Pääkkönen et al. Journal of Cloud Computing: Advances, Systems and Applications (2019) 8:9
https://doi.org/10.1186/s13677-019-0132-0

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Online architecture for predicting live video transcoding resources

Pekka Pääkkönen*, Antti Heikkinen and Tommi Aihkisalo

Abstract

End users stream video increasingly from live broadcasters (via YouTube Live, Twitch etc.). Adaptive live video streaming is realised by transcoding different representations of the original video content. Management of transcoding resources creates costs for the service provider, because transcoding is a CPU-intensive task. Additionally, the content must be transcoded within real time with the transcoding resources in order to provide satisfying Quality of Service. The contribution of this paper is validation of an online architecture for enabling live video transcoding with Docker in a Kubernetes-based cloud environment. Particularly, online cloud resource allocation has been focused on by executing experiments in several configurations. The results indicate that Random Forest regressor provided the best overall performance in terms of precision regarding transcoding speed and CPU consumption on resources, and the amount of realised transcoding tasks. Reinforcement Learning provided lower performance, and required more effort in terms of training.

Keywords: Rancher, FFmpeg, Docker, Random Forest, Reinforcement learning, RL-Keras, Gym, Cassandra

Introduction

Video content is provided to consumers with Content Delivery Networks (CDN). Typically video is streamed on-demand, but live streams are increasingly consumed (e.g. game play videos in Twitch, other live video content in YouTube Live). In order to provide video streams with high quality to end users, the original video has to be transcoded for delivery via the CDN. Transcoding is a CPU-intensive process, in which several representations of the original video are created. Typically, the end users adaptively switch between the available video representations due to the variable conditions of the (wireless) network. In live streaming cloud resources have to be capable of providing real time speed of transcoding. Thus, video transcoding as a CPU-intensive task requires powerful computing resources to be utilised. Several commercial companies (e.g. Encoding.com [1], Wowza Media System [2], Bitmovin [3]) provide services for live video transcoding. The companies should have incentive to spend less on provisioning of the transcoding services with rented or proprietary resources. Machine learned models [4] have been used for improving efficiency of transcoding with cloud resources, and some of the approaches [5–10] have been developed for the live video transcoding context. Particularly, machine learning was utilised for predicting performance of transcoding tasks with available virtual resources. The predictions were utilised for selecting the most suitable set of virtual resource(s) for transcoding.

However, architecture design for facilitating prediction of virtual resources for live video transcoding based on online data collection on Kubernetes platform hasn't been studied (to the author's best knowledge), which is the contribution of this paper. Several challenges were encountered during this research. Realisation of accurate predictions was complicated by the way Kubernetes allocates and schedules CPU cores for the transcoding tasks. Also, a new data collection method had to be designed for supporting machine learning based on live video transcoding tasks. Finally, variability of video transcoding environment complicated prediction of performance. For example, configurations of transcoded video representations (e.g. resolution, bit rate), video encoder, or cloud resources may change. In order to predict suitable cloud resources, the changes of the environment should be considered in the development of the prediction models.

The goal of this research is to find out how cloud resources can be efficiently utilised based on predictions

* Correspondence: pekka.paakkonen@vtt.fi
VTT Technical Research Centre of Finland, Kaitoväylä 1, 90570 Oulu, Finland

in a cloud platform, where transcoding is realised with Docker containers running on a cloud-based Kubernetes platform. Particularly, an online architecture for predicting live video transcoding resources has been validated with a prototype. Also, prediction of transcoding speed and CPU consumption has been analysed in several learning configurations. The results of the experiments indicate that Random Forest (RF) regressor achieved the best overall performance in prediction, when compared to Reinforcement Learning (RL) or Stochastic Gradient Descend (SGD) regressor in this particular case.

The paper is structured as follows. First, related work is presented. Subsequently, the architecture is described from different viewpoints. Then, the executed experiments are presented, which is followed by analysis of the results. Next, the lessons learnt are discussed. In the end, future work is presented, and the study is concluded. The Appendix contains detailed views (data, sequence) of the architecture.

Related work

Related work is reviewed in terms of Docker/Kubernetes, cloud based architectures for video transcoding, and machine learning techniques for cloud resource management.

Docker ecosystem is increasingly adopted in the IT industry [11, 12]. Rancher [13] is a management suite, which supports deployment of Docker-based services on multiple cloud domains. Cloud resources are registered to Rancher, and services can be deployed on the resources based on Rancher Compose [13], Docker Compose [14], or Helm chart [15] descriptions. The 2.x version of Rancher can be utilised for management of services on Kubernetes clusters. Rancher's service catalog consists of descriptions for facilitating deployment of services. Prometheus [16] is a resource monitoring tool, which is available for deployment as a part of Rancher's service catalog.

Kubernetes [17] is a system for automating the management of Dockerized applications. Pod is a concept in Kubernetes, which refers to a set of tightly-coupled containers deployed on a node [18]. In order to utilise common CPU resources efficiently with multiple Pods, access to the CPUs can be limited. CPU resources can be reserved for Pods with CPU requests and CPU limits [19]. Kubernetes guarantees a specified minimum amount of CPU cores to a Pod based on the CPU request. CPU limit is the upper level of CPU cores, which can be utilised by the Pod. If the Pod tries to use more resources than has been allowed, Kubernetes restricts access of the Pod.

Several architectures have been used for managing the video transcoding process, which should be considered in the context of this paper. Twitch usage has been analysed for motivating the need for adaptive bitrate streaming (ABR) [20]. The idea is to adjust the trade-off between increasing Quality of Experience (QoE), and reducing bandwidth by selectively deciding, which videos to transcode (for ABR). An integer linear programming model has been developed for transcoding of live adaptive streams [7]. Elasticity support for cloud computing [21], and on-demand QoE aware transcoding [22] in 5G networks have been proposed. Video transcoding tasks can be distributed among multiple nodes with Storm [23]. Morph [24] has been used as part of a distributed system for video transcoding, which is able to predict execution time of transcoding [4]. An architecture has been proposed [5], in which video files are initially divided into several Group of Pictures (GOP), which are scheduled for live video transcoding. Measurements of related GOP transcoding tasks of the same video stream are utilised for choosing the optimal virtual machine (VM). Further work has extended the solution for heterogeneous resources [25]. Resources have been allocated based on queueing theory predictions [8]. Partial pre-transcoding of videos and re-transcoding rest of the video stream based on demand has been suggested [26]. The method can significantly reduce resource consumption cost (70%), when the amount of frequently accessed videos increases. Chen [27] presented cloud-based service platform for video transcoding, which is able to lower the cost of provisioning. Performance of parallel and sequential Video on Demand (VoD) transcoding has been measured on heterogeneous VMs [28]. The aim was to create a predictive model for improving delivery time of the system. Vbench is a benchmark for comparing cloud video services [29]. The results indicated that GPUs are more suitable for live streaming scenarios due to higher speed and quality of video produced. A priority-based resource provisioning scheme for video transcoding has been developed [9]. The results indicate that the solution can guarantee Quality of Service (QoS) requirements of live video transcoding, while reducing resource consumption.

Different machine learning techniques have been utilised for supporting decision making in cloud resource usage, which is the main focus of this paper. Earlier studies reported the use of regression [30–32], Markov chain [33], decision trees [6], neural networks [31], Bayesian network [34], or polynomial approximation [22]. Deep RL refers to learning through interaction [35]. An RL agent interacts with the environment, and observes consequences of its actions based on rewards. The goal of the agent is to learn a policy, which maximises rewards. RL can be modelled as a Markov Decision Process consisting of states, actions, transitions, and rewards (and its discount factor). RL has been utilised in many real world applications such as learning to play Go (AlphaGo), Atari's

2600 video games, production scheduling [36], and video transcoding [10]. Random Forests have traditionally been successful for solving classification problems (examples in [37]). When Random Forests are applied for solving problems with big data, new approaches can be used for improving performance (e.g. sub sampling, parallel environments or online adaptations, divide-and-conquer) [38]. Finally, Stochastic Gradient Descent (SGD) is an iterative method for optimizing an objective function. SGD is one of the many gradient descent options for training of various machine learning models including neural networks. Iterative nature of the method enables online learning from mini-batches of data, which makes SGD regression interesting from the research perspective of this paper.

The review of related work indicates that video transcoding process has been scaled within the cloud domain [4, 8, 22, 28], and also from live streaming point of view [5–10]. However, architecture design for cloud resource management has not been focused on (to the authors' best knowledge), when transcoding resources are predicted based on online data collection. Especially, performance of different machine learning techniques has not been compared for live video transcoding in a Kubernetes-based environment.

Based on the literature study, the following research questions (RQ) were posed:

- RQ 1: What kind of architecture facilitates online data collection based prediction of cloud resource allocation for live video transcoding?
- RQ 2: How to allocate resources (based on online data collection) on a cloud-based computing platform for live video transcoding?

An experimental architecture was designed and implemented as a proof-of-concept prototype for providing an answer to RQ 1. Experiments were performed with the prototype for evaluating performance of prediction algorithms (RQ 2) based on statistical evaluation.

Architecture design

In this chapter architecture design is presented with use case, deployment, and sequence views with Unified Modelling Language (UML) diagrams. The Appendix includes more detailed views (data, big data, sequence) of the architecture.

Use case view

Figure 1 presents the use case view of the architecture. Live video can be produced by the end users (e.g. mobile

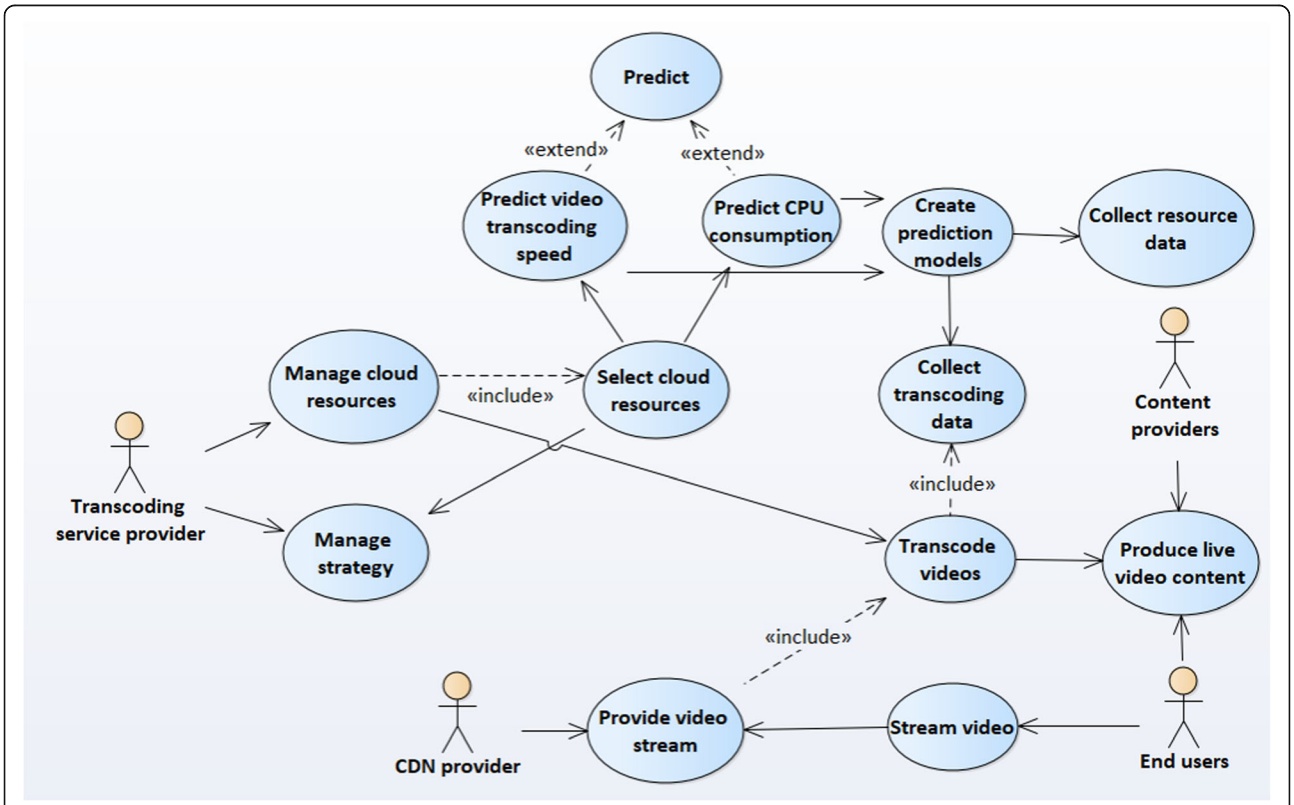

Fig. 1 Use case view of the architecture (adapted based on [6])

phone apps) or other content providers (e.g. web camera). The produced content is transcoded into different representations (resolution, bit rate, frame rate etc.). Data are collected from the transcoding process, and from the cloud resources, which are utilised for the transcoding. The collected data is utilised for the creation of prediction models with machine learning. The models are utilised for predicting transcoding speed and CPU consumption on resources. The predictions are used, when selecting an optimal cloud resource type for transcoding. A transcoding service provider manages the transcoding process, and defines a strategy for managing the transcoding process. The purpose of the strategy is to enable specification of goals by the transcoding service provider (in this work CPU consumption of VMs or transcoding speed). The main long term goal of the transcoding service provider is assumed to be provisioning of good QoS (i.e. transcoding speed) with efficient utilisation of the transcoding resources. In practise, the Transcoding service provider has to select a VM from heterogeneous resources (VMs) to be utilised for execution of the transcoding task on a cloud platform. Finally, the CDN provider delivers the transcoded video files to the end users for streaming.

