Deep Science Publishing
https://doi.org/10.70593/978-93-49910-08-9_8

DeepScience
Open Access Books

Chapter 8: Applying data engineering principles to build distributed, scalable, and fault-tolerant data systems

8.1. Introduction

Over the past two decades, we have seen the rapid adoption of distributed data management systems in industry, with many early adopters leading the way. Preferences are shifting from smaller, local, centralized, monolithic systems towards larger, distributed, concurrent, global systems that are scalable, fault-tolerant, and can provide diverse functionalities over a wider range of data types. Organizations are building next-generation data services using many innovations in distributed systems and information management technology: peer-to-peer and web services architectures, industrial-strength clustering and fault-tolerance technologies, large scale reliable storage systems, and efficient indexing and retrieval methods for unstructured data (Armbrust et al., 2010; Bass et al., 2012; Gollapudi, 2021).

At the same time, research efforts in data systems are focusing increasingly on the development of distributed, scalable, and fault-tolerant techniques that can support services such as web-search, information-sensors, click-stream analysis, peer-to-peer storage and publish/subscribe services. There have been interesting ideas, especially in the areas of scalable data access and retrieval services, reliable storage, and high-performance data dissemination services. Today, large amounts of data are being generated and collected by organizations. Simultaneously, businesses are realizing that enormous improvement in profitability can be achieved by employing new tools and approaches for data analysis: mining for knowledge; learning predictive models; performing trend analysis over historical data; performing on-line, real-time analysis and filtering of current data.

Many of these organizations are beginning to analyze transaction data from their business processes in conjunction with traditional data-analysis techniques. By

employing technology to analyze and filter the data, businesses can make intelligent decisions about managing customer relationships. With increasingly complete data repositories, businesses are increasingly looking to drive customer relationships by collecting explicit data from customers and analyzing their behavior. A strong tool-set for large scale data analysis will allow organizations to automatically enhance their intelligence infrastructure. As these organizations have entered into the hype cycle for such analysis tools, they expect an understanding ecosystem (Krishnan, 2013; Kleppmann, 2017).

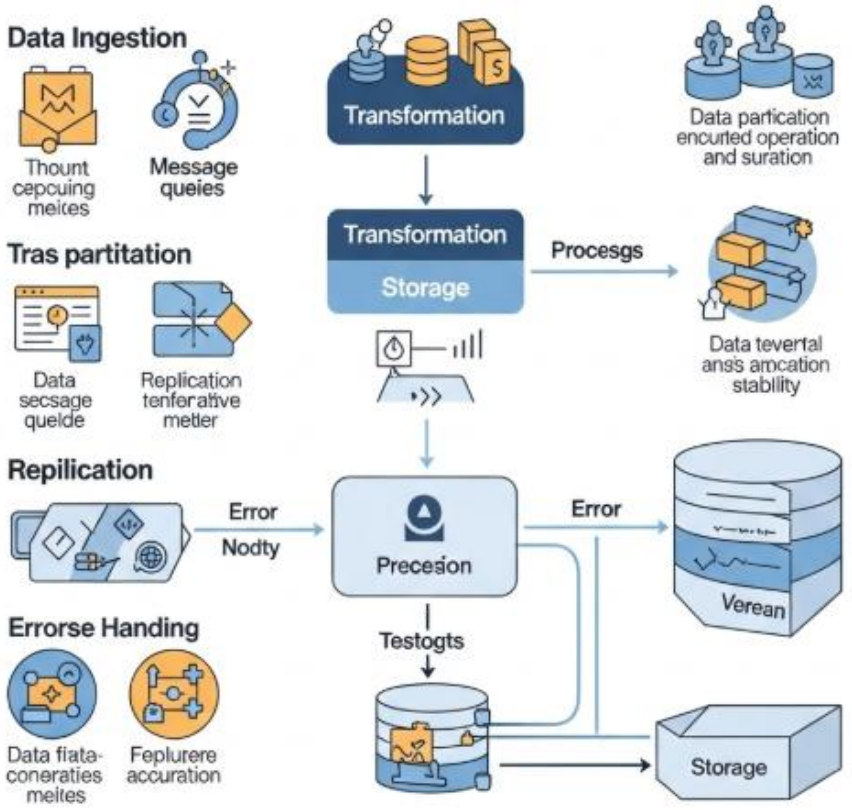

Fig 8.1: Applying Data Engineering Principles to Build Distributed, Scalable, and Fault-Tolerant Data Systems

