Data Structures for Data-Intensive Applications: Tradeoffs and Design Guidelines
Other titles in Foundations and Trends® in Databases

Consensus in Data Management: From Distributed Commit to Blockchain
Faisal Nawab and Mohammad Sadoghi

Multidimensional Array Data Management
Florin Rusu

Modern Datalog Engines
Bas Ketsman and Paraschos Koutris

Natural Language Interfaces to Data
Abdul Quamar, Vasilis Efthymiou, Chuan Lei and Fatma Özcan
ISBN: 978-1-63828-028-6

Trends in Explanations: Understanding and Debugging Data-driven Systems
Boris Glavic, Alexandra Meliou and Sudeepa Roy
ISBN: 978-1-68083-880-0

Differential Privacy for Databases
Joseph P. Near and Xi He
Data Structures for Data-Intensive Applications: Tradeoffs and Design Guidelines

Manos Athanassoulis
Boston University
mathan@bu.edu

Stratos Idreos
Harvard University
stratos@seas.harvard.edu

Dennis Shasha
New York University
shasha@courant.nyu.edu
Foundations and Trends® in Databases
Volume 13, Issue 1-2, 2023
Editorial Board

Editor-in-Chief
Joseph M. Hellerstein
University of California at Berkeley
United States

Surajit Chaudhuri
Microsoft Research, Redmond
United States

Editors
Azza Abouzied
NYU-Abu Dhabi

Gustavo Alonso
ETH Zurich

Mike Cafarella
University of Michigan

Alan Fekete
University of Sydney

Ihab Ilyas
University of Waterloo

Sunita Sarawagi
IIT Bombay
Editorial Scope

Topics

Foundations and Trends® in Databases publishes survey and tutorial articles in the following topics:

- Data Models and Query Languages
- Query Processing and Optimization
- Storage, Access Methods, and Indexing
- Transaction Management, Concurrency Control and Recovery
- Deductive Databases
- Parallel and Distributed Database Systems
- Database Design and Tuning
- Metadata Management
- Object Management
- Trigger Processing and Active Databases
- Data Mining and OLAP
- Approximate and Interactive Query Processing
- Data Warehousing
- Adaptive Query Processing
- Data Stream Management
- Search and Query Integration
- XML and Semi-Structured Data
- Web Services and Middleware
- Data Integration and Exchange
- Private and Secure Data Management
- Peer-to-Peer, Sensornet and Mobile Data Management
- Scientific and Spatial Data Management
- Data Brokering and Publish/Subscribe
- Data Cleaning and Information Extraction
- Probabilistic Data Management

Information for Librarians

Foundations and Trends® in Databases, 2023, Volume 13, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.
## Contents

1 Introduction ........................................................................ 2  
   1.1 Data Structures Are Foundational .............................. 2  
   1.2 Tradeoffs in Data Structure Design ............................ 4  
   1.3 Audience & Prerequisites ......................................... 7  
   1.4 Learning Outcomes .................................................. 8  
   1.5 Overview of the Book .............................................. 8  

2 Performance Metrics and Operational Tradeoffs ............... 9  
   2.1 Memory Hierarchy .................................................... 9  
   2.2 From Read/Update to RUM: Memory & Space Costs ... 10  
   2.3 RUM Performance Costs .......................................... 11  
   2.4 From RUM to PyRUMID ........................................... 14  
   2.5 Chapter Summary .................................................... 17  
   2.6 Questions .............................................................. 18  
   2.7 Further Readings ..................................................... 19  

3 Dimensions of the Data Structure Design Space ............... 21  
   3.1 Global Data Organization ......................................... 24  
   3.2 Global Search Algorithms When Not Using an Index ... 26  
   3.3 Search When Using an Index ..................................... 32  
   3.4 Local Data Organization .......................................... 46  
   3.5 Local Search .......................................................... 48
4 From Workloads to Data Structures

