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: September 21, 2019

Contents

Chapter 1

Introduction

Since the industrial revolution, humans strive for more automation in the industry as well as in the every day life. What was at first a cost saving measurement in factories, now also is a differentiation method for products. A new product must prove a higher level comfort to the customer than the previous generation as well as all the competitors. As such, the ambitions of the industry are focused on increasing the value of their products for the customer.

The automotive industry is one of the prime examples of this. Never was traveling from one place to another as comfortable as nowadays. Aspects like an elegant interior design, comfortable seats, air conditioning, entertainment systems and safety measurements need to be considered by car manufacturers to be competitive these days. The next step TODO: onward in this battle for the most luxurious driving experience is the autonomous driving vehicle. No longer shall the owner of a car steer it, but instead the car becomes his or hers personal chauffeur, driving the optimal route, the most comfortable way and being more reliable and safer than any human ever could.

The reason, autonomously driving cars are not common yet, is the complexity of it. Compared to already established technologies like parking assistants, entertainment systems or more efficient engine controllers, letting a computer reliably understand a certain traffic situation requires masses of input data and complex algorithms to process. As such, the problem itself becomes massive which cannot be solved that easily. The industry has no choice than to divide this into many small pieces and conquer solutions to it step by step (TODO: ref divide and conquer?).

The MEC-View research project explorers one such puzzle piece: whether and how to include external, steady mounted sensors in the decisions of partially autonomous vehicles for situations where onboard sensors are insufficient. As additional restriction, decisions made by autonomous vehicles are not allowed to disrupt the surrounding traffic flow otherwise phenomens like the TODO: Phantomstau could be caused by them. To understand and not disrupt traffic flow, one needs to study and thereby watch real traffic. Automatically analyzing traffic flows from video footage requires a lot of computation power and can be further optimized by specialized hardware such as GPUs.

This thesis will conceptualize and realize a distributed and automated computer vision pipeline which analyzes traffic flow within video footage.

1.1 MEC-View

The MECView research project - funded by the Federal Ministry for Economic Affairs and Energy - aims to supplement the field of view of automated driving cars with road-side sensor data using 5G mobile communication. The sensor information is merged into an environment model on the so-called Mobile Edge Computing (MEC) server. This server is directly attached to the radio station to ensure low latency environment model updates.

The project is tested at an intersection in Ulm, Germany. Currently, there are 15 lidar and video sensors installed. Those sensors send their detections to the (MEC) server. A fusion-algorithm merges those detections into one environment model and sends it back to the (MEC) server and to the automated cars.

Additionally, general traffic flow is analyzed to learn about movement patterns. To do so, 4k video data is captured by an air drone from real world cross roads. On each frame of such a recording, cars are detected with a neuronal network. Detected cars are tracked throughout the video to compute the movement speed and position in time of each car. In an analysis of all vehicles, hot-spots of high and low traffic flow can be determined.

Chapter 2

The Program

This chapter will discuss the program which shall be implemented. To do so, first, the problem to solve must be understood. Furthermore, existing solutions have to be considered for being used or being used as middle-ware or as being positive or negative examples.

2.1 (Ideal) Workflow

To understand the problem that shall be solved, the TODO: current workflow is considered to be able to imagine and ideal workflow. The program shall then enable the ideal workflow.

TODO: write more like the user first does abc then wants to do def...?

A tracking project starts with a video as input. The video is usually taken in 4k (3840 x 2160 pixels), a TODO: 8 bit RGB color depth and with 24 frames per second. A raw frame is therefore least 189 MiB large. The camera drone is using H.264 to compress the video to about TODO: X GiB per minute. The drone can fly for 25 Minutes straight. A single video can therefore be as large as TODO: X GiB. The system must be able to reliable store and access such an video. The user wants to upload such an video to the system in a common and easy way.

A user then wants to select a pre-defined pipeline to be executed for the uploaded video. In this context, a pipeline shall describe the general order of compute steps, called stages while the combination of a pipeline and a video shall be called project. Each stage applies an operation onto the whole video, such as detecting cars, tracking stationary reference points, estimating the camera position and assigning trajectories onto detections. The logic of such an stage is provided and not within the scope of the program to realize, but needs to be invoked and monitored instead. After selecting a pipeline, the user might want to set additionally environment variables or upload prepared files into the workspace.

The user wants to be informed about the current progress of all projects, a certain pipeline or stage. Therefore a current state is of interest, such as running, failed or succeeded for the pipeline and each stage.

Any stage might require additional user input, such as manually confirming that tracked references points are indeed reasonable. To do so, each stage must be able to pause the pipeline and the user must be able to access the intermediate results of a stage. This means, the user must able to browse, download and upload files. A stage might also have additional requirements or preferences for certain hardware, such as a GPU for the neuronal network. Which stage is executed on which execution node, is assigned by the system and not of special interest to the user. TODO: wait for node to be free?

The system also needs to provide the current console output of a stage, so that the user can pause a pipeline manually if needed. In contrasted to pausing, the user must also be able to resume a pipeline.

