Performance Review of Clipper for Deployment of Machine Learning Models

Shubham Adep

***Abstract*— Machine learning is now adopted widely. With increasing demand and competition, the deployed models need to scalable, accurate and robust under heavy load. In this paper, we review one such machine learning tool which is reliable, flexible and easy to use. Clipper has a two layered structure which provides abstraction between the machine learning model and the application which just interacts with the model by using REST API end point created by Clipper. Moreover, Clipper introduces caching, batching and model adaption techniques. We evaluated the claims proposed by [4] with an application implemented in TensorFlow under varying loads and hyper parameters to test throughput, scalability, load balancing, and performance comparison with TensorFlow Serving, a serving toolkit by Google.**

I. INTRODUCTION

In real world, deploying a machine learning model is as important as building one. The deployed model has to provide low latency, flexibility, must be accurate and robust under heavy query load. While most machine learning frameworks only focus on model training, clipper helps in making deployment easy and maintainable. Clipper has a modular architecture which allows it to deploy multiple models using different machine learning frameworks. Clipper also provides support for caching, adaptive batching and adaptive model selection which makes it robust, improves throughput and accuracy. This paper, focuses on testing performance of clipper under different scenarios and to verify how it stands compared to existing frameworks.

II. RELATED WORK

Research in the field of machine learning deployment tools has been limited, or restricted to a particular machine learning framework. Presently, the machine learning applications we come across like ad-targeting and content recommendations have an application specific deployment model which is tightly coupled with the workload it is expecting. Clipper generalizes the capability of serving models. Implementations close to Clipper are [1]TensorFlow Serving [6], LASER [2] and Velox [4]. While LASER is not publicly available and Velox has a very limited functionality, TensorFlow Serving is publicly available and has a active community of contributors. Hence TensorFlow is considered the state-of-the-art as of today.

[1]`https://www.tensorflow.org/tfx/guide/serving`

TensorFlow Serving is closely incorporated with TensorFlow [1] framework which helps it to take advantage of the GPU/TPU, but as refered in Crankshaw et al. [3] due to its static batch sizes, when rendering fewer predictions than the batch size, the overall system throughput is decreased, unlike Clipper which utilizes dynamic batching. Clipper uses additive-increase-multiplicative decrease (AIMD) scheme for dynamic batching in which it adds a fixed amount of batches until latency objective is exceded and then decreases the batch size by 10%. TensorFlow serving does not provide option for real-time feedback, [3] explains the importance of feedback by giving the example of model selection for speech recognition in which the feedback from the user for a particular accent is of particular importance for which model to query to.

Basically, Olston et al. [6] refers TensorFlow Serving as a state-of-the-art software framework for most ML serving workloads at Google and in Google's Cloud-ML offerings. And that is the fundamental difference between them and Clipper. Both the systems were being developed concurrently. Motivation of Clipper was making deployement generalized and open to models of multiple frameworks, while TensorFlow serving encapsulates the production infrastructure required inside google where they previously used ad-hoc, non-reusable solutions which became complicated as the models matured.

III. BACKGROUND

Before looking into the performance review, lets first look at how the system caters predictions for incoming requests. Clipper creates an application facing REST APIs which the applications issue to get the prediction requests. There are two layers in Clipper's architecture: Model selection layer and model abstraction layer. The prediction requests are first processed by the model selection layer which dispatches the prediction requests to one or more models through model abstraction layer. Model abstraction layer has three components, cache, batching and cross language RPC calls. First the layer checks if the query has been catered, if so it saves the processing time of the model and returns the saved prediction. If not, it calls adaptive batching which constructs the queue as per the configuration of the desired model and framework, and then uses RPC calls to send in the queries. The predictions from the model use the same path back to return to the application and updates the cache as well.

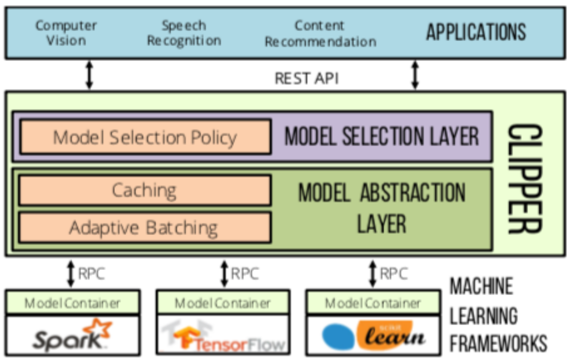

Fig. 1. Clipper architecture as described in Crankshaw et al. [3]

IV. Dataset and Experimental Setup

Experiments are going to be performed on a classifier built for deceptive review detection. The classifier is a deep neural network using pre-trained vectors as word embeddings. Pre-trained vectors are used to gather contextual closeness of words in the embeddings and also to reduce the dimentionality of the feature vector.

