Research Statement

Alekh Jindal
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
alekh@csail.mit.edu

The last decade has seen a proliferation of software systems for storing and managing large collections of data. These different systems are targeted at different types of data, or different data access patterns; examples include: transactional applications, such as banking (OLTP), analytical and reporting applications, such as finding the most popular items that have sold in a retail store (OLAP), as well as systems specialized for streaming (real-time), social networks and other graphs, and data archives. This is a sharp change from just a few decades ago where all applications were supported by monolithic data management systems (e.g., Oracle, or DB2). This proliferation of systems has occurred because it delivers significantly better performance. However, end users now need to pick the *right* data management system for their query workloads. Furthermore, modern enterprises typically see a variety of query workloads. For instance, a banking enterprise uses a transactional system for customer banking transactions, an analytical system for business intelligence, a streaming system for stock trading, and an archival system to meet regulatory requirements for data retention. As a consequence, today's companies have to manage and integrate several types of data management systems, which is tedious, expensive, and counter-productive for their business. So rather than making the world of data management easier, the *zoo* of systems sometimes has the opposite effect: it makes the life of the end users harder and more costly.

Research Overview

My research focuses on building scalable data processing systems that ease the pain of this proliferation of systems for end users, i.e., they support several query workloads, they are easy to deploy and maintain, and they have comparable query performance as specialized systems. Along this direction, I have worked on two major themes throughout my Ph.D. at Saarland University and Postdoc at MIT.

First, I looked at integrating database concepts into large-scale data flow systems, which are highly scalable yet inefficient for several query workloads. Hadoop MapReduce, for instance, has became extremely popular due to its massive scalability and ease-of-use. However, many researchers and practitioners have observed a huge performance gap between Hadoop and well-configured parallel databases on structured data (i.e., involving relational-style operators and queries). To address this, I have shown how a variety of database techniques, including indexes and joins [2, 3], optimized data layouts [12], data preparation and design [10, 11], and data cleaning can be added to data flow systems like Hadoop.

Second, I have looked at efficiently supporting several different query workloads in traditional relational databases, which typically support only a single type of query workload. We observe that each specialized database product has a different data store, e.g., row store for transactional and column store for analytical database systems, indicating that different query workloads work well with different data layouts. Therefore, a key requirement for supporting several query workloads is to support several data layouts [1, 4, 13]. I have researched multi-layout database systems in several settings, including systems that combine OTLP and OLAP [6, 9, 15], systems that combine OLAP and graph analytics [7, 8, 14], and systems that perform both full-scans and ad-hoc range queries [17].

Database Techniques for Large-scale Data Management

As noted above, the first theme of my research is around adding database techniques to data flow systems like Hadoop MapReduce. I will now discuss a few of my research projects in this area.

Hadoop++/HAIL *(Indexes, Joins, Layouts)***.** Hadoop MapReduce allows non-expert users to run complex analytical tasks over very large data sets on very large clusters and clouds. Hadoop MapReduce, however, suffers from poor performance due to inefficient scan-oriented data processing using a hard-coded query processing pipeline. This is undesirable for several analytical tasks in which users want fast access to their data. Thus, the goal of Hadoop++ is to significantly improve the performance of Hadoop MapReduce by integrating database techniques such as data layouts, indexes, and join processing into Hadoop's storage and query processing engines. The challenge is to scale these techniques to very large data sets, preserve the fault-tolerance, and keep the MapReduce interface intact. In Hadoop++, we represent the Hadoop's query processing pipeline as a database-style physical query execution plan and introduce two additional data processing techniques to improve this plan: (i) indexed data access for quickly accessing the relevant portions of a dataset, and (ii) co-partitioned join processing

for efficiently combining two or more datasets. This is done by adding two new types of data blocks to Hadoop MapReduce: indexed data blocks to support index data access and co-partitioned data blocks to support co-partitioned joins. As a result, the performance of Hadoop++ even matches the performance of a well-configured parallel DBMS. Our experimental results show that Hadoop++ is 20x faster than Hadoop and has comparable or better performance than HadoopDB (now Hadapt).

