SERF: Efficient Scheduling for Fast Deep Neural Network Serving via Judicious Parallelism

Feng Yan
University of Nevada, Reno,
Reno, NV, USA, fyan@unr.edu

Yuxiong He
Microsoft Research,
Redmond, WA, USA, yuxhe@microsoft.com

Olatunji Ruwase
Microsoft Research,
Redmond, WA, USA, olruwase@microsoft.com

Evgenia Smirni
College of William and Mary,
Williamsburg, VA, USA, esmirni@cs.wm.edu

***Abstract*—Deep neural networks (DNNs) has enabled a variety of artificial intelligence applications. These applications are backed by large DNN models running in serving mode on a cloud computing infrastructure. Given the compute-intensive nature of large DNN models, a key challenge for DNN serving systems is to minimize the request response latencies. This paper characterizes the behavior of different parallelism techniques for supporting scalable and responsive serving systems for large DNNs. We identify and model two important properties of DNN workloads: homogeneous request service demand, and interference among requests running concurrently due to cache/memory contention. These properties motivate the design of SERF, a dynamic scheduling framework that is powered by an interference-aware queueing-based analytical model. We evaluate SERF in the context of an image classification service using several well known benchmarks. The results demonstrate its accurate latency prediction and its ability to adapt to changing load conditions.**

I. Introduction

Deep Neural Network (DNN) models have recently demonstrated state-of-the-art accuracy on important yet challenging artificial intelligence tasks, such as image recognition [1], [2], [3] and captioning [4], [5], video classification [6], [7] and captioning [8], speech recognition [9], [10], and text processing [11]. These advancements by DNNs have enabled a variety of new applications, including personal digital assistants [12], real-time natural language processing and translation [13], photo search [14] and captioning [15], drug discovery [16], and self-driving cars [17].

Many cloud service providers offer DNN services as part of their machine learning platform such as Microsoft AzureML [18] and Amazon machine learning systems [19], which provide library and runtime tools for application owners to develop and deploy their DNN applications conveniently. These platforms support the training and serving of various DNN applications. Figure 1 shows the user interface of Microsoft AzureML and click to deploy model. "Experiments" session in Figure 1 is corresponding to the *training* phase, where application owners specify the neural network structure, algorithms, and data to train their DNN models. Once trained, these models can be deployed instantly on the cloud in a *serving*

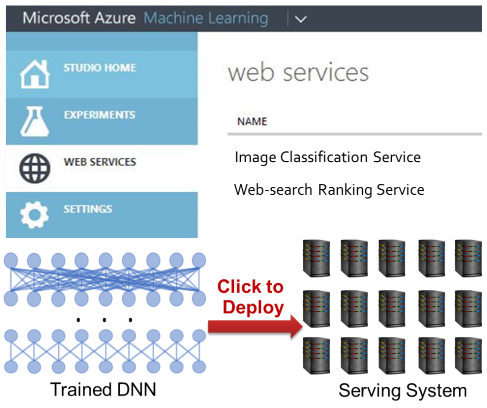


Fig. 1. Microsoft AzureML interface.

mode to process application inputs, such as images, voice commands, speech segments, handwritten text, see the "Web Services" session in Figure 1. Our paper focuses on DNN serving systems, revealing the challenges and opportunities to support fast deployment of responsive and scalable DNN applications.

DNN serving platforms must satisfy the following two requirements. First, DNNs should offer short response time to user requests. Since DNN applications process a stream of user requests, a serving system must consistently offer fast responses to attract and retain application users. Slow responses directly degrade user experience. For example, image recognition applications [1], [2], [3] take photos or even real-time camera streams as input requests and send back classification results. Since it is very similar to traditional query service, users usually expect low latency responses and may switch to another service provider if the perceived latency is high [20], [21]. Second, DNNs should support fast deployment of applications. Once DNN models are trained and ready for deployment, the serving system should make the application available to accept online user requests within a few minutes [22]. No one would use a platform that takes hours or even days to deploy or update their applications.

DNN models that achieve the best accuracy on the most challenging tasks (e.g., image, speech, etc.) are often very

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00 ©2016 IEEE

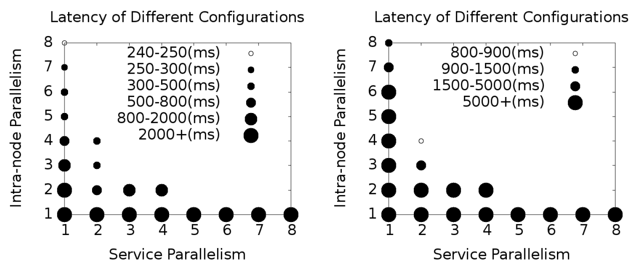


Fig. 2. Latency under low load (left plot) and high load (right plot) using different configurations (inter-node parallelism is set to 1) for ImageNet-22K.

large (containing billions of neural connections) and require significant compute cycles and memory bandwidth to serve each request [1], [3], [16]. Such large models may take seconds or even minutes to process each user request if executed in a sequential fashion on a single server. Parallel computation is a promising approach to improve the response time of DNN serving applications. In this paper, we consider three ways of exploiting hardware parallelism within and across machines for DNN serving systems. First, parallel hardware threads on a machine (e.g., chip-multiprocessor (CMP) cores) can be used for parallel processing each request to reduce response time (*intra-node parallelism*). Second, the parallel hardware threads on a machine could alternatively be used for concurrent processing of different requests to reduce waiting time (*service parallelism*). Third, the model could be partitioned across multiple machines to leverage the aggregate compute cycles and memory bandwidth for faster processing of each request (*inter-node parallelism*).

Finding good parallelism configurations to minimize DNN serving latency is important but challenging. Applying parallelism degrees blindly could harm performance. For example, service parallelism may increase memory system contention to the point of prolonging request processing time; inter-node parallelism may prolong request processing if the cross-machine communication overhead exceeds the computation speedup. Figure 2 shows the latency of serving the ImageNet-22K workload [23] under different combinations of service and intra-node parallelisms on an 8-core machine (refer to Section V-A for detailed experimental setup). The left and right scatter plots represent low and high load conditions (request arrival rate) respectively. Each point represents one parallel configuration, and the size of the point indicates its latency value. The figure demonstrates that: (1) Many parallel configurations are possible, even with only 8 cores and without considering inter-node parallelism. (2) The latency difference between the best parallel configuration and the worst parallel configuration can be significant, i.e., by orders of magnitudes. This gap grows further under higher loads. (3) The latency values and the best parallel configuration changes as a function of the load.

In order to find best parallelism configurations among many candidates, we need to quantify and compare their latency impact. One could conduct exhaustive profiling of the performance of all combinations of parallelism configurations and expected load levels. This can be very expensive and may take hours or even days. Moreover, this profiling cost repeats when models are updated. Such a method is impractical as the online services require fast deployment within a few minutes. An alternative solution is to use analytical modeling to predict request latencies under different configurations and load levels. However, the effectiveness of a parallelism technique for a DNN depends on many factors such as neural network characteristics, hardware, and the combined impact of memory contention and communication overhead, which make accurate latency prediction difficult. Therefore, neither exhaustive profiling nor analytical modeling offer a practical solution to find the best parallel configuration in a timely manner.

We present SERF (serving deep learning systems fast), a scheduling framework that employs a hybrid approach by combining lightweight profiling with queueing-based analytical modelling to quickly identify best parallel configurations for any given load. SERF needs to answer two important questions: (i) what should be profiled for accuracy but can be also profiled quickly, and (ii) how to model and predict request latency? To answer these questions, we characterize and identify two distinctive properties of DNNs: (1) The DNN service time is difficult to model accurately, but can be measured efficiently. In particular, DNN requests are homogeneous, i.e., when running under the same degree of parallelism, the service time is deterministic. This property empowers lightweight profiling, i.e., only the average service time information needs to be collected rather than more complex information such as distributions. (2) There is interference among concurrent running requests due to cache/memory contention: a request may take longer to execute in the presence of other concurrent requests even when these requests are using different cores. In this paper, we develop an interference-aware queuing-based analytical model that takes as input the service time profiling information and accurately predicts request latency under different loads. SERF adopts this hybrid approach to identify the best configuration for any given load and deploys a dynamic scheduler that adapts to load changes online nearly instantly, achieving the benefits of both empirical and analytical methods.

