Research Statement

Shivaram Venkataraman

Machine learning methods power the modern world, from recommending new products to detecting supernovae in astrophysics. The ubiquity of massive data has made these algorithmic methods viable and accurate across a variety of domains. However the end of Moore's law and the shift towards distributed computing architectures presents a challenge to this trend. To meet the scale and performance requirements of the future, we need to develop efficient systems and scalable algorithms. My research focuses on designing machine learning frameworks that explicitly take into account these scale, algorithmic, and hardware challenges at the same time – jointly designing end-to-end systems for efficiency and performance.

In my dissertation research I address the challenges in large scale machine learning through three lenses. First, how can we build *systems* that exploit the parallelism and elasticity present in modern datacenters while providing resiliency? Second, given system constraints what new *algorithms* can better tackle large scale learning problems? And finally, what *abstractions* most efficiently let us express user intent, while providing the right primitives for optimized execution?

Systems: A number of large scale machine learning applications are deployed on shared cloud computing infrastructure. In order to appropriately provision such clusters, I developed a performance prediction framework [13] that can cheaply suggest the optimal configuration to use. Further to enable low latency execution, my work proposed scheduling techniques to reduce time spent in data access (KMN [11]) and coordination (Drizzle [12]). Specifically in Drizzle, we show how benefits from batch processing and streaming systems can be combined to achieve low latency during normal execution and fault recovery .

Algorithms: Based on constraints and trade-offs present in distributed systems, I have also contributed to the development of algorithms for large scale numerical optimization. These include algorithms that can handle large kernel learning problems that don't fit in cluster memory [6] and algorithms for linear systems that can converge faster [7] than existing methods.

Abstractions: Finally, with the adoption of distributed computing environments, it is important to have abstractions that make it easier to express new algorithms. My work on Presto [10] was the first distributed framework for the R language that enabled large scale learning with a easy to use, high-level API. I also helped create large scale machine learning libraries [3, 5, 16] that enable users to easily compose a number of ML operations.

Impact: My academic research has been productized by industry, incorporated into popular open-source projects, and is used in supercomputing centers for scientific applications. My work on Presto was commercialized by HP as a part of Vertica and SparkR is a widely used open source package. Similarly my work on machine learning libraries and the concept of machine learning pipelines is a part MLlib [3], the machine learning library in Spark. Finally, my work on low latency scheduling is being deployed on supercomputers at the Lawrence Berkeley National Lab (NERSC) to accelerate matrix factorization algorithms for mass spectrometry imaging and neutrino detection.

Future research at the intersection of computing systems and algorithms for machine learning is vital to translate the promise of big data into reality. With the evolution of datacenters to include new hardware devices like accelerators and the adoption of machine learning in feedback-driven workloads like robotics, there is a greater need for developing systems that can *adapt* to changes. Thus, I would like to study how we can design adaptable systems that can automatically choose algorithms, data layouts and hardware configurations based on the latency and statistical error requirements. Further, collaborations across areas is necessary to develop such end-to-end systems. My research in graduate school has been greatly enriched by collaborations with machine learning researchers at UC Berkeley, domain scientists at NERSC and industrial developers at a number of organizations like HP Labs and Intel. These collaborations helped me better understand problems in large scale data analysis and have also prepared me to collaborate effectively across disciplines in academia and with industrial organizations.

Systems for Efficient Execution

Along with the trend of large scale machine learning, the past few years have also seen applications switch to using cloud computing infrastructure. Using shared, elastic infrastructure brings about new challenges in executing applications efficiently and my work proposes system designs to address them.

Performance Modeling: Configuring and deploying applications in the cloud is one of the major challenges faced by users. This is because choosing the right hardware configuration can significantly improve performance and lower cost, but it is often unclear what the appropriate configuration is for a given workload. Thus there is a need for tools that can predict performance under various resource configurations and then choose the best among them. To address this, we developed Ernest [13], a performance prediction framework that requires very little training data. The key insight in Ernest is that a number of machine learning applications have predictable structure in terms of computation and communication. Thus we build performance models on small samples of data and then predict performance on larger datasets and cluster sizes. To minimize the time and resources spent in building a model, we use optimal experiment design, a statistical technique that allows us to collect as few training points as required.

Low-Latency Scheduling: Schedulers used in data processing frameworks are responsible for controlling what computation is executed where. The key goal for schedulers is to minimize the amount of time spent in accessing data; i.e. provide *data locality* while ensuring coordination overheads are not high.

