Large-scale Data Analysis and Deep Learning Using Distributed Cyberinfrastructures and High Performance Computing

Richard Dodge Platania
Louisiana State University and Agricultural and Mechanical College, rplata1@lsu.edu

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations
Part of the Other Computer Engineering Commons

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_dissertations/5003

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
LARGE-SCALE DATA ANALYSIS AND DEEP LEARNING USING DISTRIBUTED CYBERINFRASTRUCTURES AND HIGH PERFORMANCE COMPUTING

A Dissertation

Submitted to the Graduate Faculty of the Louisiana State University and Agricultural and Mechanical College in partial fulfillment of the requirements for the degree of Doctor of Philosophy in The Department of Computer Science

by

Richard Dodge Platania
B.S., Louisiana State University, 2013
August 2019
Acknowledgments

This dissertation and the work within it would not have been possible without the help and support of many people. I would like to thank my major advisor, Dr. Seung-Jong Park, for being a leader and mentor to me throughout my journey through graduate school. Also, many thanks to my committee members, Dr. Jian Zhang, Dr. Seungwon Yang, and Dr. Michal Brylinski for serving and providing guidance towards my research. Additional thanks to other faculty who have assisted me with research, including Dr. Kisung Lee, Dr. Joohyun Kim, and Dr. Nayong Kim. I’d also like to give a special thanks to my friends and fellow students, Shayan Shams, Sayan Goswami, Arghya Das, and Dipak Singh, whom I could not have made the journey without. I want to also thank my family for remaining extremely supportive throughout my graduate studies. Further appreciation is necessary for my animal companions, Harkness, GB, and Davros, for keeping me company and providing emotional support.

Lastly, I want to thank my loving and patient wife, Molly, for supporting me through the ups and downs of this adventure. Without her, it would not have been possible.
# Table of Contents

ACKNOWLEDGMENTS ................................................................. ii  
LIST OF TABLES ........................................................................... v  
LIST OF FIGURES .......................................................................... vii  
ABSTRACT ....................................................................................... ix  

CHAPTER  
1. INTRODUCTION ............................................................................. 1  
  1.1. Data Analysis ......................................................................... 1  
  1.2. Deep Learning ....................................................................... 2  
  1.3. Large-scale Computing .......................................................... 3  
  1.4. Goal ..................................................................................... 4  
  1.5. Dissertation Outline .............................................................. 4  

2. HADOOP-BASED REPLICA EXCHANGE OVER HETEROGENOUS DISTRIBUTED CYBERINFRASTRUCTURES ................. 6  
  2.1. Introduction .......................................................................... 6  
  2.2. Background ......................................................................... 8  
  2.3. HaRE: Hadoop-based Replica Exchange .................................. 11  
  2.4. Performance Evaluation ....................................................... 17  
  2.5. Discussion ........................................................................... 22  
  2.6. Concluding Remarks ............................................................ 25  

3. EVALUATION OF DEEP LEARNING FRAMEWORKS OVER DIFFERENT HPC ARCHITECTURES ...................................... 27  
  3.1. Introduction .......................................................................... 27  
  3.2. Deep Learning Frameworks ................................................... 29  
  3.3. HPC Architectures ................................................................ 31  
  3.4. Evaluation ............................................................................ 32  
  3.5. Conclusion ........................................................................... 41  

4. PREDICTING ADHD USING 3D CONVOLUTIONAL NEURAL NETWORKS AND FMRI DATA ...................................................... 44  
  4.1. Introduction .......................................................................... 44  
  4.2. Related Work ....................................................................... 45  
  4.3. Methodology ....................................................................... 46  
  4.4. Experiment and Results ....................................................... 52  
  4.5. Conclusion .......................................................................... 55  

5. LARGE-SCALE DEEP LEARNING FRAMEWORK FOR REAL-TIME VEHICLE CLASSIFICATION OVER DISTRIBUTED CYBERINFRASTRUCTURE ........................................ 60
## List of Tables

2.1. Summary of simulated biological systems. ........................................... 12
2.2. Distributed computing resources experimented for HaRE .................... 14
2.3. Description of cluster hardware for single cluster scale-up and scale-out. ................................................................. 17
2.4. A comparison between the effect of a 16x increase of workload size on four node clusters at QB2, EC2, and CL:C (CloudLab Clemson). The number of mappers is fixed to 16, and the number of replicas used is 16 and 1024. ........................................... 19
2.5. Scale-across experiment with GENI. Benchmark results are shown from one cluster at one ExoGENI resource (Oakland, CA) and two clusters at two ExoGENI resources (Oakland, CA and Houston, TX). The time taken for network-heavy phases (input preparation and shuffle) are shown along with the remaining execution time. The total of the three gives the total execution time for one iteration of HaRE. Time is given in seconds. ............................... 23
3.1. Deep learning frameworks for evaluation. ........................................... 29
3.2. Hardware for deep learning evaluation................................................. 31
3.3. Deep learning models and datasets for experiments. ............................ 32
3.4. Benchmark for average time to train one batch with Caffe and AlexNet. The ILSVRC’12 dataset was used with a batch size of 256 .................... 35
3.5. Data transfer speeds using P100 with and without NVLink. .................... 35
4.1. Testing set classification results for ADHD-200 dataset. ....................... 54
4.2. The number of parameters and memory requirements for convolution, pooling, and fully connected (FC) layers in the 2D and 3D CNN models. 3D requires approximately 5 times more parameters for each layer and approximately 16 times more memory per training image. ................................................................. 55
5.1. Description of dataset. ................................................................. 73
5.2. Percentage of cars $a$ that are above varying IOU thresholds $t$. ................ 75
5.3. Fine-grained classification accuracy for RT-VDC. ................................. 76
5.4. Frames per second of different GPUs. The classification component of RT-VDC used the AlexNet-based model. The resolution of the video after processing is 608x608.
List of Figures

2.1. Replica Exchange (RE) scheme with Statistical Temperature Molecular Dynamics (STMD). Each replica runs a single STMD during $t_{EX}$ with a given temperature range, and a swap is attempted for a pair of replicas at every $t_{EX}$. This task is repeated until the entire trajectory is simulated. .................................................. 8

2.2. The schematic of HaRE presented (upper). Each round of MR contains concurrent replica runs, which is implemented in the Map phase, and an exchange task in the Reduce phase. Also, the iteration process of HaRE is described (lower). ............................... 12

2.3. Strong scaling out of HaRE from 16 to 1024 mappers (1 to 64 nodes) on the QB2 cluster. Each node has 16 mappers, and the total number of replicas is fixed to 1024. .................................................. 19

2.4. Weak scaling out of HaRE from 16 to 128 mappers (1 to 8 nodes) on QB2, EC2, and CL:C (CloudLab Clemson). Each node has 16 mappers and 16 replicas. .................................................. 20

2.5. Weak scaling up of HaRE from 1 to 32 mappers and replicas using a single node on QB2, EC2, and CL:W (CloudLab Wisconsin). .......... 21

2.6. Scale-across experiment with CRON. CRON is configured differently with 2, 3, and 4 nodes, for which a single cluster is built with up to 2 nodes. Two separate clusters (for configurations with 3 or 4 nodes) are connected with two different network connection conditions, fast and slow. These respective conditions are 10 Gbps with 60 ms latency and 100 Mbps with 30 ms latency. Two groups of experiments using $IMA$ and $IMB$ were conducted and compared. The time-to-solution for one round of MR is presented with a grey bar, and a black bar is the time-to-solution of the Map phase. $S_A$ is used for this experiment. ....................... 22

3.1. Scaling up with Caffe using AlexNet for GPU (P100 with and without NVLink) while increasing number of GPUs and batch size. .......... 32

3.2. Scaling up with Caffe using GoogLeNet for CPU while increasing the batch size. ................................................................. 33

3.3. Scaling up batch size on P100 with NVLink and KNL using Alexnet with Caffe............................................................... 34

3.4. Scaling up batch size on P100 with NVLink using GoogLeNet with TensorFlow and Caffe............................................... 37
3.5. Scaling up batch size on P100 with NVLink using VGG-19 Net with TensorFlow and Caffe. ......................... 38

3.6. TensorFlow and SINGA scale out results. Each node has one P100 GPU worker and each worker has a mini-batch size of 64. ................ 39

3.7. Caffe and TensorFlow scale out results. Each node is one KNL worker and each worker has a mini-batch size of 64. ..................... 40

3.8. Scale out result with Intel Caffe over P100 via InfiniBand and KNL via Omni-Path. For P100 results each node has one P100 GPU worker and for KNL result each node is one KNL worker and each worker has a mini-batch size of 64. ................ 41

3.9. Scale out result with TensorFlow over P100 via InfiniBand and KNL via Omni-Path. For P100 results each node has one P100 GPU worker and for KNL result each node is one KNL worker and each worker has a mini-batch size of 64. ................ 42

4.1. A visual representation of (a) 2D and (b) 3D convolutions. The blue squares/boxes represent the data and the red squares/boxes represent the receptive filter. In both examples, we see that the receptive filter maps data from three different channels, the only difference being the dimensions of the data and filter. ....................... 47

4.2. Example fMRI data from ADHD-200 dataset. ......................... 51

4.3. 3D CNN layer architecture. ........................................ 57

4.4. Training loss of 2D and 3D convolutional neural networks. ............ 58

4.5. Training accuracy of 2D and 3D convolutional neural networks. ........... 58

4.6. Scalability of 3D CNN. There are 16 workers per node. ..................... 59

5.1. An overview of RT-VDC. ........................................... 67

5.2. Distributed infrastructure of RT-VDC. .................................. 71

5.3. Visualizing the different IOU thresholds. ................................ 74

5.4. Detections and classifications from RT-VDC. .............................. 76
Abstract

Data in many research fields continues to grow in both size and complexity. For instance, recent technological advances have caused an increased throughput in data in various biological-related endeavors, such as DNA sequencing, molecular simulations, and medical imaging. In addition, the variance in the types of data (textual, signal, image, etc.) adds an additional complexity in analyzing the data. As such, there is a need for uniquely developed applications that cater towards the type of data. Several considerations must be made when attempting to create a tool for a particular dataset. First, we must consider the type of algorithm required for analyzing the data. Next, since the size and complexity of the data imposes high computation and memory requirements, it is important to select a proper hardware environment on which to build the application. By carefully both developing the algorithm and selecting the hardware, we can provide an effective environment in which to analyze huge amounts of highly complex data in a large-scale manner. In this dissertation, I go into detail regarding my applications using big data and deep learning techniques to analyze complex and large data. I investigate how big data frameworks, such as Hadoop, can be applied to problems such as large-scale molecular dynamics simulations. Following this, many popular deep learning frameworks are evaluated and compared to find those that suit certain hardware setups and deep learning models. Then, we explore an application of deep learning to a biomedical problem, namely ADHD diagnosis from fMRI data. Lastly, I demonstrate a framework for real-time and fine-grained vehicle detection and classification. With each of these works in this dissertation, a unique large-scale analysis algorithm or deep learning model is implemented that caters towards the problem and leverages specialized computing resources.
Chapter 1. Introduction

1.1. Data Analysis

Data has always driven research, and, with data sets reaching the terabyte scale, the strategies involved in storing and analysis of data have never been more important. For instance, genome sequencing technologies have advanced their throughput and capabilities to the point of producing terabyte-sized sequencing data for a single dataset. While the storing of this data is in itself a challenge, specifically with respect to hardware, the analysis requires both capable computing resources and algorithms. Because of these requirements, platforms have been developed that are capable of leveraging large-scale computing resources while abstracting the distributed computing practices from the user, thus keeping the algorithm development process as simple as possible. Platforms like this have made the data analysis process quicker and more accessible to researchers.

Since there are many different requirements for data analysis based on the type of data and task, many platforms have been developed with diverse focuses. For instance, large-scale batch processing of big data was first made widely popular by Hadoop MapReduce [1], which stems from the original MapReduce [2]. This tool enabled researchers to write code using the well-defined Map-Reduce programming paradigm while automatically distributing the data and computation across a cluster. Big data analysis continued with the development of many other open source tools, each with their own data analysis goals (e.g., Apache Spark [3] for in-memory distributed computation), and many of which exist within the Hadoop ecosystem. With the growing popularity in deep learning for data analysis, there was a demand for large-scale deep learning tools that could accommodate large data and models that can be trained in a reasonable amount of time. Out of this demand came tools like TensorFlow [4], which has enabled researchers to carry out deep learning research in a variety of computing environments with this flexible framework.
1.2. Deep Learning

Since its inception, deep learning has had the potential to be a powerful tool set for the analysis of data. Its birth occurred decades before it saw widespread use in research and industry. Ivakhnenko [5] was the first to propose and implement a deep network consisting of 8 layers. Initially, it proved difficult to apply deep learning methodology to more than small datasets on a small scale because of the computational requirements involved. Since then, the initial model proposed by Ivakhnenko has given rise to a large variation of models designed with different data and tasks in mind. Eventually, basic evolutions of this model were found to be capable of activities like handwritten digit recognition [6]. In the late 2000s and early 2010s, there was a surge in deep learning development due to the massive availability of powerful computational resources. In particular, the accessibility of deep-learning capable GPUs made this movement much more potent. Furthermore, the support of companies like NVIDIA to produce libraries dedicated towards deep learning on these high-performance GPUs helped further the deep learning movement.

Given this boom in deep learning research and development, many research fields have been advanced and sped up with respect to their data analysis capabilities due, in part, to the end-to-end nature of many of the deep learning models. In other words, the lack of requirement in hand-crafted features made the process easier and quicker for researchers to produce a turn around in their analysis. Researchers in a copious amount of fields have been either directly or indirectly influenced by deep learning. Within these fields, many tasks that have remained difficult for humans to perform are now possible through deep learning, often times with even higher accuracy (e.g., breast cancer detection and diagnosis). At this point, deep learning has extended its reach towards image recognition, natural language processing, drug discovery, bioinformatics, and many other domains.

In order to cater towards different datasets and tasks in these domains, many different deep learning models have been developed. Arguably, one of the most popular models is the Convolutional Neural Network due to its applicability towards image data. Its use
of filters that scan the input image make it possible to detect patterns across the entire image. Meanwhile, its use of weight sharing reduces the computational complexity, making the model easier to train. While its primary use is on image data, it has shown applicability to other types of data as well, such as DNA sequences [7]. Another model that has had a successful effect with respect to data analysis is the Recurrent Neural Network (RNN). Most notably, the Long Short-Term Memory (LSTM) model, a type of RNN, has shown great success in time-series and text analysis. Whereas CNNs consider the spatial relations very well in its inputs, LSTMs and RNNs specialize in considering the long-term dependencies or temporal relations in data. The number of new models or variations of existing models continues to grow along with a continuing demand for more deep learning applications in a variety of domains.

1.3. Large-scale Computing

The availability and improvement of computing resources and environments are what have supported the development and improvement of data analysis and deep learning research. Without developing algorithms catered towards these resources and their environments (e.g., parallelize algorithm for a scaled-out computational resource), research would not have progressed as quickly as it has in recent years. It is important for the continued development of both of large-scale resources and algorithms to go hand-in-hand in order for researchers to be able to continue to improve the speed and quality of their research. In addition to the hardware and software development for single-node data analysis and deep learning, efforts continue to be made to enable distributed computing in a usable and efficient manner.

Distributed computing for large-scale data analysis has been well established with the development of the Hadoop ecosystem toolkit. There is database support in the form of HBase, batch processing from the MapReduce programming model, in-memory computation from Spark, and many more, all behind an abstracted distributed computing and storage methodology. However, deep learning has not experienced a significant amount of
distributed computing support until recently. Initially, many deep learning frameworks, including TensorFlow [4], were not capable of distributed computing, essentially limiting models to that which could fit in a single compute node. Now, most of the popular frameworks offer options for scaling out or, at the very least, scaling up with multiple GPUs [4, 8, 9]. Beyond the software, certain hardwares were designed with throughput-heavy tasks, like training a neural network, in mind. The development of communication tools such as NVIDIA’s NVLink and Intel’s Omnipath were intended to alleviate the communication bottleneck exhibited by many of these large-scale data analysis and deep learning tasks. With the continued support and development of distributed computing resources and algorithms, large-scale analysis and deep learning will continue to grow and produce more impressive results in a timely manner.

1.4. Goal

The goal of this work is to develop algorithms capable of tackling barriers associated with using data analysis and deep learning for large-scale applications. This a challenging problem since the computational complexity and large memory overhead imposed by big datasets and models requires advanced deep learning techniques and hardware technologies to apply deep learning in an efficient manner. In addition, the hardware and software setup is very essential to deal with computational complexity and large memory overhead of big data and models. In this dissertation, I present works regarding large-scale data analysis and deep learning over distributed infrastructures. In more detail, the works focus on both evaluating the large-scale nature of applications in addition to evaluating the applications performance towards a scientific problem.

