Systems And Methods For Big And Unstructured Data

Theory

Christian Rossi

Academic Year 2024-2025

Abstract

The course is structured around three main parts. The first part focuses on approaches to Big Data management, addressing various challenges and dimensions associated with it. Key topics include the data engineering and data science pipeline, enterprise-scale data management, and the trade-offs between scalability, persistency, and volatility. It also covers issues related to cross-source data integration, the implications of the CAP theorem, the evolution of transactional properties from ACID to BASE, as well as data sharding, replication, and cloud-based scalable data processing.

The second part delves into systems and models for handling Big and unstructured data. It examines different types of databases, such as graph, semantic, columnar, document-oriented, key-value, and IR-based databases. Each type is analyzed across five dimensions: data model, query languages (declarative vs. imperative), data distribution, non-functional aspects, and architectural solutions.

The final part explores methods for designing applications that utilize unstructured data. It covers modeling languages and methodologies within the data engineering pipeline, along with schema-less, implicit-schema, and schema-on-read approaches to application design.

Contents

1 Introduction **1**
1.1 Big data . . . 1
1.1.1 Data analysis . . . 2
1.2 Relational databases . . . 3
1.2.1 Model characteristics . . . 4
1.3 Relational databases . . . 4
1.3.1 ER to relational transformation . . . 6
1.4 Data architectures . . . 7
1.4.1 Data partitioning . . . 7
1.4.2 Data replication . . . 7
1.4.3 Scalability . . . 8
1.5 NoSQL Databases . . . 8
1.5.1 Data lake definition . . . 8
1.5.2 Scalability . . . 8
1.5.3 CAP theorem . . . 9
1.5.4 BASE properties . . . 9
1.5.5 NoSQL history . . . 10

2 NoSQL databases **11**

CHAPTER 1

Introduction

1.1 Big data

Effectively leveraging big data requires the establishment of a comprehensive data management process that encompasses all stages of the data pipeline. This process includes data collection, ingestion, analysis, and the ultimate creation of value. Each stage is critical to transforming raw data into actionable insights that can drive decision-making and generate tangible benefits. The key components of this process are outlined below:

1. *Data collection*: gathering data from a wide range of sources is the foundation of any big data initiative.

2. *Data analysis*: the collected data must be meticulously analyzed to uncover patterns, trends, and insights. This analysis is tailored to the needs of various stakeholders. Analytical methods include descriptive analysis, which provides a snapshot of current data trends, and predictive analysis, which forecasts future developments.

3. *Value creation*: the final step in the data pipeline is the creation of value from the analyzed data. This value can manifest in several ways.

Big data is becoming increasingly prevalent due to several key factors:

- *Declining storage costs*: as hard drives and storage technologies become more affordable, organizations can store vast amounts of data more economically, making it feasible to accumulate and analyze large datasets regularly.

- *Ubiquitous data generation*: in today's digital age, we are all constant producers of data, whether through our interactions on social media, the use of smart devices, or routine activities online. This continuous data generation contributes to the exponential growth of big data.

- *Rapid data growth*: the volume of data is expanding at a rate that far outpaces the growth in IT spending. This disparity drives the need for more efficient and scalable data management solutions to keep up with the increasing data demands across various industries.

Big data is characterized by several key attributes that distinguish it from traditional data management paradigms. These attributes include:

- *Volume*: refers to the immense scale of data generated and stored. Big data encompasses vast quantities, ranging from terabytes to exabytes, made possible by increasingly affordable storage solutions.
- *Variety*: describes the diverse forms in which data is available. Big data includes structured data (such as databases), unstructured data (like text and emails), and multimedia content (including images, videos, and audio).
- *Velocity*: represents the speed at which data is generated, processed, and analyzed. Big data often involves real-time or near-real-time data streams, enabling rapid decision-making within fractions of a second.
- *Veracity*: concerns the uncertainty and reliability of data. Big data often includes information that may be imprecise, incomplete, or uncertain, requiring robust methods to manage and ensure data accuracy and predictability.

