From Auto-tuning One Size Fits All to Self-designed and Learned Data-intensive Systems

Stratos Idreos
Harvard University

Tim Kraska
MIT

ABSTRACT

We survey new opportunities to design data systems, data structures and algorithms that can adapt to both data and query workloads. Data keeps growing, hardware keeps changing and new applications appear ever more frequently. One size does not fit all, but data-intensive applications would like to balance and control memory requirements, read costs, write costs, as well as monetary costs on the cloud. This calls for tailored data systems, storage, and computation solutions that match the exact requirements of the scenario at hand. Such systems should be "synthesized" quickly and nearly automatically, removing the human system designers and administrators from the loop as much as possible to keep up with the quick evolution of applications and workloads. In addition, such systems should "learn" from both past and current system performance and workload patterns to keep adapting their design.

We survey new trends in 1) self-designed, and 2) learned data systems and how these technologies can apply to relational, NoSQL, and big data systems as well as to broad data science applications. We focus on both recent research advances and practical applications of this technology, as well as numerous open research opportunities that come from their fusion. We specifically highlight recent work on data structures, algorithms, and query optimization, and how machine learning inspired designs as well as a detailed mapping of the possible design space of solutions can drive innovation to create tailored systems. We also position and connect with past seminal system designs and research in auto-tuning, modular/extensible, and adaptive data systems to highlight the new challenges as well as the opportunities to combine past and new technologies.

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3314034

ACM Reference Format:
Stratos Idreos and Tim Kraska. 2019. From Auto-tuning One Size Fits All to Self-designed and Learned Data-intensive Systems. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299869.3314034

1 PART 1: NEW REQUIREMENTS FOR DATA INTENSIVE SYSTEMS

Data systems have always come with large sets of knobs. Such knobs allow a system to assume multiple forms with different performance properties and can be set appropriately depending on the workload, and hardware. Typically, a DBA or an auto-tuning tool decide the right setup assuming some knowledge of the workload. Overall, this paradigm has worked great for several decades but it increasingly becomes problematic for many cases of modern applications.

Rapidly Changing System Requirements. Today more than ever, we want to build, or change and adapt a data system quickly such that we can keep up with the needs of ever changing applications and hardware. New applications or new features in existing applications, with new workload patterns, appear frequently. A single system is not capable of efficiently supporting diverse workloads. This is a problem for several increasingly pressing reasons. First, new applications appear many of which introduce new workload patterns that were not typical before. Second, existing applications keep redefining their services and features which affects their workload patterns directly and in many cases renders the existing underlying storage decisions sub-optimal or even bad. Third, hardware keeps changing which affects the CPU/bandwidth/latency balance; maximum performance requires low-level storage design changes. These problems boil down to the one size does not fit all problem which holds for overall system design [59] and for the storage layer [8]. Especially, in today's cloud-based world even slightly sub-optimal designs by 1% translate to a massive loss in energy utilization and thus costs [41].

Knobs are not Enough. While the knobs exposed by modern systems do allow them to change their behavior, the problem is that the range of behaviors they can assume is still largely limited by the original design decisions made by the engineers. The range of behaviors is limited by what the original designers could foresee and efficiently engineer.

For example, a row-store system with a knob that allow it to fine tune its performance such as tuning the buffer pool size or choosing the right set of indexes will not be able to ever match the behavior of a column-store in analytics or the behavior of a key-value store in fast data ingestion. This is because, at its core it will still be defined by the good and the bad properties of its fundamental design choices. In this way, even though we have seen a plethora of innovative solutions in automatically tuning knobs of systems with works in the area of offline and online tuning as well as solutions that allow to adaptively and smoothly make some of these decision with works in the area of adaptive indexing, still those solutions are trapped within an overall system architecture that is defined strictly upfront and it is not easy to change or adapt; e.g., a column-store with adaptive indexing can build indexes automatically but is still defined by the main performance characteristics of a column-store architecture and it will not be a great solution for workloads that do not fit this overall design.

