Deep Science Publishing
https://doi.org/10.70593/978-93-49910-19-5_7

DeepScience
Open Access Books

Chapter 7: Data engineering for real-time processing, streaming analytics, and scalable decision intelligence

7.1. Introduction to Data Engineering

Data engineering emerges as a pivotal specialization within the broader landscape of computer engineering and computer science. This field's primary focus rests upon the careful design and optimization of data pipelines and architectures that serve the specific needs of a wide array of applications. These services span the realms of artificial intelligence, business intelligence, real time analytics, and other specialized domains. Nevertheless, the diversity of these use cases, with their heterogeneous requirements spanning a broad spectrum in terms of needs, scale, performance, and other supporting engineering metrics, entails that no one technology or solution is common across all these domains. Data applications display widely varying telemetry, at all levels of the technological stack (Gulisano et al., 2012; Akidau et al., 2015; Carbone et al., 2015).

A large fraction of the volumes ingested, stored, and processed typically accrue to applications that are part of real time serving systems. Furthermore, the pipelines and systems used to support the data, storage, processing, and serving needs of these applications must be optimized for throughput, latency, fault tolerance, query expressiveness, and any other domain-specific metric that is relevant, such as data retention and availability. These applications cover an important part of the data ecosystem, with reliable systems providing real time recommendations, personalization, and fraud detection functionality to users in question. Their obsolescence could potentially cause significant disruption to business ecosystems and the shape of the services offered by major economies, affecting companies that utilize recommendations to monetize their user interactions. The unprecedented scale of the customer bases already being served by some of these products provide a powerful incentive for

organizations to continually push the latency boundaries of the pipeline architecture (Stonebraker & Çetintemel, 2005; Hesse & Lorenz, 2019).

7.1.1. Overview of Data Engineering Principles

This work provides a practical introduction to data engineering. In this chapter, we will address the primary principles of data engineering, as the title suggests. Professionals from many different backgrounds and research areas will find many opportunities for contributing to the building blocks of modern data ecosystems. Throughout this work, our focus will be on the principles that enable the design and development of efficient data pipelines for data analytics, whether in batch processing or in real-time processing streaming analytics.

Data engineering is a part of the data life cycle that is dedicated to transforming, integrating, and preparing data for analysis. Data engineering is usually the product of collaboration between teams of ingestion engineering and data scientists. Exploring the frontiers of data engineering and using creativity to solve problems old and new is fun work. The output of our creative effort, effective data pipelines, are key to successful data science, data product, data journalism, and data visualization efforts. Data engineering is not a perfect replacement for data scientists' activities. However, a clear demarcation between data engineering on the one hand and data quality assessment and data science on the other is risky. We hope that the principles of data engineering that we present in this work will help in developing a productive working relationship, and productively share and open up the borders between data engineering and data science throughout the entire process of pulling data from different data sources to data products, from ingestion engineering to final analysis, visualization, and data-driven decision support.

7.2. Fundamentals of Real-Time Data Processing

Real-time data engineering consists of data and platform engineering techniques applied to situations where some customer experience relies on data and AI in a short time delay after the data was generated. A customer experience is any of the experiences of a person who consumes something, either in a business transaction and who has an experience associated with incidental that may or may not relate to a business transaction. The importance of data and algorithms in a customer decision made for every product purchase has caused organizations to leverage data and mathematics in virtually every activity that involves making a choice, and the cloud plays an important role by tying together storage, analytics, and artificial intelligence as a service.

Business operations are critical aspects of many businesses. Considering making customer experiences reliable involves managing the business risks by investing in the foundation and support for algorithm-based experiences. Real-time data management is the foundation for algorithm-based customer experiences and it is needed for three types of customer experiences: real-time experience, time-delayed experience, and background experience. Ensuring that the pipeline operates correctly, the predictions are accurate, and that the background analytics run efficiently or not enough changes are detected to trigger unnecessary updates is essential. Because the risks are significant, investments in admission, synchronizing, and monitoring components of streaming analytics require a strong business case to justify the scaling of cloud infrastructure.

The data management problems appearing in real-time processing are organically motivated. In real-time data engineering, pipeline and platform design focuses on characteristics and attributes that originated as problems or difficulties that were observed in solving data engineering problems from typical application domains.

Fig 7 . 1 : Data Engineering for Real-Time Processing

7.2.1. Key Concepts in Real-Time Data Processing

Real-time data processing is not just about responding to events in real time. Instead, the specific meaning of "real-time" that we use in data processing is closely related to the different types of data and what "timely" means in that context. In a decision tree that we will explore in this chapter, we are interested in user events, data that have temporal meaning, and events that need to be responded to shortly after they occur. These events come from streaming and/or real-time data sources, and achieving the business needs associated with those events requires the use of real-time data processing techniques, capabilities, and architectures. Furthermore, "real-time" is only one type of timeliness that we can explore, and only one of the dimensions on which we can base our classifications of data processing. Conceptually, it is simplest to think of a continuum of scenarios that vary, for different types of data, the key business need of timely response to events that occur at various collected timestamps.