To improve data locality, we studied data access properties of machine learning algorithms and found that most algorithms operate on a *sample* of the input data. Due to the use of sampling, there exists a large number of subsets of data that are all valid inputs to the job. Based on this observation, we developed KMN [11], a system that leverages these choices to perform data-aware scheduling. KMN not only uses choices to co-locate tasks with their input data but also percolates such combinatorial choices by launching a few additional tasks at every stage. This approach is helpful for minimizing the effect of stragglers in statistical applications and has been adopted by other machine learning systems like Tensorflow.

While centralized scheduling, used in batch processing systems like Spark, enables better task placement and fault tolerance, it can impose a high overhead for low latency applications. On the other hand streaming systems, such as Naiad or Flink, provide lower latency during normal execution but typically incur high latency while adapting to changes (e.g., fault recovery). We built Drizzle [12], a framework that combines the benefits of batch processing and streaming systems. Drizzle exploits the iterative properties of machine learning algorithms to achieve low latency, while retaining the fault tolerance properties of Spark. Our key insight in Drizzle is that we can decouple fine-grained execution from coarse-grained scheduling decisions. Based on this insight, we amortize scheduling overheads by grouping many iterations at once.

Scalable Algorithms

Existing machine learning algorithms often cannot scale to handle very large problems as they are unaware of systems constraints. By accounting for the execution environment we can develop more scalable algorithms. For example, while developing systems for machine learning we found that for very large problems like kernel SVMs existing iterative distributed optimization methods were insufficient as the input was too large to fit in cluster-wide memory. We developed algorithms [6] that only required a part of the input to reside in memory and thus can scale to kernel matrices up to 40TB in size. We also showed how kernel approximations like the Nystrom method can be efficiently executed in a distributed environment. For all the algorithms we also accounted for how the computation and communication costs change as the cluster size grows. We found that by studying the theoretical convergence rates and systems costs together, we could develop algorithms that made the most progress while making best use of the resources.

Recently we extended this work to devise accelerated algorithms [7] that have better convergence properties compared to classical algorithms like the Conjugate Gradient method. This work also explores the importance of sampling distributions in machine learning algorithms and shows how uniform random sampling can be much better than a fixed partitioning scheme. However implementing uniform random sampling is expensive in parallel settings and in the future I plan to study sampling schemes that are efficient to implement and have provably good convergence rates.

Abstractions for Machine Learning

Based on my experience in building systems and developing high performance algorithms, I learned that using an appropriate programming model was crucial for both improving usability and for achieving high performance. Early efforts in implementing large scale machine learning algorithms were based on the data parallel model provided by systems like MapReduce. However it is often cumbersome to write complex machine learning algorithms in data-parallel models. Many of these algorithms are best formulated as linear algebra operations on arrays. In Presto [10] we developed a distributed array-based abstraction which allows algorithm designers fine-grained control over computation and communication. We extended this work to build distributed data frames in SparkR [14]. Distributed data frames further allows users to do structured data processing and encode partition-aggregate workflows.

While distributed data structures are useful for expressing a single machine learning algorithm, a number of real-world applications are more complex and require the combination of multiple algorithms. For example a text classification program might featurize data using TF-IDF scores, then perform dimension reduction using PCA and finally learn a model using logistic regression. To address this, we developed the idea of machine learning pipelines [5] which allow users to compose simple operators. Along with this we also created a library of algorithms [3] and linear algebra operators [16] that are now a part of Apache Spark.

Beyond Analytics

While my dissertation is primarily focused on analytics applications, I am broadly interested in data processing systems and have also worked on systems for transaction processing (or OLTP) workloads.

Replication of data in distributed data stores presents a fundamental trade-off between latency and consistency. In Probabilistically Bounded Staleness [2] (PBS), we introduced a consistency model which provides expected bounds on data staleness with respect to wall clock time. Using PBS, we were able to measure the latency consistency trade-off for quorum-based systems like Cassandra [1]. Sharing distributed storage systems is also challenging while ensuring that service-level objectives (SLO) are met for throughput and latency-sensitive applications. In our work on Cake [15], we developed a coordinated, multi-resource scheduler that enforces SLOs in shared distributed storage systems.

Prior to my research at Berkeley, I studied the design of storage systems for non-volatile byte-addressable memory as a part of my masters thesis [8] at UIUC. My work proposed Consistent and Durable Data Structures (CDDSs) [9], a single-level data storage design that allows programmers to safely exploit the low-latency and non-volatile aspects of new memory technologies.