We implement SERF in the context of an image classification service based on the image classification module of the Adam distributed deep learning framework [3]. We stress that SERF is not limited to the Adam architecture, but also applicable to serving systems based on other DNN frameworks (e.g., Caffe [24], Theano [25], and Torch7 [26]) as similar parallelism decisions and configuration knobs are also available there. We conduct vast experiments by running several state-of-the-art classification benchmarks, including *ImageNet* [23] and CIFAR [2]. We show that our prediction model achieves high accuracy: the average error is less than 4% comparing to measurement results. SERF always correctly identifies best parallel configurations under a variety of benchmarks and system loads. Moreover, compared to using static parallel configurations, SERF swiftly identifies and switches to the best configuration, reducing request latency under various loads. Compared to exhaustive profiling, SERF adapts three orders of magnitude faster under dynamic and ever-changing environments, significantly reducing application deployment time.

II. Background

DNNs consist of large numbers of neurons with multiple inputs and a single output called an activation. Neurons are connected hierarchically, layer by layer, with the activations of neurons in layer $l-1$ serving as inputs to neurons in layer l. This deep hierarchical structure enables DNNs to learn complex tasks, such as image recognition, speech recognition, and text processing.

A DNN service platform supports training and serving. DNN training is offline batch processing that uses learning algorithms, such as stochastic gradient descent (SGD) [27] and labeled training data to tune the neural network parameters for a specific task. DNN serving is instead interactive processing requiring fast response per request, e.g., within 200 - 300 milliseconds, even for challenging large-scale models like ImageNet-22K. It deploys the trained DNN models in serving mode to answer user requests, e.g., for a dog recognition application, a user request provides a dog image as input and receives the type of the dog as output. The response time of a request is the sum of its service time (execution time) and waiting time. An important common performance metric for interactive workloads is the average request response time (average latency), which we adopt in our work.

In DNN serving, each user input, which we refer to as a request, is evaluated layer by layer in a feed-forward manner where the output of a layer $l-1$ becomes the input of layer l. More specifically, define a_i as the activation (output) of neuron i in layer l. The value of a_i is computed as a function of its J inputs from neurons in the preceding layer $l-1$ as follows:

$$a_i = f\left(\left(\sum_{j=1}^{J} w_{ij} \times a_j\right) + b_i\right), \tag{1}$$

where w_{ij} is the weight associated with the connection between neuron i at layer l and neuron j at layer $l-1$, and b_i is the bias term associated with neuron i. The activation function f, associated with all neurons in the network, is a pre-defined non-linear function, typically a sigmoid or hyperbolic tangent. Therefore, for a given request, its main computation at each layer l is a matrix-vector multiplication of the weight of the layer with the activation vector from layer $l-1$ (or the input vector if $l=0$).

Inter-node, intra-node, and service-level parallelisms are well-supported among various DNN models and applications [1], [3], [28]. Inter-node parallelism partitions the neural network across multiple node/machines, with activations of neural connections that cross node/machine boundaries being exchanged as network messages. Intra-node parallelism uses multi-threading to parallelize the feed-forward evaluation of each input image using multiple cores. As the computation at each DNN layer is simply a matrix-vector multiplication, it can be easily parallelized using parallel libraries such as OpenMP [29] or TBB [30] by employing a parallel for loop. Service-level parallelism is essentially admission control that limits the maximum number of concurrent running requests. We define a *parallelism configuration* as a combination of the intra-node parallelism degree, inter-node parallelism degree, and maximum allowed service parallelism degree. Note the service parallelism is defined as a maximum value instead of the exact value due to the random request arrival process, e.g., at certain moments, the system may have less requests than the defined service parallelism degree.

III. Workload Characterization

In this section, we present comprehensive workload characterization that shows the opportunities and challenges of using the various parallelism techniques to reduce DNN serving latency, as well as their implications on the design of SERF. We make four key observations: (1) Parallelism impacts service time in complex ways, making it difficult to model service times without workload profiling. (2) DNN workloads have homogeneous requests, i.e., service times under the same parallelism degree exhibit little variance, which allows SERF to measure request service time with affordable profiling cost. (3) DNN workloads exhibit interference among concurrent running requests, which motivates a new model and solution of SERF. (4) DNN workloads show load-dependent behavior, which indicates the importance of accurate latency estimation and parallel configuration adaptation according to system load.

We present workload characterization results of two well-known image classification benchmarks, CIFAR-10 [2] and ImageNet-22K [23], on servers using Intel Xeon E5-2450 processors. Each processor has 8 cores, with private 32KB L1 and 256KB L2 cache, and shared 20MB L3 cache. The detailed experimental set up for both workloads and hardware is provided in Section V.

A. Impact of parallelism on service time

Modeling the impact of parallelism on DNN serving without workload profiling is challenging because parallelism has complex effects on the computation and communication components of request service time, as shown in Figures 3 and 4.

Figure 3 shows the DNN request service speedup for different degrees of intra-node, inter-node, and service parallelism. For intra-node parallelism, the speedup is close to linear up to 3 cores, but slows down beyond 4 cores. This effect is due to the limited memory bandwidth. When the total memory bandwidth demands are close to or exceed the available bandwidth, the bandwidth per core reduces, decreasing speedup. For inter-node parallelism, increasing the parallelism degree from 1 to 2 yields a 2X service time speedup because the computation time, which is dominant, is halved, while communication time grows marginally; increasing from 2 to 4 results in super-linear speedup due to caching effects, as the working set fits in the L3 cache; increasing from 4 to 8 results in smaller speedup increase as communication starts to dominate service time. For service parallelism, parallelism degrees > 2 result in increased

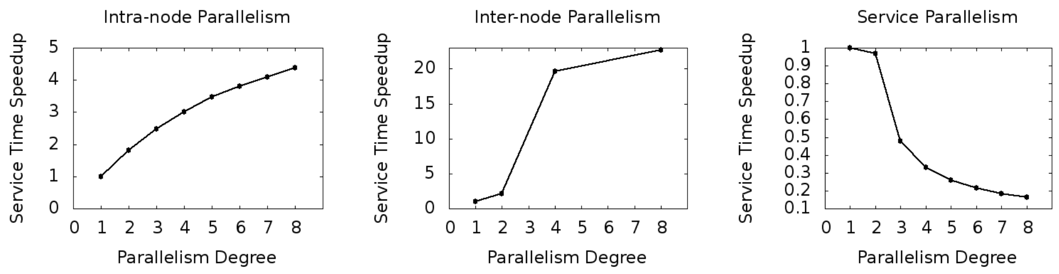


Fig. 3. Service time comparison under different parallelism techniques using ImageNet-22K. Each plot reports the speedup when increase the degree at only one parallelism (fix the other two parallelisms).

service time due to memory interference among concurrently serviced requests. These results are indicative of the impact of different parallelism on service time. Speedups can vary a lot, depending on many factors, including DNN size, the ratio of computation and communication, cache size, memory bandwidth.

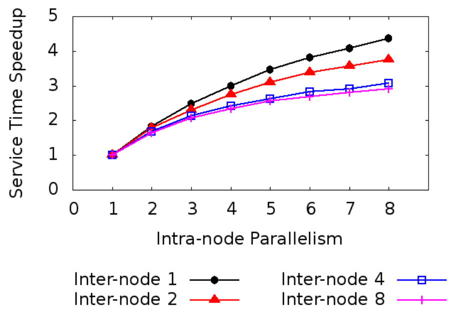


Fig. 4. Relationship between inter-node and intra-node parallelism using ImageNet-22K.