We use the 'gold standard' dataset used by Ott et al. [7]. The dataset has 1600 review texts from 20 different restaurants in Chicago area, USA, which have 800 deceptive and 800 genuine reviews. From those 800 reviews, 400 reviews are written with a negative intent and 400 reviews with a positive intent. These reviews were obtained from various online reviewing websites such as Yelp, TripAdvisor, Expedia and Hotels.com. The deceptive reviews were generated using Amazon Mechanical Turk (AMT). We divide the data set into training and testing data sets in the ratio 75:25. The test data set will be duplicated if needed to increase the queries per second given to test the deployment of the model in Clipper.

For the purpose of comparing the performance with TensorFlow serving we are using Convolutional Neural network implemented in TensorFlow on the MNIST dataset [5].

V. Experiments

For performance review we need to test for throughput, scalability and load balancing while achieving latency objective required by the system as explained in [4]. Clipper is also known as a function server, which means it can also deploy a function. Hence it is not a necessity to test the deployment with a machine learning model. But in this paper, we are going to test it with a previously implemented classifier as explained in the section: IV.

A. Throughput

Throughput in a system can be affected by many factors. Specifically, in case of Clipper, the system component parameters such as dynamic batching, delayed batching, the resources allocated to the containers and also the performance if the containers are replicated. In this experiment these are the factors which are to be tested. The CNN classifier for deceptive review detection will be deployed and linked with the application inside Clipper. Batch size is to be increased and the throughput in terms of queries per second will be observed.

We tested the system with 100,000 queries (20 queries concurrently) using [2]Apache Bench, which is a benchmarking tool. We sent the queries while increasing the batch sizes. The expected result was, the time taken for the queries to be catered and throughput should increase as we increase the batch size. Figure:2 shows the performance degradation as the batch sizes are underutilized. And it can be verified in terms of time in figure:3.

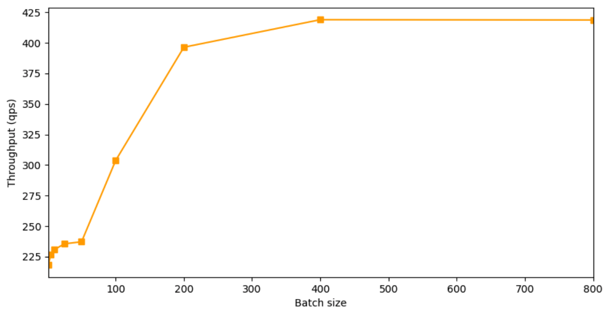

Fig. 2. Throughput of Clipper with increasing batch size

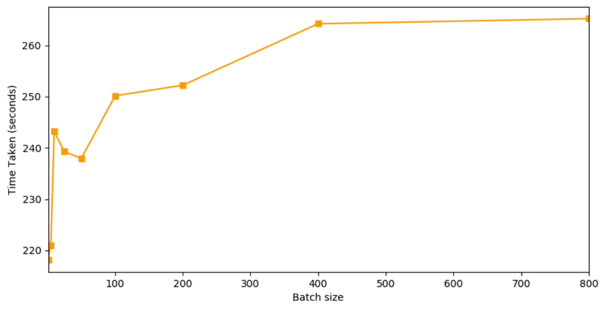

Fig. 3. Time taken to cater 100,000 queries by Clipper with increasing batch size

B. Scalability and load balancing

Clipper provides various tools for scalability. Apart from just allocating more resources to each container to process the data faster, it is also possible to deploy replicas of the same container. By doing so, the model selection layer will make sure to load balance the queries to the right container. Hence, this experiment will focus on testing the magnitudes at which the performance can be increased.

In this experiment, with the same setup as in section V-A, we increase the number of replicas while keeping the batch size and service level objective constant. With Clipper it is easy to deploy multiple replicas at the

[2]`https://httpd.apache.org/docs/2.4/programs/ab.html`

start with the model deployment. Figure:4 and figure:5 show drastic increase in the throughput and decrease in overall time taken as we deploy multiple replicas.