Self-designed and Learned Systems. The grant research challenge is whether we can easily or even automatically design systems given a problem. This would allow us to quickly spin off systems for new applications, features and hardware. And it can lead to systems that are able to drastically change their behavior as workload and hardware evolve. While this is clearly a very ambitious goal there are many intermediate steps in terms of removing the dependency on human-based decisions that critically restrict the sets of behaviors a system can assume. In particular, in this tutorial we survey recent innovations on:

(1) self-designed systems [26, 31] which know the possible design choices and their combinations for critical system design components such as data storage, and can choose the most appropriate design among drastically different choices
(2) learned systems [42, 43] which replace critical system components, restricted by human-made decisions, with models that capture numerous behaviors, making it easier to assume different performance properties.

Collectively these directions open the door for systems that can manipulate their storage and other critical components such as the choice of query processing algorithms, optimization strategies, memory requirements, in drastically more ways than past systems.

Tutorial Structure. The tutorial consists of three parts. The first part goes over the overall problem setting as described in this section. The second part discusses major research and industry trends from the past several decades that enabled systems to assume different forms or be easily extensible. In particular, we cover 1) auto-tuning (offline, and online) which allows systems to choose the right set of indexes given a workload, 2) adaptive indexing which allows systems to incrementally adapt and build their indexes as the workload evolves, 3) generalized indexing which allows engineers to easily support new data types in an existing system, as well as 4) modular systems which allow for easily swapping in and out major system components that add functionality. In the third part, we survey self-designed and learned-systems. We explain 1) the new opportunities they bring compared to past solutions, 2) how they can be applied to solve practical problems across many classes of data-intensive applications, and 3) the new research opportunities that arise from their fusion with each other and with past work.

2 PART 2: ONE SIZE FITS ALL SYSTEMS

Offline Indexing. Offline indexing [13, 32] is the earliest approach on self-tuning database systems. Nowadays, all major database products offer auto-tuning tools to automate the database physical design. Auto-tuning tools mainly rely on "what-if analysis" and close interaction with the optimizer to decide which indices are potentially more useful for a given workload. Offline indexing requires heavy involvement of a database administrator (DBA). Specifically, a DBA invokes the tool and provides its input, i.e., a representative workload. The tool analyzes the given workload and recommends an appropriate physical design.

Online Indexing. With online indexing the system continuously monitors the workload and the physical design is periodically reevaluated. System COLT [47] was one of the first online indexing approaches. COLT [55] continuously monitors the workload and periodically in specific epochs, i.e., every N queries, it reconsiders the physical design. The recommended physical design might demand creation of new indices or dropping of old ones. COLT requires many calls to the optimizer to obtain cost estimations. A "lighter" approach, i.e., requiring less calls to the optimizer, was proposed later [11]. Soft indices [47] extended the previous online approaches by building full indices on-the-fly concurrently with queries on the same data, sharing the scan operator. The main limitation of online indexing is that reorganization of the physical design can be a costly action that a) requires a significant amount of time to complete and b) requires a lot of resources. This means that online indexing is appropriate mainly for moderately dynamic workloads where the query patterns do not change very frequently. Otherwise, it may be that by the time we finish adapting the physical design, the workload has changed again, leading to a suboptimal performance.

Adaptive Indexing and Layouts. Adaptive indexing [27] is a lightweight approach in self-tuning databases. Adaptive indexing addresses the limitations of offline and online indexing for dynamic workloads; it reacts to workload changes by building or refining indices partially and incrementally as part of query processing. That is, no DBA or offline processing is needed. By reacting to every single query with lightweight actions, adaptive indexing manages to instantly adapt to a changing work load. As more queries arrive, the more the indices are refined and the more performance improves. Recently this area has received considerable attention with numerous works that study adaptivity with regards to base storage in relational systems, NoSQL systems, updates, concurrency, and time-series data management [2, 3, 7, 15, 18, 22, 23, 27, 28, 33, 46, 53, 54, 56, 57, 60]. Typically, in these lines of work the layout adapts to incoming requests. Similarly works on tuning via experiments [9], learning [4], and tuning via machine learning [1, 24] can adapt parts of a design using feedback from tests.