8.1.1. Background and Significance

Real-world applications, especially in areas such as e-commerce and finance, need to process a colossal quantity of transactional data. The existing redistributive data processing architectures achieve scalability by distributing workloads over a large cluster and using a disk-based strategy to maintain fault-tolerance. Traditional disk-centered database management software has proved to be effective for transactional workloads, thus forming a foundation technology for enterprise data processing and

management. However, the rapid growth of available system memory has shifted this perspective, as more applications may now run completely in RAM.

Such memory-centered solutions achieve very high throughput and low latency, and are easy to manage and use. However, when considering the need for high availability and support for large data stores, the issues of distributing, scaling up to clusters, and tolerating node failures become complicated. Current solutions achieve one aspect while failing at others: redistributive systems provide scalability and fault-tolerance while sacrificing performance; hyper-table and similar systems provide single-node transactional performance while failing on support for very large data sets; other systems, such as Bigtable, scaling support at the expense of transactions; use of shared memory and DRAM buffers provides performance but fail on scale for very large data sets.

8.2. Fundamentals of Data Engineering

Data engineering applies to that aspect of the process, eventually realized in code, that deals with the physical manipulation of data. This manipulation can be considered to exist on a hierarchy of levels, from low-level programmer assignment of numerical values to computer memory to high-level configuration of distributed databases. The levels of data engineering most relevant to this book are proper to historical data management and can be subsumed under two somewhat overlapping constructs. The first and more mundane is data management – that aspect of computer use that affects the physical storage of data and its retrieval for viewing and updating; that is, the data store and the associated software routines to be invoked when necessary. The second is data management design – that aspect of the design of a computer system that specifically deals with storage, retrieval, and updating of data through the establishment of data models, domain dictionaries, database management system specifications, file organization techniques, and so on.

Why even at this late date has so much data pedantry remained in the world? There are really two answers, one of which has always been there early in near the end of the Data Processing Era. During that time, which lasted at least 25 years, computers were regarded as boxes and what went on inside them was regarded as a black box using a language of integers, floating-point, and magnetic core. Surfacing early in the Data Processing Era, and still enduring in the Information Systems Era, was the data models-like record and file organization techniques, and the associated never-ending disputes over how many data elements to have in an address record and what should happen to them when one gets a message from the marketing department concerning a change in the product line.

8.2.1. Definition and Scope

Data engineering is the discipline and field of study concerned with the systems, processes, and structures that enable acquisition, storage, and analysis of data at scale. Data engineering encompasses data integration and preparation, storage, architecture, orchestration, ETL and ELT, and interoperability. It also addresses governance and security issues around data. Data engineering does not discuss statistical, mathematical, or model building issues with data. Nor does it discuss application-level use or exploitation of data. For example, program-level techniques for biological sequence searching, image classification, or natural language understanding are not within the scope of data engineering. Facilitating business intelligence and operational analytics applications is a primary goal of many data engineering systems. Data engineering also plays a key role in enabling other forms of data analysis like scientific discovery and machine learning-based prediction. It underpins applications in business, government, and science in the areas of health, safety, transportation, energy, manufacturing, and many others. Data engineering is a collaborative discipline. For successful execution, data engineering projects require the participation of software, reliability, and deployment engineers as well as domain experts, software developers, and end users of the deployed systems.

8.2.2. Importance in Modern Applications

As our applications process increasing amounts of data, the skeleton of distributed, scalable, and fault-tolerant data systems is becoming a critical shared resource. Unlike application code, which can (and often should, for innovation) be specific to a single application, these data systems must be generic; they must provide a variety of capabilities that allow a diverse set of applications to extract utility from data.

