2019 International Conference on Networked Systems (NetSys)

GPU Accelerated Planning and Placement of Edge Clouds

Patrick Kalmbach, Andreas Blenk, Wolfgang Kellerer
Technical University of Munich
Munich, Germany

Rastin Pries, Michael Jarschel, Marco Hoffmann
Nokia Bell Labs
Munich, Germany

***Abstract*—Future 5G communication will rely on edge computing to meet diverse Quality of Service Requirements, in particular, end-to-end latency in the order of milliseconds. Placing edge clouds requires solving a large-scale set cover problem. We show how the challenge posed by the size of the problem can be overcome through GPU accelerated Natural Evolutional Strategies (NES). We present a system that allows the evaluation of technology choices, latency requirements and locations that should be covered at the example of the United States.**

***Index Terms*—GPU Computing, Edge Cloud Computing**

I. Introduction

The Context: Edge Computing. Future 5G communication networks must support applications with stringent Quality of Service (QoS) requirements, in particular, end-to-end latency requirements in the order of milliseconds. Examples are: Augmented reality, virtual reality and autonomous driving [6], [7]. The latency constraints require operators to place computational resources as close to the end user as possible. This is achieved through Multi-access Edge Computing (MEC).

A MEC solution is hosted in an edge cloud, which is a resource-rich computer or cluster of computers deployed at the access network to support low latency applications [3]. Edge clouds can be equipped with acceleration hardware, e.g., Graphical Processing Units (GPUs) for computer vision or Field Programmable Gate Arrays (FPGAs) for network acceleration and deep learning tasks.

The Problem: Edge Cloud Placement. This demo showcases a planning and placement tool for edge clouds. The first step is determining to what extend a specific service should be supported, e.g., by specifying a percentage of a country's population for which the service should be available. Based on the provided information, edge clouds are placed at given locations, such that a cost function is minimized.

Edge clouds with the MEC solution hosting the above mentioned applications are located right behind a local User Plane Function (UPF), which connects multiple base stations [10]. The connections between the edge clouds and base stations serving the end users can be established through different technologies, e.g., optical fiber or micro-wave links, which has to be considered when determining a placement.

This work has been performed in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1), and it is partly funded by the German BMBF. The authors alone are responsible for the content of the paper.

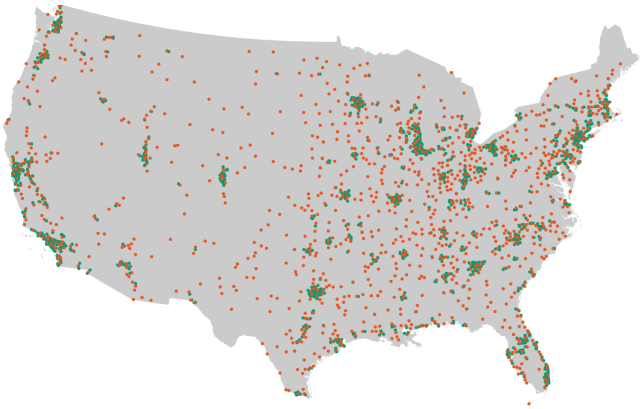

Fig. 1: A possible edge cloud placement for cities with more than 10 000 inhabitants. The orange dots represent locations at which edge clouds are placed and overlap the green dots, representing locations that should be covered.

The problem of determining an optimal placement can be reduced to a set-cover problem, which is one of Karp's original NP-complete problems [13].

The Challenge: Problem Size. The set-cover problem is a well studied problem. Still, obtaining good solutions in this context is challenging due to the *problem dimension*. Fig. 1 shows a map from the US with cities that have more than 10 000 inhabitants. In this case more than 3 000 locations must be covered. To obtain good solutions at this scale, significant effort is required to design scenario specific heuristics or approximate algorithms.

The Opportunity: GPU accelerated Natural Evolutional Strategies (NES). In this demo we show how the challenge posed by the size of the problem can be overcome through NES and GPU computing.

NES are a family of state-of-the-art Black-Box Optimization (BBO) algorithms, and as such applicable to any optimization problem, as long as its objective function can be evaluated for arbitrary points in parameter space [5].

