LSSO: Long Short-term Scaling Optimizer

Balázs Fodor[1,2], László Toka[1,2,3], Balázs Sonkoly[1,2,3]
[1]*HSNLab, TMIT, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics,*
[2]*MTA-BME Network Softwarization Research Group,* [3]*ELKH-BME Cloud Applications Research Group*

Abstract

Demand forecast-based resource allocation is a key element of both application and network service management and resource cost optimization in cloud environments. A central mechanism called scaling or auto-scaling is responsible for adjusting the allocated resources dynamically to current or predicted demands. Most of today's solutions support short-term optimization, where the long-term effect of scaling actions is not considered, therefore the overall operational cost can be sub-optimal. In this paper, we address this issue and provide a long and short-term scaling optimization method called LSSO. Our contribution is threefold. First, we introduce the formal description of the LSSO scaling problem motivated by real cloud native applications. Second, we transform the problem into a tractable graph representation and show that the optimal solution of the original problem emerges as a shortest path problem in the graph. As a result, the transformation and the optimal solution can be computed in polynomial time. Third, we evaluate and validate the proposed algorithm by numerical analysis on multiple datasets using a cost model which considers the running and scaling costs. Results suggest that if the dynamics of the traffic is fairly predictable and the scaling cost is not negligible, LSSO is able to reduce the operation costs significantly. The exact cost gain depends on the ratio of the running and scaling costs, but in realistic operation regimes it can reach even 18% and the method can also tolerate some inaccuracies in the forecast.

Index Terms

cloud computing, scaling optimization

I. Introduction

Private and public cloud systems are the leading execution environments of applications and network services due to their configurability, robustness and elasticity. Providers are able to set and change the resources needed for their application dynamically thereby optimizing the operating cost. Indeed, one of the most important aspects of resource management in cloud environment is scaling, i.e., how one can change the amount of resources allocated to the application. Horizontal scaling changes the number of instances (pods, virtual machines, etc.) in scale out/in events, while vertical scaling modifies the resources under a given instance (e.g. CPU) by scale up/down actions. Several cloud systems provide auto-scaling functionality out of the box (e.g. Kubernetes' HPA, AWS Auto Scaling, Azure Autoscale or Google Cloud's Autoscaling groups).

For the past few years there has been a growing interest in more sophisticated scaling solutions based on predicted future resource usage or traffic. Several research papers focus on optimizing the resource allocation and thereby the application operation cost based on these forecasts. Research shows that predictive scaling can increase application performance and / or reduce operating costs, however previous works have mainly focused on short-term optimization of the resource allocation or have not dealt with the costs of scaling actions.

Cloud native application development has become a leading software development paradigm [1]. The key concept is to build the application directly for public or private cloud systems using services made available natively by the provider. Often the microservices architecture is followed making use of multiple loosely coupled components. Cloud native telecommunication (telco) applications pose new challenges in terms of scaling because the characteristics of the telco realm must also be taken into account, e.g., strict requirements on response time and on service downtime, non-negligible start-up time or redistribution of traffic during scaling events. These requirements support the idea of using more general cost models during the scaling optimization. Furthermore, in telecommunication systems the user traffic shows predictable trends, thus accurate prediction models can be built for long-term forecasts and the resource allocation can be optimized for the total traffic prediction. Similar considerations can be made for smart city applications, e.g., managing autonomous vehicles in smart city environments, where the vehicle is in constant communication with its surroundings while clear patterns can be observed in the city's traffic.

In this paper we define the long short-term scaling optimization problem (LSSOP) which considers long-term predictions on instance numbers and takes a predefined cost model into account for the overall optimization. The solution is the optimal resource allocation for any given time period. Our main contributions are the following. First, we introduce the LSSO problem capturing the preferences of the application provider and the cloud application in terms of resource allocation. Second, we propose an optimizer method solving the problem which is based on a transformation to a shortest path problem and provides the optimal solution in polynomial time. Third, the proposed algorithm is evaluated on multiple datasets with different configurations to showcase the operational cost gains for different cloud native scenarios where the scaling and running costs are considered.

The rest of the paper is organized as follows. Section II gives an overview on the related work. We present the LSSO problem and its solution in Section III. Section IV provides the numerical evaluation of the proposed algorithm. Finally, our conclusions are drawn in Section V.

II. Related work

In this section we highlight the research works closely related to the resource scaling optimization we are addressing.

A. Demand prediction and short-term resource scaling

Demand forecast-based scaling techniques combined with short-term resource allocation are widely adapted to manage cloud native applications. Authors of [2]–[7] introduced machine learning based forecasting and scaling solutions for cloud applications. The algorithm proposed in [2] is able to predict the response time of the application for a short period ahead and select the best scaling action. An adaptive Virtual Network Function (VNF) scaling approach is described in [3], which uses an online learning approach to predict the traffic demand, and calculates the requested instances considering the processing capacities of the different VNF types. In [4], a classifier is presented, which is trained on historical data to predict the number of VNF instances for the next time period. Authors of [5] presented an auto-scaling system that automatically detects Service Level Agreement (SLA) violations, determines the services requiring scaling, and evaluates how much resource those need. Authors of [6] proposed a solution used by Google to configure the resources for tasks in a job using both horizontal and vertical scaling. In [7] a scaling engine is introduced that utilizes multiple machine learning forecast methods in order to always give the lead to the method that suits best the actual request dynamics. Scaling decisions are made based on the predictions of the best model.