There are many ways to classify and delimit specific features, capabilities, and constraints in real-time data processing systems. We will briefly summarize how they are explored to create a framework for understanding various data processing trade-offs so that we are able to make deliberate engineering choices when addressing any data processing challenge. In addition to providing a framework for navigating the large set of current solutions, this allows us to design future solutions that will be cataloged with the characteristics outlined here. Since data is created at different times, has different structures, and also can come from various processes, data management systems and architectures are optimized or specialized for different types of data and characteristics. "Processing," too, is a general term that refers to a number of possible transformations of data in a system.

7.3. Streaming Data Architectures

Supplying stream processing or analytics uses streaming data architectures, under a series of architectural approaches. Considering microservice architecture or event-driven architecture are the architectural pattern involved. Microservices is a recognized approach for constructive streaming applications, however, it does not enforce any temporal correlation for the exchange of messages between all microservices. An event-driven architecture allows guaranteeing temporal ordering for an arbitrary number of events. It can guarantee transactional consistency, and it focuses on an optimized mechanism for message delivery to different types of consumers. Nevertheless, it requires a specific infrastructure for data exchange and strongly couples producers and consumers. Moreover, indiscriminate usage of both architectures can introduce latency or complexity overheads in the design of a streaming system. In this sense, their trade-offs must be assessed in terms of consistency, complexity, latency, scaling, and coupling.

Microservices architecture is gaining adoption in stream processing applications, as small stand-alone executable units called microservices are assembled to offer a larger set of functionalities. Usually, each microservice exposes a set of endpoints for interaction, and a couple of microservices are responsible for the streaming aspects of an application. Microservices have low dependencies and are usually built using polyglot programming. They are easily deployable units with low operational cost, and they allow development teams to apply continuous integration pipeline tools. In addition, they provide an abstraction for scaling services where the demand is high, and they allow companies to apply various browsers for users and develop different functionalities on various programming platforms. Additionally, microservices divide functionalities, which allows teams to work independently on the same domain.

Some limitations also arise with the adoption of the microservices architecture, mostly related to their design, deployment, and implementation. From a design perspective, the decomposition of a monolithic application into microservices will involve service design derivation criteria focusing on functionalities currently decoupled in the monolithic version or how to enable the streaming aspects.

7.3.1. Microservices Architecture

Microservices architecture, or microservices, allow applications to be built as a series of independent components (services), cooperating with each other over a high-speed network. Microservices are well-known and increasingly popular. This sort of architecture is worth exploring in an IoT or data engineering context, as it has a number of interesting features and limitations.

Microservices provide business capabilities, reacting to requests from other components, or clients. Requests are sent using the HTTP protocol over a network, so the services are not directly communicating with each other using function calls or shared memory. Each service can provide a well-defined business capability, like User Registration, or Processing an Order. Other services can invoke the APIs provided by these services to get data or to trigger operations.

Microservices architectures are an evolution of service-oriented architectures, and have some resemblance with them. Microservices are easier to implement than SOA systems, because a small service is simpler and usually easier to manage than an entire platform; transitive dependencies and interfaces are usually simpler too, as they are smaller and more focused, and there are usually less integration issues, as development can happen independently and in parallel on different microservices. However, microservice architectures and SOA systems both suffer from the same problem: network communications. In a microservices application, data is exchanged over unreliable, low-

bandwidth networks, which are much slower than a local function call or direct memory access.

7.3.2. Event-Driven Architecture

In recent years, architects have adopted an event-driven architecture (EDA) for data engineering solutions in which the delivery, capture, and processing of events drive the design. An event has been defined as a significant change in a system state, which is also a meaningful event to a user or an outside observer. An EDA enables applications, services, and components to communicate and react to events in real-time by allowing events generated by producing components to trigger an action or a business process for consuming components. The components in EDA can include identity, authentication, configuration, versioning, network management, broker publishing and subscribing, chat, collaboration, logging, security, and transaction components. Communications use multiple protocols with assured and low-latency delivery, which are important for instant messaging and for many mobile applications. Throughout all of these changes, events must be reliable, secure, and auditable.

