Bridging the Semantic Gaps of GPU Acceleration for Scale-out CNN-based Big Data Processing: Think Big, See Small

*Mingcong Song[1], *Yang Hu[1], Yunlong Xu[2], Chao Li[3], Huixiang Chen[1], Jingling Yuan[4], Tao Li[1]
[1]Department of Electrical and Computer Engineering, University of Florida, Gainesville, USA
[2]School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China
[3]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[4]Wuhan University of Technology, Wuhan, China
{songmingcong, huyang.ece, stanley.chen}@ufl.edu, xjtu.ylxu@stu.xjtu.edu.cn,
lichao@cs.sjtu.edu.cn, yjl@whut.edu.cn, taoli@ece.ufl.edu

ABSTRACT
Convolutional Neural Networks (CNNs) have substantially advanced the state-of-the-art accuracies of object recognition, which is the core function of a myriad of modern multimedia processing techniques such as image/video processing, speech recognition, and natural language processing. GPU-based accelerators gained increasing attention because a large amount of highly parallel neurons in CNN naturally matches the GPU computation pattern. In this work, we perform comprehensive experiments to investigate the performance bottlenecks and overheads of current GPU acceleration platform for scale-out CNN-based big data processing.

In our characterization, we observe two significant semantic gaps: framework gap that lies between CNN-based data processing workflow and data processing manner in distributed framework; and the standalone gap that lies between the uneven computation loads at different CNN layers and fixed computing capacity provisioning of current GPU acceleration library. To bridge these gaps, we propose *D*^{3}*NN*, a Distributed, Decoupled, and Dynamically tuned GPU acceleration framework for modern CNN architectures. In particular, *D*^{3}*NN* features a novel analytical model that enables accurate time estimation of GPU accelerated CNN processing with only 5-10% error. Our evaluation results show the throughput of standalone processing node using *D*^{3}*NN* gains up to 3.7X performance improvement over current standalone GPU acceleration platform. Our CNN-oriented GPU acceleration library with built-in dynamic batching scheme achieves up to 1.5X performance improvement over the non-batching scheme and outperforms the state-of-the-art deep learning library by up to 28% (performance mode) ~ 67% (memory-efficient mode).

Keywords
GPU; Deep Learning; Distributed System; Big Data.

1. INTRODUCTION
Recent years have seen massive research efforts on developing deep neural networks (DNNs). Among these DNNs, the Convolutional Neural Networks (CNNs) [24, 36, 38] are the most popular subsets. Typically, the CNN is comprised of one or more fully connected layers and multiple convolutional layers. The most appealing feature of CNN is that it has far fewer parameters than fully connected DNNs, which makes CNNs easier to train and more practical [12]. Today, CNNs have been extensively adopted [1, 14, 22, 23, 34, 41, 46] and have substantially advanced the state-of-the-art accuracies of object recognition, which is the core function for image/ speech/ language processing applications.

Due to the inefficiency of general-purpose processors when processing CNN workloads, researchers perceive the opportunities to tap into CNN accelerators. Various accelerators based on FPGA [44], GPU [15] and ASIC [3] have been proposed recently to improve the performance of CNN workloads. Among these approaches, GPU-based accelerators gained increasing attention since a large amount of highly parallel neurons in CNN naturally matches the GPU computation pattern. Furthermore, the proliferation of commodity GPU deployment (bare metal and virtualized) in cloud data centers [2, 37] provides a mature and ready-to-use platform for cloud-based CNN acceleration. DjiNN [15] makes the first attempt to explore commodity GPU-based CNN accelerator server platform and provides beneficial implication to future warehouse-scale computer design. However, the enormous amount of data that generated in current IT big-names' warehouse-scale computers present significant challenges for scale-out CNN-based big data processing [18, 26, 47]. For example, more than 350 million photos are being posted to Facebook per day [13, 25] and 100 hours of video are being uploaded to YouTube per minute [42], such daunting amount of data arrival remarkably embarrasses the throughput of traditional standalone CNN accelerators.

Rather than pushing the limit of standalone CNN accelerators, we instead explore a complementary opportunity to benefit scale out CNN-based big data processing that leverages state-of-the-art heterogeneous CNN acceleration techniques such as commodity GPGPU and widely used distributed computing frameworks such as Hadoop. In this paper, we perform comprehensive experiments to investigate the performance bottlenecks and overheads of current GPU acceleration platform for scale-out CNN-based big data processing. We study three most representative networks: AlexNet [24], GoogLeNet [38] and VGGNet [36]. As many fine-tuned convolutional neural networks are designed based on these representative networks and share the same network architecture, we expect the observations hold valid to their derivatives.

We find significant challenges are associated with GPU acceleration support for scale-out CNN-based big data processing. First, we observe a framework semantic gap that lies between CNN-based data processing workflow and data processing manner in current distributed framework. Specifically, the lack of

PACT '16, September 11-15, 2016, Haifa, Israel
© 2016 ACM. ISBN 978-1-4503-4121-9/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2967938.2967944
*Authors with equal contribution

Table 1. CNNs

CNN Architecture	Fine-tuned Models
AlexNet	FCN-AlexNet, SOS-AlexNet, Places205-AlexNet, Hybrid-CNN, R-CNN, CaffeNet
GoogLeNet	Places205-GoogLeNet, GoogLeNet_cars
VGGNet	FCN-Xs, SOS-VGG16, ParseNet

accurate processing time estimation approach for heterogeneous GPU acceleration nodes presents as an obstacle to provide agile proactive load balance for a distributed CNN acceleration framework, which could result in significant waste of valuable computing resource that provided by the state-of-the-art heterogeneous CNN acceleration hardware.

To bridge this semantic gap, we explore the opportunity to benefit from leveraging GPU acceleration library by looking into more details of standalone CNN acceleration platform implementation. We surprisingly observe another standalone semantic gap that lies between the uneven computing loads at different CNN layers and fixed computing capacity provisioning of current GPU acceleration libraries. Specifically, the uneven GPU resource utilization pattern caused by the lack of awareness of per CNN layer computing load poses significant challenges for achieving optimal GPU utilization. This unpredictable GPU utilization also impacts the performance and restricts the possibility to accurately estimate the CNN acceleration processing time.

Motivated by our characterization observations, we propose *D³NN*, a Distributed, Decoupled, and Dynamically tuned GPU acceleration framework. *D³NN* features a distributed data processing framework, a CNN-oriented GPU acceleration library, and a novel analytical model that bridges the semantic gaps in modern scale-out CNN-based big data processing platform.

D³NN incorporates a CNN-oriented GPU acceleration library that dynamically tunes the batch size of input data at each CNN layer to bridge the standalone semantic gap by waxing mismatch of uneven computing loads at different CNN layers and fixed computing capacity provisioning of current GPU acceleration library. Based on our GPU acceleration library, we also propose an analytical model that bridges the semantic gap to provide accurate processing time estimation approach for heterogeneous GPU acceleration nodes and accurate multi-process number determination method. The distributed data processing framework employs a Producer-Consumer scheme to decouple the CNN data preparation process and CNN data processing process at each slave node, and a semaphore-based data synchronization scheme to bridge the framework semantic gap by matching the distributed data processing manner to CNN-based data processing workflow.

Note that due to the similarity of convolutional neural network architectures, *D³NN* is applicable to any newly developed CNN. Our evaluation results show the throughput of standalone processing node using *D³NN* gains up to 3.7X performance improvement over current standalone GPU acceleration platform. Our CNN-oriented GPU acceleration library with built-in dynamic batching scheme achieves up to 1.5X performance improvement over the non-batching scheme and outperforms the state-of-the-art deep learning library (cuDNN) [5] by 28% (performance mode) ~ 67% (memory-efficient mode). More importantly, as a hierarchical CNN acceleration solution, the decoupled architecture we proposed could also apply to other state-of-the-art works, such as multi-GPU servers [15] and ASIC-based CNN accelerators [3, 4].