GPU computing provides massive speed-ups for data-parallel workloads, in which the same instructions can independently be applied to many data-elements, e.g., when multiplying matrices [1]. Data-parallel workloads arise in

many disciplines, including networking [4], [11]

By considering the matrix notation of the underlying optimization problem, we can exploit the computing power of a GPU to rapidly evaluate candidate solutions, and thus significantly speed up NES.

Contribution. We illustrate how large-scale optimization problems arising in the context of edge-computing can be solved through GPU computing and black-box optimization. Our demo allows the evaluation of different technology choices, latency requirements and population coverages within minutes, placing less edge clouds than a greedy baseline.

II. Placement Problem

Problem input is a set $\mathcal{U}$ of locations u demanding to be covered with an acceptable latency. One edge cloud location can cover a subset $\mathcal{S} \subseteq \mathcal{U}$ of locations. The set cover problem identifies a number of subsets whose union is $\mathcal{U}$, and that minimize a cost function $f : \boldsymbol{\mathcal{S}} \to \mathbb{R}$. The set $\boldsymbol{\mathcal{S}} \subset 2^{\mathcal{U}}$ contains subsets of $\mathcal{U}$ for all potential edge cloud locations. The problem can be formulated as:

$$\text{argmin}_x \sum_{\mathcal{S} \in \boldsymbol{\mathcal{S}}} x_{\mathcal{S}} f(\mathcal{S}), \tag{1}$$

subject to:

$$\sum_{\mathcal{S} \in \boldsymbol{\mathcal{S}}} x_{\mathcal{S}} y_{u,\mathcal{S}} \geq 1 \qquad \forall u \in \mathcal{U} \tag{2}$$

$$x_{\mathcal{S}} \in \{0,1\} \qquad \forall \mathcal{S} \in \boldsymbol{\mathcal{S}} \tag{3}$$

The binary variable $x_{\mathcal{S}}$ indicates whether subset $\mathcal{S} \in \boldsymbol{\mathcal{S}}$ is selected for the covering. Variable $y_{u,\mathcal{S}}$ is one if $u \in \mathcal{S}$, i.e., u is covered, and zero else.

III. Natural Evolutional Strategies

NES are a family of black-box optimization methods that maintain and update a search distribution over the parameter space of the problem, from which a sample of λ search points is drawn and evaluated through a fitness function [5]. We use the fitness function:

$$\sum_{\mathcal{S} \in \boldsymbol{\mathcal{S}}} x_{\mathcal{S}} f(\mathcal{S}) + \alpha \sum_{u \in \mathcal{U}} \delta \left(\sum_{\mathcal{S} \in \boldsymbol{\mathcal{S}}} x_{\mathcal{S}} y_{u,\mathcal{S}}, 0 \right), \tag{4}$$

where we added Constraint (2) as penalty scaled with a large scalar α to the original objective. Here, δ is an indicator function returning one if the first argument is zero. The penalty is convex and guides the search into a feasible region of the solution space [12].

We do not apply NES directly to the set cover problem, since NES requires a continuous parameter space. We assume that an edge cloud is located at location i with probability $1/(1+\exp(-\theta_i))$, where θ_i is a real number. We use NES to obtain a distribution over real valued vectors θ that result in feasible placements having low cost with high probability.

Alg. 1 illustrates the overall procedure. In each iteration, λ parameter vectors $\theta_k \in \mathbb{R}^{|\mathcal{U}|}$ are sampled. Each element in each sample θ_k is then projected to the interval $[0,1]$ through element-wise application of the sigmoid function.

Data: learning rate γ, sample size λ, search distribution π, initial parameter ψ, fitness function $f(\cdot)$
Result: Optimized search distribution
while *not converged* **do**
  $\bigtriangledown_\psi J \leftarrow 0$
  for $i = 1, \ldots, \lambda$ **do**
    draw sample: $\theta_k \sim \pi(\cdot \mid \psi)$
    project to $[0,1]$: $p_k \leftarrow 1/(1+\exp(-\theta_k))$
    sample placement: $x_k \sim \text{Bernoulli}(\cdot \mid p_k)$
    evaluate fitness: $c_k \leftarrow f(x_k)$
    accumulate gradient:
    $\bigtriangledown_\psi J \leftarrow \bigtriangledown_\psi J + \frac{c_k}{\lambda} \bigtriangledown_\psi \log \pi(\theta_k \mid \psi)$
  end
  update paramter: $\psi \leftarrow \psi + \gamma \bigtriangledown_\psi J$
end

Algorithm 1: NES algorithm for far-edge cloud placement [5]. Note that θ_k, ψ, p_k and x_k are vector valued, and $\text{Bernoulli}(\cdot)$ operates element-wise on vector p_k.