1.5. Dissertation Outline

The remainder of this proposal is as follows. Chapter 2 details the development and analysis of HaRE, a framework for Hadoop-based Replica Exchange simulations. The following chapter evaluates several deep learning frameworks in terms of scaling up and out in several computing environments with state-of-the-art hardware. Next, in Chapter 4, we
delve into a deep learning application using one of the frameworks evaluated in the previous chapter. This particular application aims to use 3D convolutional neural networks for diagnosing ADHD from fMRI data. The next chapter is about a large-scale framework for real-time and fine-grained vehicle detection and classification. Lastly, Chapter 6 concludes my dissertation and gives a brief description of future works.
Chapter 2. Hadoop-based Replica Exchange over Heterogenous Distributed Cyberinfrastructures

2.1. Introduction

Large-scale sampling of biomolecules is a continuously growing, computationally intensive endeavour. In recent years, many extensions from traditional Molecular Dynamics (MD) have been developed in order to address computation time and simulation complexities. The Replica Exchange (RE) scheme is a popular alternative which has been applied to many established MD packages. It imposes the simulation of several replicas of a system, each at a different temperature. At certain intervals, trajectory exchanges are attempted between pairings of replicas. Replica Exchange Statistical Temperature Molecular Dynamics (RESTMD) is a sampling algorithm recently introduced and is thought to be promising with many merits overcoming drawbacks of conventional sampling and enhanced sampling approaches [10, 11, 12, 13]. It combines the popular RE scheme with the recently developed Statistical Temperature Molecular Dynamics (STMD) algorithm.

As the complexity and size of simulations continues to increase, researchers need to look towards distributed cyberinfrastructures (DCI) in order to address the hardware limitations of single node execution. Simulation time is a key challenge that is often addressed through the use of MPI. However, this work is concerned with replica exchange, which involves infrequent, coarse-grained communication at set synchronization points (exchange attempts). MPI implementation in an application can be challenging, and the fine-grained communication is necessary for RE, which requires replica communication only at exchange attempts. In order to overcome this challenge, strategies aimed towards implementing task-level parallelism are required. In this work, the aim is to provide this parallelism to RE scheme.

In order to provide this task-level parallelism, this work makes use of the MapReduce
(MR) paradigm. Note that MR is a programming model initially introduced for large-scale distributed data processing [14], but here the use of this model for compute-intensive, more specifically, loosely-coupled applications, is demonstrated. Hadoop [1] provides an open source implementation of MapReduce. As demonstrated in this chapter, Hadoop appears to also be an excellent framework over HPC resources and cloud-based environments, greatly benefiting those applications identified as pleasingly and loosely-coupled [15, 16, 17], as well as other complex applications often relying upon solely MPI to achieve scalability.

This chapter describes the performance of HaRE over DCI and investigate contributing factors towards performance. By doing this, it is shown that Hadoop MapReduce is a capable model for loosely-coupled applications. HaRE is executed in multiple scenarios and environments over various DCI in order to show potential performance in real environments that researchers would be expected to execute similar applications. These environmental set-ups range from physical computing nodes over high-speed networks to virtual machines over low-speed networks, and the DCI types range from traditional, homogeneous HPC clusters to fully distributed, heterogeneous clouds. Utilizing a large variety of environmental set-ups and DCI allows for successful analyzing of HaRE performance.

The chapter is organized as follows. In the following section, background of of RESTMD and Hadoop are given, followed by an introduction to related works. Next, HaRE is introduced along with its RESTMD integration. The following section describes the architecture and environmental configuration of the DCI experimented in this work. Then, experimental results for scalability performance are shown. Discussions are presented regarding contributing factors to performance in different DCI, Hadoop as a task-level parallelism framework, and future works. Lastly, there is an insight towards future work and a brief conclusion.
Figure 2.1. Replica Exchange (RE) scheme with Statistical Temperature Molecular Dynamics (STMD). Each replica runs a single STMD during $t_{EX}$ with a given temperature range, and a swap is attempted for a pair of replicas at every $t_{EX}$. This task is repeated until the entire trajectory is simulated.

2.2. Background

2.2.1. Statistical Temperature Molecular Dynamics (STMD) and RESTMD

STMD itself is an enhanced sampling method, partly inspired by the Wang-Landau method, but becoming a distinctively different novel method, in which the theoretical definition of statistical temperature is the key idea as the name suggests [18, 19]. This new method, STMD, can explore configurational space of a broad range of temperatures, and thus becomes a multicannonical approach. For HaRE, two popular simulation packages, CHARMM [20] and LAMMPS [21], were modified for allowing STMD simulations, resulting in the support of all-atom simulations. Particularly, LAMMPS-STMD is significant for the research community since the package is well known for its scalability performance with the excellent MPI support, and thus there is a great potential for its use.

RESTMD is a replica exchange method which follows a similar swapping protocol of the popular replica exchange MD. The difference is that RESTMD runs a STMD simulation instead of straightforward MD for each replica. The schematic of simulation pipeline for replica exchange is shown in Fig. 2.1. RESTMD is not only a better sampling strategy but also has many advantages over REMD, currently a de-facto protocol. As one of its key merits, RESTMD requires fewer replicas to cover the same range of temperature
space, primarily because each replica simulation with STMD can explore a broad range of temperatures. Additionally, more efficient exchanges between replicas occur [19]. These aspects are particularly advantageous for sizeable biological molecules where conventional REMD suffers to scale as the size of a target system increases. On the other hand, they are also beneficial as a HPC application. In a given computational resources (i.e. the number of nodes), a less number of replicas means more nodes to individual replicas and thus can tackle a larger system. This, in addition to the better exchange rate, results in more efficient sampling, while saving computing time. Therefore, RESTMD is an attractive choice for most of current biomedical research topics aiming towards real biological processes in a living cell, since their typical system size is beyond what REMD is capable of.

2.2.2. Hadoop MapReduce

For the task-level parallelization of HaRE, the MapReduce (MR) programming model is used, which was initially introduced for large-scale distributed data processing [14]. Its popularity has been fuelled by Hadoop [1], a software framework comprising of the Hadoop Distributed File System (HDFS) and an open-source implementation of the MR programming model. Hadoop can be easily installed on most hardware, from commodity to high end, and has gained strong support from both the open source community and leading companies in the IT industry, resulting in numerous applications in various domains. Hadoop has already shown to be an attractive framework for large-scale data processing. It is demonstrated that it is also attractive for loosely-coupled applications, such as RESTMD, by executing HaRE in various scenarios.

Hadoop continues to have an extensive impact on scientific research, especially considering the continuously growing and robust software stack complementing MapReduce. Pig [22], for instance, provides a high level language for performing large-scale data analysis and has already been adopted for large-scale sequence data analysis as seen with BioPig [23]. Spark [3] provides an additional option for scientific data analysis with its robust data structures and iterative computation using scalable, in-memory machine learn-
ing, graph processing, and more. There are already several Spark-based approaches for analyzing sequencing data [24, 25]. While Spark and similar frameworks are emerging as front-runners in Big Data, this application utilizes simulation packages that require writing to and reading from files between replica exchange iterations, essentially negating any benefit that Spark’s in-memory iterative processing provides. For this reason, Hadoop MR’s user-friendly programming model is selected. As the Hadoop community continues to grows, there will be more software built for the Hadoop ecosystem that are aimed towards developing large-scale scientific applications.

2.2.3. Related Work

Previously, there have been several modifications and demonstrations of the Hadoop framework for large-scale applications. Some of these works aimed to provide APIs designed for domain-specific scientific applications [26]. While useful, this does not focus on the improvement of performance but rather the ease-of-use. Many of the others focused on adding an iterative functionality to Hadoop [27, 28]. HaLoop, for instance, implements loop-aware task scheduling for MR, but its development has not continued past the prototype [27]. Similarly, Twister provides iterative MR functionality but also allows for long-running Map and Reduce tasks that can continue execution beyond the typical barriers exhibited in traditional MR. This sort of asynchronicity has many applications in the scientific domain, but it does not apply well to replica exchange which requires barrier synchronization for replica exchange. Other Hadoop extensions are aimed towards improving streaming performance [29, 30]. However, these improvements all have a commonality of design for data-intensive applications. Instead, this work highlights the use of Hadoop for a compute-intensive scientific application.

Other works have focused on data locality and task scheduling. Some of these consider data locality or compute capacity in order to efficiently assign Map tasks and avoid data transfer [31], and some even transparently implement MR in a multi-cluster environment [32, 33, 34]. Alternatively, others work towards replication strategies for providing
improved data locality [35]. The motivation behind these works is based on the notion of avoiding transfer of large amounts of data across multiple clusters. This work shows that the compute-intensive application, HaRE, performs well without needing modifications to the Hadoop framework, even in multi-cluster environments.

In considering traditional Hadoop MR, there have been many scientific applications. CloudBurst [36], one of the earlier scientific applications of MapReduce, performs large-scale read-mapping of next-generation DNA sequence data, consequently demonstrating Hadoop’s applicability to data-intensive genome applications. While an extension of MR has been previously applied to MD simulation trajectory analysis [37], HaRE aims to utilize MR for performing simulations. In some previous works, an early implementation of MR-based RESTMD was introduced with initial results of scalability performance, STMD performance, and the performance of STMD implemented in two different packages (CHARMM and LAMMPS) [38, 39]. Traditional MD implementations using MR are difficult due to the fine-grained communication requirement imposed by MD. For this reason, it is uncommon to see MR-based MD simulations.

2.3. HaRE: Hadoop-based Replica Exchange

2.3.1. Architecture of HaRE

This replica exchange scheme for HaRE is implemented by using the MR model. In Figure 2.2 (upper), the execution for a single iteration of MR is shown. The Map phase is responsible for multiple, concurrently running STMD simulations. In particular, each mapper configures and executes LAMMPS-STMD. The following Reduce phase manages replica exchanges using outputs produced from prior STMD simulations in the Map phase. Figure 2.2 (lower) describes the iteration process of HaRE. The preparation step is needed before each round of MR to prepare inputs for STMD runs occurring in the next round.
Figure 2.2. The schematic of HaRE presented (upper). Each round of MR contains concurrent replica runs, which is implemented in the Map phase, and an exchange task in the Reduce phase. Also, the iteration process of HaRE is described (lower).

Table 2.1. Summary of simulated biological systems.

<table>
<thead>
<tr>
<th>Simulated System</th>
<th>( S_A )</th>
<th>( S_B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Solvated GB1 hairpin peptide</td>
<td>Solvated mini protein (Crambin)</td>
</tr>
<tr>
<td>Protein structure (PDB ID)</td>
<td>1GB1</td>
<td>1CBN</td>
</tr>
<tr>
<td>Number of residues</td>
<td>16</td>
<td>46</td>
</tr>
<tr>
<td>Number of water molecules</td>
<td>1699</td>
<td>2781</td>
</tr>
<tr>
<td>Total number of atoms</td>
<td>5356</td>
<td>9002</td>
</tr>
</tbody>
</table>

2.3.1.1. Two biological systems for simulations:

Simulations in this work considered two biological systems of two small proteins, as summarized in Table 2.1, that differ in their sizes. The different system size requires different computational workloads and affects the performance of each step along with the Hadoop-based workflow. In a previous conference paper [38], time-to-solutions were compared using these two molecular systems for two different implementations for RESTMD, in order to find a right decision on the implementation. Based on results, for HaRE, one implementation is used, called \( IM_B \), with the larger system, \( S_B \), unless specifically stated.
A standard procedure for all-atom MD simulations is employed for building a molecular system in a simulation box [40]. In brief, the initial structures of the proteins were obtained from the PDB database and inserted into explicit water molecules. The CHARMM force field, CHARMM22/CMAP, was used for the two proteins, and the TIP3P explicit water model was used to solvate the proteins.

2.3.2. HaRE over Distributed Cyberinfrastructures

Heterogeneous computing environments are becoming increasingly common and useful as researchers look to leverage the benefits of combining different hardware in the same cluster. Furthermore, the federation of multiple clusters to complete some task is become more feasible and popular with the proliferation of nationwide high-speed network backbones. Because of this, heterogeneous DCI are incorporated into experiments in order to evaluate the performance of HaRE in these different environments. Five different types of computing environments are utilized for demonstration and are displayed in Table 2.2. Conventional HPC clusters generally provide a somewhat consistent and predictable performance with physical nodes and high-speed intra-cluster connections (e.g., InfiniBand). On the other hand, the four distributed infrastructures, Amazon EC2, CRON, GENI, and CloudLab are distinctly different. Amazon EC2 is a compute-on-demand IaaS cloud environment that is recently widely utilized for research and science gateways [41]. CRON [42, 43] is a testbed system that can be configured to emulate a distributed computing infrastructure connected by configurable networking conditions. The remaining two resources, GENI and CloudLab, are both built on a nationwide infrastructure comprising of many locations. GENI (Global Environment for Network Innovations) is a virtual laboratory for networking, distributed systems, and security while also hosting applications [44]. CloudLab provides the ability to build cloud environments over baremetal resources [45]. Both of these infrastructures support the federation of rack resources using nationwide, high-speed networks.
Table 2.2. Distributed computing resources experimented for HaRE

<table>
<thead>
<tr>
<th>Compute Resource</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB2</td>
<td>HPC clusters</td>
<td>LSU/LONI HPC (<a href="http://cct.lsu.edu">http://cct.lsu.edu</a>)</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>IaaS Cloud</td>
<td>(<a href="http://aws.amazon.com">http://aws.amazon.com</a>)</td>
</tr>
<tr>
<td>CRON</td>
<td>Cloud</td>
<td>Testbed with reconfigurable networking</td>
</tr>
<tr>
<td>GENI</td>
<td>Networked DCI</td>
<td>Virtual laboratory for networking and distributed systems (<a href="http://geni.net">http://geni.net</a>)</td>
</tr>
<tr>
<td>CloudLab</td>
<td>Meta-cloud</td>
<td>Facility for building clouds (<a href="http://www.cloudlab.us">http://www.cloudlab.us</a>)</td>
</tr>
</tbody>
</table>

2.3.2.1. HaRE over HPC:

HPC are the most well developed and supported of the DCI discussed in this chapter. Because of this, HPC setup for HaRE is the most straightforward. For these experiments, the new LONI QB2 cluster was used [46, 47]. QB2 has 504 compute nodes with a peak performance of 1.5 Petaflops. Each QB2 node contains two 10-core 2.8 GHz E5-2680v2 Xeon processors, 64 GB memory, and 500 GB HDD. The nodes are connected using 56 Gbps InfiniBand connections.

2.3.2.2. HaRE over Homogenous Cloud (EC2):

Amazon EC2 is easily accessible and continues to gain popularity for a wide variety of applications. Consequently, this work demonstrates execution and performance of HaRE over EC2. EC2 provides a variety of virtual machine instance types and configurations to cater to different cloud applications. However, because HaRE is a compute-intensive application, the C4 family of instance types that focus on providing computation performance is chosen. Specifically, the experiments used instances consisting of 36 vCPU cores and 60 GB memory over Amazon’s Elastic Block Storage (EBS) SSD volumes. Unlike the other DCI, EC2 VMs do not provide the inherent benefit of dedicated physical CPU cores. Because of this, performance can be unpredictable and requires more identical executions for verification.
2.3.2.3. HaRE over Heterogeneous and Virtually Distributed Cloud (CRON):

As previously explained in this section, it is crucial to demonstrate the capability and performance of HaRE over heterogeneous environments, especially with the increasing popularity of heterogeneous cloud-based environments. With the combination of computing nodes with varying hardware, it is possible to leverage them by assigning tasks or applications to nodes which have more suitable specifications. So, this work makes use of the testbed system, CRON. While it was shown that HaRE setup for traditional HPC systems is straightforward, additional efforts for configuration and setup of HaRE over CRON are required in order to perform experiments in several different virtual environments. In brief, CRON is able to emulate a cloud computing environment, composed of two separate clusters connected by a configurable network. This environment is important for the purpose of this work since the scale-across scenario of HaRE over a heterogeneous multi-cluster Hadoop environment can be investigated.

The experimental set-up over CRON examined in this work consists of two clusters, interconnected by a reconfigurable network that emulates varying network conditions. One cluster is composed of up to two nodes, each having a 6-core Intel Xeon processor with 12 GB RAM and 10 Gbps NIC, and the other cluster is configured with up to two nodes, each equipped with a 4-core AMD Opteron processor with 8 GB RAM and 10 Gbps NIC. The two clusters are connected by two different network conditions: a fast 10 Gbps bandwidth with 60 ms latency configuration and a slow 100 Mbps bandwidth with 30 ms latency. Note that these configurations are intended to represent realistic conditions, a case with minimum bandwidth and a case with high bandwidth, respectively, commonly found in network conditions between production HPC systems.

2.3.2.4. HaRE over Homogeneous and Geo-Distributed Cloud (GENI):

GENI is a nationwide networking testbed connecting various universities and research institutions by making use of nationwide high-speed Layer 2 connectivity. It is new com-
pared to the previously discussed DCI but is quickly gaining popularity. For the purpose of this chapter, it is important to demonstrate HaRE over a physically distributed scale-across scenario, as virtually demonstrated with CRON. Two ExoGENI resource locations are used: Oakland, CA and Houston, TX.

