ECO: Harmonizing Edge and Cloud with ML/DL Orchestration

Nisha Talagala Swaminathan Sundararaman Vinay Sridhar Dulcardo Arteaga
Qianmei Luo Sriram Subramanian Sindhu Ghanta Lior Khermosh Drew Roselli
ParallelM

Abstract

Edge Computing and Machine Learning are complementary advances: edge devices drive volumes of rich data that benefit ML, and ML drives insights that can justify edge investments and create killer applications such as AR/VR. We describe the challenges of ML/DL on the edge and describe guidelines for addressing them. We present Edge Cloud Orchestrator (ECO), an architecture for enabling realistic ML deployments that leverage both edge and cloud by providing an abstraction to orchestrate, manage, and automate ML pipelines. By separating the control path from the data path, ECO handles scale, heterogeneity, and advanced ML patterns.

1 Introduction

Edge devices are generating more data and more complex data than ever before [19]. Devices ranging from sensors to robots are generating vast amounts of sensor readings, images, sounds, and videos, whose ultimate value typically comes from the analytics that extract the insights hidden within. As the richness of edge data grows, so does the need for advanced analytics such as Machine Learning (ML) and Deep Learning (DL).

It is challenging to effectively deploy and manage edge ML/DL in production. Data is usually generated on edge devices, but multiple levels of compute can exist between the data generation point and the final data repository [30]. From an analytics point of view, each of these compute elements may see different *liveness*, *history*, and *perspective* of the data. Typically, edge compute nodes closest to the data generation point have highest liveness but little or no history and no perspective. Compute nodes further away from the sensor and closer to the cloud have more history and perspective but older data. An ideal ML/DL solution would optimize accuracy, latency, and infrastructure efficiency by enabling each layer to focus on the analysis that is optimal for the liveness, history, and perspective of the available data.

In real world production, data patterns change without warning. Previously trained ML/DL models can become irrelevant and subsequent predictions inaccurate. Model re-training is therefore needed and can require human intervention. Even determining when to retrain can be difficult. A common practice is to retrain frequently, but this is expensive. An ideal ML/DL solution would maintain accuracy in the face of changing data via dynamic models that can adapt, as well as a management system that can refresh models as needed.

While analytics can run virtually anywhere, the resources at each layer can be very different. An edge node could be a single server on a manufacturing floor, while a higher layer could be a public or private cloud. An ideal ML/DL solution would reduce management cost by combining the best suited analytic engines at each level of the topology into a seamless unified workflow.

This paper describes Edge Cloud Orchestrator (ECO) which orchestrates and manages ML/DL execution across distributed layers. Our contributions are: (i) We describe the key challenges to be addressed in a successful ML deployment with edge devices; (ii) We describe and advocate a graph-based overlay network approach to specify ML pipeline dependencies and an Agent/Server architecture to manage ML execution; and (iii) We illustrate the use of our approach on several common ML execution patterns for edge/cloud ML deployment.

Section 2 describes the challenges of ML/DL using edge/cloud. Section 3 describes the resulting requirements. Sections 4 and 5 describe our approach and implementation. Section 6 illustrates how our system enables both common and advanced ML patterns for edge computing. Section 7 covers related work and Section 8 concludes with a discussion and future work.

2 Motivation

The rise of IoT with its voluminous data is pushing computing back to the edge necessitating intelligent devices [36]. Interestingly, edge computing and ML are two sides of the same coin [46].

2.1 Challenges Edge brings to ML

Edge computing connects “things” to a compute/storage infrastructure. The number of such devices and the volume of data has been exponentially increasing [11, 15] and is expected to continue in the future [20]. In this section, we outline the Edge ML/DL challenges we have experienced working with a variety of edge applications including industrial IoT (such as wind energy turbines), transportation (smart trains), smart buildings, and AR/VR and distributed entertainment systems. Such Edge environments generate unique challenges for ML/DL.

