Research Statement

Matei Zaharia

Computer systems are undergoing a fundamental shift, as growing volumes of users and data, and stalling processor speeds, force applications to scale out to large clusters. Many of today's most interesting applications, such as web search, social networks, and Siri, run on clusters of thousands of nodes; and growth in web and business data means that even small startups and traditional enterprises need to run computations at scale. My research has been on algorithms, programming models and systems for this new cluster computing setting. I believe that this setting presents not only challenging problems that can shape the future computing stack, but also an opportunity to revisit traditional operating system questions (*e.g.,* how applications acquire resources), and rich analytical questions that extend to other systems. My work has ranged from analytical study of some of these questions, to algorithms and artifacts that are now in wide commercial use.

My main research projects have been along two lines: (1) designing a computing stack for rich large-scale data analysis (consisting of the Mesos cluster manager, Spark parallel runtime, Spark Streaming engine, and Shark SQL processor), and (2) analyzing several fundamental scheduling problems that arise in large clusters, including multi-resource fairness, data locality and straggler mitigation. In addition, I have taken on projects in traditional networking, client security, and, more recently, DNA sequence analysis, a "big data" problem where new algorithms and insight from computer systems can yield dramatic speedups.

Throughout these projects, my research approach is to identify the most general version of a new problem possible, and search for solutions that are elegant, widely applicable and highly implementable. I strongly believe in prototyping ideas in real systems, and to this end, I have made virtually all of my work open source. I am fortunate that most of my projects—including Mesos, Spark, Shark, and the various scheduling algorithms—have been adopted by commercial users, some at dozens of companies. These users provide not only validation for the ideas, but a fantastic means for discovering new research problems.

Data Analytics Systems

Most of my system-building research has been on a next-generation data analytics stack that addresses some of the fundamental challenges arising in current systems. While batch processing systems like Hadoop and Dryad have been widely successful, it quickly became clear that future stacks will need (1) more effective resource sharing among the increasing number of cluster applications and (2) more powerful programming abstractions that can also handle complex algorithms, interactive queries, and stream processing.

MESOS is my project to address the first problem: resource sharing among cluster applications. The problem is that the applications that run on clusters (*e.g.,* web services, storage systems and batch jobs) are growing increasingly diverse, and dynamically sharing resources across these applications is crucial for cost-efficiency. However, designing a central scheduler that is both scalable and flexible enough to handle all applications' constraints is difficult. Instead, Mesos proposes a decentralized scheduling model called *resource offers*, wherein each application controls *which* resources it accepts, but Mesos still controls *how many* resources to offer each application. We showed that resource offers allow for a far more scalable scheduler (Mesos can manage up to 50,000 nodes), while still enforcing common inter-application sharing policies, such as priority, and letting applications meet placement goals such as data locality near-optimally [3].

SPARK tackles the second problem, of a more general programming model. The popular MapReduce model was quickly found to be inefficient for three important types of applications: *multi-pass* algorithms, such as iterative machine learning; *interactive* queries; and *online processing*. While previous work developed specialized programming models for some of these applications (*e.g.,* Google's Pregel for iterative graph algorithms), we observed that a common need in all these workloads was for efficient *data sharing* across parallel operations: for instance, sharing results across iterations of a learning algorithm, or across ad-hoc queries. Spark introduces a distributed memory abstraction called *resilient distributed datasets (RDDs)* that supports a wide range of applications and provides low-cost fault tolerance through a novel lineage-based

mechanism (remembering the deterministic operations that built a dataset instead of replicating the data). We showed that Spark can express many existing specialized models (*e.g.*, Pregel or SQL), attaining similar speeds while offering fault recovery guarantees that they sometimes lack, and can be 100× faster than Hadoop for iterative and interactive workloads. More fundamentally, RDDs show that these diverse programming models can be *unified* in one engine, allowing applications to seamlessly compose them [8].

SPARK STREAMING further generalizes RDDs to support to low-latency stream processing. Previous streaming systems, such as Telegraph and Borealis, either did not provide fault tolerance, or did so in an expensive manner involving replication of the system. To scale stream processing to an order of magnitude larger clusters, we sought a design that recovers quickly from faults *without* replication. Spark Streaming achieves this by "discretizing" stream processing into a series of short, *deterministic* parallel jobs, and storing state between these in RDDs. When a node fails, it recomputes the lost RDD partitions in parallel across the cluster. The system achieves sub-second latency and similar per-node performance to commercial systems, while scaling linearly to hundreds of nodes and recovering from faults within a second. It interacts seamlessly with Spark's batch and interactive queries, creating a powerful unified platform for analytics [9].

