

Design and Experimental Evaluation of DeepMarket:
An Edge Computing Marketplace with Distributed TensorFlow Execution Capability

by
Soyoung Kim

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

Thesis Committee:
Ehsan Aryafar, Chair
Nirupama Bulusu
Wu-chi Feng

Portland State University
2019

Abstract

There is a rise in demand among machine learning researchers for powerful computational resources to train complex machine learning models, *e.g.*, deep learning models. In order to train these models in a reasonable amount of time, the training is often distributed among multiple machines; yet paying for such machines (either through renting them on cloud data centers or building a local infrastructure) is costly. DeepMarket attempts to reduce these costs by creating a marketplace that integrates multiple computational resources over a distributed TensorFlow framework. Instead of requiring users to rent expensive GPU/CPU from a third-party cloud provider, DeepMarket allows users to lend their edge computing resources to each other when they are available. Such a marketplace, however, requires a credit mechanism that ensures users receive resources in proportion to the resources they lend to others. Moreover, DeepMarket must respect users' needs to use their own resources and the resulting limits on when resources can be lent to others. In this thesis, I present the design and implementation of DeepMarket, an architecture that addresses these challenges and allows users to securely lend and borrow computing resources. I also present preliminary experimental evaluation results that show DeepMarket's performance, in terms of job completion time, is comparable to third-party cloud providers. However, DeepMarket can achieve this performance with reduced cost and increased data privacy.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor Ehsan Arya-far who has been a strong source of motivation and advising. I sincerely appreciate his guidance, support, and patience toward my thesis. He has been not only a driving force in my motivation, but has been a great influence to me as well.

I appreciate that both Professor Nirupama Bulusu and Professor Wu-chi Feng donated their time serving as my committee members and giving me a valuable opportunity to get their professional perspectives and advice on my thesis.

I would also like to thank my research team who developed and researched DeepMarket together and always gave me thoughtful support and comments. I really appreciate the collaborative work and the opportunity to be a part of this group with Susham, Sam, Martin, Amarjit, Rohan, and Shraddha.

Finally, I sincerely thank all the support I received from my father, mother, brother, Zoey, Muna, Ankita, Mahdu, Shruti, and other friends who always encouraged and helped me during my studies at Portland State University.

Contents

Abstract	i
Acknowledgements	ii
Contents	iii
List of Tables	1
List of Figures	2
1 Introduction	4
2 Background	8
2.1 Cloud Computing	8
2.2 Edge Computing	8
2.3 Apache Spark	10
2.4 Hadoop Distributed File System	10
2.5 TensorflowOnSpark	12
3 Architecture	13
3.1 DeepMarket Platform	13
3.1.1 Hardware Infrastructure	13

3.1.2	Software Infrastructure	14
3.2	High Level Architecture Design	14
3.2.1	PLUTO	16
3.2.2	Service Module	20
3.2.3	Executor Module	21
3.2.4	Sequence of Actions in DeepMarket	22
4	Experimental Evaluation	23
4.1	The MNIST Dataset	23
4.2	Experiment Objectives	24
4.3	Resource Configurations	24
4.4	Distributed Computing with Different Numbers of Workers	25
4.4.1	Distributed Computing on the Cloud with Different Numbers of Workers from the Different Data Centers	25
4.4.2	Distributed Computing on the Cloud with Different Numbers of Workers from the Same Data Center	26
4.4.3	Comparison between the Distributed Computing on the Cloud with the Different Data Centers and with the Same Data Center	27
4.5	Distributed Computing in a Local Cluster on LAN Wi-Fi Connection	28
4.5.1	Distributed Computing in a Local Cluster on a LAN Wi-Fi Connec- tion with Different Sizes of RAM	28
4.5.2	Comparison between the Distributed Computing in a Local Cluster on LAN Wi-Fi Connection and in a Cluster on the Cloud	29
4.6	Experiments with Competing Network Traffic	30
4.6.1	Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio	31

4.6.2	Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video	33
4.6.3	Comparison between the Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video and from Live Streaming Radio	35
4.7	Distributed Computing on a LAN Wi-Fi Connection with the Interference from Electrical Appliances	36
5	Related Work	40
5.1	Advantages of Edge Computing over Cloud Computing	40
5.1.1	The Cost of Cloud Computing	40
5.1.2	Why Edge Computing?	41
5.1.3	Network Latency	41
5.1.4	Bandwidth Requirements	42
5.1.5	Privacy and Security	42
5.2	Existing Infrastructure and Their Difference with DeepMarket	43
6	Discussion	45
7	Conclusion	47
8	Future Work	49
	Bibliography	50

List of Tables

4.1	Machine Configurations for the Distributed Computing Test on the Cloud with Different Numbers of Workers from the Different Data Centers	25
4.2	Machine Configurations for the Distributed Computing Test on the Cloud with Different Numbers of Workers from the Same Data Center	26
4.3	Machine Configurations for the Distributed Computing Test in a Local Cluster on a LAN Wi-Fi Connection with Different Numbers of Workers	28
4.4	Training Time Result of the Distributed Computing in a Local Cluster on a LAN Wi-Fi Connection with Different Sizes of RAM	29
4.5	Machine Configurations for the Distributed Computing Experiments with Competing Network Traffic	30
4.6	Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio	31
4.7	Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video	33
4.8	Specifications of the Microwave Oven	37
4.9	Comparison between the Distributed Computing with Microwave Interference and with Live Streaming Radio Traffic on a LAN Wi-Fi Connection	38
5.1	Existing Testbeds and Four Key Properties of DeepMarket	44

List of Figures

2.1	Edge Computing Paradigm [1]	9
3.1	DeepMarket Hardware Infrastructure	14
3.2	DeepMarket High Level Architecture Design	15
3.3	PLUTO Login User Interface	16
3.4	PLUTO Dashboard Tab	17
3.5	PLUTO “Add Job” Tab	18
3.6	An Example of Job Submission	18
3.7	PLUTO “Add Resources” Tab	19
3.8	PLUTO “Resources List” Tab	20
3.9	Sequence of Actions in DeepMarket	22
4.1	Images from the MNIST Test Dataset [2]	23
4.2	Training Time Result of the Distributed Computing on the Cloud with Different Numbers of Workers from the Different Data Centers	26
4.3	Training Time Result of the Distributed Computing on the Cloud with Different Numbers of Workers from the Same Data Center	27
4.4	Training Time Result of the Comparison between the Distributed Computing on the Cloud with the Different Data Centers and with the Same Data Center . . .	27

4.5	Training Time Result of the Comparison between the Distributed Computing in a Local Cluster on a LAN Wi-Fi Connection and in a Cluster on the Cloud with the Same and Different Data Centers	29
4.6	Training Time Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio	32
4.7	Timeout Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio	32
4.8	Training Time Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video	34
4.9	Timeout Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video	34
4.10	Training Time Result of the Comparison between the Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video and from Live Streaming Radio	35
4.11	Timeout Result of the Comparison between the Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video and from Live Streaming Radio	35
4.12	Picture of Microwave Oven, Modem, and Router Used in the Test with the Interference from Electrical Appliances	37
4.13	Training Time Result of the Comparison between the Distributed Computing with Microwave Interference and with Live Streaming Radio Traffic on a LAN Wi-Fi Connection	38
4.14	Timeout Result of the Comparison between the Distributed Computing with Microwave Interference and with Live Streaming Radio Traffic on a LAN Wi-Fi Connection	39

Chapter *I*

Introduction

As deep learning proves its usefulness in an ever greater number of applications, there is a rise in demand for faster and cheaper computational resources to manage and train ever more complex learning-based models. Currently, there are two approaches to meet this demand. One approach is to purchase computational resources, which requires a substantial upfront investment that loses its value over time. Another approach is to rent computing resources from cloud service providers. Cloud computing web services such as Amazon AWS [3], Microsoft Azure [4], and Digital Ocean [5] are a few of the alternatives to use instead of private/local computational resources. They are becoming popular as artificial intelligence and machine learning fields become more and more utilized. However, the main caveat of using these cloud platforms for training machine learning models is cost. For example, training a typical language model with 10 GPUs takes 3 weeks and costs \$1000 assuming discounted Amazon Web Services (AWS) spot prices. As a solution to this problem, this thesis introduces DeepMarket, which supports a hybrid cloud computing environment that utilizes edge computing to train machine learning models at a significantly reduced cost.

DeepMarket benefits users in two ways. First, DeepMarket will reduce the distributed computing costs by creating a marketplace where users lend each other their spare computational resources. Users can lend their resources to DeepMarket when they don't need them and then supplement them with other people's machine when they have a need. This reduces

upfront investment costs by allowing users to purchase fewer resources and supplement them with others' machines, and it eliminates the need to pay a cloud provider. Second, DeepMarket is an open-source and fully programmable research infrastructure that allows experimentation in other domains such as edge computing and pricing, among others.

DeepMarket leverages a shared economy of available computational resources, similar to Uber or Airbnb. As mentioned, users can participate in the online marketplace by lending their private and idle resources to each other through DeepMarket. By sharing their computational resources, users can earn credits which they can use later to borrow others' available resources and train their machine learning models. In this manner, researchers could use resources on DeepMarket for their computations possibly for free by sharing their available resources with others. Also, DeepMarket has a price generator mechanism dependent on the job execution time and the available/selected resources. With this, DeepMarket can proactively calculate how many credits a resource owner will get and how many credits a user should expect to spend on his job. Also, DeepMarket uses a client-server architecture where the client is a local application, thus allowing users to easily set up their personal machines as resources. Because DeepMarket is inherently distributed, targeting users' computers for computation requires the existence of a distributed resource manager, and in this thesis, I have primarily leveraged the Apache Spark distributed resource manager.

This idea of DeepMarket is mainly to train deep learning models at the edge and in a distributed manner. As mentioned before, since exploiting computational resources at the edge of the network has several advantages over using cloud computing resources, this idea is getting more attention these days. There are a few recent startups [6, 7, 8, 9] and academic research [10, 11] which proposed similar ideas to DeepMarket. DeepMarket can be specialized among these research works and startups since it relies on the sharing economy with a credit-based pricing algorithm and marketizes the execution of machine learning tasks at the same time as providing a broker service for distributed TensorFlow jobs. Also, as DeepMarket aims to provide distributed computing service on heterogeneous

resources and networks, it will allow a wide range of users to join the marketplace.

DeepMarket mainly has two research capabilities. It provides massive amounts of resources for machine learning researchers to train their models at a much-reduced cost. These researchers only need computational power, so they have an incentive to lend or borrow resources to DeepMarket due to the reduced cost. Also, they can act as real test users who respond to dynamic pricing conditions. In addition, DeepMarket provides a controllable edge for a variety of other researchers to run experiments with such real users. For example, researchers can run experiments on network economics and resource scheduling by testing their new pricing algorithms and by designing new resources to job mapping algorithms.

DeepMarket is developed in collaboration with other NeWSLab (Networks and Wireless Systems Lab) students at Portland State University (PSU). I participated in this project as a software developer and a system performance monitor. Specifically, I modeled the entire database with other team members' feedback and worked on the database creation, integration, server-side API development, and unit test cases development in the early stages of DeepMarket. Currently, I am researching other distributed computing tools for stability and improvement of the DeepMarket platform and focusing on testing and evaluating the performance of DeepMarket.