The event-driven model of operations adds additional detail and capabilities to a microservices-based architecture for event-driven applications and flows. Event-driven operations add more than simple event message processing, for example, support for collaboration and a multi-device user experience. Timer-based notifications take the place of push notifications to mobile applications. Events and their resulting notifications usually contain basic information, plus the information required for any specific event type. The specified information varies by event type — be it an instant message, a chat room update, a transaction, a page-access update, a security event, or some other type of event. Events tend to be cached, versioned, filtered, and repositioned until their time is ready for notification.

7.4. Data Ingestion Techniques

Data ingested into a data lake inevitably come from diverse sources and through different tools. While establishing a distributed data lake, data engineers encounter the challenge of ingesting data into the lake before the actual analysis. This sounds relatively straightforward, but the challenge lies in the wide diversity of data sources and tools and their specific idiosyncrasies to ingest data effectively. This chapter discusses the prerequisite of data ingestion into a data lake and focuses on the ingestion strategies and techniques.

Data ingestion refers to the process of moving data from one or more source systems into a destination storage system. Data injection techniques could be classified into different types: extract, transform, and load ingestion, extract and load ingestion, etc. ETL ingestion describes the movement of data with transformations, while EL ingestion describes the direct movement of data that can be used in the destination system without further transformation. A more general distinction defines batch ingestion and real-time ingestion. This book focuses on the second one since we are interested in streaming analytics where real-time ingestion enables real-time analytics. Data engineers usually discuss ingestion strategies such as batching, near-real-time, and real-time streaming, and these three methods basically use batch methods for a certain time slice of the data or only ingest small portions of the data in near real-time or real-time for each data arriving event.

Data ingestion is normally done when data flows into a data lake but, at times, a snapshot of a cold source system is ingested into the data lake such as while migrating legacy systems or retroactively gathering information about an event. It should also be noted that when data is ingested from multiple source systems on a regular schedule, the operation is termed as data extraction rather than data ingestion. A special case of data extraction for cloud-based data sources is called data replication; tools that provide this service are referred to data replication services.

7.4.1. Batch vs. Stream Processing

The amount of data generated by users and devices is increasing exponentially. Enterprises are challenged in continuously monitoring large amounts of data, due to volume, velocity, and/or frequency, to rapidly detect anomalies in user behavior, system behavior, and external event impact. Inside many enterprises, the data generated from their operations consist of variable frequency, variety, and volume streams while their relevant monitoring operations submit generally static frequency alerts. These enterprises seek for process analytics and data professionals capable of building real-time anomaly detection analytic methods leveraging these entire diverse data collections.

In most business analytical traditions, corporate operations are monitored leveraging traditional data warehouses fed by batch processes that periodically load data from source or operational databases to data warehouse tables. This architecture relies on periodic batch jobs primarily because most operational backend systems, used to capture business transaction data, were designed to ensure data consistency, data integrity, and data availability generated by transactions during set execution periods. Generally, batch jobs have access to the entire data collection for preprocessing since all data is consistently available. However, these designs tend to introduce significant delays in data availability for use by corporate top management. At the same time, it is recognized

that some monitoring operations do not require data availability for the entire transaction execution process and would benefit from rapid detection of anomalies and early warning alerts that could then be incrementally improved after the fact using existing static frequency batch analysis methods.

7.4.2. Message Queues and Brokers

Message queues decouple application components by providing a simple mechanism for them to communicate through a queue. When a producer wants to send a piece of information to the consumer, it pushes the data to the queue, where the consumer will later pick it up. Once it has been processed, it can be deleted. In this way, data is stored for an indeterminate time until the consumer is ready to process the message. Traditionally, this has been covered through OS process queues that manage processes related to hardware devices, and more recently through implemented libraries that were introduced in operating systems.

In data engineering, particularly with applications based on event- or message-driven architecture, more specific data queues have been developed for applications connected to data streams. In this case, a queue is a separated application whose function is to be in charge of managing the lifecycle of messages. Each message has an owner that created it and is waiting to process it. The process that consumes the message can also be separate from the one inside the queue. They can be on the same machine or even remote from each other. The queue guarantees that this message will be delivered at least once to the process that owns it, storing it until this process signals the queue that it has already processed it. If the process fails while processing it, the queue can resend it at a later time, thus preventing data loss.

7.5. Data Storage Solutions for Real-Time Analytics

Data Storage Solutions for Real-Time Analytics Traditionally, analytics has been mainly a retrospective activity. Data ingested through ETL is stored in Data Warehouses and Data Marts. The store-and-forward model makes analytics on this legacy approach have time-lags, so with a focus on the long term. Operations teams in organizations that want to embrace the culture of Data-Driven use the term real time operations based on the 0-time lag feedback from the analytics stored in Near-Real-Time Data Lakes. Analytics is not a sport of just the data aficionados; it is for everyone within the organization. Technology can help empower everyone with the necessary tools to perform analytics on the data relevant for the different domains of operations within organizations.

