Research Statement | Laurent Bindschaedler

Building the next generation of massive-scale data management systems at the intersection of operating systems, databases, and machine learning.

The demand for large-scale data-intensive computing has exploded over the past decade, triggered by new applications ranging from retail business intelligence to Internet-scale services such as Facebook and Google and the steadfast development of real-time analytics and machine learning techniques. Unfortunately, these advances have placed incredible strain on existing systems that must now face an ever-growing list of requirements and challenges. Datasets and workloads exhibit skew, and machines are heterogeneous, requiring reliable solutions to mitigate load imbalance and maximize parallelism in the presence of stragglers. More and more users need interactive applications with real-time insights. Workloads vary, and datasets keep changing in hard-to-predict ways. Algorithms and data models are constantly updated to meet new business objectives, and software ages, requiring frequent maintenance. As a result, even small organizations nowadays employ dedicated teams and typically spend a large fraction of their IT budget for the sole purpose of data management and analysis.

My research addresses core systems problems in large-scale, distributed data processing. My dissertation work at EPFL LABOS tackled the load imbalance problem, where some machines take longer than others to complete their assigned work due to skew and resource contention. Load imbalance is the bane of distributed systems as it limits the achievable parallelism. I proposed a new software architecture that mitigates load imbalance and allows these systems to maximize resource utilization and parallelism. I successfully used this architecture in several application areas ranging from data analytics to transactions in distributed key-value stores, which, unlike data analytics, are generally user-facing and require low tail latency in addition to high throughput. More recently, I explored real-time graph pattern mining in the presence of high-throughput streams of updates. Since real-world graphs are large and pattern mining is costly, I developed algorithms and systems techniques to efficiently recompute the results depending on changes in the input graph and improve performance by orders of magnitude. Finally, my postdoctoral work at MIT CSAIL focused on "learned" systems that automatically synthesize components optimized to a specific problem instance using machine learning techniques, thereby enabling the creation of bespoke data analytics systems without expensive and frequent manual tuning.

My research is experimental: it involves building systems and evaluating them. I draw much inspiration from industry, talks, meetups, personal interaction with other researchers, and social media. Most of my research so far has focused on challenges that real systems face, directly gathered from the research community and practitioners. The industry experience that I acquired before my graduate studies, mainly through my startup, also weighs heavily in my approach to problem selection. I take a principled, systems-oriented approach to understanding and solving problems: measuring and understanding limitations of existing systems, building quick prototypes to validate observations, and finally designing new abstractions to generalize the solution. I am also a firm believer in open-source and open research and make all my research artifacts readily available to researchers and practitioners, particularly all source code[1]. Finally, I aim to make scientific research and contributions more accessible to the general public, which I have done so far through blog posts[2] and science communication movies[3].

The rest of statement describes the three main projects I worked on so far and outlines my future research plans.

Mitigating Load Imbalance in Cluster Applications

Load imbalance in distributed applications refers to situations where different machines take different amounts of time to finish their assigned tasks, wasting resources and limiting parallelism as the other machines remain idle. Load imbalance takes various forms and has many possible causes, including skewed data partitioning, variance in the amount of generated intermediate state, data-dependent processing times or filtering, irregular memory accesses, hardware heterogeneity or failures, and interference from background tasks. Therefore, improving load balance is crucial for application developers and cluster operators as more balanced systems generally benefit from higher performance, faster job completion times, and better overall resource utilization.

My thesis proposed *Scatter computing*, a novel architecture for distributed data management and analytics systems that mitigates the performance impact of load imbalance [1]. Intuitively, addressing load imbalance requires decoupling tasks from the machines processing them, allowing the system to pool resources and respond to changing load

[1] https://github.com/bindscha
[2] https://binds.ch/blog
[3] https://www.imdb.com/name/nm9978952

conditions. The Scatter architecture entails rethinking distributed storage abstractions to avoid storage hotspots and designing coordination-avoidance techniques to enable multiple machines to share the work of processing a single task with low synchronization overhead. As a result, Scatter systems are highly resilient to imbalanced situations and often perform better than traditional systems in uniform cases.

I used the Scatter architecture to address load balance in a diverse set of applications, including graph analytics, general-purpose analytics, distributed databases, and machine learning, as described in the following sections.

Scale-out Graph Processing from Secondary Storage [Paper] [Slides] [Video] [Code]

Problem The availability of graph-structured data in domains ranging from social networks to national security has created renewed interest in designing systems to mine valuable information from such graphs. A serious impediment to this effort is that real-world graphs have skewed vertex degree distributions, and many graph algorithms exhibit irregular access patterns, making it hard to partition data and work. Moreover, the fast growth of graphs exacerbates these difficulties as datasets may not fit in the aggregate memory of all machines in the system.