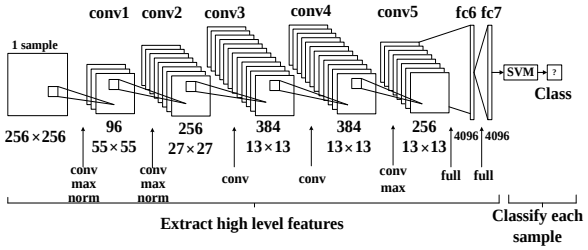


Figure 1. A representative CNN architecture - AlexNet

2. BACKGROUND

CNN: CNN can tackle various multimedia processing applications such as car classification [41], pedestrian classification [34], scene recognition [46], salient object subtilizing [45], object detection [14], video analysis [22] and image captioning [21]. In this study, our characterizations and optimizations mainly focus on three representative CNNs: AlexNet, GoogLeNet, and VGGNet. AlexNet, the ILSVRC [33] 2012 winner model, is the first study that popularized CNN in computer vision. It could be fine-tuned on other databases to implement richer functions. In recent years, deeper and more complicated convolutional networks have been developed to achieve better accuracy. To reflect the latest research on CNNs in our design, we also characterize two latest ILSVRC winners, GoogLeNet and VGGNet. Most fine-tuned CNN models are designed based on them, as shown in Table 1. Although parameters in these fine-tuned CNN models have to be re-trained using new training dataset, their architectures remain the same and the CNN performance will not change drastically.

We introduce essential preliminaries of CNN using AlexNet as an example, shown in Figure 1. It consists of five convolutional layers, three max-pooling layers, and two classifier layers. The convolutional layers perform dot products between the filters and local regions of the input image [6]. These operations dominate the execution time of CNN computation. The convolutional operations in a convolutional layer benefit from the optimized matrix multiplication libraries, such as cuBLAS [7].

We demonstrate a typical matrix multiplication based convolutional operation in Figure 2. In step ①, an operation called im2col [6] stretches out the local regions in the input image (D) into column-major matrix (D_m). Similarly, in step ② the weights of the CONV layer (F) are stretched out into filter matrix (F_m). Then the original convolutional operation could be lowered into a matrix multiplication ($F_m \times D_m$) in step ③. In Figure 2, the filter matrix F_m has dimensions $N_f \times S_f^2 N_c$, while the data matrix D_m has dimensions $S_f^2 N_c \times W_o H_o$. The output matrix O_m has dimensions $N_f \times W_o H_o$. Therefore, we can calculate the number of float point operations in a convolutional layer through the number of multiply-accumulate operations of $F_m \times D_m$:

$$\text{Conv}_{flops} = 2N_f \times S_f^2 N_c \times W_o H_o \qquad \text{(Eq1)},$$

where a single multiply-accumulate operation counts as 2 flops. The Convflops is usually used to measure the computational intensity in a convolutional layer.

GPU: A GPU has multiple streaming multiprocessors (SMs). The SM is the main SIMD processing engine and has several functional blocks, such as integer/floating point ALUs, load/store units, special functional blocks. Execution of general-purpose programs on heterogeneous GPU/CPU architectures is realized by various application programming interfaces (API) such as CUDA

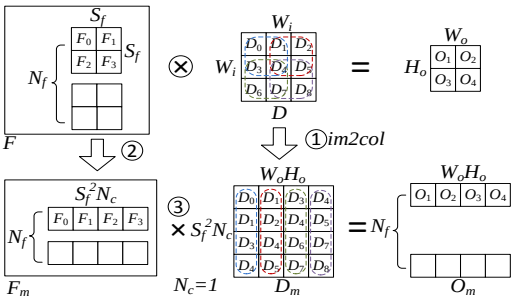


Figure 2. Convolutional operation in a CNN layer

[30], and OpenCL [32]. Using these APIs a programmer can launch thousands of threads onto GPU device from the host CPU. Multiple threads (called a warp) execute simultaneously on an SM following the same instruction multiple data (SIMD) paradigm. Several warps form a thread block (TB) that are executed on the same SM; TBs group together to form a grid that executes a GPU kernel.

During the execution of a GPU kernel, the work distributor checks whether the required resources of a TB can be satisfied by one of the SMs. These resources include the shared memory, the number of registers, and the maximum number of warps concurrently active on an SM. If all these resources requirements are satisfied by an SM, one TB will be dispatched onto it. GPUs hide memory latency through fast context switch among active warps; thus sufficient numbers of active warps should be maintained on an SM. Occupancy [8], which is the ratio of the number of active warps per SM to the maximum number of active warps, is an important metric in determining how effectively the hardware is used. Theoretical occupancy is the upper limit for occupancy imposed by the kernel launch configuration and the capabilities of the GPU, while achieved occupancy is the actual measured occupancy of the running kernel. The achieved occupancy is usually less than the theoretical occupancy mainly due to (1) limited problem size, and (2) unbalanced workload within and/or across TBs.

Hadoop: Hadoop [39] is a most widely used framework for distributed processing of large data sets across a cluster of computers. Hadoop was inspired by Google's MapReduce [10], a software framework in which an application is broken down into numerous small map/reduce tasks. Any of these tasks can be run on any node in the cluster. Hadoop includes a Distributed File System (HDFS), which is usually mounted on each slave node and provide data storage service for MapReduce.

3. CHARACTERIZATION AND IMPLICATION

To understand how to design optimized GPU acceleration system for CNN applications, we perform an in-depth, hierarchical characterization of the performance of existing computing framework and GPU acceleration library for CNN to identify inefficiencies and bottlenecks. Compared to previous work that largely focuses on performance characterization of GPU computation [15], our comprehensive characterization shed light on some ignored realities. We first describe our experimental setup in Section 3.1. In Section 3.2, we explore the bottlenecks that constitute framework semantic gap. In Section 3.3, we delve into standalone CNN processing platform to explore the limitations of current GPU-accelerated implementation for CNN, which result in the standalone semantic gap. We summarize the root causes of these inefficiencies in Section 3.4.

Table 2. Platform configurations

Item	Value
GPU Type	Nvidia Tesla K20c
GPU Core Config	2496 CUDA cores, 706 MHz
GPU Resources/Core	Max. 2048 Threads (64 Warps, 32 Threads/Warp), 48KB Shared Memory, 65536 Registers
GPU Memory	5120MB Global Memory, 2600MHz Memory Clock Rate, 320-bit Memory Bus Width
CPU Type	Intel Xeon E5530
CPU Core Config	8 Cores, 2.4GHz

3.1 Experimental Setup and Methodology

We characterize AlexNet, GoogLeNet, and VGGNet on Nvidia's Tesla K20c GPU, which is deployed at DELL PowerEdge R710 server. The detailed parameters are listed in Table 2. The Nvidia driver version is 340.32 and CUDA version is 6.5. We gather GPU runtime information using Nvidia Visual Profiler [31]. We use 8 slave nodes and 1 master node in our distributed framework characterization. The CNN models are trained by Caffe [20], an open-source deep learning framework widely used in both academia and industry. We also use Caffe to implement our CNN networks. The test data is from ILSVRC2012 (157.3GB) [11]. We scale up the above data sets appropriately with increased number of computing nodes.

We develop a state-of-the-art heterogeneous MapReduce framework that enables CPU to cooperate with GPU to do big data processing. We make heavy modifications to Hadoop framework so that it is capable of running multimedia workloads using GPU. Note that although we choose Hadoop in our current implementation, our proposed design and optimization can be integrated with other popular distributed computing frameworks such as Spark [43].

In this paper we define GPU temporal utilization as the proportion of GPU-involved runtime in the whole CNN runtime and GPU spatial utilization as GPU hardware resource utilization.

3.2 Framework Semantic Gap

3.2.1 Distributed Framework Overheads

We set out to analyze the nontransparencies in CNN workflow for distributed framework that cause the overheads in an inefficient integration of distributed CNN acceleration platform.

In a typical CNN-based application, the processing workflow consists of three stages: neural network initialization, data pre-processing and data processing. In our baseline implementation, each Hadoop map task is associated with a CNN processing process. While due to Hadoop job consists of a large amount of independent map tasks with short lifecycle, naïvely associating the whole deep learning process with map task incurs several overheads. We elaborate these overheads that are illustrated in Figure 3.