These solutions apply short-term predictions to optimize the resource allocation for the next short time period based on historical data. Although, they can achieve improvement in resource scaling, the long-term effect of the scaling action is not considered which can lead to sub-optimal costs.

B. Long-term predictions and resource scaling

There is a considerable amount of literature on utilizing long-term predictions in resource scaling and application management. Authors of [8]–[11] proposed long-term traffic prediction schemes to improve cloud application management using AutoRegressive Integrated Moving Average (ARIMA) and exponential smoothing [8] or Holt-Winters Model [9]. Authors of [10] proposed a combined solution for cloud native telco microservices, long-term forecast and a threshold based approach are used to create a resource plan and short-term prediction is applied to correct peaks. Duggan et al. [11] utilized recurrent neural networks to predict resource usage multiple steps ahead in time. These methods do not consider complex cost models and not seek for optimal scaling decisions in the long run, however they can be an initial point for our optimization which then will adjust the resource allocation to the predicted traffic.

C. Cost optimization in resource scaling

Authors of [12] proposed a reinforcement learning (RL)-based solution for horizontal and vertical resource provisioning. They use a cost function which considers the cost of scaling, running the application and violating SLA, and the agents choose their actions to minimize this cost. However, the state on which the agents are trained contains only the current resource and system information, therefore the agent can hardly understand the long-term effect of its actions. Authors of [13]–[15] developed cost models and used them for cost-aware resource provisioning in edge cloud environment [13] and for optimizing network functions in a telco environment [14], [15]. The model in [13] also considers the load prediction for the next time interval, however the applicability of the models for long-term traffic or resource predictions are not discussed. A complex VNF scaling solution is described in [16], which models the traffic, and the resource provisioning problem as a non-stationary fractional Brownian motion and considers the long-term effect of load balancing, migration cost and resource overloading penalty. This work mainly focuses on detecting and handling change points in the traffic, whereas our work relies on long-term prediction and the optimization of resource allocation, and enables a wide range of cost models. Authors of [17] proposed FOX, which is a mediator between the auto-scaler and the charging model to avoid scaling if it impacts the costs negatively. FOX delays or omits scale-in if the resource is required in the future. They address a similar problem to LSSO, however, the algorithm which modifies the scaling decisions to lower the costs is based on a simple heuristic. In contrast, our proposed solution is able to find the optimal scaling decisions in case of one cloud service. Furthermore, in that work only two simple cost models are considered, while LSSO supports a wide range of cost functions. Nevertheless, the framework proposed in [17] which provides the input for the heuristic of FOX can be applied for LSSO too. Despite its simplicity, FOX outperforms the other examined scaling algorithms according to their results, therefore LSSO and FOX are compared in this work.

III. LSSO: Long Short-term Scaling Optimizer

In this section, the long short-term scaling optimization problem (LSSOP) is introduced formally along with its required inputs and constraints. Thereafter, the long short-term scaling optimizer (LSSO) is presented, which solves LSSOP in polynomial time. Finally we show how LSSO fits into the MAPE-K control loop [18].

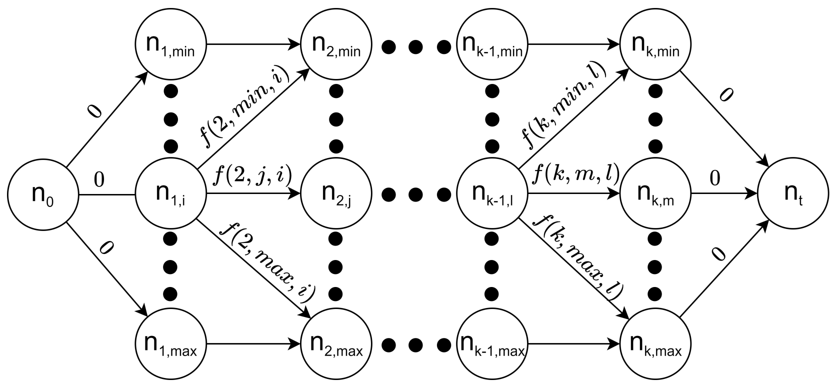

Fig. 1. Graph representation of the LSSO problem

A. Problem formulation - LSSOP

LSSOP assumes that resource scaling suggestions are given for k consecutive time intervals ($\mathbf{s} = (s_1, s_2, \ldots, s_k)$, $s_i \in \mathbb{N}$ $i \in \{1 \ldots k\}$), i.e., the short-term optimal resource values for each of the next k time intervals. s_i indicates the number of instances to be provisioned in the ith time period for a given cloud service. In case of a real life application, the instance can be a virtual machine, container, VNF or other scaling unit, which is the base of scaling during the horizontal resource provisioning. It is assumed that during the calculation of s_i, the instance number is only optimized for that particular time period. As a minimum value, at least the short-term optimal resource values must be provisioned, as allocating less resources would violate the provider or application policies used for the short-term optimization (e.g. in case of telco systems this could lead to violating SLA). Let us denote the number of instances actually provisioned in the ith time period with d_i, $d_i \geq s_i$. We further assume an arbitrary $f(i, d_i, d_{i-1})$, $i \in \{1, \ldots, k\}$ cost function which takes three parameters: i denotes the interval, while d_i and d_{i-1} denote the decisions in the current and previous intervals, respectively. The output is the execution cost of the application for the ith interval given the current and the previous decisions. We assume that evaluating the function can be done efficiently (polynomial in k). The goal is to minimize the overall cost of the application for that k time intervals so that in each time interval i, the actual number of instances has to be at least s_i. The execution cost of a given $\mathbf{d} = (d_1, d_2, \ldots, d_k)$ decision can be formulated as:

$$C_f(\mathbf{d}) = \sum_{i=2}^{k} f(i, d_i, d_{i-1})\,, \tag{1}$$

The description of our problem can be formalized as follows.