Deployment view

The deployment view of the architecture is described in Fig. 2. All the nodes were VMs, and managed with the Eucalyptus cloud computing environment. Video source-node acted as the source of video files, which were served by NGINX [39]. FFmpeg [40] (Transcoding-node) was used for video transcoding on VMs of different sizes. Node exporter (Transcoding-node) collected resource statistics to Prometheus from the transcoding nodes. FFmpeg progress information (e.g. speed statistics) was saved to Cassandra [41] via a Collector (Monitoring-node). A separate VM (Learner-node) was dedicated for creating predictors based on machine learning. The Learner periodically trained a new model based on stored transcoding and CPU consumption measurements. A Gym-environment was created for training of the Reinforcement Learning-based model (Learner-node).

The transcoding process was managed on the Management-node. Tester created configurations for the experiments, which were forwarded to the ServiceScheduler. DecisionMaker selected a suitable preallocated VM based on information provided by the Predictor. ResourceManager provided information of available cloud resources. TranscodingDataCollector calculated transcoding statistics, when a new transcoding task was started or stopped. Further, ResourceDataCollector calculated resource consumption based on information provided by Prometheus.

Kubernetes' components were installed on the Monitoring nodes, and the Transcoding nodes by Rancher. One Kubernetes cluster was used, in which the Monitoring-node contained most of the components for controlling resource allocation and scheduling. Transcoding-nodes acted as workers in the Kubernetes cluster.

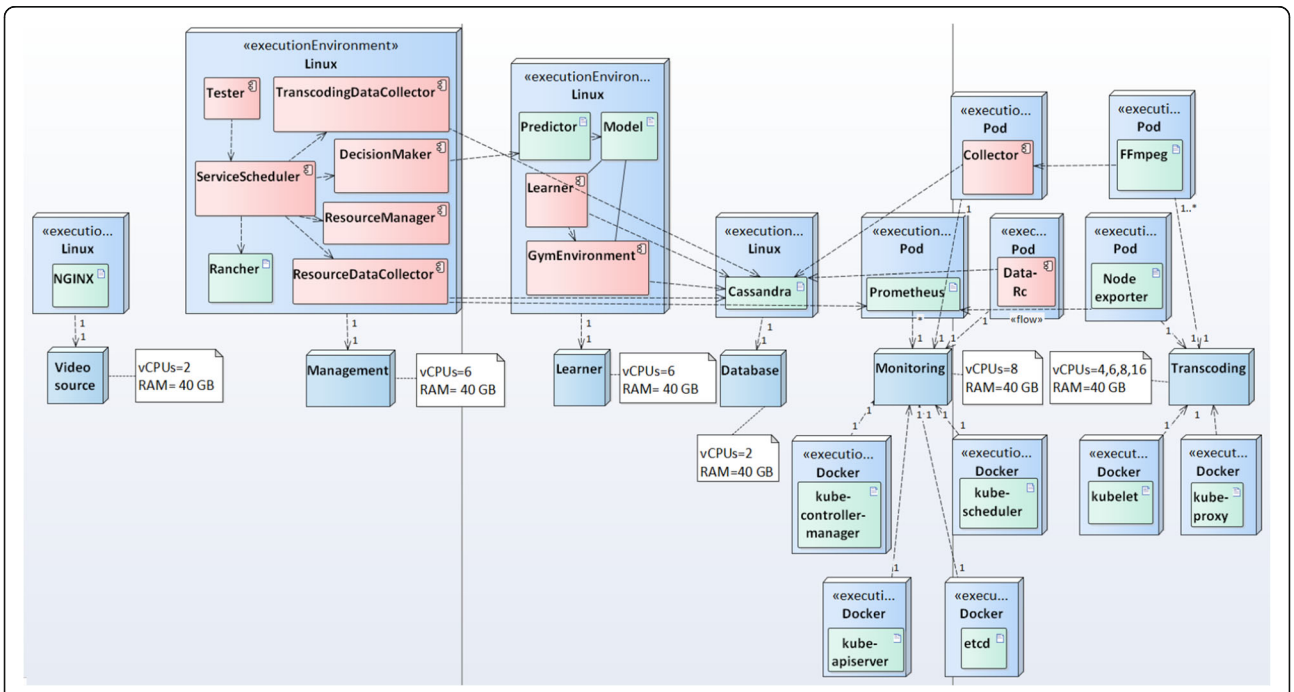

Fig. 2 Deployment view of the architecture

Sequence view

Figure 3 presents the sequence view for starting of the video transcoding service. The steps are as follows:

Steps 1–2: ServiceScheduler starts a new video transcoding task based on the service specification. A resource/VM is requested from the DecisionMaker based on the service specification and the strategy.
Step 3: Available resources/VMs are fetched from the ResourceManager.
Step 4: Count of transcoding tasks is queried from the resources.
Step 5: Count of transcoding tasks on resources is embedded into the HTTP POST request, which is sent to the Predictor. The predictor provides the received information as input to the prediction model, and returns prediction(s) as output (in HTTP 200 OK).
Steps 6–7: The prediction is saved into Cassandra, when SGD/RF is used as an algorithm for prediction.
Steps 8–9: The most suitable resource is selected based on the prediction(s). When RL is used, the smallest VM (lowest count of CPU cores) with a positive prediction is selected. When SGD/RF is used, the smallest VM with an acceptable prediction (satisfies the target in the strategy) is selected. Subsequently, it is checked if the transcoding task can be scheduled on the resource based on Kubernetes CPU requirements. Finally, a VM with the smallest CPU count is selected.
Step 10: The service specification is updated with Kubernetes CPU requirements (CPU request and CPU limit) based on the type of transcoding and resource.
Steps 11–12: The information of the service specification is transformed into service parameters, which are provided via Rancher Command Line Interface (CLI). The parameters include transcoding information for FFmpeg (e.g. target resolution, bit rate, frame rate etc.), Kubernetes CPU requirements, and the resource where the transcoding task will be deployed.
Step 13: Information of the started transcoding task is saved into Cassandra.
Step 14: The started transcoding task is indicated to the Resource-object, which keeps track of executed transcoding tasks.

Prototype

The configuration of the different nodes has been depicted in the deployment diagram (Fig. 2). The experiments were performed with a Dell Server Rack (R820, 32 CPUs, 512 GB RAM). Eucalyptus [42] is a technology for building and managing private cloud environments. Eucalyptus v3.4.2 cloud was installed on Cent OS 6.5. Rancher (v2.1.1) was used for managing VMs, which were created to the Eucalyptus cloud system. Docker CE (v17.03.2) [43] was installed on each node. Prometheus (v6.2.1) [16] was installed as a plug-in from Rancher's service catalog. Prometheus automatically collected CPU consumption data from the VMs with Node exporters (v0.15.2). Grafana [44] is a time series-based solution for monitoring data visualisation with dashboards. Grafana (v5.0.0) was used for visualising CPU consumption of the nodes. Apache Cassandra [41] is a wide column database, which is able to provide high availability and

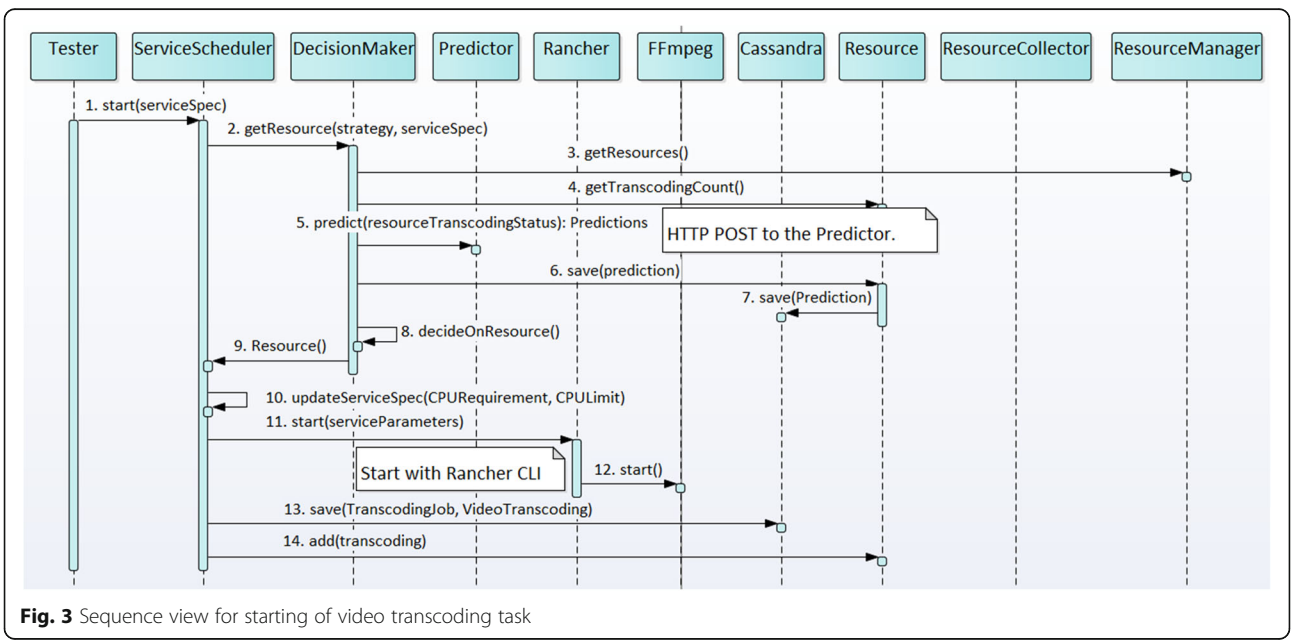

Fig. 3 Sequence view for starting of video transcoding task

scalability for big data. Apache Cassandra (v3.11.3) was used as a database for storing collected measurement data.

SGD, Random Forest predictors, and related data models were created with Python (v3.5.2) and related data science tools (scikit-learn [45], numpy [46]). Reinforcement Learning implementation was created to the Gym framework (v0.10.5) [47] with Keras (v2.2.2) [48] and Tensorflow (v1.5.0) [49]. Keras provides a high level Python-based API for accessing deep learning functionality provided by lower level libraries (e.g. Tensorflow). Tensorflow is an open source library for machine learning. Gym is a toolkit for developing RL algorithms. Gym consists of an environment and an RL agent. The agent provides actions to the environment, which returns rewards and observations as a response, which can be utilised for training of RL models. Keras-RL includes implementation of RL algorithms, and is compatible with Gym and Keras. Keras-RL (v0.4.2) [50] was used for building a neural network based model (as a DQN Agent). Flask [51] is a micro framework for development of web services. Flask (v0.12) provided a REST API framework for the predictor.

FFmpeg is a multi-media toolkit for processing (e.g. encoding, decoding, demuxing etc.) of video/audio content. A Docker image including FFmpeg (v4.0) [52] functionality was used for video transcoding to the MPEG-DASH (Dynamic Adaptive Streaming over HTTP) format [53]. With MPEG-DASH the video stream is split into short media segments. A Media Presentation Description (MPD) indicates to the video client, how the individual segments form a video stream. In order to support adaptivity, the MPD file may include descriptions of multiple video presentations. By utilising the video descriptions, the client may dynamically switch between the available video representations on a per segment basis.

Experiments

First, the design of measurement data collection functionality is presented. Subsequently, the experiments regarding real time transcoding speed and measurement variance are described. Then, the experiments for training data collection and data modelling are provided. Finally, online prediction experiments are presented.

Design of measurement data collection

The design of measurement data collection has been illustrated with an example (Fig. 4), which will eventually be utilised for building of prediction models with machine learning. Three new transcoding tasks (X, Y and Z) are created sequentially (at timestamps T1, T2 and T3). Measurement data is collected every time a new transcoding task is started or stopped. For example, when the third transcoding task is started at T3, measurement data will be collected for the previous transcoding tasks (X and Y) for the measurement period T2-T3. Average transcoding speed will be calculated for both transcoding tasks, and the lowest transcoding speed will be associated with the transcoding configuration (X and Y). The collected measurement data contain the following attributes:

features = (f1, f2), where

- f1 = number of simultaneous transcoding tasks (target resolution, bit rate, frame rate) in a single VM during T2-T3
- f2 = VM type (vCPU count = 4,6,8,16; memory = 40 GB)

label = the lowest average transcoding speed during T2-T3.

Measurement data regarding CPU consumption will be collected by calculating average CPU consumption in a VM during T2-T3 based on Prometheus measurements. The following query is sent to Prometheus [16] (measurement period = 60 s):

$$1-irate(node\_cpu\{mode = "idle"\}[60s])$$

The reply from Prometheus contains percentual CPU consumption of all CPU cores, which is accumulated to a sum. The sum (of the CPU consumption) is divided by the count of the CPU cores to get the average CPU consumption of the node per CPU core. The average CPU consumption is labelled with the transcoding configuration (see features above), when the measurement data is stored.

When the third transcoding task is stopped (at T4), transcoding speed is measured for the previous transcoding tasks (X, Y, and Z) during measurement period T3-T4, and the minimum speed in collected as a measurement (1). Also, average CPU consumption of the VM

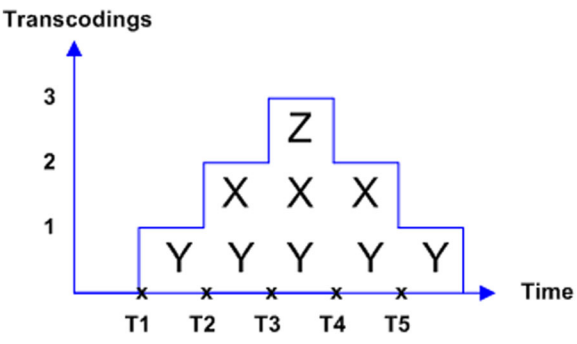

Fig. 4 Design of measurement data collection. x marks the timestamps (T1-T5)

is collected (as described above with the Prometheus query). Accuracy (2) is determined by calculating difference to the realised measurement (transcoding speed/CPU consumption). For example, a prediction was made at T3 for lowest transcoding speed (regarding all three transcoding tasks), when the third transcoding task is started. The related measurement (lowest transcoding speed) is collected at T4, when the third transcoding task is stopped.

$$M_{T4,Speed} = Min\left(Speed_{T3-T4,X}, Speed_{T3-T4,Y}, Speed_{T3-T4,Z}\right) \quad (1)$$

$$Accuracy = \frac{|M_{T4} - P_{T3}| * 100}{M_{T4}} \quad (2)$$

where P = prediction.