Figure 4 demonstrates the relationship between inter-node and intra-node parallelism: the results indicate that the degree of one parallelism technique can affect the behavior of another. More precisely, intra-node parallelism speedup depends on the degree of inter-node parallelism: speedup reduces with larger inter-node parallelism. This is because communication time is increasingly the dominant portion of service time with larger degrees of inter-node parallelism, therefore the computation time improvements of intra-node parallelism become less important to overall service time.

In summary, since parallelism efficiency depends on various factors relating to workload and hardware properties and since one parallelism technique can affect the behavior of others, it is difficult to accurately model service time. SERF circumvents this by incorporating workload profiling to predict request service time.

B. *Homogenous requests*

We observe that for a given *parallelism degree tuple*[1], defined as (service parallelism degree, inter-node parallelism degree, intra-node parallelism degree), the service times of DNN requests exhibit very little variance because the same amount of computation and communication is performed for each request. Thus, we refer to DNN requests as being homogenous. Figure 5 shows two examples corresponding to two representative cases of parallelism degrees. The first example as shown in the left plot of Figure 5 is with parallelism degree tuple of (2, 1, 4), where the majority of requests are in the range of 330ms to 340ms and the SCV (squared coefficient of variation) is only 0.03. The second example as shown in the right plot of Figure 5 is under parallelism (4, 4, 2), where most requests are in the range of 130ms to 160ms with the SCV of 0.09. The slightly larger variance can be attributed to variations in the cross-machine communication delays caused by inter-node parallelism. The magnitude of these variations is consistent with what is normally expected in computer communication systems while running a request multiple times [31].

This unique property of homogeneous requests for DNN workloads empowers lightweight profiling: the cost of measuring the service time is low, i.e., for a given parallelism degree tuple, running one or a few input requests is sufficient. In comparison, many other online services have requests with heterogeneous demands [32], [33] and require to execute many more input samples to collect service time distributions.

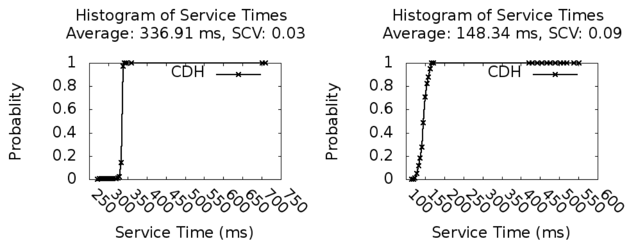


Fig. 5. CDH (Cumulative Data Histogram) of service times. The left plot is with parallelism degree tuple (2, 1, 4) and the right plot is with (4, 4, 2).

C. *Interference among concurrent requests*

For small DNNs like CIFAR-10 (the left plot of Figure 6), request service time remains almost constant when running requests concurrently under different service parallelism degrees, because there is little interference among requests due to cache/memory contention. The interference becomes more obvious for large DNNs. The right plot in Figure 6 shows the request service time of ImageNet-22K when running different number of requests. It is clear that when running more than

[1]Note that parallelism degree tuple is different from parallelism configuration. In parallelism degree tuple, each parallelism is set exactly to the degree value while in parallelism configuration, max service parallelism is an admission policy that defines the maximum allowed degree of service parallelism.

2 requests concurrently, the interference becomes severe. To explain performance interference, it is important to understand the working set of DNN serving that comprises activations and weights of the neural connections (the core operation is a matrix-vector multiplication of the weight matrix and the activation vector, see Eq. 1). Activations are derived from request input, while weights represent the model parameters and are shared by all requests. When there are no more than 2 concurrent requests, the working sets of both fit into the L3 cache. If more than three requests run concurrently, then the footprint of activations increases and the aggregate working set no longer fits in the L3 cache, resulting in more L3 cache misses, thus prolonging the request service time. This is also why large DNNs like ImageNet-22K are more likely to have interference than small ones, such as CIFAR-10.

Interference makes modeling average service time and waiting time for a given parallelism configuration much more challenging. In particular, under the same parallelism configuration, the number of running requests can vary from 0 to the maximum service parallelism of the configuration. Therefore, the service time of a particular request depends on the number of concurrent running requests at the moment of its execution, and the average service time depends on the probability distribution of the concurrency levels. The waiting time estimation is even more complex. The existing queueing and scheduling models [34] are no longer applicable as they assume independence among requests: request service time remains constant regardless of the number of concurrent requests. This property of DNN motivates us to develop new model and solution of SERF to accurately model the waiting time and latency impact of interference.

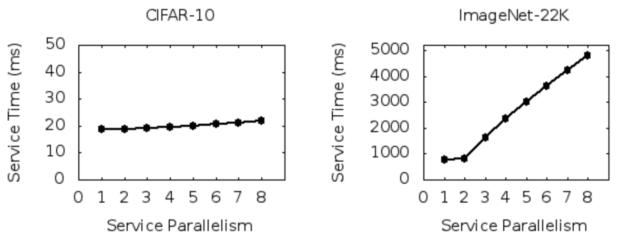


Fig. 6. Service time comparison with different number of concurrent requests.

D. Load-dependent Behavior

In serving systems, load (request arrival rate) changes dynamically over time. For a given parallel configuration, both request service time and waiting time could change under different loads. To illustrate the load-dependent behavior of different parallelism approaches, we use 6 distinctive configurations and conduct experiments under different load levels, see Table I.

The left plot in Figure 7 shows the service time of using the 6 configurations under different loads, the middle plot in Figure 7 shows their waiting time, and the right plot in Figure 7 shows their latency. The results demonstrate that for the same configuration, service time, waiting time, and latency can vary under different loads. Therefore, the ability to estimate the latency impact according to the load and a scheduler that can change the parallel configurations based on load are two necessary and important features.

Config.	Service	Inter-node	Intra-node
Config1	1	1	8
Config2	2	1	4
Config3	4	1	2
Config4	1	4	8
Config5	2	4	4
Config6	4	4	2

TABLE I
PARALLEL CONFIGURATIONS.

IV. SERF: A FRAMEWORK FOR DNN SERVING

In this section, we present the scheduling framework SERF. SERF applies a hybrid approach that integrates lightweight profiling and a queueing-based prediction model to find best parallel configurations for any given load (request arrival rate) effectively and efficiently, achieving the benefits of both empirical and analytical methods. We first discuss the scheduling objective and give an overview of SERF (Section IV-A). Then we answer the two important questions raised in the Introduction: (1) What should be profiled for accuracy yet can be profiled quickly (Section IV-B)? (2) How to model the rest and predict request latency (Section IV-C)? Finally, we discuss how to use the prediction results to dynamically change the parallelism configurations online with varying loads (Section IV-D).

A. Overview

Scheduling Objective. Common objectives for scheduling interactive serving systems are (1) to minimize response latency using a given amount of resources [33], [32] or (2) to minimize resource consumption while meeting latency SLO [35], [36]. Our scheduling framework supports both. Due to the interest of space, we focus on the first objective of minimizing response latency. We choose to optimize average latency because DNN requests are homogenous and have similar service time, reducing average latency also reduces the tail latency.

