Regression Training using Model Parallelism in a Distributed Cloud

Regression Training using Model Parallelism in a Distributed Cloud

Joel Reijonen*, Miljenko Opsenica*, Roberto Morabito*, Miika Komu*, Mohammed Elmusrati$^{\Psi}$
**Ericsson Research*, Jorvas Finland – $^{\Psi}$University of Vaasa, Vaasa Finland
*{firstname.surname@ericsson.com}, $^{\Psi}$mohammed.elmusrati@uva.fi

Abstract— **Machine learning requires a relevant amount of computational resources and it is usually executed in high-capacity centralized cloud infrastructures (e.g., data centers). In such infrastructures, resources are shared in a scalable manner through instantiation and orchestration of multiple virtualized services. Emerging trends in machine learning are distribution and parallelization of model training, which allows the execution of model training tasks in multiple distributed computational domains, with the aim of reducing the overall training time. A possible drawback in decentralization of machine learning is that performance latency issues may arise when the computation of training is geographically distributed to nodes with long distance from each other. One way to reduce latency is to utilize edge computing infrastructure, i.e., to distribute computation near the origin of the request. As edge resources can be scarce, it is important to orchestrate the model training in a parallelized manner. To this extent, in order to effectively ease the use of parallelization both in centralized and in distributed scenarios, we propose and implement a concept that we refer to Intelligent Agent (IA). An IA is responsible for instantiating and scheduling of the machine learning tasks (e.g., model training), and deriving inferences. In our solution, model training is distributed to multiple IAs in parallel. Each IA is packaged into a Linux container in order to take advantage of container portability across heterogenous deployments and to reuse existing container orchestration tools. We validate our proposal by deploying and instantiating multiple IAs across a distributed cloud environment, where each IA is accounting for a fixed amount of computational resources.**

Keywords—Intelligent agent, Model parallelism, Regression training, Intelligent cloud

I. INTRODUCTION

Cloud computing has gone through significant changes in the last decades. We have seen the transition of computing from in-house managed servers to virtualized cloud infrastructures. Application cloud deployment models have transitioned from the monolithic to microservices, and from hosted services to cloud native ones [6]. Virtualization models have also been evolving from heavier Virtual Machines (VMs) to lighter Linux containers and recently to even lighter models provided by serverless computing [18]. In general, the evolution of software virtualization has resulted in more elastic cloud offering models, enabling better scalability and flexibility of on-demand resources. These trends provide new opportunities to abstract shared pools of resources but also introduce new challenges in cloud management and optimization algorithms [19]. New generations of cloud-native applications are more flexible and elastic to deploy but their structure is often more complex due to dependencies in workloads, and due to complexity of distributed applications and their interconnectivity. Automation of application and infrastructure management are ways to cope with such growing complexity [12].

One prominent cloud technology trend is to move cloud computing from centralized clouds to the edge, closer to the end-users and real-time data sources [13]. The main advantages of an edge cloud include lower latency and improved privacy due to local processing. One disadvantage of the edge cloud is the extra management cost and data synchronization with the central cloud. Particularly challenging are scenarios with the slow and unreliable network connectivity between the edge and central cloud. This requires that the edge cloud manages local operations more autonomously by taking advantage in local cloud orchestrator. Additionally, connectivity can be improved with the new 5G wireless connectivity technology providing more reliable connectivity with the ultra-low latency. Another disadvantage is that the edge cloud can be more resource constrained, requiring rather lightweight computation locally while offloading heavier workloads to the central cloud. In the latter case, the resources of multiple edge clouds could be pooled together to be utilized in parallel. In this way, edge clouds can avoid offloading processes to the centralized clouds and minimize the impact of the latency.

To enhance cloud computing efficiency even beyond existing optimization algorithms, machine learning is a widely adopted technology in both research and industry. Machine learning can extract hidden information from processed data, and it enables continuous improvements to the optimization models which can be trained by test samples or by learning based on previous experience [7, 8]. Machine learning can require massive amounts of computational resources which is a problem especially for resource constrained edge clouds that may not have a reliable connectivity to the central cloud. One way to mitigate around this limitation is to utilize resources from other edge clouds.

In this paper, we propose a solution to decrease the magnitude of resource demanding machine learning in the edge cloud by utilizing intelligent agents. As a machine learning technique, agents train a regression model that maps the input data to the corresponding outputs. We propose an architecture for model training that supports the agents to take advantage in model parallelism. In addition to model training, the intelligent agents handle the distribution of computation, evaluation of the trained model and communication between each other. We measure the performance of these agents in a distributed cloud setting. Finally, we analyze the measurements and discuss the performance of regression model training that the agents perform in parallel.