Repeated Network Initialization: Each map task will initialize the neural network. Since the network initialization is really time- and resource- consuming, this causes a large amount of unnecessary repeated network initialization time. These are depicted as *Init* in Figure 3.

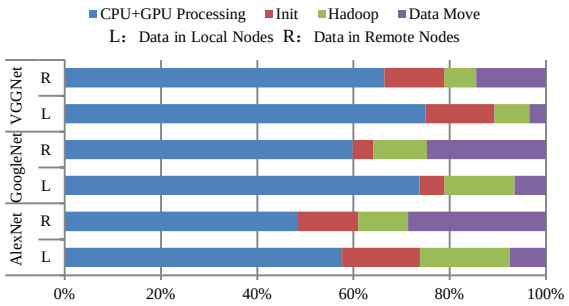


Figure 3. Overheads of distributed framework

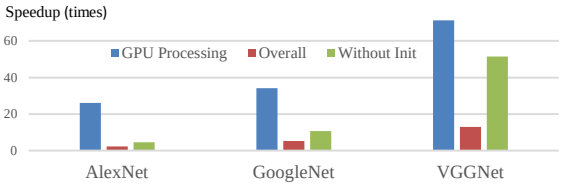


Figure 4. GPU speedup over CPU

GPU Calling Procedure: The launch of GPU processing in Hadoop also introduces overhead. The GPU processing program is invoked by the following steps [39]. First, task-tracker sends task requests to jobtracker using heart beat. Then, the jobtracker assigns certain tasks to the tasktracker based on the information of heart beat. After receiving commands from the jobtracker, tasktracker launches a child JVM to run the map task. Finally, the map task calls CUDA program via JNI [17] to start the processing. The calling procedure contributes to the overhead of *Hadoop* in Figure 3.

Hadoop Load Balance: When running multimedia workloads, the heterogeneity of nodes aggravates the cluster performance. This is because the Hadoop scheduler will execute load shedding by moving data from low-performance nodes to high-performance nodes to balance the overall application execution time, while the data processing stage will be blocked until the data preparation fetches data from remote nodes. Such waiting time is a great waste of computing resource. Figure 3 illustrates the impact of the *data movements* on the overall performance of a Hadoop cluster. We observe that fetching data remotely costs up to 29% application runtime. Under the distributed framework, the performance of CNN-based application could be expressed as:

$$T_{distribute} = T_{standalone} + t_{data\ move} + t_{ds} + t_{init} \quad \text{(Eq2)},$$

where $t_{date\ move}$ indicates the time spent on the data transfer to local disk, t_{ds} is the overhead of distributed framework and t_{init} is the overhead of repeated network initialization. The equation shows that more non-GPU related operations are introduced in the distributed framework, which leads to a low GPU temporal utilization.

Implication: Due to the inefficiency of the framework the non-computation related overheads account for up to 51% overall runtime, as shown in Figure 3. These overheads are mostly introduced due to the whole CNN process is associated with a map task. The task mapping mechanism in conventional distributed framework should be optimized for CNN based data processing flow. In addition, the modern heterogeneous distributed framework calls for efficient load balance to avoid wastes of GPU resource. An analytical model based processing time estimation for processing node would greatly optimize the task dispatch among heterogeneous processing nodes with various processing capacities and reduce the data movement among nodes. This motivates us to delve into lower standalone processing node level to explore the opportunity to achieve this goal.

3.3 Standalone Semantic Gap

To bridge the framework semantic gap, we opt to trap into the details of standalone CNN acceleration platform implementation to identify overheads and opportunities to provide insights for implementing predictive and efficient GPU-accelerated CNN processing.

3.3.1 GPU Temporal Inefficiency

We begin with a simple image recognition benchmark on three typical CNN networks to identify where the speedup bottlenecks locate at the standalone level. To maximize the system throughput, we set the input data as a batch of 64 images from ImageNet. We report the speedup of computation part, overall task with/without counting in network initialization stage in Figure 4. We observe that although GPU efficiently accelerates the computation (average 44X), the overall speedups are still very low (average 7X). Even if the network initialization time is not counted in, there is still much room for further optimization (average 60%). We then investigate the time distribution of overall tasks across all three networks and find that the GPU computation accounts for less than 20% of the task execution duration, while the most of the application runtime is consumed by network initialization (loading parameters from storage) and CPU processing (image data pre-processing). Therefore, the CNN application runtime on a standalone system could be described as (without counting in network initialization):

$$T_{standalone} = t_{cpu} + t_{memcpy} + t_{gpu} \quad \text{(Eq3)},$$

where t_{cpu} is the time of CPU pre-processing, t_{memcpy} indicates the time cost of copying parameters from CPU to GPU, and t_{gpu} is the GPU processing time. In the Eq3, the non-GPU related operations dominate the performance of CNN-based applications and GPU is idle during the most of the application execution time. Therefore, the low GPU temporal utilization leads to low overall speedup. We name this bottleneck as GPU temporal inefficiency.

Implication: Our investigation suggests that current CNN service framework requires higher computation proportion to reach desirable throughput. This could be achieved by shrinking the network initialization time and increasing the number of input batched image package in the entire CNN execution workflow.

3.3.2 Multiple Processes Contention

The multi-process technique has been introduced to improve the low data throughput. For example, Nvidia's Multi-Process Service (MPS) [29] is developed to allow multiple kernels running concurrently on shared GPU resource pool. However, our investigation suggests that current task-level parallelism solution is not optimal for deep learning applications [19, 28]. First, CNN based applications are memory-intensive. Running multiple GPU processes may exhaust GPU memory. We characterize the memory consumption by varying the batch size of input data. Figure 6 demonstrates that a single process of VGGNet could consume around 5GB GPU memory when the batch size is 64. In this case, three processes, even with MPS-enabled, could easily run out of state-of-the-art GPU memory (e.g. 12GB per K80).

Moreover, the interference and resource contention could severely degrade the performance of multiple processes. Table 3 presents

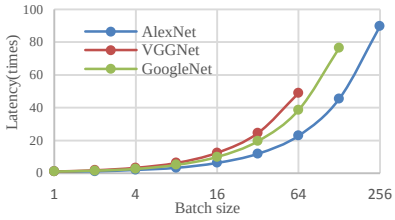


Figure 5. Batch size vs. latency

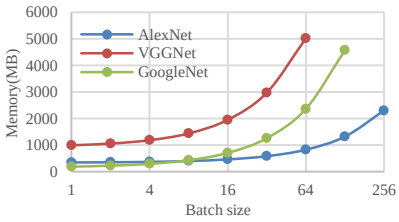


Figure 6. Memory usage

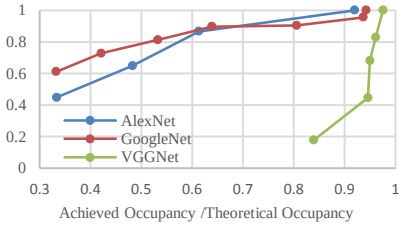


Figure 7. CDF of Achiv/Theo occupancy

Table 4. The critical kernels

Application	Kernel		Percent	Block Size	Registers /Thread	Shared Memory/Block	Warps Limit	Theoretical Occupancy
AlexNet	MM1	sgemm_sm35_ldg_nn_64×16×64×16×16	72.5%	[16, 16, 1]	71	8.27KB	24	37.5%
GoogLeNet	MM1	sgemm_sm35_ldg_nn_64×16×64×16×16	66.3%	[16, 16, 1]	71	8.27KB	24	37.5%
VGGNet	MM1	sgemm_sm35_ldg_nn_64×16×64×16×16	20.4%	[16, 16, 1]	71	8.27KB	24	37.5%
	MM2	sgemm_sm35_ldg_nn_128×8×128×16×16	37.3%	[16, 16, 1]	127	8.145KB	16	25%
	MM3	sgemm_sm35_ldg_nn_64×16×128×8×32	21.2%	[8, 32, 1]	115	12.27KB	16	25%

the computation throughput (CT: images/s) of one GPU process, two GPU processes, and two GPU processes with MPS-enabled. We can observe that the throughput of two processes is much lower than the theoretical throughput (2X). Even with the MPS-enabled, the multi-process only gain limited improvement. Here we use a metric, *Kif*, expressed as Eq4, to quantify the degree of interference. The larger *Kif* is, the more interference among multiple processes. The optimal value of *Kif* is 0, which means no resource contention.

