DeepMarket: An Edge Computing Marketplace with Distributed TensorFlow Execution Capability

Susham Yerabolu, Soyoung Kim, Samuel Gomena,
Xuanzhe Li, Rohan Patel, Shraddha Bhise, and Ehsan Aryafar
Computer Science Department, Portland State University, Portland, OR, 97205

***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 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/CPUs 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 paper, we present the design and implementation of DeepMarket, an architecture that addresses these challenges and allows users to securely lend and borrow computing resources. We 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.**

Keywords: Marketplace Design, Apache Spark, Edge Computing, Hadoop Distributed File System.

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. Purchasing machines outright, however, can require significant upfront investment that is not justified by the intermittent use that many researchers require. Renting resources from cloud providers like Amazon AWS [1] or Digital Ocean [2] is also expensive, since cloud providers need to construct and maintain cloud (*i.e.,* data center) infrastructure with significant operation and maintenance cost. Several recent works have attempted to intelligently exploit various types of cloud pricing to reduce their cost [3]–[5], but they are still fundamentally dependent on cloud providers' offering low-cost options.

One possible solution to reduce these costs is to introduce a *marketplace* for computing resources in which users can lend each other resources when they are idle, similar to Uber's or Airbnb's sharing platforms. Such a marketplace reduces upfront investment costs by allowing users to purchase fewer resources outright and supplement them with others' machines, and eliminates the need to pay a cloud provider. While a few recent startups [6]–[9] and academic works [10], [11] have proposed similar ideas, such a computing marketplace is particularly appropriate for training deep learning models, which can easily be done in a distributed manner. Algorithms for training these models can be easily adapted to run on heterogeneous resources, e.g., both CPU and GPU servers, thus allowing a wide range of users to participate in the computing marketplace. As edge computing paradigms, which aim to exploit computational resources at the edge of the network, become popular, such edge devices may also be incorporated into a computing marketplace.

In this paper, we present the design, implementation, and experimental evaluation of DeepMarket, an open-source application[1] that creates a computing marketplace for deep learning. Users that lend resources on DeepMarket receive credits for doing so, which can be used to rent resources from others in the future. In this way, researchers are incentivized to contribute their idle resources to DeepMarket; and DeepMarket automatically matches available resources to pending deep learning jobs, seamlessly executing the jobs over dispersed, heterogeneous machines. Our key contributions are as follows:

- **PLUTO GUI:** We created a graphical user interface, named PLUTO, which allows any user with Internet connectivity to create an account, lend his/her computational resources, borrow resources from others, and observe historical rental data and remaining balance. PLUTO allows a user to remotely lend his/her resources without the need to directly run PLUTO from them, provided that the authenticity of resource ownership can be verified by our backend servers.
- **Backend Servers:** We setup servers at Portland State University (PSU) that provide a variety of backend services. These services are composed of two key modules: (i) *a Services Module:* which is responsible for interfacing the users (*i.e.,* PLUTO GUIs), dynamic price generation, and updating jobs' status; and (ii) *an Executor Module:* which builds on top of Apache Spark [13] and Hadoop Distributed File System (HDFS [14]) for matching jobs to resources and executing them.

[1]For the latest project updates, please refer to [12].

- **Experimental Evaluation:** We have conducted preliminary experimental evaluation to compare the performance of our marketplace against renting computational resources on Digital Ocean. We show that DeepMarket has a similar performance (in terms of job completion time) to renting resources on Digital Ocean data centers with the same system configuration parameters (*e.g.,* number of workers, RAM size and number of cores on each machine). However, DeepMarket can potentially achieve this with reduced cost, increased privacy/security, and reduced communication latency (*i.e.,* the latency to send data and retrieve the results).

The rest of this paper is organized as follows. We discuss the system architecture along with PLUTO and server modules in Section II. We present the results of our experiments in Section III. We discuss the pros and cons of DeepMarket over existing cloud based solutions in Section IV. Finally, we conclude the paper in Section V and discuss the future work in Section VI.