Problem 1 (LSSOP)**.** LSSOP is a long short-term scaling optimization problem given in the following form:

Input:

$$\begin{gathered} \mathbf{s} = (s_1, \ldots, s_k),\ s_i \in \mathbb{N}\ i \in \{1, 2, \ldots, k\} \\ s_{\max} = \text{maximum value in } \mathbf{s} \\ f = \text{cost function} \end{gathered} \tag{2}$$

Optimization:

$$\min C_f(\mathbf{d}) = \min \sum_{i=2}^{k} f(i, d_i, d_{i-1}) \tag{3}$$

Constraints:

$$\begin{gathered} s_i \leq d_i \leq s_{\max} \\ d_i \in \mathbb{N} \end{gathered} \tag{4}$$

Output:

$$\mathbf{d} = (d_1, \ldots, d_k) \tag{5}$$

The output of LSSOP is a $\mathbf{d}$ vector containing the d_i actual instance numbers minimizing the execution cost of the application. A specific problem of LSSOP can be described with two parameters $\mathbf{s}$ and f as $LSSOP(f, \mathbf{s})$.

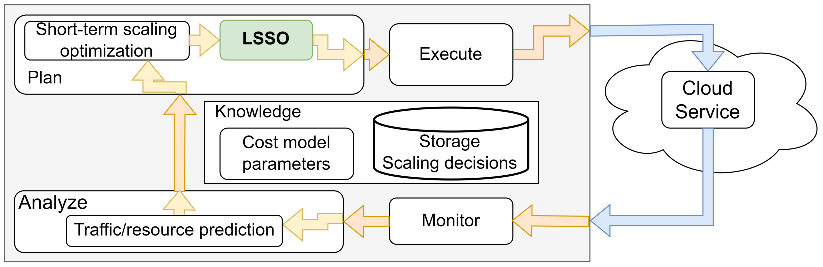
Short-term scaling optimization
LSSO
Plan
Execute
Cloud Service
Knowledge
Cost model parameters
Storage Scaling decisions
Analyze
Traffic/resource prediction
Monitor

Fig. 2. MAPE-K control loop with LSSO

B. Solving LSSOP in polynomial time: LSSO

The key idea to solve the problem is the transformation of LSSOP into a classical shortest path problem, which can be solved in polynomial time (e.g. with Dijkstra's algorithm [19]). Our algorithm, which takes an $LSSOP(f, \mathbf{s})$, calculates the graph transformation and solves the shortest path problem, is referred to as LSSO (Long Short-term Scaling Optimizer).

Let $L = LSSOP(f, \mathbf{s})$ be an arbitrary LSSOP and $s_{\min}$ the minimum value in $\mathbf{s}$. We introduce a graph representation of L, called the LSSOP graph of L, denoted by G_L. Let $G_L = G(N, E, W)$ be a weighted directed graph, where $N = \{n_0, n_t, n_{i,j} \mid i \in \{1, \dots, k\}, j \in \{s_{\min}, \dots, s_{\max}\}\}$ contains the nodes of the graph. Each node represents the number of instances (j) running at time period i. E.g. $n_{2,3}$ indicates that in the second period there are three instances running. Let $E = \{e(n_{i,j}, n_{i+1,l}) \mid i \in \{1, \dots, (k-1)\}\ j, l \in \{s_{\min}, \dots, s_{\max}\}\} \bigcup \{e(n_0, n_{1,i}) \mid i \in \{s_{\min}, \dots, s_{\max}\}\} \bigcup \{e(n_{k,i}, n_t) \mid i \in \{s_{\min}, \dots, s_{\max}\}\}$ be the set of edges, where $e(n_x, n_y)$ indicates that n_x is connected to n_y. The structure of G_L can be seen in Fig. 1, where for the sake of simplicity, $\max$ and $\min$ denote $s_{\max}$ and $s_{\min}$ respectively. There is a starting node n_0 and a termination node n_t, between them there are k columns, each column contains $s_{\max} - s_{\min} + 1$ nodes. n_0 is connected to all nodes in the first column, and the nodes of the kth column are connected to n_t. In columns $1, 2, \dots, (k-1)$, each node is connected to all nodes in the next column. The weights can be formulated as follows: $w(n_x, n_y)$ will denote the weight of the edge $e(n_x, n_y)$,

$$W = \{w(n_x, n_y) \mid e(n_x, n_y) \in E\}, \text{where}$$

$$w(n_x, n_y) = \begin{cases} 0, & \text{if } n_x = n_0 \\ & \text{ and } n_y = n_{1,i} \\ & \text{ and } i \geq s_1 \\ f(l, j, i), & \text{if } n_x = n_{l-1,i} \\ & \text{ and } n_y = n_{l,j} \\ & \text{ and } j \geq s_l \\ 0, & \text{if } n_y = n_t \\ & \text{ and } n_x = n_{k,i} \\ \infty, & \text{else} \end{cases} \tag{6}$$