Latency and Disconnected Operation: Latency sensitive applications must generate insights or react to data changes quickly. For example, in a Smart Building application such as video based intruder detection, it is not always possible to send the image data to the cloud for processing and wait seconds for a response. In Industrial IoT applications, fault detection windows can sometimes be in the 10s of milliseconds, precluding a round trip to a cloud. Even latency tolerant applications (such as health diagnostics) may require edge ML to operate in disconnected scenarios. These types of scenarios necessitate at a minimum ML/DL inference at edge, possibly with training in cloud, edge, or across both.

Security and Cost: The cost of sending petabytes or terabytes of data per hour to cloud devices can be prohibitive [39] and may incur security or privacy concerns [13] . For example, it is possible for industrial IoT applications to generate several MB/s of time series sensor data, and for other apps such as smart transportation (that capture image or video data) to generate even more. Moreover, using precious backbone bandwidth to transmit data is not always beneficial if value can be generated by bringing the compute to the data. Such constraints can affect how and where ML/Dl training is deployed.

ML Processing Overhead: Sensor data can also be complex [9, 18, 21], mandating significant resources. Edge devices need to reduce cost and power, thus requiring the complex processing on rich data be efficiently distributed across both edge and cloud.

Managing these constraints efficiently requires distributing inference and training operations in toplogies that match the needs of the specific environment, recognizing that one topology will not work for all.

2.2 Challenges ML brings to Edge

The promise of data volumes from edge could greatly benefit ML/DL [18, 38]. However, to enable effective edge ML, we need to address the following challenges.

(Re)Training: Since the models trained via ML are data dependent, one would have to periodically retrain when the data pattern changes. (Re)training is compute intensive and iterative [12]. Traditionally, offline algorithms are predominantly used to train models and such trained models are deployed in production to serve requests. Online learning enables the model to locally evolve while also serving requests. Recent advancements in ML have enabled a combination of both online and offline training methods to be used to improve accuracy [34, 41]. This is particularly relevant for edge devices since many edge devices may deploy the same centrally trained model but then evolve it online to meet edge-specific scenarios.

Diversity and Heterogeneity: There are many ML/DL Engines (such as Spark and TensorFlow). Moreover, ML pipelines from the same application may run on different analytic engines. For example, it is a common application pattern to run training as a batch job on Spark and predictions in a streaming context or REST service. [5, 7]. For latency-sensitive edge environments, streaming inference can be quite common [32], but the model used therein could have been batch-trained on the cloud.

Model and Code Propagation: ML requires models to be periodically updated. Model updates and deployment are critical to the health of ML initiatives. A single model from training could be deployed over thousands of edge nodes. For a large-scale edge installation, one may not want to deploy a new model to all of the edge instances at the same time, but rather incrementally deploy the new model with increasing confidence in the field. This requires a way to seamlessly deploy both code and models with different deployment policies.

Diagnostics: Debugging issues in a decentralized environment is challenging. In addition to issues with runtimes, algorithms, configuration parameters, model deployment, etc., ML adds an additional dimension since it is data dependent and model performance is loosely coupled with its training pipeline(s). One has to enable collection of information not only about the distributed ML pipelines but also their runtime statistics, logs, and any user-defined statistics. It is most productive to debug an ML application as a single entity and not as a set of independent entities.

The above ML-specific requirements will need to be supported for varied compute topologies to best match the edge/cloud constraints previously described.

3 Requirements for Edge/Cloud ML/DL

To effectively address the challenges highlighted in Section 2, ML needs to be distributed across edge and/or cloud instances. Multiple edge instances can use the same model, making model management important. (Re)training can be done periodically and model deployment needs to be staged. Multiple combinations of both online and offline algorithms can used to generate meaningful insights and quickly adapt to changes in the data. With these assumptions, we now list the requirements for a distributed ML deployment.

