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 22, 2019

# Contents


**1 Introduction** **1**

1.1 MEC-View . . . 2

**2 The Program** **3**

2.1 Current Workflow . . . 3

2.2 Desired Workflow . . . 4

2.3 Requirements . . . 5

**3 State of the art** **6**

3.1 Similar solutions . . . 6

3.1.1 Hadoop MapReduce . . . 6

3.1.2 Build Pipelines . . . 8

3.1.3 Camunda . . . 8

3.1.4 Nomad . . . 9

3.1.5 Further mentions . . . 9

3.2 Docker Integration . . . 10

**4 Topics derived from the workflow** **11**

4.1 Storage . . . 11

4.1.1 Hadoop File System . . . 11

4.1.2 NFS . . . 11

4.2 Coordination . . . 12

4.3 Binary distribution . . . 12

4.4 User Interface . . . 12

4.5 Defining the Problem Space . . . 12

4.6 Analyzing the Problem Space . . . 13

**5 Things to solve / decide upon** **15**
5.1 Programming language . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.1.1 Java . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.1.2 Rust . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.1.3 Scala? . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.1.4 Go? . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.2 Docker image packaging? . . . . . . . . . . . . . . . . . . . . . . . 16
5.3 REST interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.4 WebInterface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.5 CLI? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.6 Authentication / Encryption / SSL . . . . . . . . . . . . . . . . . . 16
5.7 Data Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.8 Distributed File System . . . . . . . . . . . . . . . . . . . . . . . . 16
5.8.1 HDFS . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.8.2 dCache . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
5.8.3 zsync . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
5.8.4 OpenIO . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
5.8.5 seaweed . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
5.8.6 Alluxio . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
5.8.7 GlusterFS . . . . . . . . . . . . . . . . . . . . . . . . . . 18
5.9 Winslow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

**6 Implementation** **20**
6.1 Orientation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
6.2 No unexpected behavior . . . . . . . . . . . . . . . . . . . . . . . . 20
6.3 EventSystem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
6.3.1 Via already distributed filesystem . . . . . . . . . . . . . . 20
6.4 Failure recovery . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
6.4.1 paradigm: let it crash . . . . . . . . . . . . . . . . . . . . 21

**Bibliography** **24**

# 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, the problem to solve must be understood. To gather requirements and understand the technical hurdles to overcome, this chapter is split into two sections. First, a rough glance over the current workflow is given, which is followed by a more detailed description for the desired workflow.

TODO: it is called TrackerApplication

## 2.1 Current Workflow

Currently, to analyze a video for the trajectories of the recorded vehicles, the following steps are executed manually:

1. Upload the input video to a new directory on the GPU server
2. Execute a shell script with the video as input file and let it run (hours to days) until completed. The shell script invokes a Java Program with parameters on what to do with the input file and additional parameters.
3. The intermediate result with raw detection results is downloaded to the local machine and opened for inspection. If the detection error is too high, the camera tracking has a drift or other disruptions are visible, redo the previous step with adjusted parameters.
4. Upload the video and intermediate result to a generic computing server and run data cleanup and analysis. This is again done with the same Java

Program as in step 2, but with different stage environment parameters.

5. Download the results, recheck for consistency or obvious abnormalities. Depending on the result, redo step 2 or 4 with adjusted parameters again.

6. Depending on the assignment, steps 4 and 5 are repeated to incrementally accumulate all output data (such as statistics, diagrams and so on).

Because all those steps are done manually, the user needs to check for errors by oneself. Also, if a execution is finished or failed early, there could be hours wasted if the regular check interval are too far apart.

## 2.2 Desired Workflow

The desired workflow shall be supported through a rich user interface. This user interface shall provide an overview of all active projects and their current state, such as running computation, awaiting user input, failed or succeeded.

To create a new project, a predefined pipeline definition shall be selected as well as a name chosen. Because only a handful of different pipeline definitions are expected, the creation of such does not need to happen through the user interface. Instead, it is acceptable to have to manually edit a configuration file in such rare circumstances.

Once a project is created, the user wants to select the path to the input video. This file has to be uploaded to a global resource pool at this point. The upload and download of files shall therefore also be possible through the user interface. Because a video is usually recorded in 4k (3840 x 2160 pixels), encoded with H.264 and up to 20 minutes long, the upload must be capable of handling files which are tens of gigabytes large.

