Reminiscences on Influential Papers

This issue's contributors cover works that are results of paradigm shifts where data-intensive systems expanded beyond relational model, disk-based performance optimizations, one-dimensional data, and "small" scale. Enjoy reading!

While I will keep inviting members of the data management community, and neighboring communities, to contribute to this column, I also welcome unsolicited contributions. Please contact me if you are interested.

Pınar Tözün, *editor*
IT University of Copenhagen, Denmark
pito@itu.dk

Genoveva Vargas-Solar
CNRS, University of Lyon, INSA Lyon, UCBL, LIRIS, UMR5205, Lyon, France
genoveva.vargas-solar@cnrs.fr

Stella Gatziu, Andreas Geppert, and Klaus R. Dittrich.
The SAMOS active DBMS prototype.
In Proceedings of the International Conference on Management of Data (SIGMOD), 1995.

It was the time when the database community was expanding beyond the relational model-embracing object-oriented paradigms, rethinking consistency guarantees, and redesigning DBMS architectures. I began my PhD in the midst of this momentum, focusing on the challenges posed by active rules (Event–Condition–Action - ECA). I joined a team working on early active object-oriented databases via NAOS [1] which was the active rules component of the object oriented database system O_2. Equipped with seminal papers on the first wave of approaches from the 1990s [6], I set out to compare, grasp, and replicate these proposals to prepare for meaningful contributions. My project focused on specifying and detecting simple and composite events arising not only from database operations but also from sources external to the DBMS. Of the surveyed systems, SAMOS (Swiss Active-Mechanism based Object-oriented database System) [3] was particularly influential for its rigorous event model and its efficient detection algorithm for complex event expressions. SAMOS was an active, object-oriented DBMS prototype from the University of Zurich. It added ECA rules on top of a passive OODB (ObjectStore), using a layered architecture so the base DBMS remained unmodified. Rules were compiled, stored as objects, and executed by runtime components (rule manager, composite-event detector, rule execution, SAMOS transactions). SAMOS allowed to specify **primitive** and **composite event** types. Primitive event types included time-based, method, invocations, value changes, transactional, and application-defined event types. Composite event types, were built using an *event algebra* combining *sequence* ($e_1; e_2$), *conjunction* (e_1, e_2), *disjunction* ($e_1 \mid e_2$), *negation* ($\neg e$ within an interval), and *occurrence-reduction* operators (e.g., times(e, n), first occurrence). **Parameters and temporal scopes** (e.g., object identifier, transaction identifier, time windows) constrained which occurrences participated in a composite and when to notify it. **Execution** respected **coupling modes** with respect to the execution scope of a transaction (*immediate*, *deferred* (within or at the end of the triggering transaction), or *decoupled* (separate transaction), while **priorities** order competing rules on the same event. Conditions and actions were expressed in the host OODB's DML (Data Manipulation Language) (e.g., C++/ObjectStore).

In SAMOS, primitive events were notified at runtime via `raise_event` and matched, using indexed descriptors, against the catalog of rules that may respond. Detection of composite events proceeded incrementally: each primitive event advanced a *Colored Petri net* that encoded the event pattern, and a composite was declared when the net reached its terminal marking. Rule execution adopted coupling modes through a dedicated `samTransaction` mechanism, which controlled immediate, deferred, or decoupled firing, handled nested triggers, and ensures recovery. Finally, SAMOS persisted an event history so that complex patterns could span multiple operations and, when configured, even multiple transactions or sessions. The SAMOS event-detection line of work anticipated several problems that later became central in stream processing and continuous querying. In particular, it tackled composite event detection and the management of validity scopes (i.e., detection and notification windows) both essential for observing and reacting to the evolving state of complex systems, whether within a DBMS and its applications, across distributed infrastructures, or in physical domains such as

urban systems, traffic networks, and stock markets.

The work around SAMOS, and active databases more broadly, shaped my research playbook: strategies, methodologies, and results that taught me how to balance revisiting core disciplinary problems with building real systems atop commercial platforms. Equally important, it trained me to move fluidly between theory and implementation. I then had the chance to join the University of Zurich's database group for my postdoc, a hands-on immersion in the very ideas I had studied through their papers and books (and other seminal works), now tested and extended in practice.

In an era where high-velocity data underpins many analytics challenges, searching for and discovering event patterns is central. Consolidating the knowledge built over previous decades provides a solid foundation to address today's expectations and technological shifts (storage architectures, distribution, compute capacity, virtualization, and scale) to push event-centric analytics forward with rigour and out of the box perspectives.