The rest of this paper is organized as follows. Section II provides background information for familiarizing with the research area of this paper. In Section III, we provide a detailed description of the system architecture, while Section IV focuses on describing the experimental setup. Section V presents the results of the performance evaluation of our implementation and Section VI describes related research contributions. Finally, conclusions and future work are described in Section VII.

II. Enabling tehcnologies and methods

A. Intelligent agent

With the current trend of shifting applications and services closer to the edge, there are growing requirements on optimization where machine learning can play a major role. However, computational resources are often constrained in the edge whereas machine learning is rather resource consuming. One approach to overcome this problem is to take advantage of distributing the computation of machine learning, and divide of computational load among multiple domains by utilizing intelligent agents.

The term *intelligent agent* refers to a popular concept in both Artificial Intelligence (AI) and software engineering. In the field of agent-based community, the term *intelligent agent* has been problematic since the term is widely used among the community, but a universally accepted definition is not acknowledged [1]. In this paper, we define intelligent agent to refer to an independent software that follows the principles of a weak notion of agency as defined in [1]:

- **Autonomy.** Agents have autonomy in their actions and possible states in such a way that direct human intervention is not required [1, 3].
- **Social ability.** Agents are able to establish communication with other agents via commonly utilized protocols [1, 2].
- **Reactivity.** Agents are able to interact with their deployment environment e.g. cloud [1].
- **Pro-activeness.** Agents may take the initiative when it comes to the actions in their environment i.e. agents are able to take goal driven actions without being triggered by the events of the environment [1].

Moreover, the term *intelligent agent* has also a stronger notion that appends more human-like features to the weak notion of agency that was listed above. Stronger notion of agency introduces features such as knowledge, intention and sense of free will [1, 4].

B. Regression

In machine learning, regression refers to a supervised mathematical procedure that estimates the relations between variables of interest by constructing a model [5, 8]. Construction of the model is based on the development of mathematical expressions that represent the pattern in the relation between dependent and independent variables as precisely as possible. The constructed model includes parameters that are initially unknown, constant coefficients that determine the behavior of the model. In addition to the parameters, a model has variety both in its degree and its mathematical complexity which are both dependent on the modeled mathematical operation [5]. Regression based models are used to approximate and predict the output values of certain inputs that may not even be represented in the given data.

Regression has no predefined information about the degree and the parameters of the model in the context of this paper. Therefore, the main objective of intelligent agents is to learn the parameters and the degree of the best fitting regression model. In [5], the formula for regression model of higher-order polynomials has been defined in the following form:

$$Y_i = a_0 + a_1 x_i + a_2 {x_i}^2 + a_3 {x_i}^3 + \cdots + a_p {x_i}^p + E_i$$

where Y represents the dependent variable, x represents the independent variable, coefficients $a_0, a_1, \dots a_p$ are the parameters of the model, p is the number of terms, E represents the added random error and subscript i represents the observation unit where i = 1, 2, 3 ... n.

As stated in [5], the model usually falls into the class of models that are linear in the parameters in preliminary studies of the operation, but the more realistic models are often nonlinear in the parameters. For this reason, we expand the intelligent agents to consider also nonlinear regression models, in training, even though linear models have reduced complexity compared to nonlinear models.

C. Least squares estimation

The evaluation of the trained regression models is an essential part in order to derive the model with the best fit. There are various metrics that can be used to evaluate machine learning models such as mean squared error, mean absolute error, classification accuracy, logarithmic loss, etc. In this paper, we evaluate the models by utilizing least squares estimation (LSQ). The least squares estimation is an applicable option in the evaluation of regression models since these models result numerical output values.

The least squares estimation is a method where the trained models are evaluated by comparing sums of squared deviations between the real measurements and the estimations of the models. In the least squares estimation, the model with the best fit is the model that results the smallest sum of squared deviations [5].

In this paper, intelligent agents evaluate and select the best fitting regression models by utilizing the principles of the least squares estimation. The following formula for least squares estimation can be used by calculating the sum of the squares of the residuals – $SS(Res)$ [5]:

$$SS(Res) = \sum_{i=1}^{n} (Y_{ri} - Y_{ei})^2,$$

where the Yri is the real observation, the Yei is the estimation of the model and $Yri - Yei$ is the residual.