While auto-tuning and adaptivity provided the ability to achieve many different performance properties for the same system, they do not fundamentally change the properties of a system since it still moves within a narrow design space defined strictly by its original design.

Modular/Extensible Systems and System Synthesizers. Modular systems [52] is an idea that has been explored in many areas of computer science: in database systems in particular the concept has been studied for easily adding data types [20, 21, 50, 51, 58] with minimal implementation effort, or plug and play features and whole system components with clean interfaces [10, 12, 14, 36, 44, 45]. Modularity is a very promising direction for systems building and data-intensive systems in particular (because there is no single perfect storage design). However, in practice we have only seen so far systems with "large" components which do not really allow a system to drastically change its behavior but rather to support or not specific features.

Generalized Tree Indexes. A great example of extensibility is the work on Generalized Search Tree indexes (GiST) [5, 6, 25, 37–40]. GiST aims to make it easy to extend data structures used for indexing and tailor them to specific problems and data with minimal effort. It is a template, an abstract index definition that allows designers and developers to implement a large class of indexes. The original proposal focused on record retrieval only but later work added support for concurrency [38], a more general API [5], improved performance [37], selectivity estimation on generated indexes [6] and even visual tools that help with debugging [39, 40].

3 PART 3: SELF-DESIGNED AND LEARNED SYSTEMS

Self-designed Systems. Self-designed systems rely on the notion of mapping the possible space of critical design decisions in a system. For example, the Data Calculator introduced the design space of key-value storage [31]. The design space is defined by all designs that can be described as combinations and tunings of the "first principles of design". A first principle is a fundamental design concept that cannot be broken into additional concepts, e.g., for data structure design: fence pointers, links, temporal partitioning, and so on. The intuition is that, over the past decades, researchers have invented numerous fundamental design concepts such that a plethora of new valid designs with interesting properties can be synthesized out of those. The design space presented in [31] is shown to cover state-of-the-art designs, but it also reveals that a massive number of additional storage designs can be derived. As an analogy consider the periodic table of elements in chemistry; it categorized existing elements, but it also predicted unknown elements and their properties. In the same way, we can create the periodic table of data structures [30] which describes more key-value store designs than stars on the sky. Similar efforts have created design spaces in cache coherency protocols for database servers [19] and the design of parallel algorithms [49].

A self-designed system uses the design space to automatically generate designs that fit best a target workload and hardware. To do that we need to know how the various points in the space differ in terms of the performance properties they give to the resulting system. For example, learned cost models [31] is a method that enables learning the costs of fundamental access patterns (random access, scan, sorted search) out of which we can synthesize the costs of complex algorithms for a given data structure specification. These costs can, in turn, be used by machine learning algorithms that iterate over machine generated data structure specifications to label designs, and to compute rewards, deciding which specification to try out next. For example, early results using genetic algorithms [29] and dynamic programming [31] show the strong potential of such approaches to automatically discover close to optimal storage designs. In addition, design continuums [16, 17, 26] is another direction which allows for accurate fast search for the best design. A design continuum is a performance hyperplane that connects a specific subset of designs within the set of all possible designs. Design continuums are effectively a projection of the design space, a "pocket" of designs where we can identify unifying properties among its members. Early results show that there is a design continuum that covers B-tree, LSM-tree, and Log + index key-value stores [26].

