

Distributed Placement of Machine Learning Operators for IoT applications spanning Edge and Cloud Resources

Tarek Elgamal
University of Illinois at Urbana-Champaign
telgama2@illinois.com

Klara Nahrstedt
University of Illinois at Urbana-Champaign
klara@illinois.edu

Atul Sandur
University of Illinois at Urbana-Champaign
sandur2@illinois.com

Gul Agha
University of Illinois at Urbana-Champaign
agha@illinois.edu

ABSTRACT

Internet of Things (IoT) applications generate massive amounts of real-time data that are typically processed for carrying out complex tasks such as vision and speech processing. Owners of such data strive to make predictions/inference from large streams of complex data such as video feeds, often using pre-trained neural network models. A typical deployment of IoT applications includes edge devices to acquire the input data and provide processing and storage capacity closer to the location where the data is captured. This can obviate the need to move all the data/processing to a remote cloud service. However, since edge devices are limited in computational capacity, we need to determine the optimal placement of operations between edge and remote cloud resources to optimize the performance of neural network model inference. In this paper we propose an algorithm to decide the partitioning of neural network operations across edge and cloud resources. Our algorithm is linear in the number of operations (m) of the neural network model and the overall complexity is $O(m \cdot (L!) \cdot L)$, where L is the number of resources (typical deployments include one edge resource and the cloud, in which case L is 2).

1 INTRODUCTION

The continued growth of Internet of Things (IoT) applications requires the ability to extract insights from massive amounts of data streams observed at real-time. The data could be collected from multiple input sources such as: (1) surveillance cameras for traffic monitoring and intrusion detection, (2) wearable cameras and medical devices for healthcare, (3) sensors deployed on bridges and buildings in smart cities.

Due to limited computational capability of IoT devices that collect such data, the conventional approach of performing analytics on data streams is to send them to the cloud and leverage its powerful resources to execute analytics remotely. However, given the tremendous amount of data transfer often required (e.g., when video data is collected), the latency and bandwidth requirements become very high. This calls for an alternative paradigm in which ML models are trained in the cloud and then deployed on the edge [4] for near real-time inference (i.e. execution of pre-trained ML model operations).

ML models consist of a large number of dependent operations modeled as dataflow graphs [1]. Each operation describes computation on the incoming data such as matrix multiplication or convolution. A key challenge is to automatically decide how to partition operations by assigning them to edge and cloud resources, in order to minimize the overall completion time of the entire graph of operations. Our work addresses this problem and makes the following contributions:

- (1) We propose a general model for scheduling of dataflow graphs that operate on a data stream of observations/inputs.

The model considers computation, communication and queuing delays, and captures pipeline parallelism for estimating execution time. Such pipelining arises due to multiple inputs being processed concurrently by different operators in the graph.

- (2) We draw an analogy between operator placement problem and Matrix Chain Ordering Problem (MCOP) which allows us to design a dynamic programming algorithm to solve the operator placement in $O(m \cdot (L!) \cdot L)$ complexity, where m is the number of operators and L is the number of resources.

Other approaches based on reinforcement learning (RL) techniques [3] have been proposed to solve the operator placement problem and shown to be effective for static resource configurations. However, our approach targets dynamic environments in which connectivity between edge devices change rapidly.

2 PROBLEM DEFINITION

Given a dataflow graph G with a set of vertices $G_o = \{o_1, o_2, o_3, \dots, o_m\}$, each vertex in the graph corresponds to one operator. Each operator processes n inputs, where an input is one unit of data defined by the application (e.g., tuple, video frame). Links between vertices correspond to dataflow dependency between operators. If G has a link from operator o_1 to operator o_2 , then each input n_i has to be processed first by o_1 before it is transmitted to o_2 . Let R be a resource graph R with a set of vertices $R_v = \{r_1, r_2, r_3, \dots, r_L\}$. Each resource represents an edge device (such as smartphone), intermediate server, or the cloud.

