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 29, 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 luxury enhancement will be the autonomously 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 already, is their big complexity increase. 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 and cannot be solved that easily. To solve this, the industry has no choice than to divide this into many small pieces and work out solutions to it step by step.

The MEC-View research project explorers one such step: whether and how to include external, steady mounted sensors in the decision finding process for partially autonomous vehicles in situations where onboard sensors are insufficient. To not disrupt traffic flow with non-human behavior, one needs to study and thereby watch human traffic. Automatically analyzing traffic 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 can be used to analyzes traffic flow within video footage.

1.1 MEC-View

The MECView research project[1] - 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.

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 - called TrackerApplication - 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.

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[2] 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[3] and Jenkins[4] 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 [5] or in a Jenkinsfile [6] 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[7] 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[8] 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[9], Camunda also allows custom external workers to execute a task. But it misses the

[1]Source Code Management

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[10] 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[11], 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[12] 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 dCache

"The goal of this project is to provide a system for storing and retrieving huge amounts of data, distributed among a large number of heterogenous server nodes, under a single virtual filesystem tree with a variety of standard access methods"[13]. dCache seems to be able to solve the storage access and distribution for the stages and sever nodes. When using dCache, once could store the global re-

sources distributed on the server nodes. Built-in replication would prevent access loss on a node or network failure and export through NFS[2] allows easy access on Linux based systems[14].

But the installation is complex and requires many services to be setup correctly, such as postgresql and many internal services such as zookeeper, admin, poolmanager, spacemanager, pnfsmanager, cleaner, gplazma, pinmanager, topo, info and nfs. The documentation is also rather outdated and incomplete which meant, early tests with a prototype setup took days to setup and behaved rather unstable (probably due to a wrong configuration). It is to be seen, whether such an complex and heavy system is actually required or if there are feasible alternatives.

3.1.6 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**[15] 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**[16] 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**[17] 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**[18] provides similar to Camunda a rich graphical user interface but for data transfer. Dismissed due to commercial nature.

[2]Network File System

- **qsub**[19][20] CLI[3] 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**[21] 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).

- **zsync**[22], similar to rsync, is a file transfer program. Zsync allows to only transfer new parts when a file that shall be copied already exists in an older version on the target. This tool might be useful for a complete custom resource distribution strategy.

- **OpenIO**[23] provides a distributed file system, is already provided as Docker image and provides a simple to use CLI. Because the NFS export is only available through a paid subscription plan, it was dismissed from further investigation.

- **SeaweedFS**[24] provides a scalable and distributed file system. The most interesting aspects are that it is rack-aware as well as naively supports external storage such as Amazon S3. When adding server nodes from the cloud this could allow all nodes to access the same file system while using rack-aware replication to reduce used bandwidth and latency. A local test also proved that it is easy to setup, but because it cannot hot-swap nodes and was not able to recover when the seeweed master node became unreachable.

- **Alluxio**[25] provides a distributed file system but was dismissed because it itself requires a centralized file system for the master and its fallback instances.

- **GlusterFS**[26] is another tool to provide a distributed file system with replication. It was bought by IBM but is nonetheless available through the software repository of many Linux distributions such as Debian. A local test show that the setup is very easy without any configuration files

[3]Command Line Interface

required to be setup. The replication mechanism however, requires that an even multiple of nodes of the replica value are assigned to the file system. This makes GlusterFS hard to use in a scenario, where adding and removing nodes are expected to happen frequently, and therefore it was dismissed.

3.2 Docker Integration

As describe before (see section 2.2), for easy deployment, the implementation as well as the stages shall be executed inside Docker[27] containers. 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.

TODO: explain docker here already or only in the thesis?

Chapter 4

Outcome and further work

In this chapter the main concerns are listed. For each concern the current progress is described as well as further work that needs to be done.

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 (see chapter 3).

Further testing is required to decide whether a more complex storage system is required, or the simplicity of a centralized solution outweighs the setup and maintenance overhead.

4.2 Coordination

Another important concern is the coordination of the nodes. A central coordinator with external server nodes, such as GitLab and Jenkins have, might not be sufficient for more complex and longer lasting pipelines. The probability that

the master would need to be offline while there is a stage executed is in the scenario of this implementation a lot higher than for GitLab or Jenkins, because the stage is being execution for hours or days. Coupling stage execution plans on node availability in beforehand, as well as recovering from a sudden master failure implies additional implementation complexity. Completely decentralizing the coordination does this as well and even more, but also allows the usage of the system while the master failed or is unreachable due to maintenance. With further prototyping and research a reasonable solution shall be found.

4.3 Binary distribution

In a time where containers are common and have proven to be usable, the installation of the binaries directly on the operating system they are executed shall be avoided. There shall be no manual, nor automatic but custom file copies of the binaries or images from one server to the other. Experience shows, that without a proper management, this can easily become a mess, in which it is no longer clear, which file / image belongs to which version. At the same time, making all binaries publicly available through the Docker Hub[28] is no option either. Whether a self hosted Docker Registry[29] could be the solution to this will be determined in further testing.

4.4 User Interface

Providing a useful user interface might not be important to the functionality of the system itself but for the user experience. A bad user experience will cause a system not to be used. It became common practice for a rich user experience to be web based and supported by JavaScript. For a potentially decentralized system, it is also advantageous to be able to access a disconnected node in the same manner as the remaining system, which further encourages a web based solution. Web based solutions such as React and Angular shall therefore be investigated for being used as user interface.

