Research Statement

Barzan Mozafari

Data is growing at a pace faster than Moore's law and at the same time, the demand for more complex analytics is increasing. These trends significantly hinder the scalability and usability of existing data-intensive systems. My research addresses these challenges through the design of domain-specific query languages that are more expressive, the development of query processing algorithms that scale to massive volumes of data, and the integration of these ideas into real-world systems. I have pursued this agenda in a diverse array of settings, looking at relational versus semi-structured data models, analytical and transactional workloads, and on data ranging from weblogs to stock quotes to sensor readings.

During my PhD at UCLA, I built several systems for data stream mining and complex event processing, and as a postdoc at MIT, I developed performance models for multi-tenant database clouds, applied machine learning to crowdsourcing workflows, and worked on BlinkDB—a tool to support interactive queries on terabytes of data in MapReduce clusters.

Research Approach. I build upon advanced mathematical models to deliver practical database solutions to real-world problems by adapting concepts and tools from applied statistics, complexity theory, automata theory, and machine learning. In my approach, real-world applications, solid theoretical foundations, and building systems are equally important. Collaborating with domain experts helps me identify inter-disciplinary problems that have a direct impact on real applications. Applying theoretical techniques to system problems not only leads to breakthroughs, but also provides clearer understandings of the solution. Finally, I believe that it is crucial that software systems be evaluated through real implementations to capture the full complexity of the solution and pave the way for industrial adoption, which in my opinion is an important contribution of system-oriented research. My past research reflects this philosophy where I have applied theories of nested-words, subsampling, forecasting, queuing theory, and machine learning to solve real-world problems and build better systems. In summary, my research bridges theory and practice by combining a strong theoretical background with systems building skills.

PhD Research

To achieve scalability and reliably and to lift the data management burden from application developers, there is an increasing demand for integrating complex analytics into data management systems. This creates significant challenges for which I provided solutions in two key areas: complex event processing and data stream mining.

High-performance Complex Event Processing (CEP)

CEP is a broad term, referring to any application that involves searching for *complex patterns* among raw events to infer higher-level concepts. Examples include high-frequency trading (a certain correlation in stock prices that triggers a purchase), intrusion detection (a series of network activities that indicate an attack), click stream analysis (a sequence of clicks that trigger an ad), and electronic health systems (a combination of sensor readings raising an alert). CEP applications have created a fast-growing market which has led database vendors to add new constructs (called MATCH_RECOGNIZE) to the SQL language that allow for expressing sequential patterns among the rows in a table. Seeking richer abstractions for supporting CEP applications, I introduced constructs based on Kleene-closure expressions and showed that they are significantly more powerful than those proposed by database vendors (which are provably incapable of expressing many important CEP queries). Specifically, I designed the first two database query languages that used *nested word automata (NWA)* as their underlying computational model: K*SQL [P12, P4][1] with a relational interface, and XSeq [P15, P14] with an XML interface. NWAs[2] are recent advances in the field of automata theory that generalize the notion of regular languages to capture data that has both sequential and hierarchical structures. Examples of such data are XML, JSON files, RNA proteins, and traces of procedural programs. In a nutshell, NWAs strike a balance between expressiveness of pushdown automata and tractability of finite

[1] Papers are numbered based on my CV.

[2] `http://www.cis.upenn.edu/~alur/nw.html`

state automata. I adopted NWAs as the mathematical choice of abstraction in my CEP languages for several reasons. First, many important patterns can be easily expressed as regular expressions over semi-structured data, and hence, as NWAs (which can express all monadic second order formulas with a matching relation). For instance, the β-meander motif (depicted in Fig 1) which is a protein topology with two or more consecutive antiparallel β-strands linked together by hairpin loops, is essentially a regular expression of form $(\beta^+h)^+$. Such structural patterns determine the biological functions of the protein and hence, are highly important in genomics and molecular biology (where XML is a popular data exchange format). Additionally, since NWAs can be evaluated in a single pass over the input, my languages remained amenable to streaming implementation. I also exploited the closure properties of NWAs under boolean operations to develop a modular query processor, e.g. I only implemented each optimization technique to operate on the basic structure, which could then be applied to any boolean combination of that structure.

Figure 1: The β-meander motif.

Allowing users to write arbitrary regular expressions, however, posed significant algorithmic challenges, as the resulting automata often had a high degree of non-determinism, which could be computationally prohibitive. Thus, I developed several compile-time and run-time optimization algorithms to detect unpromising threads of the automata early on, and to use the previous failures to reduce the number of unnecessary comparisons. I generalized the Knuth-Morris-Pratt algorithm for searching strings to operate on nested words, and to also handle arbitrary regular expressions with arbitrary predicates. I formally studied the complexity and expressive power of my languages, and implemented them into full-fledged systems. In particular, K*SQL solves the long-standing problem of providing a unified query engine for both relational and hierarchical data. Similarly, despite 15 years of previous research where using tree automata for XML optimization was the status quo, XSeq translates XML queries into NWAs (which are then optimized using my algorithms) and outperformes the state-of-the-art XML engines by several orders of magnitude. XSeq received the SIGMOD's best paper award in 2012.