Framework Overview. Figure 8 presents an overview of SERF, which consists of three main modules: prediction model, profiler, and scheduler. The modules are connected by the configuration reference table, which maps different load levels (represented by request arrival rate) to their corresponding best parallel configurations. For example, at arrival rate of 2 requests/second, the best configuration is with a max service parallelism 4, inter-node parallelism of 2, and intra-node parallelism of 4. The profiler takes the system information (e.g., the number of machines and cores, and workload) as input and conducts lightweight profiling and feeds the profiling results to the prediction model. The prediction model is the key component of the framework. It utilizes the profiling results to predict the latency of all combinations of parallelism under different load levels and populates the configuration reference table. This table only needs to be built once, provided that

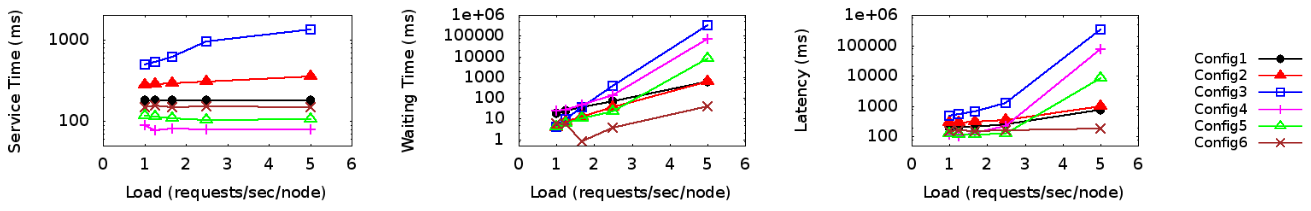
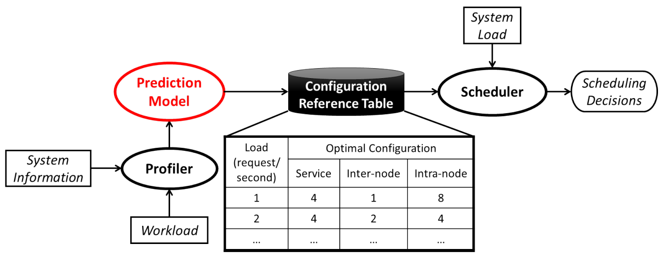


Fig. 7. Service time, waiting time, and latency under different loads using different configurations for ImageNet-22K.

DNN workload characteristics and system hardware remain the same. The scheduler uses the current system load as index to search the configuration reference table, find and adapt to the best parallel configurations.



Load (request/ second)	Optimal Configuration		
	Service	Inter-node	Intra-node
1	4	1	8
2	4	2	4
...	...	...	...

Fig. 8. Overview of SERF.

B. *Profiler*

An easy but inefficient way to achieve the scheduling objective is via exhaustive profiling: execute all possible parallelism configurations for all possible loads and find the best parallel configuration for each load. The shortcoming of such exhaustive profiling is its high cost. Assuming that there are P different configurations and there are L load levels, one needs to conduct $P \times L$ profiling experiments. In addition, measuring average latency requires a relatively long time span (to measure enough samples) to achieve statistical stability due to the stochastic queuing behaviors. Experimenting with lighter load levels requires even longer time for profiling because the large idle intervals between requests increase the duration of the experiment. Let T be the average cost to achieve statistical stability in profiling, which makes the overall cost of exhaustive profiling $P \times L \times T$.

SERF conducts lightweight profiling by measuring the *request service time* for each parallelism degree tuple of (service parallelism degree, inter-node parallelism degree, intra-node parallelism degree). For example, with the tuple (2, 4, 3), we measure the request service time by running two requests concurrently, each request across 4 server nodes and with 3 cores on each server node. Let E denote the cost of profiling request service time for a given parallelism degree tuple, the total profiling cost of SERF is $P \times E$, where P is the total number of parallelism degree combinations. The profiling of SERF has two key differences compared to exhaustive profiling, resulting in significantly lower profiling cost: (1) SERF measures the request *service time* instead of latency, and (2) SERF measures each parallelism degree tuple instead of each parallel configuration. Benefit of these profiling choices is two-fold: (1) the service time under a given parallelism degree tuple is independent of load, saving a multiplicative cost factor along the load dimension L. (2) As requests have deterministic service time under the same parallelism degree tuple and profiling the service time is independent of the queueing delays, a few profiling samples are sufficient, i.e., the value of E is small. In contrast, exhaustive profiling measures latency for each parallelism configuration, which requires running many samples to achieve statistical stability for queuing delays, i.e., T is much more costly than E, by up to 3 orders of magnitude. Therefore, SERF profiling is much more efficient than exhaustive profiling, and $P \times E \ll P \times L \times T$. We feed these profiling results to the prediction model of SERF to estimate the latency under different load levels, which is introduced next.

C. *Queueing-based Prediction Model*

We develop a queueing model that takes profiling results as input and predicts request latency under different load and parallelism configurations. The key challenge and novelty of the model is its interference-awareness, effectively quantifying the latency impact of request interference due to cache and memory contention.

1) Problem Formulation: We define the problem as predicting DNN request latency for any given parallel configuration under any given load. We denote parallelism configuration with *(maximum service parallelism $C_{service}$, inter-node parallelism C_{inter}, and intra-node parallelism C_{intra})*. The inputs of the model are:

- Load in terms of inter-arrival rate: λ, here we assume Poisson arrivals for a *short period*, i.e., exponential inter-arrival times with mean rate λ, which is typical for online services [37], [38]. Such assumption does not contradict the bursty and long-range dependence characteristics in the literature [39]. SERF continuously monitors the incoming workload and periodically updates its observed load (arrival rate).
- Profiling results: μ_i $(i = 1...c)$ represents the average service rate when i requests are running concurrently, i.e., the average service rate of the parallelism degree tuple $(i, C_{inter}, C_{intra})$.

The output of the model is the average latency for the parallelism configuration under any given load.

We model DNN serving as an interference-aware deterministic service process and formulate the problem as a $M/D_{interf}/c$ queue. Here, M represents exponential inter-arrival times. D_{interf} represents two distinctive properties of DNN workload: (1) **D**eterministic service times, modeling homogeneous requests that exhibit little service time variance for any given parallelism degree tuple (as shown in Section III-B). (2) **Interf**erence-awareness, modeling the interference among requests due to cache and memory contention (as shown in Section III-C). c stands for the maximum service parallelism, equal to $C_{service}$.

2) Technical Challenges and Key Ideas: The $M/D_{interf}/c$ queue does not have a closed-form solution. In fact, even for the simpler problems: the interference-oblivious $M/D/c$ queue that assumes deterministic service time without any interference among concurrent running requests, or interference-aware $M/M_{interf}/c$ queue that assume exponential distributed service times with interference among concurrent running requests, there is no closed-form solution. Intuitively, one may want to use $M/M_{interf}/c$ queue, $M/D/c$ queue, or $M/M/c$ queue to approximate the $M/D_{interf}/c$ queue, but such approximation has the potential of achieving bad accuracy. To illustrate why these simpler approaches can not model DNN workload, we implement these approximation methods and conduct experiments using ImageNet-22K. We compare the latency results of best configurations under different loads between testbed measurements and prediction results from $M/M_{interf}/c$ queue, $M/D/c$ queue, and $M/M/c$ queue in Figure 9. The results clearly shows that the prediction from these approaches is poor. This large discrepancy shows the importance of incorporating interference and deterministic service times into SERF prediction model and solution.

Our solution is inspired by Cosmetatos' approximation [40] that estimates $M/D/c$ model using the $M/M/c$ model with adjustment and correction, where $M/M/c$ model is a standard multi-server queue model with Poisson arrival and exponential service time. We extend the approximation approach to the interference-aware case and solve $M/D_{interf}/c$ queue in two steps. (1) Solve $M/M_{interf}/c$ queue that has interference-aware exponential service time. (2) Utilize the approximation method proposed in Cosmetatos' approximation to adjust the results of $M/M_{interf}/c$ queue to approximate the $M/D_{interf}/c$ queue. We estimate the waiting time and service time separately. Latency is estimated as the sum of these two measures.

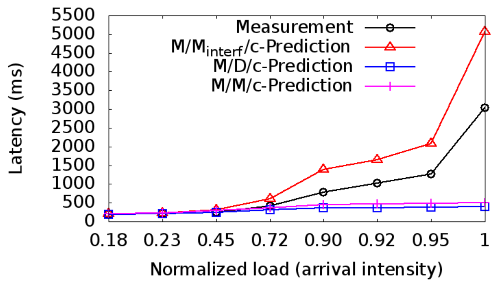


Fig. 9. Latency comparison of best configurations between measurement and standard prediction under different load.

