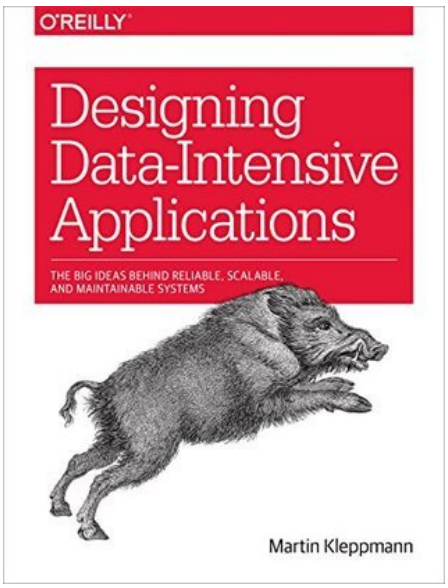

Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems

Martin Kleppmann

Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems

Martin Kleppmann

Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems Martin Kleppmann
Want to know how the best software engineers and architects structure their applications to make them scalable, reliable, and maintainable in the long term? This book examines the key principles, algorithms, and trade-offs of data systems, using the internals of various popular software packages and frameworks as examples.

Tools at your disposal are evolving and demands on applications are increasing, but the principles behind them remain the same. You'll learn how to determine what kind of tool is appropriate for which purpose, and how certain tools can be combined to form the foundation of a good application architecture. You'll learn how to develop an intuition for what your systems are doing, so that you're better able to track down any problems that arise.

Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems Details

Date	:	Published April 2nd 2017 by O'Reilly Media (first published April 25th 2015)
ISBN	:	9781449373320
Author	:	Martin Kleppmann
Format	:	Paperback 624 pages
Genre	:	Computer Science, Programming, Science, Technology, Technical, Software

From Reader Review Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems for online ebook

Irio says

I recently used Spark to count all the data stores mentioned throughout the book.

There's a total of 72 products, where Apache ZooKeeper, PostgreSQL and MySQL are the ones most mentioned, with 46, 44 and 42 citations.

The complete list is available at https://medium.com/@irio/all-data-sto...

Sebastian Gebski says

Honestly, this one took me much more time than I've expected.
Plus, it's definitely one of the best technical books I've read in years - but still, it doesn't mean you should run straight away to your bookshop - read up to the end of the review first.

I'll risk the statement that this book's content will not be 100% directly applicable to your work, BUT it will make you a better engineer in general. It's like with reading books about Haskell - most likely you'll never use this language for any practical project/product development, but understanding Haskell (& principles behind its design) will improve your functional-fu.

In this case, Martin (true expert, one of people who stood behind Kafka in LinkedIn - if I remember correctly), doesn't try to rediscover EAI patterns or feed you with CAP basics -> instead he dives deep into low level technical differences between practical implementations of message brokers, relational & non-relational databases. He discusses various aspects of distribution, but he doesn't do stop at theory. This book is all about practical differences, based on actual implementations in popular technologies.

No, 95% of us will not write stuff I tend to call "truly infrastructural". No, 95% of us will never get down to implementation of tombstones or dynamic shard re-balancing in Cassandra. But still, even reading about how those practical problems were solved will make us better engineers & will add more options (/ideas) to our palette. For some youngers - it will prove them that there's no mysterious magic behind technology they use - it's just good, solid, pragmatic engineering after all.

Great book. Truly recommended. Try it yourself & enjoy how it tastes :) I would give 6 freaking stars if I could.

Kaushal says

I wish I had this book 5 years go. A complete text on distributed systems that are extremely valuable for

hands on experience. You have to read this book multiple times to get a good grasp on concepts on distributed computing. I do feel the title is little misleading for a solid texts on distributed systems. Highly recommended.

Yevgeniy Brikman says

A must-read for every programmer. This is the best overview of data storage and distributed systems—two key concepts for building almost any piece of software today—that I've seen anywhere. Martin does a wonderful job of taking a massive body of research and distilling complicated concepts and difficult trade-offs down to a level where anyone can understand it.