The weights describe the cost of moving from one node to another using the edge. We can conclude that the solution to the original problem L is equivalent to finding the shortest path from n_0 to n_t in G_L. The size of $G_L = G(N, E, W)$ LSSOP graph depends on the original $L = LSSOP(f, \mathbf{s})$:

$$\begin{aligned} |N| &= k(s_{\max} - s_{\min} + 1) + 2 \\ |E| &= 2(s_{\max} - s_{\min} + 1) + k(s_{\max} - s_{\min} + 1)^2, \end{aligned} \tag{7}$$

nevertheless, $s_{\max} - s_{\min} + 1$ is a constant, f can be calculated efficiently, therefore both the generation of G_L and the shortest path problem on G_L can be solved in polynomial time. This proves that the optimal solution of LSSOP can be found in polynomial time, too.

C. LSSO in the MAPE-K loop

MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) is a control method which is widely adopted for elastic application resource management. The basic steps it executes are the monitoring of the application, analysis of the data, planning the actions, decisions which need to be taken and executing them. Fig. 2 shows the necessary elements required for our solution to work in a MAPE-K loop. The monitor shall collect resource usage and/or traffic metrics, which can be used during the analysis to provide long-term resource/traffic predictions. During planning, these predictions are utilized to create an initial

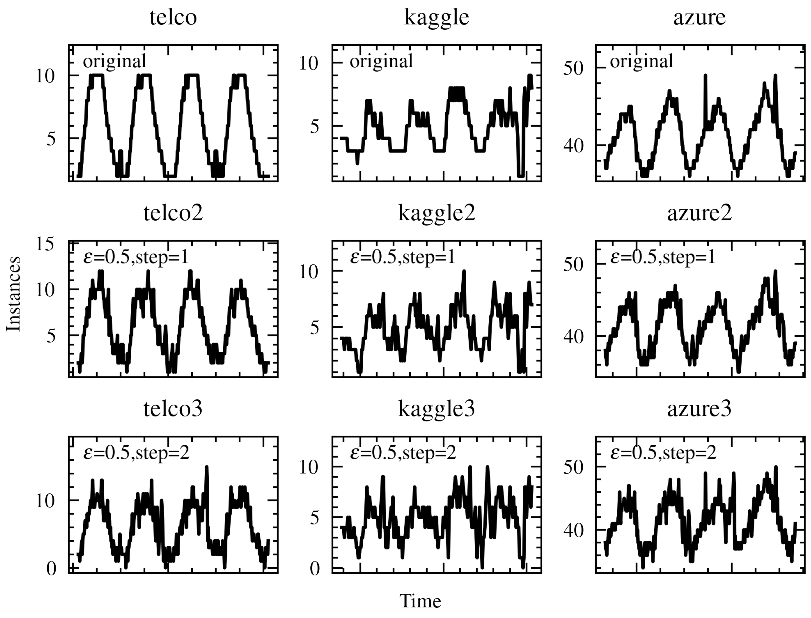

Fig. 3. Different datasets with apparent daily pattern

scaling plan. This scaling plan can be fine-tuned by LSSO and provide the optimal scaling plan to be executed. Furthermore, it can be seen, that LSSO can use the common knowledge to obtain the parameters of the cost model. In [17] authors give a detailed description about how to implement the MAPE-K loop to provide this type of scaling decisions for further optimization, therefore our current focus is on calculating the optimal scaling plan efficiently.

IV. Numerical evaluation

LSSO algorithm is analyzed and evaluated via several simulations with different configurations on multiple datasets. The evaluation covers the analysis with perfect predictions; the comparison with a heuristic algorithm called FOX [17]; and the performance evaluation using imperfect predictions.

A. Datasets

We collected CPU utilization data from three sources, converted them into instance numbers using a threshold based approach. In addition, we generated randomized versions of them (superposing artificial noise on the original data series), which yields datasets with weaker trends.

1) telco: A dataset from a deployed telco service provided by a network operator. The dataset consists of four days of CPU consumption data of a virtualized CSCF (Call Session Control Function) network element of a deployed IMS (IP Multimedia Subsystem) application [20], the time granularity is 15 minutes. It shows that the traffic and the trends are predictable. Unfortunately, the dataset cannot be shared publicly.

2) kaggle: Another dataset is collected from kaggle.com [21], that contains one and a half days of CPU utilization, and the time granularity is 1 minute. Predictable trends can be observed, however there is a little anomaly at the end of the dataset.

3) azure: The third dataset consists of CPU utilization measurements from Microsoft Azure [22] and contains one month of data with clear daily trends, sampled every 5 minutes.

The datasets hereinafter are referred to as *telco*, *kaggle* and *azure*, respectively. In order to make the optimization results comparable, the datasets have to be harmonized in time interval and scale. Therefore, each input is converted into a four days long data series with a sampling period of 15 minutes. The randomized versions are generated with a simple function which takes the instance numbers, a random factor (ϵ), and a $step$ value as input variables, and follows the original instance numbers with $1 - \epsilon$ probability, or generates a random integer around the previous instance value (in $[-step, +step]$ range) with probability ϵ. The larger the ϵ, the more noisy the data. For each original dataset, two randomized versions are created with $\epsilon = 0.5$ and $step \in \{1, 2\}$. The datasets can be seen in Fig. 3, where the first row displays the three original sources, and below them the randomized versions are displayed. The identifiers of the datasets that will be used later are placed above the graphs.