It should be noticed, that accuracy can only be calculated, when new transcoding tasks are started or the last transcoding task (Z) has been stopped. For example, when the second transcoding task is stopped at T5, the measurement cannot reliably be associated with the earlier prediction, which was related to having two transcoding tasks in a VM (at T2). Instead, the third transcoding task (Z) was started later on, which would skew the results.

Real time transcoding speed and measurement variance

Before measurement data could be collected, a few experiments were executed with the prototype. The goal of the first experiment was to find out the minimum count of vCPU cores needed for achieving real time transcoding performance with different target resolutions and bit rates (Table 1). Average and minimum transcoding speed were measured.

Table 1 Parameters in the experiment for finding the minimum count of vCPU cores in a VM, which achieves real time speed in transcoding

Parameter	Description
Resolution (bit rate)	256 × 144 (197 kilobits per second (kbps)), 426 × 240 (338 kbps), 640 × 360 (829 kbps), 854 × 480 (1380 kbps)
Video files	50 popular video files (length > 4 min). Resolution: 1920 × 1080, 24 Frames Per Second (FPS), Video format: MKV
Test length	3 h (a new transcoding task every 4 min)
Audio encoding	AAC (128 kbps)
Video encoder	H.264, GOP length = 24, closed-GOP
Output format	DASH ISO Based Media File Format, segment length = 1 s
Video transcoder	FFmpeg with real time flag (simulates live transcoding)

The goal of the second experiment was to find out how much variance different vCPU request levels create to measurements. Low variance would lead to deterministic results in prediction. The parameters of the experiment are provided in Table 2. Similar video/audio encoding, output format and transcoder were used as in the previous experiment (Table 1).

The following CV (Coefficient of Variation) metrics were used in the experiment:

$$CV = \frac{Std\left(\sum_{i=0}^{Measurements} Speed_i\right)}{\sum_{i=0}^{Measurements} Speed_i / i} \quad (3)$$

$$CV_{Avg} = \frac{\sum_{i=0}^{\text{Configurations}} CV_i}{\text{Configurations}} \quad (4)$$

, where.

Measurements = measurements in a transcoding configuration.

Speed = Average transcoding speed.

Configuration = VM associated with a specified amount of simultaneous transcoding tasks.

CV indicates the relationship between variance and mean in similar transcoding configurations. CV_{Avg} indicates average CV across all transcoding configurations. CV_{Avg} enables comparison of variance in different transcoding configurations.

Training data collection

Finally, training data was collected by creating new transcoding tasks in a setup where vCPU request and vCPU limit were fixed (Table 3) based on the results of the previous experiment (Table 2). The fixed vCPU parameters enabled low measurement variance in similar configurations. Other parameters of the experiment were similar as described in Table 2.

Machine learning methods for training of the prediction models

The main utilised machine learning methods were supervised learning and reinforcement learning. Random Forest regressor and SGD regressor were used (as supervised learning methods) for solving a regression problem, because a quantity (transcoding speed, CPU consumption) was predicted. With RL the problem was modelled as a Markov Decision Process [35], which consisted of the RL environment, and the RL agent (with state and action). Particularly, the RL environment returned a reward, when selecting a particular cloud resource (VM) for transcoding. A positive constant reward (Table 4) was returned, when a specified goal (transcoding speed or average CPU consumption) was achieved in a particular state (configuration of transcoding tasks in VMs). A negative constant reward (Table 4) was

Table 2 Parameters in the experiment for finding vCPU request level from Kubernetes, which achieves low variance in measurements

Parameter	Description
Resolution	256 × 144 (197 kbps), 640 × 360 (829 kbps), 854 × 480 (1380 kbps)
Video files	50 popular video files (length > 4 min)
Test length	10 h (a new transcoding task every 15 s)
Transcoding episode length (new transcoding tasks are scheduled)	900 s
Idle period after an episode (no transcoding tasks are scheduled)	300 s
vCPU limit	4 vCPU = 256 × 144 (197 kbps), 5 vCPU = 640 × 360 (829 kbps), 6 vCPU = 854 × 480 (1380 kbps)
vCPU request	x %*vCPU limit per VM
Minimum length of constant transcoding configuration in a VM for measurements	30 s

returned, when the goal wasn't achieved in a state. The purpose of the positive/negative reward was to indicate to the RL agent, which state transitions would lead to a desired/undesired outcome in terms of transcoding speed or CPU consumption. Particularly, the rewards were utilised in the training of a neural network (RL agent), which was utilised for decision making regarding the utilised VM for transcoding.

First, prediction models were created offline based on the training data set. The purpose of the models was to provide initial predictions, before any new data would be available for training and fine-tuning of the models in online prediction experiments. The models were trained based on the type of VM (4, 6, 8, 16 vCPUs), and the amount of simultaneous video transcoding tasks per target resolution executed in a VM (the features were described earlier in the 'Design of measurement data collection'-section). The features were labeled with minimum average transcoding speed among video transcoding tasks executed in a VM, or with average CPU consumption on a node. Table 4 presents machine learning models, and parameters of the models.

Additionally with RF and SGD, the data was cross-validated (CV) [54] with K-Fold 10. Prediction accuracy was calculated as Mean Absolute Percentage Error (MAPE) as follows (where.

Table 3 The parameter in transcoding experiment for creating a training data set

Parameter	Description
vCPU request	45%*vCPU limit for VM with 4 vCPU, 52,5%*vCPU limit for VM with 6 and 8 vCPU, 57,5%*vCPU limit for VM with 16 vCPU

P = prediction, M = measurement):

$$Accuracy_{MAPE} = \frac{\sum_{i=0}^{Samples} ABS\left(\frac{P-M}{M}\right) * 100}{Samples} \quad (5)$$

Finally, cross-validation accuracy (6) was calculated based on the accuracies of the folds (Avg = average, δ = deviation):

$$Accuracy_{CV} = Avg_{Accuracy_{MAPE}} \pm 2 * \delta_{Accuracy_{MAPE}} \quad (6)$$

Online prediction

Configuration of suitable cloud resources was predicted and scheduled online for live video transcoding. The trained models were utilised for performing initial predictions. Additionally, the models were updated online (every 30 min) based on new transcoding data. Performance of the models was compared to the reference case, in which 100% of vCPU limit was requested for each transcoding, and transcoding tasks were started randomly to an available VM. Parameters of the experiment are described in Table 5 (other parameters are similar as in Table 2).

Precision and accuracy in prediction was evaluated based on formulas 6 and 7:

$$Precision = \frac{SamplesIn\ RequiredRange}{Samples} * 100 \quad (7)$$

Online prediction: a new target resolution

In the experiment a new target resolution/bitrate (426 × 240/338 kbps) was transcoded, when there was no earlier data available for training, where the new resolution would have been transcoded. The goal of the experiment was to find out, how fast new accurate predictions can be created online. The parameters of the experiment are described in Table 6 (other parameters are similar as in Tables 2 and 5).

Online prediction: cold start

In the experiment no earlier data was available at the start of the experiment (cold start). New transcoding tasks were created until at least 100 measurement samples had been collected, and the predictor was trained every 5 min afterwards. The parameters of the experiment have been described in Table 7.

Online prediction: twitch

In the experiment online live video data sources from Twitch game play were utilised for transcoding. Video from Twitch is in HTTP Live Streaming (HLS) format, when the transcoder receives a one-second video file

Table 4 The machine learning models and utilised parameters

Model	Parameters
RL	Neural network = Input: 25 Integers; 3*(32 Unit layers+RELUs), output: Integer (linear activation) DQN Agent (target_model_update = 1e-3, nb_steps_warmup = 50, policy = Boltzmann Q Policy), Adam Optimizer (learning rate = 1e-2), Training steps = 4000 Reward: 0.4 (goal achieved), −0.5 (goal not achieved)
RF	RandomForestRegressor(n_estimators = 100)
SGD	SGDRegressor (max_iter = 1000)

every second. The parameters of the experiment have been described in Table 8.

Results and evaluation

Real time transcoding speed and measurement variance

Table 9 presents transcoding speed, when source video files were transcoded to different target resolutions. 6 vCPUs were required for transcoding the four smallest resolutions with ~ 0.99x (1.0x = real time) average transcoding speed. However, the largest resolution (1280 × 720) required too much CPU resources for the experiments. Thus, the four smallest target resolutions were utilised in further experiments.

Different values for vCPU request and vCPU limit were experimented, when three target resolutions were tested (see Table 3). The goal was to achieve standard deviation to a lower than 5% level of the mean. Table 10 presents CV regarding transcoding speed with the specified vCPU request levels. Thus, acceptable values for vCPU request (Table 10) and vCPU limit (Table 9) were found, which were utilised in training data collection.

Model training

Table 11 presents offline cross validation accuracy, when machine learning experiments were executed with the training data set. It can be seen, that RF achieves better accuracy than SGD regressor. Accuracy of CPU consumption prediction is much lower in comparison to the prediction of transcoding speed.

Table 5 Parameters in online prediction experiments

Parameter	Description
Video files	20 popular unseen video files from YouTube. Resolution: 1920 × 1080, 24 FPS, Video format: MKV
Prediction algorithms	RF, RL, SGD
Reference	vCPU request = vCPU limit
Online model training	Every 30 min
Target	Transcoding speed > 0.98 or 0.99; CPU consumption < 80 or 70%
vCPU request	45%*vCPU limit for VM with 4 vCPU, 52,5%*vCPU limit for VM with 6 and 8 vCPU, 57,5%*vCPU limit for VM with 16 vCPU

Online prediction

In the online prediction experiments, the models created with the training data set were used initially for predictions. Also, every 30 min, the models were retrained with all collected data. ~ 46 (912 data samples in a 10 h experiment) new data samples were generated during the time period (30 min), which were added to the initial data set (Table 11). Thus, the size of the data set increased by less than ~ 5% (46/912*100) during a retraining interval.

Experiments were executed with different targets set for transcoding speed. Figure 5 presents precision of machine learning algorithms, when the target of transcoding speed is varied. Random Forest and SGD regressor achieved best precision in transcoding, while all machine learning approaches achieved at least a precision of 89%. Figure 6 presents the count of executed transcoding tasks. It can be seen that, when cloud resources are predicted with RL/RF, 17–46% higher number of transcoding tasks can be realised in comparison to the reference. SGD regressor performs worst in terms of realised transcoding tasks, even though precision is high.

Figure 7 presents precision, when different CPU consumption levels are targeted. RL achieves a little higher (~ 1–3%) precision in comparison to RF. Accuracy in RF prediction was 10–10.6%, which is close to the cross validation accuracy (Table 11).

Figure 8 presents precision in simultaneous prediction of transcoding speed and CPU consumption with a different target for CPU consumption. RF achieves higher precision than RL (speed: ~ 4–5%, CPU consumption: ~ 5–12%), and also a larger amount of executed transcoding tasks (~ 24–62% (Fig. 9)). RF accuracy in CPU prediction was 12–14.7%, which is a bit lower, when compared to the accuracy in CPU consumption prediction without transcoding speed prediction. However, in overall the precision of both measures is higher, when compared to prediction of only CPU consumption (Fig. 7) or transcoding speed (Fig. 5).

Online prediction: a new target resolution

Figure 10 illustrates precision, when transcoding is executed for a new target resolution and bit rate. Figure 11 presents how precision changes during the test case. It seems that RL and RF achieve more than 90% precision after 1–2 h. Figure 12 presents achieved number of transcoding tasks in the experiments. RF achieves ~ 39% higher amount of transcoding tasks than RL.

Online prediction: cold start

Figure 13 presents precision, when transcoding is executed after a cold start. RF achieves higher precision (~ 8%) than RL, but has a lower amount of transcoding tasks (~ 4% in Fig. 14). ~ 23–28% more transcoding tasks are executed in

Table 6 The parameters in online prediction experiments with a new target resolution

Parameter	Description
Prediction algorithms	RF, RL
Online model training	Every 5 min
vCPU limit	4 vCPU = 256 × 144 (197 kbps), 5 vCPU = 640 × 360 (829 kbps), 6 vCPU = 854 × 480 (1380 kbps), 4 vCPU = 426 × 240 (338 kbps)

comparison to the reference. Figure 15 illustrates how precision improves during the test case, when new measurement data is utilised online for model training. It seems that RF achieves higher precision faster than RL. It takes 3–6 h (~ 200–400 training data samples) for both algorithms to reach a precision higher than 90%.

Figure 16 presents precision, when CPU consumption and transcoding speed are predicted with a cold start. There are no significant differences in precision, when RF and RL are compared, although RF seems to achieve a better overall precision. When the amount of executed transcoding tasks are compared (Fig. 17), RF achieves significantly higher amount of transcoding tasks (~ 63%), when CPU consumption level is targeted below 70%. However, RL achieves higher amount of transcoding tasks (~ 13%), when CPU consumption level is targeted below 80%.

Online prediction: twitch

Figure 18 presents precision, when live video streams from Twitch were transcoded. It can be observed that all predictors achieve a better precision, when compared to the reference. RF achieves the highest amount of executed transcoding tasks (Fig. 19). RL multi-agent approach realises a bit lower amount of transcoding tasks, but has a higher precision. In the reference case, the highest amount of transcoding tasks can be realised with the lowest precision.

