Reminiscences on Influential Papers

Kenneth A. Ross, editor

I'm very happy to be able to present this issue's seven reminiscences. They present compelling cases to find and read the cited papers, as well as being fun to read themselves.

I continue to invite unsolicited contributions to this column. (I haven't received any so far, but the previous issue has only been out a month or so at the time of writing.)
See `http://www.acm.org/sigmod/record/author.html` for submission guidelines.

Christos Faloutsos, Carnegie Mellon University, `christos@cs.cmu.edu`.

[Manfred Schroeder. Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise. W.H. Freeman and Company, 1991.]

"What was the single most influential work for your research?" There is a handful of truly influential papers: the R-tree, RAID, the Association Rules, each started a revolution. However, I find myself citing repeatedly this masterpiece book. Beyond George Kingsley Zipf and his famous 'law', and beyond the milestone book by Mandelbrot on fractals, Schroeder's book explains how self-similarity and power laws appear in countless phenomena, it shows how to measure the fractal dimensions, and, it gives a gentle introduction to the necessary mathematical tools for their analysis. For example, our '80-20 law' (80% of the queries go to 20% of the records) can be treated mathematically, under the name of 'multifractals', with surprising connections to Shanon's entropy and thermodynamics!

Zipf distributions, self-similarities and related power laws appear in amazingly diverse settings. They often make headlines, because they contradict head-on the traditional uniformity, independence, Poisson, and Gaussian assumptions. For example: (a) LAN traffic was found to be fractal/self-similar [Leland et al, SIGCOMM93 - best paper award] — the paper brought a revolution to queueing theory under the name of 'heavy tailed distributions'; (b) The internet follows power-laws [see the work by the CLEVER project, VLDB 99], [Barabasi et al, Nature 1999]; (c) Hits on web sites also follow a Zipf-like distribution [Bernardo Huberman et al, Science 98].

Power laws and fractals have already had impact in any sub-area of databases that deals with distributions, such as: (1) Query optimization, where we try to approximate the distribution of records in address space; ditto for benchmarks [Gray et al, SIGMOD94]; (2) Data mining, where we try to find patterns - sales patterns, customer arrival times etc. follow Zipf or 80-20 laws. (3) Text databases, where Zipf first showed his law; the sizes of postings list in an inverted index are heavily skewed, leading to clever compression and optimization methods (eg., [Tomasic et al, SIGMOD94]); (4) Web databases: not only the text parts, but also their links follow power laws, as mentioned; (5) Geographic databases, with fractal coastlines and border-lines, and skewed distributions of island sizes; (6) Spatial databases, where fractal dimensions replace the uniformity and independence assumptions (eg., [Papadopoulos et al, ICDT97]); (7) Performance analysis of disk/query traffic, where the traffic is typically bursty. Schroeder's book provides a fascinating and accessible introduction to the necessary tools and concepts.

Alon Levy, University of Washington, alon@cs.washington.edu

[Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. Proceedings of STOC, 1977.]

Chandra and Merlin's work greatly influenced my thinking and research direction long before I ever actually read the paper. They were the first to consider the problem of query equivalence for conjunctive queries (i.e., determining whether two queries return the same set of answers for any state of the database). They aimed to contrast optimizations that considered the global structure of the query with local optimization such as join ordering and access path selection. Their study of query containment and equivalence problems for different classes of queries generated a long and fruitful line of research. Until a few years ago query containment had been mostly a topic of conversation that enhanced bonding among fellow PODS aficionados. More recently, insights and results from query containment have helped solve many practical problems (e.g., semantic data caching, maintenance of physical data independence, integrity constraint violation, data integration and answering queries using views). As more and more applications require that we *reason* about data and views, query containment and equivalence will play a even more significant role.