D. Parallel machine learning

Parallel machine learning is a paradigm where multiple workers train parts of a complete model [9]. Workers are entities that handle computational operations and they have allocated resources. In context of this paper, intelligent agents are regarded as workers. We review two popular strategies in parallelization of machine learning that are data parallelism and model parallelism [9].

In data parallelism, model training is parallelized across the data dimension which refers to a parallelization where the data is split into subsets and each subset is used to train a model by multiple workers [9-11]. In this approach, the workers train a model with the same architecture and, after training, the parameters of the model are synchronized in order to compose the complete model [9, 10]. Data parallelism is an efficient and scalable strategy to be considered especially if the computational load per weight is high since the weights are the units in synchronization [9].

Respectively, model training is parallelized across the model dimension where the model is split among the workers in model parallelism. Each worker trains a different part of the model by utilizing the same data, and the workers are responsible for synchronization of the intermediate results when the input of a certain part of the model is the output of another part of the model [11]. These split parts of the model can be referred as computational nodes, and the model parallelism is an efficient strategy when the computational load per node is high because the nodes are the units in synchronization [9]. Generally, the performance of model parallelism is inferior to data parallelism since the model parallelism introduces higher latency due to increased synchronization expenses [11].

III. System architecture

We propose a solution for regression training that utilizes model parallelism as a parallelization strategy. This solution aims to support machine learning that is executed in the proximity of the source of the data which is especially useful in the context of edge clouds. In our solution, we utilize intelligent agents that are responsible for the management of machine learning activities and handling agent-to-agent communications. Agents are deployed in a distributed edge cloud where they utilize the allocated computational resources for machine learning. To the best of our knowledge, this is the very first contribution that, through intelligent agents, tries to shed light on parallelized regression training, by considering also the trade-off between performance and resource usage when training is parallelized among multiple agents.

In this paper, we parallelize the training of a regression model that maps the dependent variable to the independent variable as precisely as possible. We propose an architecture for regression in Figure 1.

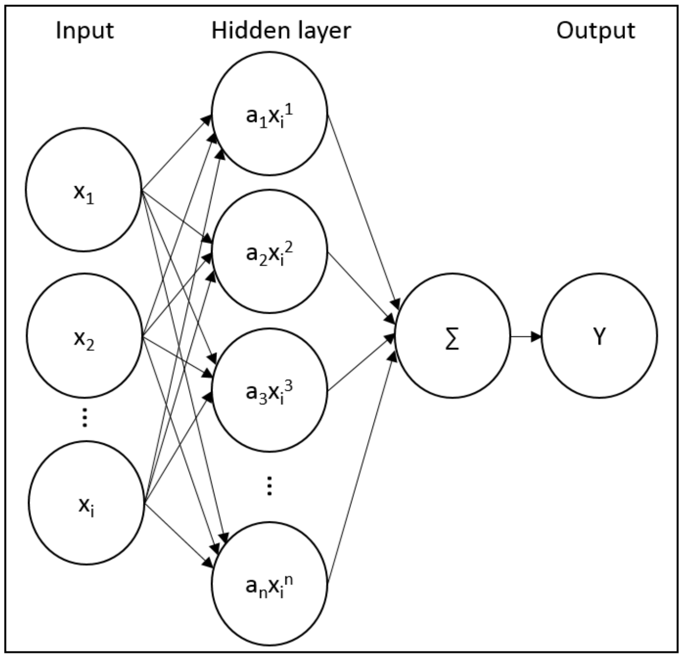

Fig. 1. Proposed functional architecture for regression based machine learning

The proposed architecture introduces visible input and output layers that are exploited in the environment. Moreover, we define a hidden layer where the inputs are weighted by the coefficient a_n and the exponent of the input increases as the number of computational nodes increases. In this architecture, machine learning is responsible of a weight optimization, establishment of synapses and finding out the optimal number of computational nodes.

In machine learning, the evaluation of the used algorithm is an essential factor as, without evaluation, the outcome may introduce undesirable results such as low accuracy, overfitting or false positives. In this paper, we propose least squares estimation (LSQ) as the evaluation metric. Least squares estimation is an efficient evaluation metric since the sum of squared deviations can be derived from the remainder of predictions of the regression model and corresponding measurements. Moreover, we define the best fitting regression model by selecting the regression model that produces the smallest sum of squared errors.

By combining the proposed regression architecture and evaluation strategy in model training, intelligent agent implements model training as described in Algorithm 1.