In this thesis, I will present and discuss the design and preliminary experimental evaluation of DeepMarket. In particular, this thesis will present the architecture, implementation, and benefits of DeepMarket, and introduce its graphical user interface (PLUTO), the service module, and the executor module. Also, I will present preliminary experiments that characterize the performance of DeepMarket.

The contents of this thesis are organized as follows. I start this thesis with background information on DeepMarket explaining why Apache Spark, Hadoop Distributed File System, and TensorFlowOnSpark are the key components of DeepMarket and where they are used. In addition, I address the difference between cloud computing and edge computing. In chapter 3, I introduce the architecture design of DeepMarket with details of each of the

three modules; PLUTO, the service module, and the executor module. In chapter 4, my experiments and the corresponding results will be presented. The first 2 experiments will be mainly focused on the performance comparison between cloud computing and DeepMarket, and the preliminary results will show that DeepMarket's job completion time on edge resources is comparable to running jobs on cloud. The next 2 experiments will evaluate DeepMarket's job execution performance in presence of dynamic network traffic.

Chapter 2

Background

2.1 Cloud Computing

Cloud computing is the most popular distributed computing model where a number of servers are remotely connected to a centralized data center [12]. Cloud computing services are getting more attention since it is convenient for researchers and other users to get remote and virtualized computing resources without purchasing local computing machines on site. Examples of the most sought after cloud computing platforms are Amazon Web Services, Microsoft Azure, and Digital Ocean. Most of the cloud computing services are based on virtualization techniques which are for sharing and utilizing physical computing resources remotely. With this, users' local resources can just request cloud computing resources and utilize the virtualized computing resources temporarily as long as they would like to use it. Local machines need only request these kinds of cloud computing service [12]. However, cloud computing has several main disadvantages such as possible long latency on WAN connections and longer execution time when processing a large data set [13].

2.2 Edge Computing

Edge computing performs computation at the edge of network which is at the proximity of data sources [1]. Therefore, edge computing supports decentralized distributed computing

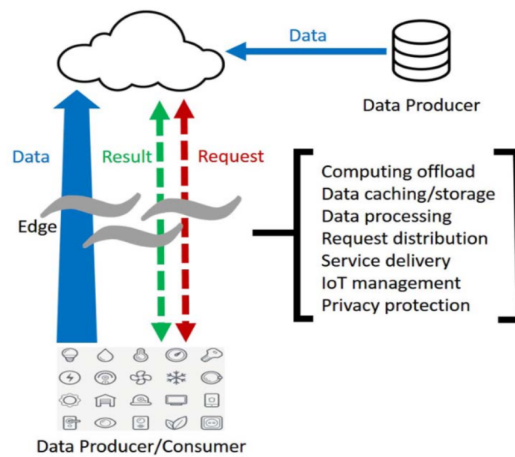


Figure 2.1: Edge Computing Paradigm [1]

with local machines at the edge which is close to users and data sources, so it doesn't have to use data center storage or extra network expense [13]. With this, edge resources and network can perform computing offloading, data caching/storage, data processing, privacy protection, and others by having data storage closer to the distributed nodes as shown in Figure 2.1 [1]. In the other words, edge devices can be both data consumer and data producer, and this allows computation partially or completely on distributed nodes [1]. Compared to cloud computing, this can minimize the disadvantages of cloud computing including the maintenance cost of the data center, network cost, and others. Also, since edge computing does not upload or process private data on the cloud, physical data collected on the network at the edge stays private, and this can protect user privacy. However, this computing method also has disadvantages. First, edge computing needs to have a certain amount of locally available computing machines which are sometimes hard to obtain for certain use cases. Second, even though users have resources, the network configuration and connection could be difficult to set up if the users are not experts or researchers. Third, edge computing could pose security risks as edge devices typically have lower computational power (*e.g.*, may not be able to fully implement cryptography/encryption algorithms).

2.3 Apache Spark

Apache Spark [14] is a cluster computing framework with strong scalability and fault tolerance mainly by the abstraction called resilient distributed dataset (RDD) [15]. RDD is a read-only collection of a partitioned object from a dataset for parallel operations [15]. Not only does this support Spark to have scalability by distributing a big amount of data to partitions, but this also helps Spark with its fault tolerance by caching this RDD in memory across nodes to reuse or rebuild the partition whenever there are any corruptions in the middle of distributed execution [15]. This is an important factor for Spark in distributed computing because there could be a lot of issues in the middle of the connection between nodes in a distributed computing environment, and this RDD can solve these problems. For example, when a node gets overloaded while processing data, the cluster manager should find out and be able to perform the computing again without losing the already processed data. Also, Spark can be a resource manager and cluster manager with a standalone mode, and it is flexible enough to use other resource and cluster managers such as Hadoop Yarn and Apache Mesos. In addition, Spark can handle a larger workload than the total size of memory in a cluster by dividing working sets and performing in-memory computation [16]. Like this, Spark can adjust the size of working sets, so when a straggler task gets detected in the node of a cluster, Spark can re-schedule smaller workloads on the node and improve the overall performance. Lastly, Spark allows DeepMarket to be able to be a hybrid cloud platform since it is easy to be installed and configured in virtual machines by cloud computing services, so DeepMarket with Spark can run applications with the combination of resources from local and cloud computing services.

2.4 Hadoop Distributed File System

DeepMarket mainly uses the Hadoop Distributed File System (HDFS) [17] as its distributed file system, so it can manage and allow multiple resources to store and process a large

amount of data across clusters [18]. HDFS could be very economical since it can be used with low-cost resources, and it is suitable for applications with a massive amount of data [19]. Also, to execute applications in a cluster with multiple resources/workers, the data should be easily accessible to all of the nodes in the cluster, and the file system should be reliable, scalable, and fault tolerant. HDFS can meet these requirements with its user-level file system with high portability, so it has been chosen to be used in DeepMarket. Also, to execute applications in a cluster with multiple resources/workers, the data should be easily accessible to all of the nodes in the cluster, and the file system should be reliable, scalable, and fault tolerant. HDFS can meet these requirements with its user-level file system with high portability, so it has been chosen to be used in DeepMarket. HDFS consists of a name node and data nodes in a cluster. The name node of HDFS divides data into smaller data blocks and distributes them on the data nodes of HDFS, and HDFS maintains a directory tree to keep track of the locations of data blocks. [18]. In other words, through this name node and the directory tree, HDFS can manage and utilize each node's file system, so it can have high portability and scalability with a smaller fraction of data blocks on each node in a cluster. Also, HDFS is reliable and fault tolerant due to its re-replication system. Distributed computing programs can have a number of problems such as a bad disk in one of the workers/resources, nodes becoming unavailable, and a corrupt or an increased part of the replication of a file [19]. HDFS can handle these problems with the communication between a name node and data nodes and making re-replication when it is needed, so the distributed file system can stay reliable and fault tolerant [19]. Due to all these reasons, DeepMarket uses HDFS to get and process input source data and to save the output data after executing distributed computing jobs.

2.5 TensorflowOnSpark

TensorFlowOnSpark is a framework developed by Yahoo to support distributed TensorFlow execution on Apache Spark and Apache Hadoop. Since the current version of DeepMarket uses HDFS and Spark, TensorFlowOnSpark is an ideal framework to use for executing distributed TensorFlow machine learning programs on DeepMarket, and it is included in the executor module of DeepMarket architecture as shown in Figure 3.2. TensorFlowOnSpark is designed to be used in either on cloud or on-premise resources like CPU and GPU, and it can be utilized with any dataset on HDFS and other data sources pulled and pushed from Apache Spark and TensorFlow [20]. Also, it is easy to integrate with other libraries or tools such as Keras and TensorBoard for TensorFlow distributed execution.

Chapter 3

Architecture

3.1 DeepMarket Platform

The DeepMarket platform consists of DeepMarket’s software and hardware infrastructure. The main goal of this platform is to create an ecosystem around the DeepMarket infrastructure by bringing together a diverse group of users (*e.g.*, machine learning users who need computational power and users who leverage DeepMarket’s controllable edge).

3.1.1 Hardware Infrastructure

The hardware infrastructure of DeepMarket is designed to be expandable by adding a variety of computational resources to DeepMarket’s network. Currently, this DeepMarket’s hardware infrastructure is developed and located at Portland State University, and it is managed by our team at the Networks and Wireless Systems Lab. Therefore, multiple individuals and groups at PSU can rent and lend a variety of computational resources to DeepMarket’s network, and they can participate in this project by acting as users/programmers and expanding this hardware infrastructure. As shown in Figure 3.1, the DeepMarket’s hardware infrastructure can be expanded to include a variety of hardware infrastructure at different locations, and we are planning to expand our hardware infrastructure network to include other universities by the end of 2020.

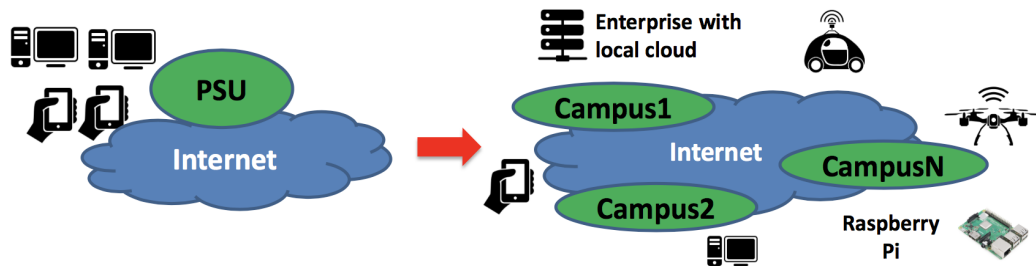


Figure 3.1: DeepMarket Hardware Infrastructure

3.1.2 Software Infrastructure

There are a variety of useful existing software which can be added and developed on top of DeepMarket’s open source software infrastructure. For example, Horovod [21], TensorFlowOnSpark [22], and MLlib [23] are possible frameworks to support parallel processing and machine learning programs. Also, HDFS [17] and IPFS [24] have reliable fault-tolerant data management systems, and Apache Spark [14], Hadoop Yarn [25], Kubernetes [26], and Apache Mesos [27] are cluster management systems which are essential to be used with distributed computing platforms. However, a key missing point in all these frameworks is that architectures are tailored towards data centers. The unique enabler of DeepMarket’s software infrastructure is to extend the architecture to edge/fog. For this, DeepMarket is built on top of Apache Spark as our cluster management system, TensorFlowOnSpark as our ML and parallel processing engines, and HDFS as our file management system.

3.2 High Level Architecture Design

DeepMarket is designed to be an open source application for the scalability and flexibility of the marketplace, so users can easily submit their jobs with a massive amount of data and share their resources with other users. In addition, DeepMarket needs to be secure and fault tolerant for the reliability and privacy matters for users. For these reasons, the current version of DeepMarket’s architecture mainly relies on the Apache Spark and Hadoop

distributed file system known as HDFS. With this basis, DeepMarket’s architecture consists of three modules as shown in Figure 3.2.