TABLE I
SIMULATION CONFIGURATIONS AND RESULTS FOR $c_u = 0.00575\$$

	Setup1	Setup2	Setup3	Setup4	Setup5	Setup6
$c_s[10^{-3}\$]$	2.875	3.833	5.75	57.5	115	287.5
c_s/c_u	0.5	0.66	1	10	20	50
Cost gain[%] in case of different datasets						
telco	0	0	0.2	9	18	46
telco2	0	0.6	1.8	33	49	71
telco3	0	1	2.9	42	58	77
kaggle	0	0.4	1.3	28	47	71
kaggle2	0	0.8	2.2	41	60	80
kaggle3	0	1.4	4	51	69	85
azure	0	0.1	0.2	5	11	28
azure2	0	0.1	0.3	10	20	41
azure3	0	0.2	0.5	15	27	51

B. Cost model

Although LSSO supports a wide range of cost models, for demonstration purposes we evaluate a telco specific one, where the necessary re-configurations during scaling shall be included in the model. Therefore the cost model presented by eq. (8) considers the cost of running (eq. (9)) and scaling (eq. (10)) the application.

$$f(i, d_i, d_{i-1}) = f_u(i, d_i, d_{i-1}) + f_s(i, d_i, d_{i-1}) \tag{8}$$

$$f_u(i, d_i, d_{i-1}) = \begin{cases} c_u(d_{i-1} + d_i) & \text{if } i = 2, \\ c_u d_i & \text{else} \end{cases} \tag{9}$$

$$f_s(i, d_i, d_{i-1}) = c_s|d_i - d_{i-1}| \tag{10}$$

Eq. (9) shows the resource usage cost and c_u identifies the cost of running one instance for one period (running cost). Because of the operation of LSSO d_1 is only considered in $f(2, d_2, d_1)$, therefore $f_u(2, d_2, d_1)$ has to contain the running cost of the first interval. Eq. (10) denotes the cost of scaling from d_{i-1} instance to d_i and c_s is the cost of adding or removing one instance (scaling cost).

C. Results

1) Analysis of LSSO with perfect predictions: In the first part of our analysis we assume perfect predictions. With this approach, the maximum performance of LSSO can be examined without the negative effect of inaccurate forecast.We analyze the nine presented datasets with different cost parameters. Our intention was to show a possible application of our proposed optimization framework with several real-world datasets with realistic configurations. The first part of Table I contains the value of c_s in the different setups, along with the ratio of c_s and c_u. c_u is calculated from the hourly running price of an Amazon t2.small instance [23], which is considered as a typical baseline. Our goal is to present scenarios where the scaling cost is smaller than the running cost (Setup 1, 2 and 3) and vice versa (Setup 4, 5 and 6). Let us introduce cost gain as a metric quantifying the impact of the optimization:

$$gain = 1 - \frac{cost_o}{cost_s}, \tag{11}$$

where $cost_o$ and $cost_s$ are calculated by applying the C cost function to the optimized ($\mathbf{d}$) and to the short-term optimal resource allocation ($\mathbf{s}$). The cost gain describes the part of the short-term optimal cost, which was spared by our solution. The second part of Table I shows the cost gains of the experiments of the six configurations with the nine datasets. As we see, in case of Setup 1, LSSO cannot further optimize the scaling plan because the scaling cost component in that scenarios is not significant enough to make it worth preventing scale-in. In other setups, improvement can be observed, moreover in Setup 4, 5 and 6 the cost gain is significant (between 5% and 85% depending on the dataset). It can also be observed that the noisier the dataset, the larger the cost gain. This is reasonable because the small scaling actions triggered by the random fluctuation can be eliminated by LSSO.

Fig. 4 presents Setup 4, 5 and 6 combined with the *telco* original instance numbers. The different setups are indicated by different line styles, and the name of the setup is displayed in the legend along with the corresponding cost gain. In these scenarios, there is a significant improvement in the final cost gain. These results are in line with the nature of the cost function. As the scaling cost becomes more prominent, the optimization seeks to avoid unnecessary scaling actions and the optimal resource allocation gets closer to the constant maximal allocation. It can be observed that in case of Setup 6, scaling is needless, however with this approach the cost could be reduced by 46% in case of the *telco* scenario. Setup 5 results 18%

