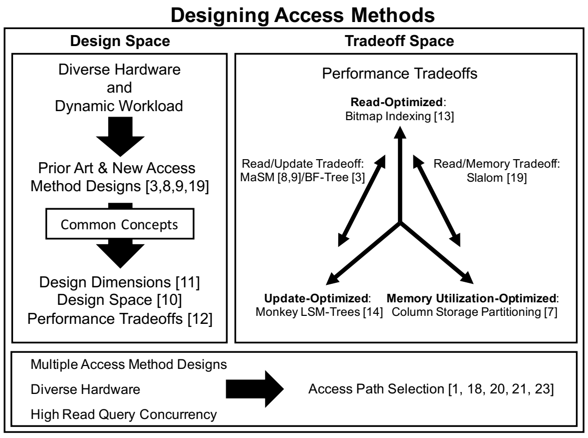
Manos Athanassoulis – Research Statement


Designing Access Methods
Design Space
Tradeoff Space
Diverse Hardware and Dynamic Workload
Prior Art & New Access Method Designs [3,8,9,19]
Common Concepts
Design Dimensions [11]
Design Space [10]
Performance Tradeoffs [12]
Performance Tradeoffs
Read-Optimized: Bitmap Indexing [13]
Read/Update Tradeoff: MaSM [8,9]/BF-Tree [3]
Read/Memory Tradeoff: Slalom [19]
Update-Optimized: Monkey LSM-Trees [14]
Memory Utilization-Optimized: Column Storage Partitioning [7]
Multiple Access Method Designs
Diverse Hardware
High Read Query Concurrency
Access Path Selection [1, 18, 20, 21, 23]

My research aims to provide tools to maintain and access data collections in an *efficient* and *tunable* way. Data is at the epicenter of human activity, as scientists, governments, companies, and individual users generate and collect data at an *unprecedented rate.* At the heart of every data-intensive operation sits a data management layer, which is responsible for efficient *storage, maintenance,* and *access* to the data, by employing a variety of combinations of algorithms and data structures, called *access methods.*

Access Method *Design Space* and *Tradeoff Space*. My research focuses on the different design elements that, when combined, produce an access method, and studies their impact on the tradeoff space between read performance, update performance, and memory utilization. My research results have been published in top data management venues (like SIGMOD, VLDB, TODS, VLDBJ). My work has received the "Best of VLDB 2010," "Best of SIGMOD 2017," and "SIGMOD 2016 Most Reproducible Paper" awards.

1 Recent and Past Work

The cost of data movement through the memory hierarchy, and the nature of new workloads that have hybrid transactional and analytical characteristics create conflicting design goals: fast read performance, high update rates, and small memory utilization. In my research I study the tradeoff between the read, update, and memory cost [12], and I associate the different design elements with their impact on the performance tradeoff [11], in order to build access methods that can navigate this tradeoff space [3, 7, 8, 9, 13, 14, 19]. Taking into account the rich design space of access methods, the diverse hardware, and the emerging workloads with hybrid analytics and increased concurrency I also revisited the question of access path selection [1, 18, 20, 21, 23].

1.1 A *Design Space* and a *Tradeoff Space* for Access Methods

My key observation is that the research efforts on building access methods repeatedly use a small set of *design dimensions* that form a conceptual *design space* of access methods [10, 11]. Each design combines different choices in a set of fundamental *design dimensions*, a process that effectively defines an access method design. Each design, in turn, addresses specific application requirements, with respect to optimizing for (i) read performance, (ii) update performance, and (iii) memory utilization. Following the classification of access methods based on (a) the design dimensions they employ and (b) the optimization of the Read, Update, and Memory (*RUM* for short) tradeoff space, each access method provides a balance between the three, and each design can shift this balance by optimizing for one of them at the expense of the other two [12].

1.2 Exploring the Design Space

To find the key design concepts of access methods and their impact on the RUM tradeoffs, I study a representative approach of each of the three extremes, one optimized for reads, one for updates, and one for memory utilization. **In this effort, I distill the key design components of each and develop new variations of these access methods that can navigate a wider spectrum of the performance tradeoff space.**

In particular, I study (i) bitmap indexing, which has low read overhead because the encoded bitvectors contain only the information necessary to identify the objects from a database matching the search query, (ii) log-structure merge trees (LSM-Trees), which have a minimal update overhead because updates and inserts are appended to and only gradually merged with the base data in an amortized fashion, and (iii) columnar storage without any additional structure, which has minimal memory and storage utilization.