Learned Systems. In learned systems traditional core data system components are replaced with models. For example a learned index [43] replaces the index part of a data structure with a model. The model may be anything from a simple linear model, a hierarchy of models, to an arbitrarily complex neural network. In turn, the data structure may be the structure used for base storage, secondary indexing or any other data structure needed in the system. Models capture data properties that in many cases are hard to capture with a generic one size fits all data structure design that has predefined workloads it supports well. In addition, a model in most cases will require much less data to represent the index information needed in a data structure, resulting in a design with much less memory footprint than standard designs. This is important especially in cloud environments where the amount of memory is a significant cost factor. In addition, replacing key-based metadata in an index with a model, effectively shifts the cost of traversing an index from memory bound to CPU bound which favors modern hardware. Early results with learned indexes [43] show that it is possible to match or even outperform traditional indexes while having a much smaller memory footprint.

The principle of replacing traditional system components with models can be applied in numerous areas of data systems design. For example recent work on learned cardinality estimation [35] shows promising results on tackling one of the oldest problems in database optimization which routinely leads to bad query plans due to stale statistics. In addition, recent work on learned optimizers shows promising results in optimizing complex query plans with many joins [48]. SageDB is a recent system proposal for a holistic design of a data system where learned components are a first class citizen in its design [42]. Models can help in many more areas, e.g., they can even be used as data representation method to store old data or to perform approximate processing [34].

Research Opportunities. The new opportunity with self-designed and learned systems is the ability to design data systems that can provide a wider range of performance behaviors compared to past designs that relied only on auto-tuning and adaptivity. For example, a self-designed system can assume any behavior as long as it is part of its design space while a learned system can assume any behavior that the models can support. In both cases we can assume many more behaviors than what a fixed a priori design can do. There are numerous open research problems with both directions, e.g., for self-designed systems: efficient search of the optimal design, quick and efficient code generation of the target design, easy extensibility to expand the supported design space, and for learned systems: efficient storage of models, efficient execution of complex models with modern hardware, robustness and interpetability of performance results.

Further, there are exciting research and practical opportunities that arise from the fusion of learned and self-designed systems as well as workload-triggered adaptivity [27], and modern takes on auto-tuning with machine learning [1]. The long term agenda is toward building systems that have models and design spaces as first class citizens while being able to adapt to query workloads as well as auto-tune any knobs exposed. In practice all systems expose knobs sooner or later, e.g., to accommodate special performance or feature needs that arise after the system goes on production.

Applications: Relational, Big Data, Data Science. In the last part of the tutorial we touch on how the new directions described can apply across numerous data-intensive areas beyond relational systems. We discuss NoSQL systems (e.g, LSM-trees, B-trees, and Log+index systems), as well as broad data science applications such as statistics-heavy processing and machine learning systems. Effectively, all these areas have in common the need to process ever growing amounts of data. The data storage and exact processing algorithms supported need to vary depending on the exact access patterns desired by the high level algorithms/workloads and by the hardware. As data grows, even the slightest sub-optimality in these decisions can cost anything from hours to days in processing. The new ideas presented in this tutorial showcase open research problems towards being able to generate close to optimal storage systems for such data-intensive applications.

4 AUDIENCE AND OUTPUT

Audience. The target audience for this tutorial is students, academics, researchers and software engineers with basic knowledge on data structures, algorithms and data system design. We assume basic understanding of fundamental data structures such as B-trees, LSM-trees, and Hash-tables. In addition, we assume basic knowledge of database system architectures, their components and how they broadly interact, e.g., a high level understanding of cost based optimizers, execution engine, storage engine, row-stores, and column-stores. The tutorial is self-contained in providing all necessary background, no prior knowledge is needed on auto-tuning, adaptive systems/indexing, self-designed and learned systems.