Thomas Heinis
Imperial College London, UK
t.heinis@imperial.ac.uk

Antonin Guttman.
R-trees: A Dynamic Index Structure for Spatial Searching.
In Proceedings of the International Conference on Management of Data (SIGMOD), 1984.

Although some may regard it as a tried and tired topic, I do find spatial indexing genuinely fascinating! It extends the indexing problem into multiple dimensions and introduces a set of challenges that are both distinctive and intellectually compelling. In particular, the total order that underpins most one-dimensional index structures does not carry over to higher dimensions. This paper [4], however, finds an elegant solution that would shape my career and the field.

I first encountered the paper at the start of my postdoc, when I began working on indexing scientific data, and it proved pivotal in setting me on a path toward developing many more spatial indexes. It has influenced not only how I think about indexing, but also how I approach academic writing. At the time, publishing in this area felt especially challenging because the field already seemed crowded - there was no shortage of prior work to contend with. In hindsight, I realise this is probably true of many research topics, but it certainly felt differently then.

As incredible as its impact on my career felt, the importance of the paper for the community, however, is far more important. It's certainly not the first paper on spatial indexing but surely a seminal one in the development and teaching of spatial indexes.

One of the striking aspects of the paper is its clarity and restraint. The premise is simple: many applications store objects that live in multi-dimensional space - rectangles, regions, points, trajectories - and we need an index that supports efficient spatial queries such as "find all objects that intersect this window." Instead of trying to shoehorn these objects into a one-dimensional order with elaborate space-filling curves or ad hoc encodings, Guttman starts from geometry itself: bound each object by a minimal bounding rectangle (MBR) and build a balanced tree where each internal node stores the MBR of its children. The R-tree is, in some sense, the obvious structure once you see it. But before this paper, "obvious" was not how the database community thought about spatial indexing.

The technical core of the paper is a tour through the practical design space of such an index. It does not hide behind a purely theoretical formulation. Instead, it works through the gritty details that any real system builder must face: how to split overflowing nodes, how to choose where to insert a new rectangle so as to control overlap and dead space, and how to propagate structural changes while preserving balance and reasonable page utilization. The split heuristics are not presented as abstract optimization problems, but as concrete trade-offs between CPU time and index quality. As a budding researcher, this was one of the first times I saw a database paper so explicitly embrace "engineering knobs" as first-class research contributions. One of the aspects that would also shape my personal approach to research and its presentation.

Equally important is what the paper does not do. It does not claim that the proposed heuristics are optimal. It does not solve every variant of spatial query or every workload pattern. Instead, it defines a clean abstraction and shows that it is versatile enough to support a variety of queries (point, range, nearest neighbour-like patterns) with competitive performance. This balance between conceptual simplicity and practical completeness is a hallmark of enduring systems work, and R-trees are an exemplary case. There is no question that this paper is and was hugely influential in the community. R-trees quickly became the de facto access method for spatial data in both research prototypes and industrial systems. They influenced spatial extensions in relational databases, underpinned early GIS engines, and inspired a whole family of variants. Many generic index frameworks, explicitly draw on the idea that you can parameterize a balanced tree by a bounding predicate and a set of split/merge policies - an idea that Guttman's design makes tangible. Even today, when we talk about "spatial indexes" in production systems, more often than not we mean "some descendant of an R-tree."

And there are many "descendants" of the R-Tree: R-tree, R+-tree, R*-tree, Hilbert R-tree, Packed R-tree, STR packed R-tree, R-link tree, R1-tree, R-file, 2+3 R-tree, 3DR-tree, MV3R-tree, MVR-tree, HR-tree, RT-tree, PR-tree, TPR-tree, TPR*-tree, BPR-tree, VCI R-tree, ATPR-tree, STR-tree, STRIPES, STRIPES-tree, Bx-tree, Bdual-tree, PR bucket quadtree, IR-tree, uR-tree, GiST, R-tree-over-GiST, SP-GiST, MGiST, MSP-GiST, RD-tree, X-tree, TV-tree, SR-tree, SS-tree, A-tree, P+-tree, Pyramid-technique, Pyramid-tree, VA-file, VA+-file, iDistance, iMinMax(θ), iMinMax(θ), LSB-

