to analyze and track those vehicles shall be automated. The development of the tracking and analysis process is not part of the thesis.

1.1 MEC-View?

1.2 The Tool

describe the tool, what it is for, what it does, current workflow

The current workflow consists of the following steps:
- define reference points in one single frame through a user input
- track stationary reference points on all other frames
- estimate the camera position for each frame
- detect vehicles in all frames
- track detections and assign them to trajectories
- perform lane detection
- record a result video
- export trajectories to a csv-file
- create charts

As listed above, at least one stage must be able to process user-input. The current progress must be observed and errors must be reported in an way, that
allows one to understand the circumstances for the cause of the error.

For easy and fast scalability, docker images shall be used to distribute the binaries onto the nodes.

1.3 Defining the Problem Space

what is required / what shall the implementation be capable of from the view of the "user"

user interaction

1.4 Analyzing the Problem Space

describe scenarios the implementations must be able to handle in order to archive the requirements?

resource tracking - global (read-only) input resources ("big" data files) - per stage evolving project files - might have some kind of version control? - dynamically detect within a stage whether user input is required - be able to continue / redo latest stage - error / warning detection / tracking! ([!a-zA-z]err[!a-zA-z])|([!a-zA-z]error[!a-zA-z]) - web technology

- retrieve required binaries - retrieve required resource files - archive output files and logs

- persistence stage/state tracking of projects/pipelines/states

Problems to solve

- stages might have individual hardware requirements

- multiple stages might require the same hardware at the same time

- stages can depend on the result of another stage

- for scalability, it shall be easy to add and remove hardware-nodes

- the video files are large (4k footage), sending decoded frames (25MB) through the network might be unreasonable

- the definition of a pipeline shall be easy to understand for good maintainability

- the hardware shall be used efficiently to achieve a high throughput

- docker images need to contain and provide all required libraries

- prevent stages from leaving other stages far behind?

- storage and distribution of intermediate results

- log collection

adding a new host - instantiate docker image and mount config and docker socket? - encrypt communication between control and worker? - possibility for decentralization - makes archiving logs and results hard

scenarios

define pipeline - define gpu stage - define cpu stage - define required input assets - define assets for each stage to be accessible in the next stage - stages depend on other stages - do it the other way around? set next stage? - next stage + "parameters" (assets to keep/transfer) - allows branching

upload resource file (video) - ... upload <path> <name-at-remote> - maybe to one common pool of resources? - free disk space?

start pipeline - select resources required by the pipeline - start

go through stages until finished - take care of cpu/gpu env requirements - if no common pool of resources: concurrently copy assets to target machines - archive

maybe: halt at stage because of error / required user interaction - allow continuation - allow download / upload of assets into this stage - free disk space?

easy installation and binary distribution - docker image per pipeline stage? - map management binary into docker -> exec - requires standalone binary - implicitly requires compatible libc env/unix system - requires administrative (docker) privileges

outputs of a stage are immutable after it has finished, stages using that data are working on a copy

nice to have: display progress captured from log (regex filter with multiple subjects/progresses per stage)

show time a stage is running

show estimated remaining time (based on captured progress)

todo list per project

project can run through multiple pipelines multiple times

nomad -> .deb archive?

deployment - web - controller - third party / nomad

start start from a certain stage pause after a stage redo a stage change variables at a stage

Chapter 2

State of the art

2.1 Existing software solutions

2.1.1 Hadoop MapReduce

focus transforming a big dataset by splitting it into many jobs, distributing it onto many workers, doing a transformation on each dataset, and merging it back together (only map -> reduce) Distributed filesystem

2.1.2 Quartz

http://www.quartz-scheduler.org/ http://www.quartz-scheduler.org/overview/

- + Java
- - requires integration
- - aimed towards running a job at a given time or in certain intervals

2.1.3 Pipeline examples: Jenkins / GitLab

Pipeline file with multiple stages a stage can be executed on a host focused on doing a job with different inputs again and again and again CI -> usually no user interaction