Algorithm 1 Regression model training

```
1   Read input data x and output data y
2   Define the highest degree n of the model
3   Split x to x_train and x_test
4   Split y to y_train and y_test
5   for i = 1 to n do
6     model_graph => $y = \sum_{k=1}^{n} {a_k}^k x$
7     model_graph.fit(x_train, y_train)
8     temp_loss = model_graph.eval(LSQ, x_test, y_test)
9     if temp_loss < loss then
10      loss = temp_loss
11      best_model = model_graph
12    end if
13  end for
14  return best_model, loss
```

In Algorithm1, the agent splits the given data into training and test sets. By splitting the data, the agent reduces the risk of overfitting the model. Overfitting is a phenomenon where the model corresponds very closely to the training data which again may lead to failures when fitting new data that has not been introduced in the training set. By utilizing training sets, the agent generates and fits the computational graph that describes the architecture of the model. After the graph is fitted, the agent evaluates the graph by utilizing least squares estimation and test sets. Based on the evaluation, the agent selects the graph that produces the smallest sum of squared error to be utilized as the regression model.

Since we propose an architecture where the inputs are weighed and increasing exponent, we propose model parallelism as the parallelization strategy. In the proposed architecture, computational load per node is relatively high, due to the increasing exponent, and therefore model parallelism is an efficient strategy [9]. Even though model parallelism is generally inferior to data parallelism, we have proposed an architecture with a single hidden layer, so the latency due to weight synchronization can be avoided.

In parallelization, we take advantage of intelligent agents. Agents act as workers that are deployed in different computational domains (Figure 2). These agents are autonomous software that are responsible for distributing model training among other agents, executing regression training (Algorithm 1) by utilizing model parallelism and handling agent-to-agent communication. We utilize container technologies in the deployment of the agents to facilitate portability in distributed edge clouds.

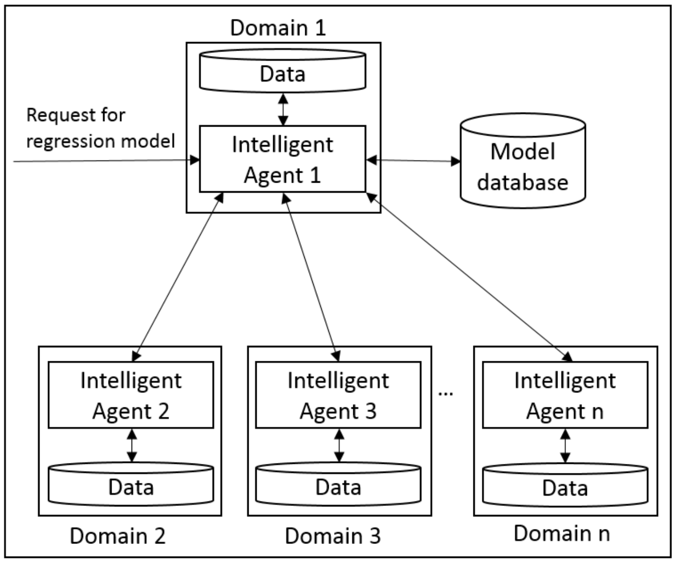

Fig. 2. Overview of the proposed intelligent agent based system

In the Figure 2, intelligent agents may request other agents to participate in decentralization of machine learning by training a certain part of the model. The requesting agent splits the model into parts among the participating agents in such a way that each agent incurs approximately the same computational load for model training. After the individual parts of the model are trained, the requesting agent gathers the parameters of the model and merges the complete model. Moreover, the requesting agent stores the complete model in a model database for long-term use.

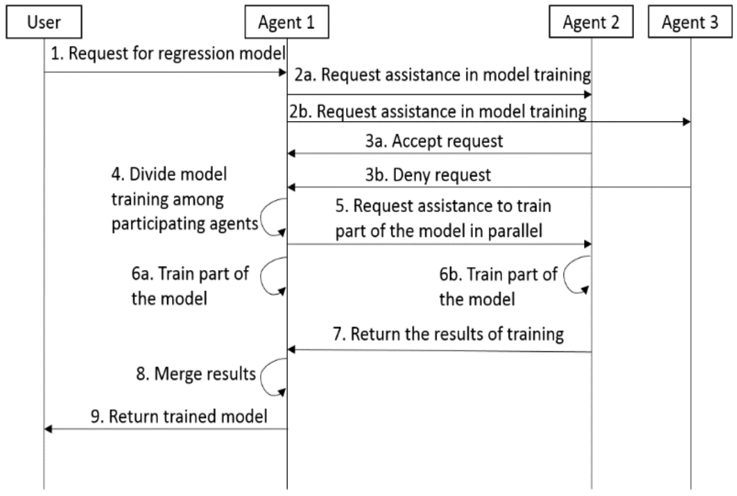
User
Agent 1
Agent 2
Agent 3
1. Request for regression model
2a. Request assistance in model training
2b. Request assistance in model training
3a. Accept request
3b. Deny request
4. Divide model training among participating agents
5. Request assistance to train part of the model in parallel
6a. Train part of the model
6b. Train part of the model
7. Return the results of training
8. Merge results
9. Return trained model