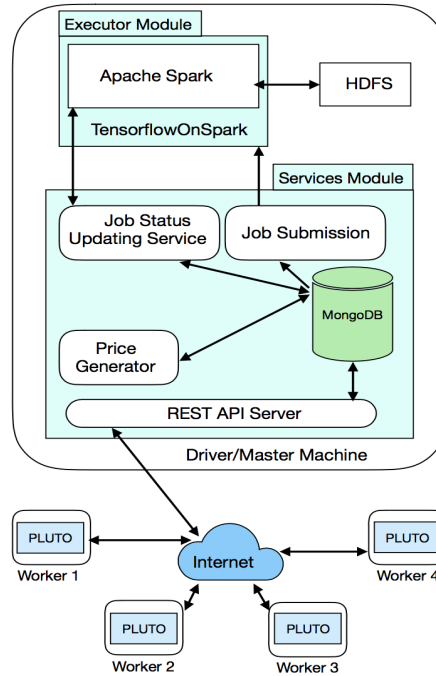


Figure 3.2: DeepMarket High Level Architecture Design

The first module, PLUTO, is a simple graphical user interface of DeepMarket. PLUTO’s main function is to display the GUI that detail the dashboard, resources, and jobs with its pricing options. The second module is called the services module which links users to the master server, which enables DeepMarket to perform job submissions, update job status, generate job price, and do other general services. The third and final piece is the executor module which takes the submitted data in order to execute and schedule the jobs. This executor module is based on the features from both Apache Spark and Hadoop distributed file systems as well as TensorFlowOnSpark library. These three modules interact with each other as shown in Figure 3.2.

Overall, users can register their accounts and resources by installing PLUTO on their machines, and throughout the Internet, a user can submit and share their machine/resources in the online marketplace to get credits which can, in turn, be used for computing jobs.

When the jobs are submitted and resources are shared by the user, all of the information will be updated to the master server, and the resources will be evaluated in order to exchange their values. Finally, in the executor module, Spark will use a possible resource manager and the Hadoop distributed file system to execute a job. How these three modules work and integrate with one another is a complex process, so this will be expanded upon in the following sections.

3.2.1 PLUTO

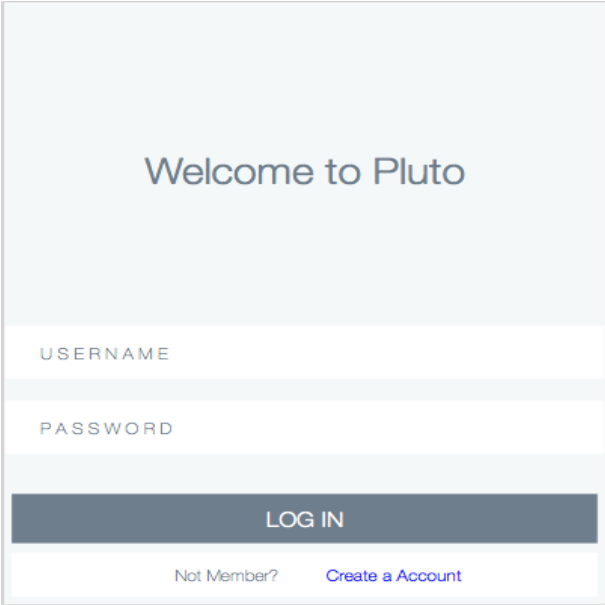
The image shows a login interface for PLUTO. It features a light blue header with the text "Welcome to Pluto". Below this is a white form area with two input fields: "USERNAME" and "PASSWORD". A dark blue "LOG IN" button is positioned below the password field. At the bottom of the form, there are two links: "Not Member?" and "Create a Account".

Figure 3.3: PLUTO Login User Interface

- To be active in the DeepMarket marketplace, users should install the PLUTO GUI on their machines. PLUTO allows users to easily access the marketplace to borrow and/or lend their resource, and exchange their credits by DeepMarket's pricing mechanism. PLUTO will initially ask users to register or log into the application as shown in Figure 3.3, but apart from that, PLUTO has only three major tabs. The first tab is the dashboard which can be seen in Figure 3.4. This tab's main function is to summarize the user's activity including his/her job status. The job status has three varieties which are running, finished, or panic.

Running and finished are self-explanatory while the panic status means that the job has been stopped for several reasons such as poor connectivity to the master server, poor conditions of resources or that the job has been aborted.

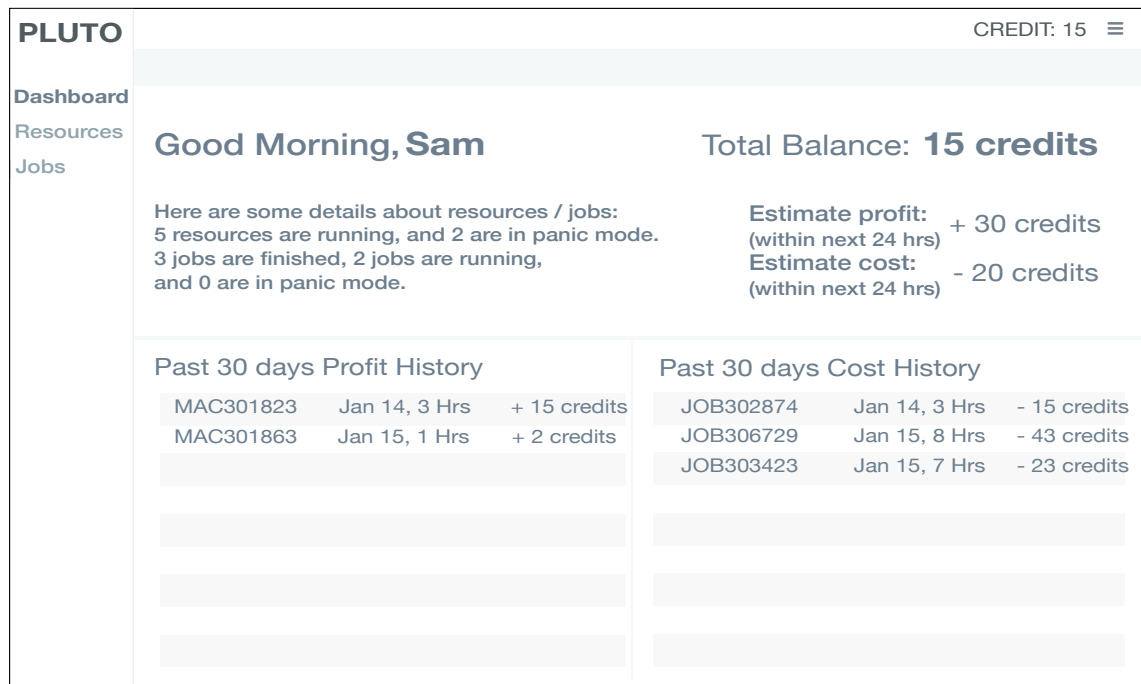


Figure 3.4: PLUTO Dashboard Tab

The second tab is the “Job Tab” which includes two sub tabs, “Add Jobs” and “Jobs List” which are shown in Figure 3.5. Using the “Add Jobs” tab, users can estimate how much their jobs will cost and adjust the price. There are 4 of the 6-hour time slots with difference prices, and users can set the time when they want to run a job. Users can then check available resources and select the number of workers and cores, the memory configuration, and the path to the source and input file needed to execute the job. These two paths will be passed to the service and executor modules to be saved in the Hadoop distributed file system, so they can be computed in a distributed manner during the selected time slot. The information submitted through this “Job Tab” will eventually get to the service module to book a job with a certain time slot, and the service module will then calculate the cost corresponding to the user’s choice of the number of CPU/GPU, RAM, and Disk. For example, in figure 3.6,

PLUTO

CREDIT: 15

Dashboard
Resources
Jobs

Add Jobs
Jobs List

Pricing Per Time Slot

Time	12 AM - 6 AM	6 AM - 12 PM	12 PM - 6 PM	6 PM - 12 AM	Available Resources
CPU:	1 Credit/hr	1 Credit/hr	1 Credit/hr	1 Credit/hr	CPU #: 12
GPU:	7 Credit/hr	7 Credit/hr	7 Credit/hr	7 Credit/hr	GPU #: 2
Memory:	2 Credit/hr	2 Credit/hr	2 Credit/hr	2 Credit/hr	Memory: 24 GB
Disk:	3 Credit/hr	3 Credit/hr	3 Credit/hr	3 Credit/hr	Disk: 40 GB

Job Submission

Workers #:
Cores #:
Memory:

Source file:

Input file:

SUBMIT

Figure 3.5: PLUTO “Add Job” Tab

Job “My Job” will be charged with CPU: 0.005 Credit/hr, GPU: 0.005 Credit/hr, Memory 0.005 Credit/hr, Disk Space: 0.005 Credit/hr and run in the Timeslot: 12:00 AM - 6:00 AM

You are required worker #: 3, Cores #: 5, Memory: 2 GB for this job submission.

Do you want to submit this job at the above mentioned rate and timeslot?

Yes
No

Figure 3.6: An Example of Job Submission

PLUTO is asking a user if the price and the time when the job will be running is acceptable before actually submitting the job. Also, they can check their job list on the “Jobs List” tab which shows the number of workers, the number of cores, memory, status (*e.g.*, scheduled, active, failed, or finished), and the logs with a job’s identification number.

The third tab is the “Resource Tab” where users can verify their machine’s configuration

PLUTO CREDIT: 15

Add Resources Resources List

Dashboard
Resources
Jobs

Resource Verification

IP Address: **VERIFY**

Machine Configuration

CPU: 8 GB Cores #: 4 RAM: 4 GB

Resource Planning

Machine Name: CPUs/GPUs #:
Cores #: RAM (GB): **EVALUATE**

Resource Submission

☒ Automated Price ☐ Offering Price: 0 credit/Hr **SUBMIT**

Figure 3.7: PLUTO “Add Resources” Tab

and check if they can lend their own machines and earn credits for DeepMarket. Like the “Job Tab”, this tab has two sub tabs, the “Add Resources” and “Resources List” tabs as shown in Figure 3.7. The “Add Resource” tab provides four functions for users. First, users should insert the machine’s IP address to verify their ownership and configuration. If the machine is connected to DeepMarket remotely, users additionally need to validate ownership by providing the credentials to PLUTO. Second, when the machine’s configuration is checked and shown to the user, he/she can choose how much memory, CPU/GPU, and cores they would like to lend to DeepMarket. Then, PLUTO checks if the part users want to lend is valid and available. Third, it calculates the credits they can earn by lending the machine through the DeepMarket’s credit system.