Finally, **SHARK** builds a fast and fault-tolerant SQL engine over Spark. Traditional parallel databases have not supported mid-query fault tolerance, focusing on pipelining for speed; and it was thought that a MapReduce-like approach, where queries run as a graph of deterministic tasks, is inefficient for SQL. Shark shows that using RDDs, column-oriented storage, and careful data placement, a MapReduce-like engine can match or exceed the speedups seen for parallel databases over Hadoop, while offering fast fault recovery properties that these systems lack. It opens an attractive new design point for analytic databases [5].

These systems are all open source, and have already seen industry applications. Mesos is being used to manage over 2,500 machines at Twitter, while Spark and Shark are used in production or late-stage prototyping at a number of startup companies and at Yahoo!'s ad analytics team.

Scheduling Policies

Apart from system design challenges, cluster computing poses several interesting resource allocation problems, some of which I have also applied to other types of systems.

MULTI-RESOURCE FAIRNESS: My most recent scheduling work was on generalizing weighted fair sharing (also known as proportional sharing) to multiple resource types. When we designed Mesos, we quickly saw that different applications needed different amounts of different resource types (*e.g.*, CPU time or memory). While weighted fair sharing is a natural policy to schedule one resource, it turns out that generalizing it to multiple ones is challenging, due to problems that do not arise in the single-resource case. Several natural generalizations do not preserve axiomatically "fair" properties, such as ensuring that each user in an n-user cluster gets at least $\frac{1}{n}$ of some resource, while others allow users to *cheat* and inflate their shares by overstating their requirement for one resource (*e.g.*, schedulers that try to maximize utilization of all resources may try to bin-pack user demands). We analyzed several policies and proposed *dominant resource fairness (DRF)*, a policy that keeps many properties of single-resource fairness [2]. DRF is the default policy in Mesos, and has also been implemented independently in Hadoop 2.0's "YARN" resource manager.

Multi-resource fairness is an example of a problem that also applies beyond clusters. For instance, we have generalized DRF to multiplex resources in *time*, similar to fair queueing algorithms for network links. The resulting algorithm, *dominant resource fair queueing (DRFQ)*, is useful in software routers, middleboxes, and hypervisors, where different flows or VMs consume different resources [1]. Another generalization, *Choosy*, considers constraints, where each user can only run on a subset of nodes (*e.g.*, those with public IPs).

STRAGGLER MITIGATION: My work on *longest approximate time to end (LATE)*, an algorithm for detecting slow nodes (stragglers) in MapReduce, was the first academic work on this problem, and spawned a rich set of follow-on research. Our implementation of LATE is also now incorporated in Hadoop [10].

DATA LOCALITY: Strictly scheduling work according to fair sharing greatly reduces data locality in clusters, by limiting the set of nodes each job can launch on. I explored an extremely simple algorithm to avoid this

problem: just *delay* scheduling each job for a short time (say, up to 1 second) to wait for a node containing one of its input files to become free. I showed, both analytically and in real clusters, that this simple algorithm attains nearly *perfect* data locality, while negligibly affecting fairness, and will improve in performance with hardware trends (*e.g.*, growing core counts). Delay scheduling represents my favorite type of analytic research: a simple, but mathematically grounded, algorithm that is eminently implementable. It has now been implemented independently in Hadoop 2.0 and Facebook's Corona system, while my own implementation, the Hadoop Fair Scheduler, is one of the most widely-used schedulers for Hadoop [7].

Other Projects

Computer Security: In a departure from server-side research, my work on Cloud Terminal looks at how to improve client security in the era of the cloud [4]. We observe that most security-sensitive applications, such as online banking, are web-based, and only employ the user's machine as a terminal (*e.g.*, a browser). We use this observation to design a minimal trusted computing base (TCB) system for accessing sensitive applications. Cloud Terminal temporarily takes over the display and keyboard from the host OS by installing itself as a hypervisor, and acts as a thin VNC terminal for an application rendered entirely in the cloud (*e.g.*, by a browser). The result is a 23,000-line TCB system that provides secure access to general applications, can authenticate itself to servers via remote attestation, and works even on machines infected by malware.

DNA Sequence Analysis: In the past year I also became interested in an emerging "big data" application domain: DNA sequencing. DNA sequencing is falling in cost faster than Moore's law, with the cost for a human genome expected to soon reach $1000. Processing this data poses interesting algorithmic and systems challenges, as the data is large (300 GB for a human genome), and comes in the form of short 100-character *reads* that must be combined, like puzzle pieces, into a full genome. My first project looked at the first analysis step, *alignment*, which maps each read to the closest substring in a reference genome. Current algorithms take multiple CPU-days to align one genome. Our project, the Scalable Nucleotide Alignment Program (SNAP), is 10–100× faster, and also *more* accurate than most tools. It achieves this through a novel pruning algorithm and a design that carefully leverages modern hardware (*e.g.*, minimizes cache misses) [6].