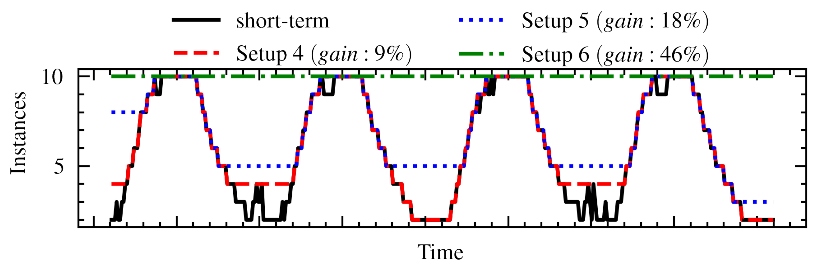

Fig. 4. Simulation results on telco original dataset for Setup 4, 5 and 6

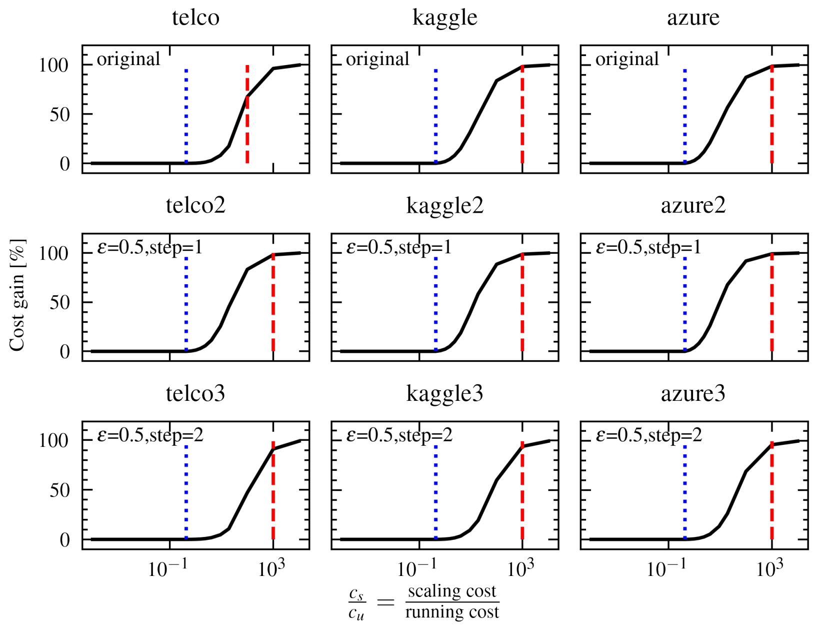

Fig. 5. LSSO limitations of optimization on the test datasets and cost function

cost improvement, where the scaling cost is 20 times higher than the running cost, and even in case of Setup 3, where the two cost parameters are the same, the improvement is noticeable (3% in case of *telco3* dataset). These characteristics may be valid for complex telco applications, where the scaling cost is high due to the longer start-up time of the virtualized network elements and the additional steps needed to be taken during scaling, e.g., load-balancing.

Several additional simulations were executed with different scaling and running costs to analyze the optimization algorithm on a wide range of possible configurations. The results are summarized in Fig. 5. It presents the cost gain as a function of the ratio of the costs: scaling cost vs. running cost. The graph reveals the main characteristics of the cost function the optimization is working with, i.e., when the scaling cost is low compared to the running cost, it is worth executing the suggested scaling actions, and therefore the optimization can only slightly improve the short-term optimal resource allocation. However, as the scaling cost becomes higher than the running cost, the cost gain rises until the upper limit is reached, where the optimization suggests a constant amount of resources without scaling. In the subplots, we marked with a blue dotted line the point from which it is not the short-term optimal resource allocation that gives the lowest cost: by applying LSSO optimization, improvement can be made. We found that this value is around $c_s/c_u \approx 0.5$ for every test dataset. The red dashed line indicates the point from which the constant allocation is the optimal decision. According to our measurements, this value is somewhere between 100 and 1000 in case of our datasets. These results suggest that LSSO can be applied in a wide range of configurations to achieve the optimal operation.

2) Comparison of LSSO and FOX: Next, we compare LSSO with a recently proposed related mechanism called FOX [17]. FOX applies a heuristic algorithm to prevent scale-in actions based on a long-term forecast which is able to optimize the short-term resource suggestions when the next decision would be scale-in. It examines the next suggested instance numbers in a predefined look-ahead window, and selects the largest value which is less or equal to the current instance number. Although,

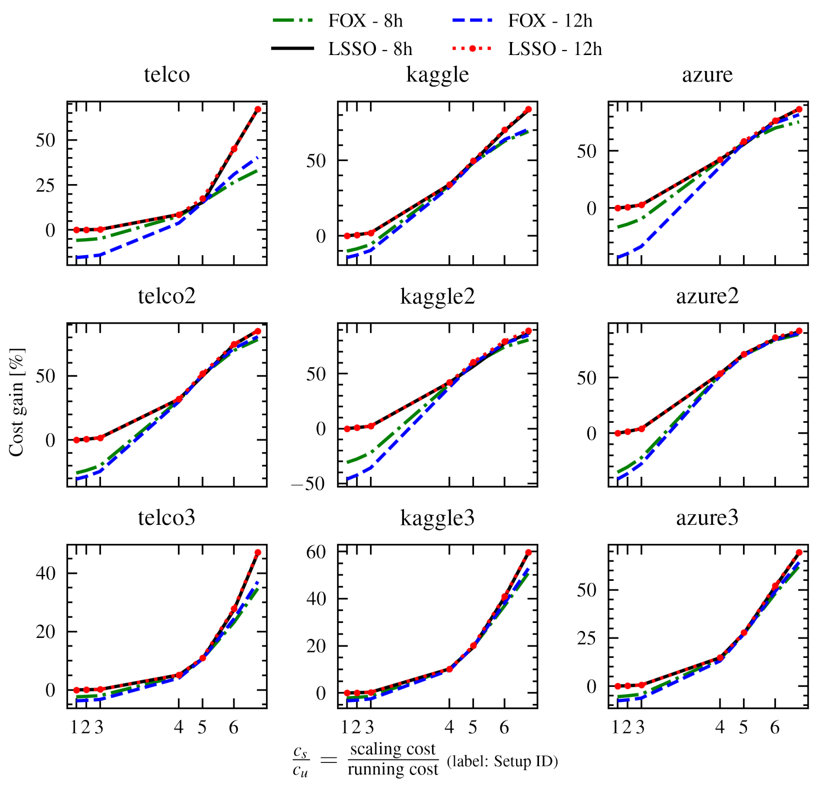

Fig. 6. Performance comparision of LSSO and FOX

TABLE II
MEAN ABSOLUTE ERROR OF THE DIFFERENT PREDICTION METHODS

	telco/telco2/telco3	kaggle/kaggle2/kaggle3	azure/azure2/azure3
P1	0.23 / 0.94 / 1.57	1.27 / 1.39 / 1.82	1.32 / 1.64 / 2.16
P2	1.08 / 1.45 / 1.99	1.72 / 1.79/ 2.34	1.86 / 1.96 / 2.44