NoSQL Databases Created as an alternative to accommodate the horizontal scaling of Big Data volumes relentlessly growing, the NoSQL products have the most intense adoption in the data engineering ecosystems of organizations. With many NoSQL technologies available today, a detailed NoSQL framework used at scale by many organizations: Key-Value Stores, Wide-Column Stores, Document Databases, and Graph Databases. These NoSQL technologies natively support TTL, so it is common to have seven days, two weeks, or thirty days aggregation but not unlimited time as traditionally happens in Data Warehouses. Many NoSQL products or compatibles might have an Analytical Engine incorporated. The last-generation NoSQL Data Management technologies are focused on democratizing analytics by allowing everyone to perform ad-hoc analytics directly within the operational NoSQL Data Store.

7.5.1. NoSQL Databases

It is well-established that relational databases have limitations when it comes to scale, which is one of the reasons NoSQL databases were created. However, NoSQL databases were built to handle much higher ingestion rates and scalable random access reads on structured data without any constraint on schema, easily performing 100,000s of inserts per second and providing a key lookup latency on average of single digit milliseconds. NoSQL databases can also scale out horizontally, addressing the scale limitation that many prediction applications run into when processing data within various windows for purposes such as anomaly detection, forecasting or detection of various events, businesses utilize for purposes such as risk management, fraud detection, inventory management, etc. Fast data confusion causes lags in prediction results using prediction models built on historical data stored in RDBMSs that NoSQLs successfully overcome by doing real-time predictions within the fast data periods. As a result, fast data analytics and future data are often stored and processed using data structures provided by NoSQL databases.

NoSQL databases are often referred to as key-value stores. Each key-value pair is read and written using a key with the associated value being simple or complex attributes defined in the framework of the NoSQL stores. In these key-value pairs, a key is assumed to be unique and is used to represent various real-time data entities, such as customers, devices, and locations. These key-value stores have restrictions on the read, write, and query capabilities permitted on the attributes, although some allow secondary indexing on the attributes for building lookup tables utilized to filter and join data within a query. Other NoSQL databases provide predefined JSON document structures that allow complex nested documents within a document. These document stores have flexible schema support for unstructured and semi-structured data, data such as customers that come from different sources and change attributes frequently can be efficiently stored.

7.5.2. Time-Series Databases

Real-time analytics often involves working with time-series data. Time-series data is a sequence of observations of a process made over time. The observations are typically made at a uniform rate, and they usually have a timestamp associated with them. Characteristics that are typically associated with time-series data are: (1) store large amounts of data, (2) data is often only appended, (3) queries typically involve a time range, (4) require fast writes, (5) also require fast aggregation queries, (6) require special aggregation functions over time intervals, (7) require special functions for processing variations of time (for example: day of week, hour of day), and (8) data are often the same and/or update frequently.

Specialized time-series databases are designed specifically to take care of the above needs while following the principles of real-time analytics. A traditional relational database is not usually optimized to accommodate the requirement of real-time analytics. In fact, genuine time-driven or event-driven analytics cannot be done efficiently through a relational database. Moreover, an OLAP solution should not be bothered for such use cases. While NoSQL solutions can do the job, time-series databases do it better and at a lower cost. TimescaleDB is a relational database designed specifically for time-series data on top of PostgreSQL. It has many capabilities of a traditional relational database for general data and provides superior timeseries functionality on both storing the timeseries data and performing queries over the timeseries.

The other popular solutions that are widely used are InfluxDB and OpenTSDB. OpenTSDB is built on top of HBase. The architecture of OpenTSDB is very similar to that of traditional RDBMSs. MySQL is used for storing metadata and indexing data that speed up query. InfluxDB is a more recent and newer entry in this space. The architecture of InfluxDB is more complex and different. InfluxDB is a custom open-source, distributed time-series database. By default, it runs as a single server, but also offers clustering capabilities. Unlike OpenTSDB or TimescaleDB, InfluxDB is not built on top of another database, though it can use other key/value stores and databases, as a part of a pipeline for data collection.

7.6. Stream Processing Frameworks

There are several frameworks available for real-time data processing, and users may choose what they prefer depending on their specific use cases. In this section, we provide description of a few popular stream processing frameworks. Apache Kafka is a general pub-sub messaging system that allows flexible message routing based on user-defined topic partitioning, which makes it possible to separate different types of messages and separate consumers of each type based on business and quality-of-service needs. Users

can also use Schema Registry to enforce users' data schema as their data evolves. In addition, for organizations that run in the cloud, fully managed Kafka services are available, so creating a Kafka cluster does not take teams days. In spite of these advantages of Kafka, there are limitations on utilizing Kafka. First, it does not provide a compute framework. Next, Kafka is not routed for low latency data flow between an ingest system and compute system, because a normal roundtrip even within the same datacenter is about several milliseconds. For these reasons, organizations that use Kafka for event log repair also run an ETL program to transfer Kafka data to their data warehouses.

