Late Breaking Results:

Robustly Executing DNNs in IoT Systems Using Coded Distributed Computing[*]

Ramyad Hadidi
Georgia Tech

Jiashen Cao
Georgia Tech

Michael S. Ryoo
Google Brain

Hyesoon Kim
Georgia Tech

ABSTRACT

Internet of Things (IoT) devices have access to an abundance of raw data for processing. With deep neural networks (DNNs), not only the demand for the computing power of IoT devices is increasing, but also privacy concerns are motivating the importance of close-to-edge computation. DNN execution by distributing its computation is common in IoT systems. However, managing unstable latencies in a network and intermittent failures are serious challenges. Our work provides robustness and close-to-zero recovery latency by adapting coded distributed computing (CDC). We analyze robust execution on a mesh of Raspberry Pis by studying four DNNs.

ACM Reference Format:
Ramyad Hadidi, Jiashen Cao, Michael S. Ryoo, and Hyesoon Kim. 2019. Late Breaking Results: Robustly Executing DNNs in IoT Systems Using Coded Distributed Computing. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3316781.3322474

1 MOTIVATION

With ubiquitous wireless networks and the availability of embedded processors, Internet of Things (IoT) devices are rapidly gaining ground. These devices have access to a variety of raw data that needs to be elucidated with tight real-time and time-sensitive constraints. Moreover, with the fast-paced advancement of deep neural networks (DNNs) not only the use cases for IoT devices are increasing, but also the demanded computational power from resource-hungry DNN-based applications are escalating. Furthermore, processing raw data on IoT devices reduces the dependency of the system on the network connectivity and cloud services while protecting users' private data. Therefore, with the proliferation of IoT devices, a new computing paradigm by using these devices has emerged as edge or fog computing, in which the data processing is performed at the edge of the network. Since IoT devices have limited resources to execute DNN models several application-level techniques such as weight pruning, resource partitioning, quantization and low-precision inference, and binarizing weights are introduced to reduce the computation load of DNNs. Moreover, several studies also examined the distributed execution of DNNs on edge devices while utilizing the mentioned technique. However, since the distributed execution is susceptible to any kind of failures, from short disconnectivity to losing a device, we may lose valuable time-sensitive information. This fact, combined with the limited number of devices and high probability of failures these systems, necessitates a robust method for tolerating failures.

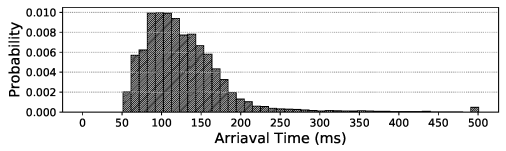

Figure 1: Arrival times in a four-device IoT system.

In this short paper, we propose a coded distributed computing (CDC) recovery method that enables distributed DNN models on IoT systems to tolerate failures (i.e., not lose time-sensitive and real-time information) and mitigate straggler problem (i.e., reduce average response time by not waiting for straggler data). Our work introduces robustness by *adapting* coded distributed computing (CDC) [4] and provides a close-to-zero recovery latency by trading off robustness with new computations. (CDC methods reduce latency in a distributed system with map-reduce workloads by increasing computation per node). Note that a critical difference of executing DNNs in IoT systems, compared to high performance computing (HPC) domain, is that in contrast with conventional multi-batch inferencing of DNNs, in IoT systems, single-batch inferencing is necessary. This is because the number of requests is limited and understating the data is time sensitive. Therefore, the introduced computations are derived by studying distribution techniques for single-batch inference in DNNs, or model-parallelism techniques. The introduced computations are similar in nature to those of DNNs, so it can easily be balanced among IoT devices using the same techniques, which is essential for an efficient system.