Chapter 5

Schedule

The following figure shows schedule for further and past work:

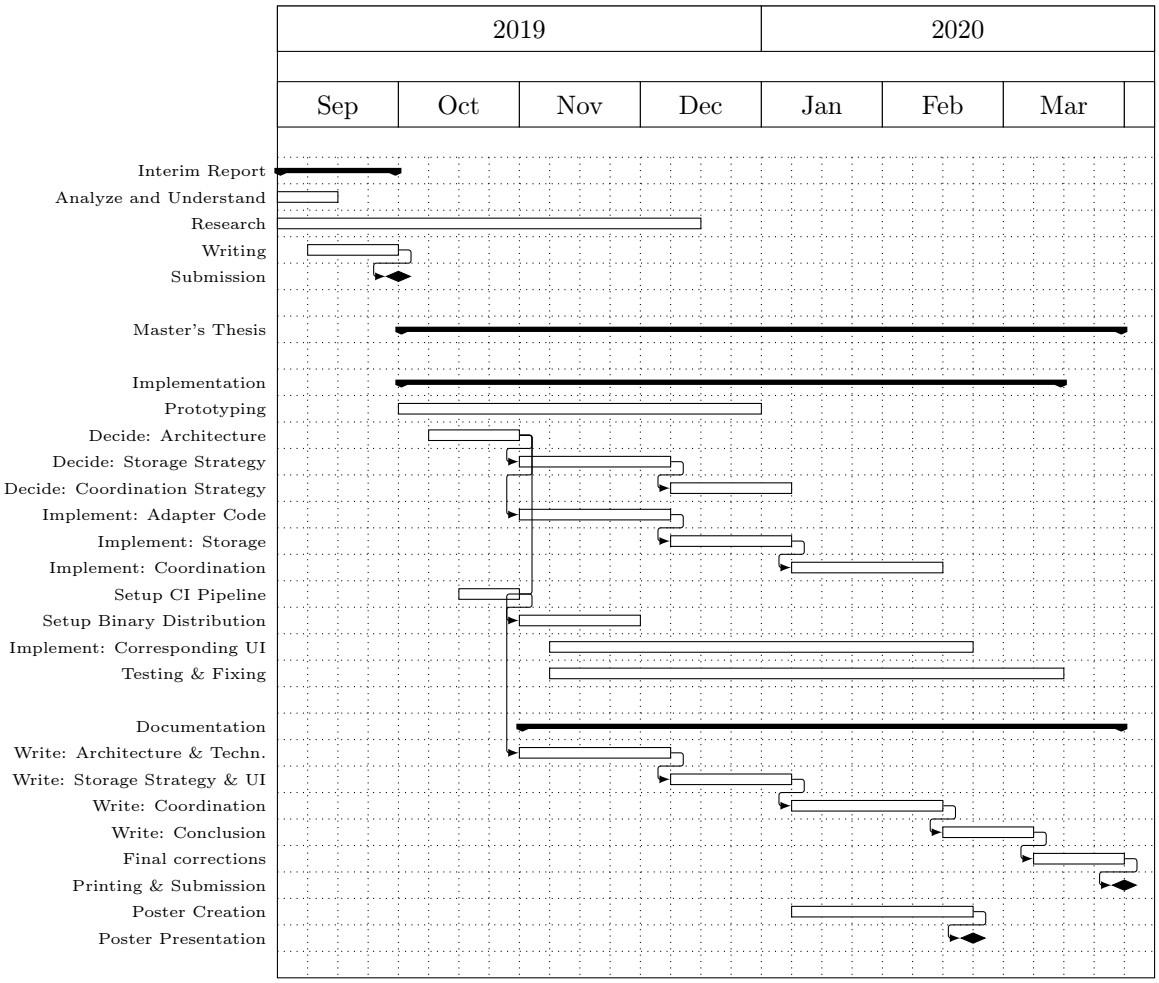

Figure 5.1: Time schedule

Bibliography

[1] MEC-View Dr. Rüdiger W. Henn. *Mobile Edge Computing basierte Objekterkennung für hoch- und vollautomatisiertes Fahren.* URL: `http://mec-view.de/` (visited on 09/29/2019).

[2] The Apache Software Foundation. *Apache Hadoop.* URL: `https://hadoop.apache.org/` (visited on 09/19/2019).

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

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

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

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

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

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

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

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

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

[12] Bc. Pavel 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).

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

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

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

[16] Data 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).

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

[18] IBM Knowledge 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).

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

[20] The University Of 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).

[21] Swiss National Supercomputing Centre. *High Throughput Scheduler.* URL: https://user.cscs.ch/tools/high_throughput/ (visited on 09/19/2019).

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

[23] OpenIO. *High Performance Object Storage for Big Data and AI.* URL: https://www.openio.io/ (visited on 09/22/2019).

[24] Chris Lu. *Simple and highly scalable distributed file system.* URL: https://github.com/chrislusf/seaweedfs (visited on 09/22/2019).

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

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

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

[28] Inc. Docker. *Docker Hub. Build and Ship any Application Anywhere.* URL: https://hub.docker.com/ (visited on 09/22/2019).

[29] Inc. Docker. *Docker Documentation. Docker Registry.* URL: https://docs.docker.com/registry/ (visited on 09/22/2019).