Output. The target learning output is as follows:

(1) understanding the need of building new tailored systems that match the application needs
(2) understanding the long term technical limitations of state-of-the-art knob-based systems
(3) exposure to the new research challenges with self-designed and learned systems
(4) exposure to the new opportunities across diverse data-intensive contexts: relational, NoSQL, and data science

5 PRESENTERS

Stratos Idreos is an associate professor of Computer Science at Harvard University where he leads the Data Systems Laboratory. His research focuses on making it easy and even automatic to design workload and hardware conscious data structures and data systems with applications on relational, NoSQL, and broad data science and data exploration problems. Stratos was awarded the ACM SIGMOD Jim Gray Doctoral Dissertation award for his thesis on adaptive indexing. He received the 2011 ERCIM Cor Baayen award as "most promising European young researcher in computer science and applied mathematics" from the European Research Council on Informatics and Mathematics. He won the 2011 Challenges and Visions best paper award in the Very Large Databases conference as well as "best of conference" selections at VLDB 2012 and SIGMOD 2017. In 2015 he was awarded the IEEE TCDE Rising Star Award from the IEEE Technical Committee on Data Engineering for his work on adaptive data systems. Stratos is also a recipient of the IBM zEnterpise System Recognition Award, a Facebook Faculty award, a NetApp Faculty award, and an NSF Career award.

Tim Kraska is an Associate Professor of Electrical Engineering and Computer Science in MIT's Computer Science and Artificial Intelligence Laboratory. His research focuses on building systems for machine learning, as well as using machine learning for systems with a broad goal to replace traditional bulky system components with efficient and succinct models. In addition, Tim works on democratizing data science with tools for interactive and visual data exploration. Before joining MIT, he was an Assistant Professor at Brown University, spent time at Google Research, and was a PostDoc in the AMPLab at UC Berkeley after getting his Ph.D. from ETH Zurich. Tim is a 2017 Alfred P. Sloan Research Fellow in computer science and received the 2017 VMware Systems Research Award, an NSF CAREER Award, an Air Force Young Investigator award, two Very Large Data Bases (VLDB) conference best-demo awards, and a best-paper award from the IEEE International Conference on Data Engineering (ICDE). Tim is also the winner of the 2018 VLDB early career award for advancing systems research on interactive data analytics.

6 ACKNOWLEDGMENTS

This work is partially funded by the USA National Science Foundation project IIS-1452595.

REFERENCES

[1] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. *SIGMOD* (2017).

[2] Ioannis Alagiannis, Stratos Idreos, and Anastasia Ailamaki. 2014. H2O: A Hands-free Adaptive Store. *SIGMOD* (2014).

[3] Victor Alvarez, Felix Martin Schuhknecht, Jens Dittrich, and Stefan Richter. 2014. Main Memory Adaptive Indexing for Multi-Core Systems. *DAMON* (2014).

[4] Michael R. Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J. Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. 2013. Brainwash: A Data System for Feature Engineering. *CIDR* (2013).

[5] Paul M Aoki. 1998. Generalizing "Search" in Generalized Search Trees (Extended Abstract). *ICDE* (1998).

[6] Paul M Aoki. 1999. How to Avoid Building DataBlades That Know the Value of Everything and the Cost of Nothing. *SSDBM* (1999).

[7] Joy Arulraj, Andrew Pavlo, and Prashanth Menon. 2016. Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads. *SIGMOD* (2016).

[8] Manos Athanassoulis, Michael S. Kester, Lukas M. Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki, and Mark Callaghan. 2016. Designing Access Methods: The RUM Conjecture. *EDBT* (2016).

[9] Shivnath Babu, Nedyalko Borisov, Songyun Duan, Herodotos Herodotou, and Vamsidhar Thummala. 2009. Automated Experiment-Driven Management of (Database) Systems. *HotOS* (2009).

[10] Don S Batory, J R Barnett, J F Garza, K P Smith, K Tsukuda, B C Twichell, and T E Wise. 1988. GENESIS: An Extensible Database Management System. *TSE* 14, 11 (1988), 1711–1730.

[11] Nicolas Bruno and Surajit Chaudhuri. 2005. Automatic physical database tuning. *SIGMOD* (2005).