Apache Flink is a general-purpose stream processing framework. A Flink program consists of reading a stream from an input base, transforming the stream and writing the stream to an output base. Flink allows programmers to easily develop low-latency and high-throughput ETL programs and easily develop a complex event processing program that joins several streams, finds the count, distinct counts or averages of some fields in a windowed time period and defines stateful functions on the resulting streams. Flink allows for flexible time control. Stream records can have timestamps in two formats, such as event time and processing time. Stateful functions that are time-dependent can be defined to wait for a specific amount of time. Flink provides unique fault-tolerance capabilities. The state of a Flink program can be saved externally. Additionally, Flink provides batch-cum-stream processing capabilities with a few additional lines.

7.6.1. Apache Kafka

Apache Kafka is an open-source distributed streaming platform. The project started in 2010 to support efficient real-time data feeds and operational data storage. Like most other projects, Kafka has been highly tuned to meet scale and availability requirements. At the time of its release, it had its limitations. However, since then, many improvements have been made to the system, making it versatile enough to be used for a variety of use cases involving real-time processing.

Apache Kafka is not a general-purpose compute engine, but rather a high-throughput, low-latency platform for handling real-time data feeds. Kafka is comparable to a message-queue system. However, unlike queues, messages in Kafka are not deleted when consumed in order to support batch consuming; instead, the topic is partitioned, and the consumers in the same consumer group share a partition. Kafka is also comparable to a data store, but is designed for extremely high throughput and distributed on commodity machines. Data is stored on disk in a replicated, fault-tolerant manner in order to allow durable message queuing. Messages can be partitioned by key or published to an arbitrary partition.

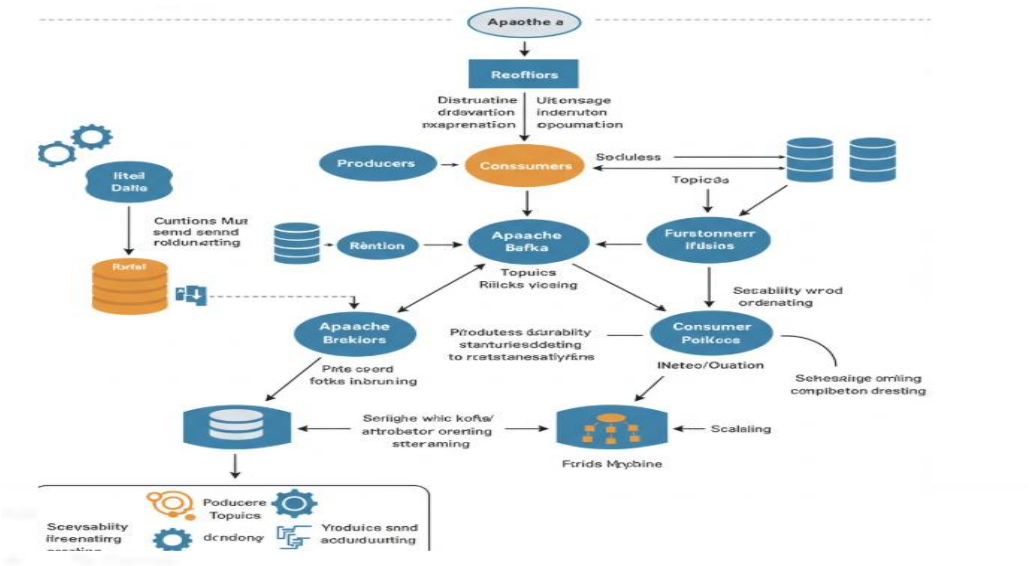

Fig 7 . 2 : Apache Kafka for Real-Time Data Streaming

7.6.2. Apache Flink

Apache Flink is a general-purpose stream processing framework that provides programming abstractions for distributed computation on data streams as well as a distributed execution engine. It is supported by the Apache Foundation and is one of the most widely adopted Big Data tools. Flink's programming model provides support for complex event processing, arbitrary stateful computations over data streams, and exactly-once state consistency driven by distributed snapshots. The Flink operators are declarative, and Flink's Lazy Evaluation Engine automatically optimizes the execution plan. Flink supports batch computation, although batch jobs are typically implemented using Spark. Internally, Flink represents batch jobs as special cases of stream computations; as a result, Flink's processing model is called bulk processing as streaming.