Scalable and Automated: The Edge applications com-

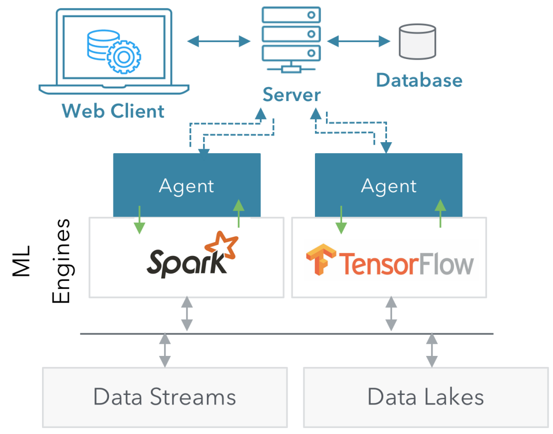

Figure 1: **Architecture of ECO**

monly scale from 100s to 1000s of nodes. Managing ML at such scale requires automating ML including model/code deployment and replacement, schedules to run training/inference pipelines (if needed), policies to decide on the staging, and propagation decisions.

Panoptic View: It is important to have a global view of ML initiatives deployed in both the edge and cloud nodes. Given the scale of possible deployments, it is important to enable centralized oversight. Moreover, support should be built to fetch runtime information from remote instances for analysis and diagnostics.

Consistent: The connectivity between the edge and cloud can be intermittent. It is important to provide the flexibility to choose the consistency guarantees for different ML updates (such as models or code). A good starting point would be to support weak, eventual, and strong consistency because these support commonly built-upon guarantees that span a spectrum of trade-offs between Consistency, Availability, and Performance (CAP).

Topology and Dependency Tracking: Understanding the relationship between training and inference pipelines can help automate model deployment, improve diagnostics, and define provenance to track issues. While this problem exists across ML, it is exacerbated by complex topologies, including large numbers of distributed interdependent pipelines. When connectivity limitations and large scale force weak or eventual consistency policies for model updates, a strong governance mechanism to track who has what when becomes critical [45].

4 Edge Cloud Orchestrator (ECO)

We describe ECO, our approach to manage ML over edge and cloud. Our solution supports a wide variety of deployment scenarios with diversity in ML engines and programming languages. Figure 1 shows the ECO architecture. ECO works on the control plane, enabling organizations to continue to run existing ML pipelines and enforce their data security mechanisms unchanged.

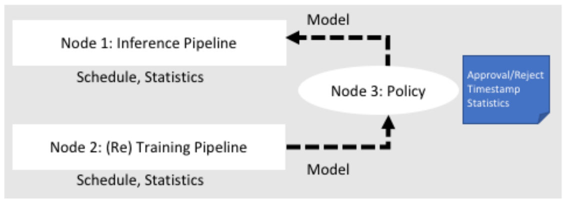

Figure 2: **Example ION:** An example ION containing a training and inference pipeline with a model approval policy node between them.

ECO employs a distributed server-agent architecture. The server is a highly available, scale-out entity that orchestrates, tracks, and manages all ML pipelines across edge and cloud systems. The server communicates with Agents, each of which manages a logical Analytic Engine that runs ML pipelines. Engines can run on edge compute devices, on the cloud, or on any compute in between. Agents can also manage standalone programs in resource-constrained environments.

4.1 Intelligent Overlay Network (ION)

We create IONs to address the core issues of having a panoptic view and ability to track dependencies. IONs contain a two-level logical graph of ML/data pipelines, policy modules, and messages. Each node of the ION graph can in turn be a Directed Acyclic Graph (DAG) of components. All executions (ML Pipelines, Policy Modules) are nodes. Nodes run on schedules (batch), continuously (streaming), or are event triggered. Nodes can pass messages between them (see Figure 2). Each node can be a DAG of components (matching the pipeline frameworks in Spark and other engines). Users specify their pipeline topology and dependencies into an ION. For example, Figure 2b shows an ION where the cloud periodically trains a batch model and then, after operator approval, pushes it to 3 edge devices for streaming inference. Through IONs, ECO can track all the interdependencies within a single ML application. ECO has the information about all ML pipelines, deployed models at each instance, and status of each of the ML pipelines.