Lastly, users can add their resources to DeepMarket, and it will be shown on the “Resources List” tab. The “Resources List” tab shows the list of the machines which the user

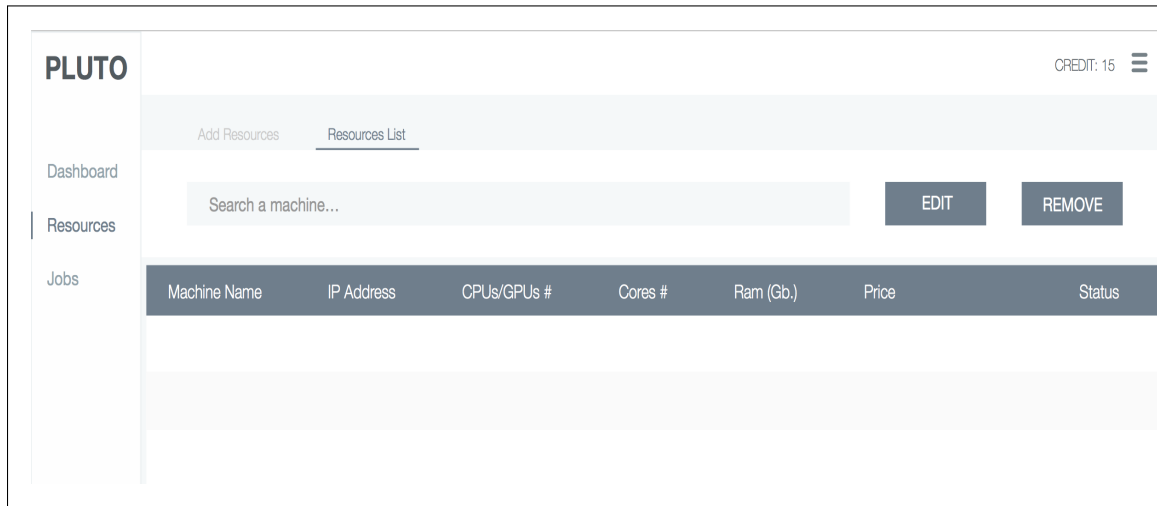


Figure 3.8: PLUTO “Resources List” Tab

lent to the DeepMarket, and it shows the detailed information such as the machine’s name, IP address, the number of CPUs/GPUs and cores, RAM, price, and the current status if the resource is online or not. This tab also includes the function to search, edit, and remove the resources from DeepMarket as shown in Figure 3.8.

3.2.2 Service Module

The Service Module is located in the driver/master machine of DeepMarket and interacts with PLUTOs on the internet as shown in Figure 3.2. This service module manages and updates DeepMarket’s entire MongoDB database through Representational State Transfer Application Programming Interface known as REST API Server, so the user, job, and resource information can be updated in a short time to all of the machines with the PLUTOs on the Internet.

In other words, when users/jobs/resources information is passed to the service module for any of the DeepMarket services, this service module updates the database and PLUTO application with the new data and transfers the job submission to the executor module to run a distributed computing TensorFlow program. Also, this module has a price generator

which updates the price every 12 am, Pacific time (GMT-7:00). It generates the prices for resources depending on different time slots every day and updates the new price on the job tab on PLUTO. This price generator is still under research, so it can produce a cheaper price for the time slot when most of resources are idle and assign a higher price for the time when the most of users want to run a job or when the most of the resources are busy based on the data from previous days.

Lastly, this service module updates the job's status. When a job is submitted, PLUTO sends the scheduled job's information to the server, and when it's the destined time for the job execution, the server sends the job information to Apache Spark driver, so Spark can execute the job and change the job's status to be "Active". Whenever the job status changes to "Failed" or "Finished", the Spark driver from the executor module asks the server on the service module to update the changed status information on its database and PLUTOs.

3.2.3 Executor Module

Similar to the service module, the executor module is located in the driver/master machine mainly to execute submitted jobs with other resources/workers. This executor module uses Apache Spark, Hadoop Distributed File System (HDFS), TensorFlow, and TensorFlowOnSpark to execute distributed computing jobs. In detail, Apache Spark is a resource manager, so it can manage and schedule resources while running distributed programs by TensorFlow and TensorFlowOnSpark framework. Also, HDFS is the distributed file system of DeepMarket, so other resources/workers can also access the massive amount of data needed to compute jobs in a distributed manner. Overall, this module interacts with the service module and PLUTOs for scheduling resources, updating jobs/resources status information, and managing the input and output data files.

3.2.4 Sequence of Actions in DeepMarket

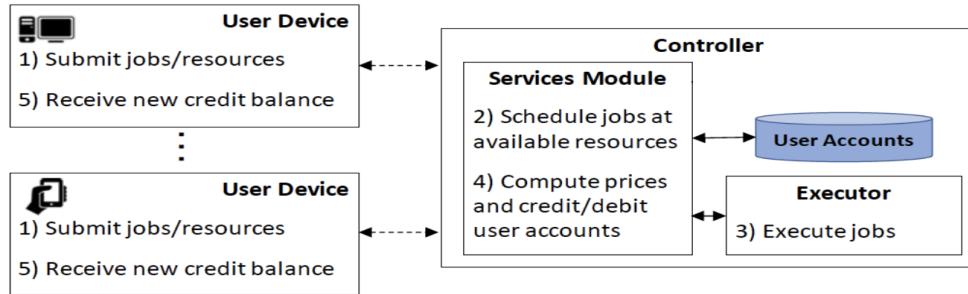


Figure 3.9: Sequence of Actions in DeepMarket

Figure 3.9 shows the sequence of actions in DeepMarket architecture. As previously mentioned, users can access and exploit DeepMarket's infrastructure through the interaction between the PLUTO GUI, the service module, and the executor module. As shown in Figure 3.9, users first need to submit their jobs and resources to the service module on the edge network from their own device. From there, the service module schedules their jobs mapping with available resources during the assigned time slot. Then, the executor module gets the information and executes the job. After completing the job, the executor module sends the result and other information to the service module, so it can compute the price and credits that will be used and update the user account information on the database. Lastly, after the service module finishes and sends the pricing information to the user device, they can receive a new credit balance with the job execution done.

Chapter 4

Experimental Evaluation

4.1 The MNIST Dataset



Figure 4.1: Images from the MNIST Test Dataset [2]

All the experimental evaluations in this chapter used the MNIST dataset. The MNIST dataset stands for Modified National Institute of Standards and Technology database [2]. It is a large database of handwritten digits as shown in Figure 4.1, and it is mainly used for training various image processing systems as well as a basic machine learning program example [2]. In this chapter, experiments are carried out to evaluate the impact of distributing jobs with multiple nodes in different environments, and this MNIST dataset is used to measure its training completion time with different setups of clusters. The accuracy rate was not measured since all the tests train a fixed data model for a given number of iterations, and the

variables of the experiments didn't affect the accuracy.

4.2 Experiment Objectives

There are 5 main experiments in this chapter. These experiments characterize how resources (*e.g.*, conventional laptops, virtual machines in a data center), their locality, and other network traffic would affect the “Job Completion Time”. This “Job Completion Time” is the time it takes to train a fixed model for a given number of epochs. In the first experiment, I wanted to test if DeepMarket would actually be able to improve the performance of a distributed machine learning program by adding more workers to the cluster. In the second experiment, I wanted to see how the locality of machines can affect the performance of distributed computing programs. In the third experiment, I compared the performance of distributed computing on the cloud and on the edge with a LAN Wi-Fi connection. Additionally, I carried an experiment to verify if adding only a fraction of a resource to a cluster could enhance the performance of distributed computing. In the fourth experiment, I tested the impact of the network traffic from everyday life to the performance of distributed computing in a cluster from DeepMarket. In the last experiment, I wanted to test the impact of uncontrolled traffic in a wireless edge environment on DeepMarket's performance.

4.3 Resource Configurations

In this chapter, there are three main different resource configurations. The first one is “Local LAN” configuration. This is a LAN setup in our lab at Portland State University or my apartment. The resources are 3 conventional laptops, and they are all connected to the same LAN Wi-Fi network. The second one is “Single DC”. In this set up, the server and resources are computers from digital Ocean, and all machines are located in a single data center in San Francisco. The third one is “Multi-DC”. In this setup, the server and resources are computers rented from Digital Ocean, and the resources belong to different data centers in

France, India, London, New York, and San Francisco.

4.4 Distributed Computing with Different Numbers of Workers

4.4.1 Distributed Computing on the Cloud with Different Numbers of Workers from the Different Data Centers

Machine Regions	Operating System	RAM	CPU	The Number of Machines	Cost/Hour
San Francisco	Ubuntu 18.04	4GB	2 cores	3	0.030/hr
France	Ubuntu 18.04	8GB	4 cores	1	0.060/hr
London	Ubuntu 18.04	8GB	4 cores	1	0.060/hr
India	Ubuntu 18.04	4GB	2 cores	1	0.030/hr
New York	Ubuntu 18.04	4GB	2 cores	1	0.030/hr

Table 4.1: Machine Configurations for the Distributed Computing Test on the Cloud with Different Numbers of Workers from the Different Data Centers

In this section, I set up 7 different machines as workers in a cluster which are from 5 different cities around the world, and this experiment was to see how the number of workers and the locality of workers would affect the performance of distributed computing. The detailed configurations of the machines are shown in Table 4.1. The server and resources are computers rented from Digital Ocean [5], and the resources belong to different data centers, and the prices are also stated in Table 4.1. The TensorFlowOnSpark MNIST example program [22] I executed for this experiment trains the MNIST dataset in a distributed computing environment, and I used the Hadoop Distributed File System and Apache Spark standalone cluster. To generate a significant difference as the number of workers increases, the result is the training completion time with 100 epochs, and each worker is provided with only 2GB of RAM and 1 core/CPU. The average completion times with different numbers of machines/workers are shown in Figure 4.2. Through this experiment, I wanted to characterize the improvement in DeepMarket’s performance as more workers are added

to the cluster. As expected, the job completion time decreased as the number of workers increased. It shows that, as DeepMarket's adoption increases, it becomes feasible for users to borrow many resources simultaneously, significantly decreasing the job completion time. Therefore, we can see that the performance of distributed computing can be improved by adding more resources/workers to the cluster.

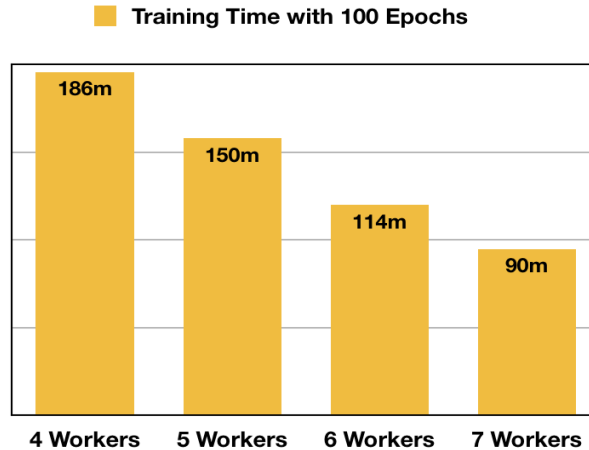


Figure 4.2: Training Time Result of the Distributed Computing on the Cloud with Different Numbers of Workers from the Different Data Centers

4.4.2 Distributed Computing on the Cloud with Different Numbers of Workers from the Same Data Center

Machine Regions	Operating System	RAM	CPU	The Number of Machines	Cost/Hour
San Francisco	Ubuntu 18.04	4GB	2 cores	5	0.030/hr
San Francisco	Ubuntu 18.04	8GB	2 cores	2	0.060/hr

Table 4.2: Machine Configurations for the Distributed Computing Test on the Cloud with Different Numbers of Workers from the Same Data Center

The experiments in this section are the same as the previous except where the workers are located. In contrast, all the workers are in the same region in San Francisco in the United States. The configurations of the machines are shown in Table 4.2. The average completion times of this experiment with different numbers of workers are shown in Figure 4.3. Similar

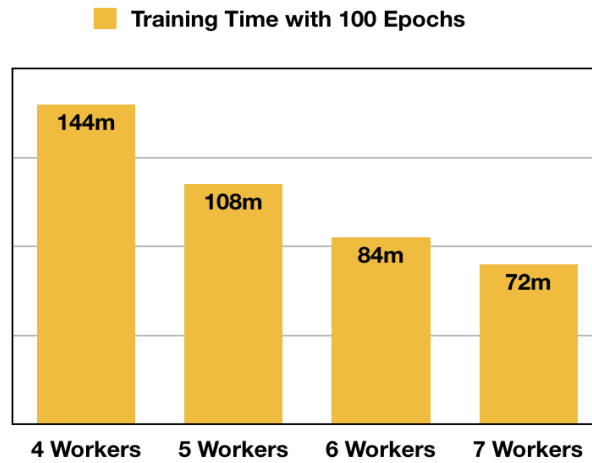


Figure 4.3: Training Time Result of the Distributed Computing on the Cloud with Different Numbers of Workers from the Same Data Center

to the previous experiment, we can see that a larger number of machines can improve the performance of distributed computing applications.