$$K_{if} = \frac{CT_{single-process} \times \text{the number of processes}}{CT_{multi-process}} - 1 \quad \text{(Eq4)}.$$

To identify the reasons of interference, we begin by investigating the GPU critical kernels of CNNs using Nvidia Visual Profiler. We define the critical kernels as the most time-consuming kernels within an application. We gather their information and corresponding micro-architectural characteristics and summarize the results in Table 4. The kernels of Matrix Multiplication (MM) account for more than 66 percent of execution time across three CNNs. For Nvidia Tesla K-Series GPUs, the matrix multiplications are conducted by the kernel of Single-precision General Matrix Multiply (SGEMM) from cuBLAS [7]. However, the MM kernel is register-intensive. Due to the demanding register request for each thread each SM is limited to simultaneously execute 24 warps in AlexNet and GoogLeNet, and 16 warps in VGGNet. The limited warps indicate limited available computing resource. The limited available computing resource shared with multiple processes leads to contentions among concurrent executing kernels.

Implication: While the notion of employing the multi-process technique to improve the throughput appears appealing, our results indicate that current multi-process solution lacks proper memory and interference management. The number of concurrently running process in multi-process implementation should also be considered judiciously. Running too few processes may lead to waste of GPU resource. While over employing processes could incur severe GPU resource contention, even memory overcommit. Though a profiled based method is helpful to determine the number of processes, this is impractical in distributed cluster-based implementation due to the diversity of CNN workloads and a large number of slave nodes. An analytical model based criterion would greatly improve the efficiency.

Table 3. Throughput and K_{if}

	2 P	2 P in MPS	1P	K_{if}	K_{if} in MPS
AlexNet	348	436	289	0.7	0.3
GoogLeNet	125	156	116	0.8	0.5
VGGNet	43	49	42	1.0	0.7

3.3.3 Inefficiency of GPU Acceleration Library

Our previous characterization implications motivate an analytical model to provide accurate performance estimation for distributed framework and process number selection in multi-process technique. In this section, we explore the computation patterns of CNN workloads and GPU acceleration library to find the answer.

Bottlenecks: We start by characterizing the convolution layers in CNN workloads. To better quantify the performance of CNN convolutional layers, we define throughput efficiency, *tpE*, which indicates the GPU computational efficiency of each layer, as:

$$tpE = \frac{Throughput}{GPU\ Peak\ Throughput} \quad \text{(Eq5)}.$$

Since the throughput of each convolutional layer could be calculated by dividing the number of executed floating point (FP) instructions by the GPU time, Eq5 can be expressed as:

$$tpE = \frac{\text{Conv}_{flops}}{GPU\ time \times GPU\ Peak\ Throughput} \quad \text{(Eq6)}.$$

We examine the GPU computational efficiency of the three networks at each layer using *tpE* and the results are shown in Figure 8. We observe that the *tpE* in most of the convolutional layers is less than 60%. In AlexNet, none of the layers has a *tpE* greater than 40%. The poor *tpE* indicates the inefficiency of current GPU computational operations for CNN applications.

Root Causes: To identify the root causes of the inefficiency of GPU computational operations, we calculate the amount of computation of each layer based on Eq1. As shown in Figure 9, most layers have a small amount of computation. The low amount of computation leads to GPU resource under-utilization (i.e. low GPU spatial utilization), which results in low occupancy in kernel level. We further characterize the ratio of achieved occupancy and theoretical occupancy, which quantifies the GPU resource utilization. The results are presented as Cumulative Distribution Function (CDF) in Figure 7. Each point represents the kernels with the same GPU resource utilization in a network. For example, 60% of the kernels have GPU resource utilization below 50% (GPU spatial inefficiency) in AlexNet and GoogLeNet. This leads to the low *tpE* value in Figure 8.

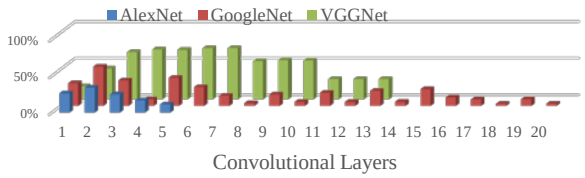


Figure 8. Throughput efficiency for each CONV layer

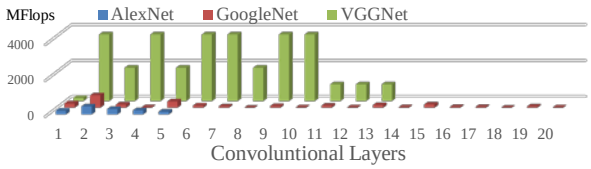


Figure 9. Varying computations of convolutional layers

Intuitively, batching multiple images/frames into a larger frame at each layer is usually adopted to improve the amount of computation in CNNs. However, due to the uneven computation amount at each CNN layers, using fixed batch size for all layers is not desirable. We conduct an experiment to compare the non-batching method with batching method based on cuBLAS. As shown in Figure 10, we observe that for those layers with ideal *tpE* (>60%), batching method does not help to further improve computational efficiency, while significantly increases the latency. Figures 5 and 6 illustrate the variation in latency and memory with the change of batch size. Without caution, the batching method will consume more memory and increase latency.

Implication: Due to the non-uniform computation amount across convolutional layers, we need to design a balanced scheme to improve *tpE* with less latency and memory usage. With the improved *tpE*, the resource utilization of GPU hardware could be pushed to near limit. This also could benefit the building of a performance estimation analytical model, because the throughputs of most convolutional layers are proportional to the GPU peak performance under this situation.

3.4 Summary

To sum, the inefficiencies of CNN-based applications stem from the GPU temporal inefficiency in framework level and the GPU spatial inefficiency in architecture level. The large number of non-GPU involved operations in Eq2 leads to GPU temporal inefficiency, which results in unsatisfactory speedup. At the architecture level, the small computation throughput of convolutional layers results in low GPU spatial inefficiency. These observations point us towards a hierarchical design that waxes the GPU resource supply-demand mismatches at the architecture level and eliminates distributed overheads at the framework level.

4. OVERCOMING BOTTLENECKS: BUILDING HIGH THROUGHPUT GPU ACCELERATION FRAMEWORK

Motivated by our characterization experiences, we present *D*^{3}*NN*, a Distributed, Decoupled, and Dynamically-tuned GPU acceleration framework for modern CNN network architectures. *D*^{3}*NN* features three novel designs.

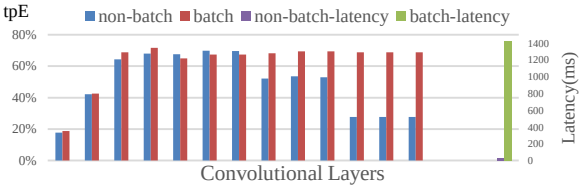


Figure 10. Non-batch vs batch in VGGNet

1. A CNN oriented GPU acceleration library that **dynamically** selects the optimal batch size of input data for each CNN layer.
2. An analytical model that enables accurate GPU processing time estimation in standalone GPU processing node. The analytical model provides insights for a load balance free distributed framework design and profiling-free multi-process technique.
3. A distributed data processing framework that **decouples** the CPU based data preparation and GPU-based data processing in CNN applications.
4. A contention mitigation scheme that alleviates the GPU resource contention caused by task-level parallelism.

4.1 Dynamic Batch Size Tuning Scheme

We first bridge the performance gaps among convolutional layers by designing a dynamic batching GPU acceleration library. We then design an analytical model that is able to accurately estimate GPU processing time.