Flink provides support for both stream-based and micro-batch processing of data. In contrast to complex event processing, which focuses on the processing of a series of related events, stream processing and micro-batch processing support stateful, long-running computational windows over continuous streams of incoming data. Flink also supports an expressive CEP model that can detect complex event patterns. To detect complex event patterns, the event data must have an order. In CEP, the patterns must eventually match the events but are not guaranteed to match in a timely fashion. In contrast, stream processing is more general in that it obtains a result immediately. This support becomes important for applications such as monitoring, failure detection, fault

localization, predictive maintenance, and intrusion detection, which involve lightweight, short-lived computations over incoming data streams that identify outliers or abnormal behaviors.

7.6.3. Apache Spark Streaming

Apache Spark is an open-source big data processing framework primarily used for batch processing. It allows processing a large collection of datasets in a parallel, distributed manner. Apart from batch processing, Spark also supports other paradigms such as graph processing, machine learning, and stream processing or stream analytics. Spark Streaming is an extension of Spark that provides a framework for processing continuous streams of data.

Spark Streaming takes the data from the stream sources and divides the received streams into batches of a predetermined duration, thereby creating a Spark Mini Job for each batch. Then each mini job is executed by the Spark framework, and the results from each batch are pushed to sink services. The main Spark framework along with Spark Streaming provides a micro-batch stream processing. Other stream processing frameworks provide real-time micro stream processing, and while some provide real micro stream processing, Apache Spark Streaming has the advantage of providing a unified framework for batch, graph, machine learning, and stream processing. A single benefit of micro-batch processing is that it achieves more fault tolerance and more scalable stream analytics than real-time processing, and hence, it is easier to program the application using Spark Streaming than the other real-time stream processing services.

7.7. Real-Time Data Transformation

Real-time data transformation or transformation on the fly is an important step in stream processing and near-real-time analytics. Transforming raw data coming from one or more data sources allows creating usable data assets for analytics, machine learning, and other tasks that require input data to have a specific structure, be in a particular format, include only certain parts of the incoming data, or stored in a way that is optimal for specific queries. Traditionally, the data transformation step in the data management and analytics pipeline has been done by enterprise tools and products as part of the data warehousing process. As companies use data warehouses not only for reporting but also as the core of their data architecture, data engineering tasks, toolsets, and processes have evolved. The demand for interactive and personalized dashboards and insights for various stakeholders requires data to be transformed in real time, immediately after being collected. To support such real-time reporting use cases, the transformation step has increasingly been moved closer to or combined with the data ingestion step. The

traditional process of connecting to a data source, transforming the data, and making it available in a fast and usable form has evolved into a process whereby the component takes care of getting the data from sources, preprocessing, sometimes also enriching or filtering the data, and converting it into a form that can be easily used for analytic queries and putting it in the warehouse or data lake for the analytics job to execute and the dashboard refreshes to pick up the data. Instead of copying raw data to a staging area for the transformation and exporting to the warehouse, data transformation is performed in transit, as data is being written.

With completion of the ingestion process, the newly transformed data is written to the data warehouse or batch analytics systems for consumption by various stakeholders. When raw data needs to be quickly available in the warehouse for ad hoc reporting, data engineers and analysts need a way to trigger a partial reload of raw data. This usually involves creating a new version of the data in the warehouse targeting a specific set of newly ingested records.

7.7.1. ETL vs. ELT

Extract, transform, load (ETL) is the traditional method for moving data into a storage repository. Individuals and systems extract data from source repositories, and then the data is transformed, changes are made that prepare the data for its intended use in reporting and analysis, most often metadata functions like datatype conversions and data validation. Then the transformed data is loaded into a target repository, often a data warehouse or a database. ETL has served as the pipeline of reliable, high-fidelity data to traditional enterprise data warehouses and cloud-based data warehouses for structured reporting and analysis.

However, the emergence of cheap storage, combined with the scalability and robustness of cloud data warehouses, have caused a paradigm shift in data processing. The modern data approach of extract, load, and transform (ELT) moves a lot of raw data into compliant data clouds and allows data engineers to create pipelines for different types of reporting and analytics tools, such as data visualization, data science, and machine learning applications. ELT approaches are not just for today’s cloud consumption of data; they extend data retention for quick response time on analytical queries while allowing new applications to be built on top of existing data. While ELT is the preferred approach for the modern hybrid data processing approach in use by many companies, ETL actually remains the better choice in many real-time data processing applications.

7.7.2. Data Enrichment Techniques

Once raw data lands as streams in data lakes and warehouses, it is rarely ready for immediate consumption. In fact, data engineers should use data transformation and enrichment processes to add backending data, annotate, clean, or format the streams before loading them into finished parts of the architecture. These backending tasks enable data analysts, data scientists, machine learning models, or reporting dashboards to work with high-quality, meaningful data. Data enrichment analyses can also be run in real-time to generate fresh data during the actual execution of batch or streaming queries. How data enrichment processes are designed and how they behave can have a dramatic impact on the development and operational time for the entire analytics platform.