Similar to CRON, the experimental set-up over GENI examined in this work consists of two clusters, interconnected by a high-speed, reconfigurable network. Each ExoGENI rack provisions virtual machines over IBM M4 servers. A total of four VMs were provisioned across two configurations. The first configuration consists of a single cluster located at an ExoGENI rack in Oakland, CA. More interestingly, the second configuration federates two geographically distant ExoGENI rack resources (Oakland, CA and Houston, TX) by creating a VLAN supported by a 10 Gbps connection. Due to a lack of network interfaces at ExoGENI and multihoming in MapReduce, the experiments are limited to a maximum of two clusters. Each VM consists of 4 CPU cores and 12 GB memory, which is comparable to the setup in CRON. For the two-cluster case, the two resource locations were chosen such that they were geographically distant and they supported the federation of rack resources over the previously mentioned 10 Gbps Layer 2 connection. By doing this, we will see how HaRE performs in an environment where both the computing resources and the network backbone is heavily shared. The network provides best-effort service, meaning that additional experiments are required for performance verification.

2.3.2.5. HaRE over Geo-Distributed HPC (CloudLab):

Adding HPC resources to a cluster is no longer limited by the locality of the additional resources. The increasing prevalence of high-speed network connections between research locations allows for a more seamless federation of external resources. CloudLab is used to investigate the performance of HaRE in an environment with hardware similar to that of the HPC investigation but with lower bandwidth and increased latency between the nodes. The effect of this network discrepancy on performance will be shown and demonstrate the overall viability of federating HPC resources for HaRE and similar applications. CloudLab
Table 2.3. Description of cluster hardware for single cluster scale-up and scale-out.

<table>
<thead>
<tr>
<th>Resource</th>
<th>CPU Cores</th>
<th>Memory</th>
<th>Disk</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB2</td>
<td>20 physical</td>
<td>64 GB</td>
<td>HDD</td>
<td>56 Gbps InfiniBand</td>
</tr>
<tr>
<td>EC2</td>
<td>36 vCPU</td>
<td>60 GB</td>
<td>SSD (EBS)</td>
<td>10 Gbps Ethernet</td>
</tr>
<tr>
<td>CloudLab (Wisconsin)</td>
<td>16 physical (32 with Intel Hyper-Threading)</td>
<td>128 GB</td>
<td>2xHDD SSD</td>
<td>10 Gbps Ethernet</td>
</tr>
<tr>
<td>CloudLab (Clemson)</td>
<td>20 physical</td>
<td>256 GB</td>
<td>2xHDD</td>
<td>40 Gbps InfiniBand</td>
</tr>
</tbody>
</table>

is part of the GENI federation network, which allows federation with both GENI and CloudLab resources. For this work, the Wisconsin and Clemson locations of CloudLab are utilized. The Wisconsin nodes each have two 8-core Intel E5-2630v3 processors with Hyper-Threading, 128 GB memory, two 1.2 TB HDD, and one 480 GB SSD. Each node has 10 Gb NIC, but the actual rate is throttled to 100 Mbps in order to represent an HPC environment connected over an external network. The Clemson nodes each have two 10-core Intel E5-2660v2 processors, 256 GB memory, and two HDD at 1 TB each. Each node has 40 Gbs InfiniBand but, like the Wisconsin nodes, it is throttled. Like GENI, the network provides best-effort service, requiring additional experimentation for performance verification.

2.4. Performance Evaluation

2.4.1. Scalability performance of HaRE

For the following scalability experiments, the number of replicas and mappers at each node are varied. The results for scale-up and scale-out were produced from the QB2 cluster, EC2, and the CloudLab testbed. The results for scale-across are from the CRON, GENI, and CloudLab testbeds. Here, the scalability performance with respect to the scale-up, scale-out, and scale-across scenarios are presented.

2.4.1.1. Scale-up and scale-out in a single cluster with HPC and EC2:

In a single cluster environment, it is essential to know the scalability as additional nodes are utilized for the target computation. In addition to this scale-out scenario, an
understanding of scale-up performance, addressing how more computational loads can be added in a single node, is also important for potential scalability. The single cluster hardware configurations for these experiments are described in Table 2.3. In Fig. 2.3, the strong scale-out results over QB2 are presented. The configuration used for this is to allow 16 mappers per node while fixing the number of replicas to 1024. The number of mappers increases up to 1024. The results suggest a good linear scaling, indicating good scalable performance of HaRE as more nodes are utilized in a cluster. In addition to traditional HPC, it is crucial to understand the performance over both cloud-based clusters. For this reason, the scalability of QB2 against EC2 and CloudLab scale-out experiments are compared, as shown in Fig. 2.4. In this experiment, the number of replicas increases along with the number of mappers from 16 to 128. Interestingly, EC2 suffers a 1.6x increase in execution time from one to two nodes (16 to 32 mappers) while both QB2 and CloudLab experience good scaling. This result is reasonable considering that both QB2 and CloudLab have direct access to physical hardware, there is lack of control over EC2 instance rack locality, and EC2 networks are less predictable.

On the other hand, in Fig. 2.5 and Table 2.4, results are presented relevant to scale-up performance. First, the workloads are increased within a single node by increasing the number of mappers and replicas from 1 to 32. Until the number of mappers and replicas reaches 16, good scalability is achieved. The number of concurrently executing mappers and the number CPU cores have a direct effect on performance. Increasing the number of concurrent mappers begins to exhibit a diminishing effect on the overall performance in HaRE. Recall that the number of available cores per node in the QB2 cluster is 20, CLW is 16, and EC2 is 36 (vCPU), implying that the number of mappers exceeding the number of cores may have an effect on the scale-up threshold. Note that the number of physical cores used in EC2 is unknown, but we clearly see a drop in performance before reaching the vCPU threshold of 36. In addition to scaling up a single node, a four-node cluster is set up with each node scaled up individually. Considering a scenario like this is necessary
Figure 2.3. Strong scaling out of HaRE from 16 to 1024 mappers (1 to 64 nodes) on the QB2 cluster. Each node has 16 mappers, and the total number of replicas is fixed to 1024.

Table 2.4. A comparison between the effect of a 16x increase of workload size on four node clusters at QB2, EC2, and CL:C (CloudLab Clemson). The number of mappers is fixed to 16, and the number of replicas used is 16 and 1024.

<table>
<thead>
<tr>
<th>Resource</th>
<th>64 Replica</th>
<th>1024 Replica</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB2</td>
<td>273</td>
<td>3971</td>
<td>14.54x</td>
</tr>
<tr>
<td>EC2</td>
<td>402</td>
<td>5692</td>
<td>14.16x</td>
</tr>
<tr>
<td>CL:C</td>
<td>346</td>
<td>4951</td>
<td>14.31x</td>
</tr>
</tbody>
</table>

because the use of Hadoop in strictly single-node scenarios is uncommon. Table 2.4 shows the performance of increasing the number replicas of a four-node cluster from 64 to 1024, a 16x increase, with the number of concurrently executing mappers per node remaining at 16. As expected, the execution time varies between the three resources, but it appears that the time is increased by a similar factor in all three cases.

2.4.1.2. **Scale-across with two network-connected clusters in CRON and GENI:**

One of the important contributions for this work is the demonstration of the scale-across scenario with unmodified Hadoop, the core of HaRE. Note that Hadoop is generally
Figure 2.4. Weak scaling out of HaRE from 16 to 128 mappers (1 to 8 nodes) on QB2, EC2, and CL:C (CloudLab Clemson). Each node has 16 mappers and 16 replicas.

considered as a framework for distributed, data-intensive computing in a single cluster, thus this work’s set-up and demonstration for a compute-intensive application over multiple distributed resources is an intriguing experiment. As described in the previous section, the CRON-based distributed system, emulating the network-connected two clusters, was built and used for scale-across experiments. Two different network configurations, designed to mirror realistic network conditions, are tested with varying number of nodes. Obtained results are presented in Fig. 2.6. The fast and slow configurations previously described are used as the number of nodes varies. The Map phase has virtually no change between the slow and fast network connections. Interestingly, the time-to-solution of a whole MR varies considerably depending upon the configuration of the two-cluster system and two implementations, $IM_A$ and $IM_B$. These two implementations were described extensively in the conference paper [38], and in brief, represent two possible implementation strategies that differ in to what extent the local file system is utilized. Striking difference is observed between the two. A possible cause is likely to be associated with network speed and underlying file I/O tasks affecting the non-Map tasks. For this, one needs to remember
that in $IM_A$, local file systems across the network should be accessed as required by its design, revealing its apparent connection to the increment in time-to-solution as the network speed becomes slower. Interestingly, the cases with $IM_B$ do not show such a pattern. $IM_B$ uses HDFS as a primary means to store information about simulations and exchanges, seemingly revealing its better scalability performance is less affected by the varying network connection. Additionally, this interpretation is further supported with the observation that an additional increase of the MR time is observed when the entire system goes from the 3-node configuration to 4 nodes, since the change suggests more file I/O operations in the case of $IM_A$ over the network connection.

It is important to see that $IM_B$ also performs well in an environment where two clusters are connected in a geo-distributed fashion. For this purpose, two configurations were set up over the GENI testbed. The first configuration consists of a single cluster of four nodes at a single resource location (ExoGENI at Oakland, CA). The second configuration creates a VLAN by adding a second cluster using a second, geographically distant resource (ExoGENI at Houston, TX). In this case, each cluster has two nodes. The execution time of
Figure 2.6. Scale-across experiment with CRON. CRON is configured differently with 2, 3, and 4 nodes, for which a single cluster is built with up to 2 nodes. Two separate clusters (for configurations with 3 or 4 nodes) are connected with two different network connection conditions, fast and slow. These respective conditions are 10 Gbps with 60 ms latency and 100 Mbps with 30 ms latency. Two groups of experiments using $IM_A$ and $IM_B$ were conducted and compared. The time-to-solution for one round of MR is presented with a grey bar, and a black bar is the time-to-solution of the Map phase. $S_A$ is used for this experiment.

The network-heavy MR phases are shown along with the total execution time. These results are presented in Table 2.5. The execution of the shuffle phase of MR, the phase in which data is transferred over network from mappers to reducers, is shown in blue. Typically, this is a huge bottleneck in MR applications. However, because HaRE is very compute-intensive and not very data-intensive, this phase shows only a slight increase between the two configurations. The input preparation, shown in black, involves STMD input preparation on the HDFS between HaRE iterations. We see a more significant increase here, but the remaining time spent for other phases (Map, Reduce, etc.) dominates the overall execution time, making this time increase less significant.

2.5. Discussion

The main contributions in this chapter include i) the development of HaRE with Hadoop MapReduce as the basis for task-level parallelism of replica exchange, ii) the configuration and execution of HaRE over several DCI of varying hardware and environment, iii)
Table 2.5. Scale-across experiment with GENI. Benchmark results are shown from one cluster at one ExoGENI resource (Oakland, CA) and two clusters at two ExoGENI resources (Oakland, CA and Houston, TX). The time taken for network-heavy phases (input preparation and shuffle) are shown along with the remaining execution time. The total of the three gives the total execution time for one iteration of HaRE. Time is given in seconds.

<table>
<thead>
<tr>
<th># of Clusters</th>
<th>Preparation</th>
<th>Shuffle</th>
<th>Other</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>317.5</td>
<td>331.5</td>
</tr>
<tr>
<td>2</td>
<td>34.5</td>
<td>10</td>
<td>326.5</td>
<td>371</td>
</tr>
</tbody>
</table>

the demonstration of scalability via scale-out and scale-across in distributed clusters, and iv) the identification and discussion of factors contributing to performance behavior in different configurations. Additionally, several configurations of HaRE and environmental factors that effect performance are discussed below.

2.5.1. Contributing factors to HaRE performance over DCI

Hadoop configuration is perhaps the most vital component in achieving the best performance for HaRE. The most important factor is likely the number of concurrent mappers per node. Because of the compute-intensive nature of STMD simulations, it was important to consider the computational load of the simulation. From there, the correct number of concurrent mappers could be determined such that the CPU is maximally utilized without harming execution time. It is equally important to not under-utilize the CPU. The scale up results show performance gain from scaling up mappers to the number of CPU cores. In particular, we can consider the four-node scale-up experiment shown in Table 2.4. Even though there is a 16x increase in the number of replicas, the time taken at each DCI only increases by a factor of 14.16-14.51. Thus, utilizing both scale-up and scale-out techniques are required for maximizing the overall performance of HaRE.

The replica exchange scheme configuration, in particular the pairing of replicas for exchange attempts, was initially believed to have an impact on the shuffle phase of MR. Paired replicas always go to the same reducer, so the pairing scheme will ultimately determine the number of connections between datanodes. However, in this case, different replica pairing strategies ultimately showed no significant gains for this computationally intensive
application. Because of this, replicas were paired based on neighboring temperature ranges, which is an approach that benefits the biological results of HaRE. However, more data intensive applications with a pairing dependency may see more significant changes to shuffle time. In these cases, it would be good to make use of some of the previously described related works that optimize data locality and scheduling [31, 32, 34, 33].

The EC2 scale-out results suggest a degradation of performance when transitioning from one to multiple instances. Although EC2 attempts to provide locality of instances, there is no guarantee that they will all be hosted in the same rack or even the same data center. So, spanning to multiple nodes can impose a large networking overhead. Furthermore, the default configuration for instances is shared tenancy, meaning that an unknown number of other users may be sharing the same hardware resources. This can further slow down inter-instance communication.

The GENI experiments revealed that network configurations had an increasing affect on input preparation and shuffle times. However, the shuffle time experienced an insignificant increase as shown in Table 2.5. Interestingly, the STMD input preparation required by HaRE at each iteration showed the most significant change. Even though the input preparation deals with extremely small files, HDFS I/O is dependent on the network and will ultimately experience a reduction in performance across clusters. Still, the scale-across results with CRON showed good scaling despite the additional preparation overhead.

2.5.2. Hadoop as a task-level parallelism framework for HaRE

Scientific HPC applications are commonly grouped as pleasingly parallel, loosely-coupled, and tightly coupled [15]. Coarse-grained parallelism strategies are effective for the former two, while fine-grain parallelism might be the right choice for the latter. A previous work showed the use of MR for a bioinformatics application, the alignment of short reads onto a reference genome, belonging to the pleasingly-parallel application and thus suitable for data parallelization [17]. In this work, applications of the replica exchange scheme, categorized as loosely-coupled ones, are focused and showed that MR is also a viable approach for
task-level parallelization. The result is a combination of the aforementioned coarse-grained parallelism for replica exchange along with fine-grained parallelism for STMD simulations through MPI. Indeed, a lower developmental cost and benefits of well-established Hadoop and MR are attractive for many HPC applications of similar algorithmic characteristics. As examined in the conference paper [38] and in this chapter, an understanding of underlying mechanisms coupled with characteristics of computational environments is crucial for better-performing implementations of Hadoop-based applications.

2.5.3. Broader impact and future works

Many outcomes from this work and presented in the conference paper [38] for the development of HaRE can be unequivocally useful for other applications algorithmically employing the replica exchange scheme. In addition to REMD, those applications include other parallel tempering approaches such as Metropolis-Coupled Markov Chain Monte Carlo ($MC^3$) in statistical learning and inference [48, 49]. Another example is a class of applications using massively parallel monte carlo methods which is increasingly gaining a lot of interests in in data analytics for Big Data. Most importantly though, HaRE’s task-level parallelism and scalability are demonstrated, implying that, for similar compute-intensive applications, the MR model, or other Hadoop-based programming models supported by software stacks, such as Spark, Hama, Tez, and others, fits well since it manages iterative tasks of parallel jobs with minimal inter-communications or global synchronization.

2.6. Concluding Remarks

Recently, methods for performing large-scale scientific applications have been gaining a lot of interest. Providing scalable solutions for researchers across widely available computing resources is of key importance. Hadoop remains a pivotol framework for creating scalable, data-intensive applications. This chapter describes HaRE, a Hadoop-based replica exchange implementation of RESTMD, and it is demonstrated that it provides good task-level parallelization and scalability with the use of the unmodified MapReduce programming model. Results with scale-up and scale-out, using multiple cluster and hardware congu-
rations over HPC, Amazon EC2, and CloudLab, imply that HaRE performs comparably in each DCI and even demonstrates linear scaling in many cases. Through the scale-across scenarios with CRON and GENI, the effect of network and geographical distribution on the network-heavy MapReduce phases of HaRE is investigated. Along with these results, factors contributing to performance variation between DCI were uncovered and discussed. In summary, Hadoop MapReduce benefits RESTMD by implementing task-level parallelism that scales well over many resource types and configurations. With this knowledge, Hadoop can be applied to similar scientific applications, those exhibiting computationally-intensive, loosely-coupled tasks, with the flexibility to scale well independent of resource selection.
Chapter 3. Evaluation of Deep Learning Frameworks over Different HPC Architectures

3.1. Introduction

Deep learning has continued to thrive as the availability of capable computing resources becomes more common. Many fields are now able to make use of deep learning for their particular applications. These applications include biomedical image analysis, social media analysis, and many more. New deep learning techniques and applications are constantly being developed as they continue to see success through many domains. With this spurring of development, many different software and hardware have been developed that cater towards deep learning workloads. There are a few existing works that focus on the comparison of deep learning hardware performance from speed and scaling perspectives. In order to more effectively compare the variety of software tools, they should be benchmarked with these metrics over various hardware environments. In this work, we aim to do this using three deep learning frameworks, several HPC environments, and state-of-the-art hardware technologies.