Future Research

Future research at the intersection of computing systems and machine learning algorithms is necessary to handle changes in hardware and evolution of workloads. Some of the research problems I plan to tackle include:

Declarative Abstractions for Machine Learning: Designing large scale machine learning applications is challenging and requires intricate knowledge of the domain, statistics and systems characteristics. Further, implementations of machine learning applications are at a low-level and describe *how* they should be executed rather than *what* should be executed. The main reason for this is that the appropriate machine learning algorithm to use often changes based on the data and hardware being used. The adoption of heterogeneous hardware like accelerators and asynchronous algorithms (e.g., HOGWILD!) further exacerbates this problem. My goal is to develop declarative machine learning abstractions that can capture the intent of a wide variety of applications and correspondingly build systems to automatically optimize execution for various hardware targets. Our work in developing the KeystoneML optimizer [5] is an initial step towards this goal. I also plan to investigate new *adaptable algorithms* that can tune their execution based on cost and latency requirements. In my recent work on Hemingway [4], we proposed techniques to model the convergence rates of machine learning algorithms and this approach can be used to develop adaptable algorithms in the future.

Systems for Heterogeneous Hardware: The evolution of datacenter hardware is leading to the adoption of technologies like GPUs, non-volatile memory (NVM), battery-backed DRAM and disaggregated 100Gbps networks. These

changes are poised to herald a new rack-scale computing era. While my earlier work [9] looked at how OLTP systems could capitalize on non-volatile memory, it is unclear how analytics frameworks will adapt to these changes. Specifically existing large scale data processing frameworks are often optimized for the shared-nothing architecture and assume that network access is always expensive. I plan to develop new system designs that can effectively utilize the performance improvements promised by next generation hardware while ensuring that the resilience and resource sharing provided by existing frameworks is maintained.

References

[1] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. PBS at Work: Advancing Data Management with Consistency Metrics. In *SIGMOD*, 2013. Demo.

[2] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Quantifying Eventual Consistency with PBS. *VLDB Journal*, 2014.

[3] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[4] X. Pan, S. Venkataraman, Z. Tai, and J. Gonzalez. Hemingway: Modeling Distributed Optimization Algorithms. In *Machine Learning Systems Workshop (Co-located with NIPS)*, 2016.

[5] E. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. *CoRR*, abs/1610.09451, 2016.

[6] S. Tu, R. Roelofs, S. Venkataraman, and B. Recht. Large Scale Kernel Learning using Block Coordinate Descent. *arXiv*, 1602.05310, 2016.

[7] S. Tu, S. Venkataraman, A. Wilson, A. Gittens, M. Jordan, and B. Recht. Accelerating block Gauss-Seidel with random coordinate selection. http://shivaram.org/drafts/acc-gs.pdf, 2016. Under review.

[8] S. Venkataraman. Storage system design for non-volatile byte-addressable memory using consistent and durable data structures. Master's thesis, University of Illinois, Urbana-Champaign, May 2011.

[9] S. Venkataraman, N. Tolia, P. Ranganathan, and R. H. Campbell. Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory. In *FAST*, San Jose, CA, Feb. 2011.

[10] S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung, and R. S. Schreiber. Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. In *Eurosys*, 2013.

[11] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica. The Power of Choice in Data-Aware Cluster Scheduling. In *OSDI*, 2014.

[12] S. Venkataraman, A. Panda, K. Ousterhout, A. Ghodsi, M. J. Franklin, B. Recht, and I. Stoica. Drizzle: Fast and Adaptable Stream Processing at Scale. http://shivaram.org/drafts/drizzle.pdf, 2016. Under review.

[13] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *NSDI*, 2016.

[14] S. Venkataraman, Z. Yang, D. Liu, E. Liang, H. Falaki, X. Meng, R. Xin, A. Ghodsi, M. Franklin, I. Stoica, and M. Zaharia. SparkR: Scaling R Programs with Spark. In *SIGMOD*, 2016.

[15] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica. Cake: Enabling high-level SLOs on Shared Storage Systems. In *SoCC*, 2012.

[16] R. Zadeh, X. Meng, A. Ulanov, B. Yavuz, L. Pu, S. Venkataraman, E. Sparks, A. Staple, and M. Zaharia. Matrix Computations and Optimization in Apache Spark. In *KDD*, 2016.