II. Architecture

DeepMarket's architecture consists of three main modules as shown in Fig. 1. The first module (PLUTO) resides on the user's machine and allows the user to *both* lend spare computational resources (and earn credit for that) as well as submit jobs (*i.e.,* borrow some available computing resources and run distributed machine learning algorithms on them).

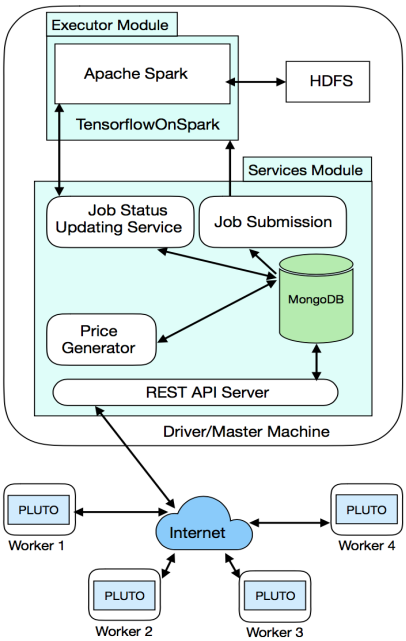

Fig. 1. **DeepMarket architecture has three main components: (i) the Pluto application, which allows a user to submit jobs, lend computing resources, or use resources from other people; (ii) a services module that is responsible for bookkeeping, billing, job status update, and dynamic price generation; and (iii) an executor module that uses HDFS and TensorFlowOnSpark for distributed data management, resource scheduling, and running the tasks across the resources.**

A user that borrows resources needs to pay for those through his/her available credit balance. The other two modules reside on servers that are managed by our team at the Networks and Wireless Systems Lab at Portland State University (PSU). The servers are responsible for many backend roles such as dynamic price generation, accounting, resource scheduling, job execution, and ensuring the security of services.

In this section, we first discuss PLUTO (Section II-A). Next, we discuss two of the most important modules on the server (master) machines: how job submission and resource additions are handled from the server side (Section II-B) and how submitted jobs are executed (Section II-C).

A. PLUTO

PLUTO is a simple and intuitive graphical user interface (GUI) developed using PyQt5 [15], which allows a user to create an account on our databases. It also provides the users of our marketplace with the capability to either lend their spare computational resources or submit jobs for execution on borrowed resources. Users can remotely lend their spare computational resources without having to login to each machine, run PLUTO, and then lend the resource. PLUTO GUI has three key tabs:

Dashboard Tab: Users can see an overview of their jobs' status (*e.g.,* running, finished, panic), their resources' status (*e.g.,* if a resource is currently being utilized by the system), the number of recent credits earned and spent, and the total remaining balance. Each newly registered user gets 50 credits by default in order to experiment with the application.

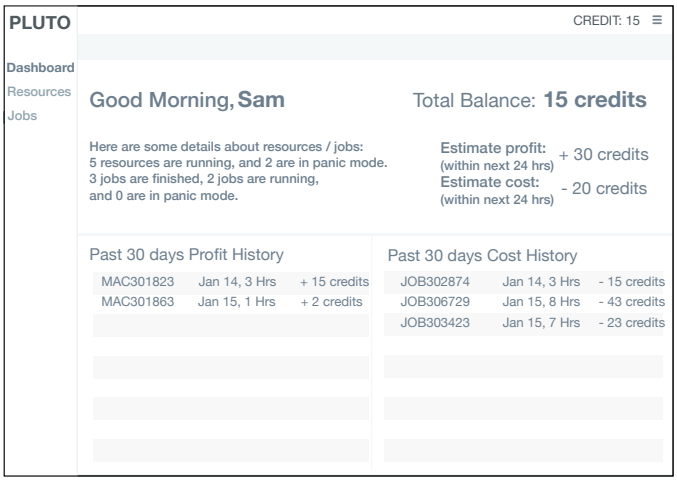

Fig. 2. **The "PLUTO Dashboard Tab" provides an overview of the jobs' status (running, finished, panic), resources' status, total balance, and a history of credits earned or spent. The panic modes includes jobs that are aborted or resources which have lost connectivity to the server. PLUTO allows a user to remotely lend multiple computing resources. In the example above, an overview of credits earned by the user across all of his machines, is shown on the dashboard.**

Resources Tab: Users can lend their current machines (on which PLUTO is installed) and add them to the pool of available computational resources at DeepMarket by providing the machine IP address and pressing the VERIFY button.

