Research Statement

Andrew Crotty

I am a systems researcher specializing in the areas of data management and data science. My research primarily explores the design and implementation of tools for large-scale data analytics. The techniques that I develop enable users like data scientists and domain experts to efficiently analyze massive amounts of data.

Past Work

Although my work has covered all layers of the data science stack, including the development of user interfaces for visual data exploration [11, 12, 13, 4, 20, 14] and techniques for natural language query translation [18, 19], I mainly focus on redesigning data analytics systems to best leverage the latest hardware. In the following, I highlight several examples of this theme from my past work in two key areas: (1) improving performance and (2) reducing resource consumption.

Improving Performance

Tupleware: Most data analytics frameworks (e.g., Hadoop, Spark) were designed for large deployments of commodity servers, where I/O is the main bottleneck and failures are common. We observed, however, that many users of these frameworks operated much smaller clusters with more reliable hardware. This mismatch inspired us to start the Tupleware project [8, 7, 6] to explore the design of an analytics framework for the "typical" user, which incorporated novel query introspection and code generation techniques to outperform existing systems by 100–1000×. Tupleware was one of the first systems to investigate just-in-time compilation of user-defined functions in distributed analytics systems; now, many popular systems have transitioned to a similar approach. For example, initial versions of Spark SQL [2] added support for compiling query expressions into Java bytecode, and Project Tungsten [1] extended this capability by generating data-centric Java bytecode for executing the entire query.

Swole: The low-level optimizations that we applied in Tupleware (e.g., inline expansion, pipelining, SIMD vectorization) can significantly improve the performance of compute-intensive workloads, but they have relatively little benefit for memory-bound workloads. For these cases, we developed Swole [9], a query execution strategy that optimizes for better data access patterns (i.e., the order of retrieving data from memory) rather than reducing the amount of work performed by the CPU. Unlike existing approaches that filter data as early as possible during processing, Swole counterintuitively defers filtering until after other operations, such that the better access patterns outweigh the overhead of wasted work. Swole outperformed state-of-the-art approaches by over 2.6× on a standard data analytics benchmark.

NAM-DB: For distributed systems, the conventional wisdom is that network I/O is the primary bottleneck and should be avoided as much as possible. Yet, with the increasing availability of high-bandwidth networking technologies like InfiniBand, this assumption no longer holds true: the network bandwidth of the most recent InfiniBand interconnects can match (or exceed) the local memory bandwidth of a server. We proposed a novel architecture for distributed analytics systems, called NAM-DB [3], that takes full advantage of InfiniBand's higher bandwidth and new features like remote direct memory access. In particular, NAM-DB introduced the concept of network-attached memory, allowing disaggregated computing resources to efficiently access pools of remote memory. Our work showed speedups of up to 2× and 5× for distributed joins and group-by aggregations, respectively.

Hist-Tree: Searching for values in a dataset is a fundamental problem in computer science, with a wide variety of algorithms (e.g., binary search) and data structures (e.g., B+trees) developed over the years. Recently, learned indexes [16] were proposed as an alternative that could offer lower lookup latency and smaller space consumption by adapting to inherent patterns in the data. My work [5] demonstrated that the main advantage of learned indexes comes instead from implicit assumptions (e.g., data sortedness) that are not fully leveraged by the comparison points. I created a new data structure called a histogram tree (Hist-Tree) that takes advantage of these same assumptions, allowing it to outperform both traditional and learned indexes. My preliminary results demonstrated that Hist-Trees could achieve up to a 1.8–2.7× reduction in lookup latency compared to state-of-the-art learned indexes while also exhibiting orders-of-magnitude improvements in index size and build time. Since the initial publication, I have improved Hist-Tree with several low-level optimizations that better target modern hardware features, including fast bitwise operations for lookups and adaptive node sizes to reduce space overhead. I am also collaborating with two major database system vendors, Snowflake and Vertica, to investigate Hist-Tree's real-world applicability. The students I am leading at Carnegie Mellon University are working with engineering teams from these companies to integrate the Hist-Tree into their platforms, which will allow us to understand the practical advantages over alternative data structures.