Domain-specific applications demand available and low-latency services. Search and recommendation are classic examples of applications that deliver value by processing high volumes of queries. In the last decade, new application domains like machine learning have become critical for the Internet. ML today is a complex combination of modeling decisions and large data processing jobs that require distributed systems for gradient computation. The rapid innovation in flows like AutoML and Federated Learning is built on these foundations. Clearly, we cannot afford to delay the execution of gaze prediction in an augmented reality application because federated learning is building a model in the cloud. The model is stored in a distributed, scalable, and fault-tolerant data system. For other application domains, like computer vision, the ML system for prediction is also the data system. These systems can thus accelerate processing even more, using shared resources for shared work. These transient demands require transient

data systems, elastic data systems which can automatically expand and contract their resources based on the demand.

8.3. Distributed Systems Overview

coordinated manner on multiple computer systems. These systems can consist of rack servers in the same data center, computers over a wide area network, or any combination thereof. Distributed systems solve multiple redundant copies of data and queries, distributing their execution across multiple computer systems. By also allowing replication of data, the processing workload for individual data queries may be reduced, allowing for increasing throughput.

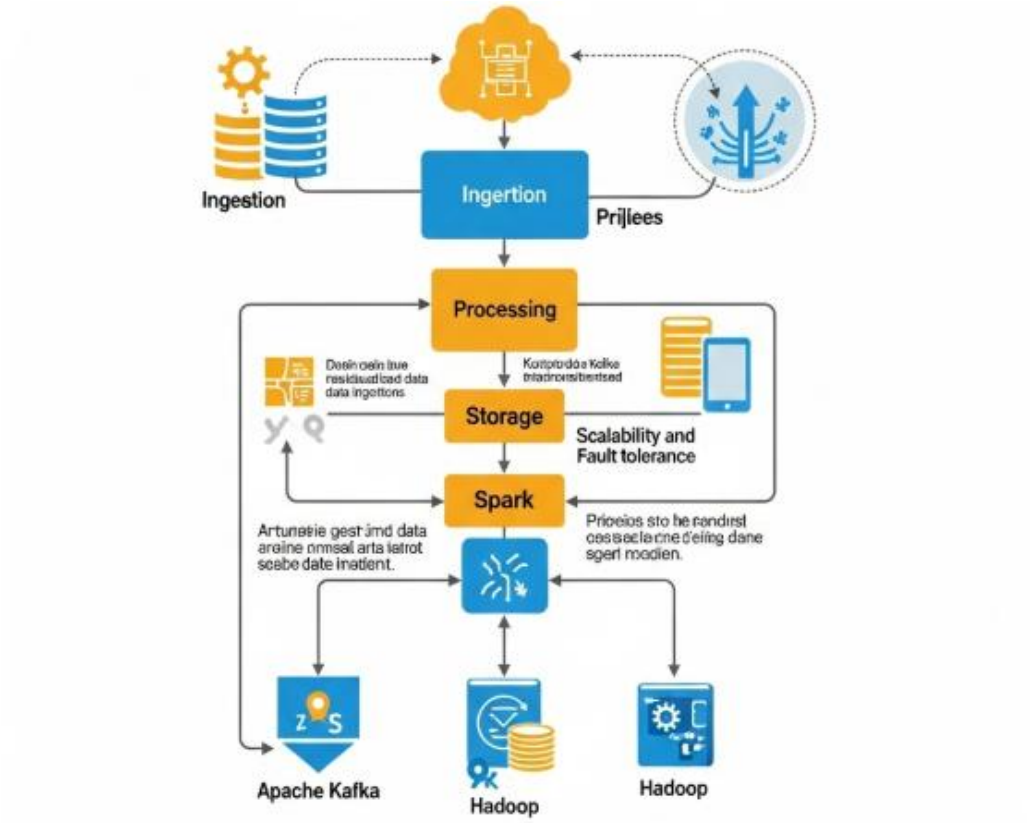

Fig 8.2: Distributed Systems Overview of Applying Data Engineering

The main two advantages of distributed systems driving their deployment are reliability and performance. Reliability is achieved through both data replication and failure isolation. The data is replicated across multiple nodes such that a failure of one node does not render any data unavailable. Failure isolation is the idea that not all data has a single point of failure. This is critically important for units of work that modify stateful

data and rely on a sequence of operations being done in a specific order. If two operations on the same stateful data are executed on two different nodes, the system has no way of guaranteeing that they will be executed in the correct order. Some operations may conflict with each other and lead to system inconsistency by violating invariants.