Each operator o_i has an execution cost when placed on any of the p resources in R_v . The execution cost for o_i is represented as $(t_{o_i,1}, \dots, t_{o_i,L})$ and the intermediate data is represented as d_i . Each resource r_k is connected to some other resource r_l with connection speed s_{kl} . The operator placement problem is defined as the mapping M of operators to resources such that the completion time of the last operator o_m is minimized. We define M as $\{M : G_o \rightarrow R_v \implies M(o_i) = \{r_l \in R_v\}\}$

3 PROPOSED APPROACH

The key idea in our approach is to leverage the fact that each operator in a dataflow graph G has to process n inputs. Hence, two dependent operators can be concurrently processing inputs in a pipeline fashion. For example, the first input that was processed by operator o_1 , can now be processed by o_2 while the second input is being processed in parallel by o_1 . This pipelining behavior is only applicable if there are enough resources to execute multiple operators in parallel. For simplicity, we assume that each resource has one thread or unit of computation, however, the method can be generalized to resources with multiple threads or containers. The pipelining behavior can reduce the computation time through parallelism however, it introduces a transmission latency when two

- $(o_{1,E}, (o_{2,E}, (o_{3,E}))$
- $(o_{1,E}, (o_{2,E}, o_{3,C}))$
- $(o_{1,E}, o_{2,C}), (o_{3,E})$
- $(o_{1,E}, o_{2,C}), (o_{3,C})$
- $(o_{1,C}, o_{2,E}), (o_{3,E})$
- $(o_{1,C}, o_{2,E}), (o_{3,C})$
- $(o_{1,C}), (o_{2,C}, o_{3,E})$
- $(o_{1,C}), (o_{2,C}), (o_{3,C})$

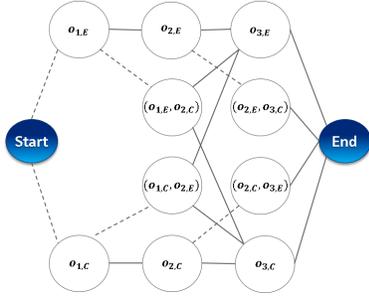


Figure 1: Different operator placements

Figure 2: Corresponding placement graph. The difference between dashed and solid edges, along with other graph details are discussed in section 3.2

dependent operators are placed on different resources. Hence, the problem becomes how to choose which operators to run in parallel in order to reduce the overall computation and communication time. Consider the following example:

3.1 Illustrative Example

Assume that we have a sequence of three operators o_1, o_2, o_3 and two different resources: (1) an edge device E , (2) a cloud VM C . The possible options to place our operators between the two resources are shown in Figure 1. Each line in the figure shows a different possibility for operator placement. The subscript $(1, E)$ means that operator 1 is placed on device E and operators inside parentheses show that they are executed in a pipelined manner.

Using the representation in Figure 1 for the placement of operators on resources helps us draw an analogy between operator placement and the matrix chain ordering problem (MCOP) [2]. Recall that in MCOP, the problem is to find the most efficient way to multiply a given sequence of matrices together. There exists a dynamic programming algorithm for solving MCOP in polynomial time. Section 3.2 describes how this analogy can be similarly used to reduce the complexity of solving operator placement problem from exponential to polynomial time in the number of operators.

3.2 Solution

In order to represent the solutions in Figure 1, we construct the *placement graph* shown in Figure 2. The dashed edges in placement graph show connections between operators running in parallel (i.e., the ones collocated inside the parentheses), however the solid edges are between operators at the boundaries of two sets of parentheses. The dashed edges have resource cost because going from one dashed edge to another means that you are executing the next operator on a new thread or a new server which results in increased overall resource consumption. The solid edges are the ones that have computation and communication costs (see Figure 2). The intuition for reduction in placement problem complexity is that the paths $[(o_{1,E}, o_{2,C}), (o_{3,C})]$ and $[(o_{1,C}, o_{2,E}), (o_{3,C})]$ have a common solid edge going from $(o_{3,C})$ to the *END* node. There are however two separate edges going to $(o_{3,C})$, which help differentiate between the two paths, one edge has the cost of path $(o_{1,E}, o_{2,C})$ and the other has the cost of path $(o_{1,C}, o_{2,E})$.