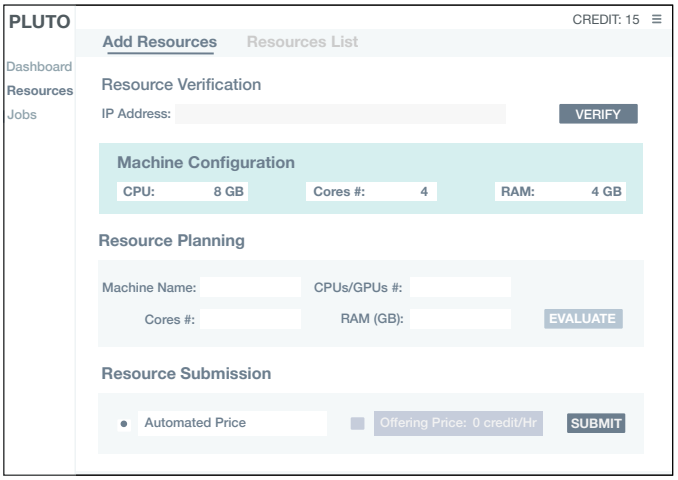

Fig. 3. **The "PLUTO Resource Management Tab" allows the user to remotely lend every machine on which the PLUTO application is installed. The user can specify the desired configuration parameters (e.g., number of cores, RAM, and in the future time availability of the machine) before lending the machine. The ownership and configuration are verified before the machine is added to the pool of resources at DeepMarket.**

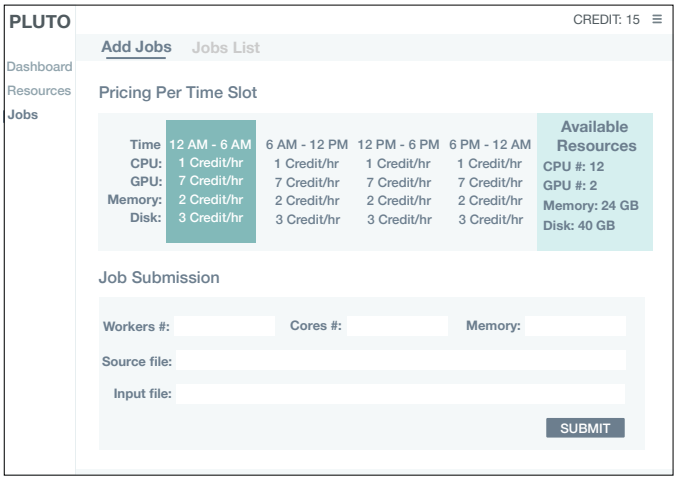

Fig. 4. **The "PLUTO Jobs Tab" is composed of two sub-tabs. In the first sub-tab (depicted here) the current price for a default configuration (across four 6-hour time slots) is shown to the user. The user can select the desired job sunning time. This sub-tab also allows the user to specify the desired number of cores, workers, and RAM to run the job on. The second sub-tab shows the status of the previously submitted jobs.**

By pressing the VERIFY, the machine configuration (CPU, number of cores, and RAM) would be displayed to the user. If a user intends to lend a remote machine (*i.e.,* a machine that is different from the current machine), the verification system asks for proper credentials to ensure resource ownership. Once the machine configuration parameters are shown to the user, the user can specify the fraction of resources which he/she is willing to lend, *e.g.,* a user may only lend half of the total cores or RAM, and use the remaining cores and RAM for local use. A user has to "EVALUATE" the specified values, so that PLUTO can ensure these values are legitimate (*i.e.,* are less than the machine configuration parameters). In our current implementation of DeepMarket, the credit that a user earns by lending his/her machines is automatically generated by our system. Our future updates to PLUTO would enable a user to specify a lending price. DeepMarket would then only use the resource if borrowers are willing to pay that price. Finally, through the "Resources List" sub-tab, a user can view the existing machines that are lent by the user and their status (*e.g.,* if they are currently being used by the system).