Table 7 The parameters in live prediction experiments with cold start

Parameter	Description
vCPU request	47,5%*vCPU limit for VM with 4 vCPU, 52,5%*vCPU limit for VM with 6, 55%*vCPU limit for VM with 8 vCPU, 60%*vCPU limit for VM with 16 vCPU
Target	Transcoding speed > 0.99; CPU consumption < 80 or 70%
Prediction algorithms	RF and RL
Minimum amount of samples for training	100

Discussion

In the following, the main lessons learnt are discussed. Accurate prediction of live video transcoding speed on Kubernetes required execution of preliminary experiments. First, live video transcoding speed was tested with each target resolution/bit rate for finding the minimum count of vCPU cores capable of achieving real time speed. The discovered values (vCPU cores) were utilised as CPU limits in Kubernetes for each target resolution/bit rate. Secondly, different CPU request levels from Kubernetes were experimented, and resulting variance in similar configurations was measured. Based on the results, CPU request was defined as a fixed percentage of CPU limit for each VM, in which the transcoding was executed. Finally, when CPU request and CPU limit was specified separately for each transcoding task, transcoding speed or CPU consumption level could be predicted accurately based on the collected measurement data.

A new data collection method had to be designed for enabling machine learning based on online data collection. Data samples were collected, when a new transcoding task was added or removed from the Kubernetes cluster. However, data samples were collected only, when the amount of transcoding tasks had been constant in a VM for a time period (30 s). If the time period would have been smaller, the variance between measurements in similar transcoding configurations would have increased (leading eventually to lower prediction accuracy). On the contrary, the variance would have been smaller with a larger time period, but fewer amount of data samples would have been collected. Thus, determination of a suitable time period for data collection may require a trade-off between the size of collected data, and prediction accuracy.

Prometheus was utilised for collection of CPU consumption data from VMs. Prometheus reported low CPU consumption (typically a few samples) every 2 h during the experiments. This may have been caused by Prometheus flushing samples from memory to disk [57]. Thus, low values (< 10%) of CPU consumption were filtered out of data modelling. The scraping interval of data from Node exporters was reduced to 5 s (default 60 s) for improving granularity of CPU consumption data.

When video was transcoded for a new target resolution and bit rate (with no previous training data), precision improved to over 90% after 1–2 h of testing. Hard-coded positive rewards were returned by the RL environment, until a minimum of 50 samples were collected for the new target resolution/bit rate. The artificial limit (50 samples) was required, because positive predictions required availability of initial training data. Similarly, a minimum amount of 100 training samples were collected, when transcoding was started without training data (a cold start). When the minimum limit

Table 8 The parameters in online prediction experiments with Twitch

Parameter	Description
Video source	Online live video from Twitch. Resolution: 1280 × 720, Video format: HLS (segment length: 1 s)
Prediction algorithms	RF, RL, RL with multiple RL agents (4)
Reference	vCPU request = x%*vCPU limit
Online model training	Every 30 min
Target	Transcoding speed > 1.0
vCPU limit	2 vCPU = 256 × 144 (197 kbps), 3 vCPU = 640 × 360 (829 kbps), 4 vCPU = 854 × 480 (1380 kbps)
vCPU request	35%*vCPU limit for VM with 4 vCPU, 45%*vCPU limit for VM with 6 and 8 vCPU, 50%*vCPU limit for VM with 16 vCPU
Video URL source	Twitch Stream API (v5) [55]
Video URL conversion	Streamlink [56]

was too low (50 training samples), RF didn't schedule any transcoding tasks for some VMs. Based on the experiences, it seems that prediction based on online data collection may require manual configuration of the initial data set size prior to predicting for new target resolution/bit rate(s).

Keras-RL [50] enabled Reinforcement Learning in the prototype. A Gym [47] environment was created for training of the predictor. When a neural network was created, training data was read initially from the database into the Gym environment. Also, target threshold for transcoding speed or CPU consumption was read from the database (Environment in Fig. 21). The Gym environment returned corresponding reward based on measurement samples of a specified transcoding configuration in a VM. For example, when the goal for transcoding speed was > 0.99, a positive reward was provided, if transcoding speed of a random sample (with specified transcoding configuration in a VM) satisfied the goal.

RL agent utilised the observed rewards for training of the neural network. A 3*32 layer neural network for a DQN Agent was trained. The input was a Keras Box [50] (25 integers), which was used for describing transcoding status of VMs (state). The output (action) was a Keras Discrete [50] (one Integer), which was used for representing a new transcoding on a specific resource. It was discovered, that possible new transcoding resolutions or VM types (VM with specified number of CPU cores and memory) should be taken into account in advance, when input/output dimensions of the neural network are designed with Keras. This would enable easier extension of the algorithm, when parameters of the environment change.

Predictions with RL required a lot of configuration for experimentation. Initially, parameters of the neural network (Table 4) were tested with the training data, and loss/reward were monitored. Constant reward type was used for training. Constant [36] and variable [10] reward types have been utilised in the earlier RL studies. The maximum level of training steps (4000), and other parameters (Table 4) were fixed based on the initial testing. However, different rewards (returned from the RL environment) were experimented in the actual tests in order to discover optimal performance in terms of transcoding tasks, transcoding speed or CPU consumption. This led to additional work, which was not required with RF/SGD.

In this work, real time streaming was mostly simulated with FFmpeg (with -re flag) by reading source files with native frame rate. The simulation enabled control to the source video files and their duration, which wasn't possible, when real time streaming sources (e.g. Twitch) were used. Real time speed (1.0x) was never achieved in practise. Instead, transcoding speed lagged a bit behind (i.e. ~ 0.99x) of real time speed. However, when video was streamed from Twitch without the simulation, real time speed was achieved. The downsides of using a real video source (such as Twitch) is possible changes to the

Table 9 Average and minimum transcoding speed with the different transcoding resolutions

Resolution	vCPUs	Average	Minimum
256 × 144	4	0.9937x	0.9803x
426 × 240	4	0.9934x	0.9872x
640 × 360	5	0.9953x	0.9933x
854 × 480	6	0.9894x	0.9782x
1280 × 720	12	0.9793x	0.9534x

Table 10 CV with the specified vCPU request levels

Resource (vCPUs)	vCPU req./ vCPU limit (%)	CV_{Avg}
4	45	0.0345
6	52.5	0.0165
8	52.5	0.0202
16	57.5	0.0335
All	–	0.0278

Table 11 Cross validation accuracy with RF and SGD regressor for transcoding speed and CPU consumption. The size of the training data set for transcoding speed was 912 samples. The size of the training data set for CPU consumption was 911 samples

Algorithm	Transcoding speed: CV accuracy (%)	CPU consumption: CV accuracy (%)
RF	2.53 +/− 0.83	9.60 +/− 7.48
SGD	2.89 +/− 0.71	15.73 +/− 11.42

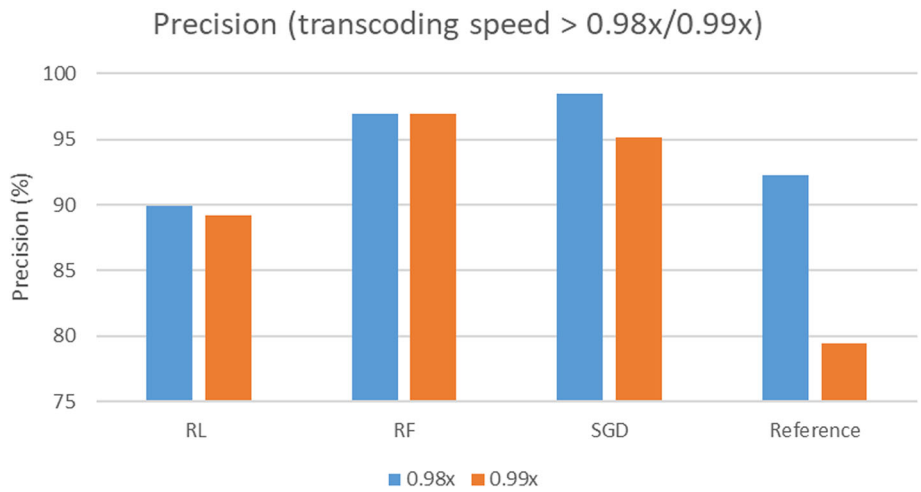

Fig. 5 Precision in online prediction, when the targeted transcoding speed is varied

API or to the video content, quota limitations of the API, and non-repeatability of experiments due to changing video content.

When video was transcoded from live sources (Twitch), transcoding speed wasn't artificially lowered to the frame rate of the original video, a lower resolution/bit rate in the original video (720p) was used, and a different type of video format (HLS) was applied. All/any of these differences may have caused higher amount of transcoding tasks (~ 2.8x) to be realised (Fig. 19), when compared to the results with a simulated live streaming video source (Fig. 6). RF performed better than RL, when Twitch was utilised as a video source. However, the multi-agent approach of RL led to improved performance. Dedicated RL agents were trained for predicting transcoding performance on a single VM, instead of predicting performance on all VMs with one RL agent.

Rancher worked well as part of the prototype system, and enabled easy setup and configuration of services on a Kubernetes cluster. A service catalog entry was created into Bitbucket [58], which contained transcoding service as a Helm description [15]. Location of the catalog entry was configured into Rancher, which caused automatic downloading of the catalog entry content into Rancher via Git [13]. New transcoding tasks were configured and started to Rancher by utilising Rancher CLI from the Service scheduler (Fig. 2). A service catalog entry was also created to Bitbucket for the Collector. Additionally, a Docker image was created for the Collector, which was stored to the Docker Hub [59].

Reliability of this research may be improved by running more test iterations in different configurations. However, the executed experiments (~ 40) required a significant amount of time (~ 2 months) to be completed. Also, tuning of the RL network/DQN agent may

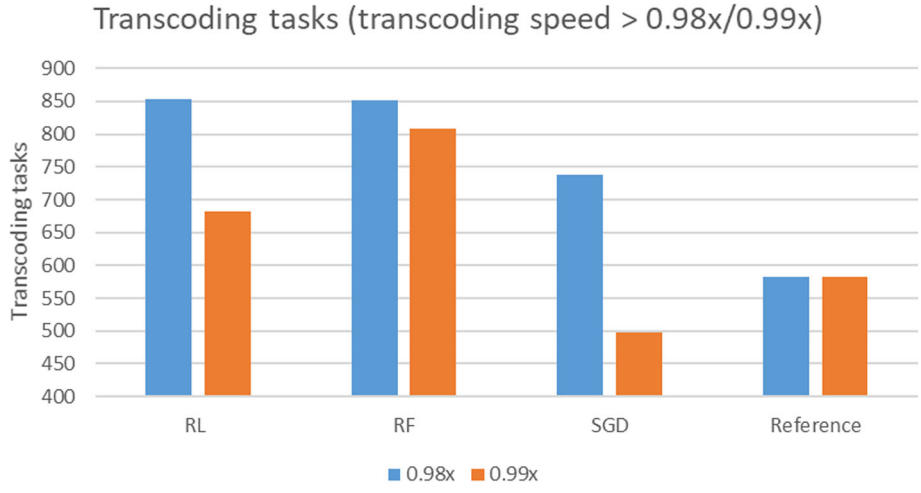

Fig. 6 Number of executed transcoding tasks, when the targeted transcoding speed is varied

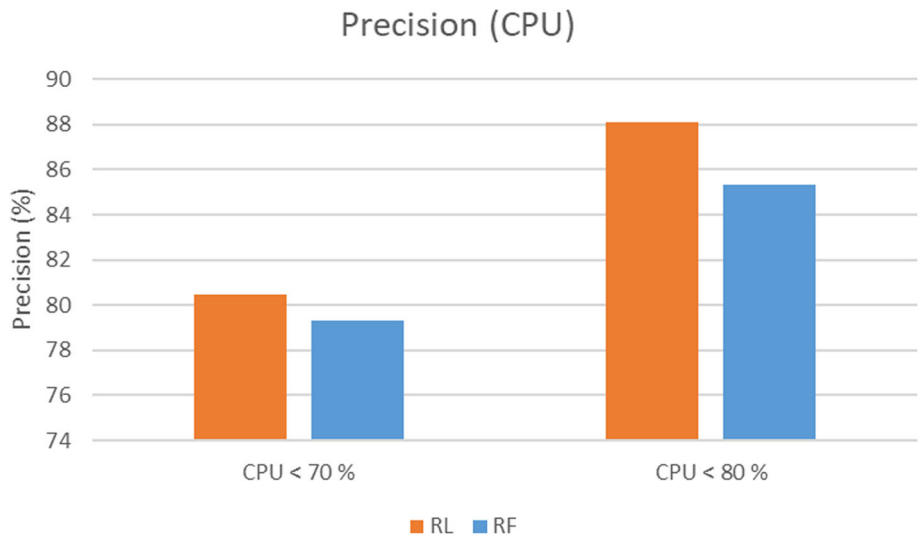

Fig. 7 Precision in CPU consumption prediction with different CPU consumption targets

lead to improved performance. In this work, the choice of parameters available for DQN agent training was experimented with the testing data set. Particularly, the effect of different configurable parameters was observed regarding loss and reward. Subsequently, the parameters (except reward) were fixed (Table 4) in the further experiments. In the future, the effect of the parameters on training model performance may be improved with a grid/randomized search.

Cost of cloud resources wasn't explicitly considered in this work. This work focused on predicting and realising required performance in terms of transcoding speed and/or CPU consumption of VMs with available cloud resources. However, when the transcoding service provider can realise more transcoding tasks with available resources (up to 46% more (Fig. 6)), fewer cloud resources need to be rented, which may lead to a lower provisioning cost.