tree, LSB forest, D-index, Local Dimensionality Reduction (LDR) index, Global Index (GI), Cluster index, Hybrid tree, M-tree, Slim-tree, PM-tree, Pivoting M-tree, MVP-tree, mvp-tree, VP-tree, GNAT, EGNAT, SAT, M+-tree, DBM-tree, DBM-tree, LAESA, OMNI, B-tree, B+-tree, B*-tree, B-link tree, ISAM, 2-D ISAM, linear hashing, extendible hashing, multidimensional extendible hashing, multidimensional order-preserving linear hashing (MOLHPE), k-d tree, adaptive k-d tree, bintree, BSP-tree, BD-tree, GBD-tree, extended k-d tree, spatial k-d tree (skd-tree), k-d-B-tree, hB-tree, hBP-tree, BV-tree, Cell tree, JP-tree, SP-tree, P-tree, PLOP-hashing, grid file, multilevel grid file, two-level grid file, twin grid file, generalized grid file, multilayer grid file, BANG file, buddy tree, z-ordering, Z-order index, Morton order, Hilbert order, Hilbert curve index, UB-tree, quadtree, PR quadtree, PM quadtree, PMR quadtree, trie, Linear R-tree, Quadratic R-tree, Greene's R-tree, Priority R-tree, Packed Hilbert R-tree, Dynamic Hilbert R-tree, Nearest-X packed R-tree, OMT packed R-tree, TGS R-tree, ND R-tree, Cache-oblivious R-tree, Buffered R-tree, Rsb-tree, RR-tree, RUM-tree, LGUR-tree, FUR-tree, RW-tree, AI+R-tree, 3D R-tree, HR+-tree, MR-tree, STAR-tree, REXP-tree, R1,2D3-tree, R1+2D3-tree, Parametric R-tree, NSI R-tree, STP-tree, GG TPR-tree, HTPR*-tree, RPPF-tree, PCFI+-index, TB-tree, CSE-tree, ANR-tree, FNR-tree, S-TPR-tree, RST-tree, IR2-tree, IRLi-tree, DIR-tree, CIR-tree, CDIR-tree, WIR-tree, WIBR-tree, KR*-tree, SKI, R*-IF, S2I, aR-tree, -tree, S-tree, Y-tree, Z-tree, cR-tree, CR-tree, CR*-tree, Hilbert CR-tree, H-R-tree, MON-tree, UTR-tree, LUR-tree, RUM+-tree, UST-tree, CT-R-tree, Sketch RR-tree, Bulk-load TPR-tree, Horizon TPR-tree, HTPR-tree, LGU TPR-tree, VRTPR-tree, CRTPR-tree, MVTPR-tree, TPRU-tree, VTPR-tree, RP-tree, RTR-tree, TP2R-tree, KR-tree, OST-tree, 3D hybrid R-tree (hybrid 3D R-tree), DR-tree, PA-tree, GS-tree, PK-tree, PH-tree, By-tree, Bs-tree, VR-tree ...

I could go on. Probably not really, but ChatGPT surely could!

The paper itself dates back to 1984 and survives in a charmingly antiquated scanned form. As far as I can tell, it was Guttman's final publication. I tried to contact him, but his LinkedIn profile indicates that he is now retired - and, I feel, very much deservedly so. I find it remarkable that someone could write a seminal, single-author paper - without even his supervisor, Mike Stonebraker - that has attracted more than 12,000 citations, and then choose not to pursue an academic career but instead move into industry - it has the energy of a mic drop moment, but in the context of academia!

Alexander Boehm
SAP SE, Germany
alexander.boehm@sap.com

Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner.
SIMD-Scan: Ultra Fast in-Memory Table Scan using on-Chip Vector Processing Units.
In Proceedings of the VLDB Endowment, 2009.

I visited VLDB 2009 as a last year PhD student, looking for interesting topics to work on after graduating. One paper especially caught my eye, as the set of authors was quite remarkable: A joint team from Intel, SAP, and the University of Potsdam, including SAP co-founder Hasso Plattner, proposed novel techniques for ultra-fast, hardware-accelerated database scans [9].

With the ultimate goal of creating an in-memory database management system (DBMS) unifying both transactional and analytical workloads in mind (as outlined by Plattner's SIGMOD 2009 keynote [7]), one of the key challenges that the paper addresses is to strike a balance between high data compression and fast query processing. At first glance, these goals seem to be conflicting: High compression is essential for in-memory databases as it reduces the amount of costly DRAM that is required for data storage. However, the decompression effort during query processing creates additional overhead, potentially reducing system performance, which is *the* key value proposition of an in-memory DBMS.

In the paper, the authors show how vectorized processing using Single Instruction, Multiple Data (SIMD) instructions can be used to efficiently decompress bit-encoded data (for bit lengths up to 32 bits). Moreover, they discuss techniques for the fast evaluation of range predicates using vectorized instructions, thereby significantly speeding up the scan operation. A follow-up paper from ADMS 2013 [8] shows how to leverage the more modern AXV2 vector processing instructions up to the point where memory bandwidth is becoming a bottleneck, underlining how meaningful and powerful the proposed concepts still are in the context of today's modern CPUs.