4.1 Point and Range Queries, Modifications, but Rare Scans
(Traditional B+-trees and Learned Tree Indexes) ............... 68
4.2 Similar Workload With a Working Set That Fits in Memory
(Fractal B+-trees) .................................................. 70
4.3 Point and Range Queries, Rare Scans, More Modifications
(Insert-Optimized Search Trees) .............................. 70
4.4 Mixed Workload With No Short Range Queries
(Hybrid Range Trees) .............................................. 75
4.5 Mixed Workload, With Ever Increasing Data Size
(Radix Trees) ....................................................... 78
4.6 Point Queries, Inserts, and Some Modifications
(Static Hashing with Overflow Pages) ....................... 79
4.7 Read-mostly With Long Range Queries
(Scans with Zonemaps) ............................................ 81
4.8 Modification-intensive With Point and Range Queries
(LSM-tree) .......................................................... 82
4.9 Modification-intensive With Point Queries Only
(LSM-hash) .......................................................... 84
4.10 When to Design Heterogeneous Data Structures .......... 85
4.11 Data Structures in Practice .................................. 86
4.12 Chapter Summary ............................................. 88
4.13 Questions ....................................................... 89
4.14 Further Readings ............................................. 92
ABSTRACT

Key-value data structures constitute the core of any data-driven system. They provide the means to store, search, and modify data residing at various levels of the storage and memory hierarchy, from durable storage (spinning disks, solid state disks, and other non-volatile memories) to random access memory, caches, and registers. Designing efficient data structures for given workloads has long been a focus of research and practice in both academia and industry.

This book outlines the underlying design dimensions of data structures and shows how they can be combined to support (or fail to support) various workloads. The book further shows how these design dimensions can lead to an understanding of the behavior of individual state-of-the-art data structures and their hybrids. Finally, this systematization of the design space and the accompanying guidelines will enable you to select the most fitting data structure or even to invent an entirely new data structure for a given workload.
1

Introduction

1.1 Data Structures Are Foundational

Data structures are the means by which software programs store and retrieve data. This book focuses on key-value data structures, which are widely used for data-intensive applications thanks to the versatility of the key-value data model. Key-value data structures manage a collection of key-value entries, with the property that a given key maps to only one value but the same value can be associated with many keys. The value part of a data entry may have arbitrary semantics. For example, it may be a record in a relational database or a Pandas DataFrame, or an arbitrary set of fields that the application knows how to parse and use in a NoSQL system. In some settings, such as when systems manage data for social networks, the value may contain a reference to a large object such as an image or video.

Physically, a key-value data structure consists of (1) the data, physically stored in some layout, (2) optional metadata to facilitate navigation over the data, and (3) algorithms to support storage and retrieval operations (Hellerstein et al., 2007; Selinger et al., 1979; Idreos et al., 2018a). Other terms used in the literature for data structures include “access methods,” “data containers,” and “search structures.”
Data systems, operating systems, file systems, compilers, and network systems employ a diverse set of data structures. This book draws examples primarily from the area of large-volume data systems which require secondary storage devices, but the core analysis and design dimensions apply to purely in-memory systems as well, where access to RAM (random access memory) is far slower than access to the cache. In fact, the analysis applies to any setting in which there are two or more levels in the memory/storage hierarchy.

Given the wealth of applications that can be modeled using key-value data, such data structures have enormous general utility. For example, a particular data structure can be used to describe (i) metadata access in files, networks, and operating systems (Bovet and Cesati, 2005; Rodeh, 2008), (ii) data access in relational systems (Hellerstein et al., 2007), (iii) data access in NoSQL and NewSQL systems (Idreos and Callaghan, 2020; Mohan, 2014), and (iv) feature engineering and model structures in machine learning pipelines (Wasay et al., 2021).

Each application, or workload, can be represented as a mixture of key-value operations (point queries, range queries, inserts, deletes, and modifications) it supports over its data. In addition, the amount of memory and persistent storage required, along with their cost, shape the requirements of a given application. For example, file systems manage file metadata and contents using data structures optimized for frequent updates. Compilers typically use hash maps to manage variables during the variables’ lifespan and use abstract syntax trees to capture the overall shape of a program. Similarly, network devices require specialized data structures to efficiently store and access routing tables.