In section 3 we show that using fixed batch size for all CNN layers may not be beneficial. Given the uneven computation amount at different CNN layers, two questions should be answered. First, *when should we apply the batching method?* Second, *how to select the best batch size if the batching method is adopted?*

Using batching or not, it highly depends on the GPU resource supply and demand at each CNN layer. We define a metric, *cpRatio*, to reflect the proportion of demanded resource to available resource at given CNN layer:

$$cpRatio = \frac{GridSize}{maxBlocks} \quad \text{(Eq7)},$$

The maximum number of blocks (maxBlocks) is defined in Eq8, which is mainly determined by the registers per thread (i.e. *r* in Eq8) and total number of registers per SM:

$$\text{maxBlocks} = \left\lfloor \frac{Total\ Registers\ per\ SM}{r * BlockSize} \right\rfloor \times num\ of\ SM \quad \text{(Eq8)}.$$

The GridSize (the number of blocks of MM kernel in a given layer) is defined in Eq9. Since in the matrix multiplication based convolution, MM kernel divides the result matrix into sub-matrices, with each sub-matrix mapped to a block.

$$\text{GridSize} = \left\lceil \frac{Result\ Matrix\ Size_{each\ layer}}{Sub-matrix\ Size_{MM\ kernel}} \right\rceil \quad \text{(Eq9)}.$$

We use *cpRatio* to indicate whether the batching method is adopted: If measured *cpRatio* is bigger than 1, then all available GPU resource (blocks) is consumed by CNN kernel. Therefore the batching method is not necessary for this layer. As shown in Figure 11, in CNN layers whose *cpRatios* are bigger than 1, the non-batching method performs better. The batching method in the last six layers achieves better performance than non-batching, where the measured *cpRatios* are all less than 1.

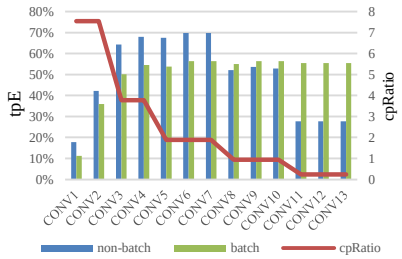


Figure 11. Effectiveness of batching method in VGGNet

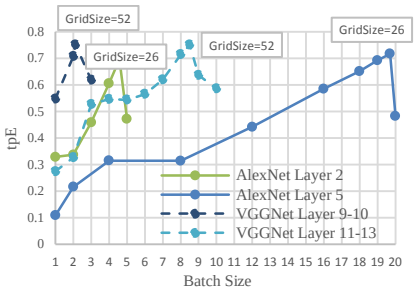


Figure 12. Performance variation

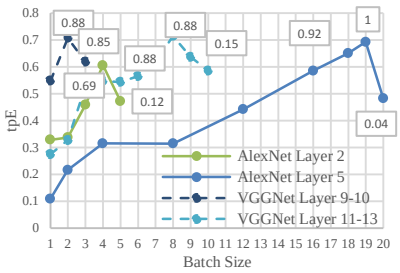


Figure 13. *Util* in different batch sizes

To identify the optimal batch size, we first exhaustedly characterize convolutional layers' performance using different batch sizes in Figure 12. We can observe the performance of CNN layers do not increase monotonically with the increasing of batch size. At the peak, its kernel's GridSize is a multiple of maxBlocks (26 for K20c), which means GPU MM kernel will achieve the best performance when it fully utilizes GPU resource. The performance of each CNN layer reaches the peak only if the value of GridSize is integer multiples of the maxBlocks. This implies the *cpRatio* is desired to be chosen as an integer to avoid performance degradation. We design a metric *Util* to indicate whether the batch size is optimal.

$$\text{Util} = \begin{cases} \text{Bc}pRatio & if \quad \text{Bc}pRatio > 0 \\ 1 & if \quad \text{Bc}pRatio = 0 \end{cases} \text{ (Eq10),}$$

where: $\text{BcpRatio} = \text{cpRatio} \bmod 1$. Figure 13 validates this policy: the point with the maximum value of *Util* always indicates the best batch size at given CNN layer. For example, the optimal batch size for AlexNet Layer No.2 is 4, where the value of *Util* is the maximum 0.88.

Based on our dynamic batching method, we design analytical models to estimate the CNN computing time and memory copy time swiftly. Since the dynamic batching method could boost the throughput of most of the convolutional layers to the maximum throughput, we can estimate the runtime of each convolutional layer based on its maximum throughput:

$$t_{CONVi} = \frac{Batch\ Size \times CONV_{flops}}{maxFLOPS \times Util} \text{ (Eq11),}$$

and then estimate the GPU computing time as:

$$t_{gpu} = \frac{\sum_{i=1}^{N} t_{CONVi}}{\delta} \quad \text{(Eq12),}$$

where δ is the ratio of convolutional layers in CNN. The memory copy time can be described as:

$$t_{memcpy} = \frac{\text{InputSize} + \text{ParameterSize}}{Throughput_{PCIE}} \quad \text{(Eq13),}$$

where InputSize is the size of pre-processed images and ParameterSize is the size of CNN parameters (mainly composed of parameters of fully connected layers).

$$\text{ParameterSize} = \sum_{i=1}^{N} N_{f_i} \times S_{f_i}^2 \times N_{c_i} \text{ (Eq14).}$$

4.2 Distributed and Decoupled Framework

D^3*NN* exploits three novel mechanisms for the efficient data processing in the distributed data processing framework: (1) Producer-Consumer scheme that decouples the data preparation and data processing at each slave node to hide the data preparation latency; (2) Semaphore-based data synchronization scheme that ensures the maximum single node throughput; and (3) Analytical model-based resource allocation that maximizes resource utilization.

Comments: *M* consumers work on a slave node and the depth of semaphore is *N*, which means *N* consumers can access GPU simultaneously

```
1:   sem_data = 1      //access data pool atomically
2:   sem_GPU = N     //N consumers share a GPU, N <= M
3:   sem_prod = 2M   //prepare 2 data for each consumer
4:   sem_cons = 0     //consumer will be waked-up by producer
```

Figure 14. Semaphores initialization

```
1:   down → sem_prod
2:     copy data from HDFS to data pool
3:     down → sem_data
4:       make data available to consumers
5:     up → sem_data
6:   up → sem_cons            //wake up a consumer
```

Figure 15. Producer workflow

```
1:   create CNN and do the initial work
2:   while(1) {
3:     down → sem_cons
4:       down → sem_data
5:         select a data from data pool
6:       up → sem_data
7:       load and pre-proceesing by CPU
8:       down → sem_GPU
9:         GPU do processing
10:      up → sem_GPU
11:    up → sem_prod    //notify producer to prepare next data
12:    pid = fork() //fork a new thread to upload result to HDFS
13:    if(pid ==0)
14:      child thread upload result to HDFS
15:  }
```

Figure 16. Consumer workflow

Producer-Consumer scheme: The overall architecture of *D*^{3}*NN* distributed framework is shown in Figure 17. Compared to baseline system in Section 3.2, we re-organize the task execution flow at slave node by separating the deep learning based multimedia process into data preparation process and data processing process. We employ a data preparation engine (Producer) and a data processing engine (Consumer), which run simultaneously on each slave node. The data preparation engine consists of map tasks, which act as data preparation processes and are responsible for downloading data from distributed storage to local disk. The data processing engine consists of data processing

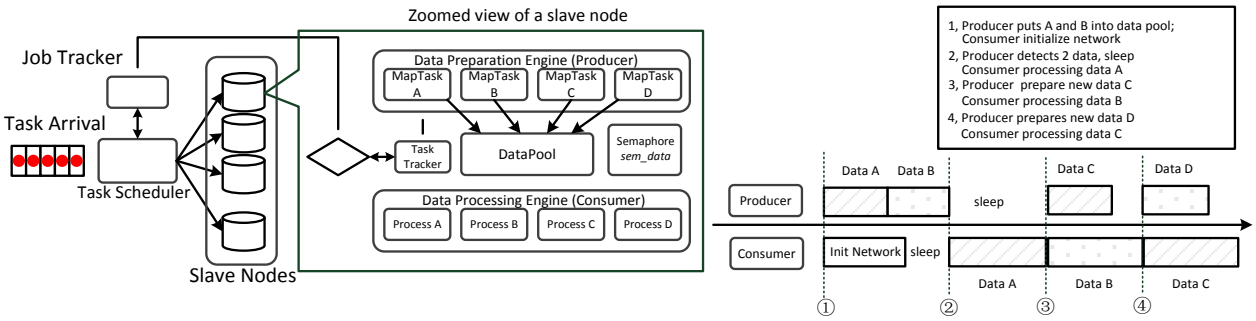


Figure 17. An overview of D[3]NN workflow

processes, which are in charge of data processing and network initialization. We employ a data pool to buffer the data produced by data preparation engine, which is a critical region for the producers and consumers to access data atomically.