Another ongoing project exploits *similarity* in the genome to improve analysis. In profiling SNAP, we found the main source of both slowdown and error was reads that align well to hundreds of genome locations, where each location must be found and compared with. We are addressing this problem by (1) using a distributed algorithm to *identify* clusters of similar regions in advance, and (2) designing *"similarity-exploiting algorithms"* that match faster against a group of similar strings than testing each one separately, by sharing subcomputations on identical substrings. While this work is ongoing, early results show that similar regions explain most of SNAP's errors, and similarity-exploiting algorithms can be 9× faster for such regions.

Future Research

In future work, I plan to build on my experience in large-scale data analytics, and to expand my work to other areas of computer systems and cloud computing. Some areas I am interested in include:

Cluster Computing: If the datacenter is the new computer, what should be its operating system? This question summarizes the way I would like to approach cluster systems. As clusters start being used for more and more applications, I believe that many of the problems that motivated traditional OSes will arise in this setting, and taking an OS perspective can lead to long-term solutions. For example, Mesos showed that there is a need for dynamic resource sharing across cluster applications, analogous to the sharing in a single-node OS; and RDDs show that there are data sharing abstractions that work across a wide variety of applications. However, numerous questions remain. For instance, as in-memory computing becomes more popular, what should a cluster memory manager look like? How should it allocate memory across multiple Spark applications, and evict RDDs when it runs out? Similarly, when applications have different priorities, how should a scheduler evict tasks to make room for more important ones? These questions are arising in real Mesos, Spark and Hadoop deployments, and require both new abstractions and analytical work.

DEBUGGING: One of the clearest needs from talking to Hadoop and Spark users is for debugging tools for large applications. Users have a difficult time understanding the correctness and the performance of distributed programs. One approach I am currently exploring is to leverage the deterministic nature of most cluster computing models to let users selectively *replay* part of the computation at low cost.

RESOURCE SHARING: My work on DRF showed that there is considerable subtlety in defining weighted fair allocations for multiple resources, and that basic properties of the policies we use for one resource can be violated. While we have done some work (Choosy) to get a similar sharing policy for jobs with constraints, I would like to find a much more general concept of a "resource," and a general policy for this context.

UNIFIED ANALYTICS: Spark Streaming shows that it is possible to combine streaming, batch and interactive queries in the same engine, which is highly attractive in practice. For example, users can run ad-hoc queries over stream state from Spark's interactive shell, or join streams against data computed offline. I believe that such unified systems will be essential in the future, as most "big data" applications (*e.g.*, those using social or business data) provide significant value from running in real time. However, there are major challenges to making such a system practical, such as prioritizing computation across streaming and ad-hoc queries to respect deadlines, and automatically determining how to share results across queries.

Overall, my philosophy is to find the fundamental problems in new systems, and search for highly practical solutions. While I do not expect all my projects to have the same deployment success as Mesos and Spark, I also hope to continue building deployable algorithms and systems. It has been some time since systems research could truly affect practice on a broad scale, as in the early days of the Internet or UNIX, and I believe that cluster computing affords the greatest opportunity to do so yet. The combination of rapidly emerging problems and friendliness to open source allows for rich, intellectually rewarding research that can directly shape the platforms powering some of today's most exciting applications.

References

[1] Ali Ghodsi, Vyas Sekar, **Matei Zaharia**, and Ion Stoica. Multi-resource fair queueing for packet processing. In *SIGCOMM*, 2012. (BEST PAPER AWARD).

[2] Ali Ghodsi, **Matei Zaharia**, Benjamin Hindman, Andrew Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.

[3] Benjamin Hindman, Andrew Konwinski, **Matei Zaharia**, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[4] Lorenzo Martignoni, Pongsin Poosankam, **Matei Zaharia**, Jun Han, Stephen McCamant, Dawn Song, Vern Paxson, Adrian Perrig, Scott Shenker, and Ion Stoica. Cloud terminal: Secure access to sensitive applications from untrusted systems. In *USENIX ATC*, 2012.

[5] Reynold Xin, Joshua Rosen, **Matei Zaharia**, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: SQL and rich analytics at scale. *To appear at SIGMOD 2013; draft at* `tinyurl.com/shark-tr`. (BEST DEMO AT SIGMOD 2012).

[6] **Matei Zaharia**, William J. Bolosky, Kristal Curtis, Armando Fox, David Patterson, Scott Shenker, Ion Stoica, Richard M. Karp, and Taylor Sittler. Faster and more accurate sequence alignment with SNAP. *CoRR*, abs/1111.5572, 2011.

[7] **Matei Zaharia**, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys*, 2010.

[8] **Matei Zaharia**, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012. (BEST PAPER AWARD).

[9] **Matei Zaharia**, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: A fault-tolerant model for scalable stream processing. *UC Berkeley Technical Report UCB/EECS-2012-259*, 2012.

[10] **Matei Zaharia**, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving MapReduce performance in heterogeneous environments. In *OSDI*, 2008.