Another reason this paper and line of work appealed to me is that they answered questions I felt should have been available in the Artificial Intelligence literature. Query containment algorithms are essentially inference algorithms over finite models for specific forms of logical sentences. As such, they are key to the field of Knowledge Representation, much of which strives to represent the world in some form of logic. However, with the exception of work on Description Logics, AI researchers have overlooked the opportunity to discover these classes of sentences for which sound and complete reasoning is possible. Hence, I believe that query containment algorithms will also play a greater role in AI applications, such as knowledge-base inference, planning and knowledge-base verification.

Pat O'Neil, UMass/Boston, poneil@cs.umb.edu

[C. Mohan. Concurrency Control and Recovery Methods for B+-Tree Indexes: ARIES/KVL and ARIES/IM. Performance of Concurrency Control Mechanisms in Centralized Database Systems, Prentice-Hall 1996, ISBN 0-13-065442-6]

For a number of years I was uncomfortable about my understanding of how locking is used by vendors to prevent transactional phantoms, especially since I knew that the "Predicate Locking" approach mentioned in most texts had been dropped by System R many years ago (see: Astrahan et al., TODS 1(2), 1976). When the KVL and IM locking approaches used at IBM were first published by Mohan in VLDB-90 and SIGMOD-92, I was anxious to learn them thorougly, but I found it difficult to fully grasp the concepts in a quick reading. It wasn't until a few years ago that I began covering the combined KVL and IM paper above in detail, and presenting it to my database internals class. I now believe that the ideas underlying these locking protocols are probably the most subtle in the database field. Since they are not easily grasped, and since all the researchers I know are extremely busy, I think they have received less attention than they deserve.

I have heard practitioners complain jokingly that Mohan's papers seem designed to provide the detail necessary for experienced programmers to perform immediate implementation. There is a certain amount of truth to this, and I for one find it a wonderful thing. I think an excellent database internals text could be written by simply expanding on the ideas in this paper (latches, locking by hashing, lock durations, logging, B-tree concurrency, etc.). The text would be particularly valuable in an academic setting because the techniques covered, some of which were a revelation to me, are ones that are ACTUALLY USED by IBM database programmers. I cannot help but think that many university researchers (both faculty and students) could overcome perceived isolation from industrial realities by studying this seminal work. Even practicing database system programmers, inside and outside IBM, who have not already spent time on this paper, would be well advised to expand their horizons by reading it carefully!

Eric Simon, INRIA, France, `eric.simon@inria.fr`.

[Robert Paige and Shaye Koenig. Finite differencing of computable expressions. ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, pages 402-454.]

This paper introduces a general method to optimize repetitive calculations of an expression of the form $E = f(x_1, ..., x_n)$ within a program. The method first analyzes the structure of the program and isolates program regions in which expression E can be transformed without affecting the semantics of the program. The transformation of E consists of: initializing a variable, say *curr_E*, that contains the first computed value of E, adding instructions to maintain the value of *curr_E* each time a parameter x_i is assigned a change dx_i, and replacing all further occurences of E by *curr_E*. The method uses a library of differentiation rules that describe how to incrementally maintain a primitive function f when one of its operand changes. Primitive functions include arithmetic operations and operations on sets. Addititional composition rules enable to combine the use of differentiation rules to differentiate a compound expression f. However, the most salient feature of the method is to formalize heuristic rules to decide whether the transformation of an expression E in a program region is *profitable* or not. For instance, one of these rules called *continuity* stipulates that a function f should be "easily" computable from its previous value and a "small" change in one of its operand. Then, the authors provide many syntactic rules to characterize continuous functions.

It took me some time to read and understand this article many years ago because the context in which the problem was set and the language used to describe the solution were quite unusual for a person with a database background. I suspect this is why this article is still unknown to many database researchers. However, this article had a strong impact on my research. With Francoise Fabret, we applied the finite differencing method to the optimization of production rules in databases. At that time, most efforts focused on database implementations of RETE and TREAT networks, and criteria to decide which of the two should be used to process a given set of production rules. However, it was not recognized that a central issue was to decide which intermediate calculations were worth to cache and incrementally maintain. The finite differencing method provided the appropriate framework to study this issue. Later on, with Francois Llirbat, we refined the method and applied it to the optimization of database triggers. Bob Paige and his colleagues continued to improve their method over the years and applied it to a variety of problems (including database integrity control). Even if the principles of finite differencing seem now natural to us in light of the many recent works on the choice of views that are worth materializing to optimize the repetitive execution of a query, I think that the pioneering effort of Bob Paige deserves special recognition.