Data Stream Mining

I contributed to the design and development of Stream Mill Miner (SMM) [P13, P2, P1], which is a Data Stream Management System (DSMS) built to support complex analytics for on-line data mining. In particular, I focused on *load shedding*, which is a key component of stream processors: if data's arrival rate exceeds the system's capacity, queues build up and the latency increases without bound. In these situations, the system has to shed some of its load at the cost of accuracy. My work was the first to provide a general model for load-shedding, by formalizing it as an optimization problem and introducing the concept of adaptive and mining-aware load shedding [P11]. I proved that the resulting sampling strategy yields the best accuracy for a large class of queries (including conventional aggregates and most mining queries). Recently, we have generalized our algorithm to trace the propagation of uncertainty throughout almost arbitrary query networks [P5] (as long as the operators are mean-like and asymptotically normal statistics). Interestingly, our uncertainty propagation framework has found important applications beyond load shedding. In particular, Veolia (the largest environmental solutions company in the world) has provided me with a research grant to explore applications of our framework in integrating the impact of natural uncertainties into their complex system of designing sustainable environmental solutions (Veolia uses rainfall statistics to design viable water canals and safety pipes and hence, they need to integrate the overall impact of their data uncertainties into their risk analysis tools).

I also worked on the problem of mining frequent patterns from data streams. An example of a *frequent pattern*, say in online advertisement, can be that "90% of the users who clicked on link1 and link2, also clicked on link3 in the same session". Despite numerous optimizations proposed over the past decade, mining frequent patterns is still too expensive to be performed in real-time. I made the following observation: we can mine the data only once (say, at the beginning) and thereafter only keep "monitoring" the mined patterns to detect any changes in the patterns. Then, the challenge was finding a way of monitoring thousands of patterns that would be significantly faster than mining those patterns from scratch. Thus, I introduced the concept of *conditional counting*: given two collections of sets, say A and B, and a minimum frequency f, for each set S$\in$A, return the number of sets in collection B that contain S, say $c = |\{T \in B | S \subseteq T\}|$, only if $c \geq f$. In the special case of $f = 0$, conditional counting reduces to a simple containment counting. My mining algorithm was 2–3 orders of magnitude faster than the best algorithm at the time. In addition, the algorithm that I designed for efficient conditional counting using prefix trees, remains as the state-of-the-art for containment counting to this day (an

order of magnitude faster than hash-trees).

I worked on a number of other projects during my PhD. For instance, I designed algorithms for publishing privacy-preserving views of a database that enable Naïve Bayesian Classification with zero accuracy loss [P10]. In a different project [P8], I studied how users create and maintain data in the social web, where I applied extensive time-series analysis on historical logs of Wikipedia.

Postdoctoral Research

While my PhD focused on richer query languages and more efficient algorithms for CEP and data stream mining, my postdoc focused on improving the performance and robustness of new generations of data-intensive systems: database-as-a-service clouds, MapReduce-based data warehouses, and crowdsourced databases.

DBSeer: Workload Management for Multi-tenant Database Clouds

Operating a database management system (DBMS) that hosts many tenants, particularly in a database-as-a-service cloud, is a challenging task. Many transactions that run fine in isolation may run very slowly when run together, due to their complex interactions. Applications may generate unpredictable load that puts strain on different resources (RAM, disk, or CPU) at different times. Unfortunately, trial-and-error is the current solution for many operational needs of a database such as provisioning and what-if scenarios, e.g. *How much disk-bandwidth is needed to reach a target latency?* or *What happens if tenants A and B are co-placed on the same DBMS node?* Needless to say, trial-and-error is human-intensive and impractical for operating a large database cloud. To address these challenges, I have developed a statistical approach to *performance modeling and prediction* in multi-tenant databases, implemented as a tool called *DBSeer*. DBSeer collects a small set of low-overhead statistics from the DBMS and the operating system, measured during normal system operation. DBSeer matches the natural fluctuations of user requests and the resulting fluctuations in the collected statistics in order to infer the parameters of a set of models for the given DBMS and the workload(s) running on it. These models enable us to *attribute* system load, on a per-resource basis to different queries, tenants, or applications. DBSeer can accurately predict different performance metrics such as the maximum throughput and disk I/O with 5–20% error [P16, P18]. DBSeer avoids over-provisioning of resources by up to two orders of magnitude compared to manual tuning techniques and even classical regression techniques.

BlinkDB: Bounded Error and Bounded Response Time on Unbounded Data

It is becoming common for companies to own terabytes to petabytes of data, which are analyzed either in enterprise data warehouses or in MapReduce-style frameworks (e.g., HIVE). Unfortunately, these systems are not suitable for interactive or exploratory tasks. For instance, computing a simple aggregate over 10 terabytes of data stored on 100 machines can take $\sim$30 minutes on HIVE/Hadoop, which is unacceptable for rapid problem diagnosis or interactive data exploration. To enable sub-second query processing on such data, in collaboration with UC Berkeley, we have built a massively parallel, approximate query engine, called *BlinkDB* [P17, P6]. The observation is that one can often make reasonable decisions in the absence of perfect answers. Thus, BlinkDB users specify their SQL-like queries (which may contain complex machine learning algorithms encoded as user-defined functions) along with a maximum latency or error tolerance, and receive approximate answers accompanied by statistical error guarantees. To achieve sub-second latency over multiple terabytes of data, BlinkDB runs the query on pre-built samples of the data, choosing the best such sample at query runtime.