Semaphore-based data synchronization scheme: We design a synchronization mechanism using a semaphore to coordinate the data preparation (Producer) and data processing (Consumer) in Figures 14-16. First, the producer starts to put data into data pool. At the same time, the consumer completes initial work and enters the sleep state to wait for *sem_consumer*. Once the producer completes a task of data preparation, it will trigger the consumer to process the incoming data. When the consumer is processing current data, the producer continues to prepare the next data for it. If there are more than 2M available data in data pool, the producer will enter the sleep state until the consumer completes a task of data processing and increases *sem_producer*. In this way, the consumer will keep pace with the producer.

A typical workflow of data preparation engine and data processing engine is presented in Figure 17. By running data preparation task and data processing task in parallel with different phases, the data processing time is well overlapped with data preparation time. Since the consumer services are never terminated, the network initialization only needs to be executed once, thereby eliminating considerable overheads. Therefore, the gaps resulted from distributed framework has been bridged and the distributed processing time is mainly dominated by the time of standalone CNN processing. Now Eq2 can be rewritten as:

$$T_{distribute} = T_{standalone} = t_{cpu} + t_{gpu} + t_{memcpy} \quad \text{(Eq15)}.$$

An analytical model based resource allocation: As mentioned in our motivations, the analytical model plays a critical role to bridge the semantic gap in scale-out CNN based big data processing platform. We leverage our analytical model to estimate the processing capacity of each computing node in distributed framework. This helps us to avoid great overheads on passive load balance among heterogeneous cluster, and always keeps the high utilization of heterogeneous hardware.

The analytical model could also guide us to select optimal process number in multi-processing technique with the lowest contention. Based on the analytical model, we can estimate the current GPU temporal utilization as:

$$gpuUtil = \frac{t_{gpu}+t_{memcpy}}{T_{standalone}} \quad \text{(Eq16)}.$$

Therefore, the optimal process number equals to *1/gpuUtil*. And this could boost GPU utilization to nearly 100%. We will further discuss this interference management scheme.

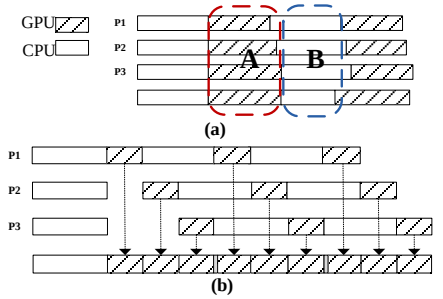


Figure 18. Contention management scheme

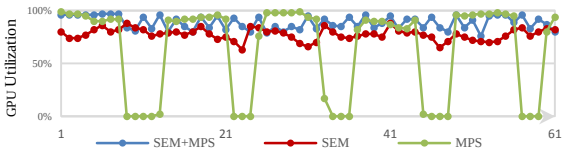
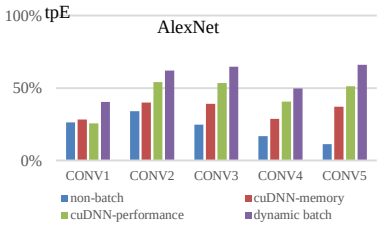


Figure 19. Real measurement of GPU utilization

4.3 Multi-Processing Contention Mitigation

As discussed in Section 3.3.2, the task-level parallelism of CNN based processes will cause intensive interference. In region A of Figure 18(a), all processes contend for GPU resource almost simultaneously, thus leading to interference and GPU memory over-committing. However, the GPU utilization decreases to zero in region B, as all processes are in their data preprocessing stage executed by CPU. This imbalance of GPU usage inevitably causes low GPU utilization. We also show the measurement of GPU utilization statistics with MPS-enabled in Figure 19. We observe a noticeable intermittent pattern in GPU utilization.

We propose a contention mitigation scheme to handle task-level parallelism bottlenecks. As shown in Figure 18(b), our contention management scheme pipelines the GPU processing tasks to avoid GPU resource contention, thus improves GPU utilization. Since K_{if} is high among CNN based applications, running too many processes does not help improve the data processing throughput. The contention mitigation scheme first calculates the optimal number of processes (*1/gpuUtil*) using Eq16. It then provides an access mechanism based on semaphore primitive. For contention-intensive applications (e.g. VGGNet), the initial value of the semaphore is 1, so that only one process can access the GPU resource at a time. After this process completes GPU processing and releases the semaphore, another process immediately acquires the semaphore and accesses GPU resource. In this way, the contention could be avoided and GPU remains busy as shown in *SEM+MPS* of Figure 19.



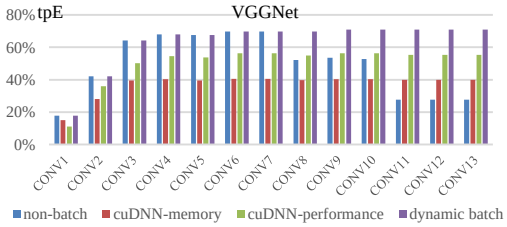


Figure 20. The comparison of throughput efficiency among CONV layers

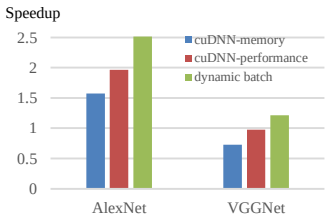
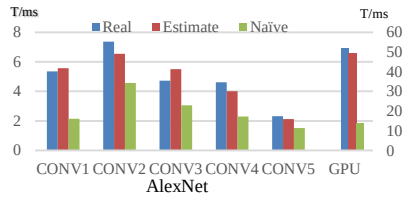


Figure 21. Speedup over non-batch



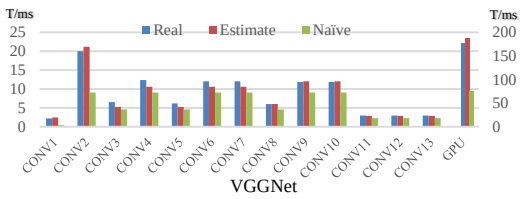


Figure 22. Evaluate the accuracy of analytical model

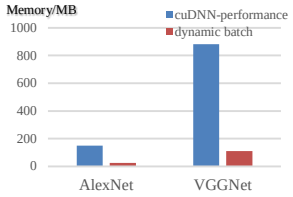


Figure 23. Memory usage

Note that the decoupled architecture is not simply designed for distributed cluster. This design also applies to a single node design with single GPU or multiple GPUs. This is because the GPU idle time is inevitable in single node design with CPU pre-processing. Worse, the current multi-process scheme for GPU incurs severe interference because of the GPU resource contention.

5. EVALUATION

5.1 Architecture Level Evaluation

We evaluate our dynamic supply-demand optimization scheme on two typical CNNs (AlexNet and VGGNet). In our dynamic method, matrix multiplication is based on cuBLAS and the stretching out the local regions into column vectors is implemented by the kernel im2col. Table 5 illustrates the batch size of each layer calculated using Eq10. Note that we avoid choosing the batch size as prime number (though it may be optimal) at each layer, because a big prime number causes a huge overall batch size. Here the overall batch size for a CNN is the least common multiple of batch size in each layer. Since different layers have different batch size, each layer needs to run $\frac{\text{overall batch size}}{\text{batch size of this layer}}$ times to perform an overall processing.