Solution and Impact I designed Chaos, a scale-out graph processing system for secondary storage that enables analytics on very large graphs with trillions of edges [7]. Following the Scatter architecture principles, Chaos does away with elaborate partitioning schemes and instead spreads the data for each partition uniformly at random across all machines. We achieve high performance and load balance by leveraging this uniform data placement and a work-stealing strategy that allows multiple machines to process a single partition. As a result, Chaos is faster than competing systems by over an order of magnitude on many graph workloads and datasets. Chaos also pushed the limit of the graph size that can be processed in a cluster of commodity servers by analyzing the connectivity of a graph with over 250 trillion edges on 20 machines in a little over 10 hours, a feat previously only achieved by supercomputers. Five years later, Chaos still holds the 5th position in the Graph500 ranking of the largest graphs processed by computer.

Taming Skew in Large Scale Analytics [Paper] [Slides] [Video] [Code]

Problem Application runtimes in distributed data analytics frameworks such as Apache Hadoop and Spark are often unpredictable and underperforming on many input datasets and deployments due to, e.g., data and compute skew, slow or faulty machines, cluster heterogeneity. This load imbalance introduces stragglers that cause other machines to sit idle, degrading performance for the entire parallel job. The underlying cause for these issues is static work partitioning, i.e., creating fixed-size tasks that cannot be dynamically broken down. Since these systems inherently require that a single worker process each task, they have little recourse but wait for stragglers.

Solution and Impact I built Hurricane, a high-performance general-purpose analytics system inspired by the Scatter architecture that generalizes the work-stealing strategy of Chaos to design an adaptive task partitioning scheme that achieves fast execution times and high cluster utilization [5]. Hurricane introduced a new "data bag" storage abstraction and a task cloning strategy that allows workers to share a task and automatically adapts parallelism at runtime. Therefore, Hurricane provides orders-of-magnitude performance improvements in skewed workloads without introducing performance degradation in uniform workloads and datasets. After the paper's publication, I worked with a team of students from the University of Toronto to implement a fully productized open-source version of Hurricane.

Disaggregation for Distributed LSM-based Databases [Paper] [Slides] [Video] [Code]

Problem Distributed databases that use Log-Structured Merge-Tree (LSM) storage engines such as RocksDB often suffer from unpredictable performance and low utilization due to skew and background operations. Skew causes CPU and I/O imbalance, which degrades overall throughput and response time. Current LSM-based databases address skew by resharding data across machines. However, this operation is expensive because it involves bulk data migration, which affects throughput and response time for users. Similarly, background operations such as flushing and compaction can cause significant I/O and CPU bursts, leading to severe latency spikes, especially for queries spanning multiple machines such as range queries or transactions. These problems are hard to address in existing systems because the storage engines operate independently of each other and thus are unaware of resource usage and background operations on different machines.

Solution and Impact I designed Hailstorm, an LSM-optimized distributed file system that runs transparently below database deployments and reduces storage contention while providing cluster awareness to LSM storage engines [2]. Hailstorm automatically rebalances data from each storage engine across machines, reducing storage contention and enabling overloaded machines to offload background tasks. The Hailstorm design demonstrates the benefits of my Scatter architecture in user-facing systems. Using Hailstorm, I doubled the throughput of the widely popular MongoDB database in write-intensive workloads and multiplied scan throughput by over 20×. Hailstorm also achieves over 2× faster throughput for PingCAP's TiDB on industry-standard benchmarks TPC-C and TPC-E.

Alleviating Degradation with Non-IID Data in Machine Learning (Under Submission)

Problem Stochastic Gradient Descent (SGD) is a popular optimization algorithm that is used by a variety of machine learning applications due to its fast learning rate and small memory footprint. However, SGD is notoriously hard to parallelize in traditional MapReduce environments due to its sequential nature and high communication overheads.

Moreover, most parallel implementations of SGD, especially Apache Spark and Parameter Server, assume independent identically distributed data (IID) across partitions. Unfortunately, this assumption does not always hold in practice, especially in the presence of skewed samples or when the partitioning across workers itself exhibits skew. In such cases, the SGD algorithm may suffer from significant performance degradation or fail to converge.

Solution and Impact I built Snowball, a distributed load-balanced implementation of SGD. Snowball supports asynchronous gradient updates through a distributed key-value store and uses a data bag abstraction to allow parallel workers to sample from the entire dataset without partitioning it, reducing the impact of skew. Snowball leverages the Scatter architecture to mitigate a different type of skew in a seemingly perfectly load-balanced situation: all workers always perform exactly the same amount of work, but some workers perform unnecessary or counterproductive work. As a result, Snowball achieves high compute and storage utilization on all machines in the cluster while ensuring faster convergence even in the presence of significant skew. Snowball's shared data bag design also dampens the effects of asynchronous weight updates that often impact Parameter Server implementations.