4.4.3 Comparison between the Distributed Computing on the Cloud with the Different Data Centers and with the Same Data Center

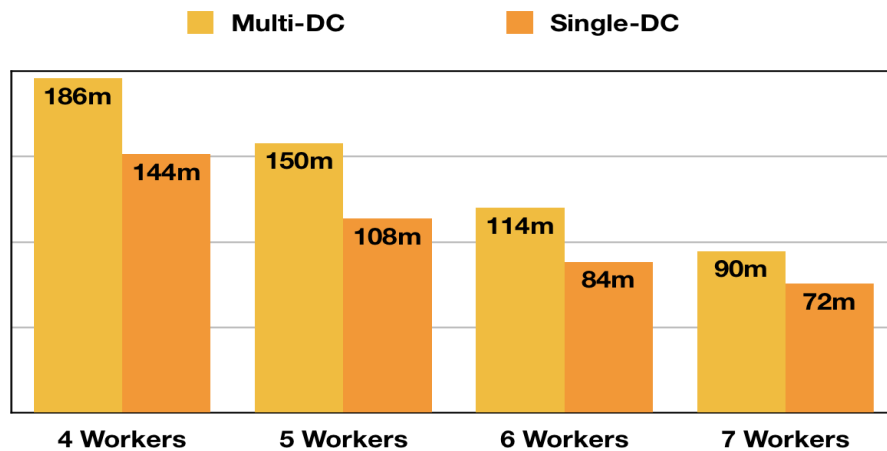


Figure 4.4: Training Time Result of the Comparison between the Distributed Computing on the Cloud with the Different Data Centers and with the Same Data Center

The previous two experiments clearly show the different performance between when the machines are located in different data centers and in the same data center, and it is presented

in Figure 4.4. The main variable I made in these experiments is the geographical location of the data centers where the Digital Ocean cloud service lends the worker machines. When the data center of workers are located in multiple cities and far away from each other, the training completion time is longer than the training time with the machines from a single data center. This shows that DeepMarket will be able to improve the performance of the distributed computing by connecting and recommending closer users' resources together into a cluster.

4.5 Distributed Computing in a Local Cluster on LAN Wi-Fi Connection

4.5.1 Distributed Computing in a Local Cluster on a LAN Wi-Fi Connection with Different Sizes of RAM

	Operating System	RAM	CPU
Machine 1	Ubuntu 16.04	4GB	4 cores
Machine 2	Ubuntu 16.04	4GB	4 cores
Machine 3	Ubuntu 16.04	8GB	4 cores

Table 4.3: Machine Configurations for the Distributed Computing Test in a Local Cluster on a LAN Wi-Fi Connection with Different Numbers of Workers

The experiment in this second section was set up in our lab at PSU. The configurations of the physical machines are shown in Table 4.3. I used three Ubuntu machines in the lab, and all laptops were connected to the same Wi-Fi network. Each machine lent only 1GB RAM and 1 CPU core to a cluster. Similar to the previous experiments, the MNIST example's training completion time was measured with 5, 10, and 100 epochs. This experiment was conducted to see how the number of training iteration or the workload of the job will affect the performance. Also, this experiment was repeated with a larger size of RAM, 2GB, to verify if adding only a fraction of a resource (*e.g.*, 1 CPU core out of 4 CPU cores) to a cluster could improve the performance of DeepMarket. The result is shown in Table 4.4.

It shows that, as the number of the epoch increase, the training completion time linearly increases. Also, we can see that adding 1 GB of RAM to a cluster also improved the performance of training time. The training completion time was shortened by 80 minutes when it was with 100 epochs by adding 1 GB of RAM. The marginal gain in performance explicates the benefit in fractional resource lending/borrowing, and it also confirms that DeepMarket can successfully leverage even a fraction of resource.

Number of Epochs	Training Time with 1 GB RAM	Training Time with 2 GB RAM
5	15 min	14 min
10	32 min	28 min
100	360 min	280 min

Table 4.4: Training Time Result of the Distributed Computing in a Local Cluster on a LAN Wi-Fi Connection with Different Sizes of RAM

4.5.2 Comparison between the Distributed Computing in a Local Cluster on LAN Wi-Fi Connection and in a Cluster on the Cloud

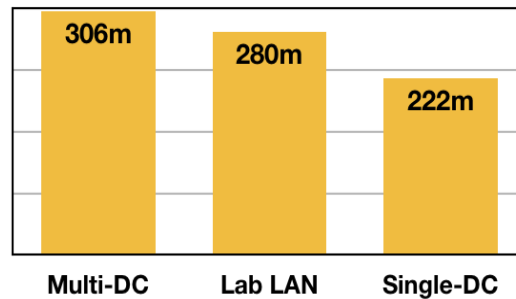


Figure 4.5: Training Time Result of the Comparison between the Distributed Computing in a Local Cluster on a LAN Wi-Fi Connection and in a Cluster on the Cloud with the Same and Different Data Centers

This section compares all of the averages job completion time results from the previous experiment setups as shown in Figure 4.5. All of these experiments were conducted with 3 worker machines, and each machine lent only 1 CPU core and 2 GB RAM to a cluster across

three schemes; “Multi-DC”, “Single-DC”, and “Lab LAN”. The “Multi-DC” result with different data centers has the worst performance, and the “Single-DC” result with a single data center has the best performance. It was unexpected that the “Lab LAN” result from our PSU lab with wireless LAN Wi-Fi connection didn’t outperform the “Single-DC” result with a single data center. However, this still shows that “Lab LAN” experiment’s performance was comparable to the performance with data center based architectures because it outperformed the “Multi-DC” result. Also, since these results didn’t include the time to transfer the input source file and retrieve the output result, it can’t conclude that cloud computing with a single data center can outperform distributed computing on the edge. As proven in [1] and [28], distributed computing at the edge reduces the network latency between user and data center since it offloads its computation at the edge, so it can have better performance than cloud computing services.

4.6 Experiments with Competing Network Traffic

	Spark	HDFS	IP Address	OS	RAM	CPU
Machine 1	Master/Worker	Namenode	192.168.1.26	Ubuntu 16.04	14.5 GB	8 cores
Machine 2	Worker	Datanode	192.168.1.29	Ubuntu 16.04	2.7 GB	4 cores
Machine 3	Worker	Datanode	192.168.1.30	Ubuntu 18.04	2.7 GB	4 cores

Table 4.5: Machine Configurations for the Distributed Computing Experiments with Competing Network Traffic

In this section, I wanted to verify the impact of competing traffic on DeepMarket performance. Distributed computing incurs message passing among worker nodes, which may happen through wireless communication in an edge environment, so I wanted to see if the performance on DeepMarket could be interfered with by other competing wireless network traffic. For example, competing network traffic can be generated and cause some interference on the performance of DeepMarket when a worker machine is also downloading YouTube

videos at the same time as computing distributed programs. Also, it can happen when a separate non-worker machine is using the same wireless LAN Wi-Fi connectivity with worker machines and generates heavy competing traffic. For the following two subsections, I conducted the same test measuring the MNIST training completion time as the previous experiments, but I also generated other competing network traffic at the same time by turning on a live streaming radio program or a YouTube video. The configurations of the machines are shown in Table 4.5. For this section, I used 3 worker machines, each lending only 1 CPU core and 2GB of RAM, to execute the MNIST example program. I operated all of the tests in this section with the same wireless Wi-Fi LAN connectivity in my apartment, and all of three physical worker machines were located in my apartment as well.

4.6.1 Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio

	Test 1	Test 2	Test 3	Test 4	Test 5	Average of Results
Without Any Radio Traffic on Workers	21 min (2 Timeout)	21 min (2 Timeout)	10 min (1 Timeout)	11 min (1 Timeout)	10 min (1 Timeout)	14.6 min (1.4 Timeout)
Live Radio Traffic on a Separate Machine	10 min (1 Timeout)	11 min (1 Timeout)	31 min (3 Timeout)	11 min (1 Timeout)	11 min (1 Timeout)	14.8 min (1.4 Timeout)
With Live Radio Traffic on a Worker/Master	1.2 min	21 min (2 Timeout)	11 min (1 Timeout)	31 min (3 Timeout)	21 min (2 Timeout)	17 min (1.6 Timeout)
With Live Radio Traffic on 2 Workers	1.6 min	11 min (1 Timeout)	21 min (2 Timeout)	41 min (4 Timeout)	31 min (3 Timeout)	21.1 min (2 Timeout)
With Live Radio Traffic on 3 Workers	11 min (1 Timeout)	11 min (1 Timeout)	41 min (4 Timeout)	11 min (1 Timeout)	31 min (3 Timeout)	21 min (2 Timeout)

Table 4.6: Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio

The experiment I conducted in this section was with live streaming radio traffic. I used an online website streaming real-time live radio programs in Portland, and I turned on a radio program on a separate machine or on worker machines during the experiments. In Table 4.6, the results are shown with the detailed information. First, I turned on a radio program on a separate machine to generate radio streaming traffic, separately from the master and worker

machines. The results are shown in the second row of Table 4.6. After that, I executed the same test generating radio traffic on worker machines. The third row shows the job completion time when generating the radio traffic on only a worker/master machine. The fourth and last rows show the tests results having radio traffic on two and all of the three worker machines.