I learned a lot about replication, partitioning, linearizability, locking, write skew, phantoms, transactions, event logs, and more. I'm also a big fan of the final chapter, The Future of Data Systems, which covers ideas such as "unbundling the database" (i.e., using an event log as the primary data store, and handling all other aspects of the "database", such as secondary indexes, materialized views, and replication, in separate "derived" data systems), end-to-end event streams, and an important discussion on ethics in programming and data systems.

The only thing missing is a set of summary tables. I'd love to see a list of all common data systems and how they fair across many dimensions: e.g., support for locking, replication, transaction, consistency levels, and so on. This would be very handy for deciding what system to pick for my next project.

As always, I've saved a few of my favorite quotes from the book:

"Document databases are sometimes called schemaless, but that’s misleading, as the code that reads the data usually assumes some kind of structure—i.e., there is an implicit schema, but it is not enforced by the database. A more accurate term is schema-on-read (the structure of the data is implicit, and only interpreted when the data is read), in contrast with schema-on-write (the traditional approach of relational databases, where the schema is explicit and the database ensures all written data conforms to it). Schema-on-read is similar to dynamic (runtime) type checking in programming languages, whereas schema-on-write is similar to static (compile-time) type checking."

"For defining concurrency, exact time doesn’t matter: we simply call two operations concurrent if they are both unaware of each other, regardless of the physical time at which they occurred. People sometimes make a connection between this principle and the special theory of relativity in physics, which introduced the idea that information cannot travel faster than the speed of light. Consequently, two events that occur some distance apart cannot possibly affect each other if the time between the events is shorter than the time it takes light to travel the distance between them."

"A node in the network cannot know anything for sure—it can only make guesses based on the messages it receives (or doesn’t receive) via the network."

"The best way of building fault-tolerant systems is to find some general-purpose abstractions with useful guarantees, implement them once, and then let applications rely on those guarantees."

"CAP is sometimes presented as Consistency, Availability, Partition tolerance: pick 2 out of 3. Unfortunately, putting it this way is misleading because network partitions are a kind of fault, so they aren't something about which you have a choice: they will happen whether you like it or not. At times when the network is working correctly, a system can provide both consistency (linearizability) and total availability. When a network fault occurs, you have to choose between either linearizability or total availability. Thus, a better way of phrasing CAP would be either Consistent or Available when Partitioned."

"The traditional approach to database and schema design is based on the fallacy that data must be written in the same form as it will be queried. Debates about normalization and denormalization (see"Many-to-One and Many-to-Many Relationships") become largely irrelevant if you can translate data from a write-optimized event log to read-optimized application state: it is entirely reasonable to denormalize data in the read-optimized views, as the translation process gives you a mechanism for keeping it consistent with the event log."

"As algorithmic decision-making becomes more widespread, someone who has (accurately or falsely) been labeled as risky by some algorithm may suffer a large number of those "no" decisions. Systematically being excluded from jobs, air travel, insurance coverage, property rental, financial services, and other key aspects of society is such a large constraint of the individual's freedom that it has been called "algorithmic prison". In countries that respect human rights, the criminal justice system presumes innocence until proven guilty; on the other hand, automated systems can systematically and arbitrarily exclude a person from participating in society without any proof of guilt, and with little chance of appeal."

"Predictive analytics systems merely extrapolate from the past; if the past is discriminatory, they codify that discrimination. If we want the future to be better than the past, moral imagination is required, and that's something only humans can provide."

Bill says

Some quite valuable content diluted with less useful content. I think I'd much prefer to read this author's focused articles or blogs than recommend that someone slog through this.

I'm still not quite sure who the intended audience of this book is, but it's definitely not me. The intro chapter discusses the example of Twitter's fan-out writes and how they balanced typical users with celebrities who have millions of followers. Because of that intro, I expected a series of architecture patterns and case studies from running systems at scale. What follows was nothing like that.