Mining Graph Patterns in Real-Time [Paper] [Slides] [Video] [Code]

Extracting insight from streams of data is becoming a must-have feature in many systems that support business decisions. Indeed, periodic recomputation on snapshots of the data is wasteful and often too slow to support interactive data analysis. I explored real-time analytics in the context of graphs.

Problem: Graph pattern mining, i.e., finding instances of interesting patterns (matches) in a graph dataset, has wide-ranging applications in social networks, chemistry, credit card fraud detection, and semantic web. Since graph mining applications can easily take hours to run, even on modest-sized graphs, it is desirable to incrementally maintain the set of all matches as the graph is updated instead of recomputing everything from scratch upon receiving updates. Mining dynamic graphs introduces a new set of challenges. First, new algorithms are necessary to ensure correctness under updates and not miss any matches or produce duplicates. Second, a different software architecture is required to support high-throughput streams with thousands to millions of updates per second while maintaining mining results in real-time. In particular, this architecture must assign updates to workers in a balanced fashion while minimizing data transfer and synchronization across workers at scale.

Solution and Impact I designed Tesseract, a distributed graph mining system that automatically executes any existing static algorithm in dynamic graphs [4]. Tesseract introduces a novel change detection algorithm that efficiently determines the exact modifications for each update. Moreover, by favoring task parallelism over data parallelism, Tesseract can decompose a stream of graph updates into per-update mining tasks and dynamically assign these tasks to a set of distributed workers. Tesseract supports millions of updates per second with low latency and is several orders-of-magnitude faster than periodic recomputation on snapshots. Finally, Tesseract outperforms static mining systems on the entire graph, despite the overheads of supporting graph updates, thanks to its low memory requirements, efficient storage, and communication.

Benchmarking "Learned" Systems [Paper] [Slides] [Video]

The use of machine learning to tune data management systems or synthesize components tailored to a specific problem instance has recently become a popular research direction. The promise of learned systems is their ability to automatically adapt to new workloads, data, or hardware without time-consuming tuning by humans, thereby dramatically reducing the cost and accessibility of data analytics. During my postdoc at MIT, I set out to understand the characteristics of these new systems and propose some new approaches to analyze and evaluate them.

Problem: Although learned systems and components have shown orders-of-magnitude performance improvements under laboratory conditions, it remains largely unclear how these numbers will hold up in more realistic production environments. In particular, traditional benchmarks such as TPC or YCSB that evaluate performance under a stable workload and data distribution are insufficient to characterize these systems due to the latter's ability to overfit to the benchmark. As a result, companies are often reluctant to incorporate these techniques in mainstream systems due to a lack of evidence of how they would perform under varying conditions.

Solution and Impact: I have presented several ideas for designing new benchmarks that are better suited to evaluate learned systems [3]. New benchmarks should abstain from using fixed workloads and data distributions as their characteristics are easy to learn. Similarly, they should strive to measure adaptability through descriptive statistics and outliers rather than average metrics that "hide" too much information. I also proposed techniques and metrics to incorporate model training and cost savings into benchmark results, two key characteristics that can no longer be ignored in learned systems. We are currently collaborating with Microsoft Research and other researchers on this project, and I have implemented some of these ideas in a benchmark prototype.

Future Work: Next Generation Data Management Systems

Today's data management and analytics systems are plagued by problems such as stragglers, frequent updates to data, changing workloads, high costs of maintenance, and lack of flexibility. Attempts to address these limitations have led researchers and practitioners to design and deploy many systems optimized for various requirements. But, unfortunately, these systems are highly brittle: they cannot generalize to different applications, only work in relatively static workload conditions and execution contexts, and rarely support live updates.

How should we build the next generation of large-scale data management systems at the confluence of real-time analytics and learned systems? Moreover, how can we guarantee good performance for these new systems in unfavorable conditions, e.g., skew or resource contention? As a new faculty member, I plan to explore every aspect of this problem and provide the necessary building blocks for future data management systems. **I want to design *general* systems that just work without extensive manual tuning.** Such systems would *unify different paradigms* in a single system, efficiently *support updates*, and remain highly *scalable and predictable* under changes. At a high level, this line of research first entails understanding and characterizing the real-world conditions (data, workload, hardware) in which these systems are expected to run. Second, I will leverage machine learning techniques to design systems and components that can self-tailor to a specific problem instance without manual intervention. Finally, I will apply algorithmic and systems techniques to ensure generality, reliability, and predictability for these systems.