The results can be compared to related work. This work is continuation of the earlier work [6], in which live video transcoding was predicted based on offline supervised learning with the earlier Rancher (v1.6) platform. The main difference of this work is the capability of online data collection and machine learning for prediction of suitable cloud resources. Also, the focus in this work was on the Kubernetes-based platform with the newer version of Rancher (v2.1). In overall, prediction performance has improved in this work. The main

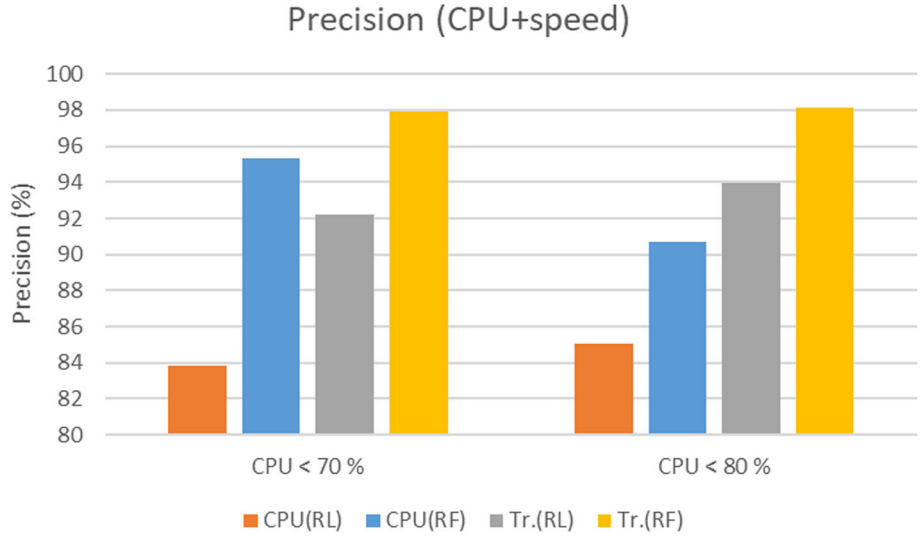

Fig. 8 Precision in CPU consumption and transcoding speed prediction

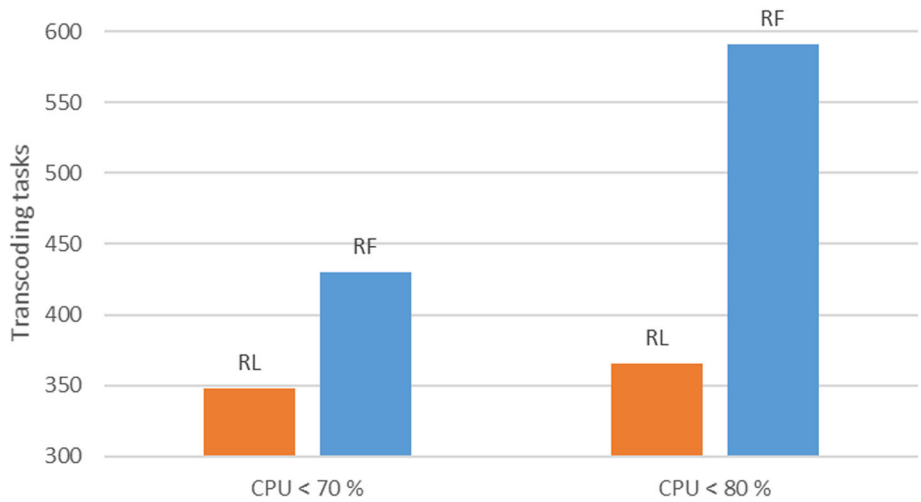

Fig. 9 Amount of executed transcoding tasks, when CPU consumption and transcoding speed were predicted simultaneously

reason may be related to the improved granularity in data collection, and limitation of CPU usage for Kubernetes Pods. In the earlier work, samples were collected in isolated experiments, which were executed offline. The approach presented in this paper is to collect new samples for training online, when a transcoding task is started/stopped.

Other works have focused on live video transcoding [5] [7] [8] [9] [10]. Some of the works are based on simulations [5], solving an ILP problem on commercial decision optimisation technology [7], predicting with a queueing model [8], or RL for live video transcoding with High Efficiency Video Coding (HEVC) [10]. None of the works [5, 7–10] focus on predicting live video transcoding on a Kubernetes-based platform. A preemptive priority-based resource provisioning scheme [9] is closest to our work. The idea is to split video into chunks, which are dispatched into resources with queueing based on QoS requirements. Neural networks and Model Predictive Control were used for predicting video chunk arrival rate. The predictors were validated with a cloud system, which consisted of 20 homogeneous Docker containers, while we focused on the allocation of heterogeneous VMs for live video transcoding. RL-approach for HEVC-based video transcoding [10] didn't consider allocation of cloud resources, but instead focused on optimisation of transcoding based on video quantisation parameters, number of threads, dynamic voltage frequency scaling, and sequence of RL agents. Other approaches [4, 25] have mainly concentrated on

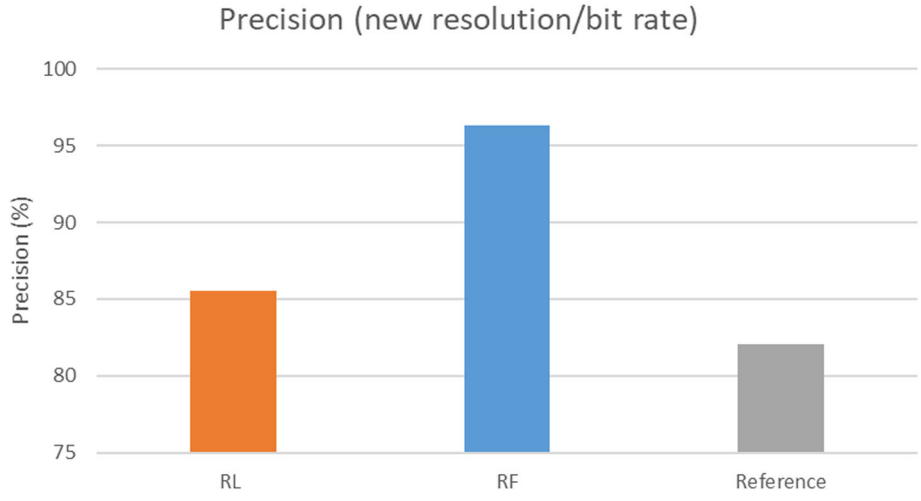

Fig. 10 Precision, when video is transcoded for a new target resolution/bit rate

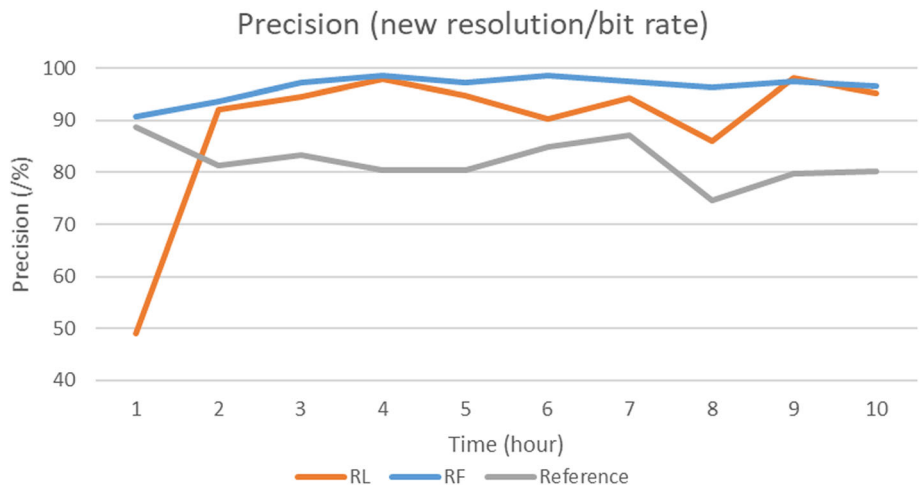

Fig. 11 Hourly precision, when video is transcoded for a new target resolution/bit rate

VoD transcoding. VoD transcoding with Docker has been predicted [4], but prediction on homogeneous VM types was focused on. VoD transcoding tasks were scheduled on heterogeneous resources, where transcoding time was estimated based on related GOP tasks in a queue [25]. None of the studies utilise both simulated and real live video sources in the experiments.

Future work

The simulated video sources were located on the same cloud system with the transcoding VMs. Thus, the impact of location on transcoding speed wasn't taken into account, which can be considered as future work.

The presented online architecture may be considered as a starting point, when designing architecture of future systems, which predict performance of CPU intensive workloads on Kubernetes platform based on online data collection. Also, the selected technologies of this paper may be considered, when building such new systems.

The presented online architecture in the small transcoding cluster is able to collect a relatively small amount of data. It would be interesting to collect big data in a real operating environment, in which thousands of transcoding tasks would be executed in parallel. In such a future scenario, incremental approach of online learning becomes essential, because it may not be feasible to train models with the whole data set(s), as was done in this work.

Conclusion

This work focused on the management of cloud resources on Kubernetes platform for live video transcoding. The first research question (RQ 1) was related to architecture design for facilitating prediction of cloud resource

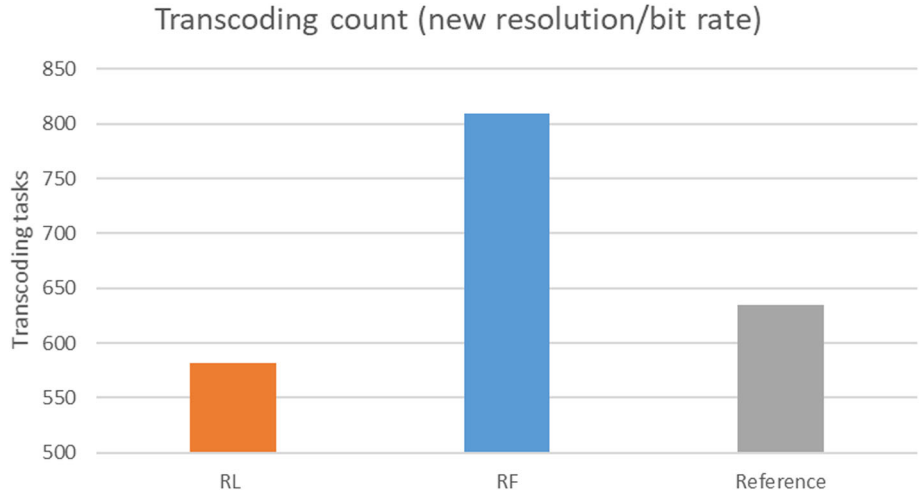

Fig. 12 Number of transcoding tasks, when video is transcoded for a new target resolution/bit rate

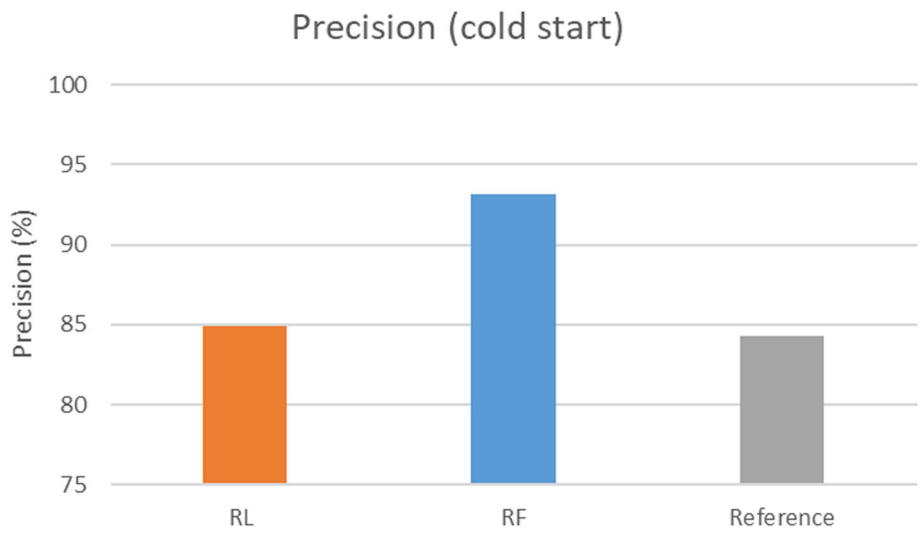

Fig. 13 Precision, when video is transcoded with a cold start

allocation for live video transcoding based on online data collection. An experimental architecture was designed and validated as a proof-of-concept with a prototype, which provides an answer to the research question. The main components of the architecture were transcoding nodes, resource monitoring, Kubernetes cluster, database, predictor, online learner, decision maker, transcoding/resource data collector, service scheduler, and cloud management solution. FFmpeg was used for live video transcoding. Prometheus monitored CPU consumption of cloud resources, which were utilised for transcoding. Training data (transcoding and resource monitoring data) was collected to Cassandra database, when a transcoding task was started or stopped. A new method was designed for supporting machine learning based on online data collection from transcoding tasks. The collected data was utilised in the training of the prediction models. Online learners (Keras-RL, Python scikit-learn) were used for training predictors for cloud resource management. Decision maker utilised predictions for deciding the most suitable resource for live video transcoding. Service scheduler created transcoding tasks on a Kubernetes cluster with Rancher's cloud management platform.