4.2 Server and Agents

The ECO server is responsible for managing and executing IONs. Each node of the ION is an executable component (such as pipeline and policy) that runs on either a schedule or an event trigger (such as a model approval). The ECO server maps the nodes of an ION (each execution) to one or more Agents. With this construct, different parts of an ION can run on geographically distributed analytic engines ranging from small edge-based engines to powerful multi-node cloud-based engines.

The Agent's job is to coordinate with the server, control execution of ML pipelines on its assigned analytic engine, and monitor the progress and state of ML pipelines. Agents periodically connect to the server and update the state of the running ML pipelines. The server can also request (either on a schedule or via a user triggered event) the status from an agent. The ECO agents integrate with their respective analytic engines using standard engine APIs (i.e., the engines are unmodified). The agent also schedules execution of ML pipelines at specified intervals and relaunches jobs on failure.

The ECO server communicates with the agents and launches the pipelines and policies as needed through the Agents. It transfers the models and other dependent messages between agents. The ECO server gathers runtime statistics/events from all execution elements and stores them in a SQL-compliant database. This database, combined with the dependency/relationship information specified in the ION, enables a global timeline record and centralized view that is used for diagnostics, governance, auditability, and overall optimization.

4.3 Scalability

The server is responsible for control and information dissemination to the agents, and each agent, in turn, is responsible for communication with ML instances to execute the ML pipelines. The decoupling of responsibility between the server and its agent, along with operating on the control path, enable scalability of our solution.

4.4 Heterogeneity

Heterogeneity of analytic engines is supported via the Agent model. Each agent communicates with the server via the same protocol but communicates with engines in engine-specific ways. With this model, heterogeneity is seamlessly accommodated. For example, an ML algorithm written in Java can be trained on a Spark cluster in the cloud, with the resulting model used in a Python-based streaming inference pipeline at the edge.

4.5 Disconnected Operation

Each agent maintains network connectivity with its ML engine(s) and at least intermittent connectivity with the server. Agents buffer during disconnection from the server and the managed pipelines continue to run (statistics are temporarily stored locally). When disconnected for long periods of time, intermediate statistics may be lost if the edge device has limited storage. The server queues control actions (such as model updates) and issues them when connection is re-established.

5 Implementation

The ECO is $\sim$50K lines of Java and works in all environments. We chose a platform-neutral implementation to support many edge/cloud configurations [4, 6, 33]. For storing ECO-gathered metadata (statistics, models, configurations, and other objects), we use SQLite [22] and InfluxDB [24]. We use Zookeeper for ECO Server quorum and Zookeeper's storage for the ION configurations [23]. The ECO server/agents communicate via REST APIs. ECO agents are multi-threaded and require a minimum of 128MB of memory. In the current version, the services are not automatically scaled based on the local site requirements. In an edge-only configuration, an ECO server can run on an edge or gateway compute. Otherwise, we expect the ECO server to run in a cloud environment and the Agents to run wherever needed. To date, we have tested our system with up to 512 edge instances, each running on at least 128MB of memory and a range of compute cores.

We currently support three Analytic Engines (Spark [54], Flink [3], and TensorFlow [8]) and pipelines in Scala, Java, and Python. We have tested our system with both online and offline ML algorithms.

6 Use Cases

We illustrate the versatility of ECO/ION via use cases demonstrating ML edge challenges: Federated Learning, Transfer Learning, and Staged Model Deployment.

6.1 Federated Learning

Federated Learning avoids transferring data to the cloud and leverages independent edge models [42]. These edge models are merged to learn a global predictive model including the ability to track the relationship between these models [43]. This approach also ensures adaptability of these models to changing incoming data.

Figure 2(a) shows the ION graph for Federated Learning. Each edge node learns models, and the distributed "edge" models are periodically sent to a *merge*, which combines them to create a global model.