Below is a short description of two concrete research projects I plan to pursue. In the short term, I plan to investigate self-tailoring storage systems that can adapt to changes in data, workload, and hardware and automatically reconfigure themselves to meet application requirements. There are currently significant open challenges in this area, particularly in data layout, partitioning, caching, indexing, maximizing parallelism, and supporting updates efficiently. I also intend to expand my research to entire self-optimizing applications in distributed real-time analytics, graph processing, and machine learning applications. The wide acceptance of such applications among practitioners will largely depend on their reliability and autonomy.

Workload-Driven Database Storage

Current databases generally rely on designers to define the storage layout and partitioning of each table. Unfortunately, the exact workload for a database is hardly ever known ahead of time, and production workloads evolve, resulting in database designers often selecting "safe" layout defaults, such as clustered primary key. I plan to analyze and characterize database workloads to collect high-level statistics from production environments. I will then identify important signals and patterns in these high-level statistics to build a storage layout advisor to select the best-suited layout for a specific workload. A key challenge for this system will be minimizing storage reconfiguration overhead as the workload changes and data is updated. I am in the process of validating the basic idea using production workloads from Microsoft Research during my postdoctoral research and hope to cross-validate these results with more industrial partners soon. Eventually, I want to design a self-tailoring database storage engine that can optimize I/O performance through partial materialization, caching, and replication.

Unified Bespoke Graph Processing

Efficiently executing different graph workloads within the same system is challenging as graph applications exhibit widely varying characteristics. For example, analytics and graph mining applications rely heavily on sorted adjacency lists that do not play well with graph queries and transactions. Graphs are also notoriously hard to partition across machines. As a result, most graph processing systems target specific workloads and use specialized data structures, requiring costly extract-transform-load and partitioning upon updates or execution of a different application. It is not uncommon for these pre-processing steps to take longer than the actual computation [6]. I will build a graph processing system that can efficiently execute different applications by automatically fine-tuning configuration and synthesizing optimized components. Besides, graphs intrinsically contain locality information, enabling the design of workload-driven caching strategies across the storage hierarchy or customizing the type of traversal technique (breadth-first or depth-first) based on the specific algorithm and characteristics of the graph. Similarly, other aspects and components of graph processing systems are amenable to runtime optimization, e.g., type of parallelism used (data vs. task), scheduling and message passing (vertex relabeling, high-degree vertex prioritization, asynchronous vs. synchronous computation), or aggregation of results. A key challenge for this line of research will be to identify which components to target for maximum effectiveness. Another challenge is designing APIs to capture the potential for performance acceleration through machine learning techniques. Finally, certain components will likely need to be co-designed to avoid interference. As part of this project, I will also work on mining anomalies and gathering insights from streaming graph data. Fast anomaly detection is essential for people and organizations that rely on data to make decisions and has many applications, e.g., understanding/stopping the spread of "fake" news through social media and finding causes of failures or performance bugs in large networks and infrastructure.

Bibliography

[1] Laurent Bindschaedler. 2020. *An Architecture for Load Balance in Computer Cluster Applications*. Technical Report. EPFL.

[2] Laurent Bindschaedler, Ashvin Goel, and Willy Zwaenepoel. 2020. Hailstorm: Disaggregated Compute and Storage for Distributed LSM-based Databases. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM.

[3] Laurent Bindschaedler, Andreas Kipf, Tim Kraska, Ryan Marcus, and Umar Farooq Minhas. 2021. Towards a Benchmark for Learned Systems. In *37th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2021, Chania, Greece, April 19-22, 2021*. IEEE, 127–133. https://doi.org/10.1109/ICDEW53142.2021.00029

[4] Laurent Bindschaedler, Jasmina Malicevic, Baptiste Lepers, Ashvin Goel, and Willy Zwaenepoel. 2021. Tesseract: Distributed, General Graph Pattern Mining on Evolving Graphs. In *Proceedings of the 16th EuroSys Conference (EuroSys '21)*. ACM.

[5] Laurent Bindschaedler, Jasmina Malicevic, Nicolas Schiper, Ashvin Goel, and Willy Zwaenepoel. 2018. Rock You Like a Hurricane: Taming Skew in Large Scale Analytics. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 20.

[6] Jasmina Malicevic, Baptiste Lepers, and Willy Zwaenepoel. 2017. Everything you always wanted to know about multicore graph processing but were afraid to ask. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*. 631–643.

[7] Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel. 2015. Chaos: Scale-out Graph Processing from Secondary Storage. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 410–424.