The second research question (RQ 2) focused on the allocation of resources on a cloud computing platform for live video transcoding. For enabling accurate predictions, CPU cores had to be allocated separately for each transcoding task based on the target resolution, and VM. Many experiments were executed in different configurations, in which transcoding on cloud resources was predicted with SGD regressor, RF and RL. A regression problem was solved with SGD regressor and RF, in

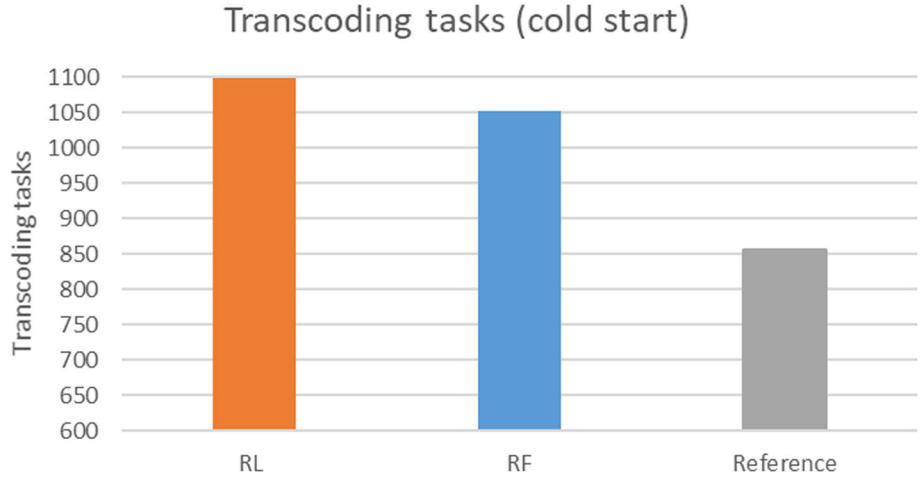

Fig. 14 Number of transcoding tasks, when video is transcoded after a cold start

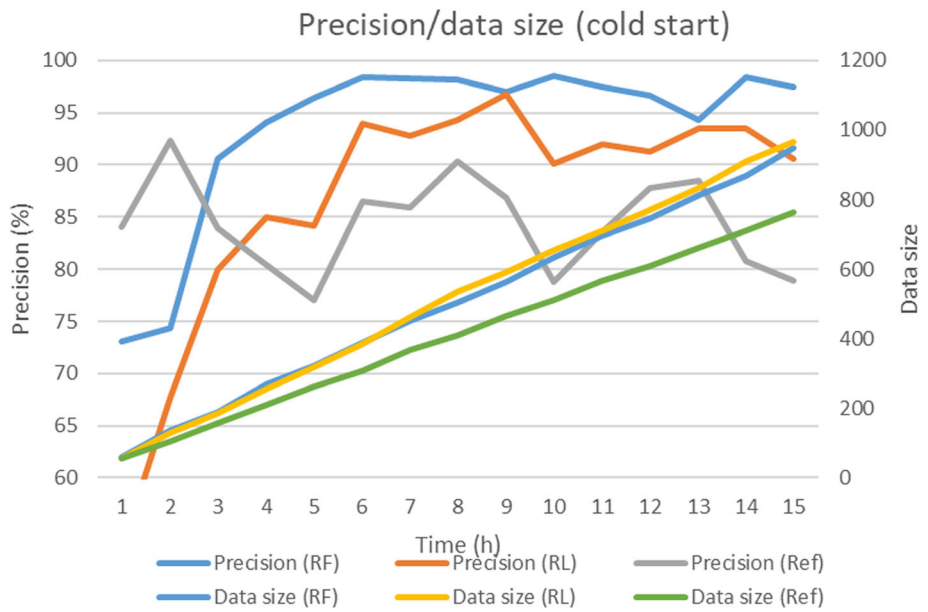

Fig. 15 Precision and size of measurement data with RF/RL, when transcoding after a cold start

which transcoding speed or CPU consumption was predicted, and utilised for allocation of cloud resources. With RL the problem was modelled as a Markov Decision Process [35], which consisted of the environment, and the agent (with state and action). In this case, the RL agent returned a reward for selecting a particular cloud resource (VM) for transcoding. The rewards were utilised in decision making regarding the allocated VM.

The initial experiments were executed for finding out the minimum amount of CPU cores required for achieving real time transcoding speed with different target resolution/bit rates. The results of further experiments indicated, that low variance in similar configurations can be achieved, when CPU request level of Kubernetes Pods is limited as a percentage of CPU limit. CPU limit was determined separately for each transcoding task based on target resolution/bit rate and VM (based on results achieved in the initial experiments). Low variance enabled deterministic prediction of transcoding speed and CPU consumption. Training of the initial models offline indicated that prediction of CPU consumption had a significantly lower accuracy (~ 10–16%), when compared to the prediction accuracy of transcoding speed (~ 2–3%). Online prediction experiments with transcoding speed

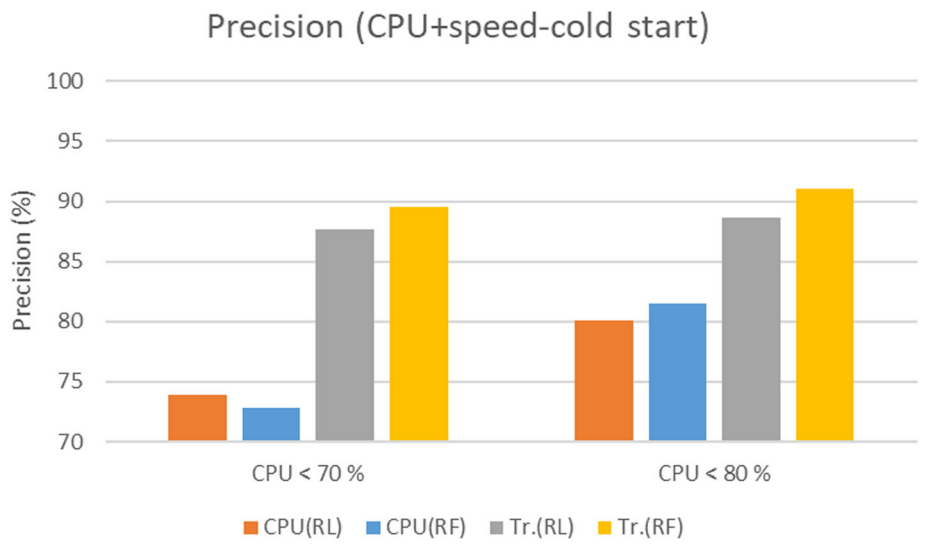

Fig. 16 Precision, when video is transcoded after a cold start, and CPU consumption, and transcoding speed are predicted

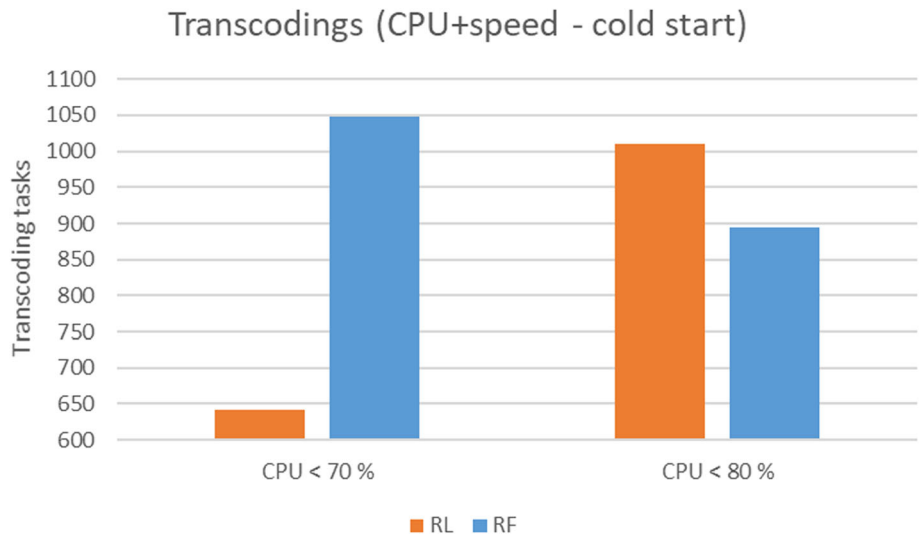

Fig. 17 Transcoding task count, when video is transcoded after a cold start, and CPU consumption, and transcoding speed are predicted

based on models trained offline indicated best overall performance with RF (38–46% more transcoding tasks, when compared to the reference). SGD regressor achieved a good precision, but a significantly lower amount (13–38%) of executed transcoding tasks in comparison to RL/RF. Thus, SGD regressor wasn't used in further experiments.

Additionally, experiments were performed for new target resolutions, which didn't have previous training data. Instead, training data was collected and used for training of the prediction models online. When video was transcoded for a new target resolution/bit rate, RF achieved higher precision (~ 96%) and amount of executed transcoding tasks than RL. When new transcoding tasks were started without any initial training data (cold start), a small data set (50–100 samples) had to be collected for training of the initial prediction model. During online training, precision improved to over 90% within 3–6 h from the start of the experiment. Both RL/RF achieved 23–28% higher amount of transcoding tasks, and higher precision in comparison to the reference. RL achieved higher amount of transcoding tasks, but a lower precision than RF. Finally, both transcoding speed and CPU consumption were predicted simultaneously, and RF had a slightly better precision (CPU: ~ 73–82%, transcoding speed: ~ 90–91%) than RL, but the predictors achieved inconclusive performance in terms of realised transcoding tasks. Experiments with real live video data source (Twitch) indicated that RF had also a better performance than RL.

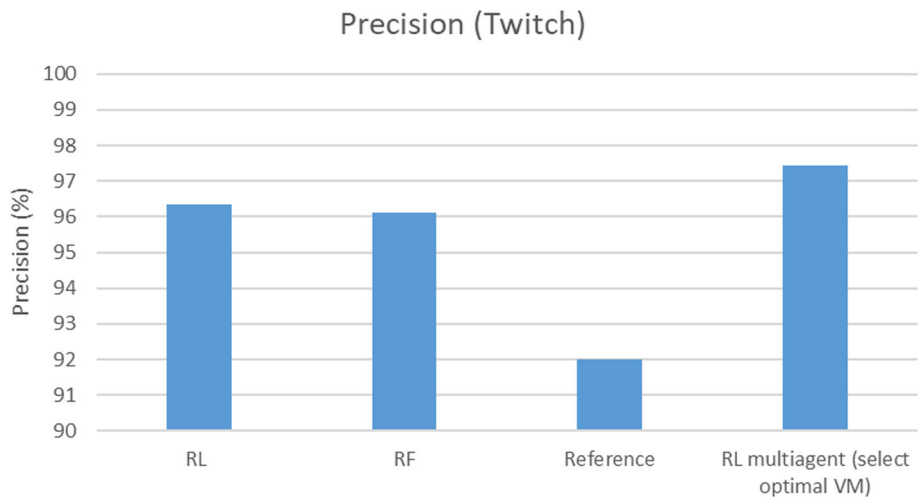

Fig. 18 Precision, when video is transcoded from Twitch

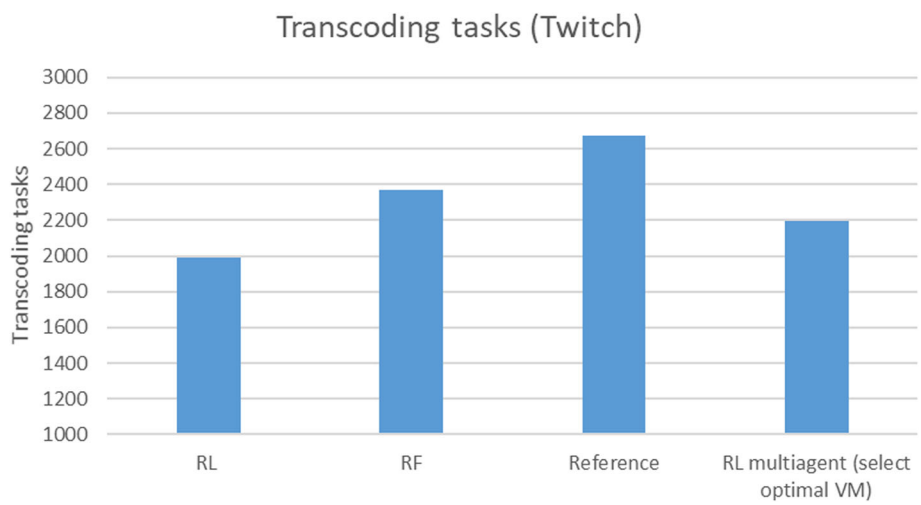

Fig. 19 Transcoding task count, when video is transcoded from Twitch

In overall, it can be concluded that SGD achieved a good precision, but a low number of realised transcoding tasks, when compared to RF or RL. RF had in almost all cases the best performance in either transcoding speed or CPU consumption prediction. RF was also easier to deploy than RL. Only, when prediction was started without any data (cold start) or transcoding speed and CPU consumption was predicted simultaneously, RL achieved in some cases a larger amount of realised transcoding tasks. Additionally, a multi agent approach with RL may improve transcoding performance.