this approach is originally designed to optimize scaling decisions using hourly instance based pricing, the scale-in preventing approach is applicable to our cost function as well, when the scaling cost is considerable. Here, we also assume perfect predictions, however, a forecasting window of 8 and 12 hours is used, to prevent the algorithms to optimize over the whole dataset.

The result of the comparison can be seen in Fig. 6. In the legend, the applied forecast window length is displayed next to the name of the algorithm. The x-axis labels help identify the previously defined 6 setups for which the simulations were performed. For Setup 1,2 and 3 LSSO outperforms FOX as it is able to apply only the short-term suggested values, when no further optimization can be made. However FOX's heuristic has strict rules, and it is not able to adapt. On the other hand, when the scaling cost is considerable (Setup 4,5 and 6), FOX comes close to the cost gain of LSSO. According to our simulations, only a few percent differences can be observed in favor of LSSO in most cases. However in case of *telco*, *kaggle* and the datasets in the last row, it can be seen, that in case of Setup 6, and when the scaling cost, running cost ratio is larger than 50, LSSO outperform FOX significantly.

It can be concluded that LSSO can achieve larger cost gain in general than FOX, however in certain scenarios, FOX gives similar results. The advantage of LSSO besides the better performance is that it supports a wide range of cost functions and therefore a wide set of use cases and configurations are enabled.

3) Evaluation of LSSO using imperfect predictions: In the last part of the evaluation, we consider the case of inaccurate predictions. To achieve this, two forecast methods are used. The first one uses the previous day's traffic for the current day, the other one uses the same with some added noise, to make the prediction accuracy weaker. Table II contains the mean absolute error of the two prediction algorithm, where $P1$ stands for the forecast using the previous day, and $P2$ is the noisy version. The result is shown in Fig. 7, where as a reference, the simulation with perfect prediction is also displayed indicating the maximum achievable cost gain. In the legend, the name of the prediction and the applied forecast window are displayed. These simulations are considered as the most realistic ones, as they assume imperfect prediction and limited forecast window size.

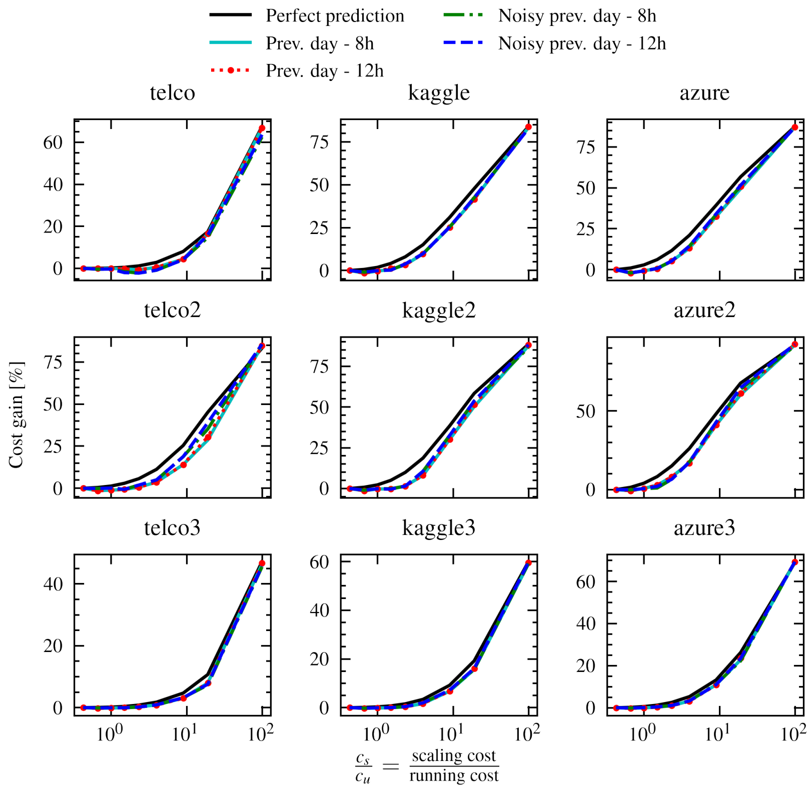

Fig. 7. LSSO performance with imperfect predictions

The range of the x-axes is from 0.5 to 100 presenting the most interesting operating regime. It can be seen, that the maximum cost gain can never be reached while the scaling cost is low. Moreover, the cost gain can also be negative compared to the short-term optimal allocation. The reason is that the short-term resource suggestion is preferred when the scaling cost is small, and the optimization using inaccurate values might execute unnecessary scaling actions which will increase the cost. As the scaling cost gets higher, the effect of inaccurate predictions becomes less and less noticeable, as the optimized allocation uses more instances than suggested to prevent scaling actions, and a small inaccuracy in instance number will not take effect. As a result, the cost gain fairly approximates the cost gain of the perfect prediction.