The book suffers greatly from being overly academic and abstract. It tries to achieve both breadth and depth. Much of the tone felt like an encyclopedia of data-related technologies. The author namedrops dozens of technologies, many of them outside mainstream use. The low-level database chapter was insufferably detailed. I would've stopped reading there but it was a tech book club pick at work and I felt compelled to finish it.

Does this author know a lot about data tech? Definitely. Did I learn things from this book? Sure. Do I have a stronger understanding of issues related to data? I think so. Will I approach data problems differently in the future? Possibly. Did I come away with strong endorsement of some application architecture(s) to consider when building my next system? Not really.

I feel like most of that must be available in better forms elsewhere. If not, someone please write some more books.

Juan Ignacio says

My full notes: https://juanignaciosl.github.io/books...

IMHO this book is a modern classic, a must read for every software engineer and developer. I’m certain that it will be reread it from time.

Ahmad hosseini says

This book changed my view to designing application!
What is the meaning of Data-Intensive?
We call an application data-intensive if data is its primary challenge- the quality of data, the complexity of data, or the speed at which it is changing.

Who should read this book?
I think that all developers must read this book. If you develop applications that have some kind of server/ backend for storing or processing data, and your application use the internet, then this book is for you.

Why should you, as an application developer, care how the database handles storage and retrieved internally?
You do need to select a storage engine that is appropriate for your application, from the many that are available. In order to tune a storage to perform well on your kind of workload you need to have a rough idea of what the storage engine is doing under the hood.

What is scope of this book?
This book compares several different data models and query languages and turns to the internals of storage engine such as Cassandra, Redis, MongoDB, and etc. and looks at how database lay out data on disk. It also pays to distributed data and distributed system and their challenges like consistency, scalability, fault tolerance, and complexity.

Stepan Kuzmin says

???????, ?????? ?????, ?? ??????? ??????? ???????? (?????????? ?????????? ? ????????? ? ????? ??? MapReduce). ?????????? ?????????????? ????? ??? ??????? ?????????????? ?????? — ??????????? ????????? «?????????????? ??????». ? ????? — ???????? ?????.

Ye Lin Kyaw says

There should be a 6-star rating for this book.

Emanuele Blanco says

A clear and detailed overview of the challenges modern applications have to face while dealing with data and the current state-of-the-art. From SSTables to event sourcing, Martin Kleppman gives great insights on what every engineer/architect should know when designing systems that deal with any kind of data. Highly recommended.

David Bjelland says

Like you'd expect of a technical book with such a broad scope, there are sections that most readers in the target audience will probably find either too foundational or too esoteric to justify writing about at this kind of length, but still - at its best, I shudder to think of the time wasted groping in the dark for an ad hoc understanding of concepts it explains holistically in just a few unfussy, lucid pages and a diagram or two.

Definitely a book I see myself reaching for as a reference or memory jogger for years to come.

Bodo Tasche says

This book is an amazing must have for every backend developer. Highly recommended.

Oleksii Zuiev says

The book gives comprehensive overview of design aspects for systems working with data. For each of them it goes deep enough to describe needed concepts and principles and implementation options. And if you want to go deeper, after each chapter there are big lists of references to relevant research papers, specific implementations etc. The book ends with a chapter where the author gives his subjective view on where the industry is moving. Which is distinct to the rest of the book but still is an interesting read.

I enjoyed the way the book is written. It is well structured, detailed enough to explain most complex parts, yet not verbose, diagrams are provided where needed. I appreciated objective writing style and references for most of the claims.

I can recommend this book to engineers working on systems working with data. It does not give direct advice or guidance on how to design systems but succeeds in creating good mental map to navigate this complex area.

Brian says

(5.0) excellent summary/foundation/recommendations for distributed systems development, covers a lot of the use cases for data-intensive (vs compute-intensive) apps/services. I recommend to anyone doing service development.

Recommendations are well-reasoned, citations are helpful and are leading me to do a lot more reading.

Thank you for finding and sharing this one, @Chet. I think this will be a book we assign as a primer for working at Goodreads going forward. At least some of the (later) chapters.