Bitmap Indexing: A Read-Optimized Access Method. Bitmap indexes offer efficient read performance because of the highly compressible data representation through bitvectors they utilize. The main benefit is that the query result, e.g., all objects with a specific value, is readily available and has small size, because bitvectors can be heavily compressed through encoding. However, updating a bitmap index is expensive because it requires the costly cycle of decoding, updating, and re-encoding. **Using the abstractions of update buffering and query-driven reorganization, I presented at SIGMOD '16 a new bitmap index called UpBit, which allows for efficient updates employing adaptive merging updates with the main bitvectors [13].** This paper received the "Most Reproducible Paper" award.

LSM-Trees: An Update-Optimized Access Method. Log-Structure Merge Trees maintain data in a hierarchy of sorted arrays (*sorted runs*) of exponentially increasing size. Incoming data and updates are buffered in a small update buffer that, when full, is sorted and flushed on secondary storage. This iterative merging repeats for the following levels of the hierarchy; when there are enough sorted runs of the same length, they are merged into a larger sorted run. This data organization creates an increased lookup (multiple sorted runs might be visited for a single lookup). To limit this lookup cost, state-of-the-art solutions allocate memory for auxiliary data structures: fence pointers to avoid binary search on storage, and Bloom filters to avoid visiting a sorted run altogether. **At SIGMOD '17, we presented Monkey [14], a new LSM-Tree design that optimizes the main memory allocation for Bloom filters and allows for a tunable performance – between more read-optimized and more update-optimized – for any given memory budget.** This work was invited into the "Best of SIGMOD 2017" special issue of TODS.

Column Storage: A Memory Utilization-Optimized Access Method. By storing each attribute separately, a column-oriented physical layout offers efficient access to the desired attributes without bringing unnecessary data, typically, boosting analytical queries. Further, if the data is sorted based on one column, it also offers efficient searching on this attribute. Maintaining a second copy sorted on a different column further improves relevant read queries at the expense of higher memory utilization. In both cases, however, there is a high update cost because we must to touch multiple locations (one per data copy per column). **I study the balance of updating in-place data in columnar storage by employing partitioning and thus minimizing data movement [7].** We employ a range partitioning scheme with empty slots scattered throughout the column to facilitate mixed workloads. Reads target only the relevant partition and updates need to move data between partitions by swapping their elements. We develop a method to find the optimal partitioning scheme with respect to data movement for a given workload sample (or training set).

1.2.1 Designing Specialized Access Methods

I further developed access methods that offer different ways to balance the RUM overheads. In particular, **I developed MaSM (sort for Materialized Sort-Merge), a flash-aware access method for data analytics that effectively hides the update overhead from the read query cost [8, 9].** I further developed **Bloom filter Tree (BF-Tree), the first approximate tree indexing approach, which has tunable index size at the expense of a tunable, small number of false positives [3].** The impact of false positives can be mitigated easily, particularly when the data is stored on flash devices. BF-Tree is very efficient for ordered or clustered data and it supports updates at the cost of increased false positives. To deliver efficient indexing without a priori workload knowledge we designed an in-situ query engine, called Slalom, that accommodates workload shifts. **Slalom makes on-the-fly partitioning and indexing decisions based on information collected by light-weight monitoring of user activity [19].** Slalom offers performance benefits by taking into account user query patterns to (i) logically partition raw data files and (ii) build for each partition lightweight partition-specific indexes (Bloom filters, Zonemaps, or B^+-Trees). Slalom can operate for any given main-memory budget.

1.3 Choosing the Right Access Method: Access Path Selection

Classifying, developing, and studying the performance tradeoffs of access methods for modern systems naturally raises the question of which one to use and under which conditions. As a result, we revisited the question of access path selection in data management systems in today's setting. We studied the impact of query concurrency in modern main-memory query engines [1], and then we studied the query engines that support work sharing, a natural run-time optimization for workloads with read concurrency [20, 21]. **Finally, putting everything together, we developed a new access path selection model for main-memory data analytics systems that takes into account the underlying hardware, the query selectivity, the dataset size, and the level of query concurrency [18].** We further investigated the efficiency of evaluating a predicate within the memory hierarchy by proposing and simulating with the gem5 cycle-accurate CPU simulator, a futuristic main-memory controller that can perform selection over column-stores directly in memory without moving the data through the memory hierarchy [23].

1.4 Other Topics in Data Management

Flash Storage in the Database Stack. In my PhD research, I also studied the performance tradeoffs and the impact of flash storage in the database stack for analytical and transactional workloads [2, 4]. I showed that a log-structured storage engine for a transactional workload better exploits flash as the underlying storage layer [22]. I further proposed a new physical storage model for graph-based data that can naturally exploit the efficient yet small I/O from flash devices [5, 6].