While performance improvement is one of the driving pushes towards adopting distributed data systems, it is also one of the foremost challenges associated with them. To understand the challenges and limitations in optimizing performance in distributed systems, it is important to understand what factors contribute to performance in such systems. For a single machine executing a process, the relevant characteristics are clock cycle speed, the number of cores, throughput of main memory and cache, and performance of the storage medium. These factors are relevant in achieving performance speedups and improvements in efficiency of work done per computational resource.

8.3.1. Characteristics of Distributed Systems

In this section, we briefly discuss the characteristics of distributed systems and highlight the key differences that define distributed data processing. There are a lot of definitions on what a distributed system is. The broadest definition is as follows:

A distributed system is a system that consists of multiple autonomous components communicating via a computer network.

While this definition is broad enough to capture any sort of networked system, there are some aspects of distributed systems and applications that differentiate them from systems that communicate via a computer network in a microkernel architecture. Some of these characteristics are as follows:

Transparency: The users perceive the distributed system as a whole and not as a combination of different components. The clients of a web application see it as a whole entity regardless of how many components run on how many servers. The designers of the system will have to understand the complexities introduced by distribution, such as result synchronization and communication failure, among others, and use those to provide a uniform interface. The operating mechanism behind a distributed database is usually different from that of a centralized database, but the end users see both as the same.

Scalability: A distributed system is often composed of multiple servers and each of these servers has capabilities that are limited by the underlying hardware. A distributed database or computation has to be designed to effectively utilize the resources available on all servers such as CPU, memory, and network bandwidth to provide liveness, availability, and performance.

Autonomous components: Each of the components in a distributed system is, or can be, an autonomous unit. In the context of a distributed data system, each of the component databases can operate independently and, within certain limits, tolerate failures of other component databases.

8.3.2. Challenges in Distributed Data Processing

Distributed file systems, cluster schedulers, and bulk data processing systems represent contemporary implementations of several decades of research into file system, task scheduling, and programming model design. This overwhelming body of work indicates decades of practical experience in building and using these systems. This experience makes understanding the underlying design choices essential for developing the next generation of computing infrastructure. Before diving into the details of several distributed data processing systems, it is necessary to understand the main challenges—and possible engineering tradeoffs—perceived over the years.

Despite their shared goal of providing a scaled-up computation platform, the above-mentioned systems address different problems, implement different interfaces, and expose different tradeoffs. The main difference in these approaches is in the specific data movement operations that they optimize for. This data movement task—the actual data transfers for obtaining the inputs and returning the outputs for a task execution—accumulates very fast in modern services processing large volumes of data. Depending on the nature of the tasks deployed at the cluster, the data movement may actually account for an overwhelming portion of the overall processing, overshadowing the computation logic, specifically in adversarial situations where different user jobs compete for shared resources. Hence, systems that merely aim to offer as much computation power as possible fail to consider this important factor and are of limited use. Other systems that do offer a variety of optimized data movement operations, expose the underlying complexity and do not automate the high-level data movement design choice boilerplate, making it completely impractical for the majority of domain developers to actually use those.

8.4. Scalability in Data Systems

One of the most important considerations in any system that stores data is its power to scale. If there are 100 stored data objects, throughput will be a function of object size, but if there are a billion objects, it will also depend on the number of storage nodes and how they are organized. As web-based applications for social interaction, shareware distribution, and instant messaging grow in popularity, the data systems used in these and other applications are frequently subject to demands that would cripple a traditional

system. Millions of users interact simultaneously, creating data objects for themselves and their friends — and perhaps later deleting them — at rates of 1000 per minute. Data about user interactions is logged for analysis of user behavior and for behavioral advertising. If only a fraction of the appropriate subset of data can be mined for these last two functions, response times become too long, degrading user experience.