Stream and batch data enrichment processes share a number of similar characteristics. They describe a number of similar data enrichment techniques that address different problems, such as feature engineering for predictive analytics, backending processes that operate at wide scales, complex extracts from deep data sources, batching, or data quality operations that modify pretty much everything at the aspect of a single record, including correcting for delays in source data arrival times. Regardless of enrichment use case to be addressed, hatching of enrichment processes has been motivated by three key factors: network and processing costs, the presence of non-homogeneous environments and diversity of data sources; and the support for asynchronous or temporal dependencies or relations in data space. Designers of data enrichment processes and the operations that wrap and chain them have different goals. They can be interested in keeping the focus as much as possible on the design and then automatically optimizing for costs, or in guaranteeing some forms of costing transparency. Enrichment approaches also differ in how they factor and distribute computations among evolving states.

7.8. Real-Time Analytics and Insights

The analytical components of a real-time data architecture and pipelines apply techniques such as data preparation to clean and transform data and analytical models to derive new metrics or insights on the data. Keeping with the principles of data latency and amplification, it is important that the latency of these analytical models and processes are as low as possible, ideally real-time latency. Traditional approaches that train models offline and operate on a batch frequency are insufficient. Also, training and operating complex models that combine deep learning and other approaches may be impractical, so we need to focus on analytics that work well under real-time latency, low training frequency, and simple models. In this perspective, we categorize two types of analytics with specific model types: common real-time analytics based on time series forecasting with process security and predictive analytics based on predictive classification models.

Dashboards and Visualization

The simplest form of insights is a dashboard that visualizes a collection of signal values over time. While dashboards can be built using historical data, the best practice is to perform real-time updates. These dashboards use simple plots, such as line and area charts, that show the value of each variable for users to explore. In order to derive a deeper level of real-time insights, our experience is that the dashboards need to expose a small set of metrics that have increasing business value. Rather than just providing this information on the dashboards, they should also offer more detailed catalogs so that users can interact and explore the other variables easily.

Predictive Analytics

One of the most popular examples of real-time predictive algorithms is the classification model that determines whether an incoming message is spam. Frequent checkpoints of these classification models allow coming up with a reasonable estimate of the current predictive model and any incoming new messages are classified using the current estimate in real-time without any delay. Other predictive algorithms, including ones that forecast timestamped values for the future based on historical data, have been recently gaining popularity.

7.8.1. Dashboards and Visualization

While real-time processing is typically used just for the analysis and detection of certain events of interest, streaming analytics allows you to see the results at any time of the processing pipeline. This functionality is generally exposed as a dashboard with various components like timelines, displays, charts, and maps that illustrate the results of the processing. Dashboards are not only used for exploration and querying but also to visualize patterns over time of the metrics and aggregated data. The visualization of the results should ideally provide the user with insights about the real-time behavior of the system, helping him to understand patterns that can be further investigated and operationalized. As opposed to data engineering for static batch processing, the design of these dashboards should account for two aspects. The first aspect is the actual ability to visualize and analyze data in real time. The second aspect is that stream analytics processes are usually built to be lighter than batch jobs. In other words, data flows derived from data streams to detected events are typically based on metrics and aggregates to optimize performance and costs. Some examples will illustrate the capabilities of dashboards, their use for data exploration and querying, and finally, their use for visualization over time of patterns in metrics and aggregates. Dashboards are used in all sorts of domains, both personal and business. They can be found in applications related to finance, health, security, climate control, travel, and beyond. They

are commonly found in business intelligence but also in network intrusions systems and business process management, as well as network and server monitoring. Finally, advanced applications, including smart cities and industrial use cases, are relying heavily on real-time dashboards, animation, and visualization.

7.8.2. Predictive Analytics

Real-time analytics are used increasingly frequently in organizations across a wide variety of domains to help detect new situations of interest and identify needed actions in anticipatory fashion, rather than merely responding after-the-fact and/or with only partial design. Predictive analytic techniques are of this kind, as they help answer the

Fig 7 . 3 : Real-Time Predictive Analytics: Demand Forecasting

question: "Given the current and predicted future state of the world based on real-time estimates from sensors and historical knowledge of past patterns and outcomes over time, how can various parameters be adjusted to optimize our measures of success?" Such analytics have been implemented increasingly in recent years for a number of business applications. For example, several companies have integrated predictive analytic capabilities into their apps, tools, and platforms for dynamic pricing in order to help organizations adjust their prices faster and more effectively. Predictive analytics are also used in connection with device sensor data in order to forecast equipment or parts failures and recommend needed maintenance before problems occur, in the domain of the Internet of Things. This is referred to as predictive maintenance.