1.1.1 Data analysis

As the amount of data continues to grow, our methods for solving data-related problems must also evolve. In the traditional approach, analysis was typically performed on a small subset of information due to limitations in data processing capabilities. However, with big data, we adopt an innovative approach that allows us to analyze all available information, providing a more comprehensive understanding and deeper insights.

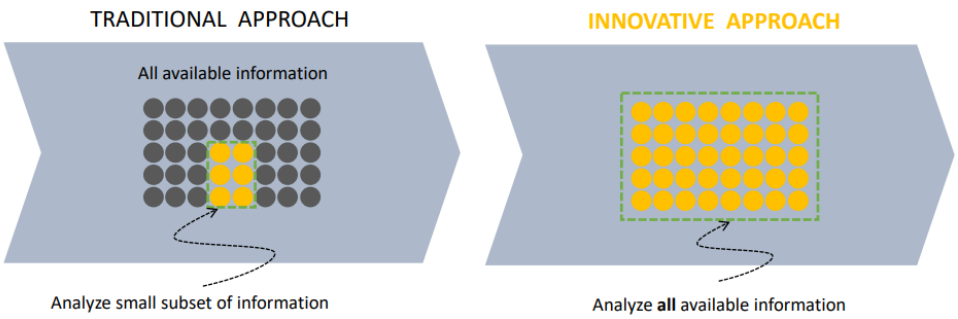

Figure 1.1: More data analyzed

In the traditional approach, we typically start with a hypothesis and test it against a selected subset of data. This method is limited by the scope and size of the data sample. In contrast, the innovative approach used in big data allows us to explore all available data, enabling the identification of correlations and patterns without pre-established hypotheses. This data-driven exploration opens up new possibilities for discovering insights that might have been overlooked using traditional methods.

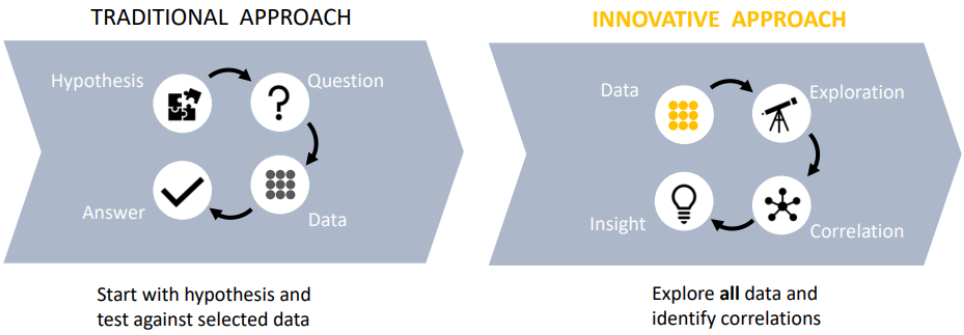

Figure 1.2: Data driven exploration

In the traditional approach, we meticulously cleanse data before any analysis, resulting in a small, well-organized dataset. In contrast, the innovative approach involves analyzing data in its raw form and cleansing it as necessary, allowing us to work with a larger volume of messy information.

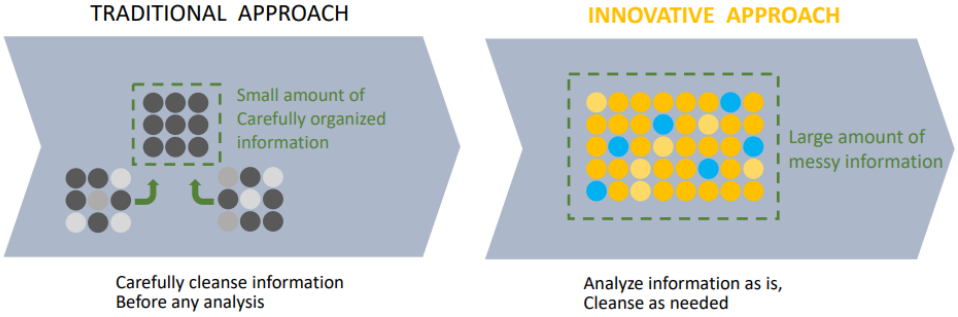

Figure 1.3: Less effort

In the traditional approach, data is analyzed only after it has been processed and stored in a warehouse or data mart. Meanwhile, the innovative approach focuses on analyzing data in motion, in real-time as it is generated.