My main contribution in BlinkDB is an optimization module that decides on the best set of samples to build given a storage or time budget. This component is critical because achieving high accuracy and low latency is contingent upon having built the right set of samples in advance. In particular, simply storing uniform random samples leads to poor performance for certain classes of queries. For example, imagine a query comparing sales in NYC to those in Cambridge, MA. A 0.01% sample will yield thousands of records about NYC, but only a few about Cambridge, and thus it will not be possible to provide non-trivial error bounds about Cambridge. However, a *stratified* sample that has kept all the unique values in the columns that are queried by the user can be significantly more effective. The challenge that I had to overcome was that the queries are not known *a priori*, and building stratified samples for all subsets of hundreds of columns is infeasible. I solved this problem by formulating it as an integer linear program that combines three key factors: (i) the distribution of different columns, (ii) the storage cost of different stratified samples, and (iii) the frequency of column-sets in the past queries without losing the ability to answer new ad-hoc queries. We have shown that, by accepting 1% error,

BlinkDB can execute a wide range of queries by 2 orders of magnitude faster than current MapReduce stacks such as HIVE/Hadoop or SPARK. This implies that exploratory applications can now run 100× more queries with the same resources. For these reasons, BlinkDB's recent open-source release[3] has received much industrial attention, e.g. Facebook is scheduled to deploy BlinkDB on one of their 80-node clusters in February 2013.

Integrating machine learning into crowdsourcing workflows

Crowdsourcing is a powerful means of performing tasks that are difficult for computers, e.g. audio transcription or image annotation. Unfortunately, asking humans to label massive datasets (e.g. millions of daily tweets or images) can be very expensive and slow. I recently developed an active learning algorithm (based on non-parametric bootstrap) that reduces the number of questions posed to the crowd by 1–2 orders of magnitude compared to the baseline, and by 2–8× compared to previous active learning schemes (see [U2] for details).

Future Research

The projects that I started as a postdoc offer great potential for future research opportunities, which I would like to continue exploring in the next 3–4 years as I move to a new university.

Bringing Elasticity to Database Clouds

I see tremendous opportunities in building several important systems around DBSeer as an underlying tool. First, a run-time proxy for enforcing *soft* performance isolation can be built, where tenants of a database service will be allowed to exceed their set quota of resources, but will be elastically pushed back as soon as other tenants demand their share of resources. This multiplexer will have to invoke DBSeer to predict the impact of applying different admission policies on different tenants and use such information to make optimal workload management decisions at run-time (say to minimize the violations of service-level-agreements). This will significantly advance current workload managers, which are either based on (wasteful) *hard* isolation of resources (via virtualization and no multiplexing) or on tedious manual configuration of admission policies by an experienced admin. Similarly, many other research directions can be taken in using and extending DBSeer to support:

- *Diagnosis / Performance Inspection*: which transactions are causing the spike in latency?
- *Capacity planning*: How much disk-bandwidth, memory or CPU is needed to deliver a target throughput?
- *Automatic Turning*: Given a set of database configurations, which one leads to the best performance?
- *Billing*: What is each workload's contribution to the overall resource consumption of a database?

Each of these represents an important and promising direction of research that can easily become the topic of a PhD thesis. In fact, several major companies have already expressed interest in adapting and using DBSeer to improve their products, including Teradata, EMC, and Facebook's mysql team.

Database Design through Robust Optimization

A common problem with many database systems is that they rely on past workloads to choose an optimal physical design (e.g. a set of indices, materialized views or partitions), without any notion of how this optimality degrades as future workloads deviate from the past. As a result, performance often falls off a cliff even with minor changes of the workload. I plan on applying a combination of robust optimization (from operational research) and probabilistic forecasting to produce designs that are more robust. My high-level idea is to invoke existing designers many times, each time on a *carefully perturbed* version of the past workload, and rule out brittle parts of the design. The first challenge is how to perturb the workload such that certain statistical properties of the distribution are preserved. The next challenge is how to combine multiple designs and synthesize a more robust one that degrades gracefully when the workload undergoes a change. Companies such as Vertica have already committed their support toward an initial prototype that I have started implementing [U1]. However, tailoring this broad approach to different types of physical designs in a database (indexing, view materialization, data partitioning and placement, etc.) embodies a rich set of exciting research problems. Finally, the general idea of robust optimization is an attractive direction that can be pursued beyond databases. At a high level, almost any system that interacts with a dynamic and unpredictable environment (e.g., sensor networks, mobile devices) may benefit from making decisions that are more robust. Therefore, I intend to collaborate with colleagues in other disciplines to explore similar opportunities in different contexts.

[3] `http://blinkdb.org`