Machine learning techniques are increasingly being applied to perform predictive analytics, either as stand-alone techniques or as part of an online machine learning system that integrates real-time data, typically either as part of an unsupervised or a semi-supervised learning framework. The employment of such techniques enables the more accurate prediction of variables that are strongly linked with other operationally significant metrics. These metrics could be the demand for airplane seats on specific flights, hotel rooms in a vacation area, or other perishable products or resources. In such implementations, the forecasts could be performed with different granularities of space and time and for different future time intervals, thereby helping organizations operate within the constraints imposed by such forecasts and optimization methods.

7.9. Conclusion

In this chapter, we have briefly reviewed foundational, emerging, and future concepts in data engineering, with emphasis on data storage and access management pipelines. We have illustrated concrete applications and example solutions, underlying the focus on data engineering for real-time processing and streaming analytics. In doing so, we have considered the context and communicated ideas in no technical jargon, for accessibility to business-oriented professionals. For professionals in engineering and development, we have included concepts that can be built upon for learning data engineering in practice, deepening technical skills through reading lists and exploration key paths.

Future trends in data engineering that we expect include pushing the movement of engineering closer to analytical tasks with low coding requirements for analysts, automating more of the engineering understands for robustness in production use, automated transfer to the cloud with appropriate trust and security controls, scanner-oriented stores for business ecosystems, and additional convergence on relational stores. Other areas of emphasis that we see for data engineering include intelligence-based automated monitoring and alerting for detection of pipeline failures, a greater emphasis on semantic data for wider application of business terminologies and ontologies, and greater implementation of human workflows around pipelines to avoid the uninteresting cycle of human decisions followed by automation. As a company, we also see software answer as a strong API and building block organization philosophy for assembling digital solutions traceable from their sources, easily integrating with external solutions and data sources, and being interpretable and understandable, built up from smaller pieces, for better ownership, maintenance, and evolution over time.

7.9.1. Summary and Future Trends in Data Engineering

Data engineering provides the foundations of essential infrastructure, methods, and components that enable the little-known secret jewel of data science: its volume. The diversity, heterogeneity, size, and velocity of data at scale is an important descriptor of the petabyte and exabyte era of data. Because data is so very transient, we cannot think of it as a series of piles that we warehouse and then ask for occasional reporting and decision support querying. Data is a living entity, transforming and streaming through the business, requiring a different set of principles to maximize value. This is the core tenet of data engineering for real-time processing and streaming analytics.

Real-time processing and streaming analytics tackle key data science challenges: latency, spike demand, and the derision of batch. Applications for real-time processing include targeted and personalized message delivery, advertising, fraud detection, and market monitoring. Streaming analytics allows us to continuously discover trends, patterns, and anomalies in data. This facilitates better and faster decision-making capabilities that are mission critical in a highly competitive landscape. The complexity of data varies greatly based on its provenance and dynamics. Data freshness is crucial for completing the mission critical tasks that both empower and embolden the business. The growth of the Internet of Things is increasing the scale and breadth of key performance indicators, encompassing both the enterprise and its customers at the edge, further demanding the new processes and patterns enabled by streaming analytics. Organizations will need to rethink their strategy with a focus on driving cost-effective and efficient ingestion pipelines and exploring old and new business questions for these transient data streams. Data will become ubiquitous, moving from systems-of-record to systems-of-engagement, in the never-ending journey to derive maximum business value.

References

Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., ... & Whittle, S. (2015). The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. Proceedings of the VLDB Endowment, 8(12), 1792–1803. https://doi.org/10.14778/2824032.2824076

Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink™: Stream and Batch Processing in a Single Engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 36(4), 28–38. https://doi.org/10.48550/arXiv.1501.00437

Gulisano, V., Jiménez-Peris, R., Patiño-Martínez, M., & Valduriez, P. (2012). StreamCloud: An Elastic and Scalable Data Streaming System. IEEE Transactions on Parallel and Distributed Systems, 23(12), 2351–2365. https://doi.org/10.1109/TPDS.2012.33

Stonebraker, M., & Çetintemel, U. (2005). "One Size Fits All": An Idea Whose Time Has Come and Gone. Proceedings of the 21st International Conference on Data Engineering, 2–11. https://doi.org/10.1109/ICDE.2005.1

Hesse, D., & Lorenz, M. (2019). Decision Intelligence: Transforming Analytics into Action. Applied Marketing Analytics, 5(2), 123–135. https://doi.org/10.2139/ssrn.3456131