We compare our dynamic batch method with the non-batch method and batch method used in cuDNN, which is a state-of-the-art DNN library developed by NVIDIA [19]. The cuDNN now allows control over the balance between performance and memory footprint using different algorithms [9]. We include these two policies in our comparison. One is IMPLICIT_GEMM, which uses no extra working space and is memory-efficient. The other is GEMM and it is the fastest approach. Under the same overall batch size, we report the throughput efficiency (*tpE*) of the non-batch method, cuDNN (with memory-efficient and performance-preferred policies) and our dynamic batching method at each layer in Figure 20. Experimental results (Figure 21) show that our dynamic batching scheme can achieve up to 1.5X performance improvement compared with the one without batching. Our method even outperforms the state-of-the-art deep learning library (cuDNN) by up to 67% (memory-efficient policy) and 28% (performance-preferred policy). Moreover, as shown in Figure 23, our method consumes less GPU memory than cuDNN with performance-preferred policy. To summary, our analysis suggests that dynamic batching is an effective method to tackle the imbalance among multiple layers for CNNs with less latency and memory usage.

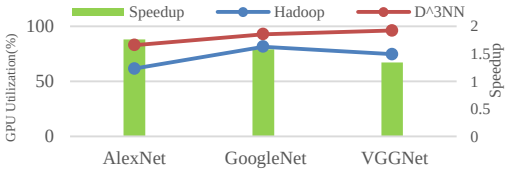


Figure 24. Hadoop vs D^3NN

Since the analytical model plays a critical role to bridge the semantic gaps in scale-out CNN based big data processing platform, we continue to evaluate the accuracy of our analytical model. Here we leverage the analytical model to calculate the runtime then use it to compare with real runtime and a naive method. Figure 22 depicts the runtime comparisons of the per-convolutional layer and total GPU. By dividing the number of floating point operations at each layer by peak performance of specific GPU, the naïve method only obtains 27-40% accuracy. Our results show that we can predict the performance of CNN with only 5-10% error.

5.2 System Level Evaluation

We evaluate the effectiveness of D^3NN in terms of system performance and load balance.

Performance: We evaluate the performance improvement of D^3NN under the three CNNs. We first report the GPU utilization and application speedup in Figure 24. We observe that D^3NN outperforms our baseline significantly in GPU utilization (average 30% improvement) and runtime (average 1.5X speedup). We then evaluate the impact of the decoupled implementation on single node throughput. The results are shown in Table 6. Mode *Overall* denotes the overall image processing throughput on the standalone platform. Mode *No-Init* denotes the throughput without counting in the network initialization. Mode *Proc-only* denotes the throughput of data processing stage on a standalone platform, while excluding the data pre-processing and network initialization. Mode D denotes the throughput of a single slave node in D^3NN. We calculate the throughputs in Modes *Overall*, *No-Init* and *Proc-only* by leveraging the built-in tool in our processing program. The throughput in Mode D is calculated through the task statistics

Table 5. The recommend batch size for each layer

	AlexNet		VGGNet	
CONV Layers	**cpRatio**	**Batch size**	**cpRatio**	**Batch size**
1	1.82	None	7.54	None
2	0.22	4	7.54	None
3	0.15	18	3.77	None
4	0.08	9	3.77	None
5	0.05	18	1.89	None
6	X	X	1.89	None
7	X	X	1.89	None
8, 9, 10	X	X	0.94	2
11, 12, 13	X	X	0.24	8

Table 6. The comparison of throughputs (images/s)

Slave	**K20c**			
Mode	**Standalone**			**Single Node**
	Overall	**No-Init**	**Proc-only**	**D**
AlexNet	39	75	289	350
GoogLeNet	43	59	116	125
VGGNet	18	31	42	43

from Hadoop. Our results shows: (1) the throughput of D gains remarkable improvement (up to 7.9X) compared to the throughput of Overall and even has 3.7X speedup over the throughput of *No-Init*, which justifies our decoupling optimization. (2) The throughput of D outperforms the throughput of *Proc-only*, which implies that the GPU has been fully utilized.

Load balance: We further evaluate the effectiveness of *D³NN*'s load balance capability. In this experimental setup, we deploy *D³NN* on a cluster with three different GPU computing nodes (Titan X, K20c, and M2050), and compare the performance of *D³NN* with the scheme of equally allocating tasks to each node (*Equal* in short). In *D³NN*, the tasks are allocated based on the ratio of their performance derived from our analytical model. The ratio of load balance is calculated by the number of completed tasks for each node divided by the node's processing capability, which is the real processing throughput profiled at standalone mode. The results are shown in Table 7. We can see that the ratios of D3NN are close to 1, which means *D³NN* has a superior load balance capability. For the Equal scheme, since the total running time of the cluster is determined by the slowest node (M2050), the runtime will be up to 2.6X more than *D³NN*.

6. RELATED WORK

Deep Learning: Sirius [16] is an open end-to-end intelligent personal assistant based on DNN services. The DjiNN [15] further brings the community the characterization of GPU acceleration server system running DNN services and provides insights into designing future warehouse-scale computer architectures for DNN services. Our work distinguishes itself from DjiNN in four aspects: (1) We perform comprehensive characterizations of GPU acceleration platform instead of computation scope and explore the root causes of low system throughput; (2) We disclose the inefficiency of multi-process technique and propose semaphore-based optimization; (3) We propose dynamic batching scheme and an analytical model for CNN based GPU processing. (4) We present a decoupled distributed framework to facilitate the GPU acceleration at scale.

Several recent projects [3, 4, 12, 27, 44] begin to use FPGAs and ASICs as the accelerators to achieve high performance at low energy. Most of them are on-going efforts and could not be flexibly deployed and programmed at large-scale. Recently, Nvidia released its deep learning library (cuDNN) [5] based on batching method. However, it does not consider using dynamic batching to tackle the compute-intensive CNN, such as VGGNet.

Table 7. Evaluation of load balance

Task Allocation		**Titan X**	**K20c**	**M2050**
AlexNet	Equal	0.51	0.94	2.62
	D³NN	0.96	0.98	1.00
GoogLeNet	Equal	0.48	0.94	2.58
	D³NN	0.93	0.95	1.00
VGGNet	Equal	0.54	1.06	2.66
	D³NN	0.93	0.95	1.00

GPU Task Execution: Sethia et al. [35] proposed Equalizer, a low overhead hardware runtime system that dynamically adapts the resource to the needs of running kernel. When a GPU kernel is running, Equalizer could adjust the number of concurrent thread blocks based on the runtime profiling information. Equalize will not improve the performance of CNN with small batch size because of its underutilization of GPU resource. To fully utilize GPU resource, our proposed techniques determine the GridSize of kernel based on the problem size of each CNN layer before the kernel is executed. Therefore, Equalizer could be incorporated into our proposed schemes to further ensure that GPU resources match the requirement of the executing kernels. Similar to our work, Xu et al. [40] also proposed optimization for task execution in CPU-GPU heterogeneous systems to improve the utilization of GPU resources. However, their optimization objective is maximizing the schedulability of real-time tasks (i.e., the amount of work of real-time tasks that can be accomplished before deadlines), while our objective is maximizing the throughput of the whole CPU-GPU system.

7. CONCLUSION

This work presents the first in-depth empirical study on characterizing the performance of GPU acceleration system for CNN applications. Our characterization results demonstrate two significant semantic gaps: framework gap and standalone gap. Framework gap indicates the mismatch between CNN-based data processing workflow and data processing manner in current distributed framework. The uneven computing loads at different CNN layers and fixed computing capacity provisioning of current GPU acceleration library results in the standalone gap. Motivated by our characterization findings, we propose *D³NN*, a Distributed, Decoupled, and Dynamically tuned GPU acceleration framework for modern CNN architectures. More importantly, *D³NN* features a novel analytical model that enables accurate time estimation of GPU accelerated CNN processing with only 5-10% error. Our evaluation results show the throughput of standalone processing node using *D³NN* gains up to 3.7X performance improvement over current GPU acceleration platform.