3) Solving $M/D_{interf}/c$ queue: **Waiting time estimation.** We follow the two steps described in Section IV-C2 to solve for the waiting time.

(1) Solving $M/M_{interf}/c$ queue. Recall that μ_i $(i = 1...c)$ is provided by profiling and represents the average service rate when i requests are concurrently running, p_i is the probability of i requests in the system. Let $\rho_i = \frac{\lambda}{\mu_i}$ and $\rho = \frac{\rho_c}{c} = \frac{\lambda}{c \cdot \mu_c}$. Based on the state transition diagram shown in Figure 10 and global balance equations, we obtain:

$$p_n = \begin{cases} \frac{\prod_{i=1}^{n} \rho_i}{n!} \cdot p_0 & (0 \le n \le c-1) \\ \frac{\rho^{n-c} \cdot \prod_{i=1}^{n} \rho_i}{c!} \cdot p_0 & (n \ge c), \end{cases} \tag{2}$$

where p_n is the steady-state probability of state n, which represents n requests in the system (sum of requests in the queue and in service). p_0 represents the probability that the system is idle, i.e., no request is in the system. Since all probabilities sum to 1:

$$\begin{aligned} \sum_{k=0}^{\infty} p_k &= p_0 \cdot (1 + \sum_{k=1}^{c-1} \frac{\prod_{i=1}^{k} \rho_i}{k!} + \frac{\prod_{i=1}^{c} \rho_i}{c!} \cdot \sum_{k=c}^{\infty} \rho^{k-c}) \\ &= p_0 \cdot (1 + \sum_{k=1}^{c-1} \frac{\prod_{i=1}^{k} \rho_i}{k!} + \frac{\prod_{i=1}^{c} \rho_i}{c! \cdot (1-\rho)}) = 1. \end{aligned} \tag{3}$$

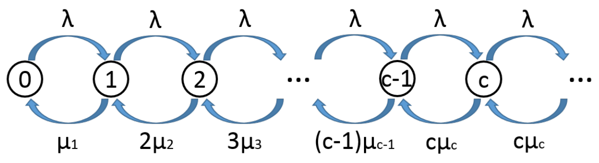


Fig. 10. State transition diagram for the $M/M_{interf}/c$ queue. Each state represents the number of requests in the node.

Let $H = 1 + \sum_{k=1}^{c-1} \frac{\prod_{i=1}^{k} \rho_i}{k!} + \frac{\prod_{i=1}^{c} \rho_i}{c! \cdot (1-\rho)}$, then:

$$p_0 = H^{-1}. \tag{4}$$

Assume that $L_q(\lambda)$ is the average number of requests waiting in the queue, by definition we have:

$$L_q(\lambda) = \sum_{k=c}^{\infty} (k-c) \cdot p_k, \tag{5}$$

together with Eq. 2, we have:

$$\begin{aligned} L_q(\lambda) &= \frac{p_0 \cdot \prod_{i=1}^{c} \rho_i}{c!} \cdot \sum_{k=c}^{\infty} (k-c) \cdot \rho^{k-c} \\ &= \frac{p_0 \cdot \prod_{i=1}^{c} \rho_i}{c!} \cdot \frac{\rho}{(1-\rho)^2}. \end{aligned} \tag{6}$$

Using Little's law [41], the waiting time in the queue can be computed as:

$$W_q^{M/M_{interf}/c}(\lambda) = \frac{L_q(\lambda)}{\lambda} = \frac{p_0 \cdot \prod_{i=1}^{c} \rho_i}{\lambda \cdot c!} \cdot \frac{\rho}{(1-\rho)^2}, \tag{7}$$

(2). Approximating $M/D_{interf}/c$ using $M/M_{interf}/c$. Cosmetatos' approximation proposed in [40] states that the waiting time in the queue can be approximated as:

$$W_q^{M/D/c} \approx \frac{1}{2}(1+f(s)\cdot g(\rho))\cdot W_q^{M/M/c}, \tag{8}$$

where

$$f(s) = \frac{(c-1)\cdot(\sqrt{4+5c}-2)}{16c}, \tag{9}$$

$$g(\rho) = \frac{1-\rho}{\rho}, \tag{10}$$

$\rho = \frac{\lambda}{c\cdot\mu}$, λ is the average arrival rate, and μ is the average service rate. This approximation can be adjusted for the interference-aware case: we use the $M/M_{interf}/c$ queue with the same correction terms as $f(s)$ and $g(\rho)$ as above to approximate the $M/D_{interf}/c$ queue as follows (using Eq. 7 and Eq. 8):

$$W_q^{M/D_{interf}/c}(\lambda) \approx \frac{1}{2}(1+f(s)\cdot g(\rho))\cdot \frac{p_0\cdot\prod_{i=1}^{c}\rho_i}{\lambda\cdot c!}\cdot\frac{\rho}{(1-\rho)^2}. \tag{11}$$

Service time estimation. Although service time under the same parallelism degree tuple is deterministic and can be profiled, the service time under a given parallel configuration could change with load and needs to be predicted. This is because of the random requests arrival process and interference, e.g., at different moments, the system may have different number of concurrent running requests (ranging from 0 to the defined maximum service parallelism), which results in different interference and therefore different service times. We use the PASTA (Poisson Arrivals See Time Averages) property [42] to compute the average service time $S^{M/D_{interf}/c}(\lambda)$ under arrival rate λ as follows:

$$\begin{aligned} S^{M/D_{interf}/c}(\lambda) &= \frac{1}{\mu_1}\cdot p_0 + \frac{1}{\mu_2}\cdot p_1 + \frac{1}{\mu_3}\cdot p_2 + ... \\ &\quad + \frac{1}{\mu_c}\cdot p_{c-1} + \frac{1}{\mu_c}\cdot\sum_{i=c}^{\infty} p_i \\ &= \sum_{i=1}^{c}\frac{p_{i-1}}{\mu_i} + \frac{\prod_{i=1}^{c}\rho_i}{\mu_c\cdot c!\cdot(1-\rho)}. \end{aligned} \tag{12}$$

Latency estimation. The average latency $W^{M/D_{interf}/c}$ equals to the average time spent in waiting in queue $W_q^{M/D_{interf}/c}$ plus the average time spent in execution $S^{M/D_{interf}/c}$:

$$\begin{aligned} W^{M/D_{interf}/c}(\lambda) &\approx \sum_{i=1}^{c}\frac{p_{i-1}}{\mu_i} + \frac{\prod_{i=1}^{c}\rho_i}{\mu_c\cdot c!\cdot(1-\rho)} \\ &\quad + \frac{1}{2}(1+f(s)\cdot g(\rho))\cdot\frac{p_0\cdot\prod_{i=1}^{c}\rho_i}{\lambda\cdot c!}\cdot\frac{\rho}{(1-\rho)^2}. \end{aligned} \tag{13}$$

In the above formula, recall that μ_i is an input from profiling, λ is affected by inter-node parallelism C_{inter} as it defines how many machines to serve each request, c equals to the maximum allowed service parallelism $C_{service}$, and the intra-node parallelism is restricted by F/c, where F is the number of cores in a node. Therefore, for a given system and workload, latency can be computed under different combinations of service parallelism, inter-node parallelism, and intra-node parallelism. Eq. 13 is used to populate the configuration reference table that is the core of SERF.

The above solution is derived for a single serving unit (with C_{inter} number of machines). For a cluster, the cluster can be divided into serving units based on the inter-node parallelism C_{inter}, e.g., for a cluster with N machines, there are N/C_{inter} units and each unit has an arrival rate of $\lambda = \frac{\lambda_{all}}{N/C_{inter}}$, where λ_{all} is the request arrival rate at the cluster.