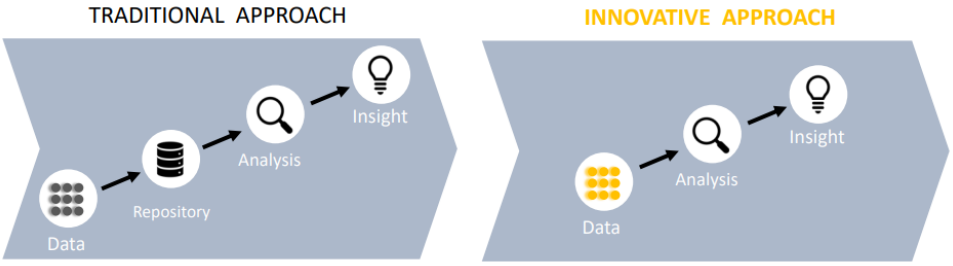

Figure 1.4: Streaming data

1.2 Relational databases

The design levels of a database are the following:

- *Conceptual database design*: constructing an information model, independent from all physical consideration for an enterprise. In entity relationship databases we have: entities, relationships, attributes, attribute domain, and key attributes.

- *Logical database design*: building an organization database based on a specific data model
- *Physical database design*: implementing a database using specific data storage structure(s) and access methods,

1.2.1 Model characteristics

In entity-relationship database models, key components include entities and relationships. An entity represents a distinct real-world object that can be differentiated from others, characterized by a set of attributes. These entities are grouped into an entity set, which consists of similar entities that share the same attributes. Each entity set is identified by a unique key made up of a set of attributes, and each attribute has a defined domain.

Relationships are associations among two or more entities. A relationship set is a collection of similar relationships, where an n-ary relationship set R connects n entity sets $E_1, \ldots, E_n$. Each relationship in this set involves entities from the corresponding entity sets. Notably, the same entity set can participate in different relationship sets or assume various roles within the same set. Additionally, relationship sets can have descriptive attributes. Uniquely, a relationship is defined by the participating entities without relying on descriptive attributes, while the cardinality indicates the number of potential connections between the entities. ISA hierarchies can further enhance the model by adding descriptive attributes specific to subclasses.

Aggregation comes into play when modeling a relationship that involves both entity sets and a relationship set. This technique allows us to treat a relationship set as an entity set, facilitating its participation in other relationships.

Conceptual design Crucial design choices involve determining whether a concept should be modeled as an entity or an attribute, and deciding if it should be represented as an entity or a relationship. It is essential to identify the nature of relationships, considering whether they are binary or ternary, and whether aggregation is appropriate. In the ER model, it is important to capture a significant amount of data semantics. However, some constraints cannot be represented within ER diagrams.

1.3 Relational databases

The SQL standard was first proposed by E. F. Codd in 1970 and became available in commercial DBMSs in 1981. It is based on a variant of the mathematical notion of a relation. Relations are naturally represented by means of tables.

Given n sets $D_1, D_2, \ldots, D_n$, which are not necessarily distinct:

Definition (*Cartesian product*)**.** The Cartesian product on $D_1, D_2, \ldots, D_n$, denoted as $D_1 \times D_2 \times \cdots \times D_n$, is the set of all ordered n-tuples $(d_1, d_2, \ldots, d_n)$ such that $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$.

Definition (*Mathematical relation*)**.** A mathematical relation on $D_1, D_2, \ldots, D_n$ is a subset of the Cartesian product $D_1 \times D_2 \times \cdots \times D_n$.