Fig. 3. Overview of the message sequence between user and intelligent agents

In the Figure 3, a user requests regression model from an intelligent agent. The agent requests assistance for model training from the other agents that are within a feasible range. Agents may accept or deny the request depending on the availability of their resources. Requesting agent divides the model training among the agents who accepted the request. When partitioning the model training, the agent utilizes model parallelism as a parallelization strategy so that each participating agent has a dedicated part of the model to train. After the agents have finalized model training, they send the trained parts of the model to the requesting agent. Finally, the requesting agent merges the results and returns trained model to the user.

IV. Experimental setup

We ran our experiments using eight KVM-based VMs (c1m1 image with Ubuntu 18.04, 1 VCPU, 1 GB RAM, 20 GB disk). The VMs were running on an OpenStack (Pike version) environment operated at Ericsson Research datacenter (ERDC) in Lund (Sweden). For each VM, we executed a testing procedure to ensure a homogeneous performance of the machines before we deployed intelligent agents. As a testing procedure, the agents performed a training of a simple linear regression. If the agent performed model training longer than expected, we instantiated a new VM to replace the overloaded VM.

The experiments included measuring the performance of regression model training. In experiments, the utilized data simulates the pattern of a single sin wave with random deviation that draws samples from a uniform distribution (Figure 4).

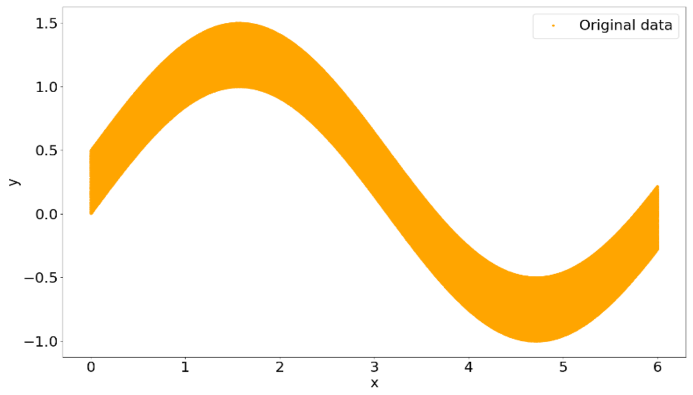

Fig. 4. Overview of the simulated data

The utilized data consists of million rows of two-dimensional data where the independent variable x varies between values 0.0 and 6.0, and dependent variable y varies between values of -1.0 and 1.5. Even though the dependent variable introduces both positive and negative values, least squares estimation, as an evaluation metric, is still valid in model training since the deviations are squared.

When intelligent agents train the regression models, the agents split the data into training and test sets. By splitting the data, the agents guarantee that the regression models are not overfitted. Overfitting is a rather common problem in machine learning, where the trained model 'memorizes' the pattern of the data instead of deriving a generalization. Hence, the effect of overfitting has a greater effect in scenarios that are not described in training, so utilization of overfitting model may lead to unpredictable results.

We experiment regression-based machine learning by utilizing up to eight participating agents that are responsible for performing model training in parallel. In this experiment, we measure the time that is consumed in model training, data reading, data partitioning, and request handling. Each measurement has been repeated ten times in order to reduce the impact of the background noise that may occur in the measurements.

V. Results and discussion

In this section, we describe the results of parallelized regression training using the proposed simulated data set and the ERDC as a test environment. These results provide insights about the trained regression model and the trends of the execution times. Measured execution times consist of four major workloads that are model training, data reading, data partitioning and agent-to-agent request handling.

After intelligent agent(s) have finalized regression model training, we receive a regression model that describes the pattern of the data. Figure 5 shows how the regression model fits the experimental data, with blue dots representing the training data and orange dots representing the test data.