Reducing Resource Consumption

WimPi: The primary motivation behind the Tupleware project was that most users of data analytics frameworks tend to favor high-end hardware, with deployments typically ranging from a single machine to small clusters. These setups provide excellent performance, but they often ignore other important factors (e.g., purchase price, energy utilization, cooling costs). Therefore, during my postdoc, I started the WimPi [10] project to show that a cluster of inexpensive single-board computers (SBCs), like the Raspberry Pi, could achieve similar performance to high-end servers at a fraction of the cost and energy consumption. Although others had previously explored similar ideas for computationally simple and easily partitionable use cases (e.g., key-value stores), so-called "wimpy" nodes have been rejected as unsuitable for more complex data analytics workloads. WimPi takes advantage of recent hardware improvements driven by advances in mobile computing to show that SBCs can provide considerably more "bang for the buck" in this setting. Overall, the results demonstrate up to several orders of magnitude in cost reductions coupled with substantial energy savings compared to on-premises and cloud deployments, all without a significant increase (and sometimes even an improvement) in absolute runtimes.

DeepSqueeze: Datasets continue to grow in size and complexity, driven primarily by a "log everything" mentality under the assumption that the data will be useful for future analysis or compliance purposes. However, data storage and transmission costs have not kept pace, making effective compression algorithms more crucial than ever. Columnar encoding schemes (e.g., dictionary, run-length, delta) are highly effective for tabular data, but they do not consider potential relationships among columns (e.g., functional dependencies, correlations). Semantic compression techniques, on the other hand, leverage such relationships by storing only a subset of the columns necessary to infer the others (e.g., ZIP Code is sufficient to determine city/state), but existing approaches cannot effectively identify complex relationships across more than a few columns at a time. To overcome these limitations, we built DeepSqueeze [15], a new tool for the long-term archival of tabular data that uses autoencoders to map tuples to a lower-dimensional space, thereby capturing high-dimensional relationships that existing approaches cannot. We showed that DeepSqueeze achieves over a $4\times$ size reduction compared to state-of-the-art alternatives.

Future Directions

I plan to continue working on new techniques to capitalize on emerging trends in hardware and workloads. Since systems research is inherently an applied area of computer science, my approach involves building software prototypes to validate my ideas. In the following, I describe two research directions I am enthusiastic about pursuing as a new faculty member.

Systematizing Hardware Specialization

During my Ph.D. and postdoc, my work usually focused on a single hardware bottleneck (e.g., CPU, memory) or a single new technology (e.g., InfiniBand, SBCs), but I believe that the rapidly evolving hardware landscape will make this approach unsustainable going forward. The cloud further exacerbates this problem by making a wide range of hardware platforms readily available to anyone.

Consequently, I am excited about the idea of developing a more principled approach to system design based on a universal cost model, which will allow us to reason about—and perhaps automate—the specialization of data analytics systems for any hardware platform. This cost model would encompass physical resource requirements in terms of the (1) system, (2) workload, and (3) hardware. Providing details about any two of these dimensions would then allow the cost model to predict the third. For instance, one could specify the resource requirements for a given system and workload, and the cost model would return the optimal hardware configuration.

The first step in realizing this vision is to develop a new declarative language for specifying resource requirements. In the initial version, software engineers could explicitly annotate the resources necessary for the input, output, and intermediates of individual system components (e.g., query plan operators). Long-term, I believe that these resource requirements can be inferred through static code analysis (similar to my work on Tupleware) or even estimated using machine learning techniques in cases where resource consumption is not straightforward.

A project of this scope will create myriad opportunities to collaborate with computer scientists from areas outside of core systems research. Since hardware plays a central role, computer architecture researchers are natural collaborators. Researchers working in the areas of programming languages and compilers could also help with aspects of automated software analysis.

Monitoring & Observability Use Cases

Observability is gaining traction as a key capability for understanding the internal behavior of large-scale system deployments. Instrumenting these systems to report quantitative telemetry data called metrics enables engineers to monitor and maintain services that operate at an enormous scale, empowering them to respond to any issues that might arise. To be useful, metrics must be ingested, stored, and queryable in real time, but many existing solutions cannot keep up with the sheer volume of generated data.

Together with collaborators at Brown University, Intel Labs, and Slack, I am building a new storage engine called Mach [17] specifically to handle high-volume metrics data. Mach has a lean, loosely coordinated architecture and, similar to many popular libraries (e.g., Berkeley DB, LevelDB, RocksDB), provides a simple API to store and retrieve data. Our preliminary results show that Mach achieves up to $10\times$ higher write throughput and $3\times$ higher read throughput compared to several widely used alternatives.