The trending development of deep learning tools is providing users with more options that can cater specifically to their needs. Typically, these needs stem from available hardware resources. For instance, a user with access to a large-scale commodity cluster may aim to utilize a different deep learning tool than a user with access to a single multi-GPU machine. This isn’t to say that one single tool isn’t suitable for multiple environments. Rather, the different tools were created to include features that benefit certain hardware setups. Caffe is a popular deep learning framework that excels at single- or multi-GPU training on a single machine, making it more accessible to the general user [9]. On the other hand, Apache SINGA strives to provide scalable deep learning in a distributed environment [50, 51]. Combining multiple ideologies, TensorFlow looks to perform well on a
variety of platforms, including scaled-up CPU or GPU machines, scaled-out clusters, and mobile devices [4]. There are several works that benchmark the performance of these and other frameworks with respect to batch training time [52, 53]. Caffe in particular has been thoroughly evaluated. However, works concerning the evaluation and comparison of TensorFlow, SINGA, or other distributed deep learning tools through scaling are limited. We aim to contribute to the already existing benchmarks with scaling results. The three aforementioned frameworks are used in this work. Many other tools have been developed with certain performance goals in mind, and we plan to explore these in future works [54, 55].

Since many of the recently developed software for deep learning were created with specific computing environments in mind, it is also important to consider the ongoing advancement of related hardware technologies and their impact on software performance. The spotlight tends to remain on the advancement of general purpose GPUs because of their high performance vector computation and deep learning’s reliance on such. However, it is still important to visualize CPU and other computing hardware capability with respect to deep learning. It is interesting to see if the recent and upcoming technologies, such as Intel’s Knight’s Landing Xeon Phi processor with Omni-Path interconnect, can compete with GPU in deep learning. On top of this, GPU memory is limited, and the size of data continues to grow. Hence, CPU or other technologies may see increased use in deep learning. GPU-related technologies are beginning to adapt to overcome this issue and the data transfer bottleneck it imposes on GPU-based training. Previous works have been done to evaluate hardware performance for deep learning, and we aim to add to these with the inclusion of new frameworks and hardware [56, 52, 53]. Notably, our work evaluates the impact of NVIDIA’s recently developed NVLink technology for deep learning, which is as an alternative to traditional PCIe Gen3 for CPU to GPU and GPU to GPU data transfer, and also Intel’s Knight Landings (KNL), which was very recently introduced to overcome the problem of limited memory on GPUs while having powerful and fast vector and tensor operations. This technology is unique in that it is Intel’s first Xeon Phi accelerator that
can replace a host processor, eliminating the need for PCIe transfer.

The remainder of the chapter is organized as follows. First, we describe the deep learning frameworks and hardware used in this work. Following this, we evaluate the different frameworks with various metrics on differing hardware setups. Before concluding, a brief discussion regarding challenges is given. After conclusion, we talk about future works to which we intend to extend this work.

3.2. Deep Learning Frameworks

Table 3.1. Deep learning frameworks for evaluation.

<table>
<thead>
<tr>
<th>FRAMEWORK</th>
<th>VERSION</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>v1.0</td>
<td><a href="http://caffe.berkeleyvision.org/">http://caffe.berkeleyvision.org/</a></td>
</tr>
<tr>
<td>TensorFlow</td>
<td>v0.12</td>
<td><a href="https://www.tensorflow.org/">https://www.tensorflow.org/</a></td>
</tr>
<tr>
<td>Apache SINGA</td>
<td>v0.30</td>
<td><a href="https://singa.incubator.apache.org/">https://singa.incubator.apache.org/</a></td>
</tr>
</tbody>
</table>

For this work, we selected three frameworks for evaluation: Caffe from Berkeley, TensorFlow from Google, and Apache SINGA [9, 4, 51, 50]. These were selected based on a combination of popularity, performance, and distributed computing capability. Because of the difficulty in compiling the different frameworks over various hardware and developing similar models for benchmarking, we limited ourselves to only these. For example, ensuring correct compilation and execution of each framework over the P100 and Knight’s Landing took much effort. Furthermore, producing code for identical deep learning models and training for each framework is difficult. In a future work, we plan to diversify our selection of frameworks.

3.2.1. Caffe

Caffe was developed at the Berkeley Vision and Learning Center and is one of the most popular deep learning frameworks currently available. It is made popular in part by its fast benchmark training time and ease of use for programming novices. Because of its popularity, the large community of users and developers have released many different flavors
of Caffe, each attempting to add a new feature or improve an existing one. For instance, Intel released their own version with optimization for Xeon processors\(^1\). Additionally, there have been several releases that add MPI to the framework, consequently enabling scaled-out deep learning. In this work, we make use of the base version of Caffe from Berkeley for GPU comparisons and for evaluating Intel’s Knights Landing (KNL) we use the Intel version of Caffe.

3.2.2. TensorFlow

TensorFlow was originally developed by the Google Brain Team at Google’s Machine Intelligence research organization. It has since then become an open-source project. Deep learning models can be expressed with a data flow graph, where each vertex represents some computation, similar to a neuron in a neural network. Recently, distributed training was added to TensorFlow. This framework was selected for our work because of its fast community growth and distributed training capability. Furthermore, because it is still quite new, there is a lack of validated benchmarks available that measures the performance of TensorFlow over different hardware.

3.2.3. Apache SINGA

SINGA is a lesser known deep learning platform designed with distributed training in mind. The lack of popularity is likely due to the fact that it is an Apache project that is still in incubation. However, it offers desirable features in a deep learning tool. The deep learning model definition, which is done by defining layers, is similar to that of Caffe, allowing easy migration of model configurations from one framework to another. More importantly, it enables effortless scaling out and flexible client/server configuration for synchronous or asynchronous training. The authors have published promising results demonstrating the performance of scaling out and both synchronous and asynchronous training [51, 50]. In spite of its lack of popularity, we select SINGA in order to compare the scalability with other frameworks in multiple different distributed HPC environments.

\(^1\)Available at https://github.com/intel/caffe
3.3. HPC Architectures

Table 3.2. Hardware for deep learning evaluation.

<table>
<thead>
<tr>
<th>HARDWARE</th>
<th>CORES</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel E5-2680v2 Xeon Processor</td>
<td>20</td>
<td>64 GB</td>
</tr>
<tr>
<td>IBM Power8 Processor</td>
<td>20</td>
<td>256 GB</td>
</tr>
<tr>
<td>IBM Power8+ Processor with NVLink interface</td>
<td>20</td>
<td>256 GB</td>
</tr>
<tr>
<td>NVIDIA Tesla P100 with NVLink interface</td>
<td>3584</td>
<td>16 GB</td>
</tr>
<tr>
<td>Intel Phi 7230 processor(KNL)</td>
<td>64</td>
<td>96 GB</td>
</tr>
</tbody>
</table>

3.3.1. Deep learning with CPU

More often than not, training deep learning models with CPU is not ideal. It is well known that GPUs provide an extreme advantage in cases of vector computation, which makes up the majority of computation during training. In practice, the CPU typically acts as the parameter server, performing scalar updating of parameters received from GPUs and redistributing the updated values to the GPUs. However, it is still important to evaluate CPU-based training because not all users have access to GPUs with sufficient memory for training larger data. In Table 3.2, the CPUs used in this work are described. The main difference between the Power8 and Power8+ is that the Power8+ supports NVLink, which will be described in the following subsection. We investigate the Intel’s Knight’s Landing performance on various deep learning frameworks. Intel Knight’s Landing is interesting since they it can replace the host CPU, eliminating the need for data transfer over PCIe, while having acceptable number of cores and hyper threads to accommodate the needs of vector operations in deep learning. In addition, it can has access to a larger memory pool by using RAM, enabling training with a larger batch size and model and making faster convergence possible.

3.3.2. Deep learning with GPU

Training evaluation using GPU is arguably more important than the CPU evaluation. GPUs outperform CPUs in vector computation, which makes up the majority of compu-
Figure 3.1. Scaling up with Caffe using AlexNet for GPU (P100 with and without NVLink) while increasing number of GPUs and batch size.

Of particular interest is the Tesla P100 with NVLink. NVLink is a recently developed technology by NVIDIA that intends to provide faster data sharing between CPU to GPU and GPU to GPU. In the evaluation, we will go into detail on their performance over the P100 and in order to make a clear comparison and find out the effectiveness of the the new NVlink technology. We select and compare NVIDIA Tesla P100 GPUs, one with support of NVLink and the other with PCIe Gen3 connection to CPU.

Table 3.3. Deep learning models and datasets for experiments.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>DATASET</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet [57]</td>
<td>ILSVRC’12</td>
<td>image-net.org/challenges/LSVRC/2012</td>
</tr>
<tr>
<td>VGG-19 [58]</td>
<td>ILSVRC’12</td>
<td></td>
</tr>
<tr>
<td>GoogLeNet [59]</td>
<td>ILSVRC’12</td>
<td></td>
</tr>
<tr>
<td>LeNet [60]</td>
<td>MNIST</td>
<td>yann. lecun. com/exdb/mnist</td>
</tr>
<tr>
<td>ConvNet</td>
<td>CIFAR-10</td>
<td><a href="http://www.cs.toronto.edu/">www.cs.toronto.edu/</a> kriz/cifar.html</td>
</tr>
</tbody>
</table>

3.4. Evaluation

We divide the evaluation into three parts. First, we investigate the performance of different CPU, GPU, and Knight’s Landing models on training time. Second, we analyze
### 3.4.1. CPU and GPU performance

Before considering scaling up or out, it is important to see the performance of a single compute resource. Our first experimental consideration is the affect of CPU and GPU model on the time taken to train a batch of images. In Table 3.4, we provide the training time per batch with various CPU and GPU configurations. These results were gathered using Caffe and the AlexNet model with the ImageNet Large Scale Visual Recognition
Figure 3.3. Scaling up batch size on P100 with NVLink and KNL using Alexnet with Caffe. Comptetion 2012 (ILSVRC’12) dataset. Each scenario consists of only one worker (whether it be CPU or GPU) and has a batch size of 256. As expected, the two configurations using GPU give the best performance, and the P100 GPU with NVLink vastly outperforms CPU and KNL. Later in this section, we will investigate this performance more closely in order to visualize the effect of NVLink on training time. There is an interesting trend with the CPUs; there is performance degradation with any configuration where simultaneous multithreading (SMT) is on. In other words, the number of threads exceeding the number of cores hurts performance. As it is shown and expected in 3.4, GPU and Intel’s Knights landing outperform CPU remarkably, so the paper will concentrate more on accelerator hardware than benchmarking CPU.

Previously, Table 3.4 demonstrated the added benefit of the P100 and NVLink over the PCIe. To analyze the data transfer speedup provided by NVLink more closely, we provide Table 3.5, which describes the data transfer rate of NVLink against PCIe Gen3. The DeepBench toolkit\(^2\) was used to gather these results. We used the All-Reduce technique, which relies on keeping the parameters on all instances of the model across 4 GPUs synchronized.

\(^2\)Available at https://github.com/baidu-research/DeepBench
Table 3.4. Benchmark for average time to train one batch with Caffe and AlexNet. The ILSVRC’12 dataset was used with a batch size of 256

<table>
<thead>
<tr>
<th>HARDWARE</th>
<th>AVERAGE TIME (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel E5-2680v2 Xeon</td>
<td>51.76</td>
</tr>
<tr>
<td>IBM Power8 (SMT=OFF)</td>
<td>47.66</td>
</tr>
<tr>
<td>IBM Power8 (SMT=4)</td>
<td>79.28</td>
</tr>
<tr>
<td>IBM Power8 (SMT=8)</td>
<td>106.56</td>
</tr>
<tr>
<td>IBM Power8+ (SMT=OFF)</td>
<td>37.98</td>
</tr>
<tr>
<td>IBM Power8+ (SMT=4)</td>
<td>58.83</td>
</tr>
<tr>
<td>IBM Power8+ (SMT=8)</td>
<td>60.49</td>
</tr>
<tr>
<td>NVIDIA Tesla K20X</td>
<td>1.61</td>
</tr>
<tr>
<td>NVIDIA Tesla P100 (NVLink)</td>
<td>0.15</td>
</tr>
<tr>
<td>NVIDIA Tesla P100 (PCIe)</td>
<td>0.32</td>
</tr>
<tr>
<td>Intel’s Knights Landing</td>
<td>0.88</td>
</tr>
</tbody>
</table>

by making sure all instances of the model have the same copy of the gradients before taking an optimization step. We measured the time spent to synchronized parameters for four P100 GPUs, with and without NVlink, in one node. As it is shown in Table 3.5, except for the smallest case of 100000 floats, NVlink outperforms PCIe by an approximate factor of two and the difference is much more obvious when the amount of data is increased. As a result, it is obvious that NVlink contributes to faster training and shorter time to completion very effectively.

Table 3.5. Data transfer speeds using P100 with and without NVLink.

<table>
<thead>
<tr>
<th>NUM.FLOATS</th>
<th>BYTES</th>
<th>NVLINK</th>
<th>PCIe</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>400000</td>
<td>0.238(msec)</td>
<td>0.186(msec)</td>
</tr>
<tr>
<td>3097600</td>
<td>12390400</td>
<td>1.427(msec)</td>
<td>2.861(msec)</td>
</tr>
<tr>
<td>4194304</td>
<td>16777216</td>
<td>1.914(msec)</td>
<td>3.858(msec)</td>
</tr>
<tr>
<td>6553600</td>
<td>26214400</td>
<td>3.088(msec)</td>
<td>6.029(msec)</td>
</tr>
<tr>
<td>16777217</td>
<td>67108868</td>
<td>7.519(msec)</td>
<td>16.358(msec)</td>
</tr>
</tbody>
</table>
3.4.2. Scaling up a single node

After laying the groundwork for different CPU and GPU performance, we will see how well different framework and hardware combinations scale up. In practice, many users utilize a single machine with either multiple CPU cores or GPUs as workers. Here, we will see how different frameworks, Caffe and TensorFlow in particular, scale up with respect to CPU and GPU. Starting with Figure 3.1, we have the training time for Caffe with Alexnet on P100 GPUs while scaling up the batch size. In the figure, P100(NV) and P100(PC) represent P100 with NVlink and P100 with PCIe connection respectively and the number after (x) shows number of GPU used in the benchmark. We scale up from one to 4 GPUs across one node with batch sizes from 256 to 1024 images. As it is expected, increasing the batch size increases the computation time linearly and increasing the number of GPUs decreases the training time linearly. Figure 3.1 illustrates Caffe shows very good scalability. Another highlighted feature the Figure 3.1 is the NVLink speedup. The marked gap shows the difference in training time for P100 with NVlink and P100 with PCIe for batch size of 1024 images. The mentioned difference is the speedup solely caused by having faster communication link between CPU and GPUs.

Briefly, we give the scaling performance with respect to CPU. Figure 3.2 has the training time for Caffe with Googlenet with scaled batch size from 32 to 512 images. While the CPU performance is significantly behind GPU in terms of speed, CPU still exhibits good scaling.

As mentioned before, Intel has released Intel’s Knights Landing to overcome the limited GPU memory problem which limits the batch size while having fast vector and matrix operations by providing 64-72 number of cores, depending on the model, and 4 threads per core. In addition, KNL is Intel’s first Xeon Phi accelerator that can replace a host processor eliminating the need for PCIe transfer which in turn contribute to better performance. In Figure 3.3, we compare time for one training iteration for P100 with NVlink and KNL by scaling up the batch size in Alexnet with Caffe. Figure 3.3 shows although KNL is
showing acceptable speed, it is outperformed by P100 by almost factor of 3 regarding the performance.

To compare Caffe and TensorFlow more closely, we evaluate their training times with two larger, popular networks, VGG-19 and GoogLeNet (Inception V1). Figure 3.4 gives the time per training iteration using a batch size of 128 with the GoogLeNet (Inception V1) model. Both frameworks exhibit good scaling, but Caffe performs each iteration almost two times faster than TensorFlow. In moving to a larger network, we have VGG-19 results in Figure 3.5. The frameworks’ performance is much closer in this case, however TensorFlow began to exhibit some scaling issues with four nodes.

