Litz: Transparent Elasticity for High-Performance Machine Learning

Aurick Qiao*,† Abutalib Aghayev†
Weiren Yu* Haoyang Chen* Qirong Ho* Garth A. Gibson† Eric P. Xing*,†
*Petuum, Inc. †Carnegie Mellon University

1. Introduction and Overview

Modern clouds and data-centers are multi-tenant environments in which the set of running jobs and available resources (CPU, memory, etc.) at any given time are constantly changing [2, 11, 5]. At the same time, Machine Learning (ML) is quickly becoming a dominant application among modern distributed computing workloads. It is therefore highly desirable for ML applications executing in such an environment to be *elastic*, being able to opportunistically use additional resources when offered, and gracefully release acquired resources when requested. Elasticity is beneficial for both the individual job and for the cluster as a whole. An elastic job can make use of idle resources to complete within a shorter amount of time, and still make progress when some of its resources are removed. A cluster-wide job scheduler can dynamically re-allocate resources to speed up urgent real-time or interactive jobs, and ensure fairness by preventing jobs from holding highly contested resources for long periods of time.

Recent advancements in distributed ML frameworks, such as GraphLab [9], Petuum [14], Adam [4], Nomad [15] and various parameter servers [8] have improved the performance of distributed ML applications by an order of magnitude or more over general-purpose frameworks. They exploit unique properties of ML algorithms not always found in conventional data-processing applications, such as bounded staleness tolerance [6], uneven convergence [7], serial and parallel dependency structures in the ML model [7, 15], opportunities for bandwidth management and network message re-prioritization [12], and network message compression [13, 4]. However, there has not been as much focus on supporting elasticity, limiting the usefulness of ML frameworks in real-world computing environments.

Although there are many factors that complicate building an elastic ML framework, we summarize the properties of ML applications which we believe to be the most challenging to support:

1. Memory access patterns which encourage application developers to co-locate mutable model parameters with immutable data entries, resulting in workers that are stateful.
2. Dependency structures in their models which are exploited by efficient distributed implementations through careful scheduling of computation.
3. Robustness against errors which is exploited by ML systems and algorithms to obtain higher performance by giving up deterministic execution and consistency of memory accesses.

Thus, an elastic framework that is efficient and general within the ML domain should support **stateful workers**, **model scheduling**, and **relaxed consistency**. To address the need for elastic machine learning and the challenges listed above, we designed and implemented a new system called *Litz*, which achieves the following goals:

1. **Transparent Elasticity**: When physical machines are added or removed during execution, state and computation are migrated to the available machines without active intervention from the application.
2. **High-Performance ML**: Litz's programming model can express key distributed ML techniques such as stateful workers, model scheduling and relaxed consistency, allowing high-performance applications to be implemented.

At a high level, Litz's programming model exposes an actor-like API in which the application defines a single logical *driver* and a set of logical *executors* which communicate with each other via messages. Application code only executes when invoked by the framework during handling of certain messages and events. In this way, the framework controls when the application code executes, enabling it to migrate the driver and executors between physical machines without active involvement from the application. The specific techniques used in Litz are as follows.
Input Data Over-Partitioning: The input data is distributed across the set of executors, which are mapped by the framework to physical machines. Typically, a larger number of executors

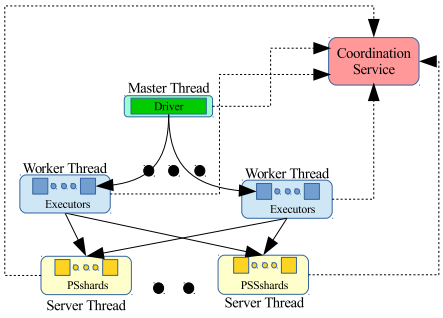

Figure 1. High-level architecture of Litz. The driver in the master thread dispatches micro-tasks to be performed by executors on the worker threads. Executors can read and update the global model parameters distributed across PSshards on the server threads.

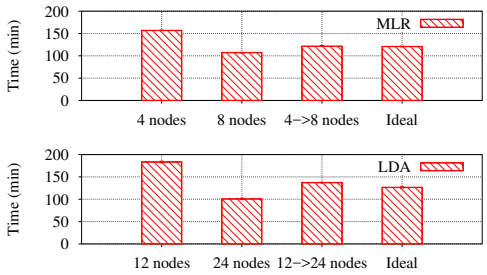

Figure 2. Static, scale-out, and ideal scale-out execution times for MLR and LDA implemented on Litz. We scale out MLR from 4 nodes to 8 nodes, and LDA from 12 nodes to 24 nodes.

are created than the number of physical machines, so that excess executors can be migrated to any additional physical machine that become available for use.

Parameter Server and Micro-Tasks: Update calculations are decomposed into short-lived (typically shorter than 1 second) units of computation called *micro-tasks*, each of which calculates a partial update using the input data on a single executor. Micro-tasks enable the application to specify finer-grained dependencies, and gives the run-time system more frequent opportunities to rebalance. During its execution, a micro-task is granted read/update access to a global parameter server via a key-value interface and applies partial updates to the model parameters by modifying application-defined state in the executor and/or updating globally-shared values in the parameter server.