As data-intensive applications emerge and evolve over time, using efficient data structures becomes critical to the viability of such applications, sometimes resulting in a three orders of magnitude performance change, as shown by Chatterjee et al. (2022). The reason is that data movement is the major bottleneck in data-intensive applications. Data movement is largely governed by the way data is stored, i.e., by the data structure. Thus, we expect that there will be an ongoing need for new data structures as new applications appear, hardware changes and data grows. Currently, research in academia and industry produces several new data structure designs every year, and this pace is expected
to grow. At the same time, with a growing set of new data structures available, even the task of choosing from an off-the-shelf data structure, that is, one that can be found in textbooks, has become more complex.

This book aims to explain the space of data structure design choices, how to select the appropriate data structure depending on the goals and workload of an application at hand, and how the ever-evolving hardware and data properties require innovations in data structure design. The overarching goal is to help the reader both select the best existing data structures and design and build new ones.

1.2 Tradeoffs in Data Structure Design

Every data structure represents a particular workload- and hardware-dependent performance tradeoff (formalized by Athanassoulis et al., 2016 and Idreos et al., 2018b). In order to choose an existing data structure or to design a new data structure for a particular workload on particular hardware, you should understand the possible design space of data structure design clearly and formally. That is the focus of this book. To motivate that discussion, let us look at a few examples of designs and tradeoffs when considering the workload (Section 1.2.1) as well as the underlying hardware (Section 1.2.2) and how they both evolve over time.

1.2.1 Workload-Driven Designs

Optimizing for a Workload. Consider a workload that consists of a small number of inserts and updates together with a large number of point and range queries. In order to balance the read and the write cost, many applications employ a B+-tree, originally proposed by Bayer and McCreight (1972) and later surveyed by Graefe (2011). B+-trees have a high node fanout, so that traversing from root to leaf requires few secondary memory accesses, and their top levels are cached in the faster levels of the memory hierarchy (Section 4.1). Further, a B+-tree supports range queries by maintaining all the keys sorted in the leaf nodes and by connecting the leaf nodes in a linked list. As the number of insertions and updates increase, however, leaf nodes must be reorganized and maybe even split, which can become a performance bottleneck.
1.2. Tradeoffs in Data Structure Design

To address workloads having many inserts, a completely different approach is taken by a data structure called the log-structured merge-tree (LSM-tree). LSM-trees were originally introduced by O’Neil et al. (1996), and their many variants were surveyed by Luo and Carey (2020). As we will see in Section 4.8, LSM-trees place all updates in a common memory buffer which is flushed to disk when it becomes full. As more buffers accumulate, they are merged to form larger sorted data collection. This design employs an *out-of-place* policy of handling modifications, surveyed in detail by Sarkar and Athanassoulis (2022), in which there can be many key-value pairs in the structure having the same key. (For a given key $k$, the most recently inserted key-value pair for $k$ has the current value.)

Thus two different workloads – one with more read queries and one with more insert operations – suggest different data structures.

![Figure 1.1: Adaptive data organization using the most recent search as a hint. In this example, a range query for values 15 to 60 leads to partitioning the base data in three non-overlapping partitions, one with values less than 15, one with values between 15 and 60, and one with values greater than 60.](image)

**Adapting to a Workload.** Because the central theme of this book is to design a data structure given the expected workload, we also consider designing data structures that gradually adapt to the ideal design. To illustrate this point, consider that the $B^+$-tree and the LSM-tree, as originally designed, impose a sorted order within disk-resident nodes in order to answer any point or range query. An adaptive data structure may start with one or more unsorted nodes, but sorts them gradually in an opportunistic way, as shown in Figure 1.1. Chapter 5 will explain the concept of database cracking, proposed by Idreos et al. (2007a)
and further expanded by Idreos et al. (2007b) and Idreos et al. (2009). Intuitively, cracking uses the access patterns of incoming queries to continuously and incrementally physically reorganize the underlying data with the goal of improving the performance of future queries.

1.2.2 Memory-Driven And Storage-Driven Designs

As a complement to workload-based considerations, hardware advances create new challenges, needs, and opportunities in data structure design. Over the years, the memory and storage hierarchy has been enriched with devices such as solid-state disks, non-volatile memories, and deep cache hierarchies. Here, we discuss a few key hardware considerations for data structures, and we expand on them in Chapters 6 and 7.