The elements in the resulting vector $p_k \in [0,1]^{|\mathcal{U}|}$ can be interpreted as the probability of placing a far-edge cloud at a specific location. A concrete placement is sampled by performing for each element in p_k a bernoulli experiment, resulting in the binary vector $x_k \in \{0,1\}^{|\mathcal{U}|}$. The cost, i.e., fitness of placement x_k can then be assessed, and the gradient for the parameters of the search distribution be calculated. Finally, the parameters of the search distribution are updated.

Because of the problem size, we use an isotropic multivariate normal distribution as search distribution π. This allows for fast sampling of new parameter vectors [5]. We keep the variance σ of each dimension fixed. Consequently, ψ corresponds to the mean vector μ. As suggested in [5], we use fitness shaping to make the procedure more robust to outlier individuals, and less prone to early convergence.

Since the fitness of a placement can be easily evaluated, i.e., does not require simulations or executions on a physical system, we can leverage a GPU to execute the for-loop in Alg. 1 in parallel for all samples.

To this end, we vectorize the fitness function in Eq. (4):

$$X^T c(\mathcal{U}) + \alpha \left(\delta(X^T Y, 0) \cdot \mathbb{1} \right). \tag{5}$$

The matrix $X \in \mathbb{R}^{|\mathcal{U}| \times \lambda}$ stores the sampled placements as columns. Function $c(\cdot)$ returns a vector of costs for placing a far-edge cloud at each location. The matrix-vector product $X^T c(\mathcal{U})$ thus represents the cost for each sampled placement. Matrix $Y \in \{0,1\}^{|\mathcal{U}| \times |\mathcal{U}|}$ is a symmetric matrix, representing which locations each city would cover, should a far-edge cloud be placed there. The result of the matrix-matrix product $X^T Y$ is a $\mathbb{N}^{\lambda \times |\mathcal{U}|}$ matrix, containing for each sampled placement how often every location is covered. The element-wise indicator function $\delta(\cdot)$ returns a binary matrix, in which entries corresponding to uncovered locations are set to one. Multiplied with the one-vector and a scalar constant, the resulting vector represents the penalty for leaving cities uncovered.

TABLE I: Configurable parameters. The user can choose between different algorithms, change the input or adapt hyperparameters.

Demo component	Configurable Parameter
Algorithm Input	Population that should be covered
	Cost model (greenfield or existing infrastructure)
	Path scale
	Required end-to-end latency
Available Algorithms	Approximate Algorithm CPU
	Evolutional Strategies CPU
	Evolutional Strategies GPU
Algorithm Parameters (ES only)	Number of samples per generation
	Learning rate

The *space complexity* or *runtime* of one iteration of Alg. 1 is dominated by the matrix-matrix product in the penalty term with $\mathcal{O}(\lambda \mid \mathcal{U} \mid^2)$. The computational and space complexity can be reduced to $\mathcal{O}(\lambda \mid \mathcal{U} \mid)$ using a sparse matrix representation for Y. Less than $1\,\%$ of the entries in Y are non-zero, and this number grows only linearly with the amount of locations for the considered small latency regime. The main advantage of the sparse matrix representation is the reduction in space complexity, since the reduction in computational complexity is dampened by the processing power of a GPU.

Using the vectorized formulation allows the rapid evaluation of large samples, which is important to obtain good solutions, and allows NES to converge in time similar to the approximate algorithm presented in [13].

The optimization procedure described in this paper generalizes to optimization problems that can readily be expressed and evaluated through linear algebra. The presented approach could be used with machine learning based optimization procedures such as [2], [8], [9], e.g., to create good data-sets for supervised learning in large optimization problems.