We will make Mach freely available as open-source software in the near future, and our industry collaborators plan to begin evaluating the system in production. Yet, I believe that we have just begun to scratch the surface of this space, as extending Mach to support other critical aspects of observability workloads will present interesting research challenges. For example, integrating log data will require the investigation of fundamental design considerations (e.g., trading off compression speed with size, enabling search over compressed data), whereas supporting traces might necessitate the development of entirely new data models (e.g., graph-based approaches to facilitate root cause analysis). Again, such a major undertaking will naturally provide opportunities for broader collaborations, in particular for researchers working in the areas of distributed systems, networking, and data center applications.

[1] S. Agarwal, D. Liu, and R. Xin. Apache Spark as a Compiler: Joining a Billion Rows per Second on a Laptop. https://databricks.com/blog/2016/05/23/apache-spark-as-a-compiler-joining-a-billion-rows-per-second-on-a-laptop.html, 2016.

[2] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark SQL: Relational Data Processing in Spark. In *SIGMOD*, pages 1383–1394, 2015.

[3] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian. The End of Slow Networks: It's Time for a Redesign. *PVLDB*, 9(7):528–539, 2016.

[4] C. Binnig et al. Towards Interactive Data Exploration. In *BIRTE@VLDB*, pages 177–190, 2017.

[5] A. Crotty. Hist-Tree: Those Who Ignore It Are Doomed to Learn. In *CIDR*, 2021.

[6] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, C. Binnig, U. Çetintemel, and S. Zdonik. An Architecture for Compiling UDF-centric Workflows. *PVLDB*, 8(12):1466–1477, 2015.

[7] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, U. Çetintemel, and S. B. Zdonik. Tupleware: "Big" Data, Big Analytics, Small Clusters. In *CIDR*, 2015.

[8] A. Crotty, A. Galakatos, and T. Kraska. Tupleware: Distributed Machine Learning on Small Clusters. *IEEE Data Eng. Bull.*, 37(3):63–76, 2014.

[9] A. Crotty, A. Galakatos, and T. Kraska. Getting Swole: Generating Access-Aware Code with Predicate Pullups. In *ICDE*, pages 1273–1284, 2020.

[10] A. Crotty, A. Galakatos, C. Luckett, and U. Çetintemel. The Case for In-Memory OLAP on "Wimpy" Nodes. In *ICDE*, pages 732–743, 2021.

[11] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: Interactive Analytics through Pen and Touch. *PVLDB*, 8(12):2024–2027, 2015.

[12] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. The Case for Interactive Data Exploration Accelerators (IDEAs). In *HILDA@SIGMOD*, 2016.

[13] P. Eichmann, A. Crotty, A. Galakatos, and E. Zgraggen. Discrete Time Specifications In Temporal Queries. In *CHI Extended Abstracts*, pages 2536–2542, 2017.

[14] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting Reuse for Approximate Query Processing. *PVLDB*, 10(10):1142–1153, 2017.

[15] A. Ilkhechi, A. Crotty, A. Galakatos, Y. Mao, G. Fan, X. Shi, and U. Çetintemel. DeepSqueeze: Deep Semantic Compression for Tabular Data. *SIGMOD*, 2020.

[16] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The Case for Learned Index Structures. In *SIGMOD*, pages 489–504, 2018.

[17] F. Solleza, A. Crotty, S. Karumuri, N. Tatbul, and S. Zdonik. Mach: A Pluggable Metrics Storage Engine for the Age of Observability. *CIDR*, 2022.

[18] N. Weir, A. Crotty, A. Galakatos, A. Ilkhechi, S. Ramaswamy, R. Bhushan, U. Çetintemel, P. Utama, N. Geisler, B. Hattasch, S. Eger, and C. Binnig. DBPal: Weak Supervision for Learning a Natural Language Interface to Databases. In *CAST@VLDB*, 2019.

[19] N. Weir, P. Utama, A. Galakatos, A. Crotty, A. Ilkhechi, S. Ramaswamy, R. Bhushan, N. Geisler, B. Hattasch, S. Eger, U. Çetintemel, and C. Binnig. DBPal: A Fully Pluggable NL2SQL Training Pipeline. *SIGMOD*, 2020.

[20] E. Zgraggen, A. Galakatos, A. Crotty, J. Fekete, and T. Kraska. How Progressive Visualizations Affect Exploratory Analysis. *TVCG*, 23(8):1977–1987, 2017.