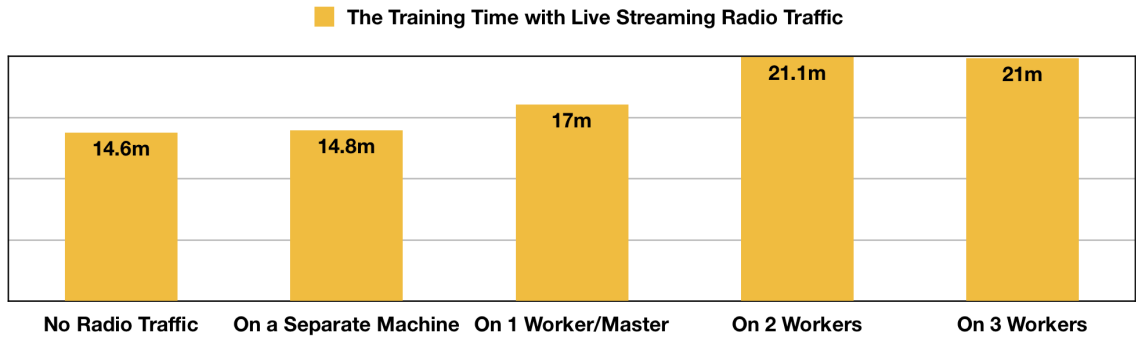


Figure 4.6: Training Time Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio

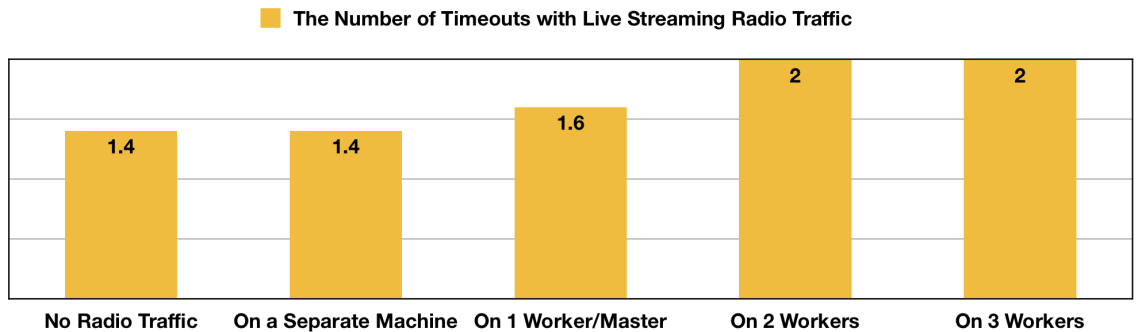


Figure 4.7: Timeout Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from Live Streaming Radio

Also, I calculated the average out of the 5 test results for each experiment to create the graphs in Figures 4.6 and 4.7. Figure 4.6 shows the average of the training time results, and it presents that the network traffic on a separate machine almost doesn't cause any interference to the performance of computation since it has nearly the same result as the average training time when there was no competing traffic. However, generating radio network traffic on

worker machines made a slight influence on the training completion time. It increased as the number of workers having live radio traffic increased. There is another graph with the average timeout results in Figure 4.7. As expected, the number of timeouts increased as the number of workers with radio network traffic increased. Timeout is a warning provided by TensorFlowOnSpark. According to TensorFlowOnSpark, it happens when it delays more than 10 minutes while feeding partition due to a variety of reasons. In other words, when a timeout occurred, this meant it took more than 10 minutes to process RDD (resilient distributed dataset) partition to other nodes via the executor's multiprocessing manager. It is just a warning, and there was no problem or error to finish training the program. When it happens the executor's multiprocessing manager tries feeding the part of partition with another node again and finishes the partitioned task. Users can manually change the duration of timeout to longer than 10 minutes if needed.

4.6.2 Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video

	Test 1	Test 2	Test 3	Test 4	Test 5	Average of Results
Without Any Youtube Video Traffic on Workers	21 min (2 Timeout)	21 min (2 Timeout)	10 min (1 Timeout)	11 min (1 Timeout)	10 min (1 Timeout)	14.6 min (1.4 Timeout)
Youtube Video Traffic on a Separate Machine	1.8 min	31 min (3 Timeout)	11 min (1 Timeout)	31 min (3 Timeout)	11 min (1 Timeout)	17.2 min (1.6 Timeout)
Youtube Video Traffic on a Worker/Master	31 min (3 Timeout)	21 min (2 Timeout)	11 min (1 Timeout)	11 min (1 Timeout)	11 min (1 Timeout)	17 min (1.6 Timeout)
Youtube Video Traffic on 2 Workers	11 min (1 Timeout)	21 min (2 Timeout)	31 min (3 Timeout)	11 min (1 Timeout)	21 min (2 Timeout)	19 min (1.8 Timeout)
Youtube Video Traffic on 3 Workers	31 min (3 Timeout)	21 min (2 Timeout)	11 min (1 Timeout)	21 min (2 Timeout)	11 min (1 Timeout)	19 min (1.8 Timeout)

Table 4.7: Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video

In this section, I conducted the same experiments as the previous section, but with generating YouTube video traffic instead of streaming live radio traffic on worker/master machines or a separate machine. Table 4.7 shows the results. Similar to the radio experiment results, the

training time and the number of timeouts increased as the number of machines with Youtube video traffic increases. The average results are presented in Figures 4.8 and 4.9. These Figures show a slightly different trend when YouTube traffic was generated on a separate machine compared to the previous experiments. As shown in Figure 4.8, the training time with Youtube traffic on a separate machine took longer than the training time without any traffic. Also, the result generated with YouTube video traffic on a separate machine is nearly the same as the result generated with YouTube video traffic on a worker/master machine. Except this, the overall trend of increasing training time as the volume of traffic increases was the same.

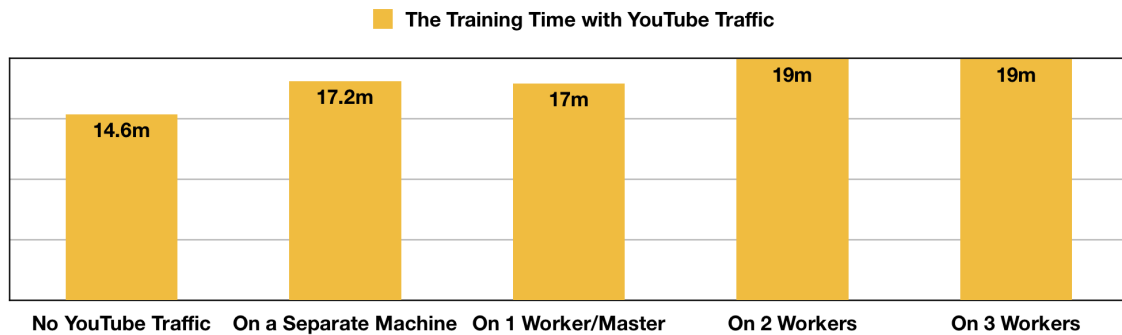


Figure 4.8: Training Time Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video

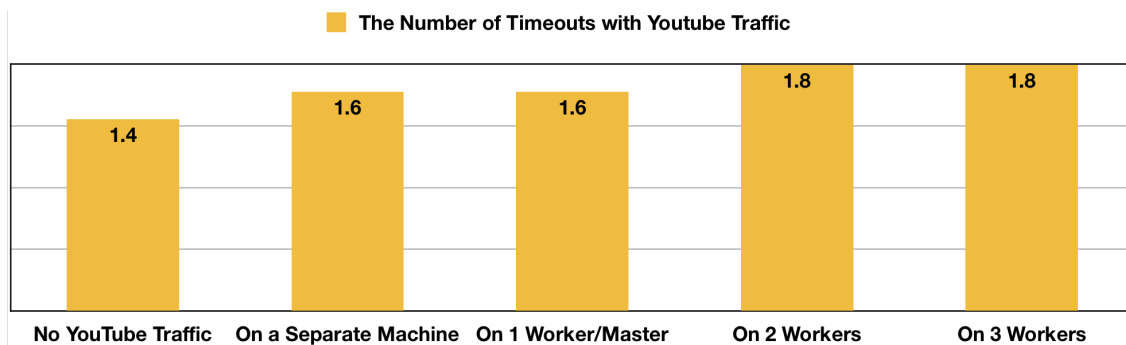


Figure 4.9: Timeout Result of the Distributed Computing Test on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video

4.6.3 Comparison between the Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video and from Live Streaming Radio

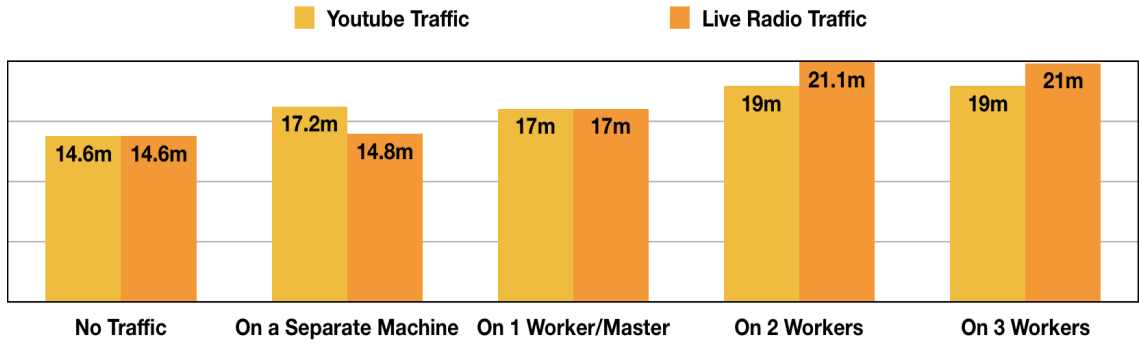


Figure 4.10: Training Time Result of the Comparison between the Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video and from Live Streaming Radio

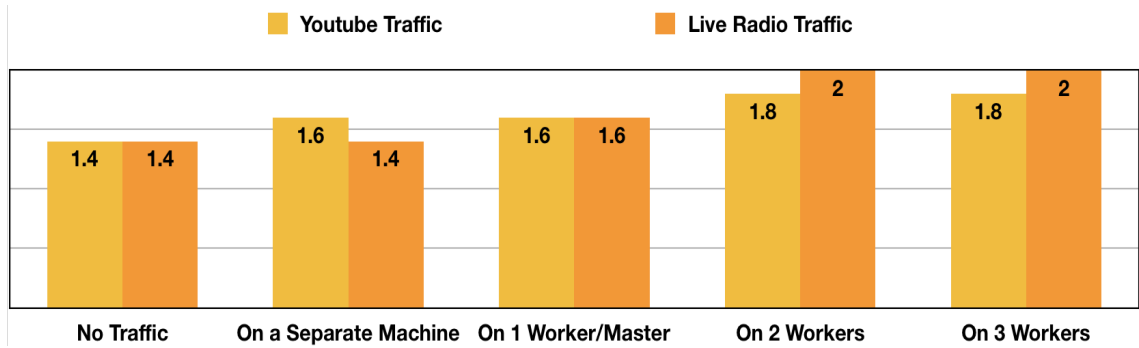


Figure 4.11: Timeout Result of the Comparison between the Distributed Computing on a LAN Wi-Fi Connection with Competing Network Traffic from YouTube Video and from Live Streaming Radio

To summarize, I compared the previous results with live streaming radio traffic and YouTube video traffic to see how different impact they would have on the performance of DeepMarket. The comparison results of the training time and the number of timeouts with these two kinds of competing traffic are shown in Figures 4.10 and 4.11. The major trend is the same. The training time and the number of timeouts increase as the number of machines with the traffic increases. Specifically, Figure 4.10 shows a 37% average increase in job completion

time when there is competing traffic in 2 or 3 machines. Also, Figure 4.10 presents that the training time with the live streaming live radio traffic took longer than the time taken with the YouTube video traffic. It indicates that live radio traffic was more influential on the performance of the computation than YouTube network traffic. Overall, it is shown that generating any of these network traffic on worker machines increased the training time and the number of timeouts.