Jobs Tab: This tab provides several functionalities: (i) it displays the current price of running jobs (as announced by the server) per CPU/GPU/RAM/Disk unit (*e.g.,* 1 Credit/Hr for 1GB of RAM). The prices are denoted over four six-hour time slots, beginning at 12 AM at our server location. The variability of prices at different time slots allows users with less credits to schedule their jobs at cheaper times. (ii) The "Jobs Tab", also provides an interface for users to submit jobs and run distributed TensorFlow programs. In addition to the desired job running time, the user specifies the desired number of workers, cores and RAM per worker, and the "HDFS path" to the source files and input files (data files). These are all needed when running a ML job. Once a user submits the job, the total price of the job execution will be displayed, and the user can cancel the job execution before it is sent to the backend servers. Finally, a sub-tab of the "Jobs Tab" denotes the previous submitted jobs and their status (*e.g.,* completed, running, aborted).

B. Services Module

The services module resides on the server and interacts with machines that run PLUTO through Internet. This module is responsible for many bookkeeping applications (*e.g.,* users/resources/jobs database) as well as dynamic price generation, ensuring security of the market, and updating jobs' status (both internally and across PLUTO applications). We discuss the most important aspects.

REST API Server: We have developed a secure REST (Representational State Transfer) API (Application Programming Interface) service to save detailed information about users, job submissions, and resources. By conforming to the REST architectural style, we ensure interoperability across different computer systems on the Internet. This increases the performance, speed, and scalability of our architecture.

Price Generator: There are 4-time slots (each with a 6 hour period) to choose from when submitting a job through PLUTO's Jobs Tab. Each time slot will have a different price at which the use of resources will be billed. Currently, the price generator service runs at 12:00 AM every morning to generate random prices for computational resources (CPU, GPU, RAM, Disk) for the next 24 hours. We are currently working on a smart pricing algorithm to generate these prices for each time slot based on the previous jobs' execution times, availability of resources, and historical and/or predicted demand for resources. A user's job that spans multiple time slots is billed based on the usage and price of each time slot.

Job Submission Service: As users submit their jobs, the jobs and their configurations (*e.g.,* the selected time slots) get stored in a database. The job submission service keeps checking this database every 1 min for any new job submission

entry. It then submits the job, based on the selected time slot, to the Spark Driver. The Spark driver, co-located on the server, is part of the *Executor Module* and handles the job execution.

Job Status Update Service: Apache Spark has a web application UI/API to show the job execution status and worker status. Our "Job Status Update Service" utilizes the spark API to update the submitted job status and the host machine status in the corresponding databases. These updates are also reflected on the PLUTO applications.

C. Executor Module

Similar to the services module, the executor module resides on the server and is responsible for data file management, selection (scheduling) of resources, and execution of distributed TensorFlow programs. We use a combination of Apache Spark, TensorFlowOnSpark, and HDFS to achieve these goals.

Hadoop Distributed File System (HDFS): Since training of deep learning or machine learning models would require large amounts of data, it is necessary that this data is available to each worker machine in a reliable and fault tolerant manner. We use HDFS to realize these needs. HDFS is a distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware and provides high throughput access to application data and is suitable for applications that have large data sets [15].