2.1.4 Camunda

https://camunda.com/

Rich Business Process Management tool, many types of tasks, steps, transitions, triggers and endpoints. Focused upon moving a dataset along the matching path of the process. Out of the box graphical user interface for process definition and interaction. Allows custom external worker through queues. Misses capability to control which task to process on which worker through fine grained filters and how to allocate and distribute resources(?). Requires custom plugins for more advanced user forms, not designed for that. Not designed provide an overview on the docker machines, cluster state nor logs, file up and download

2.1.5 Cubernetes

too heavy?

2.1.6 Luigi

Similar, but locked-down on python (+machine learning)? https://www.datarevenue.com/en/bl
to-scale-your-machine-learning-pipeline

2.1.7 Nomad

Deployment and management of containers rich REST API can handle resource requirements device plugins / GPU support

++ available through debian / ubuntu std-repositories

2.2 Docker

Chapter 3

Things to solve / decide upon

3.1 Programming language

3.1.1 Java

3.1.2 Rust

3.1.3 Scala?

3.1.4 Go?

3.2 Docker image packaging?

3.3 REST interface

3.4 WebInterface

3.5 CLI?

3.6 Authentication / Encryption / SSL

3.7 Data Model

3.8 Distributed File System

3.8.1 HDFS

3.8.2 dCache

used by 10 of 13 top research facilities https://www.dcache.org/manuals/dcache-whitepaper-light.pdf can replicate data-pools, access through NFS (and many more) is possible

used in grid computing facilities, integration with LDAP and Kerberos possible, supports tertiary storage, supports GssFtp, GsiFtp/GridFtp, HPSS, CERNs GridFileAccessLayer GFAL

complex installation many dependencies: postgresql, configuration of interdependent internal service: pool, poolmanager, glzma?, zookeeper

documentation is lückenhaft, outside of dcache.org only veraltet versions are found

too much overhead for just having a distributed file system

3.8.3 OpenIO

limited to distributed file system

provides docker image

simple CLI, focused on managing storage containers and replicas

Java SDK?

Supports NFS (for Linux workers), and Samba/SMB for Windows/Linux clients

NFS only through paid plan

3.8.4 seaweed

datacenter and rack aware in volume replication scenarios

easy to setup

single binary

mount through FUSE

[volumes] <-> [master] <-> [filer] <-> [clients] bzw filer mit master und volumes

master halten die zuweisung file -> addresse filer machen nur ein lookup und zuweisen oder sowas aber der client frägt filer an und der muss dann zu irgendeinem master die verbinundg aufbauen und nachschlagen dh wollte pro physicalischen server 1 volume, 1 master, 1 filer haben damit einfach dezentral kommen und gehen kann aber problem 1: anzahl der master soll immer ungerade sein problem 2: du kannst nicht einfach master on-the-fly hinzufügen und musst stattdessen teilweise die neu starten mitm parameter: hey da drüben ist noch ein master problem 3: es läuft nicht zuverlässig

3.9 Winslow

`https://de.wikipedia.org/wiki/Frederick_Winslow_Taylor` coordination withing a container - start nomad and join existing nomad instances - start weed and join existing weed masters

requires winslow to winslow communication

might need to restart services, must ensure that happens not everywhere at the same time

using distributed filesystem as configuration storage? - hard for initial start / problematic if down - but automatically distributes configurations + allows replications

Chapter 4

Implementation

log strategy?

how to handle changes in configuration on a restart - how to sync with nomad - how to handle still running jobs on an now invalid configuration? - keep copy of old configuration?

4.1 Orientation

bash -> variable substitution

4.2 No unexpected behavior

no null, instead use Optional

lists are never null nor Optional but empty or filled instead

see de.itd.tracking.winslow.config.*

called defensive programming? - good to be error-resilient - bad in performance critical scenarios

Appendix A

Extra data