Our experiment deploys 3 edge ML pipelines running *Online KMeans*. Our test dataset uses 2 features and 4 dynamically varying clusters. The results shown in Figure 4 demonstrate that the system was able to not only adapt to the changes in their data pattern but also learn across edge instances without transferring data from the edge to the cloud.

6.2 Transfer Learning

In Transfer Learning, [16, 37, 40, 48, 49, 51], a "generic" model is trained offline (likely in the cloud) and the resulting model is refined with incremental training at each local edge node. It is popular in face recognition where the "generic" model is trained on millions of faces and then "edge" models customize further for a target user subset [14, 52]. Periodically, the "generic" model could be retrained to improve baseline accuracy and updates would periodically need to be sent to the edge instances.

Figure 2(b) shows the ION graph for this use case. The "General" model is trained (and retrained) on a cloud engine. Once an operator is satisfied with the accuracy, the model is deployed to the edge nodes. The edge nodes combine the "generic" model with its "edge" model using an incremental training pipeline. We have enabled transfer learning in our system for four algorithms. On

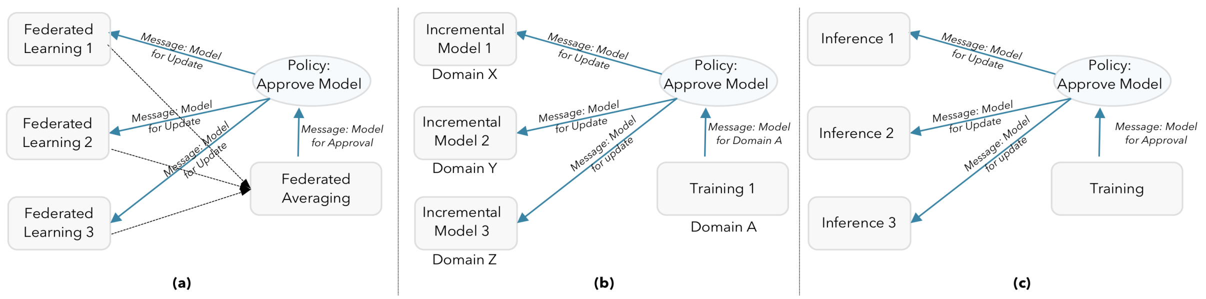

Figure 3: **Example Use Cases.** *(a)* ***Federated Learning:*** *(i) The "edge" models are sent to the ECO server; (ii) ECO server forwards the models to the Agent in the cloud for combining them; (iii) The combined model is sent back to the server; and (iv) The server forwards the combined model to the Agents at the edge. (b)* ***Transfer Learning:*** *(i) A model is trained in an offline way in the cloud and the agent forwards the trained model; (ii) This (pre)trained model is deployed to the ML instances via the Agents at the edge; (iii) (Not Shown) the edge ML instances use the pre-trained model to initialize their learning and continue to learn and predict in their environment. (c)* ***Model Deployment:*** *Models are deployed in phases using ECO server. Periodically the server checks the efficacy of the models and propagates the new model to other agents on a user-defined policy.*

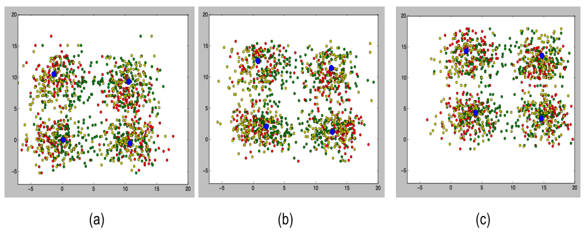

Figure 4: Data on each of the 3 edge nodes is shown in a different color. Blue points represent the global clusters calculated at the server.

average, the accuracy at the edge was 40% higher when transfer learning was used in our experiments.

6.3 Staged Model Deployment

Figure 2(c) shows the ION for a staged model deployment from training (at cloud) to inference (at edge). We focus on deploying in the presence of intermittent connectivity, edge node failures, and the likelihood of a partial update increasing with scale of edge nodes. When configured for strong consistency, ECO guarantees consistent model updates via two phase commit between the server and agents. The server can also use the statistics collected at the agents and stage model deployment in multiple phases with increasing levels of confidence. For example, a model can be deployed to 10 percent of the nodes, monitored, and then deployed to the next 10 percent, etc. As such, the confidence of deploying a model across 1000s of nodes can be gradually increased.