Definition (*Relation domains*)**.** The sets $D_1, D_2, \ldots, D_n$ are called the domains of the relation.

Definition (*Relation degree*)**.** The number n is referred to as the degree of the relation.

Definition (*Cardinality*)**.** The number of n-tuples is called the cardinality of the relation.

In practice, cardinality is always finite.

Definition (*Ordered set*)**.** A mathematical relation is a set of ordered n-tuples $(d_1, d_2, \ldots, d_n)$ such that $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$, where:

- There is no specific ordering between n-tuples.
- The n-tuples are distinct from one another.

The n-tuples are ordered internally: the i-th value comes from the i-th domain.

Example:
Consider a simple mathematical relation:

$$\text{game} \subseteq \text{string} \times \text{string} \times \text{integer} \times \text{integer}$$

For instance:

Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	1	2
Roma	Milan	0	1

Each of the domains has two roles, which are distinguished by their position. The structure is positional.

We can move towards a non-positional structure by associating a unique name (attribute) with each domain, which describes the role of the domain. For instance:

Home team	**Visiting team**	**Home goals**	**Visitor goals**
Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	1	2
Roma	Milan	0	1

Definition (*Relation schema*)**.** A relation schema consists of a name (of the relation) R with a set of attributes $A_1, \ldots, A_n$:

$$R(A_1, \ldots, A_n)$$

Definition (*Database schema*)**.** A database schema is a set of relation schemas with different names:

$$R = \{R_1(X_1), \ldots, R_n(X_n)\}$$

Definition (*Instance of a relation*)**.** A relation instance on a schema $R(X)$ is a set r of tuples on X.

Definition (*Instance of a database*)**.** A database instance on a schema $R = \{R_1(X_1), \ldots, R_n(X_n)\}$ is a set of relations $r = \{r_1, \ldots, r_n\}$, where r_i is a relation on R_i.

The relational model imposes a rigid structure on data:

- Information is represented by tuples.
- Tuples must conform to relation schemas.

There are at least three types of null values:

- *Unknown value*: there is a domain value, but it is not known.
- *Non-existent value*: the attribute is not applicable for the tuple.
- *No-information value*: we don"'t know whether a value exists or not (logical disjunction of the above two).

DBMSs typically do not distinguish between these types of nulls and implicitly adopt the no-information value.

An integrity constraint is a property that must be satisfied by all meaningful database instances. It can be seen as a predicate: a database instance is legal if it satisfies all integrity constraints. Types of constraints include:

- Intrarelational constraints (e.g., domain constraints, tuple constraints).
- Interrelational constraints.

Definition (*Key*)**.** A key is a set of attributes that uniquely identifies tuples in a relation.

A set of attributes K is a superkey for a relation r if r does not contain two distinct tuples t_1 and t_2 such that $t_1[K] = t_2[K]$. K is a key for r if K is a minimal superkey (i.e., there exists no other superkey K' of r that is a proper subset of K).

Primary Keys The presence of nulls in keys must be limited. A practical solution is to select a primary key for each relation, on which nulls are not allowed. Primary key attributes are underlined in notation. References between relations are realized through primary keys.

Foreign Keys Data in different relations are correlated by means of values of (primary) keys. Referential integrity constraints are imposed to ensure that these values correspond to actual values in the referenced relation.

1.3.1 ER to relational transformation

To transform an Entity-Relationship (ER) diagram into a relational database schema, the following steps should be performed:

1. *Create a separate table for each entity*:
    - Each attribute of the entity becomes a column in the corresponding relational table.
    - Each instance of the entity set becomes a row in the relational table.
2. *Handle relationships*:
    - For each relationship in the ER diagram, decide whether to represent it as a separate table or as a foreign key in an existing table.
    - Binary relationships with a one-to-many or many-to-one cardinality can often be handled by adding a foreign key to the table corresponding to the many side.
    - Many-to-many relationships typically require the creation of a separate relationship table, where foreign keys from the related entities form the primary key of the new table.