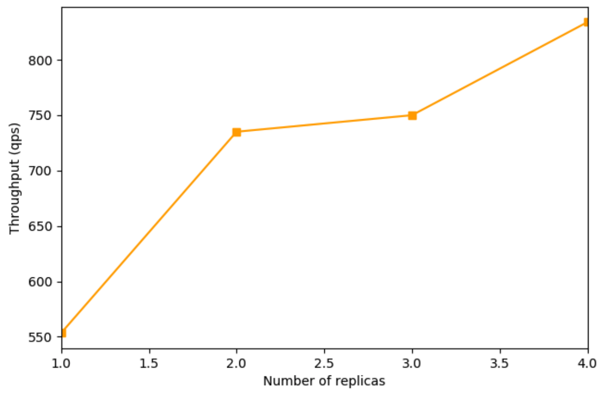

Fig. 4. Throughput of Clipper with increasing number of replicas

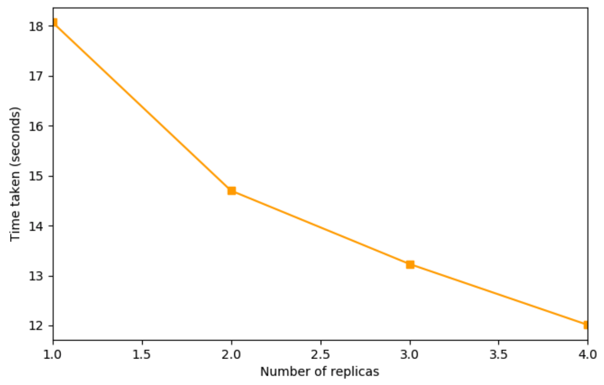

Fig. 5. Time taken to cater 100,000 queries by Clipper with increasing number of replicas

C. *Service Level Objective*

Latency is an important factor in deployment, it is important to maintain the latency and performance of the model. Some models have an exception in the trade-off between the accuracy and latency if the user can be shown a default value if the confidence of the models is collectively low. As all the communications in the Clipper environment are proceeded with RPC calls, and also batching and processing of queries can cause the latency to be high. Clipper allows the user to set a latency objective, and it manages the batch size accordingly, to not let the latency exceed the desired interval. Hence this experiment focuses on testing how the model performs (in terms of throughput) when latency objective is varied.

For this experiment we reduced the load queries from 100,000 to 10,000. Also, for this experiment we use dynamic batching as used in [3]. With the capability to set batch size dynamically, Clipper will dynamically adjust the batch sizes as we vary the Service Level Objective. Figure:6 illustrates how the throughput of the system is increased as we increase the SLO micro.

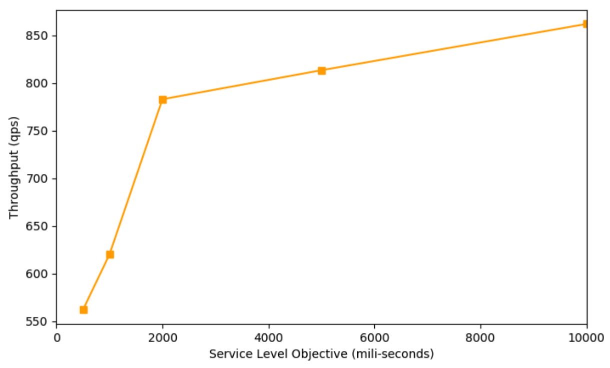

Fig. 6. Throughput as we increase the Service Level Objective (SLO micro)

D. *Performance comparison*

As Olston et al. [6] mentions that development of Clipper and TensorFlow serving were concurrent, and that Clipper is the closest effort to TensorFlow-Serving. Hence its important to measure the motivation and performances of both the systems. As TensorFlow Serving is tightly coupled with TensorFlow, we need an implementation of an experiment in TensorFlow which can be used to test the two systems. Throughput will be compared in this experiment. Time taken by Clipper is expected to be less than Tensorflow due to the communication overheads.

Review Detection model explained in IV is deployed using with TensorFlow Serving. Number of queries was kept to 100,000 with 20 requests sent concurrently and the Apache Bench was used as a benchmarking tool for Clipper and TensorFlow Serving. Figure: 7 shows throughput for both the deployment tools with increasing batch size. The difference in performance is due to more communication overhead in Clipper, whereas TensorFlow Serving is tightly coupled with TensorFlow. Figure:8 depicts how the total latency of catering the queries varies with increasing batch size. [3] claims that with TensorFlow C++ API the performance was close to TensorFlow serving, but in our case we are using TensorFlow's Python API which was expected to perform below the TensorFlow Serving. Figure: 9 illustrates the performance in terms of time taken for the review model implemented in pure tensorflow compared to the experiments shown in figure: 8 which were implemented in Keras and performs worse due to the overhead.