![Figure 3.4. Scaling up batch size on P100 with NVLink using GoogLeNet with TensorFlow and Caffe.](image)

**3.4.3. Scaling out with multiple nodes**

In this section, we will provide some initial analysis of scaled-out deep learning with TensorFlow, SINGA and Intel Caffe over Intel’s Knight Landings using Omni-Path interconnect and P100 using InfiniBand. First, we observe TensorFlow, Caffe, and SINGA results for scaling out GPUs with the LeNet architecture and MNIST dataset in Figure 3.6. This shows the number of images trained per seconds as more nodes are added. Each node
Figure 3.5. Scaling up batch size on P100 with NVLink using VGG-19 Net with TensorFlow and Caffe.

has one P100 GPU worker with a mini-batch size of 64 images. The network interconnect used is 56 Gbps InfiniBand. Performance is depicted in terms of number of images trained per millisecond. At this point, distributed training is a very experimental feature in TensorFlow. Regardless, it provides good scaling as more nodes are added. While SINGA certainly benefits from scaling out, it lags behind the other two frameworks severely in terms of training throughput. The Intel version of Caffe also shows good scalability while providing the best throughput in terms of number of images trained per millisecond.

Since Intel Caffe was designed with particular effort towards utilizing Intel processors and coprocessors (i.e., Knight’s Landing), we investigate its performance with respect to scaling out over nodes with Knight’s Landing (KNL) processors. These results include TensorFlow’s performance in order to have a point of comparison. SINGA had compilation issues, so it is not included in this result. Figure 3.7 shows TensorFlow and Caffe results for scaling out with LeNet and MNIST mini-batch size of 64 images on each KNL. The number of images trained per millisecond is reported as more KNL machines are added. Each machine has one KNL. The network interconnect used is Intel Omni-Path. These
Figure 3.6. TensorFlow and SINGA scale out results. Each node has one P100 GPU worker and each worker has a mini-batch size of 64.

results show the benefit of leveraging a particular hardware technology when developing a deep learning tool. Caffe outperforms TensorFlow in terms of image throughput as well as scalability. In fact, TensorFlow exhibits poor scalability, likely due to it not being developed with KNL in mind.

It is evident, based on Figures 3.6 and 3.7, that GPU outperforms KNL. We can more directly observe this in Figures 3.8 and 3.9, which provide comparisons of scaling out Caffe and TensorFlow, respectively, using KNL with Omni-Path and P100 with InfiniBand. Again, the LeNet model and MNIST dataset are used with a mini-batch size of 64 images. With regards to both Caffe and TensorFlow, the P100 provides drastically more throughput than the KNL.

3.4.4. Discussion

Efficiently scaling up single-node training can become challenging, particularly when using GPUs. GPU training introduces an additional overhead in transferring data to and from the CPU or other GPUs. This bottleneck limits the scalability of multiple-GPU machines. In this work, we used a newly developed technology, NVLink, to speed up
Figure 3.7. Caffe and TensorFlow scale out results. Each node is one KNL worker and each worker has a mini-batch size of 64.

data transfer. Figure 3.1 demonstrated the effect of having NVlink by comparing the time between P100 GPU with and without NVLink with the same batch size.

Scalable distributed training is arguably more challenging than scaling up in a single node because it is subject to the same CPU and GPU data transfers but with an additional, more severe bottleneck in the network. However, it is still important to push towards efficient distributed training in order to mitigate larger models and the constant increase in available data for training. In order to make this advancement, deep learning software should make clever use of model and data parallelism, have a strong communication architecture in place, and leverage state-of-the-art technologies.

Certain hardware technologies are already being well leveraged by deep learning tools, as shown in this paper. However, as exhibited by Intel Caffe in Figures 3.7 and 3.8, it shows tremendous speedup over TensorFlow when using the Knight’s Landing but remains slower than when using the P100. In other words, Intel Caffe, developed with utilizing Intel Xeon processors and coprocessors, is still faster with P100 over the KNL. This is likely due to the early state of development of deep learning over KNL, especially when CUDA and
Figure 3.8. Scale out result with Intel Caffe over P100 via InfiniBand and KNL via OmniPath. For P100 results each node has one P100 GPU worker and for KNL result each node is one KNL worker and each worker has a mini-batch size of 64.

CUDNN, libraries utilized by Caffe for deep learning over GPU, are much more established.

3.5. Conclusion

In this work, we evaluated three different deep learning tools, Caffe, TensorFlow, and SINGA, over a variety of hardware setups. Analysis was provided in terms of speed and scaling, promoting a better understanding of these tools and their performance in different scenarios. Of particular interest was the performance analysis using NVIDIA’s NVLink technology over PCIe and using Intel’s Knights Landing hosts with Intel Omni-Path interconnect.

As a result of these experiments, we have the following observations:

- Apache SINGA may exhibit good scaling, but it leaves a lot to be desired in terms of training time compared to Caffe and TensorFlow.
- We have seen that P100 GPUs with NVLink consistently provide the best performance, in terms of the training time and communication speed, and significantly
Figure 3.9. Scale out result with TensorFlow over P100 via InfiniBand and KNL via Omni-Path. For P100 results each node has one P100 GPU worker and for KNL result each node is one KNL worker and each worker has a mini-batch size of 64.

outperform Intel’s Knight Landing.

- We have investigated the effect of Omni-Path versus InfiniBand as interconnect and showed that with even faster communication link, KNL is still behind the P100 GPUs in terms of performance and scalability.

- The experimental results obtained from the analysis of the different frameworks are as follows:

  1. Computation time: We have observed that Caffe is faster than other examined frameworks and it is outperforming TensorFlow. Larger networks, especially VGG-19, show the two frameworks much closer in timing, indicating that TensorFlow’s performance suffers with smaller networks like LeNet. Both Caffe and TensorFlow outperform SINGA by a large margin.

  2. Scalability results: All three frameworks, Caffe, TensorFlow, and SINGA, scale-up with multiple GPUs on one node and scale-out with multiple nodes linearly,
while again Caffe outperforms TensorFlow, which confirms our previous observation. However, when utilizing KNL and Omni-Path, Tensorflow does not scale well. This is not unexpected since it was not designed to run over KNL or other similar processors and coprocessors.

To our best knowledge, this is the first work evaluating NVLink and Intel’s Knights Landing for deep learning benchmarks. In future works, we plan to add more frameworks for benchmarking and expand our hardware environment with an aim towards scaling out. As deep learning continues to thrive, it will continue to be necessary to evaluate the tools developed over state-of-the-art hardware.

3.5.1. Future work

One aspect in which we plan to extend this work involves adding more novel hardware that may be of interest to evaluate includes other Xeon Phi co-processors (e.g., Knight’s Corner) and FPGAs. On top of adding additional hardware technologies, we plan to include additional popular deep learning models and frameworks. Specifically, we aim to evaluate more frameworks capable of distributed training. With this, further experiments and more detailed analysis towards scaling out deep learning will be done.
Chapter 4. Predicting ADHD using 3D Convolutional Neural Networks and fMRI Data

4.1. Introduction

Deep learning continues to increase in popularity as we see many disciplines adopting it for various purposes. In particular, the field of medical imaging makes use of deep learning for predictive diagnosis and treatment of patients. However, these images can be complex and cause difficulty in successfully applying deep learning. For instance, fMRI data is multidimensional, consisting of many two-dimensional image slices along the depth of the brain. Furthermore, this data is typically captured over a period of time, resulting in a time series of three-dimensional fMRI data.

Convolutional neural networks (CNNs) have shown great success when applied to image classification tasks in various domains [59, 57]. However, most applications of CNNs employ a 2D convolution. This technique considers the spatial relations along the width and height of the image, but it ignores the spatial relations along the depth dimension in 3D images. It is important to note that, in this context, depth is not referring the different channels of the input (e.g., RGB channel) but rather the pixels along the depth of the 3D volume. Ignoring this spatial relation causes the deep learning model to overlook crucial information during the learning process.

In order to overcome the multidimensional complexity of fMRI data, we developed a 3D convolution in Apache SINGA [51, 50]. SINGA is a distributed deep learning platform for large-scale networks. Because of the computational complexity and size of the 3D CNNs, it is necessary to distribute the workload over a cluster for acceptable training time. We apply a 3D CNN to the ADHD-200 dataset [61, 62] and show improved results over conventional 2D CNN for predicting diagnosis of ADHD in patients.

The rest of the chapter is organized as follows. We summarize our related work in Section 4.2. In Section 4.3, we discuss the methodology behind our work. This will go into greater detail on 2D and 3D convolutions, SINGA, the dataset preparation, and our
CNN model. We evaluate the performance by comparing our 2D and 3D CNN models in Section 4.4 and conclude the chapter in Section 4.5.

4.2. Related Work

To date, there have been several works regarding both 2D and 3D CNNs applied towards medical imaging data and other domains. In a recent work, Saman Sarraf et al. used fMRI data and 2D CNN to classify between brains with Alzheimer’s disease and normal brains [60, 63]. Similar to the our model architecture in our work, they used two CNN layers, each followed by one pooling layer, and ending with two fully connected layers. The main difference with our model is that our model employs 3D convolution. They show the strength of their model by showing an average accuracy of 96.86% for five different training runs. Another deep learning application for Alzheimer’s prediction by Adrein Payan et al. uses a 3D CNN to predict the Alzheimer’s status of a patient with respect to an MRI scan of the brain [64]. In this experiment, the ADNI data set consisting of 2265 scans was used\(^1\). Their network consisted of three parallel CNNs, which were pretrained with a sparse auto encoder, followed by pooling and fully connected layers. Their model resulted in an accuracy of 89.47%. This highlights the success of using 3D CNN for MRI data. However, in our work, we use fMRI data for predicting ADHD instead of Alzheimer’s disease. Further use of 3D convolution is shown by Shuiwang Ji et al [65]. They proposed a 3D CNN for recognizing human action. Different from our 3D convolution, their third dimension is with respect to time, whereas ours is with respect to depth in a 3D volume. Their network consists of three convolution layers, two subsampling layers, and one fully connected layer, and they achieved an accuracy of 90.2% in classifying actions into six different categories, including jogging, walking, running, hand waving, hand clapping, and boxing. In another work, Daniel Maturana and Sebastian Scherer proposed the Voxnet architecture, which uses 3D CNNs for real-time object recognition. With their architecture, they reached accuracies of 92% and 83% on ModelNet40 and ModelNet10 respectively [66]. Similar to our work,

\(^1\)Data available at http://http://adni.loni.usc.edu/.
they consider volumetric image datasets but for a different task. 3D CNNs have been used by Du Tran et al. for learning spatiotemporal features from videos [67]. They have reported the achieved accuracy among different data sets including, but not limited to, action recognition with Sport1M (85.2%) and scene classification with YUPENN (98.1%). Again, action recognition in this work considers time instead of volume depth as the third dimension in convolution. Bhaskar Sen et al. implemented Hidden Markov Models on the ADHD-200 competition dataset to differentiate between health control, ADHD inattentive, and ADHD combined types with an accuracy of 63.01% and 62.06% respectively [68, 61]. This work has the same goal as our work but uses a different model. Further work with the ADHD-200 dataset uses Bayesian Networks and Deep Belief Networks [69, 70]. Adhish Prasoon et al. applied 2D CNNs for cartilage segmentation on MRI scans and achieved 99.93% accuracy [71]. This shows the applicability of CNN to MRI data with a different goal in mind. Heung-Il Suk et al. have applied stacked auto encoders towards predicting Alzheimer’s diagnosis from MRI and PET datasets [72]. They achieved mean accuracies of 92.38% (MRI), 92.20% (PET), and 95.35% (MRI + PET), and showed good discrimination between different diagnoses. It is another example of Alzheimer’s diagnosis with medical imaging data, but this work, unlike our work and previously mentioned works, uses a different deep learning model. All of the mentioned works either employ 3D CNN or attempt to train deep learning models with medical imaging data. Uniquely, our work uses 3D convolutions for diagnosis of ADHD from fMRI data.

4.3. Methodology

4.3.1. Convolutional Neural Networks

As discussed in the previous section, CNNs have seen success through many domains that include analysis and classification of image datasets. The convolutional layers’ connectivity is modeled after the functionality of the visual cortex. This concept promotes a local connectivity pattern between neurons, allowing learning based on local spatial correlations. The convolutional layer has become a standard in image classification as is evident by its
use in many popular network models, such as LeNet, GoogleNet, and AlexNet [59, 60, 57]. Even more so, it has shown applicability to higher dimensional data, such as volumetric images or videos [64, 65, 66].

4.3.1.1. 2D vs. 3D Convolution

A visual comparison between 2D and 3D convolution is given in Figure 4.1. Both types of convolution function similarly, with the important difference being the dimension of the data and receptive filters. 2D convolution is the standard technique used in CNNs because most data processed by CNNs is only 2-dimensional. Even when the dimensionality is higher, 2D convolution can still be applied to the data with some preprocessing of the data. For example, an fMRI volume can be split into 2D slices that are each trained individually. Alternatively, each 2D slice can be concatenated to form a larger 2D image representing...
a matrix of the smaller images. However, these techniques fall short when considering the spatial relation between the 2D slices along the depth of the volume. Instead, we show that it is beneficial to utilize 3D convolutions for volumetric imaging data.

Implementing 3D convolution introduces higher computation and memory overheads, which may be another reason for its lower popularity. In referring to Table 4.2, it is evident that 3D convolution imposes a much higher overhead in terms of computation and memory. However, it allows the network to learn features that are more representative of the original 3D data. So, while the network will take more time and memory to process a single image, the classification accuracy is likely to improve.

### Algorithm 1 3D Convolution

**Input:** `srclayer`  
**Output:** `layer`

**Initialization:**  
`srcdata ← srclayer.data`

1. **for** $n = 0$ to `batchsize` **do**  
2.  
3.  
4.  
5. **end for**

4.3.2. Implementation in SINGA

SINGA is a recently developed distributed deep learning platform designed for large-scale neural network training [51, 50]. It aims to provide scalability by supporting both data and model parallelism. This allows the user to partition both the dataset and the neural network among workers. Furthermore, it enables flexible server and worker partitioning, allowing the user to create a variety of synchronous and asynchronous training frameworks such as AllReduce, Sandblaster, and Downpour [73]. SINGA supports both Mesos [74] and the Hadoop Distributed File System (HDFS)[75] for distributed processing and storage and further scalability. SINGA’s programming model provides an abstraction allowing users to define their deep learning models in terms of layers. Currently, SINGA supports many of the well known deep learning models, such as convolutional or recurrent neural networks. Both CPU and GPU training are available. GPU training accelerated by the
Algorithm 2 vol2col

Input: srcdata

Output: coldata

Initialization:
1: for $c = 0$ to $\text{columns}_c$ do
2: \hspace{1em} $\text{offset}_d \leftarrow (\frac{c}{\text{filter}_w})/\text{filter}_h \% \text{filter}_d$
3: \hspace{1em} $\text{offset}_h \leftarrow (\frac{c}{\text{filter}_w}) \% \text{filter}_h$
4: \hspace{1em} $\text{offset}_w \leftarrow c/\text{filter}_w$
5: \hspace{1em} $\text{offset}_c \leftarrow c/\text{filter}_d/\text{filter}_h/\text{filter}_w$
6: for $d = 0$ to $\text{columns}_d$ do
7: \hspace{1em} for $h = 0$ to $\text{columns}_h$ do
8: \hspace{2em} for $w = 0$ to $\text{columns}_w$ do
9: \hspace{3em} $n \leftarrow ((c \times \text{columns}_d + d) \times \text{columns}_h + h) \times \text{columns}_w + w$
10: \hspace{3em} if padding($d, h, w$) then
11: \hspace{4em} coldata[n] \leftarrow 0
12: \hspace{3em} else
13: \hspace{4em} $m \leftarrow ((\text{offset}_c \times \text{depth} + \text{pad}_d) \times \text{height} + \text{pad}_h) \times \text{width} + \text{pad}_w$
14: \hspace{4em} coldata[n] \leftarrow \text{srcdata}[m]
15: \hspace{3em} end if
16: \hspace{2em} end for
17: \hspace{1em} end for
18: end for
19: end for
20: return coldata
cuDNN library [76]. As part of our work, we implemented 3D convolution and pooling into the SINGA framework.

Our contribution to SINGA for this work is the implementation of several layers and a parameter initialization method. Both 3D convolution and pooling layers were added. The existing 2D layers were used as a base and modified accordingly. The algorithms involved in 3D convolution are presented in Algorithms 1 and 2. Algorithm 1 describes the process of computing convolution for each individual image in the training batch. In Algorithm 2, we illustrate the procedure, \texttt{vol2col}, called in Algorithm 2. It is converting volume data into a column structure so that the convolution algorithm can apply the dot product with the weights and add the biases. In addition to the convolution and pooling, the image preprocessing layer was extended to work with image volumes. A 3D local response normalization (LRN) layer was also added, however we did not utilize it in the CNNs described in this paper as it did not improve the results. Finally, we implemented the Xavier initialization method [77] in order to improve the initial learning. These newly developed layers and features were used together with existing layers and features to create the CNNs in this work.