D. Scheduler

The scheduler takes the current system load as input, searches the configuration reference table, finds and adapts to the best parallelism configuration. To enable quick configuration switching, the entire DNN model is pre-installed on each server, and each input is sent to the server with a mapping of servers to input partitions. This informs the server of which partition of the DNN model to use to process the input, and which servers to communicate with for cross-machine neural connections.

To sum up, we explore two distinctive properties of DNN workload — homogeneous requests with interference — to develop SERF. SERF combines lightweight profiling with an interference-aware queueing model to predict DNN serving latency. It finds the best parallel configuration for any given load and then deploys a dynamic scheduler to adapt to varying loads online nearly instantly.

V. Experimental Evaluation

We present experimental results demonstrating how SERF improves DNN serving performance with respect to minimizing response latency. Specifically, we evaluate the following properties of SERF: (i) accuracy of the latency prediction model, (ii) correct identification of best parallel configuration under different loads, (iii) adaptability to load dynamism compared to a static configuration, and (iv) efficient best configuration search compared to exhaustive profiling.

A. Experimental Setup

System Overview: We prototyped SERF based on the Adam distributed DNN system [3], which supports service parallelism through admission control, intra-node parallelism using OpenMP [29], and inter-node parallelism by partitioning the model across different machines. In order to quickly switch the configurations, the entire model parameter is pre-installed on each server, and each input is augmented to the server with a mapping of servers to input partitions. As most distributed DNN serving platforms support part or all of these parallelisms, SERF can be used in other systems as well.

Workload: We evaluate SERF using 3 popular image recognition tasks of varying complexity with Poisson request arrivals:

- CIFAR-10 [2]: classifies 32x32 color images into 10 categories. The DNN is moderately-sized, containing about

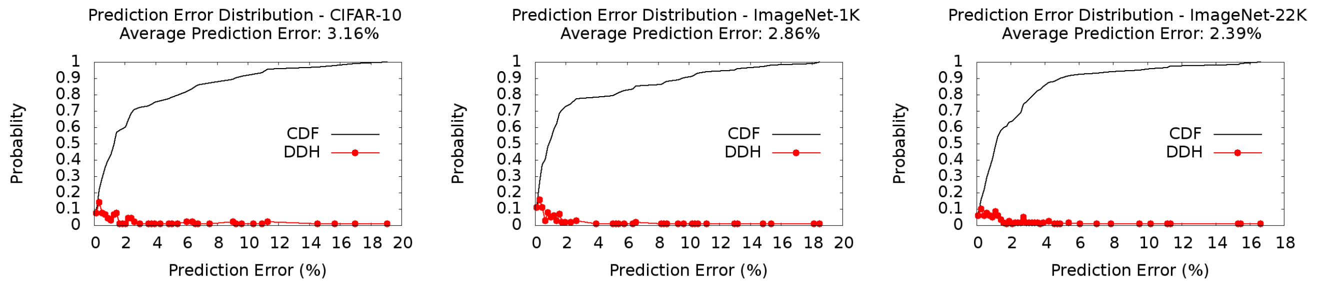


Fig. 11. CDF (Cumulative Distribution Function) and DDH (Data Density Histogram) of prediction errors for different workloads.

28.5 million connections in 5 layers: 2 convolutional layers with pooling, 2 fully connected layers, and a 10-way output layer.

- ImageNet-1K [23]: classifies 256x256 color images into 1,000 categories. The DNN is moderately large, containing about 60 million connections in 8 layers: 5 convolutional layers with pooling, 3 fully connected layers, and a 1,000-way output layer [2].
- ImageNet-22K [23]: the largest ImageNet task, which is to classify 256x256 color images into 22,000 categories. This DNN is extremely large, containing over 2 billion connections in 8 layers: 5 convolutional layers with pooling, 3 fully connected layers, and a 22,000-way output layer [3].

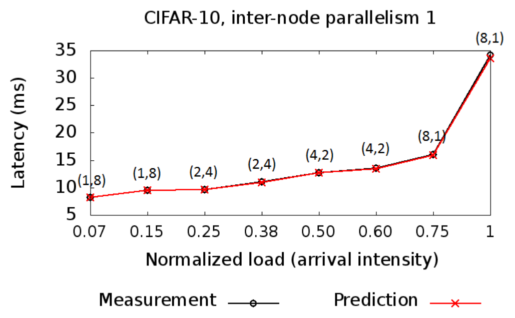
Hardware Environment: Experiments are run on a computing cluster of 20 identically configured commodity machines, communicating over Ethernet through a single 10Gbps (bidirectional) NIC. Each machine is dual-socket, with an Intel Xeon E5-2450 processor of 8 cores running at 2.1GHz on each socket. Each machine has 64 GB of memory and a 268.8 GFLOP/s SIMD FPU.

B. Accuracy of Latency Prediction Model

This section evaluates the accuracy of the latency prediction model, based on Eq. 13, by comparing predicted values to measured values. Figure 11 shows for each workload the average and distribution of prediction errors for all relevant prediction cases. A relevant prediction case is a combination of a parallel configuration that has performance impact for a workload and a load level. For example, CIFAR-10 has 20 parallel configurations because inter-node parallelism degrees > 1 do not make sense for its small size. The larger ImageNet-1K and ImageNet-22K have 40 and 80 parallel configurations because inter-node parallelism degrees of up to 2 and 4 are relevant, respectively. For each benchmark we consider 10 load levels evenly spread across low load to high load, so that there are 200, 400, and 800 relevant prediction cases for CIFAR-10, ImageNet-1K, and ImageNet-22K, respectively. The results show that the prediction is accurate and the errors are insignificant: the average error is 2-4%, the 90th percentile is $< 10\%$, and the 95th percentile is $< 12\%$.

C. Identifying Best Configurations

We use our prediction model to identify the best configuration under different load levels and then compare with the



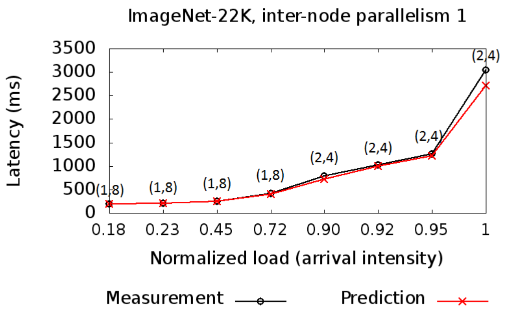


Fig. 12. Best configurations and according latency under different loads.

testbed measurement ground truth. The experimental results show that SERF *always* correctly identifies the best configuration. The top plot in Figure 12 depicts the best configuration and according latency of serving CIFAR-10 workload under different load levels. It is clear when the load increases, the latency of best configuration also increases due to queuing effects. Moreover, we observe the following:

- When load is low, intra-node parallelism is useful since the service time is the dominant factor in latency. Intra-node parallelism helps reduce service times and therefore achieves better overall latency.
- When load is high, since the interference caused by service parallelism is low, service parallelism is required because the waiting time becomes the dominant factor and it reduces waiting time more efficiently by allowing more requests to run in parallel.

For large DNNs like ImageNet-22K, the observation is interesting and counter-intuitive, see the bottom plot in Figure 12. Different from CIFAR-10, even under high loads, the best parallel configuration is still with service parallelism of only 2. Intuitively, when the load is high, admitting more requests into the system could yield better performance and the maxi-

Benchmark	Method	# of configs	# of load levels	# of profile exp to run	Each profile time (min)	Total time (min)
ImageNet-22K	Exhaustive	80	10	800	34.50	27600
	SERF	80	0	80	0.07	5.52
ImageNet-1K	Exhaustive	40	10	400	10.83	4332
	SERF	40	0	40	0.02	0.87
CIFAR-10	Exhaustive	20	10	200	0.36	72
	SERF	20	0	20	7.19×10^{-4}	1.44×10^{-2}

TABLE II
COST COMPARISON BETWEEN EXHAUSTIVE PROFILING AND SERF FOR DIFFERENT BENCHMARKS.

mum service parallelism should be best. This counter-intuitive result is a consequence of the high interference. When the interference among requests is high, service parallelism may result in significantly increased service time, which exceeds the waiting time benefit brought by allowing more requests running in parallel, causing higher latency.