Appendix

Big data view

Figure 20 presents the big data view of the architecture. The view has been created based on a published big data reference architecture [60], where functionality is described with rectangles, data stores with ellipsis, and data flows with arrows. Similar functionalities have been mapped into functional areas. Node exporter extracts resource consumption data periodically from the nodes (VMs), and raw measurement data is stored to Prometheus. The Collector extracts transcoding statistics from

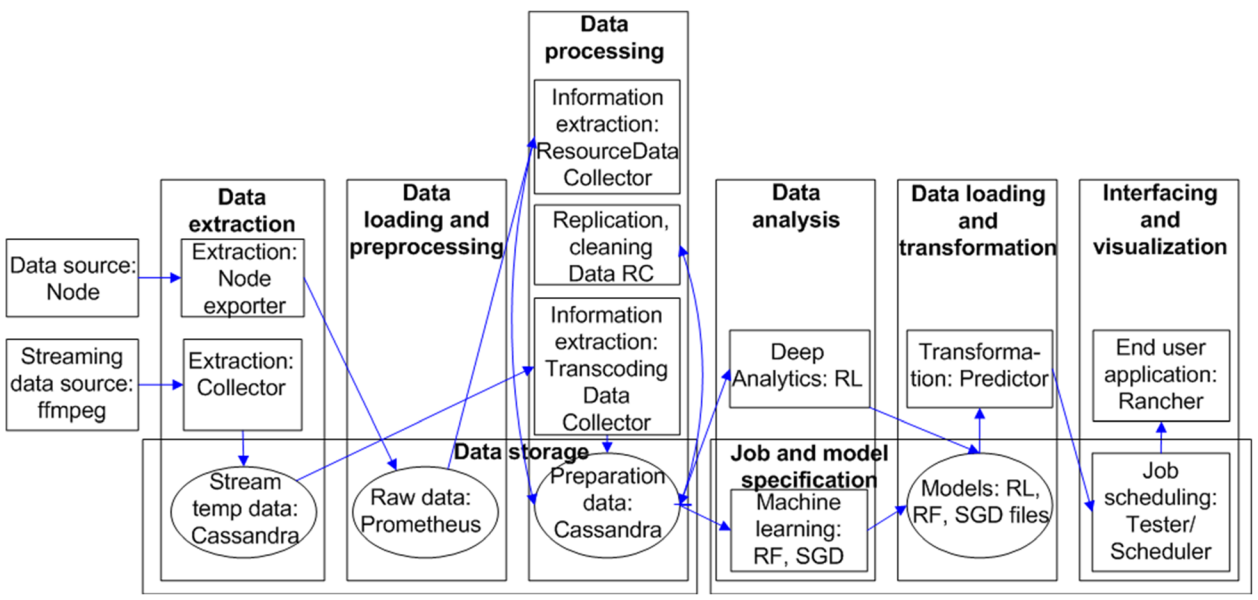

Fig. 20 Big data view of the architecture

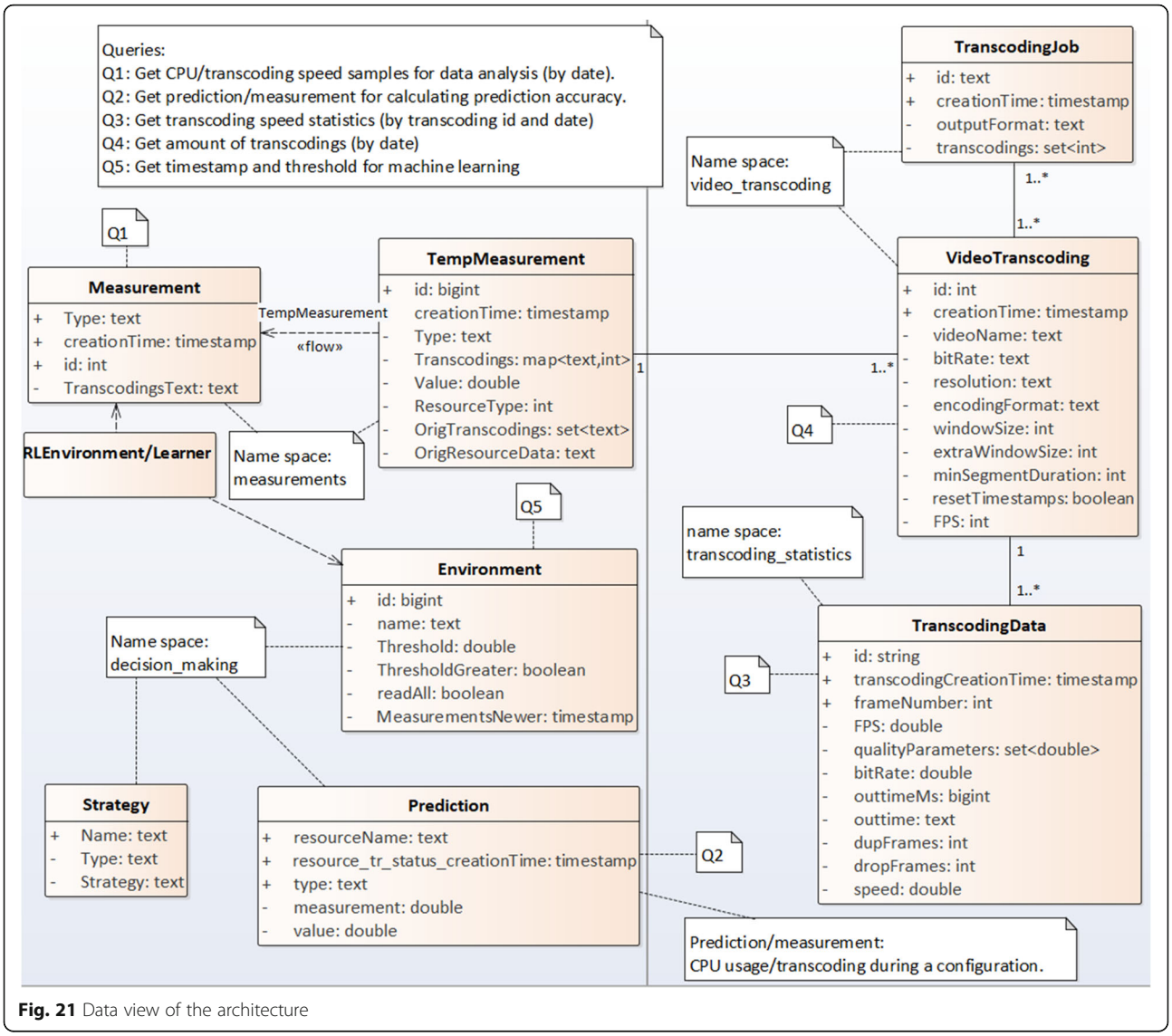

Fig. 21 Data view of the architecture

FFmpeg as a stream, which is stored temporarily into Cassandra (TranscodingData in Fig. 21). New information (average speed, transcoding configuration, VM type) is extracted from the temporary transcoding data, and the information is stored into a Preparation data store (TempMeasurement in Fig. 21), which holds processed data. ResourceDataCollector queries raw CPU consumption data from Prometheus, and extracts it into the Preparation data store (average CPU consumption, transcoding configuration, VM type). Data RC reads contents of the Preparation data store (TempMeasurement), cleans and replicates it for machine learning purposes (Measurement in Fig. 21). Data is analysed with machine learning techniques (RF, SGD) and Deep analytics (RL), and new data models are created for providing predictions. The predictor transforms predictions into decisions regarding a suitable VM for live video transcoding, which is utilised by the transcoding scheduler. Finally, the transcoding process can be visualised in the end user application (Rancher's UI).

Data view

Figure 21 presents data view of the architecture. The data structures indicate how data was stored into Cassandra. TranscodingJob contained information of video transcoding jobs, which may consist of multiple VideoTranscodings. VideoTranscoding stored configuration information of a transcoding task. TranscodingData stored statistics, which were collected during a transcoding from FFmpeg. TempMeasurement stored transcoding speed or CPU consumption, and associated configurations of transcoding tasks in a VM. Data was periodically transferred into a permanent store (Measurement) for machine learning. Environment contained information related to the

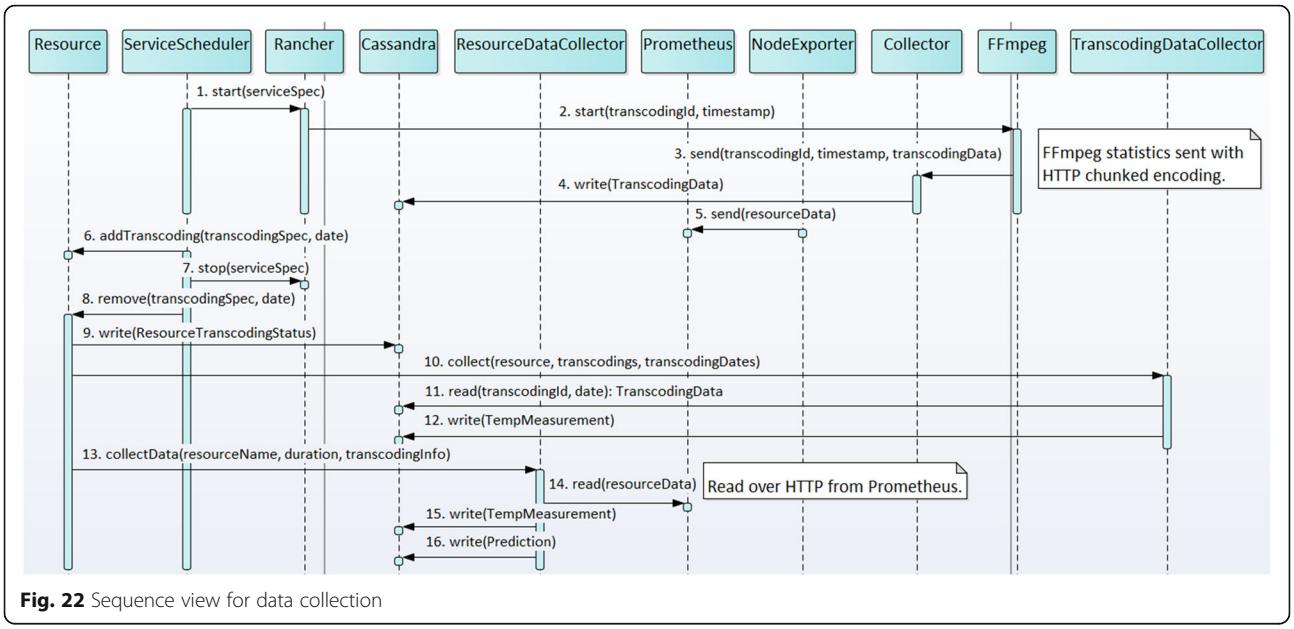

Fig. 22 Sequence view for data collection

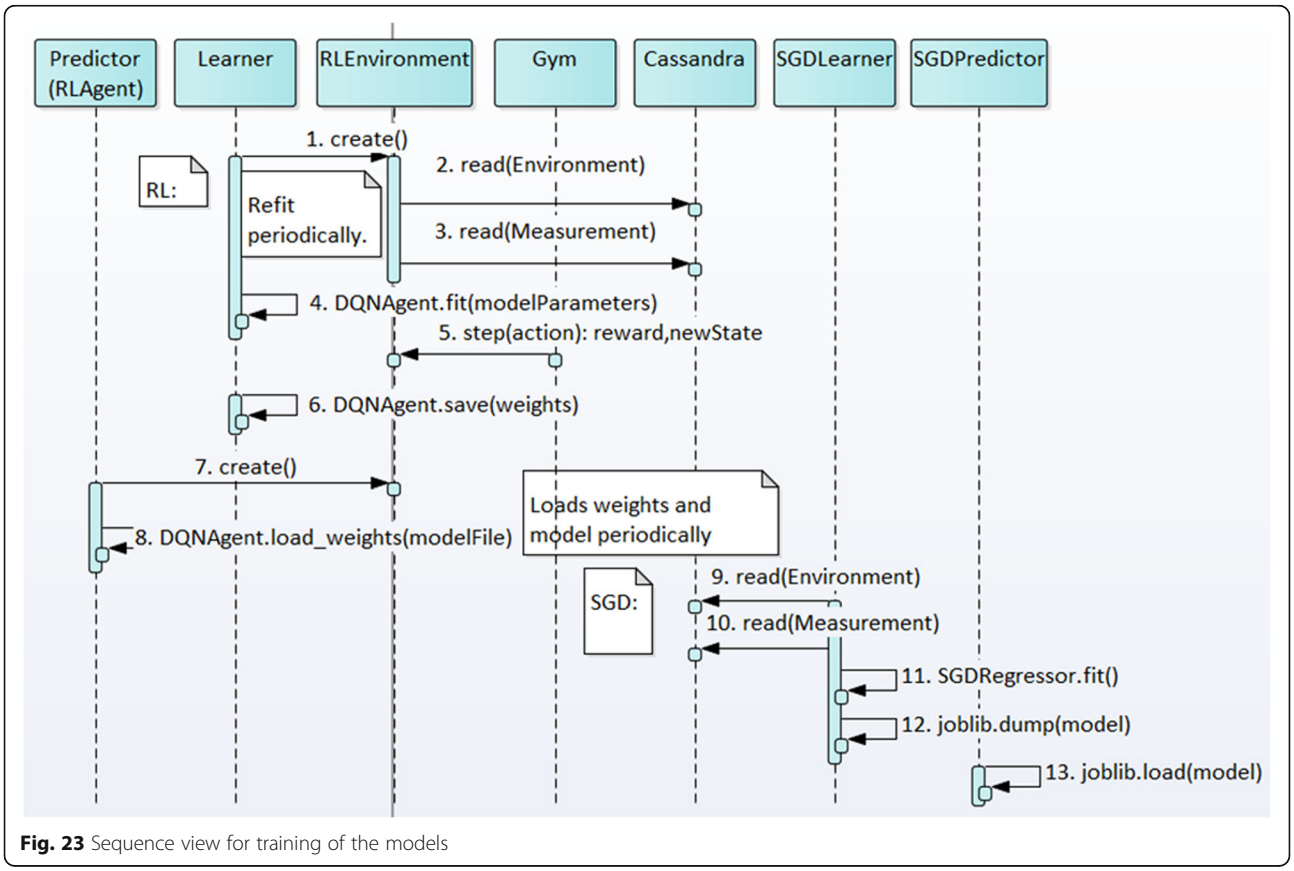

Fig. 23 Sequence view for training of the models

creation of models based on machine learning. It indicated the range of data to be utilised for training, and the goal/threshold for creating a model based on RL. Prediction was used for storing predictions created with SGD/RF, which were matched with measurements. Strategy contained high-level goals for transcoding or VM usage.

Sequence views

Figure 22 presents sequence view for data collection. The steps are as follows:

Step 1: ServiceScheduler starts a new video transcoding task based on the service specification. The service is created to the VM with Rancher CLI.

Steps 2–3: Timestamp and identifier of the transcoding are provided as parameters, when the transcoding process is started. The parameters are transferred with FFmpeg progress information to the Collector.

Step 4: The Collector accepts a HTTP chunked encoding stream connection from each FFmpeg process, and saves the received data (timestamp, transcoding identifier, and progress information) into Cassandra. Timestamp and transcoding identifier are used later (steps 10–12), when transcoding statistics are collected.

Step 5: Resource consumption information is periodically (every 5 s) scraped from Node exporters, which are executed on each transcoding node.

Step 6: The transcoding is added to the Resource. Subsequently, steps 9–16 are executed.

Step 7–8: Transcoding is stopped, which is indicated to the Resource. Date is used for mapping to the Transcoding-object.