To illustrate unreliability in the communication latency of IoT (i.e., straggler problem), Figure 1 shows a histogram of the arrival times for data packets in a four-device IoT system with Raspberry Pi 3s that use a WiFi network. A master device sends data packets to three other devices, each of which performs the computation for a fully-connected layer of size 2048. The measured time for the computation of a fully-connected layer of size 2048 on a single device is 50 ms. As seen, around 34% of the arrival times is within 100 ms, and 42% is within 150 ms. So, even after 2x the computation time, around 34% of the packets have not arrived yet. Likewise, to understand how failures are destructive in an IoT system, we

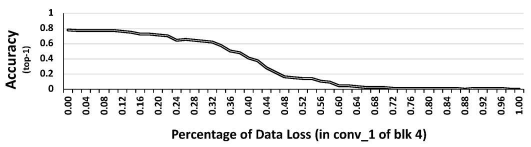

Figure 2: Accuracy drop of Inception v3 with data loss.

[*]Supported by NSF CSR 1815047. All correspondence to rhadidi@gatech.edu.

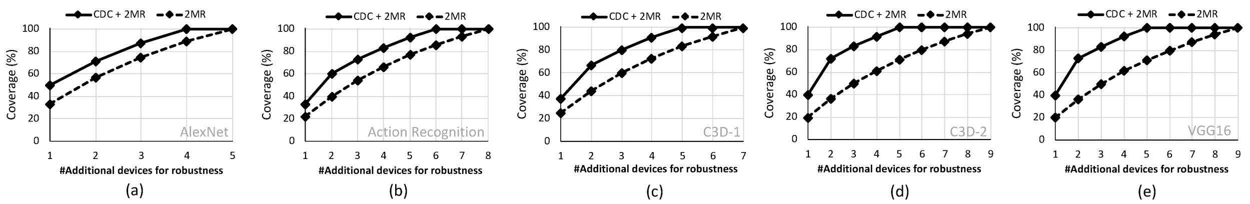

Figure 3: Coverage of one failure with the number of additional devices for robustness, 2MR and CDC+2MR, with equal failure probability/device: (a) AlexNet on 5, (b) Action recognition [5] on 8, (c) C3D [1] on 7 (d) C3D on 9, and (e) VGG16 [2] on 8 devices.

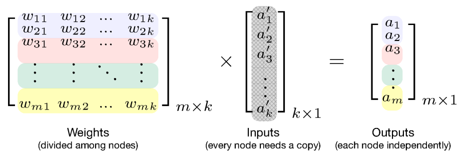

Figure 4: Output-splitting method distribution.

perform another experiment. In this experiment, the result of which is shown in Figure 2, the accuracy drop in Inception v3 is shown when some part of data in only one layer is lost. Although DNN models contain many layers, as seen, for data loss of $> 30\%$ the accuracy drop is significant and with data loss of $> 50\%$ the accuracy is close to zero. In fact, complex DNN models are more sensitive to data loss because they rely more on the nuances in data.

2 PROPOSED APPROACH & FINDINGS

To execute DNNs on an IoT system, model-parallelism methods, which reduce the amount of work and the memory footprint of a task are utilized [1–3]. The two extremes of model-parallelism are input and output splittings. In output splitting, creating outputs is divided among the devices. Therefore, for each activation, its whole computation is performed on a device. In input splitting, each device computes a part of the entire output. The input is split and each device computes all parts of the output that are dependent on their particular part of the input. Since the computation of fully-connected layers are basically a matrix multiplication, these splitting methods are easily applied on these layers. The convolution layer, which applies the same set of weights (i.e., filters) to patches of input, can be also split with similar techniques. To do so, similar to machine learning frameworks, we transform the computations as a single matrix-matrix multiplication (GEMM). The essence of the transformation is to unroll the input patches (a 3D matrix) and filters (a 4D matrix) in 2D in a way that a single matrix-matrix multiplication produces the unrolled version of the output in 2D. Figure 4 illustrates how output splitting affects weight and output matrices for an example with four devices. Since each device calculates a set of separate outputs, the output matrix is created separately by each device and concatenated later. Such separation in output generation divides the weight matrix along the y-axis, which has a one-by-one relationship with the output matrix. Since input splitting does not have this property, it is not analyzed here.