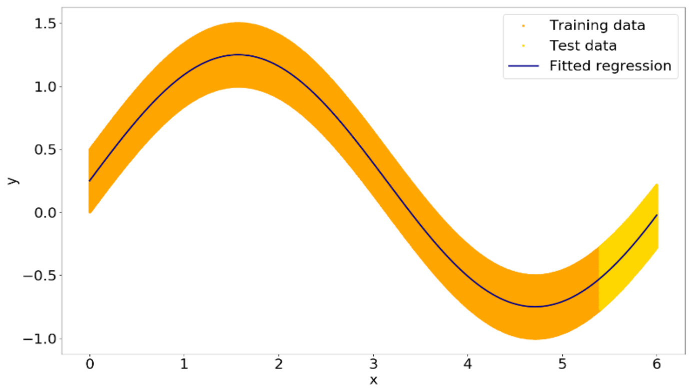

Fig. 5. Overview of the fitted regression model where the original data is split into training and test data sets

Figure 5 depicts how the trained regression model fits both training and test data. Even though the training data does not introduce the pattern that the test data set described, the trained model has learned the generalized pattern so the model also fits the test data. Based on the fit, we can state that the trained regression model does not introduce overfitting, and the model has a precise overall fit since the selected model also produces the smallest sum of squared deviations in least squares estimation.

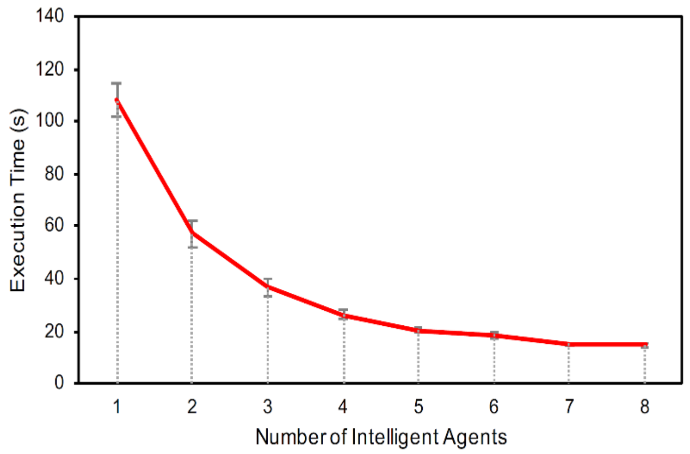

Fig. 6. Overall execution time of machine learning decreases as the number of intelligent agents increases

Figure 6 shows the overall processing time when a growing number of intelligent agent(s) is instantiated. Overall processing time refers to the period of time that is needed for the agent(s) to provide a trained regression model after they have been requested to train one. As the number of the participating agents increases, the overall processing time decreases. Moreover, the parallelization of model training has a greater amplitude in the left hand side of the graph. This phenomenon can be explained with certain fixed operations, such as data reading, that consume constant period of time when employing model parallelism. However, the model parallelism reduces the time consumption of model training which has high correlation to overall time consumption.

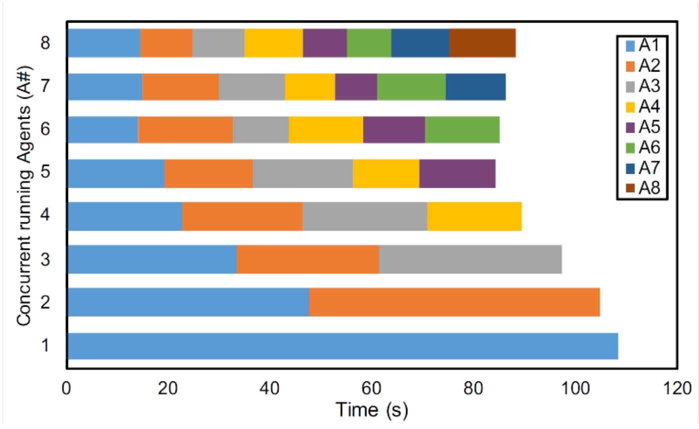

Fig. 7. Combined time consumption of each participating agent(s) is the total sum of time consumption that the agent(s) spend in machine learning

Figure 7 shows the combined time consumption of each participating agent in parallelized model training. Combined time consumption decreases as the number of participating agents increases from one to five. After having more than five participating agents, the combined time consumption starts to increase This increase can be explained by agents executing certain fixed operations that consume a constant period of time. Based on these observations, parallel model training achieves peak in performance while having five participating agents when it comes to the combined time consumption.