4.7 Distributed Computing on a LAN Wi-Fi Connection with the Interference from Electrical Appliances

In this section, I wanted to verify the impact of uncontrolled traffic in a wireless edge environment on DeepMarket's performance. Since most of users and resources in DeepMarket will be connected with wireless Wi-Fi network connectivity, there are possibilities that electrical appliances around them such as microwave ovens, wireless telephones, and microphones could cause interference with DeepMarket's job execution [29]. One of the most common examples of this uncontrolled traffic in a wireless edge environment is the interference from a microwave oven, so I conducted the same experiment with the same setup from the previous section, but operating a microwave oven during the job execution time. In theory, when a wireless Wi-Fi operates with 2.4GHz frequency, a microwave oven can cause interference on the Internet connection since microwave ovens also operate on the same range of frequency [30].

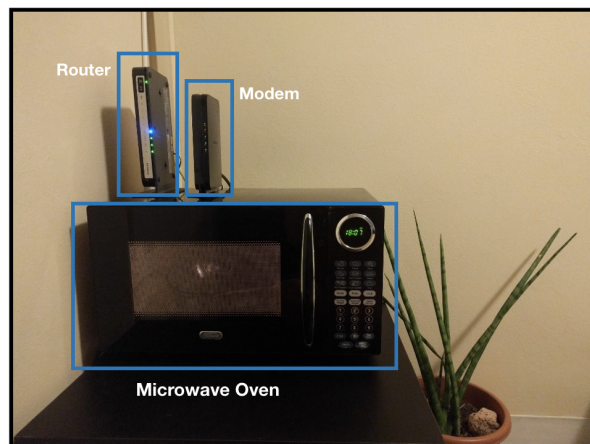


Figure 4.12: Picture of Microwave Oven, Modem, and Router Used in the Test with the Interference from Electrical Appliances

	Specifications of the Microwave Oven
Power Consumption	120V, 60Hz, 1350W (Microwave)
Output	900W
Operation Frequency	2450MHz
Outside Dimensions (H*W*D)	11 1/16 * 19 * 15 1/2 inches
Oven Cavity Dimensions (H*W*D)	8 11/16 * 13 3/8 * 12 5/8
Oven Capacity	0.9. cu. ft.
Cooking Uniformity	Turntable System
Net Weight	Approx. 30 lbs

Table 4.8: Specifications of the Microwave Oven

Figure 4.12 shows the picture of the set up I had in my apartment for this experiment. I put my personal modem and router on the microwave oven to make it generate some network interference with the wireless Wi-Fi Internet connectivity. During this experiment, I placed a microwave-safe container with water inside of the microwave oven and operated it during the experiments' execution time. Also, I made sure that the frequency of my Wi-Fi channel is 2.4 GHz and checked the specification of the microwave oven as shown in 4.8. The microwave oven has 2.45 GHz frequency, so the Wi-Fi connectivity and this microwave

oven operated in the same range of frequency so they could have a chance to interfere with each other. The results of this experiment are shown in Table 4.9 with the previous experiments' results for the comparison. Figures 4.13 and 4.14 visualized the comparison between these and previous experiments' results.

	Test 1	Test 2	Test 3	Test 4	Test 5	Average of Results
Without Any Completing Traffic	2.3 min	1.2 min	11 min (1 Timeout)	11 min (1 Timeout)	21 min (2 Timeout)	9.3 min (0.8 Timeout)
With Live Streaming Radio on a Separate Machine	1.2 min	10 min (1 Timeout)	11 min (1 Timeout)	31 min (3 Timeout)	11 min (1 Timeout)	12.84 min (1.2 Timeout)
With Microwave Oven Traffic	1.2 min	11 min (1 Timeout)	31 min (3 Timeout)	1.6 min	2.8 min	9.52 min (0.8 Timeout)

Table 4.9: Comparison between the Distributed Computing with Microwave Interference and with Live Streaming Radio Traffic on a LAN Wi-Fi Connection

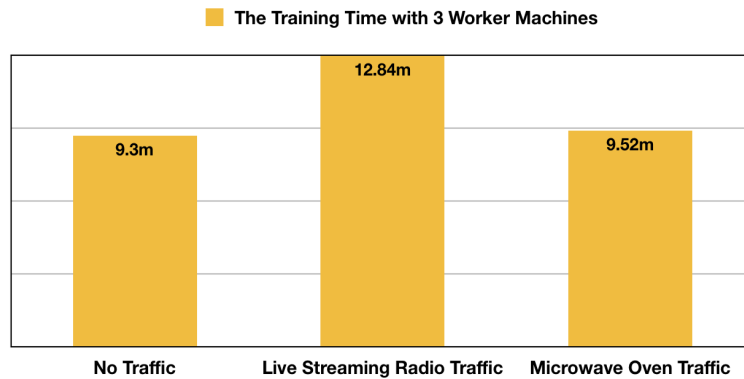


Figure 4.13: Training Time Result of the Comparison between the Distributed Computing with Microwave Interference and with Live Streaming Radio Traffic on a LAN Wi-Fi Connection

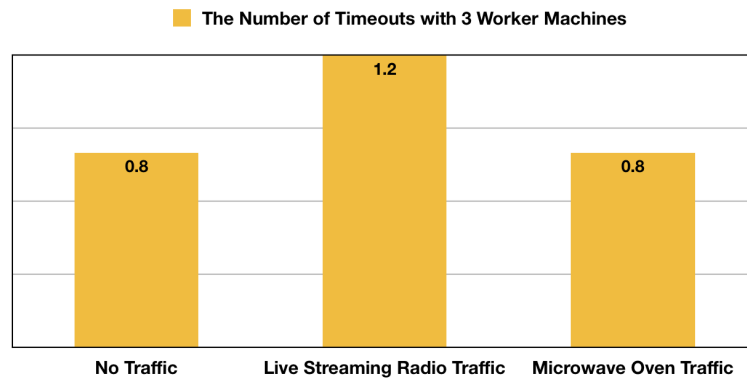


Figure 4.14: Timeout Result of the Comparison between the Distributed Computing with Microwave Interference and with Live Streaming Radio Traffic on a LAN Wi-Fi Connection

Figure 4.13 shows the average training time of the experiments without any competing traffic, with live radio traffic, and with operating a microwave oven near the Wi-Fi router. Figure 4.14 shows the average number of timeouts of the three different experiments. As shown in these two Figures 4.13 and 4.14, microwave oven interference only marginally increased the job completion time on average. The average training time with the microwave oven operation was slightly longer than the average training time without any competing traffic. I expected that this experiment would have similar results as the experiment with live radio traffic, but this experiment shows that a microwave oven only slightly interferes with the Wi-Fi connection. Also, it shows that since a microwave oven or electrical appliances generate sporadic interference, it has a lower interference power.

Chapter 5

Related Work

5.1 Advantages of Edge Computing over Cloud Computing

There has been a lot of research going on regarding the comparison of edge computing and cloud-centric computing. Cloud computing is well known to be used for distributed computing rather than other computing methods since it is a specialized distributed computing paradigm with cost reduction, dynamical scalability, managed computing power, storage platforms and other services [31]. However, even though cloud computing can provide more cost reduction compared to the past, researchers are seeking a cheaper way to compute a big amount of data. As an alternative to this cloud service, edge computing is getting attention to be used for distributed computing, and DeepMarket is also suggesting to use distributed computing on the edge decentralizing the data center. Therefore, in this section, I want to look at the research works related to the advantages of edge computing over cloud computing.

5.1.1 The Cost of Cloud Computing

The main reason that edge computing is getting more and more attention is that the cost to use cloud computing services has become an important issue for many student researchers. Cloud computing services are not easily accessible because of the cost, so some individuals

have to use their private resources or public resources from large companies to compute vast amounts of data. In addition, cloud computing has to maintain the data center, which can cause extra cost with low utilization by its resource standing and fragmentation [28]. This data center also causes server, infrastructure, power requirements, and networking expense [28].

5.1.2 Why Edge Computing?

As shown in [1], cloud computing causes a high cost especially in network latency, maintenance, and energy consumption. As a solution to this problem, several researchers showed that edge computing is able to improve the performance and expense of cloud computing while being more economical. Edge computing allows computation at the edge network which has computation resources to process data. This offloads parts of the data workload from the cloud [1], so it can improve the performance of computation with less network traffic as well as having a smaller risk of a data bottleneck. In addition, as shown in [32], the management and processing cost by cloud computing can be improved with edge computing with 95% of edge data reduction. Specifically, it has shown that 3 years of management and processing cost can be decreased from \$80,531.00 to \$28,927.00 [32].

5.1.3 Network Latency

Deep learning distributed programs usually have a large amount of data to process, and the most common processing method is by using Apache Hadoop and Spark, which need to use part of a computing resource to manage software, subsequently losing its performance [33]. Therefore, it's a given result that cloud computing, with the data center having all the computation and storage resource on the cloud, has a higher network latency compared to the edge computing [33].

5.1.4 Bandwidth Requirements

As stated in [28], the main keys to reducing the cost of cloud computing is networking and infrastructure since they cause at least 40% of the cost from its data center. Also, if we take another look at [1], the response time for facial recognition applications was reduced from 900 ms to 168 ms by computing it at the edge instead of at the cloud. In addition, as proven in [1], offloading computing tasks from the cloud can reduce 30-40% of energy consumption. This is mainly due to the advantages of edge computing in terms of lower bandwidth requirements and less complexity of managing and processing data [34].

5.1.5 Privacy and Security

In this day and age, user privacy is more important than it ever has been and so is the importance of usability. People want their data to be safe and secure. With the numerous data leaks throughout the past few years, the population has become understandably cautious in this regard. On cloud computing, users do not know about where their data stored and if there are any privileged users who can release their information [31]. In contrast, edge computing is safe in terms of data privacy since the data does not get stored in a centralized service or storage [1]. Also, cloud computing is mainly in a web form and usually over SSL to create and manage users' jobs and other information, so several security issues can occur through communications by emails and authentication processes [31]. Therefore, edge computing on private resources is more secure in this term [31]. However, edge computing itself cannot replace cloud computing because it still has security and configuration challenges as stated in [4]. Overall, cloud computing itself is no longer the best platform to be used for distributed computing, and there are needs for a cheaper, safer, and easily accessible way to compute distributed programs.

5.2 Existing Infrastructure and Their Difference with DeepMarket

There are existing infrastructures/testbeds to provide low-cost access to massive computing resources. I introduce these testbeds in this section and compare them against DeepMarket.

DeepMarket has similar objectives to these testbeds but specifically aims to have four key properties. First, it will support access to large amounts of resources to execute a massive scale of computations. Second, machine learning abstraction will be built in DeepMarket, so low-level system configurations are not needed. Third, DeepMarket will support heterogeneous resources at the edge with both wired and wireless networks. Lastly, its pricing mechanism will allow researchers low-cost access to resources. To study and figure out which other existing infrastructures were available, I evaluated them with these properties in Table 5.1. It shows the existing testbeds/infrastructures, and they are evaluated by DeepMarket’s four key properties. A “*” indicates that a testbed satisfies the property, and a “+” indicates partial satisfaction. A “-” means the testbeds or infrastructures did not satisfy the property at all.