Apache Spark: There are several frameworks and libraries like Horovord [16], MxNet [17], Apache Spark MLib [18], TensorFlowOnSpark [19] to efficiently utilize the GPUs/CPUs and train Machine Learning models faster through parallelism. We have chosen Apache Spark along with TensorFlowOnSpark as the underlying framework for DeepMarket since the computational resources utilized to train the distributed TensorFlow models can be conventionally used laptops or desktops. Apache Spark can: (1) flexibly use different cluster managers like Hadoop YARN, Apache Mesos, and Kubernetes; (2) Provide Fault Tolerance as machines stop working for a variety of reasons (*e.g.,* when a user unintentionally shuts down the resource); (3) Handle Stragglers[2]; (4) Integrate with Hadoop Distributed File System.

TensorFlowOnSpark: By combining salient features from the TensorFlow deep learning framework with Apache Spark and Apache Hadoop, TensorFlowOnSpark enables distributed deep learning on a cluster of GPU and CPU servers. TensorFlowOnSpark supports all types of TensorFlow programs, enabling both asynchronous and synchronous training and inferencing. It supports model parallelism and data parallelism as well as TensorFlow tools such as TensorBoard on Spark clusters [19].

III. Experimental Evaluation

We have conducted preliminary experiments to characterize the performance of our system. We conducted experiments leveraging the MNIST distributed TensorFlow program, which is the "Hello World" program of Machine Learning. The MNIST (Modified National Institute of Standards and Technology) database, is a large database of handwritten digits that is used for training various image processing systems [21]. It contains 60,000 digits ranging from 0 to 9 for training digit recognition systems, and another 10,000 digits as test data. Each digit is normalized and centered in a gray-level image with size 28x28 or with 784 pixel in total as the features. A few examples of MNIST dataset are shown in Fig. 5.

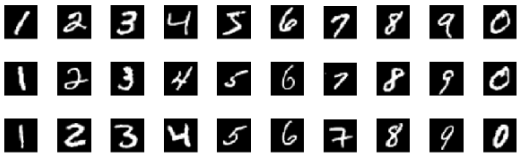

Fig. 5. MNIST Dataset Examples

Our key objective in our performance evaluation is to characterize how resources (*e.g.,* conventional laptops, virtual machines in a data center) and their locality affect the performance of DeepMarket. We chose "job Completion Time", *i.e.,* the time it takes to train a fixed model for a given number of epochs [3] as our key performance indicator. We observed similar training *accuracy results*, irrespective of type/ locality of resources, and hence we do not report those results.

We evaluated the performance of DeepMarket for three different setups of computational resources and backend servers: (i) **Lab LAN:** This is a LAN setup in our lab at PSU. Our resources are three conventional laptops and our server is a single Desktop Ubuntu machine, and they are all connected to the same WiFi network. Our architecture currently only supports Ubuntu machines (the limitation comes from our implementation of the "executor module"), which limited the number of workers that we could create in our lab. Fig. 6 shows the configuration of our lab resources (*i.e.,* laptops that serve as worker machines). (ii) **Multi-DC:** The server and resources are computers rented from Digital Ocean and the resources belong to different data centers. Fig. 7 shows the location of resources in this setup. (iii) **Single DC:** The server and resources are computers from digital Ocean, however, all machines are located in a single data center in San Francisco.

Before executing the experiments, we uploaded the data files and the source file of the MNIST distributed TensorFlowOnSpark program to the HDFS hosted on the server machine. In "Lab LAN" and "Single-DC" setups, the server is located on the same network and data center as worker machines, respectively. In "Multi-DC" setup, the server is located in San Francisco.

Experiment 1: We first evaluate the training completion time of our LAN testbed with physical laptops and desktop. We vary the number of epochs from 5 to 100 and conduct

[2]Stragglers: A machine that takes an unusually long time to complete one of the last tasks in the computation. Stragglers can arise for a variety of reasons. For example, a machine with a bad disk may experience frequent correctable errors that slow its read performance from 30 MB/s to 1 MB/s [20].

[3]One epoch is when an entire dataset is passed forward and backward through the neural network only once.