The Hadoop Aggressive Indexing Library (HAIL), a follow-up of Hadoop++, takes indexing in Hadoop even further. HAIL utilizes idle CPU cycles in the Hadoop MapReduce pipeline to aggressively create as many indexes as possible, each for a different data block replica when uploading data. This has several consequences. First, several indexes are available at query time for incoming MapReduce jobs. As a result, there is a higher likelihood of being able to do an index scan, which is much faster than a full scan. Second, there is no preparation time to create the indexes, as in Hadoop++. Instead, indexes are available as soon as the data is uploaded. Finally, since HAIL integrates index creation tightly with the Hadoop upload pipeline, the overhead of creating the index is negligible in the overall data upload costs. As a result, indexes are created almost for free, in contrast to Hadoop++, which has very high index creation costs. HAIL runs up to 68x faster than Hadoop while having comparable or better (due to binary representation) data upload time.

Apart from indexes and joins, data layouts are crucial design decisions for good performance. Therefore, we looked at data layouts in the Hadoop storage engine, the Hadoop Distributed File System (HDFS). We found that the default row layout is not suitable for many cases. We proposed a new data layout, coined the Trojan Layout, that internally organizes data blocks into attribute groups to improve the data access time for a given workload. Our design exploit the way HDFS stores multiple replicas of each data block on different computing nodes. Trojan HDFS automatically creates a different Trojan Layout per replica to fit different parts of the query workload better. As a result of this heterogeneous replication, we are able to schedule incoming MapReduce jobs to data block replicas with the most suitable Trojan Layout. Still, the Trojan Layout fully preserves the fault-tolerance properties of MapReduce. Our results demonstrate that the Trojan Layout allows MapReduce jobs to read their input data up to 4.8x faster than the default row layout and up to 3.5x faster than the PAX layout.

Hadoop++ is one of the most cited papers of VLDB 2010 and we have an international patent pending on the core ideas of heterogeneous replication in HAIL.

Cartilage *(Data Preparation & Cleaning)***.** Data preparation, or database design in general, is a significant step before performing data analysis. Increasingly, this step is now also a precursor to several big data applications. Such data preparation could involve logical transformations, e.g., data cleaning, data integration, sampling, as well as physical data transformations, e.g., partitioning, indexing, compression, at different data granularities. Unfortunately, current large-scale data management platforms provide little or no support for such data preparation. The goal of Cartilage is to allow developers to have full control over data preparation in a distributed file system, such as HDFS. Cartilage provides a dataset abstraction between the logical user dataset and the actual physical HDFS files. We introduced *data plans*, analogous to query plans, to enable users to declaratively specify their transformations, from the logical dataset to the physical HDFS files. The Cartilage execution engine maintains the lineage of data transformation and developers can exploit this lineage to make fine-granular transformation decisions. Cartilage sits on top of HDFS and could be used with several data processing systems that have HDFS as the underlying storage. We used Cartilage to build a scalable data cleaning system for violation detection and repair. This data cleaning system compiles the data quality rules provided by the user into a series of transformations and runs them in a distributed fashion. As a result, this new data cleaning system outperforms baseline systems by up to more than two orders of magnitude over a variety of data cleaning tasks.

Towards One-size-fits-all Database Architectures

The second theme of my research is around making relational databases more adaptive to the workload they are running, allowing a single system to work better for multiple types of queries or data. My key projects in this direction are as follows.

The One-size-fits-all Vision. My research fits in a broader vision of a highly flexible data managing system, which automatically sets the initial configuration and adapts to changing workload later on, with improved performance, lowered cost and better maintainability. This vision for a new system, coined *OctopusDB*[1], initially stores a log or *journal* of data operations. Thereafter, depending on the workload, OctopusDB creates arbitrary physical representations, called *Storage Views*, of that journal. As a result of this flexible data storage layer, OctopusDB can mimic a variety of systems and efficiently support dynamic query workloads. We formulated the query optimization problem as storage view selection problem and we proposed a holistic storage view optimizer that uses a cost model for querying, updating, and transforming the storage views. Throughout my Ph.D. and Postdoc, I built several prototypes using different data management systems including MySQL, BerkeleyDB,