Data storage and management systems that can meet these demands must be designed and constructed so that they can be scaled economically as the enormous volume of similar data grows. They must be distributed so that the job of managing all of the data is efficiently handled by a large number of computers rather than by a single machine. High system throughput is of little use if it can only be achieved by a supercomputer. By these criteria, almost all of the data storage and management systems being used to handle big data today are housed on clusters of commodity computer nodes connected by a Local Area Network and running a Linux-based operating system.

8.4.1. Vertical vs. Horizontal Scaling

Before we delve into the various scaling techniques available in modern data systems, we should distinguish between two different classes of scaling. A system can be scaled in one of two directions—vertically or horizontally. The most typical form of scaling, what is called vertical scaling, is one in which a single node is made larger (usually by adding more and faster core processors, more memory, and more I/O bandwidth and capacity). The main virtue of vertical scaling lies in its simplicity. Entities that have outgrown a small system, such as a single workstation, are able to move to a larger but still small system without having to change any programming or data management paradigms (the same is true when a department or laboratory entity moves to a larger mid-range system).

But although vertical scaling is simple, it is fraught with potential problems. The foremost one is physical limitation—there can be only so much packaging space, power, and dissipated heat from large numbers of chips. When I was doing research in the area of multiprocessor performance more than twenty years ago, I would have been foolish to bet that anything larger than a two-processor symmetric multiprocessor was ever going to be commercially viable, given the limitations of power and heat dissipation, interprocessor bandwidth, and reduced per-chip performance scalability. I am now very grateful for my stupidity. Although it took a long time, multi-piece chips or very large chips are now available, and allow performance scaling for traditional workloads. And for some highly parallel workloads, larger chips have been built, and multi-chip processors are available that support viable performance scaling.

8.4.2. Techniques for Achieving Scalability

When faced with a scalability need, one of the first design considerations is how to increase capacity without compromising original design decisions. For example, it is possible to serve many more requests with the same number of machines if the machine servers are fast enough. However, dependably and cheaply increasing capacity by merely scaling-up vertically is often not a design choice. Likewise, other original design decisions may constrain future response time. Scaling out horizontally using clusters of commodity components typically leads to design modifications to support distribution, including creating better defined and simpler abstractions, designing around failure, and using replication and partitioning.

Once the software has made the necessary architectural adaptations for scale-out design and has implemented coarse-grained abstractions and away from failure, provisioning and monitoring concerns become paramount. Machines need to be added or removed from the cluster, and requests routed appropriately. Often there is a third-party service that handles this task; when this or a comparable service is not available, we need to build Reserved Kick-Ass Data Systems to automate scale-out.

8.5. Conclusion

Throughout this book we have analyzed various design patterns, methodologies, and technologies to implement scalable and resilient data systems, stressing that these systems are enablers of technical innovativeness in many different areas in the economy. You have acquired the knowledge needed to plan, build, or operate such systems, and built some of your own by following case studies and doing the exercises at the end of most chapters. When thinking back to what challenges motivated this book, we see at least three kinds. First, there is still much to learn about the specific ways of implementing storage, processing, or streaming systems. These are exciting challenges in areas such as system architecture, resource allocation, storage mechanisms, or programming models. The second challenge is related to making data systems better collaborators: how do we enable data scientists, data engineers, and the domain users to interact more effectively to achieve the goal of efficient machine learning systems? The last challenge is the interaction of data systems with other areas in computer systems. Recent years have seen amazing developments in the area of AI and machine learning. The question remains whether these developments will affect how we build data systems. Adaptive systems could be able to make more intelligent decisions about storage, I/O, or compute allocation decisions. There have also been recent claims that the great advantage of deep learning will come as much from the development of effective pre-trained models as from building better systems. Will this reduce some of the traditional strengths in the area of systems?