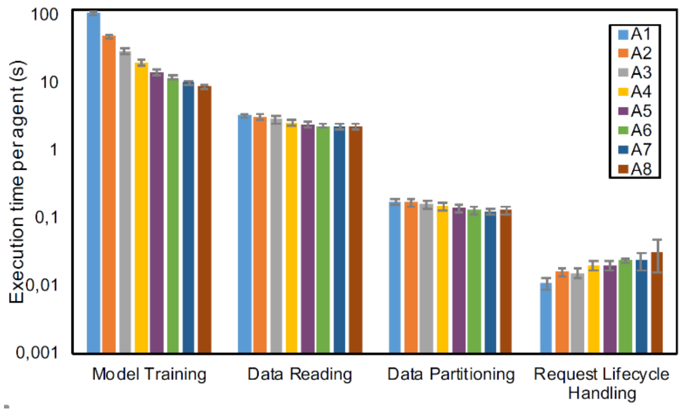

Fig. 8. Overview of the average workload execution time per participating agent(s)

Figure 8 shows the average workload execution time per participating agent(s). In Figure 8, we have improved visualization of the execution times by utilizing logarithmic scale in y axis. From these workloads, the execution time of model training decreases and request handling increases as the number of participating agents increases.

Based on these results, the determination of the optimal number of participating agents depends on the objective of the optimization. If the objective is to reduce computation time for machine learning, then the number of participating agents should be high. However, if the objective is to reduce cloud resource usage, then the most optimal performance can be achieved by employing five participating agents.

VI. Related work

Gupta & Pujari [15] proposed a multi-agent system where agents are responsible for handling different computational tasks in the context of medical diagnosis systems. In the proposed architecture, the agents – which are characterized by the definition of common attributes – execute specific tasks and share their results with other peer agents. Finally, the exchanged results are aggregated by a specific agent that generates a report based on the results.

Alessandrini et al. [16] proposed the integration of artificial intelligence (AI) into Service Oriented Architectures (SOA) by utilizing an agent-based approach. In such an implementation, agents handle the coupling of AI and SOA-based services in a flexible manner. The agents provide machine learning functionality (e.g., neural networks), and the results of the machine learning are utilized by the coupled service.

Abid et al. [17] introduced a method to deploy intelligent agents as a multi-agent system in a cloud environment. The agents execute intelligent services such as k-means clustering for data mining and web service searching. The use intelligent agents in cloud-based services allows to automate cloud service selection, due to clustering, as well as introducing several mutual advantages both for cloud service users and providers. These advantages enable evaluation and comparison of cloud based services by cloud service providers.

Meng et al. [23] presented an open source distributed machine learning library called MLlib. The library provides various features implementing different multiple machine learning techniques and pipelines in a distributed manner by taking advantage of Apache Spark. In large-scale learning, the library supports data and model parallelism strategies. The main difference is that the MLlib library takes advantage in cluster-based computing whereas our proposed solution utilizes agent-based computing. To be more precise, agent-based computing consists of autonomous agents that can accept or decline workloads depending on their resource utilization, whereas clusters operate as a single unit as enforced by the orchestrator.

In previous work [14], we proposed a cloud native multi-agent system where the agents provide machine learning as a service. In this work, the agents provided regression based optimization for a miniature version of an autonomous ship. The agents are deployed in both edge and central clouds: the heavy computation is offloaded to the central cloud while lightweight operations are executed on the edge (Figure 9).

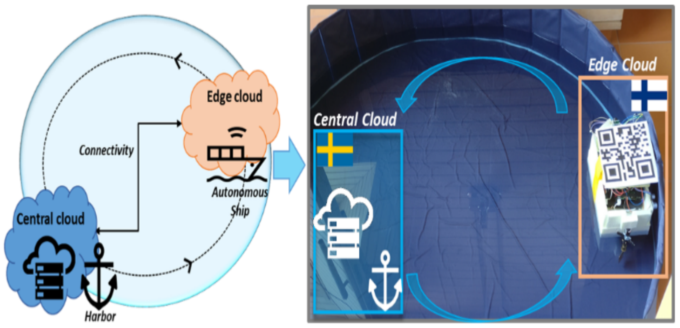

Fig. 9. Overview of prototype version of an autonomous ship that utilizes distributed cloud

The proposed multi-agent system in [14] has been demonstrated as a part of a national Finnish ecosystem called DIMECC. DIMECC is a co-creation ecosystem which consists of more than 2 000 research and development professionals who are working in more than 400 different organizations [22]. The contribution of this paper builds on and extends this work.

As highlighted from the literature review, agents and intelligent agents are a well-established concept that has been largely used in previous years. The context in which agents can be used can vary within a wide set of heterogenous scenarios.