7 Related Work

ML on edge is described in prior works [26, 27, 31, 42, 47, 55]. These efforts focus on reducing the data transfer, memory footprint, and CPU cycles of ML pipelines. Our ECO work is complementary; ECO Agents can work with such edge-optimized ML engines.

Recent Amazon and Microsoft initiatives orchestrate between cloud and edge instances [10, 17]. These limit users to the vendor's cloud and do not support interoperability. Our approach is cloud- and edge-neutral.

Existing orchestrators [1, 25, 28, 29, 35, 50, 53] focus on performance or resource utilization. ECO is the first system we are aware of that supports distributed ML/DL deployments using heterogeneous ML engines.

Dataflow-specific orchestrators [2, 44] are designed to track dependencies between components (or stages) within a data management pipeline. ECO is complementary to these approaches and can work above them.

8 Summary and Future Work

We described the challenges involved in edge/cloud ML and demonstrated an architecture for managing edge/cloud ML pipelines and dependencies. We use a graph-based overlay network approach (i.e., ION) to model pipeline dependencies and map IONs to analytic engines to enable a wide range of geographically distributed topologies, analytic engines, and ML patterns. Via this approach, we have met all of the challenges described in Section 3. Via the overlay network approach, ML computation can be distributed as needed to manage cost and performance, with other practical realities such as engine heterogeneity and disconnected operation supported. We do not explicitly discuss security in this paper, but since ECO is an application running on Edge and Cloud nodes, our approach layers on top of existing mechanisms for authentication and secure access. For future work, we intend to focus on further demonstrating the scale and flexibility of our ECO platform with different ML patterns, as well as better quantifying the performance of our Server and Agents.

References

[1] *CYCLONE: A Platform for Data Intensive Scientific Applications in Heterogeneous Multi-cloud/Multi- provider Environment* (Apr. 2016), Zenodo. Supported by CYCLONE project.

[2] Apache. airflow. Internet draft, 2017.

[3] Apache. flink. Internet draft, 2017.

[4] Amazon web services, 2018. https://aws.amazon.com.

[5] Dataiku, 2018. https://www.dataiku.com.

[6] Microsoft azure, 2018. https://azure.microsoft.com.

[7] Pipelineai, 2018. https://pipeline.ai.

[8] ABADI, M., AND ET AL. Tensorflow: Large-scale machine learning on heterogeneous systems. Internet draft, 2015.

[9] ALSHEIKH, M. A., LIN, S., TAN, H.-P., AND NIYATO, D. Toward a robust sparse data representation for wireless sensor networks. In *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN 2015)* (Washington, DC, USA, 2015), LCN '15, IEEE Computer Society, pp. 117–124.

[10] AMAZON. Internet Draft, 2018.

[11] ATZORI, L., IERA, A., AND MORABITO, G. The internet of things: A survey. *Comput. Netw. 54*, 15 (Oct. 2010), 2787–2805.

[12] BAYLOR, D., BRECK, E., CHENG, H.-T., FIEDEL, N., FOO, C. Y., HAQUE, Z., HAYKAL, S., ISPIR, M., JAIN, V., KOC, L., KOO, C. Y., LEW, L., MEWALD, C., MODI, A. N., POLYZOTIS, N., RAMESH, S., ROY, S., WHANG, S. E., WICKE, M., WILKIEWICZ, J., ZHANG, X., AND ZINKEVICH, M. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2017), KDD '17, ACM, pp. 1387–1395.

[13] BURKARD, C., AND LAGESSE, B. Analysis of causative attacks against svms learning from data streams. In *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics* (New York, NY, USA, 2017), IWSPA '17, ACM, pp. 31–36.