D. Benefits over Exhaustive Profiling

We evaluate SERF here against exhaustive profiling for identifying the best parallel configurations under different load levels. The experimental results verified both SERF and exhaustive profiling *always* correctly identifies the best configuration. However, the cost of SERF is significantly lower than exhaustive profiling. Assume that the system has 80 different parallel configurations and the performance reference table has 10 entries (e.g., 10 different load levels). SERF requires only 80 quick profiling experiments while exhaustive profiling requires 800 expensive profiling experiments to build the performance reference table. The time for each profiling experiment and the total time to build the performance reference table is shown in Table II. Note SERF requires much less time for each profiling experiment because it only samples the service time and the service time is deterministic without load impact (i.e., sample the service time of only 10 requests) while each exhaustive profiling experiment needs to measure the average latency, which needs many samples to achieve statical stability (e.g., when measuring latency less than 5000 sample requests, the results become very unstable). The results suggest that the time cost of SERF is more than 3 orders of magnitudes lower than exhaustive profiling, and the time savings grows with the size of the DNN workload and the number of performance reference table entries. Even if compared to lightweight profiling, e.g., only do profiling under high load, the cost of SERF is still more than 2 orders of magnitudes lower.

E. Benefits over Static Configuration

Request arrival rate and system load changes dynamically for online services [38]. In this section, we demonstrate how SERF outperforms fixed configurations by adapting to load changes. We use three baseline cases for comparison. Fixed-low is a best configuration in low load and Fixed-mod is a best configuration in moderate load. Fixed-other is another configuration that performs better than Fixed-low in moderate loads and better than Fix-mod in low loads. We compare the performance of SERF and these baseline cases in a dynamic user environment with load changes from moderate to low and then back to moderate, see Figure 13. The y-axis is the latency measured in ms, the x-axis represents the experiment's elapsed time. Fixed-low and Fixed-mod perform well under the loads that they are optimized for, but perform poorly when load changes. Fixed-other achieves more stable performance, but not best under any load levels. SERF outperforms all these baseline scheduling methods and consistently adapts to the load change to achieve lowest latency. This experiment validates the need for adaptivity of SERF in a dynamic workload environment, where, for example, a best configuration for high loads could be sub-optimal for low loads. In addition, the profiling cost of these fixed configurations is more than 2 orders of magnitude higher than SERF, e.g., for ImageNet-22K, it takes nearly 2 days to identify Fixed-low or Fixed-mod configuration by profiling, and it takes even longer for Fix-other as profiling needs to be done for multiple loads. In comparison, SERF only takes a few minutes for identifying the best parallel configurations under various loads.

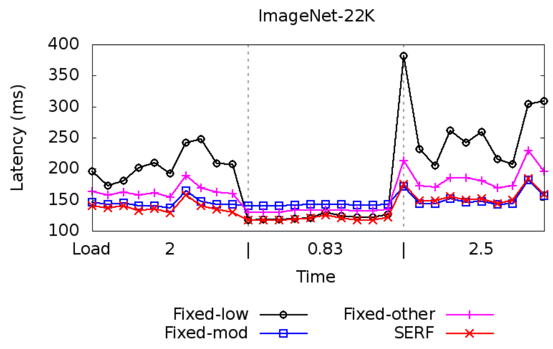


Fig. 13. Latency comparison under dynamic load environment for different scheduling approaches.

F. Discussion

Scalability. When SERF works in large systems, the number of profiling experiments scales linearly with the total number of parallelism combinations. Because each profiling takes less than a few seconds, even for large systems running large and scalable applications with thousands of parallelism combinations, the profiling takes no more than a few hours. This profiling time can be further reduced to a few minutes if profiling experiments are conducted in parallel or in coarser granularity. In addition, the computation of the queueing model is efficient, i.e., constant with respect to the cluster size. Therefore, SERF is scalable to schedule large systems.

Generalization to other workloads. SERF also has a potential to be used in other applications because SERF is developed based on two abstracted properties: homogeneous

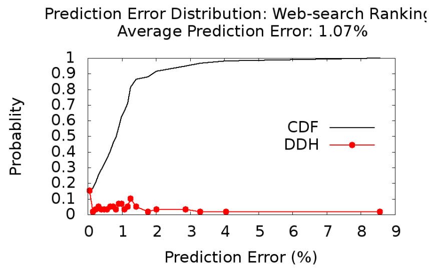


Fig. 14. CDF and DDH of prediction errors.

request service demand and interference among requests running concurrently. Applications with similar properties can also benefit from our approach. We use web-search ranking [43] as an example for evaluation as it represents a typical supervised machine learning problem. We instrument the implementation in [43] to make it a parallel version to simulate a serving system. We run extensive experiments with various load levels using different paralellism configurations. We show the average and distribution of the prediction errors in Figure 14. The results show SERF is quite accurate as the average error is only 1.07%, the 90th percentile is $< 3\%$, and the 95th percentile is $< 5\%$. The experimental results also show that SERF always identifies the best configuration correctly. In the interest of space, we omit detailed discussion here.

VI. Related Work

DNN Serving. The state-of-the-art accuracy of DNNs on important artificial intelligence tasks, such as image recognition [1], [2], [3], speech recognition [9], [10], and text processing [11] has made the end-to-end latency of large-scale DNN serving systems an important research topic. Parallelism has being shown to be critical for good DNN performance at scale. Prior work [1], [3] has shown that parallel training on a cluster of commodity CPU machines achieves high throughput thus can train big DNN models (billions of connections) in a reasonable amount of time (days instead of months). Although these training platforms focus on improving system throughput instead of request latency, the parallelism mechanisms proposed there are directly translated to serving platforms as inter-node, intra-node and service parallelisms. Several recent work on DNN serving investigate hardware acceleration using GPUs [44], FPGAs [45], and ASICs [28]. They focus on mapping DNN computation to customized hardware, but parallelism has also been shown critical to offer low latency. All these prior studies develop DNN serving platforms that support all or a subset of the parallelism mechanisms exploited in our paper. However, none of them investigates scheduling frameworks that make parallelism configuration choices based on DNN characteristics, hardware characteristics, and system load, which is the focus of SERF. SERF is complementary to the above work and can be used as a scheduling framework for these serving platforms to identify best parallelism configurations and maximize their parallelism benefits.

Interactive Serving. There is a host of research in parallelizing request processing to reduce response latency, and request scheduling in a multiprocessor environment to reduce average latency. There has been a lot of work on measuring and mitigating interference among co-located workloads [46], [47]. The main theme is to predict performance interference among workloads and discover optimal workload co-locations to improve system utilization while meeting user performance goals. These studies treat each workload as a blackbox, and they do not consider solutions that involve modifying the workload (e.g., changing parallelism degree). Adaptive parallelism for interactive server systems uses intra-node and service parallelism to reduce request latency. Raman *et al.* propose an API and runtime system for dynamic parallelism [33], where developers express parallelism options and goals, such as minimizing mean response time. Jeon *et al.* [32] propose a dynamic parallelization algorithm to decide the degree of request parallelism in order to reduce the average response time of Web search queries. Both approaches assume independent service time among requests, thus they do not consider interference among concurrent running requests, which is a key property of DNN workload supported by SERF. Another line of work [48] proposes to use parallelism to reduce tail latency. DNN requests, however, are homogeneous with similar service time, making these techniques ineffective. Finding best parallel configurations has also been studied on other applications and systems, such as database, data analytics, MapReduce [49], [50], [51]. However, none of these prior work leverages the distinctive properties of DNN workloads to exploit request homogeneity and interference awareness as SERF does.