VII. Conclusions and future work

Model parallelism is an efficient strategy to parallelize machine learning especially when the computational load is high on each node. In this paper, it has been demonstrated how to decrease the overall computation time in regression training by utilizing model parallelism. Moreover, parallelized training supports machine learning in a computationally constrained environments since the training of the model is split among the participating nodes. We implemented parallel regression training by utilizing autonomous software that we refer to as intelligent agents. The agents are responsible for distributing machine learning activities and establishing agent-to-agent communication.

Our measurements show that the parallelization decreases overall execution time as the number of participating agents increases. However, the summed execution time of the participating agent(s) does not follow a similar trend since its optimal performance occurs when training is parallelized among five participating agents. Therefore, determining the optimal number of participating agents depends on the objective of the optimization: to minimize the computation time for machine learning or to minimize cloud resource usage.

As future work, we propose an alternative implementation by utilizing state of the art frameworks such as TensorFlow, PyTorch or MXNet to train a regression-based model in a parallelized manner

References

[1] M. Wooldridge & N. R. Jennings, Intelligent agents: theory and practice, 1994, pp. 3–9

[2] M. R. Genesereth & S. P. Ketchpel, Software agents, 1994, pp. 46–55

[3] C. Castelfranchi, Guarantees for autonomy in cognitive agent architecture, 1994, pp. 55–71

[4] Y. Shoham, Agent-oriented programming, 1993, pp. 52–56

[5] J. O. Rawlings, S. G. Pantula & D. A. Dickey, Applied regression analysis: a research tool, 1988, pp. 1–6, 233–237

[6] R. Rodger, The tao of microservices, 2018, pp. 33–46

[7] H. Brink, J. W. Richards & M. Fetherolf, Real-world machine learning, 2017, pp. 3–5

[8] E. Alpaydin, Introduction to machine learning, 2010, pp. 3–10

[9] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, 2014, Cornell University arXiv e-print, pp. 1–5

[10] J. Reinders, A. Robinson & M. McCool, Structured parallel programming: patterns for efficient computation, 2012, pp. 38–42

[11] R. Buyya, R. N. Calheiros & A. V. Dastjerdi, Big data: principles and paradigms, 2016, pp. 110–114

[12] C. Meirosu, W. John, M. Opsenica, T. Mecklin, F. Degirmenci & T. Dinsing, DevOps: fueling the evolution toward 5G networks, 2017, Ericsson technology review, [online], available: https://www.ericsson.com/en/ericsson-technology-review/archive/2017/devops-fueling-the-evolution-toward-5g-networks

[13] C. Boberg, M. Svensson & B. Kovács, Distributed cloud – a key enabler of automotive and industry 4.0 use cases, 2018, Ericsson technology review, [online], available: https://www.ericsson.com/en/ericsson-technology-review/archive/2018/distributed-cloud

[14] J. Reijonen, Decentralized machine learning for autonomous ships in a distributed cloud environment, 2018, University of Vaasa master's thesis, pp. 66–69

[15] S. Gupta & S. Pujari, A multi-agent system (MAS) based scheme for health care and medical diagnosis system, 2009, 2009 international conference on intelligent agents & multi-agent systems

[16] M. Alessandrini, W-M. Lippe & W. Nuesser, Intelligent service system: an agent-based approach for integrating artificial intelligence components in SOA landscapes, 2009, IEEE/WIC/ACEM international conference on web intelligence and intelligent agent technology, pp. 496–499

[17] M. Abid, S. Umar & S. Shahzad, A recommendation system for cloud services selection based on intelligent agents, 2018, Indian journal of science and technology vol 11, pp. 2–5

[18] C. C. Fox, V. Ishakian, V. Muthusamy & A. Slominski, Report from workshop and panel on the status of serverless computing and function-as-a-service (FaaS) in industry and research, 2017, white paper from first international workshop on serverless computing (WoSC) 2017, pp. 3–12

[19] M. Lukša, Kubernetes in action, 2018, pp. 3–12

[20] T. Janssen, What is cloud-native? Is it hype or the future of software development?, 2018, [online], available: https://stackify.com/cloud-native/

[21] I. Lee, The evolution of cloud computing, 2018, [online], available: https://www.embotics.com/blog/the-evolution-of-cloud-computing

[22] DIMECC, 2018, [online], available: https://www.dimecc.com/

[23] X. Meng et al., Mllib: machine learning in apache spark, 2016, The journal of machine learning research, vol 17, pp. 1-7