[1] An octopus may adapt to its surroundings using a camouflage unmatched by any other species on earth. It may change both the color and the texture of its skin. Additionally, some octopus species may even mimic movements and shape thus impersonating other species, e.g., http://marinebio.org/species.asp?id=260

PostgreSQL, IBM DB2, Hadoop, as well as stand alone Java and C prototypes, as steps towards this vision and to demonstrate the benefits of our approach. Below I describe three concrete follow-up projects and prototypes towards this vision.

1. OLTP and OLAP. In many situations, end users want to perform analytical queries (OLAP) in the same transactional processing (OLTP) system. Over the years, column stores have demonstrated significantly superior performance over row stores for analytical workloads. Database vendors, however, offer column and row stores as different database products. Therefore, we proposed Trojan Columns to inject column store (OLAP) functionality into a given row-oriented (OLTP) database system. The idea is to decouple logical row-oriented database table from its physical columnar representation. To implement this, Trojan Columns uses UDFs as a pluggable storage layer to write and read data. With Trojan Columns, the user does not need to change his schema and his queries remain almost unchanged. Trojan Columns work very well with non-nested and high selectivity queries, improving the performance of a closed source database system by up to 9x on unmodified TPC-H queries, and up to 13x on simplified TPC-H queries from other column stores.

Apart from row and column stores, several hybrid OLTP/OLAP workloads require hybrid data stores, i.e., vertical partitioning anywhere between row and column layouts. I looked at vertical partitioning in two ways. First, I proposed an adaptive vertical partitioning algorithm, which starts from a row (or a column store) and adaptively creates vertical partitioning based on the observed query workload. Second, I looked at how to pick a vertical partitioning algorithm. This is important because a number of vertical partitioning algorithms have been proposed over the last three decades. However, it is not clear how good different algorithms are, or even how to compare them. In my research, I isolated the core algorithms from the implementation specific settings and introduced a systematic way of comparing different vertical partitioning algorithms based on the following four questions: (i) *how fast*, (ii) *how good*, (iii) *how fragile*, and (iv) *where does it make sense*. My findings reveal that: (1) we can do four orders of magnitude less computation and still find the optimal layouts, (2) the benefits of vertical partitioning depend strongly on the database buffer size, and (3) vertical partitioning does not work well with highly fragmented data access patterns, e.g., in TPC-H-like benchmarks.

2. OLAP and Graph Analytics. End users increasingly want to perform the graph analytics directly with the relational engine, instead of moving data back and forth to a dedicated graph system. Furthermore, there is a need to combine relational analytics (OLAP) with graph analytics for end-to-end graph processing workflows. Thus, my goal in this project is to develop tools and techniques for efficient in-database graph analytics. Specifically, I looked at four things. First, I compared different systems for graph analysis and found that column stores, when properly tuned and when executing carefully written queries, can provide very good performance, competitive to dedicated graph stores. I further worked with our friends at HP Vertica to come up with several query optimizations techniques to tune the performance of graph queries on the column-oriented Vertica relational database. Second, I looked at the ease-of-use of doing graph analytics in relational databases. Given that it is awkward to express graph queries in SQL, we proposed two alternatives: (i) a *vertex-centric* query interface, which is a popular query language for graph analytics, and (ii) a new language called GRAPHiQL, which hits the middle ground between declarative- and procedural-style languages and allows developers to combine both the table and the graph operations. Third, we looked at providing better support for iterations. We developed a technique which loads the graph into main memory and in an optimized data structure. As a result, column stores can process iterative graph queries significantly faster than before and comparable to specialized main-memory graph systems. And fourth, we are working with Intel to look at the end-to-end graph processing workflows of their customers to determine the most suitable tools and techniques for them.