[12] Michael J Carey and David J DeWitt. 1987. An Overview of the EXODUS Project. *IEEE DEBULL* 10, 2 (1987), 47–54.

[13] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *VLDB* (1997).

[14] Surajit Chaudhuri and Gerhard Weikum. 2000. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. *VLDB* (2000).

[15] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2017. Monkey: Optimal Navigable Key-Value Store. *SIGMOD* (2017).

[16] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2018. Optimal Bloom Filters and Adaptive Merging for LSM-Trees. *TODS* (to appear (2018).

[17] Niv Dayan and Stratos Idreos. 2018. Dostoevsky: Better Space-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging. *SIGMOD* (2018).

[18] Jens Dittrich and Alekh Jindal. 2011. Towards a One Size Fits All Database Architecture. *CIDR* (2011).

[19] Michael J Franklin. 1993. *Caching and Memory Management in Client-Server Database Systems.* Ph.D. Dissertation. University of Wisconsin-Madison.

[20] David Goldhirsch and Jack A Orenstein. 1987. Extensibility in the PROBE Database System. *IEEE DEBULL* 10, 2 (1987), 24–31.

[21] Goetz Graefe. 1994. Volcano - An Extensible and Parallel Query Evaluation System. *TKDE* 6, 1 (feb 1994), 120–135.

[22] Goetz Graefe, Felix Halim, Stratos Idreos, Harumi Kuno, and Stefan Manegold. 2012. Concurrency control for adaptive