Name	CPUs	RAM	O.S Version
Machine1	4	4GB	Ubuntu16.0
Machine2	4	8GB	Ubuntu16.0
Machine3	4	4GB	Ubuntu16.0

Fig. 6. We used three Ubuntu laptops (each with 4 cores) to build our local resource cluster. All laptops are connected to the same WiFi network.

Machine Region	CPUs	RAM	Cost/Hr	No. of Machines	O.S Version
San Francisco	2	4GB	0.030/hr	3	Ubuntu 18.10
France	4	8GB	0.060/hr	1	Ubuntu 18.04
London	4	8GB	0.060/hr	1	Ubuntu 18.04
India	2	4GB	0.030/hr	1	Ubuntu 18.04
New York	2	4GB	0.030/hr	1	Ubuntu 18.04

Fig. 7. Digital Ocean machine configurations, when resources belong to different locations, *i.e.,* different data centers. Rental price is also shown.

experiments with both 1 GB and 2 GB RAM on each worker. Each worker only lends one of its CPU cores. We depict the results in Fig. 8. Several observations can be made. First, the increase in number of epochs linearly increases the completion time, which is because the training data needs to be gone through multiple times. We also observe that additional RAM only marginally reduces the completion time, emphasizing the benefit of renting/borrowing only a partial resource. Finally, we observe that our application can successfully authenticate users and their resources, and allow users to lend only a part of their computing resource (*i.e.,* a single core or 1 GB RAM).

Experiment 2: Our next objective is to compare the performance of DeepMarket across all three setups. We use three resources in all setups, each resource with 1 GB of RAM and 1 core. The results for 100 epochs are plotted in Fig. 9 (we observed similar performance results for a fewer number of epochs). We observe that using resources that belong to the same data center (Single-DC) results in the fastest job completion time. Resources in a single Data center are connected to one another through fiber, which significantly increases the speed of message passing across workers when running distributed TensorFlow programs. Additionally, our rented CPUs had a slightly faster clock, which helped with job completion time compared to "Lab LAN". However, we emphasize that these results do not take into account the time it takes to submit a job (*i.e.,* transfer source file) and retrieve the results at the user. DeepMarket provides a marketplace for computational resources anywhere in the network, particularly at the edge. This can significantly reduce the total time it takes for an edge device to offload its computation and retrieve

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

Fig. 8. Job completion time when three workers are connected to the same WiFi LAN. Each worker only lends a single core.

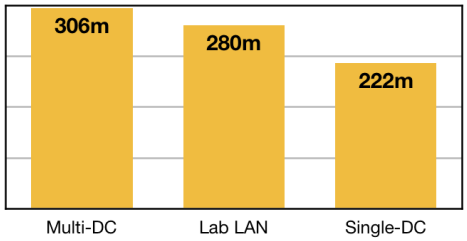

Fig. 9. Comparison across all schemes with 100 epochs. Each setup uses 3 worker machines, each worker with 1 core and 1 GB of RAM. The completion time in minutes is shown on each bar.

the results. The benefits of edge computing in reducing the communication latency has been documented in many prior works, including [22], [23].

Note that different resources from different data centers that belong to a single provider (*e.g.,* resources in Multi-DC) are typically connected to one another through high capacity backbone (Internet) links. However, we observe that "Lab LAN" with WiFi connectivity among resources still outperforms "Multi-DC". Overall, we observe that renting resources on conventional laptops, has a comparable performance to renting computational resources on cloud providers.

Experiment 3. In our final experiment, we investigate how the increase in the number of worker machines impacts the job completion time. We consider the same worker configuration of the previous experiment (*i.e.,* each worker with one core and 1 GB of RAM), but vary the number of workers. The results for "Multi-DC" and "Single DC" are depicted in Fig. 10. We observe that the "Single-DC" architecture consistently outperforms the "Multi-DC" architecture and the completion time across both schemes reduces as a function of number of workers. The results show that as DeepMarket scales in terms of user adoption, it becomes feasible for users to borrow many resources simultaneously and significantly reduce their job completion time.