3. Scan and Ad-hoc Range Queries. Over the recent years, *Database Cracking* (also referred to as adaptive indexing) has emerged as a fascinating line of work in database research. The basic idea of database cracking is that apart from full scans, end users also have ad-hoc filter queries for which they want to have better performance without the upfront indexing effort. In my research, I have identified several limitations of existing cracking algorithms, proposed novel algorithms to improve the convergence and robustness of database cracking, introduced fingerprinting of different indexing methods, and offered promising directions for future research. We have open sourced our implementation and extensions.

OctopusDB won the *Best Outrageous Ideas and Vision* paper award at CIDR 2011. We also successfully filed a US Patent based on the major ideas in this project. In addition, my work helped receive a funding of 1.1 million Euros from German Ministry of Education and Science, which supported much of my PhD research. Furthermore, our work on database cracking won a *Best Paper Award* at VLDB 2014 and was invited to the *Best of VLDB 2014* Issue of VLDB Journal.

Future Research Agenda

In the future, I plan to continue applying the principles of one-size-fits-all in several newer directions, as well as branch out into a number of new research areas. I summarize a few of these ideas here.

Newer Directions for One-size-fits-all Approach: Given that the one-size-fits-all approach makes data management practical and easier for the end users, I see two new opportunities in this approach:

- *Data management over heterogenous hardware.* We are increasingly seeing different data management systems for different hardware, e.g., disk, SSD, memory, and GPUs. These systems are further accompanied with hardware-specific techniques, e.g., B^+-trees for disk, CSB^+-trees for cache conscious main-memory setting, Bw-tree for lock free processing on multi-cores, and FD-trees for SSD. As a result, end users are once again burdened with picking the right systems and coordinating between them, including gluing different systems together and moving data around, making their lives much harder. Therefore, we need to build systems which support multiple hardware configurations and allow users to do data management on their existing, often heterogeneous, hardware, without requiring them to manage several systems.

- *Data management with heterogenous cluster computing engines.* It is increasingly common to have multiple query processors within the same cluster computing framework. For example, the Hadoop ecosystem now encompasses query processors for MapReduce, graphs, streams, etc., each having a different programming model, and yet working on the same underlying HDFS storage backend. And many data processing tasks involve queries spanning multiple query processors. Therefore, instead of having the end users stitch these heterogenous query processors together, we need a unified approach to move data between different query processors as well as to transform the data for each query processor.

Apart from leveraging my earlier research experience, the above research theme also offers opportunities for collaborating with colleagues from other areas of computer science, e.g., compilers (to compile the query over heterogenous hardware), and programming languages (to synthesize the data for different computing engines).

Proactive Data Management Systems: Traditional data management systems are either *passive*, i.e., they do not anything beyond what is specifically requested by the user, or they are *reactive*, i.e., they do additional work only in response to observed user actions (i.e., the query workload). My vision is for *proactive data management*, wherein the system starts working in response to the data itself, as soon as it is generated, to derive value from it, both better and faster. This means that the system does not wait for the users to initiate data processing or to supply queries before the system begins optimizing its performance for the data loaded and the available hardware resources. This also means that data ingestion and query processing are no longer two distinct steps in the data processing pipeline. Rather, the proactive data management system harnesses the idle resources on modern hardware, including multi-cores, fast networks, large memory, and massive disk storage, to make data processing a continuous process, starting right from the data generation all the way through the querying and data extraction process, preempting bad performance and improving the user experience all along. I would like to explore this broad vision in my future research. In particular, I plan to explore the following three sub-areas in this direction:

- *Dropping the Assumption of a Query Workload.* Many systems have been proposed that try to generate an optimized physical database design from a user-supplied workload. Thus, preparing the data (database design) according to the query workload is critical. Unfortunately, the assumption that a workload is available may not hold in many ad-hoc and exploratory data analyses. Thus, we need a robust database design for ad-hoc query workloads such that all queries benefit immediately, with the design becoming progressively more performant for particular queries if the user repeatedly access data through similar queries [5]. Such a data-driven design philosophy reduces the barriers to effective data analytics for analysts. Analyst are free to try ad-hoc queries without getting intimated by terrible query performance to start their analysis.