indexing. *PVLDB* 5, 7 (2012), 656–667.
[23] Richard A Hankins and Jignesh M Patel. 2003. Data Morphing: An Adaptive, Cache-Conscious Storage Technique. *VLDB* (2003).
[24] Max Heimel, Martin Kiefer, and Volker Markl. 2015. Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation. *SIGMOD* (2015).
[25] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. 1995. Generalized Search Trees for Database Systems. *VLDB* (1995).
[26] Stratos Idreos, Niv Dayan, Wilson Qin, Mali Akmanalp, Sophie Hilgard, Andrew Ross, James Lennon, Varun Jain, Harshita Gupta, David Li, and Zichen Zhu. 2019. Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn. In *CIDR (2019)*.
[27] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007. Database Cracking. *CIDR* (2007).
[28] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2009. Self-organizing Tuple Reconstruction in Column-Stores. *SIGMOD* (2009).
[29] Stratos Idreos, Lukas M Maas, and Mike S Kester. 2017. Evolutionary Data Systems. *CoRR* abs/1706.0 (2017). arXiv:1706.05714
[30] Stratos Idreos, Kostas Zoumpatianos, Manos Athanassoulis, Niv Dayan, Brian Hentschel, Michael S. Kester, Demi Guo, Lukas M. Maas, Wilson Qin, Abdul Wasay, and Yiyou Sun. 2018. The Periodic Table of Data Structures. *IEEE DEBULL* 41, 3 (2018), 64–75.
[31] Stratos Idreos, Kostas Zoumpatianos, Brian Hentschel, Michael S Kester, and Demi Guo. 2018. The Data Calculator: Data Structure Design and Cost Synthesis from First Principles and Learned Cost Models. *SIGMOD* (2018).
[32] Yannis E Ioannidis and Eugene Wong. 1987. Query Optimization by Simulated Annealing. *SIGMOD* (1987).
[33] Oliver Kennedy and Lukasz Ziarek. 2015. Just-In-Time Data Structures. *CIDR* (2015).
[34] Martin L. Kersten and Lefteris Sidirourgos. 2017. A Database System with Amnesia. In *CIDR (2019)*.
[35] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR (2019)*.
[36] Yannis Klonatos, Christoph Koch, Tiark Rompf, and Hassan Chafi. 2014. Building Efficient Query Engines in a High-Level Language. *PVLDB* 7, 10 (2014), 853–864.
[37] Marcel Kornacker. 1999. High-Performance Extensible Indexing. *VLDB* (1999).
[38] Marcel Kornacker, C Mohan, and Joseph M. Hellerstein. 1997. Concurrency and Recovery in Generalized Search Trees. *SIGMOD* (1997).
[39] Marcel Kornacker, Mehul A. Shah, and Joseph M. Hellerstein. 1998. amdb: An Access Method Debugging Tool. *SIGMOD* (1998).
[40] Marcel Kornacker, Mehul A. Shah, and Joseph M. Hellerstein. 2003. Amdb: A Design Tool for Access Methods. *IEEE DEBULL* 26, 2 (2003), 3–11.
[41] Donald Kossman. 2018. Systems Research - Fueling Future Disruptions. In *Keynote talk at the Microsoft Research Faculty Summit*. Redmond, WA, USA.
[42] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed Chi, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. 2019. SageDB: A Learned Database System. In *CIDR (2019)*.
[43] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. *SIGMOD* (2018).
[44] Justin J. Levandoski, David B. Lomet, and Sudipta Sengupta. 2013. LLAMA: A Cache/Storage Subsystem for Modern Hardware. *PVLDB* 6, 10 (2013), 877–888.
[45] Justin J. Levandoski, David B. Lomet, and Sudipta Sengupta. 2013. The Bw-Tree: A B-tree for New Hardware Platforms. *ICDE* (2013).
[46] Zezhou Liu and Stratos Idreos. 2016. Main Memory Adaptive Denormalization. *SIGMOD* (2016).
[47] Martin Luhring, Kai-Uwe Sattler, Karsten Schmidt, and Eike Schallehn. 2007. Autonomous Management of Soft Indexes. *ICDEW* (2007).
[48] Ryan C. Marcus and Olga Papaemmanouil. 2019. Towards a Hands-Free Query Optimizer through Deep Learning. In *CIDR (2019)*.
[49] Tim Mattson, Beverly Sanders, and Berna Massingill. 2004. *Patterns for Parallel Programming*. Addison-Wesley Professional.
[50] John McPherson and Hamid Pirahesh. 1987. An Overview of Extensibility in Starburst. *IEEE DEBULL* 10, 2 (1987), 32–39.
[51] Sylvia L Orborn. 1987. Extensible Databases and RAD. *IEEE DEBULL* 10, 2 (1987), 10–15.
[52] David Lorge Parnas. 1979. Designing Software for Ease of Extension and Contraction. *TSE* 5, 2 (1979), 128–138.
[53] Eleni Petraki, Stratos Idreos, and Stefan Manegold. 2015. Holistic Indexing in Main-memory Column-stores. *SIGMOD* (2015).
[54] Holger Pirk, Eleni Petraki, Stratos Idreos, Stefan Manegold, and Martin L. Kersten. 2014. Database cracking: fancy scan, not poor man's sort! *DAMON* (2014).
[55] Karl Schnaitter, Serge Abiteboul, Tova Milo, and Neoklis Polyzotis. 2006. COLT: Continuous On-Line Database Tuning. *SIGMOD* (2006).
[56] Felix Martin Schuhknecht, Alekh Jindal, and Jens Dittrich. 2013. The Uncracked Pieces in Database Cracking. *PVLDB* 7, 2 (2013), 97–108.
[57] Daniel Dominic Sleator and Robert Endre Tarjan. 1985. Self-Adjusting Binary Search Trees. *J. ACM* 32, 3 (1985), 652–686.
[58] Michael Stonebraker, Jeff Anton, and Michael Hirohama. 1987. Extendability in POSTGRES. *IEEE DEBULL* 10, 2 (1987), 16–23.
[59] Michael Stonebraker and Ugur Cetintemel. 2005. "One Size Fits All": An Idea Whose Time Has Come and Gone. *ICDE* (2005).
[60] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for interactive exploration of big data series. *SIGMOD* (2014).