Model Scheduling and Relaxed Consistency: Litz enables both model scheduling and bounded staleness by letting the application specify dependencies between micro-tasks. If micro-task A is a dependency of micro-task B, then (1) B is executed before A and (2) B sees all updates made by A. This strict ordering and consistency guarantee lets the application perform model scheduling by defining an ordering for when certain updates are calculated and applied. On the other hand, if neither A nor B is a dependency for the other, then they may be executed in any order or in parallel, and may see none, some, or all of the updates made by the other. This critical piece of non-determinism lets the application exploit the error-tolerance property of ML by allowing the run-time system to cache and use stale values from the parameter server between independent micro-tasks.

2. Implementation and Evaluation

Litz is implemented in approximately 6500 lines of C++ code using the ZeroMQ [3] library for low latency communication and Boost's Coroutine2 [1] library for low overhead context-switching between micro-tasks. The run-time system is comprised of a single *master thread* along with a collection of *worker threads* and *server threads*, as shown in Fig. 1. The application's driver exists in the master thread and its executors exist in the worker threads. The key/value pairs comprising the parameter server are distributed across a set of logical *PSshards* stored in the server threads. Additional worker and server threads may join at any time during the computation, and the run-time system can re-distribute its load to make use of them. They may also gracefully leave the computation after signaling to the master thread and allowing their load to be transferred to other threads.

First we compare our Litz implementations of Multinomial Logistic Regression (MLR) and Latent Dirichlet Allocation (LDA) with those that are distributed with the open-source versions of Bösen [12] and STRADS [7], respectively. Since we wish to demonstrate that Litz by itself is able to efficiently support the same applications supported by these more specialized frameworks, we closely follow their implementations in Bösen and STRADS, sharing a significant portion of the core algorithm code. Comparing Litz with Bösen running the MLR application on 25% of the ImageNet ILSVRC2012 dataset [10], our MLR implementation on Litz converges about 8× faster. We believe the performance deficit of Bösen arises from implementation overheads such as unnecessary memory copying and inefficient locking. Comparing Litz with STRADS running the LDA application, our LDA implementation on Litz converges only 1.06× slower, well within the margin for error. These results show that Litz's programming model can be implemented in a way that is competitive in performance with the state-of-the-art.

Second, we evaluate Litz's performance when scaling a running application out to a larger number of nodes. We compare our scale-out experiments with static executions of the applications using the pre-scaling number of nodes. The static pre-scaling execution for MLR completes in ≈157min while the scale-out execution completes in ≈122min, and the static pre-scaling execution for LDA completes in ≈183min while the scale-out execution completes in ≈137min. Additionally, we define and compare with a simple *ideal* scale-out execution time which intuitively measures the total run-time of a job that instantly scales out and adapts to use the additional nodes. MLR achieves a less than 1% overhead while LDA is less than 5%. We see similar performance results for scaling in.

References

[1] Boost Coroutine2. www.boost.org/doc/libs/1_63_0/libs/coroutine2/.

[2] Kubernetes. http://kubernetes.io.

[3] ZeroMQ. http://zeromq.org.

[4] CHILIMBI, T., SUZUE, Y., APACIBLE, J., AND KALYANARAMAN, K. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association, pp. 571–582.

[5] HINDMAN, B., KONWINSKI, A., ZAHARIA, M., GHODSI, A., JOSEPH, A. D., KATZ, R., SHENKER, S., AND STOICA, I. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2011), NSDI'11, USENIX Association, pp. 295–308.

[6] HO, Q., CIPAR, J., CUI, H., LEE, S., KIM, J. K., GIBBONS, P. B., GIBSON, G. A., GANGER, G., AND XING, E. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds. 2013, pp. 1223–1231.

[7] KIM, J. K., HO, Q., LEE, S., ZHENG, X., DAI, W., GIBSON, G. A., AND XING, E. P. Strads: A distributed framework for scheduled model parallel machine learning. In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, ACM, pp. 5:1–5:16.

[8] LI, M., ANDERSEN, D. G., PARK, J. W., SMOLA, A. J., AHMED, A., JOSIFOVSKI, V., LONG, J., SHEKITA, E. J., AND SU, B.-Y. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association, pp. 583–598.

[9] LOW, Y., BICKSON, D., GONZALEZ, J., GUESTRIN, C., KYROLA, A., AND HELLERSTEIN, J. M. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow. 5*, 8 (Apr. 2012), 716–727.

[10] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., BERG, A. C., AND FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[11] SCHWARZKOPF, M., KONWINSKI, A., ABD-EL-MALEK, M., AND WILKES, J. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)* (Prague, Czech Republic, 2013), pp. 351–364.

[12] WEI, J., DAI, W., QIAO, A., HO, Q., CUI, H., GANGER, G. R., GIBBONS, P. B., GIBSON, G. A., AND XING, E. P. Managed communication and consistency for fast data-parallel iterative analytics. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (New York, NY, USA, 2015), SoCC '15, ACM, pp. 381–394.

[13] XIE, P., KIM, J. K., ZHOU, Y., HO, Q., KUMAR, A., YU, Y., AND XING, E. P. Distributed machine learning via sufficient factor broadcasting. *CoRR abs/1511.08486* (2015).

[14] XING, E. P., HO, Q., DAI, W., KIM, J. K., WEI, J., LEE, S., ZHENG, X., XIE, P., KUMAR, A., AND YU, Y. Petuum: A new platform for distributed machine learning on big data. *IEEE Trans. Big Data 1*, 2 (2015), 49–67.

[15] YUN, H., YU, H.-F., HSIEH, C.-J., VISHWANATHAN, S., AND DHILLON, I. Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *Proceedings of the VLDB Endowment 7*, 11 (2014), 975–986.