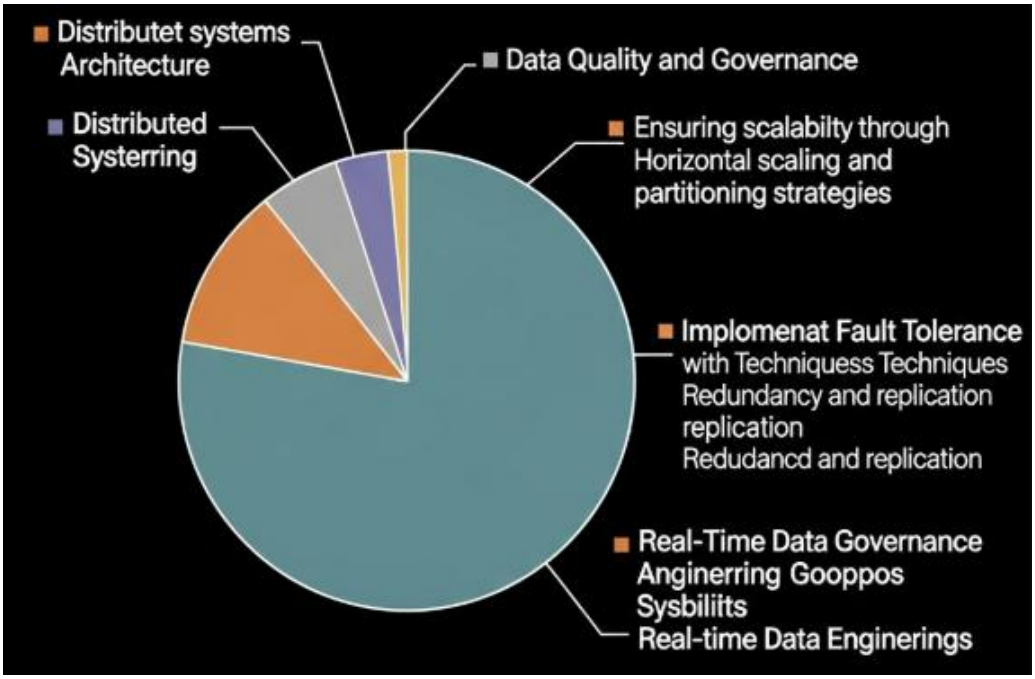

Fig : Data Engineering Principles to Build Distributed

8.5.1. Future Trends

Chapter 8 has outlined a veritable zoo of data systems that a modern software architect may have to deal with during the construction of large-scale applications. As related to the discussion in Chapter 1, it seems evident that data systems have evolved in response to nine identifiable problem threads. So how might these threads grow and mutate in the future? What trends do data systems architects need to keep an eye on as they evaluate new systems and techniques?

A long-standing desire in the database community has been to invent mechanisms to automate all or part of the work of a database administrator. With the exception of auto-tuning SQL query optimizers for fairly restricted queries, this desire has not been realized in practice. A more recent trend is to build self-managing data systems. This is an idea that is gaining traction with the recent emergence of self-managing capabilities in data systems, where additional servers are automatically provisioned to share the load when certain load thresholds are reached, or which takes advantage of community sharing to load balance essentially interactively as demand for concurrent query execution on a single cluster of hardware fluctuates. But these are relatively modest steps. How the all-the-way-to-self-managing data systems can be automated remains an open research question.

Another area of continuous growth is support for Big Data workloads in traditional data systems. Deployed data processing systems originally built to support operational workloads have taken on some analytics workloads as users have learned that many queries are amenable to being expressed as SQL-92 queries. Similarly, analytics data systems designed for large batch workloads are continuously extending beyond their traditional workload domains to support all more modes of operation.

References

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50–58.

Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley.

Gollapudi, S. (2021). Applied Machine Learning Operations (MLOps). Apress.

Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.

Krishnan, K. (2013). Data Warehousing in the Age of Big Data. Morgan Kaufmann.