Transactional Processing: Scalable Write-Ahead Logging. Write-ahead logging is frequently on the critical path of transactional database processing. We investigate the impact of modern hardware – both processors and storage – on logging, and we present a new holistic approach for write-ahead logging called Aether [16]. **Aether involves three techniques: early-lock release (ELR), flush-pipelining, and a consolidation array of log requests using efficient in-memory locking [15].** This work was included in the VLDB Journal in the "Best of VLDB 2011" special issue. The extended version investigates further the scalability of logging in multi-socket servers [17].

2 Research Outlook

Overview. My long term research goal is to build data systems that can navigate a performance, functionality, and privacy tradeoff. To that end, I plan to work on every part of the full stack about how we reason for, design, and build data systems, starting from the underlying hardware, going all the way to the applications sitting on top. Motivated by the continuously shifting requirements of data-intensive applications and the increasing complexity of workloads, I have identified three key research directions: *developing efficient yet private data systems on public cloud infrastructure*, *building the necessary framework to exploit new and future hardware in future data systems designs*, and *studying storage tradeoffs more deeply in light of the disruptive advancements in storage technologies.* I am well-equipped to address these research challenges given my experience with designing and balancing the performance tradeoffs of access methods [3, 8, 9, 11, 12, 13, 14], and studying the system design implications of new hardware [2, 4, 5, 6, 15, 16, 17, 22].

Private Data Systems. To ensure that data in public clouds are kept private, database-as-a-service cloud providers encrypt them. Facilitating faster accesses on encrypted data through indexing leaks information; every index access reveals something more to a potential malicious eavesdropper. The fundamental research question is how *build secure indexes, consecutive accesses to which cannot be aggregated* to construct information about the dataset by the eavesdropper. A key challenge is to provide tunable guarantees; a user may be willing to allow more leakage than another. For example an indexing scheme with a limited, yet tunable, time-to-live can provide such guarantees. Similarly, using the inherently tunable differential privacy query answering allow for tunable privacy. An end-to-end tunable private data system can combine operating over encrypted data and operating with differential privacy as long as there are entities that can be trusted in the data management workflow. These components can now be orchestrated by an advisor which decides which subsystem to use. Data systems with *tunable privacy* go beyond the state of the art; they provide tunable and modular privacy in a cloud environment with *multi-tenancy.* That way the same infrastructure will be able to support multiple applications with different performance and privacy requirements.

Performance Tradeoffs of New Storage Technologies: A New I/O Model. A unifying theme of my research is to study and understand the performance characteristics and tradeoffs of new hardware in terms of processing, memory, and storage. A frequently faced challenge is to correctly model the different performance characteristics of the underlying hardware, such as the ability of flash devices to serve multiple concurrent I/Os with a negligible or small performance penalty [5]. I am very interested in defining an expressive, simple, easy-to-adopt new I/O model that would incorporate key characteristics of new and ideally future hardware, including the level of supported access concurrency, the potential performance asymmetry between reads and writes, and how the performance varies for different access granularity. The end goal of a new I/O model is to provide the tools to perform algorithm and system design with a clean abstraction that matches more accurately the key aspects of the underlying hardware. This approach will help understanding the underlying storage and memory of a single instance, but also, the storage offerings at the level of cloud offerings.

To give a concrete example, many algorithms today have been designed with the assumption that a read and write request have the same latency. Having this assumption it makes sense to evict one dirty page from a bufferpool when we want to read in one new page that is not there. In this example we will have to perform a write to facilitate a read. The write is typically handled by a background thread and there is a performance balance. However, as the writes become more expensive than reads, then this logical choice of paying the price of one write for this one read, becomes suboptimal. With the current I/O model we do not have enough tools to argue about a new design and, as a result, the problem is addressed with ad hoc solutions. A model that captures latency asymmetry, enables us to propose eviction policies and reason about their efficiency and impact.

Treating the Storage Hierarchy as a Variable. The memory and the storage hierarchy of computer systems is decided on chip designing time based on thorough benchmarking. Then, the design for the cache memories is fixed, and the main memory and secondary storage have a limited number of options. Consequently, software is build assuming a small set of fixed designs. I propose to turn this question around, and given a monetary, capacity, or energy budget and a workload, calculate the optimal memory hierarchy. In a world that computing is becoming exclusively cloud-based, calculating the "optimal memory hierarchy" would allows us to migrate in the cloud to chips closest to what a workload needs, and take advantage of a more heterogeneous cloud.

References

[1] Ioannis Alagiannis, Manos Athanassoulis, and Anastasia Ailamaki. Scaling up analytical queries with column-stores. In *Proceedings of the International Workshop on Testing Database Systems (DBTest)*, pages 8:1–8:6, 2013.

[2] Manos Athanassoulis. *Solid-State Storage and Work Sharing for Efficient Scaleup Data Analytics.* PhD thesis, École Polytechnique Fédérale de Lausanne, 2014.