[14] CHATTERJEE, M., AND LEUSKI, A. CRMActive: An Active Learning Based Approach for Effective Video Annotation and Retrieval. In *Proceedings of ACM International Conference on Multimedia Retrieval (ICMR)* (Shanghai, China, June 2015), ACM, pp. 535–538.

[15] DAECHER, A. Internet of things: From sensing to doing. Wall Street Journal, 2016.

[16] DAI, W., YANG, Q., XUE, G.-R., AND YU, Y. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ICML '07, ACM, pp. 193–200.

[17] FAMILIAR, B. *Microservices, IoT, and Azure: Leveraging DevOps and Microservice Architecture to Deliver SaaS Solutions*, 1st ed. Apress, Berkely, CA, USA, 2015.

[18] FENG, L., KORTOÇI, P., AND LIU, Y. A multi-tier data reduction mechanism for iot sensors. In *Proceedings of the Seventh International Conference on the Internet of Things* (New York, NY, USA, 2017), IoT '17, ACM, pp. 6:1–6:8.

[19] GARTNER, I. Managing the data tsunami: How ai and edge computing will enhance iot analytics. Internet Draft, 2017. https://www.gartner.com/doc/3817038/managing-data-tsunami-ai-edge.

[20] GUBBI, J., BUYYA, R., MARUSIC, S., AND PALANISWAMI, M. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst. 29*, 7 (Sept. 2013), 1645–1660.

[21] HAMANN, H. F. From smart sensors to smarter solutions with physical analytics. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2015), SenSys '15, ACM, pp. 3–3.

[22] HIPP, D. R. Sqlite, 2015. https://www.sqlite.org/download.html.

[23] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2010), USENIX-ATC'10, USENIX Association, pp. 11–11.

[24] INFUXDB. Storage engine and time-structured merge trees (tsm), 2017.

[25] ISARD, M., PRABHAKARAN, V., CURREY, J., WIEDER, U., TALWAR, K., AND GOLDBERG, A. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 261–276.

[26] KING, M. C. Edge mlmachine learning on the edge, 2018. https://www.foghorn.io/edge-ml-machine-learning-edge/.

[27] KUMAR, A., GOYAL, S., AND VARMA, M. Resource-efficient machine learning in 2 KB RAM for the internet of things. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (2017), pp. 1935–1944.

[28] LAMPSON, B. W. A scheduling philosophy for multi-processing systems. In *Proceedings of the First ACM Symposium on Operating System Principles* (New York, NY, USA, 1967), SOSP '67, ACM, pp. 8.1–8.24.

[29] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems* (June 1988).

[30] LUAN, T. H., GAO, L., LI, Z., XIANG, Y., AND SUN, L. Fog computing: Focusing on mobile users at the edge. *CoRR abs/1502.01815* (2015).

[31] MA, C., SMITH, V., JAGGI, M., JORDAN, M. I., RICHTÁRIK, P., AND TAKÁČ, M. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37* (2015), ICML'15, JMLR.org, pp. 1973–1982.

[32] MAYER, C., MAYER, R., AND ABDO, M. Streamlearner: Distributed incremental machine learning on event streams: Grand challenge. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems* (New York, NY, USA, 2017), DEBS '17, ACM, pp. 298–303.

[33] MERKEL, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J. 2014*, 239 (Mar. 2014).

[34] METZEN, J. H., KIRCHNER, F., EDGINGTON, M., AND KASSAHUN, Y. Towards efficient online reinforcement learning using neuroevolution. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (New York, NY, USA, 2008), GECCO '08, ACM, pp. 1425–1426.

[35] OUSTERHOUT, J. Scheduling techniques for concurrent systems. In *Proceedings of the 3rd International Conference on Distributed Computing Systems* (1982), pp. 22–30.

[36] OVENDEN, J. Edge computing and the future of machine learning. Internet Draft, 2018.