VI. Discussion and Conclusion

In this work, we implemented experiments to test Clipper as a reliable machine learning deployment tool. We carried out experiments for testing throughput, scalability and latencies of 100,000 queries. When tested with increasing batch sizes, number of replicas, and varying Service Level Objective, clipper turned out to be scalable with 0-5% failure rate for the queries. The failure rate depicts that the system was pushed to a limit

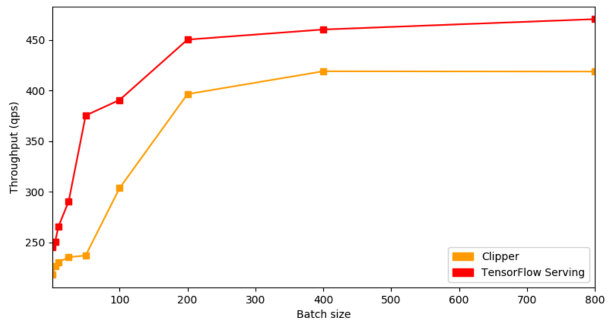

Fig. 7. Comparison in performance between TensorFlow Serving and Clipper

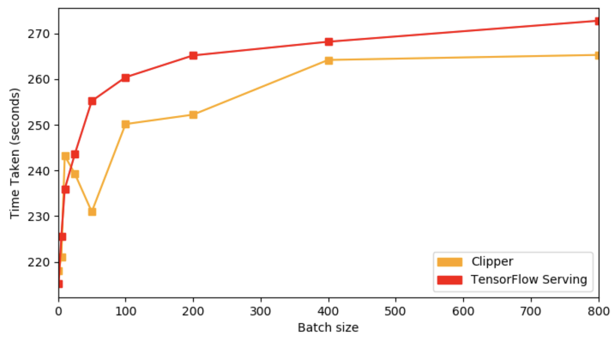

Fig. 8. Comparison in performance between TensorFlow Serving (Implemented with Keras) and Clipper in terms of time taken

to test the aspects with increasing queries to a point that the system could not handle it. When compared to TensorFlow Serving which is a close competitor, it performed below par, but considering the flexibility and ease of deployment, clipper should be the best option as of the date, which supports deployment of machine learning models with multiple frameworks.

References

[1] Martin Abadi et al. “TensorFlow: A system for large-scale machine learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: `https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf`.

[2] Deepak Agarwal et al. “LASER: A Scalable Response Prediction Platform for Online Advertising”. In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM ’14. New York, New York, USA: ACM, 2014, pp. 173–182. ISBN: 978-1-4503-2351-2. DOI: `10.1145/2556195.2556252`. URL: `http://doi.acm.org/10.1145/2556195.2556252`.

[3] Daniel Crankshaw et al. “Clipper: A Low-Latency Online Prediction Serving System”. In: *CoRR* abs/1612.03079 (2016). arXiv: `1612.03079`. URL: `http://arxiv.org/abs/1612.03079`.

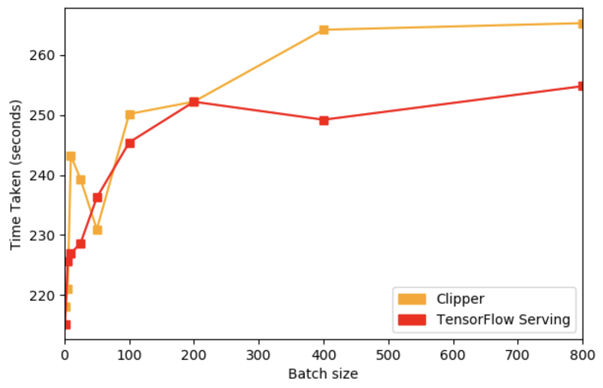

Fig. 9. Comparison in performance between TensorFlow Serving (Implemented in pure tensorflow) and Clipper in terms of time taken

[4] Daniel Crankshaw et al. “The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox”. In: *CoRR* abs/1409.3809 (2014). arXiv: `1409.3809`. URL: `http://arxiv.org/abs/1409.3809`.

[5] *Deep MNIST for experts*. URL: `https://www.tensorflow.%20org/versions/r0.10/tutorials/mnist/pros/index.%20html.`.

[6] Christopher Olston et al. “TensorFlow-Serving: Flexible, High-Performance ML Serving”. In: *Workshop on ML Systems at NIPS 2017*. 2017.

[7] Myle Ott et al. “Finding deceptive opinion spam by any stretch of the imagination”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 309–319.