The user might find the results of a stage not meeting the expectations. In such a scenario, the users wants to redo the stage with adjusted input parameters such as environment variables, or a differently prepared workspace.

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

TODO: docker

Chapter 3

State of the art

In this chapter, programs solving similar problems - as described before - are looked further into. Some might manifest as middle-ware, some might teach about proven strategies and some might show pitfalls one should avoid.

3.0.1 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

For big data transformation, Hadoop MaReduce is well known TODO: prove. With MapReduce, the input data is split into blocks of data and distributed onto Mappers. Mappers then apply the business logic and output intermediate results in form of key/value pairs. After shuffling, the Reduce stage will combine values from key/value pairs with the same key. In the final output, each key is unique.

This strategy very powerful to process large input data. Mappers and Reducers can work independently on their data-sets and therefore scale very well when adding further instances TODO: prove.

Using MapReduce in the computer vision pipeline could be implemented like the following:

- Each video is split into many frames and each frame is applied to a mapper
- A mapper tries to detect all vehicles on a frame and outputs the position of it

- The Reduce then tries to link the detections of a vehicle for each frame
- The final result would be a set of detections and therefore positions of each vehicle in the video

But at the moment, this approach is infeasible due to at least two facts:

1. It is not always trivial to reasonable link the detections of a vehicle. For example, a vehicle can be hidden behind a tree for a few frames until visible again. In addition, MapReduce requires the combination to be performed per common key. Until one is trying to link the detections of multiple frames, the is no common identifier that could easily be used as key. The position of a moving vehicle cannot be used as key, neither can the color or size, because of the noise of the camera, deviation in detection output and perspective distortions.
2. MapReduce is a great in combining many machines to solve a big computational problem. But at the moment, this is neither a desired nor given condition. At the moment, there a TODO: three very powerful workstations with special hardware (GPUs). Therefore it is perfectly acceptable, when each workstation works through a complete video at a given time instead.

3.0.2 Build Pipelines

Build pipelines such as GitLab[8] and Jenkins[14] provide multi stage work distribution on multiple machines. In a common use-case, this is used to build binaries out of source code, after a new commit into the TODO: SCM - Source Code Management repository was made. At IT-Designers GmbH both are being used for scenarios like this. A pipeline definition in GitLab CI/CD [7] or in a Jenkinsfile [15]

3.1 Existing software solutions

IBM InfoSpheere [1]

GitLab [7]

Jenkins [15]

Quartz?? [22]

CSCS High Throughput Scheduler?? [2]

qsub job submission

3.1.1 Quartz

[23] [24]

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

3.1.2 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

3.1.3 Camunda

[11] [9] [10]

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

3.1.4 Cubernetes

too heavy?

3.1.5 Luigi

Similar, but locked-down on python (+machine learning)? [20]

3.1.6 Celery

[21]

3.1.7 Nomad

TODO: that dude that did the webui

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

vs kubernetes [13]

++ available through debian / ubuntu std-repositories

3.2 Docker

Chapter 4

Topics derived from the workflow

In this chapter, topics related to the desired workflow are analyzed and potential solutions discussed. Because of the work in progress nature of the thesis in this early state, the list of potential solutions might be incomplete. Also, the solution that is favored mostly at this stage, might not reflect the solution of the final implementation.

4.1 Storage

One of the central concerns is the storage management. The program needs to make input files available on each execution node an collect the results once the computation is complete. There are a few main architectural strategies to approach this. Simplified, either at a centralized location which is accessed by all execution nodes, a copy of the input files all execution nodes or distributed between all execution nodes with a set redundancy. The advantages and disadvantages can depend on the specific implementation and is therefore discussed in combination of such.

4.1.1 Hadoop File System

[4] [5] TODO: redudancy for evenly distributed

4.1.2 NFS

local/per node cache?

4.2 Coordination

4.3 Binary distribution

4.4 User Interface

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.

4.5 Defining the Problem Space

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

user interaction

4.6 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 5

Things to solve / decide upon

5.1 Programming language

5.1.1 Java

5.1.2 Rust

5.1.3 Scala?

5.1.4 Go?

5.2 Docker image packaging?

5.3 REST interface

5.4 WebInterface

5.5 CLI?

5.6 Authentication / Encryption / SSL

5.7 Data Model

5.8 Distributed File System

5.8.1 HDFS

secondary nameserver only through common centralized filesystem federation does not provide unified root

5.8.2 dCache

used by 10 of 13 top research facilities [16] [6]

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

zookeeper admin poolmanager spacemanager pnfsmanager cleaner gplazma pinmanager billing httpd topo info nfs pool

5.8.3 zsync

[18]

5.8.4 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

5.8.5 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

5.8.6 Alluxio

requires centralized filesystem for masters

[17]

5.8.7 GlusterFS

[19]

Bought by IBM very minimalistic included in ubuntu and debian repositories setup easy, without a lot of configuration (none to be precise) node information is spread on all nodes, no master/slave but replication requires that a multiple of it are assigned to the volume - can be circumvented by adding peers to a volume only every second peer (if replica is 2) geo-replication is interesting

5.9 Winslow

[3]

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 6

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?