[3] Manos Athanassoulis and Anastasia Ailamaki. BF-Tree: Approximate Tree Indexing. *Proceedings of the VLDB Endowment*, 7(14):1881–1892, 2014.

[4] Manos Athanassoulis, Anastasia Ailamaki, Shimin Chen, Phillip B. Gibbons, and Radu Stoica. Flash in a DBMS: Where and How? *IEEE Data Engineering Bulletin*, 33(4):28–34, 2010.

[5] Manos Athanassoulis, Bishwaranjan Bhattacharjee, Mustafa Canim, and Kenneth A. Ross. Path Processing using Solid State Storage. In *Proceedings of the International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*, pages 23–32, 2012.

[6] Manos Athanassoulis, Bishwaranjan Bhattacharjee, Mustafa Canim, and Kenneth A. Ross. Querying Persistent Graphs using Solid State Storage. In *Proceedings of the Annual Non-Volatile Memories Workshop (NVMW)*, 2013.

[7] Manos Athanassoulis, Kenneth S. Bøgh, and Stratos Idreos. Casper: Optimal Workload-Aware Column Layout for HTAP. *under submission*, 2017.

[8] Manos Athanassoulis, Shimin Chen, Anastasia Ailamaki, Phillip B. Gibbons, and Radu Stoica. MaSM: Efficient Online Updates in Data Warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 865–876, 2011.

[9] Manos Athanassoulis, Shimin Chen, Anastasia Ailamaki, Phillip B. Gibbons, and Radu Stoica. Online Updates on Data Warehouses via Judicious Use of Solid-State Storage. *ACM Transactions on Database Systems (TODS)*, 40(1), 2015.

[10] Manos Athanassoulis, Niv Dayan, and Stratos Idreos. Principles, Tradeoffs, and Opportunities in Data Access Method Design. *under submission*, 2017.

[11] Manos Athanassoulis and Stratos Idreos. Design Tradeoffs of Data Access Methods. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Tutorial*, 2016.

[12] Manos Athanassoulis, Michael S. Kester, Lukas M. Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki, and Mark Callaghan. Designing Access Methods: The RUM Conjecture. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 461–466, 2016.

[13] Manos Athanassoulis, Zheng Yan, and Stratos Idreos. UpBit: Scalable In-Memory Updatable Bitmap Indexing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2016. Most Reproducible Paper

[14] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. Monkey: Optimal Navigable Key-Value Store. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79–94, 2017. "Best of SIGMOD 2017"

[15] Ryan Johnson, Manos Athanassoulis, Radu Stoica, and Anastasia Ailamaki. A New Look at the Roles of Spinning and Blocking. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, page 21, 2009.

[16] Ryan Johnson, Ippokratis Pandis, Radu Stoica, Manos Athanassoulis, and Anastasia Ailamaki. Aether: A Scalable Approach to Logging. *Proceedings of the VLDB Endowment*, 3(1-2):681–692, 2010. "Best of VLDB 2010"

[17] Ryan Johnson, Ippokratis Pandis, Radu Stoica, Manos Athanassoulis, and Anastasia Ailamaki. Scalability of write-ahead logging on multicore and multisocket hardware. *The VLDB Journal*, 21(2):239–263, 2011.

[18] Michael S. Kester, Manos Athanassoulis, and Stratos Idreos. Access Path Selection in Main-Memory Optimized Data Systems: Should I Scan or Should I Probe? In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 715–730, 2017.

[19] Matthaios Olma, Manos Karpathiotakis, Ioannis Alagiannis, Manos Athanassoulis, and Anastasia Ailamaki. Slalom: Coasting Through Raw Data via Adaptive Partitioning and Indexing. *Proceedings of the VLDB Endowment*, 10(10):1106–1117, 2017.

[20] Iraklis Psaroudakis, Manos Athanassoulis, and Anastasia Ailamaki. Sharing Data and Work Across Concurrent Analytical Queries. *Proceedings of the VLDB Endowment*, 6(9):637–648, 2013.

[21] Iraklis Psaroudakis, Manos Athanassoulis, Matthaios Olma, and Anastasia Ailamaki. Reactive and Proactive Sharing Across Concurrent Analytical Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 889–892, 2014.

[22] Radu Stoica, Manos Athanassoulis, Ryan Johnson, and Anastasia Ailamaki. Evaluating and Repairing Write Performance on Flash Devices. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, pages 9–14, 2009.

[23] Sam Likun Xi, Oreoluwa Babarinsa, Manos Athanassoulis, and Stratos Idreos. Beyond the Wall: Near-Data Processing for Databases. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, page 2, 2015.