1.4 Data architectures

The data schema ensures typing, coherence, and uniformity within a system.

Definition (*Transaction*)**.** A transaction is an elementary unit of work performed by an application.

Each transaction is encapsulated between two commands: `BEGIN TRANSACTION` and `END TRANSACTION`. During a transaction, exactly one of the following commands is executed:

- `COMMIT WORK` (commit): confirms the successful completion of the transaction.
- `ROLLBACK WORK` (abort): reverts the system to its state before the transaction began.

Definition (*OnLine Transaction Processing*)**.** A transactional system (OLTP) is a system that defines and executes transactions on behalf of multiple, concurrent applications.

1.4.1 Data partitioning

The main goal of data partitioning is to achieve scalability and distribution. Partitioning divides the data in a database and allocates different pieces to various storage nodes. This can be done in two ways:

- *Horizontal partitioning* (sharding): data is divided by rows, where different rows are stored on separate nodes. Sharding is often used to distribute data in large-scale systems, spreading the load across multiple machines.
- *Vertical partitioning*: data is divided by columns, where different columns are stored on different nodes. This method is useful when certain columns are accessed more frequently than others, allowing for optimization of data retrieval.

Partitioning has its advantages and disadvantages. On the plus side, it allows for faster data writes and reads, and comes with low memory overhead. However, it can also lead to potential data loss if not properly managed, especially in cases of node failures or partition mismanagement.

1.4.2 Data replication

The aim of data replication is to provide fault-tolerance and reliable backups. In replication, the entire database is copied across all nodes within a distributed system, ensuring that there are multiple copies available in case of failure.

Replication offers certain benefits. For instance, it provides faster data reads since multiple copies of the data are stored on different nodes, and it greatly increases the reliability of the system, as the risk of losing all copies of the data is significantly reduced.

However, replication also comes with certain drawbacks. It leads to high network overhead, as nodes must constantly synchronize data to ensure consistency. Additionally, replication increases memory overhead since the full dataset is duplicated across all nodes in the system.

1.4.3 Scalability

We aim to create a system with elasticity.

Definition (*Elasticity*)**.** Elasticity refers to the ability of a system to automatically scale resources up or down based on demand, ensuring efficient use of resources and cost-effectiveness without compromising performance.

Data Ingestion Data ingestion is the process of importing, transferring, and loading data for storage and future use. It involves loading data from a variety of sources and may require altering or modifying individual files to fit into a format that optimizes storage efficiency.

Data Wrangling Data wrangling is the process of cleansing and transforming raw data into a format that can be analyzed to generate actionable insights. This process includes understanding, cleansing, augmenting, and shaping the data. The result is data in its optimal format for analysis.

1.5 NoSQL Databases

NoSQL databases are designed to provide greater flexibility in handling data, making them ideal for systems that require scalable and dynamic data structures. Unlike traditional relational databases, which rely on rigid schemas, most NoSQL databases operate without an explicit schema, or they may have an implicit, flexible schema that can evolve over time.

The absence of a fixed schema allows NoSQL databases to handle a wide variety of data formats, making them suitable for unstructured or semi-structured data, such as JSON, XML, or key-value pairs. This flexibility enables the efficient storage and retrieval of large-scale data in modern applications, where data structures may change frequently and unpredictably.

Paradigmatic shift introduce by Big Data: from Schema On Write ... • Long lasting discussion about the schema that can accommodate all needs • Some analysis can no longer be performed because the data were lost at writing time Load data first, ask question later All data are kept, the minimal schema need for an analysis is applied when needed New analyses can be Write Data introduced in any point in time

Object relational mapping • Impedance Mismatch • Object-Relational Mapping problem (and solution) • Object Orientation (OODB) • Commercial failure!

The process that extracts data from heterogeneous data sources, transforms it in the schema that better fits the analysis to perform and loads it in the system that will perform the analysis.