6.1 Orientation

bash -> variable substitution

6.2 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

6.3 EventSystem

6.3.1 Via already distributed filesystem

https://docs.oracle.com/javase/tutorial/essential/io/move.html events/ directory with files being named after a integer, being the unique next event id write event to tmp/, then atomically move to events/ without replacing existing ids (which would indicate an collision)

6.4 Failure recovery

what if an instance suddenly crahes/disconnects/fails

heartbeats to detect, with max number of allowed skips (so no "timeout")

assign pipeline to a node? (supervising node) on failure, try to recover what has already been processed by re-assigning pipeline and then trying to load prev-state

6.4.1 paradigm: let it crash

Bibliography

[1] I. K. Center. *IBM InfoSphere. DataStage.* URL: https://www.ibm.com/support/knowledgecenter/en/SSZJPZ_9.1.0/com.ibm.swg.im.iis.ds.design.doc/topics/c_ddesref_Server_Job_Stages_.html (visited on 09/19/2019).

[2] S. N. S. Centre. *High Throughput Scheduler.* URL: https://user.cscs.ch/tools/high_throughput/ (visited on 09/19/2019).

[3] W. contributors. *Frederick Winslow Taylor. Wikipedia, The Free Encyclopedia.* URL: https://en.wikipedia.org/w/index.php?title=Frederick_Winslow_Taylor&oldid=913471357 (visited on 09/19/2019).

[4] T. A. S. Foundation. *Apache Hadoop.* URL: https://hadoop.apache.org/ (visited on 09/19/2019).

[5] T. A. S. Foundation. *Apache Hadoop. Documentation.* URL: https://hadoop.apache.org/docs/current/ (visited on 09/19/2019).

[6] P. Fuhrmann. *dCache, the Overview.* URL: https://www.dcache.org/manuals/dcache-whitepaper-light.pdf (visited on 09/19/2019).

[7] I. GitLab. *GitLab CI/CD. Pipeline Configuration Reference.* URL: https://docs.gitlab.com/ee/ci/yaml/ (visited on 09/19/2019).

[8] I. GitLab. *The first single application for the entire DevOps lifecycle.* URL: https://about.gitlab.com/ (visited on 09/21/2019).

[9] C. S. GmbH. *Process Engine API.* URL: https://docs.camunda.org/manual/7.6/user-guide/process-engine/process-engine-api (visited on 09/19/2019).

[10] C. S. GmbH. *Rest Api Reference.* URL: `https://docs.camunda.org/manual/7.8/reference/rest` (visited on 09/19/2019).

[11] C. S. GmbH. *Workflow and Decision Automation Platform.* URL: `https://camunda.com/` (visited on 09/19/2019).

[12] HashiCorp. *Nomad.* URL: `https://www.nomadproject.io/` (visited on 09/19/2019).

[13] HashiCorp. *Nomad. Nomad vs. Kubernetes.* URL: `https://www.nomadproject.io/intro/vs/kubernetes.html` (visited on 09/19/2019).

[14] jenkins.io. *Jenkins. Build great things at any scale.* URL: `https://jenkins.io/` (visited on 09/19/2019).

[15] jenkins.io. *Using a Jenkinsfile.* URL: `https://jenkins.io/doc/book/pipeline/jenkinsfile/` (visited on 09/19/2019).

[16] T. M. Patrick Fuhrmann. *dCache. Scope of the project.* URL: `https://www.dcache.org` (visited on 09/19/2019).

[17] C. Phipps. *Alluxio. Data Orchestration for the Cloud.* URL: `https://www.alluxio.io` (visited on 09/19/2019).

[18] C. Phipps. *zsync. Overview.* URL: `http://zsync.moria.org.uk/` (visited on 09/19/2019).

[19] I. Red Hat. *Gluster. Free and open source scalable network filesystem.* URL: `https://www.gluster.org` (visited on 09/19/2019).

[20] D. Revenue. *Distributed Python Machine Learning Pipelines.* URL: `https://www.datarevenue.com/en/blog/how-to-scale-your-machine-learning-pipeline` (visited on 09/19/2019).

[21] A. Solem. *Celery: Distributed Task Queue.* URL: `http://www.celeryproject.org` (visited on 09/19/2019).

[22] I. Terracotta. *Quartz Quick Start Guide.* URL: `http://www.quartz-scheduler.org/documentation/quartz-2.3.0/quick-start.html` (visited on 09/19/2019).

[23] I. Terracotta. *QuartzJob Scheduler.* URL: `http://www.quartz-scheduler.org/` (visited on 09/19/2019).

[24] I. Terracotta. *QuartzJob Scheduler. Overview.* URL: `http://www.quartz-scheduler.org/overview/` (visited on 09/19/2019).