- *Anticipating Query Workloads in Advance.* Several database applications have patterns of data access and update, e.g., iterative graph analysis, or repeated scans in a data visualization system. Such applications repeatedly incur the cost of doing similar effort, such as data movement, transformation, and checkpointing, thereby slowing them down considerably. We need to anticipate such query workloads in advance and eliminate many of the data processing overheads, just ahead in time. In our earlier work, we showed that we can predict the performance of a given workload [16], the question now is whether we can predict the workload as well.

- *Guiding Users to Discover Query Workloads.* The data deluge in today's world requires significant amount of effort from the users to process their datasets and still several pieces of data remain unprocessed. Therefore, we need data-driven techniques to suggest queries to users and guide them in their data exploration process. Examples include queries that could produce interesting results based on freshness, information content, or outliers. Or, queries that could be answered quickly, due to prior data processing, or with high quality, due to prior data integration and cleaning efforts. As a result, users are less likely to miss anything useful and reach from data to insights quickly.

The above vision for a proactive data management system explores newer ways of processing and making sense of data. This is challenging and exciting at the same time, and I am looking forward to realizing this vision.

References

[1] J. Dittrich and A. Jindal. Towards a One Size Fits All Database Architecture. *CIDR*, pages 195–198, 2011, *Best Outrageous Ideas and Vision Paper*.

[2] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). *PVLDB*, 3(1):518–529, 2010.

[3] J. Dittrich, J.-A. Quiane-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad. Only Aggressive Elephants are Fast Elephants. *PVLDB*, 5(9):1591–1602, 2012.

[4] A. Jindal. The Mimicking Octopus: Towards a one-size-fits-all Database Architecture. *VLDB PhD Workshop*, pages 78–83, 2010.

[5] A. Jindal. Robust Data Transformations . *CIDR Abstract*, 2015.

[6] A. Jindal and J. Dittrich. Relax and Let the Database do the Partitioning Online. *VLDB BIRTE*, pages 65–80, 2011.

[7] A. Jindal and S. Madden. GraphiQL: A Graph Intuitive Query Language for Relational Databases. *IEEE BigData*, pages 441–450, 2014.

[8] A. Jindal, S. Madden, M. Castellanos, and M. Hsu. Graph Analytics using the Vertica Relational Database. *Under Submission*, November, 2014.

[9] A. Jindal, E. Palatinus, V. Pavlov, and J. Dittrich. A Comparison of Knives for Bread Slicing. *PVLDB*, 6(6):361–372, 2013.

[10] A. Jindal, J. Quiane, and S. Madden. Cartilage: Adding Flexibility to the Hadoop Skeleton. *SIGMOD*, pages 1057–1060, 2013.

[11] A. Jindal, J. Quiané-Ruiz, and S. Madden. Fine-grained Data Preparation and Storage. *Under Submission*, December, 2014.

[12] A. Jindal, J.-A. Quiané-Ruiz, and J. Dittrich. Trojan Data Layouts: Right Shoes for a Running Elephant. *SOCC*, pages 21:1–21:14, 2011.

[13] A. Jindal, J.-A. Quiané-Ruiz, and J. Dittrich. WWHow! Freeing Data Storage from Cages. *CIDR*, 2013.

[14] A. Jindal, P. Rawlani, E. Wu, S. Madden, A. Deshpande, and M. Stonebraker. Vertexica: Your Relational Friend for Graph Analytics! *PVLDB*, 7(13):1669–1672, 2014.

[15] A. Jindal, F. M. Schuhknecht, J. Dittrich, K. Khachatryan, and A. Bunte. How Achaeans Would Construct Columns in Troy. *CIDR*, 2013.

[16] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and Resource Modeling in Highly-Concurrent OLTP Workloads. *SIGMOD*, pages 301–312, 2013.

[17] F. M. Schuhknecht, A. Jindal, and J. Dittrich. The Uncracked Pieces in Database Cracking. *PVLDB*, 7(2):97–108, 2014, *Best Paper* and invited to VLDB Journal's *"Best of VLDB 2014"* issue.