4.3.3. ADHD Dataset

For this work, we utilize the ADHD-200 competition sample dataset [61, 78]. In particular, we use the NIAK preprocessed dataset available through the NITRC NeuroBureau website\textsuperscript{2}. The reason for using this preprocessed data is that the fMRIs are time- and motion-corrected. This process removes noise in the data, resulting in our model performing more efficient learning. The training dataset consists of resting-state fMRI and anatomical datasets from 776 children and adolescent patients (ages 7-21 years old) that were gathered across 8 independent imaging sites. Of these 776 patients, 491 were typically developing individuals (i.e., no ADHD) and the remainder with subtypes of ADHD (i.e., ADHD-inattentive or ADHD-combined). The testing dataset provided by the ADHD-200

\textsuperscript{2}Data available at http://fcon_1000.projects.nitrc.org/indi/adhd200.
competition adds 195 additional patients’ fMRI datasets. Although there was accompanying phenotypic information for each patient, we did not utilize it for this work.

![Example fMRI data from ADHD-200 dataset.](image)

**Figure 4.2.** Example fMRI data from ADHD-200 dataset.

### 4.3.3.1. Data Preprocessing

Because the data was already preprocessed by the NeuroImaging Analysis Kit (NIAK) pipeline [62], further preprocessing was minimal. The fMRI volumes were extracted from the Nifti files and resized to 48x48x25. This process removed some 2D slices that contained little to no information and, as a result, would hinder the training process. Additionally, this ensured that data aggregated from all sites and patients were of a uniform dimension. Labels were attached to each fMRI corresponding to the ADHD diagnosis of the patient from which the fMRI was taken. The original dataset had very few (less than 2%) occurrences of a particular diagnosis (ADHD-hyperactive), and this fourth diagnosis was not included in the ADHD-200 competition. Because of this, it was removed from the training set as to not hinder the learning process. Additionally, the training set had many more occurrences of typically developing children than children with ADHD (491 vs. 285). For this reason, we balanced the training set to more evenly represent each diagnosis. This resulted in an approximately even number of each of the three diagnoses labels (i.e., 0 for no ADHD, 1 for ADHD-inattentive, and 2 for ADHD-combined) in each training batch.
Doing this avoids biased learning towards one diagnosis prediction. As a final preprocessing step, the image pixel intensities were scaled down by a factor of 0.003 to a value in the range of $[0, 1]$ because this showed better training results.

4.3.4. Training Model

4.3.4.1. Architecture

Two different models were trained for our comparisons. Both models are based on the well-known LeNet architecture\[60\]. One was trained with 2D convolution and pooling layers while the other with 3D convolution and pooling layers. Other than the convolution and pooling layer configurations, the model initialization parameters are identical. This allows a fair comparison between 2D and 3D convolutions. The overall layer setup of the model can be viewed in Figure 4.3. Not pictured in the figure are three activation layers following the two convolution and first fully connected (FC) layer. After including these activation layers, the model consists of 13 layers.

4.3.4.2. Hyperparameter Configurations

For training the model, classic stochastic gradient descent was used with a momentum of 0.9. After optimization, We found that the ideal initial learning rate was 0.0001, which decreased every 8 epochs by a factor of 10. The batch size used was 512. Each convolution layer used a stride of size one and applied padding to the input in order to maintain the overall structure of the data. For instance, given a stride of one and a filter size $f$ along any dimension, the amount of padding $p$ on that dimension is calculated as $p = (f - 1)/2$. A dropout layer was configured with 40% dropout in order to avoid the model overfitting the training data. For all layers with weights, the Xavier initialization method was used \[77\].

4.4. Experiment and Results

Our experiments were performed using the LONI QB2 cluster\(^3\). Each node has two 10-core 2.8 GHz E5-2680v2 Xeon processors, 64GB memory, 500GB HDD, and two NVIDIA

\(^3\)http://loni.org/
Tesla K20x GPUs. The cluster network uses 56Gb/s InfiniBand with 2:1 oversubscribed mesh topology. The results are given in terms of classification performance and scalability performance.

4.4.1. ADHD Classification

We compare the training loss and accuracy of 2D and 3D convolution in Figures 4.4 and 4.5. The 2D convolution loss (blue) reaches convergence much earlier than the 3D convolution (red). Furthermore, its converged loss is significantly higher than that of the 3D convolution. The accuracy observes a similar pattern with the 3D convolution converging at a much higher accuracy. These results give a good insight towards how successful each model is at learning. Although each model exhibits a similar slowdown around the training step 1000 in Figures 4.4 and 4.5, the 3D model has clearly shown more success in learning by having a lower loss and higher accuracy.

In Table 4.1, we give the testing results using the holdout dataset from the ADHD-200 competition. The fMRI accuracy depicts how well the model classifies each individual fMRI into an ADHD diagnosis. Patient accuracy is the accuracy of the predicted diagnosis of ADHD to a patient. This is calculated by taking the average of the predictions of each patient’s individual fMRI predictions. Specificity describes the model’s performance in correctly diagnosing a patient as typically developing, and sensitivity describes the performance in correctly diagnosing a patient as ADHD positive. ADHD subtype diagnosis shows how well the model is able to discern between ADHD subtypes in patients who were correctly predicted to be ADHD positive (i.e., patients who were not predicted to be normally developing). The final metric is the ADHD-200 score, which was calculated based on the ADHD-200 competition. This score is determined as follows; correct subtype diagnosis (i.e., typically developing, ADHD primarily inattentive type, or ADHD combined type) awards one point. Diagnosis of ADHD but with an incorrect subtype diagnosis with awards half of a point To clarify, a predicted diagnosis of ADHD-combined (alternatively, -inattentive) with an actual diagnosis of ADHD-inattentive (alternatively, -combined) re-
Table 4.1. Testing set classification results for ADHD-200 dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>fMRI Acc.</th>
<th>Patient Acc.</th>
<th>Specificity</th>
<th>Sensitivity</th>
<th>ADHD Subtype Diag.</th>
<th>ADHD 200 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chance</td>
<td>33%</td>
<td>33%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>38.9%</td>
</tr>
<tr>
<td>3D</td>
<td>63.7%</td>
<td>60.1%</td>
<td>86.2%</td>
<td>60.0%</td>
<td>77.3%</td>
<td>64.6%</td>
</tr>
<tr>
<td>2D</td>
<td>56.7%</td>
<td>57.7%</td>
<td>67.4%</td>
<td>59.0%</td>
<td>76.7%</td>
<td>60.7%</td>
</tr>
</tbody>
</table>

wards half of a point, whereas a predicted diagnosis of typically developing with an actual diagnosis of ADHD-inattentive (or -combined) awards no points.

We give a baseline, referred to as chance in Table 4.1. This classification assumes an equal prediction chance for each of the three diagnoses, resulting in a 33% patient prediction accuracy and 38.9% ADHD-200 score. The 2D CNN improves on these results with 56.7% patient accuracy and 60.7% ADHD-200 score. Improving even further, the 3D CNN results are 63.7% and 64.6%. Both the 2D and 3D CNNs exhibit an interesting ability to nearly match the performance of patient-based diagnosis with a single fMRI. In fact, the 3D results show that it manages to more accurately classify an individual fMRI over a patient. This tells us that the model may have overfit to some patients’ fMRIs during the learning process. Furthermore, the 3D CNN shows excellent specificity and sensitivity, exceeding the 2D CNN specificity results by a significant margin. Lastly, both the 2D and 3D CNNs showed similar results with respect to ADHD subtype diagnosis, with the 3D outperforming the 2D by a slight margin.

4.4.2. Scalability

In Figure 4.6, we give the scalability of our 3D CNN with SINGA. The number of workers and servers are increased from 16 to 128 and 1 to 8, respectively. Average training time per iteration (mini-batch) was calculated over 200 iterations. The batch size used was 512. Synchronous training was used, meaning that all workers must synchronize with a server between each training step, and servers must synchronize between each other between each training step. Asynchronous training exhibits less of a memory overhead.
Table 4.2. The number of parameters and memory requirements for convolution, pooling, and fully connected (FC) layers in the 2D and 3D CNN models. 3D requires approximately 5 times more parameters for each layer and approximately 16 times more memory per training image.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Param. 2D</th>
<th>Param. 3D</th>
<th>Mem. 2D</th>
<th>Mem. 3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>0</td>
<td>0</td>
<td>9.216kB</td>
<td>230.4kB</td>
</tr>
<tr>
<td>Conv</td>
<td>520</td>
<td>2520</td>
<td>184.3kB</td>
<td>4.239MB</td>
</tr>
<tr>
<td>Pool</td>
<td>0</td>
<td>0</td>
<td>46.08kB</td>
<td>506.9kB</td>
</tr>
<tr>
<td>Conv</td>
<td>9050</td>
<td>27050</td>
<td>115.2kB</td>
<td>1.267MB</td>
</tr>
<tr>
<td>Pool</td>
<td>0</td>
<td>0</td>
<td>28.8kB</td>
<td>144kB</td>
</tr>
<tr>
<td>FC</td>
<td>3600500</td>
<td>18000500</td>
<td>2kB</td>
<td>2kB</td>
</tr>
<tr>
<td>FC</td>
<td>1503</td>
<td>1503</td>
<td>0.012kB</td>
<td>0.012kB</td>
</tr>
<tr>
<td>Total</td>
<td>3611583</td>
<td>18031573</td>
<td>385.6kB</td>
<td>6.39MB</td>
</tr>
</tbody>
</table>

and, therefore, can scale better. However, our synchronous training experiment shows good scalability. Referring to Table 4.2, we can see the huge increase in number of parameters and memory size per training image going from 2D to 3D models. This memory requirement is the amount of memory needed to store each training image during one training batch. It is worth noting that the actual memory requirements of the model would be much higher because many layers are left out of this table, and backpropagation approximately doubles the memory requirement per image. More importantly, this table gives insight to just how much more computationally and memory intensive 3D CNN is over 2D CNN. The 3D CNN imposes approximately 5 times more parameters and 16 times more memory per image. Respectively, this will cause a significant increase in computation and memory requirements during training. The scalability shown by SINGA and our model enables training such resource-heavy models in a reasonable amount of time.

4.5. Conclusion

Deep learning has made great strides in recent years. However, without proper techniques for handling the extreme complexity and size of data, further research in medical imaging and similar fields will be hindered. Utilizing 3D convolutions allows CNNs to more accurately extract features from 3D data by considering the spatial correlation along the depth of the data. SINGA enables large-scale training across clusters for scalable training.
of neural network models. With 3D CNN implemented in SINGA, we showed a significant improvement over 2D CNN for diagnosing ADHD based on fMRI data. In a future work, we plan to extend these techniques to more large-scale networks to improve the accuracy, as well as the scalability.
Figure 4.3. 3D CNN layer architecture
Figure 4.4. Training loss of 2D and 3D convolutional neural networks.

Figure 4.5. Training accuracy of 2D and 3D convolutional neural networks.
Figure 4.6. Scalability of 3D CNN. There are 16 workers per node.
5.1. Introduction

Accurate and real-time video stream analysis is important with respect to several real-world problems. In terms of criminal activity, real-time analysis on video streams becomes extremely important in the case of detecting and tracking and recovering stolen or suspect cars. There was an estimated 773,139 vehicle thefts in 2017, costing nearly 6 billion dollars nationwide [79]. For public transportation, rerouting and optimizing buses or emergency vehicles based on the dynamic flow of traffic can help improve or even save lives. A study showed that delayed ambulances caused an average of 700 deaths every year in Ireland [80]. In order to help combat problems such as these, it is important to have both fast and accurate vehicle detection and classification.

Real-time deep learning inference on video streams is a difficult problem when the data is complex and consists of many classes. In the case of fine-grained classification, the complexity of the data comes from the similarity of the features defining the different classes. The fine-grained features are what will ultimately give distinction between the classes. Consequently, fine-grained vehicle classification, which considers different vehicle makes and models, remains a difficult problem. Additionally, the large number of possible makes and models further complicates the vehicle classification task. This classification problem requires a robust deep learning model that is capable of distinguishing between a large number of visually similar classes. While YOLO [81] and other popular object detection models are capable of multi-class classification, distinguishing fine-grained features between cars is a challenge. Thus, the goal is to have the YOLO-based component focus on distinguishing cars from other objects and a second component focusing on learning fine-grained features that differentiate similar looking car makes and models.

In this work, we introduce RT-VDC, a framework for real-time vehicle detection and classification over a distributed infrastructure. Our unique contributions include the gath-
ering and preprocessing of car images from the web and video streams, a new deep learning tool consisting of two models performing two tasks (vehicle detection and make/model classification), and a distributed framework for large-scale training and analysis of traffic video streams in real time, suitable for a campus or similar environment. We make use of the YOLO architecture for the detection of cars and well-known Convolutional Neural Network architectures for the classification task. Since a separate component performs the make and model classification, this allows for faster retraining and fine-tuning since it is not considering car detection. RT-VDC is able to maintain competitive results with state-of-the-art classification tools while extending classification task to a large-scale environment with real-time results.

Our contributions can be briefly summarized as follows:

1. Creation of a new dataset by gathering and preprocessing of car images from the web and video streams.

2. A new deep learning tool consisting of two components performing two tasks, real-time car detection and fine-grained car classification.

3. A distributed framework for large-scale training and analysis of traffic video streams in real time, suitable for a campus or similar environment.

4. Evaluation and comparison of RT-VDC with different methods.

The remainder of this chapter is organized as follows. We first give some background on topics related to RT-VDC. After this, related works will be discussed. Then, we go into detail on the architecture and components of RT-VDC. Our techniques for gathering and processing data will follow, and we then provide some results before concluding the paper.
5.2. Background

5.2.1. Object Detection

Object detection is the task of finding different instances and, often predetermined, types of objects within an image. It has remained an important part of image and video analysis in many domains, such as social sciences, bioinformatics, and engineering. For instance, object detection has been used to detect lesions in mammogram images [82]. Over the years, deep learning has helped improve upon the performance of object detection from a standpoint of accuracy and speed. This is due largely in part to standardized datasets and competitions such as the Microsoft COCO dataset [83]. Real-time object detection is a challenge with respect to being able to maintain acceptable results while providing real-time analysis. In many cases, both detection and classification are more optimally performed as a single task [81, 84]. YOLO, for instance, has managed to boast an impressive processing rate of 244 frames per second for the smaller version of its architecture. Some of the remaining challenges for these types of works is when the number of classes is high and the task is fine-grained classification. Later in this chapter, we will show that YOLO suffers in terms of accuracy when the number of classes for the same dataset is increased and that the classification component of RT-VDC improves upon this problem.

5.2.2. Fine-grained Classification

Fine-grained image classification requires analyzing objects in images that are visually very similar, such as distinguishing between different species of birds [85]. This extends to the problem of car make and model classification. A challenge of fine-grained classification, as is present within this work, is both the intra- and inter-class variation on a fine-grained level. For instance, different models, and even makes, of cars tend to look alike except for certain fine-grained features. This is an example of low inter-class variation. The two cars are of different make and model but exhibit a very similar overall structure and look. Where they differ comes down to smaller, more important features, such as the emblem, grill, and
headlights. Beyond this, cars with the same make can look extremely different, such as a truck and sedan from the same make. This is an example of high inter-class variation. Overall, we are dealing with many cases of high intra-class variation and low inter-class variation when considering car make and model classification. This sort of classification task requires a more robust model in order to recognize and distinguish these fine-grained features from the objects.

5.2.3. Convolutional Neural Networks

Convolutional neural networks (CNN), generally, receive an image as input of size \((h, w, c)\), where \(h\) is the height, \(w\) is the width, and \(c\) describes the number of channels in the image (e.g., RBG). For example, AlexNet [57] generally assumes an input of size 224x224x3. Since CNNs are designed with the assumption that the input is an image, they are correspondingly structurally designed to cater towards this. The most notable part of the CNN is, as expected, the convolution layers. These layers uniquely consist of filters that independently perform convolutions across the entire image. In doing so, they are capable of learning and detecting patterns anywhere in the image. Typically, each convolution layer has many filters that each learn different patterns. Additionally, these layers implement weight sharing which has two benefits. First, it reduces the computational complexity and memory overhead of the model. Second, it allows the filters to detect patterns at different positions across the image. In short, each filter is capable of learning a specific pattern that it can detect across the entire image. Between each convolution layer, there are sometimes normalization layers, however they have no seen as much usage in state-of-the-art models. Pooling layers have also seen less usage as their original purpose was to reduce the computation and memory overhead, which is not as much of a requirement now with the ever increasing power of computing resources. Finally, the end of CNNs often has a fully connected layer. It is worth noting that fully CNNs exist that opt to not make use of many of these intermediate layers.
5.3. Related Works

There have been many works dedicated to deep learning based object detection or image classification, especially in recent years. In fact, several works have managed to perform detection and classification as a single task [81, 84]. One of the basis for performance for many of these detection platforms is frames per second (FPS). YOLO boasts one of the better performances, with the second version reaching above 40 FPS and their tiny-YOLO model reaching 244 FPS. One of the downsides of the models with higher FPS is the sacrifice in detection and classification accuracy (e.g., tiny-YOLO does not maintain the accuracy of regular YOLO). Additionally, as we will later show, models like YOLO struggle when the number of classes increases in a fine-grained classification problem. In this work, we want to leverage the fast detection of YOLO while adding a second component that can perform fine-grained classification on the detected vehicles.