1.5.1 Data lake definition

1.5.2 Scalability

"Traditional" SQL system scale vertically: • Adding data to a "traditional" SQL system may degrade its performances • When the machine, where the SQL system runs, no longer performs as required, the solution is to buy a better machine (with more RAM, more cores and more disk) • Big Data solutions scale horizontally • Adding data to a Big Data solution may degrade its performances • When the machines, where the big data solution runs, no longer performs as required, the solution is to add another machine

1.5.3 CAP theorem

A transaction is a unit of work enjoying the following properties: • Atomicity: atransaction is an atomic transformation from the initial state to the final state • Consistency: the transaction satisfies the integrity constraints • Isolation: a transaction is not affected by the behavior of other, concurrent transactions • Durability: the effect of a transaction that has successfully committed will last forever

Theorem 1.5.1. *It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:*

- *Consistency: all nodes see the same data at the same time.*
- *Availability: node failures do not prevent other survivors from continuing to operate (a guarantee that every request receives a response about whether it succeeded or failed).*
- *Partition tolerance: the system continues to operate despite arbitrary partitioning due to network failures (e.g., message loss)*

A distributed system can satisfy any two of these guarantees at the same time but not all three.

In a distributed system, a network (of networks) in inevitable (by definition). Failures can, and will, occur to a networked system -¿ partitioned tolerance should be accommodated. Then, the only option left is choosing between Consistency and Availability- i.e., CA doesn't make any sense (expect when we have, e.g., a single-site databases; 2-phase commit, cache validation protocols)) Not necessarily in a mutually exclusive manner, but possibly by partial accommodation of both → trade-off analysis important CP: A partitioned node returns a correct value, if in a consistent state; a timeout error or an error, otherwise AP: A partitioned note returns the most recent version of the data, which could be stale. It's a trade-off! It's a continuous space! It's not at DBMS level, It's at operation level!

CA relational databases CP and AP NoSQL

1.5.4 BASE properties

ACID properties may not hold -¿ no properties at all then??? • focuses on availability of data even in the presence of multiple failures • spread data across many storage systems with a high degree of replication Rationale: • It's ok to use stale data (Accounting systems do this all the time. It's called "closing out the books.") ; it's ok to give approximate answers • Use resource versioning -¿ say what the data really is about – no more, no less. • The value of x is 5, at time T and date D •- So, shift the PH from 0-6 (acidic) to 8-14 (basic) – pure water's PH is 7 and neutral • Can some compromise be made between C and A?: • instead of completely giving up on C, for A • Instead of completely giving up on A, instead of c

(Basically Available, Soft-State, Eventually Consistent) • Basic Availability: fulfill request, even in partial consistency. • Soft State: abandon the consistency requirements of the ACID model pretty much completely • Eventual Consistency: at some point in the future, data will converge to a consistent state; delayed consistency, as opposed to immediate consistency of the ACID properties. purely a liveness guarantee (reads eventually return the requested value); but does not make safety guarantees, i.e., an eventually consistent system can return any value before it converges

No general answer to whether your application needs an ACID versus BASE consistency model. • Given BASE's loose consistency, developers need to be more knowledgeable and rigorous about consistent data if they choose a BASE store for their application. • Planning around

BASE limitations can sometimes be a major disadvantage when compared to the simplicity of ACID transactions. • A fully ACID database is the perfect fit for use cases where data reliability and consistency are essential.

1.5.5 NoSQL history

• MultiValue databases at TRW in 1965. • DBM is released by AT&T in 1979. • Lotus Domino released in 1989. • Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface. • Graph database Neo4j is started in 2000. • Google BigTable is started in 2004. Paper published in 2006. • CouchDB is started in 2005. The research paper on Amazon Dynamo is released in 2007. • The document database MongoDB is started in 2007 as a part of a open • source cloud computing stack and first standalone release in 2009. • Facebooks open sources the Cassandra project in 2008. • Project Voldemort started in 2008. • The term NoSQL was reintroduced in early 2009.

KINDS NOSQL

CHAPTER 2

NoSQL databases