Queueing Models. Here we outline some results that are related to the M/D/c queue abstraction used in our work. While the solution of the M/M/c system is exact [52], there are no exact solutions for M/D/c systems. We note the existence of the Allen-Cunnen approximation formula for GI/G/c [41] and Kimura's approximation [53], both of which can also apply to M/D/c since M is a special case of GI. Alternatively, an $M/D/c$ system can be approximated using an n-stage Erlang for the service process, essentially by approximating the system using a $M/Ph/1$ queue. While the $M/Ph/1$ queue can be solved using the matrix-geometric method [54], the $M/Ph/c$ suffers from the well known problem of state space explosion. We direct the interested reader to [34] for an overview of various results on the M/D/c queue that have been developed since the early 1930s. However, none of the above approximation methods for $M/D/c$ systems can be easily adapted to estimate latency of $M/D_{interf}/c$ systems. Here we extend the approximation by Cosmetatos to achieve this goal.

VII. Conclusions

We presented SERF, a scheduling framework for DNN serving systems, which combines lightweight profiling with an interference-aware queueing-based prediction model. SERF efficiently identifies best parallel configurations to minimize average request latency and it dynamically adapts to varying loads almost instantly.

VIII. Acknowledgments

This work is supported by NSF grant CCF-1218758.

References

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks." in *NIPS*, 2012.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[3] T. Chilimbi, J. Apacible, K. Kalyanaraman, and Y. Suzue, "Project adam: Building an efficient and scalable deep learning training system," in *OSDI*, 2014.

[4] J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille, "Explain images with multimodal recurrent neural networks," *CoRR*, 2014.

[5] H. Fang, S. Gupta, F. N. Iandola, R. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, C. L. Zitnick, and G. Zweig, "From captions to visual concepts and back," in *CVPR*, 2015.

[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *CVPR*, 2014.

[7] J. Y. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond Short Snippets: Deep Networks for Video Classification," in *CVPR*, 2015.

[8] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. J. Mooney, and K. Saenko, "Translating videos to natural language using deep recurrent neural networks," in *NAACL HLT*, 2015.

[9] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *TASLP*, 2012.

[10] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep-Speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[11] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," *CoRR*, 2010.

[12] D. Talbot, "How Microsoft Cortana improves upon Siri and Google Now," http://www.tomshardware.com/news/microsoft-cortana-unique-features,26506.html, accessed: 2015-11-20.

[13] R. Mcmillan, "How Skype used AI to build its amazing new language translator," http://www.wired.com/2014/12/skype-used-ai-build-amazing-new-language-translator/, accessed: 2015-11-20.

[14] C. Rosenberg, "Improving photo search: A step across the semantic gap," http://googleresearch.blogspot.com/2013/06/improving-photo-search-step-across.html, accessed: 2015-11-20.

[15] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "A picture is worth a thousand (coherent) words: building a natural description of images," http://googleresearch.blogspot.com/2014/11/a-picture-is-worth-thousand-coherent.html, accessed: 2015-11-20.

[16] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande, "Massively Multitask Networks for Drug Discovery," *arXiv preprint arXiv:1502.02072*, 2015.

[17] F. Nelson, "Nvidia demos a car computer trained with deep learning"," http://www.technologyreview.com/news/533936/, accessed: 2015-11-20.

[18] Microsoft, "Microsoft azure machine learning," http://azure.microsoft.com/en-us/services/machine-learning/, accessed: 2015-11-20.

[19] Amazon, "Amazon machine learning," https://aws.amazon.com/machine-learning/, accessed: 2015-11-20.

[20] T. Hoff, "Latency is everywhere and it costs you sales–how to crush it," *HS*, 2009.

[21] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating user-perceived quality into web server design," *CN*, 2000.

[22] Microsoft, "Publish an azure machine learning web service," https://azure.microsoft.com/en-us/documentation/articles/machine-learning-publish-a-machine-learning-web-service/, accessed: 2015-11-20.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR*, 2009.

[24] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[25] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," DLUFL, 2012.

[26] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn*, 2011.

[27] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *COMPSTAT*, 2010.

[28] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014.

[29] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *CSE*, 1998.

[30] C. Pheatt, "Intel® threading building blocks," *CCSC*, 2008.

[31] C. Demichelis and P. Chimento, "IP packet delay variation metric for IP performance metrics (IPPM)," 2002.

[32] M. Jeon, Y. He, S. Elnikety, A. L. Cox, and S. Rixner, "Adaptive parallelism for web search," in *EuroSys*, 2013.

[33] A. Raman, H. Kim, T. Oh, J. W. Lee, and D. I. August, "Parallelism orchestration using DoPE: the degree of parallelism executive," in *PLDI*, 2011.

[34] H. Tijms, "New and old results for the M/D/c queue," *JEC*, 2006.

[35] C. Mega, T. Waizenegger, D. Lebutsch, S. Schleipen, and J. Barney, "Dynamic cloud service topology adaption for minimizing resources while meeting performance goals," *IBM Journal of R & D*, 2014.

[36] Z. Zhang, L. Cherkasova, and B. T. Loo, "Optimizing cost and performance trade-offs for mapreduce job processing in the cloud," in *NOMS*, 2014.

[37] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "On the nonstationarity of internet traffic," in *SIGMETRICS*, 2001.

[38] M. F. Arlitt and C. L. Williamson, "Internet web servers: Workload characterization and performance implications," *ToN*, 1997.

[39] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A nonstationary poisson view of internet traffic," in *INFOCOM*, 2004.

[40] G. P. Cosmetatos, "Approximate explicit formulae for the average queueing time in the processes (M/D/r) and (D/M/r)," *Infor*, 1975.

[41] L. A. Baxter, "Probability, statistics, and queueing theory with computer sciences applications," *Technometrics*, 1992.

[42] R. W. Wolff, "Poisson arrivals see time averages," *Operations Research*, 1982.

[43] A. Mohan, Z. Chen, and K. Q. Weinberger, "Web-search ranking with initialized gradient boosted regression trees," *JMLR*, 2011.

[44] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers," in *ISCA*, 2015.

[45] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA*, 2015.

[46] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Souffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *ISCA*, 2011.

[47] R. Kettimuthu, G. Vardoyan, G. Agrawal, P. Sadayappan, and I. T. Foster, "An elegant sufficiency: load-aware differentiated scheduling of data transfers," in *SC*, 2015.

[48] M. E. Haque, Y. H. Eom, Y. He, S. Elnikety, R. Bianchini, and K. S. McKinley, "Few-to-many: Incremental parallelism for reducing tail latency in interactive services," in *ASPLOS*, 2015.

[49] X. Liu and B. Wu, "Scaanalyzer: a tool to identify memory scalability bottlenecks in parallel programs," in *SC*, 2015.

[50] R. Susukita, H. Ando, M. Aoyagi, H. Honda, Y. Inadomi, K. Inoue, S. Ishizuki, Y. Kimura, H. Komatsu, M. Kurokawa, K. Murakami, H. Shibamura, S. Yamamura, and Y. Yu, "Performance prediction of large-scale parallell system and application using macro-level simulation," in *SC*, 2008.

[51] M. Sarwat, S. Elnikety, Y. He, and M. F. Mokbel, "Horton+: a distributed system for processing declarative reachability queries over partitioned graphs," *VLDB*, 2013.

[52] L. M. Leemis and S. K. Park, *Discrete-event simulation: A first course*, 2006.

[53] T. Kimura, "Approximating the mean waiting time in the GI/G/s queue," *JORS*, 1991.

[54] M. F. Neuts, *Matrix-geometric solutions in stochastic models: an algorithmic approach.* Courier Corporation, 1981.