Apart from being impressive engineering, the key insight from the paper is what remarkable opportunities emerge when people collaborate across various domains. By bringing together hardware experts and database engineers, the paper demonstrates the high impact that can be achieved with hardware/software co-design. While the database community has recognized this importance early on as e.g. demonstrated by SIGMOD's DAMON workshop which is successfully running since over 20 years now, papers like the one by Willhalm et al. are an outstanding example of the practical benefits that can be achieved.

To me, similar opportunities exist when looking "up" the processing stack towards the applications and application servers running "on top" of database management systems. Like with hardware/software co-design, there is the potential to optimize the end-to-end system performance by taking a joint look from an application and database perspective, leading to application/database co-design.

Pamela Delgado
HEIG-VD, HES-SO, Switzerland
pamela.delgado@heig-vd.ch

Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica.

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.

In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI), 2011.

I started my PhD during the aftermath of the big data era, where its large-scale nature spawned several interesting systems and data management challenges. Many of the methods and principles introduced during that decade are still relevant, and some *incarnation* of them are used in today's platforms. While some influential papers, like MapReduce [2], are widely recognized, others have had a significant impact in ways that are less immediately visible. One such paper is *Mesos: A platform for Fine-Grained resource sharing in the data center* [5]. As I explored the state of the art in resource management during the early stages of my PhD, this particular work really stood out.

One particular feature of Mesos was its two level scheduling design. The original idea introduced in Mesos was to have a thin coarse-grained scheduling layer and leave the detailed decisions to the corresponding framework. Concretely, the fact of combining more than one type of scheduling co-existing in a cluster was inspiring for our work on hybrid scheduling.

Another aspect of this paper that resonated with me was the principle of including flexibility by design: the ability to accommodate a wide range of use cases (in the case of Mesos, Hadoop, MPI, and Torque batch scheduling) in order to use resources more efficiently. Flexibility is indeed something I still value deeply when designing systems today. This flexibility also applies to hardware heterogeneity, an aspect that is highly relevant in current Machine Learning ecosystems.

One more arguably relevant aspect is data locality. Over the years the importance of this aspect has changed from a non-issue for centralized systems, to crucial for distributed systems under potential network bottleneck limitations. In the GPU and LLM era this is an aspect that has gained even more prominence, motivating the revision of ideas as the ones exposed in Mesos.

Nowadays, one can trace the influence of Mesos in widely used systems for orchestration and deployment of applications, such as Kubernetes. Such systems offer a finer-grained resource management approach when sharing resources—one of the key ideas introduced in Mesos. Another such example is the idea of proposing a two-level scheduler, also seen in systems like YARN.

Of course, the proposed system was not flawless. For instance, Mesos offers resources, but frameworks can reject this offer, which has its impracticalities and challenges. However, the ideas and methods introduced in this work were and still are definitely out-of-the-box and inspiring not just to me, but also to other researchers and systems engineers. The imprint of its design and the principles it introduced have found wide adoption in a broad range of applications.

1. REFERENCES

[1] Christine Collet, Thierry Coupaye, and T Svensen. NAOS-Efficient and modular reactive capabilities in an Object-Oriented Database System. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 132–143, 1994.

[2] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[3] Stella Gatziu, Andreas Geppert, and Klaus R. Dittrich. The SAMOS active DBMS prototype. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, page 480, New York, NY, USA, 1995. Association for Computing Machinery.

[4] Antonin Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, page 47–57, New York, NY, USA, 1984. Association for Computing Machinery.

[5] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, page 295–308, USA, 2011. USENIX Association.

[6] Norman W Paton and Oscar Diaz. Active database systems. *ACM Computing Surveys (CSUR)*, 31(1):63–103, 1999.

[7] Hasso Plattner. A Common Database Approach for OLTP and OLAP Using an in-Memory Column Database. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 1–2. ACM, 2009.

[8] Thomas Willhalm, Ismail Oukid, Ingo Müller, and Franz Faerber. Vectorizing Database Column Scans with Complex Predicates. In Rajesh Bordawekar, Christian A. Lang, and Bugra Gedik, editors, *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2013, Riva del Garda, Trento, Italy, August 26, 2013*, pages 1–12, 2013.

[9] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner. SIMD-Scan: Ultra Fast in-Memory Table Scan using on-Chip Vector Processing Units. *Proc. VLDB Endow.*, 2(1):385–394, 2009.