Step 9: Transcoding status on the resource is saved into the database.

Step 10–12: Statistics are collected for transcoding tasks, which are executed on the resource. Transcoding statistics are read from Cassandra, and the lowest average transcoding speed is saved (to TempMeasurement).

Step 13–14: Resource consumption information is collected from Prometheus.

Steps 15–16: Average CPU consumption on the resource is saved into measurements together with transcoding configuration information. Also, the measured average CPU consumption is updated to the related prediction, when SGD/RF was used for predicting CPU consumption.

Figure 23 presents sequence view for training of the models. The steps are as follows:

RL:

Step 1: Learner creates a new RL environment (Gym).
Steps 2–3: RLEnvironment reads configuration information from the Environment-table, and transcoding related data from the Measurement-table.
Step 4: A DQNAgent is created and fitted. A neural network is trained.
Step 5: Step-function is called by the Gym-environment for getting reward related to a particular action. Gym-environment stores state, which contains count of transcoding tasks on VMs. The reward is created based on how well the target is achieved (e.g. transcoding speed) in a specific future state. The reward is calculated based on the measurements, which were read earlier (Step 3).
Step 6: The weights of the model are saved into a file.
Steps 7–8: A predictor is started, which creates a new RLEnvironment, and reads the weights from the saved model.

SGD:

Step 9–10: SGDRegressor reads configuration information from the Environment-table, and transcoding related data from the Measurement-table.
Step 11: The model is created based on all measurement samples.
Step 12: The final model is saved into a file.
Step 13: SGDPredictor loads the saved model for providing predictions.

Abbreviations

ABR: Adaptive Bit Rate; API: Application Programming Interface; CDN: Content Delivery Network; CLI: Command Line Interface; CPU: Central Processing Unit; CV: Coefficient of Variation; DASH: Dynamic Adaptive Streaming over HTTP; DQN: Deep-Q-Network; FPS: Frames Per Second; GPU: Graphical Processing Unit; HEVC: High Efficiency Video Coding; HLS: HTTP Live Streaming; HTTP: Hypertext Transfer Protocol; kbps: kilobits per second; MAPE: Mean Absolute Percentage Error; MPD: Media Presentation Description; MPEG: Moving Picture Experts Group; QoS: Quality of Service; REST: Representational State Transfer; RF: Random Forest; RL: Reinforcement learning; RQ: Research question; SGD: Stochastic Gradient Descend; UML: Unified Modelling Language; VM: Virtual machine; VoD: Video on Demand; YAML: Yet Another Markup Language

Acknowledgements

This research was conducted in CELTIC-Plus VIRTUOSE-project.

Authors' contributions

PP designed the research in collaboration with AH and TA. PP performed the literature review, designed and implemented the prototype, executed the experiments, analysed the results, and wrote the article. AH and TA contributed to the review of the article, and provided feedback. All authors read, and accepted the final version of the article.

Funding

This research has been partially funded by Business Finland and VTT Technical Research Centre of Finland.

Availability of data and materials

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Received: 1 February 2019 Accepted: 20 June 2019
Published online: 09 July 2019

References

1. Encoding.com (2019). Live event streaming. https://www.encoding.com/http-live-streaming-hls/. Accessed 24 Jan 2019

2. Wowza Media System (2019). Live Video Streaming. https://www.wowza.com/live-video-streaming. Accessed 24 Jan 2019.
3. Bitmovin (2019). Bitmovin video encoding. https://bitmovin.com/encoding-service/. Accessed 24 Jan 2019.
4. Gao G, Hu H, Wen Y (2016) Resource provisioning and profit maximization for transcoding in clouds: a two-timescale approach. IEEE T on Multimedia 19:836–848. https://doi.org/10.1109/TMM.2016.2635019
5. Li X, Salehi MA, Bayoumi M (2016) VLSC: video live streaming using cloud services. In: Proceedings of the IEEE international conferences on big data and cloud computing, social computing and networking, sustainable computing and communications. IEEE, Piscataway, 8–10 October 2016
6. Pääkkönen P, Heikkinen A, Aihkisalo T (2018) Architecture for predicting live video transcoding performance on Docker containers. In: IEEE international conference on services computing. San Francisco, Piscataway, 2-7 July 2018.
7. Aparicio-Pardo R, Blanc A, Pires K, Simon G (2015) Transcoding live adaptive video streams at a massive scale in the cloud. In: Proceedings of the 6th ACM multimedia Systems conference. ACM, New York, 18–20 March 2015
8. Wei L, Cai J (2016) QoS-aware resource allocation for video transcoding in clouds. T on circuits and Syst for video tech 27:49–61. https://doi.org/10.1109/TCSVT.2016.2589621
9. Gao G, Wen Y, Westphal C (2018) Dynamic priority-based resource provisioning for video transcoding with heterogeneous QoS. Transactions on circuits and Systems for Video Technology. https://doi.org/10.1109/TCSVT.2018.2840351
10. Costero L, Iranfar A, Zapater M, Igual FD, Olcoz K, Atienza D (2019) MAMUT: multi-agent reinforcement learning for efficient real-time multi-user video transcoding. Paper presented at the design, automation, and test in Europe, Florence, Italy, 25–29 March 2019.
11. Datadog (2019) 8 surprising facts about real Docker adoption. https://www.datadoghq.com/docker-adoption/. Accessed 24 Jan 2019
12. Peinl R, Holzschuher F, Pfitzer F (2016) Docker cluster Management for the Cloud – survey results and own solution. J Grid Comput 14:265–282. https://doi.org/10.1007/s10723-016-9366-y
13. Rancher Labs (2019) Rancher 2.0 documentation. https://rancher.com/docs/rancher/v2.x/en/. Accessed 24 Jan 2019
14. Docker Docs (2019) Docker compose file reference. https://docs.docker.com/compose/compose-file/. Accessed 24 Jan 2019
15. Helm (2019). Helm charts. https://docs.helm.sh/developing_charts/. Accessed 24 Jan 2019
16. Prometheus (2019) Prometheus. https://prometheus.io/. Accessed 24 Jan 2019
17. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, Omega, and Kubernetes. Queue - Containers 14:70–93. https://doi.org/10.1145/2898442.2898444
18. Medel V, Tolon C, Arronategui U, Tolosana-Calasanz R, Banares JA, Rana OF (2017) Client-side Scheduling Based on Application Characterization on Kubernetes. In: Pham C (ed) Economics of Grids, Clouds, Systems, and Services. Lecture notes in computer science, vol 10537. Springer, Cham
19. Heidari P, Lemieux Y, Shami A (2016) QoS assurance with light virtualization - a survey. In: Proceedings of the IEEE international conference on cloud computing technology and science, Luxembourg City. IEEE, Piscataway, 12–15 Dec 2016.
20. Pires K, Simon G (2014) DASH in Twitch: Adaptive Bitrate Streaming in Live Game Streaming Platforms. In: Hassan M (ed) Proceedings of the Workshop on Design, Quality and Deployment of Adaptive Video Streaming. ACM, New York, 2 Dec 2014.
21. Dutta S, Taleb T, Ksentini A QoE-aware elasticity support in cloud-native 5G Systems. In: Proceedings of the IEEE international conference on communications. IEEE, Piscataway, pp 22–27 May 2016
22. Dutta S, Taleb T, Frangoudis PA, Ksentini A (2016) On-the-fly QoE-aware transcoding in the Mobile edge. In: Proceedings of the IEEE global communications conference. IEEE, Piscataway, 4–8 Dec 2016
23. Chang ZH, Jong BF, Wong WJ, Wong MLD (2016) Distributed video transcoding on a heterogeneous computing platform. In: Proceedings of the IEEE Asia Pacific conference on circuits and Systems. IEEE, Piscataway, pp 25–28 Oct 2016
24. Gao G, Wen Y (2016) Morph: a fast and scalable cloud transcoding System. In: Proceedings of the ACM on multimedia conference. ACM, New York, 15–19 Oct 2016
25. Li X, Salehi MA, Bayoumi M, Tzeng N, Buyya R (2018) Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services. IEEE T Parall Distr 29:556–571. https://doi.org/10.1109/TPDS.2017.2766069
26. Darwich M, Beyazit E, Salehi MA, Bayoumi M (2017) Cost efficient repository Management for Cloud-Based on-Demand Video Streaming. In: Proceedings of the 5th IEEE international conference on Mobile cloud computing, services, and engineering. IEEE, Piscataway, pp 6–8 April 2016
27. Chen K, Chang H (2017) Complexity of cloud-based transcoding platform for scalable and effective video streaming services. Multimed Tools Appl 76:19557–19574. https://doi.org/10.1007/s11042-016-3247-z
28. Benkacem I, Taleb T, Bagaa M, Flinck H (2018) Performance benchmark of transcoding as a virtual network function in CDN as a service slicing. In: Proceedings of the IEEE wireless communications and networking conference. IEEE, Piscataway, 15–18 April 2018
29. Lottarini A, Ramirez A, Coburn J, Kim MA, Ranganathan P, Stodolsky D, Wachsler M (2018) Vbench: benchmarking video transcoding in the cloud. In: Proceedings of the twenty-third international conference on architectural support for programming languages and operating Systems. ACM, New York, pp 24–28 March 2018
30. Iqbal Q, Dailey MN, Carrera D, Janecek P (2011) Adaptive resource provisioning for read intensive multi-tier applications. Future Gener Comp Sy 27:871–879. https://doi.org/10.1016/j.future.2010.10.016
31. Islam S, Keung J, Lee K, Liu A (2011) Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener Comp Sy 28:155–162. https://doi.org/10.1016/j.future.2011.05.027.
32. Samreen F, Elkhatib Y, Rowe M, Blair GS (2016) Daleel: simplifying cloud instance selection using machine learning. In: Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey, pp 25–29 April 2016
33. Gong Z, Gu X, Wilkes J (2010) PRESS: predictive elastic ReSource scaling for cloud systems. In: Proceedings of the international conference on network and service management. IEEE, Piscataway, pp 25–29 Oct 2010
34. Shyam GK, Manvi SS (2016) Virtual resource prediction in cloud environment: a Bayesian approach. J Netw Comput Appl 65:144–154. https://doi.org/10.1016/j.jnca.2016.03.002
35. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) A brief survey of deep reinforcement learning. In: IEEE Signal Proc Mag, vol 34, pp 26–38. https://doi.org/10.1109/MSP.2017.2743240
36. Waschneck B, Reishstaller A, Belzner L, Altenmuller T, Bauernhansl T, Knapp A, Kyek A (2018) Optimization of global production scheduling with deep reinforcement learning. Procedia CIRP 72:1264–1269. https://doi.org/10.1016/j.procir.2018.03.212
37. Verikas A, Gelzinis A, Bacauskiene M (2011) Mining data with random forests: a survey and results of new tests. Pattern Recogn 44:330–349. https://doi.org/10.1016/j.patcog.2010.08.011
38. Genuer R, Poggi J, Tuleau-Malot C, Villa-Vialaneix N (2017) Random Forests for Big Data. Big Data Res 9:28–46. https://doi.org/10.1016/j.bdr.2017.07.003.
39. NGINX (2019) NGINX. https://www.nginx.com/. Accessed 24 Jan 2019
40. FFmpeg (2019) FFmpeg. https://www.ffmpeg.org/. Accessed 24 Jan 2019
41. The Apache Software Foundarion (2019) Apache Cassandra. http://cassandra.apache.org/. Accessed 24 Jan 2019
42. AppScale Systems (2019) Eucalyptus. https://www.eucalyptus.cloud/. Accessed 12 Apr 2019
43. Docker (2019) Docker. https://www.docker.com/. Accessed 24 Jan 2019
44. Grafana Labs (2019) Grafana. https://grafana.com/. Accessed 24 Jan 2019
45. scikit-learn (2019) scikit-learn. https://scikit-learn.org/stable/. Accessed 24 Jan 2019
46. numpy (2019) NumPy. http://www.numpy.org/. Accessed 24 Jan 2019.
47. OpenAI (2019) Gym. https://gym.openai.com/docs/. Accessed 24 Jan 2018.
48. Github (2019) Keras: the Python deep learning library. https://keras.io/. Accessed 24 Jan 2019
49. Tensorflow (2019) Tensorflow. https://www.tensorflow.org/. Accessed 24 Jan 2019.
50. Github (2019) Keras-RL. https://github.com/keras-rl/keras-rl/. Accessed 24 Jan 2018
51. Flask (2019) Flask. http://flask.pocoo.org/. Accessed 24 Jan 2019.
52. Docker Hub (2019) JRottenberg FFmpeg image. https://hub.docker.com/r/jrottenberg/ffmpeg/. Accessed 24 Jan 2019.
53. ISO/IEC 23009–1:2014 (2014) Information technology - Dynamic adaptive streaming over HTTP (DASH) Part 1: Media presentation description and segment formats
54. Refaeilzadeh P, Tang L (2009) Liu H (2009) cross-validation. In: Liu L, Özsu MT (eds) Encyclopedia of database Systems. Springer, Boston, MA

55. Twitch (2019) Twitch API v5. https://dev.twitch.tv/docs/v5/. Accessed 12 Apr 2019
56. Streamlink (2019) Streamlink https://streamlink.github.io/. Accessed 12 Apr 2019
57. Zaitsev P (2018) Prometheus 2 Time Series Performance Analyses https://www.percona.com/blog/2018/09/20/prometheus-2-times-series-storage-performance-analyses/. Accessed 24 Jan 2019
58. Bitbucket (2019). Bitbucket. https://bitbucket.org/. Accessed 24 Jan 2019
59. Docker Hub (2019) Docker Hub. https://hub.docker.com/. Accessed 24 Jan 2019
60. Pääkkönen P, Pakkala D (2016) Reference architecture and classification of technologies, products, and Services for big Data Systems. Big Data Res 4: 166–186. https://doi.org/10.1016/j.bdr.2015.01.001

Publisher's Note