To sum up, with the increase of scaling cost, the prediction accuracy becomes less prominent, and near maximal cost gain can be achieved, which shows that the examined extent of inaccuracy is tolerable by LSSO.

V. Conclusion

Making use of advanced time-series forecasting methods and models, we have the opportunity to optimize the operation of cloud applications and cloud native telco services, thereby reducing the overall operating costs. We have proposed a formal description of a scaling optimization problem (LSSOP) and provided an algorithm to solve it in polynomial time (LSSO), which includes the transformation of the problem into a shortest path graph theory problem. The method supports a wide range of cost models, where the cost can be formulated as a function of the current and previous scaling decisions. Our LSSO algorithm was evaluated with multiple illustrative examples from different application realms with different configurations. For the evaluation we presented a cost model, which considers the running cost and scaling cost. We found that if the traffic dynamics of a given application is fairly predictable (e.g., in case of telco applications), LSSO can be invoked to significantly reduce the operation costs. Furthermore, we compared LSSO to a heuristic approach (FOX) which optimizes resource allocation based on long-term suggestions. Results show that in some cases FOX approximates the cost gain of LSSO, but LSSO supports a wider set of use cases and it outperforms FOX in each scenario. The effect of imperfect predictions is also considered, and it can be stated that with the increase of scaling cost, the negative effect of imperfect prediction on the cost gain decreases, and the cost gain closes the maximum which is achievable by perfect prediction.

In conclusion, our results suggest, that LSSO is applicable in real-world scenarios, and outperforms both the short-term allocation and the selected baseline long-term suggestion based algorithm.

Acknowledgement

This work was supported by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund through projects $i)$ no. 135074 under the FK_20 funding scheme, $ii)$ 2019-2.1.13-TÉT_IN-2020-00021 under the 2019-2.1.13-TÉT-IN funding scheme, and $iii)$ 2019-2.1.11-TÉT-2020-00183 under the 2019-2.1.11-TÉT funding scheme. L. Toka was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. Supported by the ÚNKP-22-5-BME-317 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

References

[1] "Cloud Native Computing Foundation," https://www.cncf.io/, (Accessed on 11/13/2021).
[2] M. Wajahat *et al.*, "Using machine learning for black-box autoscaling," in *IEEE IGSC*, 2016.
[3] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 486–494.
[4] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling vnfs using machine learning to improve qos and reduce cost," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.
[5] G. Yu *et al.*, "Microscaler: Automatic scaling for microservices with an online learning approach," in *IEEE ICWS*, 2019.
[6] K. Rzadca *et al.*, "Autopilot: workload autoscaling at Google," in *EuroSys*, 2020.
[7] L. Toka *et al.*, "Machine learning-based scaling management for kubernetes edge clusters," *TNSM*, vol. 18, no. 1, pp. 958–972, 2021.
[8] M. Carvalho *et al.*, "Long-term SLOs for reclaimed cloud computing resources," in *ACM SOCC*, 2014.
[9] S. Subramanian and A. Kannammal, "Real time non-linear cloud workload forecasting using the Holt-Winter model," in *IEEE ICCCNT*, 2019.
[10] D.-H. Loung *et al.*, "Predictive autoscaling orchestration for cloud-native telecom microservices," in *IEEE 5GWF*, 2018.
[11] M. Duggan *et al.*, "Predicting host CPU utilization in cloud computing using recurrent neural networks," in *ICITST*. IEEE, 2017, pp. 67–72.
[12] F. Rossi *et al.*, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *IEEE CLOUD*, 2019.
[13] C. Li *et al.*, "Cost-aware automatic scaling and workload-aware replica management for edge-cloud environment," *Journal of Network and Computer Applications*, vol. 180, p. 103017, 2021.
[14] V. Eramo *et al.*, "Proposal and investigation of a reconfiguration cost aware policy for resource allocation in multi-provider NFV infrastructures interconnected by elastic optical networks," *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4098–4114, 2019.
[15] C.-H. Wang *et al.*, "Dynamic cloud network control under reconfiguration delay and cost," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 491–504, 2019.
[16] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao, "Dynamic resource scaling for vnf over nonstationary traffic: A learning approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 2, pp. 648–662, 2020.
[17] V. Lesch *et al.*, "Fox: Cost-awareness for autonomic resource management in public clouds," in *ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 4–15.
[18] J. O. Kephart *et al.*, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
[19] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[20] "ETSI TS 123 228 V16.5.0 - technical specification," https://www.etsi.org/deliver/etsi_ts/123200_123299/123228/16.05.00_60/ts_123228v160500p.pdf, (Accessed on 04/01/2022).
[21] "CPU utilization — Kaggle," https://www.kaggle.com/datasets/gauravduttakiit/cpu-utilization, (Accessed on 03/30/2022).
[22] "Predicting-cloud-CPU-usage-on-Azure-data," https://github.com/amcs1729/Predicting-cloud-CPU-usage-on-Azure-data, (Accessed on 03/30/2022).
[23] "EC2 On-Demand Instance Pricing – Amazon Web Services," https://aws.amazon.com/ec2/pricing/on-demand/, (Accessed on 03/31/2022).