O'REILLY®

Compliments of MEMSQL

Data Warehousing in the Age of Artificial Intelligence

Gary Orenstein, Conor Doherty,
Mike Boyarski & Eric Boutin

MEMSQL

ADAPT AND LEARN IN REAL TIME

The Real-Time Data Warehouse

www.memsql.com

Data Warehousing in the Age of Artificial Intelligence

Gary Orenstein, Conor Doherty, Mike Boyarski, and Eric Boutin

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Data Warehousing in the Age of Artificial Intelligence
by Gary Orenstein, Conor Doherty, Mike Boyarski, and Eric Boutin

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editor: Colleen Toporek
Production Editor: Justin Billing
Copyeditor: Octal Publishing, Inc.
Proofreader: Jasmine Kwityn

Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

August 2017: First Edition

Revision History for the First Edition
2017-08-22: First Release

978-1-491-99793-2

[LSI]

Table of Contents

CHAPTER 1

The Role of a Modern Data Warehouse in the Age of AI

Actors: Run Business, Collect Data

Applications might rule the world, but data gives them life. Nearly 7,000 new mobile applications are created every day, helping drive the world's data growth and thirst for more efficient analysis techniques like machine learning (ML) and artificial intelligence (AI). According to IDC,[1] AI spending will grow 55% over the next three years, reaching $47 billion by 2020.

Applications Producing Data

Application data is shaped by the interactions of users or actors, leaving fingerprints of insights that can be used to measure processes, identify new opportunities, or guide future decisions. Over time, each event, transaction, and log is collected into a corpus of data that represents the identity of the organization. The corpus is an organizational guide for operating procedures, and serves as the source for identifying optimizations or opportunities, resulting in saving money, making money, or managing risk.

1 For more information, see the Worldwide Semiannual Cognitive/Artificial Intelligence Systems Spending Guide.

Enterprise Applications

Most enterprise applications collect data in a structured format, embodied by the design of the application database schema. The schema is designed to efficiently deliver scalable, predictable transaction-processing performance. The transactional schema in a legacy database often limits the sophistication and performance of analytic queries. Actors have access to embedded views or reports of data within the application to support recurring or operational decisions. Traditionally, for sophisticated insights to discover trends, predict events, or identify risk requires extracting application data to dedicated data warehouses for deeper analysis. The dedicated data warehouse approach offers rich analytics without affecting the performance of the application. Although modern data processing technology has, to some degree and in certain cases, undone the strict separation between transactions and analytics, data analytics at scale requires an analytics-optimized database or data warehouse.

Operators: Analyze and Refine Operations

Actionable decisions derived from data can be the difference between a leading or lagging organization. But identifying the right metrics to drive a cost-saving initiative or identify a new sales territory requires the data processing expertise of a data scientist or analyst. For the purposes of this book, we will periodically use the term *operators* to refer to the data scientists and engineers who are responsible for developing, deploying, and refining predictive models.

Targeting the Appropriate Metric

The processing steps required of an operator to identify the appropriate performance metric typically requires a series of trial-and-error steps. The metric can be a distinct value or offer a range of values to support a potential event. The analysis process requires the same general set of steps, including data selection, data preparation, and statistical queries. For predicting events, a model is defined and scored for accuracy. The analysis process is performed offline, mitigating disruption to the business application, and offers an environment to test and sample. Several tools can simplify and automate the process, but the process remains the same. Also, advances in data-

base technology, algorithms, and hardware have accelerated the time required to identify accurate metrics.

Accelerating Predictions with ML

Even though operational measurements can optimize the performance of an organization, often the promise of predicting an outcome or identifying a new opportunity can be more valuable. Predictive metrics require training models to "learn" a process and gradually improve the accuracy of the metric. The ML process typically follows a workflow that roughly resembles the one shown in Figure 1-1.

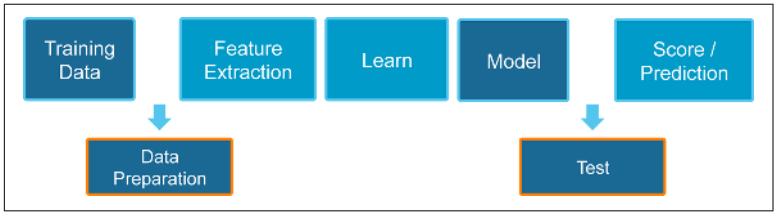

Figure 1-1. ML process model

The iterative process of predictive analytics requires operators to work offline, typically using a sandbox or datamart environment. For analytics that are used for long-term planning or strategy decisions, the traditional ML cycle is appropriate. However, for operational or real-time decisions that might take place several times a week or day, the use of predictive analytics has been difficult to implement. We can use the modern data warehouse technologies to inject live predictive scores in real time by using a connected process between actors and operators called a *machine learning feedback loop*.