Emre Sevinç says

I consider this book a mini-encyclopedia of modern data engineering. Like a specialized encyclopedia, it covers a broad field in considerable detail. But it is not a practice or a cookbook for a particular Big Data, NoSQL or newSQL product. What the author does is to lay down the principles of current distributed big data systems, and he does a very fine job of it.

If you are after the obscure details of a particular product, or some tutorials and "how-to"s, go elsewhere. But if you want to understand the main principles, issues, as well as the challenges of data intensive and distributed system, you've come to the right place.

Martin Kleppmann starts out by solidly giving the reader the conceptual framework in the first chapter: what does reliability mean? How is it defined? What is the difference between "fault" and "failure"? How do you describe load on a data intensive system? How do you talk about performance and scalability in a meaningful way? What does it mean to have a "maintainable" system?

Second chapter gives a brief overview of different data models and shows the suitability of them to different use cases, using modern challenges that companies such as Twitter faced. This chapter is a solid foundation for understanding the difference between the relational data model, document data model, graph data model, as well as the languages used for processing data stored using these models.

The third chapter goes into a lot of detail regarding the building blocks of different types of database systems: the data structures and algorithms used for the different systems shown in the previous chapter are described; you get to know hash indexes, SSTables (Sorted String Tables), Log-Structured Merge trees (LSM-trees), B-trees, and other data structures. Following this chapter, you are introduced to Column Databases, and the underlying principles and structures behind them.

Following these, the book describes the methods of data encoding, starting from the venerable XML & JSON, and going into the details of formats such as Avro, Thrift and Protocol Buffers, showing the trade-offs between these choices.

Following the building blocks and foundations comes "Part II", and this is where things start to get really interesting because now the reader starts to learn about challenging topic of distributed systems: how to use

the basic building blocks in a setting where anything can go wrong in the most unexpected ways. Part II is the most complex of part the book: you learn about how to replicate your data, what happens when replication lags behind, how you provide a consistent picture to the end-user or the end-programmer, what algorithms are used for leader election in consensus systems, and how leaderless replication works.

One of the primary purpose of using a distributed system is to have an advantage over a single, central system, and that advantage is to provide better service, meaning a more resilient service with an acceptable level of responsiveness. This means you need to distribute the load and your data, and there a lot of schemes for partitioning your data. Chapter 6 of Part II provides a lot of details on partitioning, keys, indexes, secondary indexes and how to handle data queries when your data is partitioned using various methods.

No data systems book can be complete without touching the topic of transactions, and this book is not an exception to the rule. You learn about the fuzziness surrounding the definition of ACID, isolation levels, and serializability.

The remaining two chapters of Part II, Chapter 8 and 9 is probably the most interesting part of the book. You are now ready to learn the gory details of how to deal with all kinds of network and other types of faults to keep your data system in usable and consistent state, the problems with the CAP theorem, version vectors and that they are not vector clocks, Byzantine faults, how to have a sense of causality and ordering in a distributed system, why algorithms such as Paxos, Raft, and ZAB (used in ZooKeeper) exist, distributed transactions, and many more topics.

The rest of the book, that is Part III, is dedicated to batch and stream processing. The author describes the famous Map Reduce batch processing model in detail, and briefly touches upon the modern frameworks for processing distributed data processing such as Apache Spark. The final chapter discusses event streams and messaging systems and challenges that arise when trying to process this "data in motion". You might not be in the business of building the next generation streaming system, but you'll definitely need to have a handle on these topics because you'll encounter the described issues in the practical stream processing systems that you deal with daily as a data engineer.

As I said in the opening of this review, consider this a mini-encyclopedia for the modern data engineer, and also don't be surprised if you see more than 100 references at the end of some chapters; if the author tried to include most of them in the text itself, the book would well go beyond 2000 pages!

At the time of my writing, the book is 90% complete, according to its official site there's only 1 more chapter to be added (Chapter 12: Materialized Views and Caching), so it is safe to say that I recommend this book to anyone working with distributed big data systems, dealing with NoSQL and newSQL databases, document stores, column oriented data stores, streaming and messaging systems. As for me, it'll definitely be my go-to reference for the upcoming years for these topics.