We note that the shortest path between start and end nodes is the solution of the operator placement problem. For example, if the shortest path is $[(o_{1,E}, o_{2,C}), (o_{3,C})]$, then the placement will be o_1 on device E while o_2 and o_3 are on the device C .

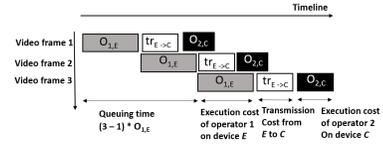


Figure 3: Execution of tasks $o_{1,E}$ and $o_{2,C}$ in parallel in different devices.

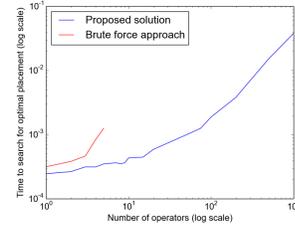


Figure 4: Scalability comparison between our proposed approach and brute force approach.

Limitations: Our approach has an implicit assumption that when operators $(o_{1,E}, o_{2,C})$ are executed concurrently, they will fully utilize available resources and that's why $(o_{3,C})$ has to wait for device C resources to be free. However, this may not be the case; in the future, we plan to explore the optimal number of inputs that can be processed concurrently by o_2 to fully utilize device C .

3.3 Cost Calculation:

We describe how to calculate the overall computation and communication cost of operators inside the parentheses in figure 2, such as cost of $(o_{1,E}, o_{2,C})$ in the path $[(o_{1,E}, o_{2,C}), (o_{3,C})]$. Figure 3 illustrates how $o_{1,E}$ and $o_{2,C}$ operators are processing inputs in a pipeline fashion. Let's assume we have 3 video frames and two operators o_1 and o_2 that process each video frame in sequence. The first operator is placed on device E and the second on device C . Frame 1 is first provided to operator o_1 and then transmitted to device C where operator o_2 can be applied to it, while operator o_1 is being applied to frame 2 concurrently. The overall cost is the time for the last video frame to finish execution at o_2 . The cost of the n^{th} video frame is given by:

$$Cost((o_{1,E}, o_{2,C})) = (n-1)t_{o_{1,E}} + t_{o_{1,E}} + tr\{E \rightarrow C\} + t_{o_{2,C}}$$

where $tr\{E \rightarrow C\}$ is the cost of transmitting data from E to C .

4 EVALUATION

We evaluate the scalability of our approach in terms of time to search for a placement that minimizes the operators' execution time. We compare our solution with a brute force (BF) approach that searches over all possible operator placements. BF searches over $O(r^m)$ solutions, where m is the number of operators and r is the number of resources. We vary the number of operators in the input graph. Both approaches have the same placement result for instances in which BF finished in reasonable time (up to 5 operators). Moreover, we show the scalability of our approach in Figure 4. Results show that the proposed algorithm linearly scales in the number of operators as opposed to an exponential increase in the running time of BF approach.

REFERENCES

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Manuál, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. (2015). <http://download.tensorflow.org/paper/whitepaper2015.pdf>
- [2] Phillip G. Bradford, Gregory J. E. Rawlins, and Gregory E. Shannon. 1998. Efficient Matrix Chain Ordering in Polylog Time. *SLAM J. Comput.* 27, 2 (1998), 466–490. DOI : <http://dx.doi.org/10.1137/S0097539794270698>
- [3] Azalia Mirhoseini, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Benoit Steiner, Yuefeng Zhou, Naveen Kumar, Rasmus Larsen, and Jeff Dean. 2017. Device Placement Optimization with Reinforcement Learning. <https://arxiv.org/abs/1706.04972>
- [4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (Oct 2016), 637–646. DOI : <http://dx.doi.org/10.1109/JIOT.2016.2579198>