IV. Scenario and Demo Presentation

Table I lists the parameters of our demonstration that can be set through a web-based Graphical User Interface (GUI). The parameters control three aspects: The algorithm input, the used algorithm and hyperparameters applicable to NES.

Parameters relating to the algorithm input affect the sets $\mathcal{U}$ and $\boldsymbol{\mathcal{S}}$ and the cost and utility function. The size of $\mathcal{U}$ can be controlled through the desired coverage of the population, e.g., by specifying that only cities with more than a specific number of inhabitants should be covered. The cost model affects the cost and utility function. A greenfield deployment assumes no existing infrastructure. Optimization then corresponds to the minimization of the number of placed edge clouds. Potential edge cloud locations with existing infrastructure, e.g., datacenters, can have a reduced cost, which influences the placement. The parameter *path scale* controls the set $\boldsymbol{\mathcal{S}}$ by scaling the euclidean distance between two locations with a random number between one and path scale.

We provide three different algorithms: The approximate algorithm from [13] and two NES implementations, one running on CPU and the other on GPU. The NES implementation on CPU is primary intended to showcase the performance boost obtained through GPU acceleration. The GUI exposes two hyperparameters for the NES algorithms: learning rate and number of samples per epoch. We will demonstrate how both parameters impact the convergence speed and solution quality.

Obtaining a result for one parameter setting takes from seconds to minutes. In the former case the optimization can be performed live. A dynamic plot is then continuously updated with the newest best solution to indicate the optimization progress. For settings which take longer than a minute precomputed results can be loaded. Live optimization takes places on a nVidia DGX station equipped with four Tesla V100 GPUs and an Intel Xeon E5-2698 CPU.

The obtained placement is illustrated on a map of the US once the optimization is finished, which is illustrated in Fig. 1. Locations that should be covered are plotted in green, edge cloud locations in orange.

References

[1] Andr R. Brodtkorb, Trond R. Hagen, and Martin L. Saetra. Graphics processing unit (GPU) programming strategies and trends in GPU computing. *JPDC*, 73(1):4 – 13, 2013.

[2] Andreas Blenk et al. In *Big-DAMA '17*, pages 19–24, New York, NY, USA, 2017. ACM.

[3] Chao Hu et al. Mobile Edge Computing A key technology towards 5g. White Paper 11, European Telecommunications Standards Institute, 2015.

[4] Christian Schulz et al. GPU computing in discrete optimization. Part II: Survey focused on routing problems. *EURO Journal on Transportation and Logistics*, 2(1):159–186, May 2013.

[5] D. Wierstra et al. Natural Evolution Strategies. *Journal of Machine Learning Research*, 15:949–980, 2014.

[6] Dario Sabella et al. Toward fully connected vehicles: Edge computing for advanced automotive communications. White Paper, 5GAA Automotive Association, 2017.

[7] Gerhard Fettweis et al. The Tactile Internet. Technical report, ITUT Technology Watch, 2014.

[8] Johannes Zerwas et al. Ahab: Data-driven virtual cluster hunting. In IFIP, editor, *IFIP Networking 2018*, Zurich, Switzerland, May 2018.

[9] Patrick Kalmbach et al. Empowering self-driving networks. In *SelfDN 2018*, pages 8–14, New York, NY, USA, 2018. ACM.

[10] Sami Kekki et al. MEC in 5g networks. Technical report, European Telecommunications Standards Institute, June 2018.

[11] Younghwan Go et al. APUNet: Revitalizing GPU as Packet Processing Accelerator. In *NSDI 17*, pages 83–96, Boston, MA, 2017. USENIX Association.

[12] Angel Fernando Kuri-Morales and Jess Gutirrez-Garca. Penalty Function Methods for Constrained Optimization with Genetic Algorithms: A Statistical Analysis. In Carlos A. Coello Coello, Alvaro de Albornoz, Luis Enrique Sucar, and Osvaldo Cair Battistutti, editors, *MICAI 2002*, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[13] Petr Slavk. A Tight Analysis of the Greedy Algorithm for Set Cover. In *STOC '96*, pages 435–441, New York, NY, USA, 1996. ACM.