Once a pipeline is started, it shall execute the stages on the most fitting server node until finished, failed or a user input is required. Throughout, the logs of the current and previous stage shall be accessible as well as uploading or downloading files from the current or previous stages workspace. In addition to the pipeline pausing itself for user input, the user shall be able to request the pipeline to pause after the current stage at any moment. When resuming the pipeline, the user wants to overwrite the starting point to, for example, redo the latest stage.

Mechanisms for fault tolerance shall detect unexpected program errors or failures of server nodes. Server nodes shall be easily installed and added to the existing network of server nodes. Each server node might provide additional hardware (such as GPUs), which shall be detected and provided.

For the ease of installation and binary distribution, Docker Images shall be used for running the Java Program for analyzing the videos as well the to be implemented management software.

TODO: describe: project/pipeline -> stage?

## 2.3 Requirements

From the desired workflow, the following requirements can be extracted (shortened and incomplete due to early project stage):

- Rich user interface
- Storage management for global resource files as well as stage based workspaces
- Pipeline definition through configuration files
- Handling of multiple projects with independent progress and environment
- Reflecting the correct project state (running, failed, succeeded, paused)
- Log accumulation and archiving
- Accepting user input to update environment variables, resuming and pausing projects as well as uploading and downloading files into or from the global resource pool or a stages workspace.
- Assigning to be started stages to the most fitting server node
- Detecting program errors (in a stage execution)
- Cope with server node failures
- Docker Image creation for the Java Binary as well as the program implementation, preferred in an automated fashion.

# Chapter 3

# State of the art

In this chapter, programs solving similar problems, as described in the desired workflow, or dealing with a subset of the problem are looked into. The reason for this is to use well established or suitable programs as middle-ware to reduce implementation overhead. Where this is not possible, one might be able to gather ideas and learn about proven strategies to use or pitfalls to avoid while implementing custom solutions.

## 3.1 Similar solutions

This sections focuses on programs trying to provide somewhat similar workflows.

### 3.1.1 Hadoop MapReduce

For big data transformation, Hadoop MapReduce[6] is well known. 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 accompanied with a value.

This strategy has proven to be very powerful to process large amount input data because Mappers and Reducers can work independently on their data-sets and therefore scale very well when adding further instances.

If the implementation were to be based on Hadoop MapReduce to achieve the desired workflow, it could be done 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 their position, orientation, size and so on
- The Reducer then tries to link the detections of a vehicle through multiple frames
- The final result would be a set of detections and therefore all positions for each vehicle in the video

But at the moment, this approach seems to be unfitting due to at least the following reasons:

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, there 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. The current implementation of the TrackerApplication is archiving this by finding similarities between detections, but for the Mapper this would required to be expressed as a deterministic key.

2. MapReduce is 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 handful of very powerful workstations with specialized hardware. Therefore it is perfectly acceptable and sometimes required, when each workstation works through a complete video at a given time instead.

### 3.1.2 Build Pipelines

Build pipelines such as GitLab[10] and Jenkins[17] can also distribute the execution of a stage onto another server node. In a common use-case, such build pipelines are used to build binaries out of source code, after a new commit into a SCM[1] repository was made. At IT-Designers GmbH GitLab as well as Jenkins are commonly used for scenarios exactly like this. A pipeline definition in GitLab CI/CD [9] or in a Jenkinsfile [18] describe stages and commands to execute. Each stage can be hosted on another node and be executed sequential or in parallel to each other.

Although this seems to be quite fitting for the desired workflow, there are two issues. First of all, such a pipeline does not involve any user input besides an optional manual start command. The result is then determined based on the state of the input repository. Second, such an pipeline is designed to determine the output (usually by compiling) whereas each run ins independent from the previous and a repeated run shall provide the same result as the previous. Usually, a new run is only caused by an change of the input data. However, the desired workflow differs in this aspects. A redo of a stage can depend on the result of the previous stage, for example if the results are poor or the the stage failed. Instead of having multiple complete pipeline runs per project, the desired workflow uses a pipeline definition as base for which the order can be influenced. Also, intermediate results need to influence further stages, even if repeated.

### 3.1.3 Camunda

Camunda[13] calls itself a “Rich Business Process Management tool” and allows the user to easily create new pipelines by combining existing tasks with many triggers and custom transitions. Camunda is focused upon visualizing the flow and tracking the data through a pipeline. The Camundas Process Engine[11] also allows user intervention between tasks.