Divesh Srivastava, AT&T Labs-Research, divesh@research.att.com

[Catriel Beeri, Raghu Ramakrishnan. On the Power of Magic. *Journal of Logic Programming* 10: 255-299 (1991). PODS 1987: 269-283.]

This is the paper that made me appreciate the beauty and the versatility of the Magic Sets approach to query optimization. It is a paper that I have read many times–the first time for a graduate course in databases, the second time because I was Raghu's student, but subsequently of my own free will–and I have learned something new each time. The paper formally studied sideways information passing strategies (SIPS), and established the fundamental nature of SIPS to a variety of query evaluation methods in the literature. I have used the key ideas of Magic Sets many times over the years, as have many other researchers, and I will always remember this paper as one that has had a significant impact on my own research.

Victor Vianu, University of California, San Diego, vianu@cs.ucsd.edu

[Ashok K. Chandra, David Harel. Computable Queries for Relational Data Bases. *JCSS* 21(2): 156-178 (1980).]

I have a vivid memory of meeting Ashok Chandra about fifteen years ago, while we were both visiting Serge Abiteboul at INRIA. After the ritual coffee, Serge, Ashok and I took a walk amid the barracks that had once served as NATO headquarters. Ashok, who was a kind of guru to us young ones, challenged us with a question that goes to the existential core of our field: what makes databases different? Does our area have a legitimate, well-motivated *raison d'être*, or does it amount to no more than an ad-hoc collection of recipes from programming languages, data structures, and algorithms? It is a question that he had tackled together with David Harel in this elegant paper, which in some sense established the theory of query languages as a field of research. Its definition of database query, using the notion of genericity, provided the foundation for much of the work that followed. Personally, I was profoundly influenced by this paper and much of my subsequent research with Serge on query languages was, at the bottom of it, an attempt to answer Ashok's question.

Gerhard Weikum, University of the Saarland, Germany, weikum@cs.uni-sb.de

[George Copeland, William Alexander, Ellen Boughter, and Tom Keller. Data Placement in Bubba. *SIGMOD* 1988, Chicago, Illinois, ACM, pp. 99-108, June, 1988]

This paper, which came out of the Bubba project at MCC, was the first to address the physical database design problem for parallel database servers, with particular focus on the partitioning and allocation of (relational) data across multiple disks or processing nodes. These issues are key to good performance tuning. To this end, the paper introduced the fundamental notion of data heat as a measure for the disk access load attributed to a data unit or collection of units, and the notion of temperature to normalize heat by the consumed space. Based on these metrics, the paper developed an elegant framework and heuristic algorithms for choosing which data should be placed on which disk so as to balance the disk load, and which data should be cached in memory so as to minimize the overall disk load.

I had the great opportunity of spending a postdoc year in the Bubba group at MCC where I could learn about this subject directly from the paper's authors. Later, their work was my main inspiration when I started working on dynamic data placement and migration in the early nineties. In this research of mine the notions of heat and temperature proved to be extremely useful for reasoning about load distribution and for developing algorithms that continuously adjust the allocation of data based on online statistics about access patterns, for example, to "cool down" hot disks. I have also seen fairly recent papers on the caching of query results in data warehouses to benefit greatly from the Bubba tuning framework. The paper by Copeland et al. is a true landmark paper, especially when you consider that this work was done before the industrial advent of parallel database systems. The problem of automating the physical database design for a cluster-based parallel data server, in the spirit of a zero-admin, self-tuning solution, has still not been solved in a truly comprehensive, industrial-strength manner, but this seminal paper is an excellent starting point and absolutely mandatory reading for everybody working on this highly relevant problem.