8. ACKNOWLEDGMENTS

This work is supported in part by NSF grants 1527535, 1423090, 1320100, 1117261, 0937869, 0916384, 0845721(CAREER), 0834288, 0811611, 0720476, by SRC grants 2008-HJ-1798, 2007-RJ-1651G, by Microsoft Research Trustworthy Computing, Safe and Scalable Multi-core Computing Awards, and by three IBM Faculty Awards. Jingling Yuan is supported by NSFC grant 61303029.

9. REFERENCES

[1] Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G. and Yu, D. 2014. Convolutional neural networks for speech recognition. IEEE/ACM Transactions on audio, speech, and language processing. 22, 10 (2014), 1533–1545.

[2] Amazon G2 instance: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using_cluster_co mputing.html.

[3] Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y. and Temam, O. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (2014), 269–284.

[4] Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z. and Sun, N. 2014. DaDianNao: A Machine-Learning Supercomputer. Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (2014), 609–622.

[5] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B. and Shelhamer, E. 2014. cuDNN: Efficient Primitives for Deep Learning. arXiv preprint arXiv:1410.0759. (Oct. 2014).

[6] CS231n: Convolutional Neural Networks for Visual Recognition: http://cs231n.github.io/convolutional-networks/.

[7] cuBLAS: https://developer.nvidia.com/cuBLAS.

[8] CUDA Profiler User's Guide: docs.nvidia.com/cuda/profiler-users-guide/.

[9] cuDNN v2: Higher Performance for Deep Learning on GPUs: http://devblogs.nvidia.com/parallelforall/cudnn-v2-higher-performance-deep-learning-gpus/.

[10] Dean, J. and Ghemawat, S. 2008. MapReduce : Simplified Data Processing on Large Clusters. Communications of the ACM. 51, 1 (2008), 1–13.

[11] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L. 2009. ImageNet: A large-scale hierarchical image database. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). (2009), 1097–1105.

[12] Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., Feng, X., Chen, Y. and Temam, O. 2015. ShiDianNao: Shifting Vision Processing Closer to the Sensor. Proceedings of the 42nd Annual International Symposium on Computer Architecture (2015), 92–104.

[13] Facebook, Ericsson and Qualcomm 2013. A focus on efficiency.

[14] Girshick, R., Donahue, J., Darrell, T., Berkeley, U.C. and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). (2014), 580–587.

[15] Hauswald, J., Kang, Y., Laurenzano, M.A., Chen, Q., Li, C., Dreslinski, R., Mudge, T., Mars, J. and Tang, L. 2015. Djinn and Tonic: DNN as a Service and Its Implications for Future Warehouse Scale Computers. Proceedings of the 42Nd Annual International Symposium on Computer Architecture (2015), 27–40.

[16] Hauswald, J., Laurenzano, M.A., Zhang, Y., Li, C., Rovinski, A., Khurana, A., Dreslinski, R.G., Mudge, T., Petrucci, V., Tang, L. and Mars, J. 2015. Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers. Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (2015), 223–238.

[17] He, W., Cui, H., Lu, B., Zhao, J., Li, S., Xue, J. and Feng, X. 2015. Hadoop+: Modeling and Evaluating the Heterogeneity for MapReduce Applications in Heterogeneous Clusters. Proceedings of the 29th ACM on International Conference on Supercomputing (2015), 143–153.

[18] Hu, Y., Li, C., Liu, L. and Li, T. 2016. HOPE: Enabling Efficient Service Orchestration in Software-Defined Data Centers. Proceedings of the 2016 International Conference on Supercomputing (2016), 10:1–10:12.

[19] Hu, Y., Song, M., Chen, H. and Li, T. 2016. Towards Efficient Server Architecture for Virtualized Network Function Deployment: Implications and Implementations. Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (2016).

[20] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T. and Eecs, U.C.B. 2014. Caffe : Convolutional Architecture for Fast Feature Embedding. Proceedings of the 22Nd ACM International Conference on Multimedia. (2014), 675–678.

[21] Karpathy, A. and Fei-Fei, L. 2015. Deep visual-semantic alignments for generating image descriptions. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). (2015), 3128–3137.

[22] Karpathy, A. and Leung, T. 2014. Large-scale Video Classification with Convolutional Neural Networks. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2014), 1725–1732.

[23] Kim, Y. 2014. Convolutional neural networks for sentence classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). (2014), 1746–1751.

[24] Krizhevsky, A., Sutskever, I. and Hinton, G.E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. Advances In Neural Information Processing Systems. (2012), 1097–1105.

[25] Li, C., Hu, Y., Liu, L., Gu, J., Song, M., Liang, X., Yuan, J. and Li, T. 2015. Towards Sustainable In-situ Server Systems in the Big Data Era. Proceedings of the 42nd Annual International Symposium on Computer Architecture (2015), 14–26.

[26] Li, C., Hu, Y., Zhou, R., Liu, M., Liu, L., Yuan, J. and Li, T. 2013. Enabling Datacenter Servers to Scale out Economically and Sustainably. Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (2013), 322–333.

[27] Liu, D., Chen, T., Liu, S., Zhou, J., Zhou, S., Teman, O., Feng, X., Zhou, X. and Chen, Y. 2015. Pudiannao: A polyvalent machine learning accelerator. Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (2015), 369–381.

[28] Liu, L., Cui, Z., Xing, M., Bao, Y., Chen, M. and Wu, C. 2012. A software memory partition approach for eliminating bank-level interference in multicore systems. Proceedings of the 21st international conference on Parallel architectures and compilation techniques (2012), 367–376.

[29] MULTI-PROCESS SERVICE: https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf.

[30] NVIDIA CUDA Programming Guide: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[31] NVIDIA Visual Profiler: https://developer.nvidia.com/nvidia-visual-profiler.

[32] OpenCL: http://www.khronos.org/opencl/.

[33] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision. 115, 3 (Sep. 2015), 211–252.

[34] Sermanet, P., Kavukcuoglu, K., Chintala, S. and Lecun, Y. 2013. Pedestrian detection with unsupervised multi-stage feature learning. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2013), 3626–3633.

[35] Sethia, A. and Mahlke, S. 2014. Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution. Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (2014), 647–658.

[36] Simonyan, K. and Zisserman, A. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR. abs/1409.1, (2014).

[37] SoftLayer offers Nvidia's most powerful GPU as-a-service: http://www.datacenterdynamics.com/app-cloud/softlayer-offers-nvidias-most-powerful-gpu-as-a-service/94407.fullarticle.

[38] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. 2015. Going deeper with convolutions. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). (2015), 1–9.

[39] White, T. 2012. Hadoop: The definitive guide. " O'Reilly Media, Inc."

[40] Xu, Y., Wang, R., Li, T., Song, M., Gao, L., Luan, Z. and Qian, D. 2016. Scheduling Tasks with Mixed Timing Constraints in GPU-Powered Real-Time Systems. Proceedings of the 2016 International Conference on Supercomputing (2016), 30:1–30:13.

[41] Yang, L., Luo, P., Loy, C.C. and Tang, X. 2015. A Large-Scale Car Dataset for Fine-Grained Categorization and Verification. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2015), 3973–3981.

[42] Youtube press statistics: http://youtube.com/yt/press/statistics.html.

[43] Zaharia, M., Chowdhury, M., Das, T. and Dave, A. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. (2012).

[44] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B. and Cong, J. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (2015), 161–170.

[45] Zhang, J., Sameki, M., Ma, S., Price, B., Mech, R., Shen, X., Betke, M., Sclaroff, S. and Lin, Z. 2015. Salient object subitizing. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2015), 4045–4054.

[46] Zhou, B. and Lapedriza, A. and Xiao, J. and Torralba, A. and Oliva, A. 2014. Learning Deep Features for Scene Recognition using Places Database. Advances In Neural Information Processing Systems (2014).

[47] Zhou, R., Chen, H. and Li, T. 2015. Towards Lightweight and Swift Storage Resource Management in Big Data Cloud Era. Proceedings of the 29th ACM on International Conference on Supercomputing (2015), 133–142.