Some authors have made an effort to compile datasets for car detection and classification. One is the Stanford Cars dataset [86]. It consists of 16,185 images of 196 classes (make and model) of cars. They demonstrated average recognition accuracy of up to 67.8% on this dataset. Since then, it has been improved on by other works we will discuss. Another dataset is the Comprehensive Cars (CompCars) dataset, which consists of 136,726 images of 1716 classes (make and model) of cars [87]. However, it is important to note that they do not use all 1,716 classes for their three classification tasks. In fact, the most they use is 431 since training a model for a fine-grained classification task on a huge number of classes is unfeasible.

Fine-grained classification has begun to receive more attention in recent years with the availability of more datasets consisting of classes distinguished by fine-grained features. In an earlier work, Xiao et al. create a pipeline that integrates three different types of attention in order to propose candidate patches, select relevant patches, and localize discriminate parts [85]. Another earlier fine-grained classification work by Xie et al. implemented multi-task learning by using both a fine-grained and hyper-class recognition model to perform
In moving towards fine-grained vehicle classification tasks, there are several works that focus on this. In the CompCars paper, the authors perform three tasks on three different datasets. First, they perform whole vehicle classification on a subset containing 431 car models (classes). Then, they do attribute predictions using a model that is trained on the first subset but tested on a second subset of 111 car models. The last task is car verification on a subset consisting of 1,145 models. It is interesting to note that the authors had better results training on testing on all views of cars instead of single views. For this reason, we do not make use of training independent models based on car view.

An early work by Sochor et al. aims to extract additional information from video streams to boost their CNN classification accuracy [89]. The additional information includes 3D bounding boxes, rasterized low-resolution shape of the vehicle, and 3D vehicle orientation. With this additional information, they get an accuracy of up to 87.8% on the CompCars dataset. A work by Biglari et al. uses latent SVM for automatically detecting discriminative parts of each vehicle category while simultaneously learning a part-based model for each category [90]. They achieve up to 97.43% accuracy on the CompCars dataset, but they use an unconventional 281 class subset rather than the standard 431 car models defined in the original CompCars paper. In a work by Dehgan et al., they use Convolutional Neural Networks to classify make, model, color [91]. They report accuracy as high as 95.88% on the Stanford dataset and 91.2% and 81.9% for CompsCars dataset with GoogLeNet and AlexNet, respectively. A further use of CNN by Liu et al. implemented a hierarchical joint CNN-based model with two components [92]. The first is a region proposal network that generates regions of interest within a car that may have fine-grained features (e.g., headlight, logo, etc.). The second component learns the feature representations from these regions for fine-grained classification of the car. While RT-VDC is similar to this, their work uses the object detection to find regions of interest within a single car image, while RT-VDC uses object detection to find a separate cars from other
objects within a larger image. Their work achieves up to 92.1% accuracy on the Stanford dataset and 95% on the CompCars dataset. Similar to the process used by RT-VDC, Yu et al. implements a two-step process of vehicle detection and classification [93]. They utilize Faster R-CNN for their detection in contrast to our use of YOLO. They similarly use a CNN model for the vehicle classification portion but also add a joint Bayesian network to the process. This process achieves an accuracy of 85% vehicle detection at the speed of 5 images per second. The classification task has an 89% accuracy on 208 unknown classes.

There are several similar works that focus on a smaller number of classes, making them less of fine-grained classification tasks. One framework, DAVE, follows a similar strategy as we do [94]. They use two separate CNN models, a fast vehicle proposal (detection) and an attribute learning network (classification of pose, color, and type). However, this work is evaluated on a much smaller number of classes (e.g., 6 and 12 vehicle types) with the Pascal VOC 2007 dataset [95]. In combining their detection and classification tasks, they achieve a speed of 4 frames per second. In another work, Taek et al. propose to use an ensemble of deep global networks and a mixture of local expert networks to learn the features of the different vehicle categories using popular CNN models (AlexNet, GoogLeNet, and Resnet) [96]. Another work makes use of reinforcement learning through visual attention-based image processing and CNNs to similarly classify vehicle type [97]. Taking advantage of multi-view images, Kim et al. using deep learning methodology to classify vehicle types with the help of multi-view surveillance camera footage [98].

Some works focus specifically on the task of vehicle reidentification, which is being able to confirm if one image of a vehicle, possibly from a different time or angle, is the same vehicle from another image. One work made efforts to create a large-scale benchmark dataset for this task based on real-world surveillance videos [99]. They also use this dataset to evaluate six existing methods for vehicle reidentification. Another work attempts to also develop a large-scale image database for vehicle reidentification [100]. However, they also develop a method for measuring the similarity of vehicles using a two-branch deep CNN
that projects images into a Euclidean space.

5.4. RT-VDC

In this section, we will discuss the architecture of RT-VDC from three perspectives: the overall framework, the deep learning components responsible for vehicle detection and classification, and the distributed infrastructure that supports RT-VDC.

5.4.1. Framework Overview

In Figure 5.1, we give an overview of RT-VDC. The process begins with a video stream from which individual frames are extracted. A parameter $n$ is defined such that every $n$ number of frames are selected for car detection. This parameter can be tweaked in order to improve performance. Unless stated otherwise, we leave this value as one for evaluation purposes. Each frame is then fed into the vehicle detection component. The result of this is a set of bounding boxes, each corresponding to a detected car. The bounding boxes are then used to extract the vehicles which are then processed by a second model responsible for vehicle classification. More details regarding these models are given in Section 5.4.2. The classifications and bounding boxes are then used to draw a box and label on the
vehicle in the video. The overall algorithm of RT-VDC is given in Algorithm 3.

**Algorithm 3** RT-VDC

1: \( \text{skip} \text{count} \leftarrow 0 \)
2: \( \text{frame} \text{count} \leftarrow 0 \)
3: \( \text{labels}_\text{save} \leftarrow \emptyset \)
4: \( \text{boxes}_\text{save} \leftarrow \emptyset \)
5: while \( \text{frame} \) do
6: if \( \text{frame} \text{count} \% \text{skip} \text{detect} = 0 \) then continue\end{if}
8: end if
9: \( \text{boundingBoxes} = VDM(\text{frame}) \)
10: if \( \text{skip} \text{count} > 0 \) then
11: \( \text{labels} = \text{labels}_\text{save} \)
12: \( \text{findBoxIOUs}(/\text{boxes}_\text{save}, \text{boundingBoxes}) \)
13: \( \text{skip} \text{count} = \text{skip} \text{count} - 1 \)
14: else
15: \( \text{cars} = \text{extract}(\text{boundingBoxes}, \text{frame}) \)
16: \( \text{labels}, \text{probs} = VCM(\text{cars}) \)
17: if \( \text{probs} >= \text{skip} \text{threshold} \) then
18: \( \text{skip} \text{count} = \text{skip} \text{classify} \)
19: end if
20: end if
21: \( \text{labels}_\text{save} = \text{labels} \)
22: \( \text{boxes}_\text{save} = \text{boxes} \)
23: \( \text{drawBoxes}(\text{labels}, \text{boundingBoxes}) \)
24: end while

5.4.2. Deep Learning Components

The following subsections will discuss the two deep learning components of RT-VDC in more detail. Namely, the vehicle detection model and the vehicle classification model.

5.4.2.1. Vehicle Detection

For our vehicle detection model, we tweaked the YOLO (You Only Look Once) [81] model to make it faster for live streaming. Although YOLO itself supports simultaneous bounding box prediction and classification, it did not provide the classification accuracy desired on such a large amount of classes, such as the number found in the Stanford and CompCars datasets (this will be explained in the results). YOLO in its original format provides bounding boxes for each object and also classifies each object in bounding box
separately. Since we use different network for classification, we trained this YOLO model to only learn and detect bounding boxes for cars. The benefits of this change is two-fold: 1) it makes the network faster with less parameters to learn 2) it makes the network more accurate in finding the bounding boxes.

Our adapted model takes frames from video streams and is configured to detect only cars. Each frame is resized to 608x608. Although YOLO has pretrained models, we further train YOLO using our dataset described in Section 5.5. For every \( n \)th frame, we impose a \( k \times k \) grid on the image. Then, for each grid cell, the model predicts a vector \((c, x, y, w, h, P)\), where

- \((c)\) is a confidence value that a vehicle exists in that grid cell. Similar to YOLO, we define the confidence to be the probability that the grid cell contains the center of a ROI, multiplied by the IOU (intersection over union) ratio of the ROI and the grid cell area.

- \((x, y)\) are coordinates of the center of the bounding box, relative to the grid cell. A grid cell is only responsible for the bounding box whose center is inside the grid cell. Hence, \(0 \leq x \leq 1\) and \(0 \leq y \leq 1\).

- \((w)\) and \((h)\) are the width and height of the bounding box, respectively, relative to the size of the image.

- \((P)\) is a vector with probabilities that each class corresponds to the object within the bounding box. The length of this vector is determined by the number of classes the model is trying to predict. Since we are training this model for only cars, it has a length of one.

In summary, the model makes 5 predictions \((c, x, y, w, h)\) per grid cell. The total number of values predicted by the model is \(5k\). Note that, the prediction for each of the values is based on the whole image, not the part of the image in the corresponding grid cell.
For this work, we used YOLOv2 since it has the highest potential FPS of the three versions while keeping the input size at 608x608. YOLOv3 and tiny YOLO configurations have potential higher FPS, but these require both downsizing the input data further and sacrificing accuracy. The model we train is a stack of 18 convolution layers, each with 3x3 filters, stride of 1, and padding of 1. A minimal number of pooling layers (five) is used in order to minimize the downsampling of the data.

5.4.2.2. Vehicle Classification

RT-VDC has the ability to make use of many popular convolutional neural network models for vehicle classification [57, 60, 101, 59, 102]. The trade off of this choice is between accuracy and inference time. Since it is well defined that this trade off exists, where larger more complex networks tend to have a higher potential accuracy with longer training and inference times, we do not evaluate this tradeoff in this work. Instead, we take one of the mildly simpler CNN models, AlexNet [57], and compare it to one of the more complex CNN models, Inception v4 [102], in terms of classification accuracy. The configuration of the AlexNet and Inception v4 models follow the same structural design as the original models.

An additional measure we take, in order to speed up the inference time, is frame skipping based on confidence. If, for all extracted cars in a frame, the previous confidence value (i.e., highest probability value) is above a certain threshold, only the vehicle detection module will execute. In other words, the classification component will be skipped. The same label will be applied to the video for the frames skipped. This skip can only happen for a predefined number of frames (by default, 5). Further details of this are given in Algorithm 3.

5.4.3. Infrastructure for Distributed and Real-time Inference

The overall distributed infrastructure of RT-VDC is depicted in Figure 5.2. Each camera can be connected to a local server that is capable of performing inference on the video streams. For further training of the classification component, data is sent back to a central
server and updates are propagated to the local servers. Each inference node is capable of running independently from one another and is able to fetch updates from a centralized server based on a predefined number of steps. In order to optimize inference time, both components are loaded and remain in memory while processing the video streams. For the purpose of RT-VDC, we make use of TensorFlow v1.10 and several Python scripts to handle data preprocessing, distribution, training, and inference. OpenCV is for video and image visualization. The implementation of YOLO is supported by the Darknet library [103].

5.5. Vehicle Dataset

In this section, we will discuss the gathering and preprocessing of images for our dataset. Some of the sources are preexisting while others were created for this work.

5.5.1. Data Gathering

Two existing datasets were utilized for the purposes of training our models. The first is the Stanford Cars dataset [86]. It consists of 16,185 images of 196 classes (make and model) of cars. The second is the Comprehensive Cars (CompCars) dataset, which consists of 136,726 images of 1,716 classes (make and model) of cars [87]. It is worth noting that, while the CompCars dataset contains car part images (e.g., headlight), we did not make use of them in the training of our models for this work. Another important factor of this dataset is that it is currently unfeasible to train a model to identify all 1,716 classes. As such, most works that use this dataset for evaluation use a subset of 431 classes as defined in the original CompCars paper [87]. Aside from these datasets, data was gathered using
two different methods as follows.

5.5.1.1. Web Crawl

For gathering more images from the web, we developed code using the Google Custom Search API. Searches were performed given the keywords in the form of "make model" (e.g., Chevrolet Silverado). This was done for 14 popular car models in an attempt to make the Stanford and CompCars dataset more robust. Since there is a potential for noisy data in a random web crawl, these images were processed by our vehicle detection model described in Section 5.4.2.1. Any images with a confidence score below 50% were filtered out. The majority of these filtered images were of the inside of cars, car logos, or cars impeded by people. As described in Table 5.1, the web crawl resulted in an additional 6,858 images over 14 classes.

5.5.1.2. Traffic Cameras

Louisiana State University has cameras set up throughout the campus, particularly at major traffic intersections or gates entering the inner portion of the campus. We make use of these cameras to gather more data to train RT-VDC. In order to gather ideal images for training our models, we chose one week’s worth of video from heavy traffic hours (7:00 A.M. to 9:00 A.M. and 4:00 P.M. to 6:00 P.M.) from three different cameras, each having a unique angle on vehicles. Images were hand selected based on the visibility of cars and subsequently processed, as described in the next section. Due to the difficulty in hand labeling and the sub optimal resolution of the videos, only 464 images remained that were suitable for training. The benefit of using this small dataset is to fine-tune the classification model for future classification based on the cars detected from these cameras.

5.5.2. Data Preprocessing

For the existing datasets (Stanford and CompCars), no further processing was done other than random mirroring, rotation, and cropping. With the web crawl dataset, as previously mentioned, the images were processed by our detection component in order to
filter out any images with a confidence score below 50% (most of these were car logos or cars impeded by other objects). With respect to the video stream data, further preprocessing was required. Each video stream was first broken into frames. From each frame, bounding boxes were drawn around cars and these cars were extracted. Since the same car can exist in multiple sequential frames, the same car would only be extracted again once at a different angle or in a different position with respect to the camera. Further effort was taken to avoid selecting car images that were obstructed by other cars or objects. It is worth noting that this was a manual process since labels were required for each of these extracted cars.

5.6. Results

Since our framework performs two different tasks, detection and classification, we divide our results into multiple sections. First, we will describe the accuracy of our detection model. Second, we give the accuracy of the classification model. Finally, we provide some analysis of the speed of our model.

5.6.1. Vehicle Detection

For the task of vehicle detection, we evaluate our model, which is based on the YOLO [81] architecture. To perform this evaluation, we used the testing set of the Stanford cars dataset since they provide ground truth bounding boxes. This set consists of 8,041 images. The standard metric used to evaluate object detection accuracy is Intersection Over Union (IOU). The IOU is calculated as follows:
Figure 5.3. Visualizing the different IOU thresholds

\[ IOU = \frac{box_p \cap box_{gt}}{box_p \cup box_{gt}} \]  

(5.1)

where \( box_p \) and \( box_{gt} \) refer to the predicted and ground truth bounding boxes, respectively.

The accuracy given the IOU is calculated as follows:

\[ a = \frac{\sum_i c_i}{n} \]  

(5.2)

where:

\[ c_i = \begin{cases} 1, & \text{if } IOU_i \geq t \\ 0, & \text{otherwise} \end{cases} \]  

(5.3)

Table 5.2 gives the percentages of cars that meet certain IOU thresholds. Figure 5.3 allows us to visualize these different thresholds. Based on this, and based on various object detection competitions using 50% IOU as their starting point for evaluation, we can consider an IOU of 50% corresponds to a successful detection [83]. According to Table 5.2, over 78% of cars were detected with an IOU value of over 50%. The IOU value quickly drops off as we increase the threshold to 75% and 90%. Overall, the average IOU value we get for this dataset is 61.58%.

### 5.6.2. Vehicle Classification

In this section, we will discuss the accuracy of the vehicle classification component. The goal of this component is to achieve acceptable accuracy (i.e., accuracy similar to or
Table 5.2. Percentage of cars $a$ that are above varying IOU thresholds $t$.