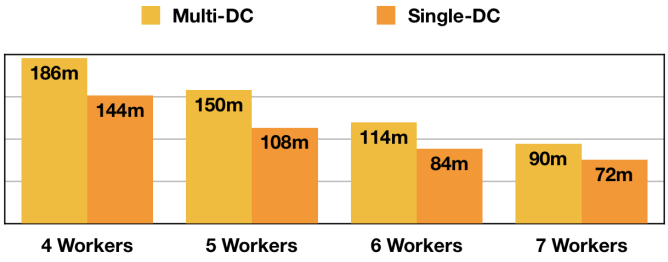

Fig. 10. Training completion time as a function of number of workers. The completion time in minutes is shown on each bar.

IV. Discussion

In this section, we 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 be possibly 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 [22].

Latency: As our 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 [23].

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 the 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 the system reliability. One way to address the issue is to build a scoring system (*e.g.,* [24]) that ranks the reliability of resources and their owners. DeepMarket can then use resources from users with a better ranking. Additional, we can build a redundancy when using resources to counter unexpected job terminations.

V. Conclusion

We 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. We discussed the design of PLUTO, a GUI that simultaneously allows a user to lend and borrow computational resources. We also presented some of the key aspects of our backend services. Finally, we showed through experiments that renting resources on DeepMarket has a similar job completion time to renting resources on the cloud providers. 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 potential increase in user privacy as DeepMarket spreads users' data across resources owned by different lenders.

VI. Future Work

Our current implementation of DeepMarket supports machines with Linux OS. Further, each computational resource needs to have an installation of Apache Spark. We are in the process of releasing a new version of our software with support for Docker [25]. Docker is a lightweight container that can be run on any OS without the need for Spark installation. In addition, it provides a virtual environment in a resource machine, ensuring both the security of lenders' resources and borrower's jobs. In parallel to Docker, we are developing a dynamic pricing mechanism to match jobs and computational resources based on supply and demand. Finally, we are in the process of openly releasing our software to the research community. This would allow others to experiment with the framework and contribute to many aspect of the project such as pricing and blockchain-based credit system design.

References

[1] "Amazon Web Services (AWS)," https://aws.amazon.com/
[2] "Digital Ocean Cloud Computing," https://www.digitalocean.com/
[3] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *ACM SIGCOMM Computer Communication Review*. ACM, 2015, vol. 45, pp. 71–84.
[4] L. Zheng, C. Joe-Wong, C. G. Brinton, C. W. Tan, S. Ha, and M. Chiang, "On the viability of a cloud virtual service provider," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1, pp. 235–248, 2016.
[5] M. Khodak, L. Zheng, A. S. Lan, C. Joe-Wong, and M. Chiang, "Learning cloud dynamics to optimize spot instance bidding strategies," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2762–2770.
[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)*, 2013, pp. 219–228.
[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] "DeepMarket Project Website," https://deepmarket.cs.pdx.edu/
[13] "Apache Spark," https://spark.apache.org/
[14] "HDFS: Hadoop Distributed File System," https://hadoop.apache.org/
[15] "PyGt5 5.11.3," https://pypi.org/project/PyQt5/
[16] "Meet Horovod: Ubers Open Source Distributed Deep Learning Framework for TensorFlow," Available: https://eng.uber.com/horovod/
[17] "Apache MXNet (Incubating)," https://mxnet.apache.org/
[18] "Apache Spark's scalable machine learning library (MLib)," https://spark.apache.org/mllib/
[19] "TensorFlowOnSpark," https://github.com/yahoo/TensorFlowOnSpark
[20] "MapReduce: Simplified Data Processing on Large Clusters," https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
[21] "MNIST database," https://en.wikipedia.org/wiki/MNIST database
[22] D. A. Maltz A. Greenberg, J. Hamilton 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.
[23] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, 2016.
[24] R. Zhou and K. Hwang, "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing," *IEEE Transactions on Parallel & Distributed Systems*, , no. 4, pp. 460–473, 2007.
[25] "Docker Container System," https://www.docker.com/