The testbeds in the first row in Table 5.1 are distributed systems and networking testbeds. As shown in the table, these testbeds can allow users to access computing and network resources on the clusters. Emulab [35], GENI [36], and OneLab [37] are able to join into a cluster for distributed edge computing by wireless network connection, but they don’t provide supports of machine learning abstractions and pricing capability, so they are designed for only short term access to the resources at the edge. The second row shows Akraino [38] and Steel [39]. Akraino and Steel can be operated at the edge resources, but they don’t provide access to large amounts of resources. In addition, on the fourth row, BOINC [40] supports a large scale of resources and pricing capabilities, but they are not heterogeneous edge resources and networks. The last row includes TensorFlow, Apache Spark [14], PyTorch [41], and OpenAI[42] which are machine learning libraries, and they can be integrated into DeepMarket.

Properties		Large Amounts of Resources	Machine Learning Abstraction	Heterogeneous Edge Resources	Pricing Capabilities
Testbeds	EmuLab, GENI, OneLab	+	–	+	–
	Akraino, Steel	–	–	*	–
	Cloud testbeds	*	–	+	–
	BOINC	*	–	–	+
	Machine Learning Libraries	–	*	–	–

* : **Satisfaction** + : **Partial Satisfaction** – : **No Satisfaction**

Table 5.1: Existing Testbeds and Four Key Properties of DeepMarket

As shown, it is not common to find a testbed to satisfy more than two properties among the DeepMarket’s four key properties as shown in Table 5.1. To meet these properties, DeepMarket should be researched/updated with the features of existing testbeds and integrated with other infrastructures if needed.

Chapter 6

Discussion

In this section, I discuss the pros and cons of DeepMarket over third-party cloud providers regarding cost, performance (*e.g.*, total job execution time), privacy and security.

Resiliency: The massively distributed nature of DeepMarket makes the architecture resilient in case of severe outages. Data centers aggregate computational resources at centralized locations which increases their susceptibility to outages (*e.g.*, an attack that can wipe out the infrastructure). Thus, to increase resiliency, cloud providers would need to construct backup centers, which increases the cost.

Cost: With a smart pricing algorithm, DeepMarket can optimize its client-consumer ratio leading to a reduced cost for all consumers. The pricing mechanism could incentivize users to share their resources to get credits to run their own programs using others' additional resources, so the price of running a program could possibly be for free. Further, unlike cloud providers, DeepMarket would not incur any extra cost to maintain computational and storage resources such as cooling, server, personnel, and energy consumption cost, among others. Several studies have shown that these costs constitute a large portion of total cost in operating data centers [28].

Latency: As DeepMarket's network of resources scales, the latency of communication between resources and users (*i.e.*, users who want to run jobs) reduces. This is because DeepMarket would be able to match resources and jobs based on proximity, which reduces

the time it takes for a user to send his/her data to the computational resource and retrieve the results. Further, the overall traffic towards the network reduces, which reduces the risk of facing/creating data bottlenecks [1].

Privacy: Data encryption is supported by cloud providers. However, a user's data is handled by only a single operator. DeepMarket can spread users' data across machines owned by different lenders, which can increase data privacy.

Reliability: A key benefit of cloud providers is their system reliability, as they can guarantee the resource operation for the time that is desired by any user. In DeepMarket, it is possible for a user to unexpectedly terminate the resource operation (*e.g.*, forcibly shut down the machine). This reduces system reliability. One way to address the issue is to build a scoring system (*e.g.*, [43]) that ranks the reliability of resources and their owners. DeepMarket can then use resources from users with a better ranking. Additionally, DeepMarket can build redundancy when using resources to counter unexpected job terminations.

Chapter 7

Conclusion

In this thesis, I presented the design and implementation of DeepMarket, an edge computing marketplace that allows users to lend or borrow computational resources and run distributed ML programs. In addition, I introduced DeepMarket's platform with its software and hardware infrastructure. I discussed the design of PLUTO, a GUI that simultaneously allows a user to lend and borrow computational resources. I also presented some of the key aspects of DeepMarket's backend services. In chapter 4, I verified that adding only a fraction of a resource (*e.g.*, 1 out of 4 CPU cores) to a cluster could improve the performance of DeepMarket, and I characterized the improvement in DeepMarket's performance as more workers are added to the cluster. This showed that it becomes feasible for users to borrow many resources simultaneously, significantly decreasing the job completion time, as DeepMarket's adoption increases. Also, I showed through other experiments that renting resources on DeepMarket has a similar job completion time to renting resources on the cloud providers. This means that DeepMarket can further reduce job completion time when taking into account the time it takes to upload data and retrieve the results. However, as DeepMarket scales, it can match jobs to resources that are closer to users. This can significantly reduce the overall job completion time (*i.e.*, when also taking into account the time to submit data and retrieve the results). These benefits are amplified by the reduction in cost due to lower maintenance cost and the potential increase in user privacy as DeepMarket spreads users'

data across resources owned by different lenders. Furthermore, the last two experiments examined the reliability of DeepMarket with possible dynamic competing network traffic. Finally, in chapters 5 and 6, I compared the DeepMarket against the related work and third party cloud service providers.

Chapter 8

Future Work

DeepMarket is currently available in machines with Ubuntu operating systems and requires installation of Apache Spark. The next version of DeepMarket will support more heterogeneous computational resources with Docker [44]. Docker containers can be operated on any operating systems and manage the configurations, so users will not have to download and configure Spark and other software such as HDFS, Python, TensorFlow, and TensorFlowOnSpark. This also will improve DeepMarket's security because it will limit computations to the container environment instead of the physical resource. In addition, DeepMarket should have a better smart pricing mechanism. To provide a shared marketplace, the pricing algorithm is important to users and should be made with all the aspects of a user's need. It requires a lot of research on how to gather and evaluate the demand and supply of computational resources and how we will apply the data to generate pricing for different times and configurations of resources.

Bibliography

- [1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: vision and challenges. *IEEE Internet of Things Journal*, 2016.
- [2] MNIST database. https://en.wikipedia.org/wiki/MNIST_database.
- [3] Amazon Web Services (AWS). <https://aws.amazon.com/>.
- [4] R. J. Dudley and N. Duchene. *Microsoft Azure: Enterprise Application Development*. 2010.
- [5] Digital Ocean Cloud Computing. <https://www.digitalocean.com/>.
- [6] M. Lynley. Snark AI looks to help companies get on-demand access to idle GPUs. TechCrunch. <https://techcrunch.com/2018/07/25/snark-ai-looks-to-help-companies-get-on-demand-access-to-idle-gpus/>.
- [7] Kings Distributed Systems. <https://kingsds.network/>.
- [8] Golem Network. <https://golem.network/>.
- [9] SONM: Decentralized Fog Computing Platform. <https://sonm.com/>.
- [10] T. Zhu, Z. Huang, A. Sharma, J. Su, D. Irwin, A. Mishra, D. Menasche, and P. Shenoy. Sharing renewable energy in smart microgrids. In *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 219–228, 2013.

- [11] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. In *Technical Report HPL-2002-57, HP Labs*, 2002.
- [12] D. Liu and Z. Libin. The research and implementation of cloud computing platform based on docker. In *2014 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP). IEEE*, 2014.
- [13] B. Ikhwan, E. M. Goortani, M. B. A. Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe. Evaluation of docker as edge computing platform. In *Proceedings of IEEE Conference on Open Systems (ICOS) (pp. 130-135). IEEE.*, 2015.
- [14] Apache Spark. <https://spark.apache.org/>.
- [15] M. Zaharia, C. Mosharaf, M. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud 10.10-10 (2010)*, 2010.
- [16] R. Xin, P. Deyhim, A. Ghodsi, X. Meng, and M. Zaharia. Graysort on apache spark by databricks. *GraySort Competition*, 2014.
- [17] HDFS: Hadoop Distributed File System. <https://hadoop.apache.org/>.
- [18] J. Shafer, S. Rixner, and A. L. Cox. The hadoop distributed filesystem: Balancing portability and performance. In *Proceedings of 2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS). IEEE*, 2010.
- [19] D. Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website 11.2007 (2007): 21*, 2007.
- [20] TensorFlowOnSpark. <http://www.odpms.org/2017/05/tensorflowonspark/>.
- [21] Meet Horovod: Uber’s Open Source Distributed Deep Learning Framework for TensorFlow. Available: <https://eng.uber.com/horovod/>.

- [22] TensorFlowOnSpark GitHub Repository. <https://github.com/yahoo/TensorFlowOnSpark>.
- [23] Apache Spark's scalable machine learning library (MLlib). <https://spark.apache.org/mllib/>.
- [24] J. Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
- [25] M. Hibler, Ro. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, and K. Webb J. Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference, Boston, MA*, 2008.
- [26] K. Hightower, B. Burns, and J. Beda. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. 2017.
- [27] D. Kakadia. *Apache Mesos Essentials*. 2015.
- [28] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. In *ACM SIGCOMM computer communication review 39.1, (2008): 68-73*, 2008.
- [29] Wi-Fi RF Spectrum; interferences; how to detect them, case study: microwave oven. <https://www.acrylicwifi.com/en/blog/wi-fi-rf-spectrum-interferences-how-to-detect-them-case-study-microwave-oven/>.
- [30] Why Does Your Microwave Oven Mess With The Wi-Fi Connection? <https://io9.gizmodo.com/why-does-your-microwave-oven-mess-with-the-wi-fi-conne-1666117933>.
- [31] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Proceedings of IEEE Grid Computing Environments Workshop*, 2008.

- [32] The Vital Role of Edge Computing in the Internet of Things. <https://wikibon.com/the-vital-role-of-edge-computing-in-the-internet-of-things/>.
- [33] M. Femminella, M. Pergolesi, and G. Reali. Performance evaluation of edge cloud computing system for big data applications. In *5th IEEE International Conference on Cloud Networking (Cloudnet) (pp. 170-175). IEEE.*, 2016.
- [34] Edge Computing. <https://searchdatacenter.techtarget.com/definition/edge-computing>.
- [35] M. Hibler, Ro. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, and K. Webb J. Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference, Boston, MA*, 2008.
- [36] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. GENI: A federated testbed for innovative network experiments. In *Computer Networks*, 61, pp.5-23, 2014.
- [37] S. Fdida, F. Timur, and M. Sophia. Onelab: Developing future internet testbeds. In *European Conference on a Service-Based Internet. Springer, Berlin, Heidelberg*, 2010.
- [38] Akraino Edge Stack. The Linux Foundation. <https://www.lfedge.org/projects/akraino>.
- [39] S. A. Noghabi, J. Kolb, P. Bodik, and E. Cuervo. Steel: Simplified development and deployment of edge-cloud applications. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, 2018.
- [40] Compute for Science, 2019. BOINC. <https://boinc.berkeley.edu>.
- [41] N. Ketkar. *Introduction to pytorch*. 2017.
- [42] Resources, 2019. OpenAI. <https://openai.com/resources/>.

- [43] R. Zhou and K. Hwang. Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing. *IEEE Transactions on Parallel & Distributed Systems*, (4):460–473, 2007.
- [44] Docker Container System. <https://www.docker.com/>.