One of the main supporting reason for it Camunda is the out of the box rich graphical user interface for process definition and interaction. Through its API[12], Camunda also allows custom external workers to execute a task. But it

[1] Source Code Management

misses the capability to control which task shall be processed on which worker node which is required by the desired workflow. It does also not provide any concept on how to allocate and distribute resources. The user interface - while being rich overall - is quite rudimentary when it is about configuring tasks and would therefore require custom plugins to be developed for more advanced user interactions.

Camunda is also not designed to reorder stages or insert user interactions at seemingly random fashion. Also, the user itself is considered more as a worker, that gets some request, “executes” this externally and finally marks the request as accepted or declined. Mapping this to the desired workflow does not feel intuitive. Finally, There is also no overview of task executors, no centralized log accumulation and no file up or download for project global resources.

### 3.1.4 Nomad

Nomad[14] by HashiCorp is a tool to deploy, manage and monitor containers, whereas each job is executed in its own container. It provides a rich REST API and can consider hardware constraints on job submissions. Compared to Kubernetes[15], which is similar but more focused on scaling containers to an externally applied load, it is very lightweight. It is also available in many Linux software repositories - such as for Debian - which makes the installation very easy.

Because there were no grave disadvantages found (depending on a third party library can be a disadvantage in flexibility, error-pronous and limit functionality) Nomad is being considered as a middle-ware to manage and deploy stages. Others[20] seem to be using Nomad to manage and deploy containers for similar reasons. Nonetheless, further testing and prototyping will be required for a final decision.

### 3.1.5 Further mentions

The following list shall acknowledge programs that behave similar to the previously three mentioned strategies. Programs there are listed here, were looked into, but not in-depth because while looking into them, miss-fits were detected early on. Listed in no specific order.

- **Quartz**[26] is a Java based program to schedule jobs. Instead of doing so by using input, Quartz executes programs through a timetable and in certain intervals.
- **Luigi**[24] also executes pipelines with stages and is written in python. The advertised advantage is to define the pipeline directly in python code. But, this is at the same time the only way to define pipelines which contradictions with the existing Java TrackerApplication implementation.
- **Calery**[25] is focused on task execution through message passing and is written in Python. Intermediate results are expected to be transmitted through messages. Because is no storage strategy and python adapter-code would have been required, Calery was dismissed.
- **IBM InfoSphere**[1] provides similar to Camunda a rich graphical user interface but for data transfer. Dismissed due to commercial nature.
- **qsub**[4][16] CLI[2] using in HPC to submit jobs onto a cluster or grid. Dismissed due to an expected high setup overhead, non-required multi-user nature and the fact, that it only provides a solution for the job submission.
- **CSCS**[2] High Throughput Scheduler (GREASY). Dismissed for similar reasons as qsub, although it is more light weighty and hardware agnostic (it can consider CUDA/GPU requirements).

## 3.2 Docker Integration

As describe before (see section 2.2), for easy deployment, the implementation of the management shall be executed from a Docker image [5], as well as the stages. This allows easier isolation of the stages and workspaces from each other and other host programs. Because one needs to communicate with the Docker daemon, this increases the complexity for the implementation. But thanks to third party libraries, the increase in complexity can be limited.

[2]Command Line Interface

# 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

[6] [7] 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 [19] [8]

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

[22]

### 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

[21]

### 5.8.7 GlusterFS

[23]

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] W. contributors. *Qsub. Wikipedia, The Free Encyclopedia.* URL: https://en.wikipedia.org/w/index.php?title=Qsub&oldid=745279355 (visited on 09/22/2019).

[5] I. Docker. *Enterprise Container Platform.* URL: https://www.docker.com/ (visited on 09/22/2019).

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

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

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

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

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

[11] 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).

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

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

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

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

[16] T. U. O. Iowa. *Basic Job Submission. HPC Documentation Home / Cluster Systems Documentation.* URL: https://wiki.uiowa.edu/display/hpcdocs/Basic+Job+Submission (visited on 09/22/2019).

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

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

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

[20] B. P. Peroutka. *Web interface for the deployment andmonitoring of Nomad jobs. Master's thesis.* URL: https://dspace.cvut.cz/bitstream/handle/10467/80106/F8-DP-2019-Peroutka-Pavel-thesis.pdf (visited on 09/22/2019).

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

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

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

[24] 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).

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

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