[37] PAN, S. J. Transfer learning with applications on text, sensors and images. In *Proceedings of the 2Nd Workshop on Machine Learning for Interactive Systems: Bridging the Gap Between Perception, Action and Communication* (New York, NY, USA, 2013), MLIS '13, ACM, pp. 7–7.

[38] PAPAGEORGIOU, A., CHENG, B., AND KOVACS, E. Real-time data reduction at the network edge of internet-of-things systems. In *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)* (Washington, DC, USA, 2015), CNSM '15, IEEE Computer Society, pp. 284–291.

[39] POLYZOTIS, N., ROY, S., WHANG, S. E., AND ZINKEVICH, M. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (New York, NY, USA, 2017), SIGMOD '17, ACM, pp. 1723–1726.

[40] RAINA, R., BATTLE, A., LEE, H., PACKER, B., AND NG, A. Y. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ICML '07, ACM, pp. 759–766.

[41] SCHELTER, S., SOTO, J., MARKL, V., BURDICK, D., REINWALD, B., AND EVFIMIEVSKI, A. Efficient sample generation for scalable meta learning. In *2015 IEEE 31st International Conference on Data Engineering* (April 2015), pp. 1191–1202.

[42] SMITH, V., CHIANG, C., SANJABI, M., AND TALWALKAR, A. Federated multi-task learning. *CoRR abs/1705.10467* (2017).

[43] SMITH, V., CHIANG, C., SANJABI, M., AND TALWALKAR, A. Federated multi-task learning. *CoRR abs/1705.10467* (2017).

[44] SPOTIFY. Luigi, 2018. https://github.com/spotify/luigi.

[45] SRIDHAR, V., SUBRAMANIAN, S., SUNDARARAMAN, S., ROSELLI, D., AND TALAGALA, N. Model governance: Reducing the anarchy of production ml. In *Proceedings of the 2018 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2018), USENIXATC'18, USENIX Association.

[46] SWIM. Edge computing and machine learning: Two sides of the same coin. Internet Draft, 2018.

[47] SWIM. Swim edge learning. Internet Draft, 2018.

[48] TAYLOR, M. E., AND STONE, P. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ICML '07, ACM, pp. 879–886.

[49] TAYLOR, M. E., STONE, P., AND LIU, Y. Transfer learning via inter-task mappings for temporal difference learning. *J. Mach. Learn. Res. 8* (Dec. 2007), 2125–2167.

[50] VAVILAPALLI, V. K., MURTHY, A. C., DOUGLAS, C., AGARWAL, S., KONAR, M., EVANS, R., GRAVES, T., LOWE, J., SHAH, H., SETH, S., SAHA, B., CURINO, C., O'MALLEY, O., RADIA, S., REED, B., AND BALDESCHWIELER, E. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (New York, NY, USA, 2013), SOCC '13, ACM, pp. 5:1–5:16.

[51] YANG, Q. When deep learning meets transfer learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (New York, NY, USA, 2017), CIKM '17, ACM, pp. 5–5.

[52] YE, H., SHAO, W., WANG, H., MA, J., WANG, L., ZHENG, Y., AND XUE, X. Face recognition via active annotation and learning. In *Proceedings of the 2016 ACM on Multimedia Conference* (New York, NY, USA, 2016), MM '16, ACM, pp. 1058–1062.

[53] ZAHARIA, M., KONWINSKI, A., JOSEPH, A. D., KATZ, R., AND STOICA, I. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 29–42.

[54] ZAHARIA, M., XIN, R. S., WENDELL, P., DAS, T., ARMBRUST, M., DAVE, A., MENG, X., ROSEN, J., VENKATARAMAN, S., FRANKLIN, M. J., GHODSI, A., GONZALEZ, J., SHENKER, S., AND STOICA, I. Apache spark: A unified engine for big data processing. *Communications of the ACM 59*, 11 (Oct. 2016), 56–65.

[55] ZHANG, X., AND MAHOOR, M. H. Task-dependent multi-task multiple kernel learning for facial action unit detection. *Pattern Recogn. 51*, C (Mar. 2016), 187–196.