<table>
<thead>
<tr>
<th>IOU Threshold ($t$)</th>
<th>Percentage of Cars ($a$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>93.4%</td>
</tr>
<tr>
<td>50%</td>
<td>78.56%</td>
</tr>
<tr>
<td>75%</td>
<td>27.36%</td>
</tr>
<tr>
<td>90%</td>
<td>2.1%</td>
</tr>
</tbody>
</table>

close to other existing works). Table 5.3 shows the results for both AlexNet and Inception v4 for Stanford and CompCars datasets. For the purpose of these results, we reduced the number of classes in the CompCars dataset significantly as was similarly done by Yang et al. [87], where they reduced the number of classes to 431 for the vehicle image classification task. We follow the same method used in their work to reduce the classes and evaluate the classification accuracy based on this subset. It is worth noting that the CNNs used in this work, AlexNet and Inception v4, were not capable of producing sufficient accuracy with all 1,716 classes. This seems to be the case with other works regarding this dataset, which explains the reason for the reduction to 431 classes. We use YOLO as a baseline since, in this work, we argue that supplementing YOLO with a second component for fine-grained classification will benefit the overall accuracy.

For the Stanford dataset, when only considering makes, YOLO performs well with 94.43% accuracy. However, when considering different models and years (196 classes), YOLO has a significant dropoff in accuracy at 1.39%. Given this result, and the result of the other two models (AlexNet and Inception v4), it is clear that there is a benefit to having a second component for fine-grained classification. With the reduced CompCars dataset, AlexNet is able to produce acceptable results with 30.47% accuracy (57.81% if considering top-5). For a frame of reference, Yang et al. [87] achieved 76.7% accuracy. Inception v4, however, was not able to achieve the same level of accuracy with 12.5% (36.41% for top-5). While other works reported models like GoogLeNet (Inception v1) achieving higher results, we did not have the same experience with a larger and more complex model [91, 92].
Table 5.3. Fine-grained classification accuracy for RT-VDC.

<table>
<thead>
<tr>
<th>Model</th>
<th>Stanford-49</th>
<th>Stanford-196</th>
<th>CompCars-431</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLO (Top-1)</td>
<td>94.43%</td>
<td>1.39%</td>
<td>–</td>
</tr>
<tr>
<td>AlexNet (Top-1)</td>
<td>87.3%</td>
<td>72.43%</td>
<td>30.47%</td>
</tr>
<tr>
<td>AlexNet (Top-5)</td>
<td>96.81%</td>
<td>88.77%</td>
<td>57.81%</td>
</tr>
<tr>
<td>Inception v4 (Top-1)</td>
<td>91.68%</td>
<td>85.1%</td>
<td>12.5%</td>
</tr>
<tr>
<td>Inception v4 (Top-5)</td>
<td>97.26%</td>
<td>89.54%</td>
<td>36.41%</td>
</tr>
</tbody>
</table>

5.6.3. Real-time Classification

In order to evaluate the speed of our framework, we use video footage from a major campus intersection in order to accurately depict how fast the model is under heavy stress (i.e., consistent and many cars in view). The video was taken during heavy traffic time (8:00 AM to 9:00 AM) to maximize the amount of traffic. Table 5.4 gives the frames per second (FPS) given three state-of-the-art GPUs. Since traffic flow and severity can fluctuate, the FPS calculation is the maximum number of frames possible for real-time inference. This was over the calculated over the course of one hour. For each frame, every car detected is batched together for inference in order to optimize the time. It is important to note that, since we are aiming for real-time, we cannot batch frames/images, which explains our lower FPS compared to that advertised by YOLO (40 FPS for v2). In comparison to other works, Zhou et al. achieved as high as 46 for vehicle detection but only 4 FPS with the verification task which includes vehicle type classification [94]. So, to perform

![Figure 5.4. Detections and classifications from RT-VDC.](image-url)
Table 5.4. Frames per second of different GPUs. The classification component of RT-VDC used the AlexNet-based model. The resolution of the video after processing is 608x608.

<table>
<thead>
<tr>
<th>GPU</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>P40</td>
<td>10</td>
</tr>
<tr>
<td>P100</td>
<td>10.6</td>
</tr>
<tr>
<td>V100</td>
<td>12.5</td>
</tr>
</tbody>
</table>

region proposal and get car type, pose, and color, it is 4 FPS. Their evaluation was done on the PASCAL VOC 2007 dataset. In another work, Yu et al. claimed to process 5 images per second for vehicle classification [93]. These images are from the CompCars dataset. RT-VDC achieves a higher FPS than both of these works as shown in 5.4.

5.7. Conclusion

In this work, we presented RT-VDC, a large-scale deep learning framework for streamed vehicle classification over distributed infrastructures. Our contributions included the gathering and preprocessing of car images from the web and video streams, a new deep learning tool consisting of two components performing two tasks (vehicle detection and make/model classification), and a distributed framework for large-scale training and analysis of traffic video streams in real time, suitable for a campus or similar environment. We showed that RT-VDC is capable to achieving acceptable accuracy results while still producing real-time results that are better than existing works. Consequently, it can be observed that, in the case of fine-grained features, it is beneficial to have a separate model dedicated towards the classification task. In future works, efforts could be made to increase the FPS and the classification component can be improved to take more advantage of fine-grained features.
Chapter 6. Conclusion

In this dissertation, I presented research intended to advance large-scale data analysis and deep learning. This was done through the utilization of various algorithms designed for distributed cyberinfrastructures and high performance computing environments. In each case, the algorithm was developed with both the data, task, and computing environment in mind. An evaluation of these algorithms and the different computing environments were given.

In the first work, I presented HaRE, a Hadoop-based framework capable of performing Replica Exchange Molecular Dynamics simulations across a variety of distributed cyberinfrastructures. HaRE was evaluated in scale-up and scale-out scenarios in these different environments. Further analysis and discussion was provided to show how an application exhibiting task-level parallelism was able to leverage hardware in certain scenarios.

Next, efforts were made to evaluate popular deep learning frameworks in scale-out and scale-up scenarios in different computing environments. Three frameworks were chosen with both popularity and potential scalability in mind: TensorFlow, Caffe, and SINGA. Furthermore, several state-of-the-art hardwares were evaluated with these different frameworks including Knight’s Landing processor and NVIDIA’s NVLink interconnect. Through these experiments, insights were given towards the benefits of different frameworks with certain hardware setups.

Following these experiments, Apache SINGA, a deep learning framework capable of large-scale deep learning, for the task of ADHD prediction from fMRI images. Since fMRI can be considered as 3-dimensional data with its stacked 2-dimensional slices, we implemented the 3-dimensional convolution algorithm in the SINGA framework. Following this, we exhibited the 3D Convolutional Neural Network model’s accuracy and scalability.

Lastly, TensorFlow was used as a basis to develop real-time deep learning inference on video streams for car make and model classification. Since vehicle make and model classification is a fine-grained classification task, it is more challenging than classic object
detection and classification, as that seen with the ImageNet dataset [104]. Consequently, we implemented a two-stage process in our framework where the first component is responsible for distinguishing and detecting cars from other objects in the video stream, and the second component is solely responsible for fine-grained classification of the vehicles. The framework exhibits real-time detection and classification while maintaining respectable accuracy.

Overall, in this dissertation, I evaluated different large-scale frameworks and algorithms for data analysis and deep learning. Several applications and algorithms were developed with the framework and computing environment in mind. Because of this, the applications were able to achieve optimal computing performance while still targeting a difficult scientific problem.
Appendix. Copyright Information
This Agreement between Richard Platania ("You") and John Wiley and Sons ("John Wiley and Sons") consists of your license details and the terms and conditions provided by John Wiley and Sons and Copyright Clearance Center.

License Number: 4574300283929
License date: Apr 22, 2019
Licensed Content Publisher: John Wiley and Sons
Licensed Content Publication: CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Licensed Content Title: Hadoop-based replica exchange over heterogeneous distributed cyberinfrastructures
Licensed Content Author: Richard Platania, Shayan Shams, Chui-Hui Chiu, et al
Licensed Content Date: Jun 3, 2016
Licensed Content Volume: 29
Licensed Content Issue: 4
Licensed Content Pages: 14
Type of use: Dissertation/Thesis
Requestor type: Author of this Wiley article
Format: Electronic
Portion: Full article
Will you be translating?: No
Title of your thesis / dissertation: LARGE-SCALE ANALYSIS AND DEEP LEARNING WITH BIOLOGICAL DATA USING DISTRIBUTED CYBERINFRASTRUCTURES AND HIGH PERFORMANCE COMPUTING
Expected completion date: Aug 2019
Expected size (number of pages): 110
Requestor Location: Richard Platania
280 McDonald Ave

Baton Rouge, LA 70808
United States
Attn: Richard Platania

Publisher Tax ID: EU826007151
Total: 0.00 USD

TERMS AND CONDITIONS
This copyrighted material is owned by or exclusively licensed to John Wiley & Sons, Inc. or one of its group companies (each a "Wiley Company") or handled on behalf of a society with which a Wiley Company has exclusive publishing rights in relation to a particular work (collectively "WILEY"). By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction.
(along with the billing and payment terms and conditions established by the Copyright Clearance Center Inc., ("CCC's Billing and Payment terms and conditions"), at the time that you opened your RightsLink account (these are available at any time at http://myaccount.copyright.com).

Terms and Conditions

- The materials you have requested permission to reproduce or reuse (the "Wiley Materials") are protected by copyright.

- You are hereby granted a personal, non-exclusive, non-sub licensable (on a stand-alone basis), non-transferable, worldwide, limited license to reproduce the Wiley Materials for the purpose specified in the licensing process. This license, **and any CONTENT (PDF or image file) purchased as part of your order**, is for a one-time use only and limited to any maximum distribution number specified in the license. The first instance of republication or reuse granted by this license must be completed within two years of the date of the grant of this license (although copies prepared before the end date may be distributed thereafter). The Wiley Materials shall not be used in any other manner or for any other purpose, beyond what is granted in the license. Permission is granted subject to an appropriate acknowledgement given to the author, title of the material/book/journal and the publisher. You shall also duplicate the copyright notice that appears in the Wiley publication in your use of the Wiley Material. Permission is also granted on the understanding that nowhere in the text is a previously published source acknowledged for all or part of this Wiley Material. Any third party content is expressly excluded from this permission.

- With respect to the Wiley Materials, all rights are reserved. Except as expressly granted by the terms of the license, no part of the Wiley Materials may be copied, modified, adapted (except for minor reformatting required by the new Publication), translated, reproduced, transferred or distributed, in any form or by any means, and no derivative works may be made based on the Wiley Materials without the prior permission of the respective copyright owner. **For STM Signatory Publishers clearing permission under the terms of the STM Permissions Guidelines only, the terms of the license are extended to include subsequent editions and for editions in other languages, provided such editions are for the work as a whole in situ and does not involve the separate exploitation of the permitted figures or extracts**, you may not alter, remove or suppress in any manner any copyright, trademark or other notices displayed by the Wiley Materials. You may not license, rent, sell, loan, lease, pledge, offer as security, transfer or assign the Wiley Materials on a stand-alone basis, or any of the rights granted to you hereunder to any other person.

- The Wiley Materials and all of the intellectual property rights therein shall at all times remain the exclusive property of John Wiley & Sons Inc, the Wiley Companies, or their respective licensors, and your interest therein is only that of having possession of and the right to reproduce the Wiley Materials pursuant to Section 2 herein during the continuance of this Agreement. You agree that you own no right, title or interest in or to the Wiley Materials or any of the intellectual property rights therein. You shall have no rights hereunder other than the license as provided for above in Section 2. No right, license or interest to any trademark, trade name, service mark or other branding ("Marks") of WILEY or its licensors is granted hereunder, and you agree that you shall not assert any such right, license or interest with respect thereto.

- **NEITHER WILEY NOR ITS LICENSORS MAKES ANY WARRANTY OR REPRESENTATION OF ANY KIND TO YOU OR ANY THIRD PARTY,**
EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO THE MATERIALS OR THE ACCURACY OF ANY INFORMATION CONTAINED IN THE MATERIALS, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, ACCURACY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, USABILITY, INTEGRATION OR NON-INFRINGEMENT AND ALL SUCH WARRANTIES ARE HEREBY EXCLUDED BY WILEY AND ITS LICENSORS AND WAIVED BY YOU.

- WILEY shall have the right to terminate this Agreement immediately upon breach of this Agreement by you.

- You shall indemnify, defend and hold harmless WILEY, its Licensors and their respective directors, officers, agents and employees, from and against any actual or threatened claims, demands, causes of action or proceedings arising from any breach of this Agreement by you.

- IN NO EVENT SHALL WILEY OR ITS LICENSORS BE LIABLE TO YOU OR ANY OTHER PARTY OR ANY OTHER PERSON OR ENTITY FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, INDIRECT, EXEMPLARY OR PUNITIVE DAMAGES, HOWEVER CAUSED, ARISING OUT OF OR IN CONNECTION WITH THE DOWNLOADING, PROVISIONING, VIEWING OR USE OF THE MATERIALS REGARDLESS OF THE FORM OF ACTION, WHETHER FOR BREACH OF CONTRACT, BREACH OF WARRANTY, TORT, NEGLIGENCE, INFRINGEMENT OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA, FILES, USE, BUSINESS OPPORTUNITY OR CLAIMS OF THIRD PARTIES), AND WHETHER OR NOT THE PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.

- Should any provision of this Agreement be held by a court of competent jurisdiction to be illegal, invalid, or unenforceable, that provision shall be deemed amended to achieve as nearly as possible the same economic effect as the original provision, and the legality, validity and enforceability of the remaining provisions of this Agreement shall not be affected or impaired thereby.

- The failure of either party to enforce any term or condition of this Agreement shall not constitute a waiver of either party's right to enforce each and every term and condition of this Agreement. No breach under this agreement shall be deemed waived or excused by either party unless such waiver or consent is in writing signed by the party granting such waiver or consent. The waiver by or consent of a party to a breach of any provision of this Agreement shall not operate or be construed as a waiver of or consent to any other or subsequent breach by such other party.

- This Agreement may not be assigned (including by operation of law or otherwise) by you without WILEY's prior written consent.

- Any fee required for this permission shall be non-refundable after thirty (30) days from receipt by the CCC.

- These terms and conditions together with CCC's Billing and Payment terms and conditions (which are incorporated herein) form the entire agreement between you and
WILEY concerning this licensing transaction and (in the absence of fraud) supersedes all prior agreements and representations of the parties, oral or written. This Agreement may not be amended except in writing signed by both parties. This Agreement shall be binding upon and inure to the benefit of the parties' successors, legal representatives, and authorized assigns.

- In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall prevail.

- WILEY expressly reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

- This Agreement will be void if the Type of Use, Format, Circulation, or Requestor Type was misrepresented during the licensing process.

- This Agreement shall be governed by and construed in accordance with the laws of the State of New York, USA, without regards to such state's conflict of law rules. Any legal action, suit or proceeding arising out of or relating to these Terms and Conditions or the breach thereof shall be instituted in a court of competent jurisdiction in New York County in the State of New York in the United States of America and each party hereby consents and submits to the personal jurisdiction of such court, waives any objection to venue in such court and consents to service of process by registered or certified mail, return receipt requested, at the last known address of such party.

**WILEY OPEN ACCESS TERMS AND CONDITIONS**

Wiley Publishes Open Access Articles in fully Open Access Journals and in Subscription journals offering Online Open. Although most of the fully Open Access journals publish open access articles under the terms of the Creative Commons Attribution (CC BY) License only, the subscription journals and a few of the Open Access Journals offer a choice of Creative Commons Licenses. The license type is clearly identified on the article.

**The Creative Commons Attribution License**

The [Creative Commons Attribution License (CC-BY)](http://creativecommons.org/licenses/by/) allows users to copy, distribute and transmit an article, adapt the article and make commercial use of the article. The CC-BY license permits commercial and non-

**Creative Commons Attribution Non-Commercial License**

The [Creative Commons Attribution Non-Commercial (CC-BY-NC) License](http://creativecommons.org/licenses/by-nc/) permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.(see below)

**Creative Commons Attribution-Non-Commercial-NoDerivs License**

The [Creative Commons Attribution Non-Commercial-NoDerivs License (CC-BY-NC-ND)](http://creativecommons.org/licenses/by-nc-nd/) permits use, distribution and reproduction in any medium, provided the original work is properly cited, is not used for commercial purposes and no modifications or adaptations are made. (see below)

**Use by commercial "for-profit" organizations**

Use of Wiley Open Access articles for commercial, promotional, or marketing purposes requires further explicit permission from Wiley and will be subject to a fee. Further details can be found on Wiley Online Library [http://olabout.wiley.com/WileyCDA/Section/id-410895.html](http://olabout.wiley.com/WileyCDA/Section/id-410895.html)
Other Terms and Conditions:

v1.10 Last updated September 2015
Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.
References


[22] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: A not-so-foreign language for data processing, 2008.


Vita

Richard Platania received his Bachelor’s degree in Computer Science from Louisiana State University in 2013. He continued his studies at LSU to pursue his Ph.D. where he specialized in Big Data and Deep Learning related research. Richard originally hails from Pittsburgh, Pennsylvania but has lived in Baton Rouge, Louisiana for over a decade. In his free time, he enjoys spending time with his wife, dog, and cat and trying as many different beers and bourbons as possible.