**Optimizing for the Storage/Memory Hierarchy.** In a storage hierarchy, the lower levels offer a lot of storage at a low price but at high access latency, and as we move higher, that is, closer to the processor(s), the storage is faster but smaller and more expensive per byte. In the storage/memory hierarchy there is always a level that is the bottleneck for a given application, which depends on the size of the application data relative to the storage capacity available at the different levels of the hierarchy.

Originally, B⁺-trees tried to minimize disk accesses by maximizing the fanout. As the memory sizes grew, however, much of the data could fit into random access memory or non-volatile secondary memory. This changed the tradeoffs dramatically. For example, in-memory B⁺-trees perform best with small fanout, as shown by Kester et al. (2017).

**Memory Wall.** While the memory hierarchy is expanding with technologies like high-bandwidth memory as outlined by Pohl et al. (2020), a key hardware trend for several decades has been the growing disparity between processor speed and the speed of off-chip memory, termed the memory wall by Wulf and McKee (1995). Since the early 2000s, operating systems, as discussed by Milojicic and Roscoe (2016), and data management systems, as discussed by Johnson et al. (2009), have been carefully re-designed to account for the memory wall by optimizing the use of cache memories.
Storage Devices Evolve. In addition, secondary storage itself has reached a crossover point. Traditional hard disks have long since hit their physical speed limits (Athanassoulis, 2014), and have largely been replaced by shingled disks and flash-based devices (Hughes, 2013). Shingled disks increase the density of storage on the magnetic medium, changing the performance properties of disks because the granularity of reads and writes changes (Hughes, 2013). Flash-based drives offer significantly faster read performance than traditional disks, but suffer from relatively poor write performance. Further, flash-based drives are equipped with a highly functional firmware layer called the Flash Translation Layer (FTL), which can be rewritten to yield dramatic changes in performance. Thus, flash hardware performance depends on both hardware and firmware. Such changes create a need for new data structures to optimize for different hardware/firmware combinations.

1.3 Audience & Prerequisites

This book aims to be used as part of graduate-level classes on designing complex data structures. It assumes at least an undergraduate-level familiarity with data structures.

Specifically, we assume that the reader is already familiar with basic data structures like arrays, linked lists, binary trees, queues, stacks, heaps, and hash tables, all taught in introductory courses for data structures and algorithms (Cormen et al., 2009). Such basic data structures sit at the core of the more complex designs we outline. Most of the use cases and the presented designs are motivated by data-intensive systems, hence building on classical data structures used in database systems like B+-trees (Bayer and McCreight, 1972), open hashing (Ramakrishnan and Gehrke, 2002), and LSM-trees (O’Neil et al., 1996).

Other than that, the book is self-contained. We will describe all algorithms used in navigating complex data structures and outline how to combine the various design decisions we introduce.
1.4 Learning Outcomes

After reading this book, you will be able to reason about which existing data structure will perform best given a workload and the underlying hardware. In addition, you will be able to design new and possibly hybrid data structures to handle workloads with different composition, locality, and access patterns.

1.5 Overview of the Book

Here is the outline of our book. We recommend that you read the chapters sequentially.

- Chapter 2 introduces the fundamental performance metrics for data structures with respect to the most important key-value operations and hardware properties.

- Chapter 3 presents the core set of design principles, primarily based on workload characteristics, that largely guide the design of key-value data structures.

- Chapter 4 starts by explaining the design and performance characteristics of traditional data structures based on the design principles. We then discuss how to use the design principles to design new data structures for arbitrary workloads.

- Chapter 5 discusses the need for and design principles underlying adaptive data structures. We also illustrate use cases in which adaptivity leads to greatly improved performance.

- Chapter 6 discusses how data structures are utilized in big data applications, including databases, file systems, and machine learning.

- Chapter 7 discusses additional design considerations that can influence the detailed deployment of data structures ranging from deploying data structures in a setting with concurrent execution, in the context of distributed systems, to new hardware, new type workloads, and new application requirements.
References


References


References


Full text available at: http://dx.doi.org/10.1561/1900000059


References


References


tedb.pdf.


dreos.pdf.


References


References


References


References


Full text available at: http://dx.doi.org/10.1561/1900000059


References


References