Now, we present a basic example of CDC for robustness. Consider a fully-connected layer with two input (a') and output element (a). Assume we are distributing this computation so that one device performs the computations of one output element (i.e., output splitting). Now, by adding a row to the weight matrix with the value of $[w_{11} + w_{21} \; w_{12} + w_{22}]$, we can create a summation of two outputs, or $a_1 + a_2$. Therefore, with such addition, we have:

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{11} + w_{21} & w_{12} + w_{22} \end{bmatrix} \times \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_1 + a_2 \end{bmatrix}. \tag{1}$$

Since the summation of the weights can be done *offline* and does

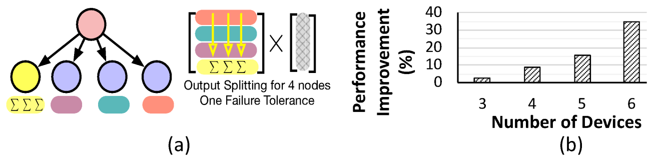

Figure 5: (a) New CDC node and its weight (b) Straggler mitigation performance improvement on AlexNet from [2].

not change afterward, we can write $w_{11} + w_{21}$ as $w^{cdc}_{:1}$ and $w_{12} + w_{22}$ as $w^{cdc}_{:2}$. Newly added weights to the weight matrix are the column sums of the weight matrix (Figure 5a). This new row in the weight matrix creates a new output that is the sum of a_1 and a_2. By adding another device that performs this computation, we can guarantee to recover from missing one value with only a subtraction. In fact, the subtraction of two local values is much faster than recovery. Another benefit of this technique is that this extra computation in nature is similar to the computation of a_1 and a_2; thus, the distribution of its computation follows the same rules. In reality, however, a device computes hundreds of output elements. To extend this technique to multiple outputs per device we basically need to perform our extra calculation based on below weight matrix,

$$\begin{bmatrix} w_{11}+w_{(\frac{m}{2}+1)1} & w_{12}+w_{(\frac{m}{2}+1)2} & \cdots & w_{1k}+w_{(\frac{m}{2}+1)k} \\ w_{21}+w_{(\frac{m}{2}+2)1} & w_{22}+w_{(\frac{m}{2}+2)2} & \cdots & w_{2k}+w_{(\frac{m}{2}+2)k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\frac{m}{2}1}+w_{m1} & w_{\frac{m}{2}2}+w_{m2} & \cdots & w_{\frac{m}{2}k}+w_{mk} \end{bmatrix}_{\frac{m}{2}\times k}. \tag{2}$$

in which k and m are the total number of input and out elements, respectively. In fact, this extended weight matrix, has all the properties of our basic example. Furthermore, with another extension, we can tolerate more failures. In short, by adding new devices that perform computations based on the summation of some rows of weights instead of all of them, we can increase robustness to multiple failures. To see how our method covers whole system, we study several distributed implementations [1, 2, 5] with tolerance to one failure with 2MR-only (N-modular redundancy with $N = 2$) and CDC+2MR. As seen, since CDC requires fewer devices than 2MR to cover the devices with model parallelism, the number of additional devices for full coverage for CDC+2MR is smaller than that of 2MR. The amount of difference depends on how layers are distributed with model parallelism and how many devices are used per layer. Figure 5b shows the performance benefits of straggler mitigation using our method for an AlexNet distributed system.

REFERENCES

[1] Hadidi, R., et al. Collaborative execution of deep neural networks on internet of things device. *arXiv preprint* (2018).
[2] Hadidi, R., et al. Distributed perception by collaborative robots. *IEEE RA-L, Invited IROS 2018 3*, 4 (Oct 2018), 3709–3716.
[3] Hadidi, R., et al. Real-time image recognition using collaborative iot devices. In *ReQuEST '18 co-located with ASPLOS'18* (2018), ReQuEST '18, ACM.
[4] Li, S., et al. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Trans. Inf. Theory 64*, 1 (Jan 2018), 109–128.
[5] Ryoo, M. S., et al. Extreme low resolution activity recognition with multi-siamese embedding learning. In *AAAI'18* (Feb. 2018), IEEE.