The Modern Data Warehouse for an ML Feedback Loop

Using historical data and a predictive model to inform an application is not a new approach. A challenge of this approach involves ongoing training of the model to ensure that predictions remain accurate as the underlying data changes. Data science operators mitigate this with ongoing data extractions, sampling, and testing in order to keep models in production up to date. The offline process

can be time consuming. New approaches to accelerate this offline and manual process automate retraining and form an ML feedback loop. As database and hardware performance accelerate, model training and refinement can occur in parallel using the most recent live application data. This process is made possible with a modern data warehouse that reduces data movement between the application store and the analysis process. A modern data warehouse can support efficient query execution, along with delivering high-performance transactional functionality to keep the application and the analysis synchronized.

Dynamic Feedback Loop Between Actors and Operators

As application data flows into the database, subtle changes might occur, resulting in a discrepancy between the original model and the latest dataset. This change happens because the model was designed under conditions that might have existed several weeks, months, or even years before. As users and business processes evolve, the model requires retraining and updating. A dynamic feedback loop can orchestrate continuous model training and score refinement on live application data to ensure the analysis and the application remain up to date and accurate. An added advantage of an ML feedback loop is the ability to apply predictive models to previously difficult-to-predict events due to high data cardinality issues and resources required to develop a model.

Figure 1-2 describes an operational ML process that is supervised in context with an application.

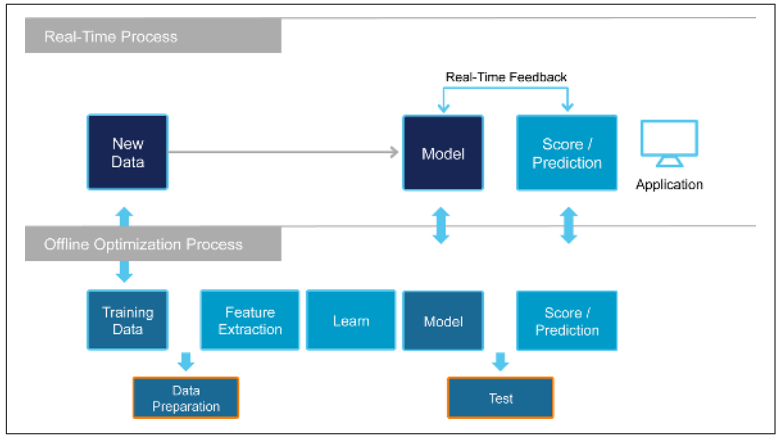

Figure 1-2. The operational ML process

Figure 1-3 shows the use of a modern data warehouse that is capable of driving live data directly to a model for immediate scoring for the application to consume. The ML feedback loop requires specific operational conditions, as we will discuss in more depth in Chapter 2. When the operational conditions are met, the feedback loop can continuously process new data for model training, scoring, and refinement, all in real time. The feedback loop delivers accurate predictions on changing data.

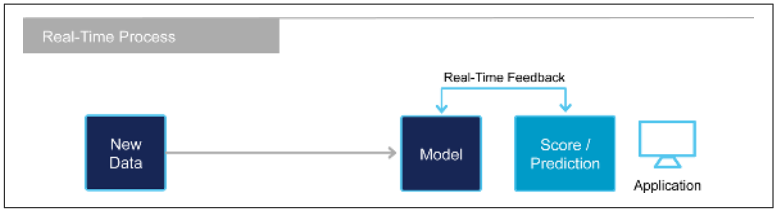

Figure 1-3. An ML feedback loop

CHAPTER 2

Framing Data Processing with ML and AI

A mix of applications and data science combined with a broad data corpus delivers powerful capabilities for a business to act on data. With a wide-open field of machine learning (ML) and artificial intelligence (AI), it helps to set the stage with a common taxonomy.

In this chapter, we explore foundational ML and AI concepts that are used throughout this book.

Foundations of ML and AI for Data Warehousing

The world has become enchanted with the resurgence in AI and ML to solve business problems. And all of these processes need places to store and process data.

The ML and AI renaissance is largely credited to a confluence of forces:

- The availability of new distributed processing techniques to crunch and store data, including Hadoop and Spark, as well as new distributed, relational datastores
- The proliferation of compute and storage resources, such as Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), and others

- The awareness and sharing of the latest algorithms, including everything from ML frameworks such as TensorFlow to vectorized queries

AI

For our purpose, we consider AI as a broad endeavor to mimic rational thought. Humans are masters of pattern recognition, possessing the ability to apply historical events with current situational awareness to make rapid, informed decisions. The same outcomes of data-driven decisions combined with live inputs are part of the push in modern AI.

ML

ML follows as an area of AI focused on applying computational techniques to data. Through exposure to sample data, machines can "learn" to recognize patterns that can be used to form predictions. In the early days of computing, data volumes were limited and compute was a precious resource. As such, human intuition weighed more heavily in the field of computer science. We know that this has changed dramatically in recent times.

Deep Learning

Today with a near endless supply of compute resources and data, businesses can go one step further with ML into *deep learning* (DL). DL uses more data, more compute, more automation, and less intuition in order to calculate potential patterns in data.

With voluminous amounts of text, images, speech, and, of course, structured data, DL can execute complex transformation functions as well as functions in combinations and layers, as illustrated in Figure 2-1.

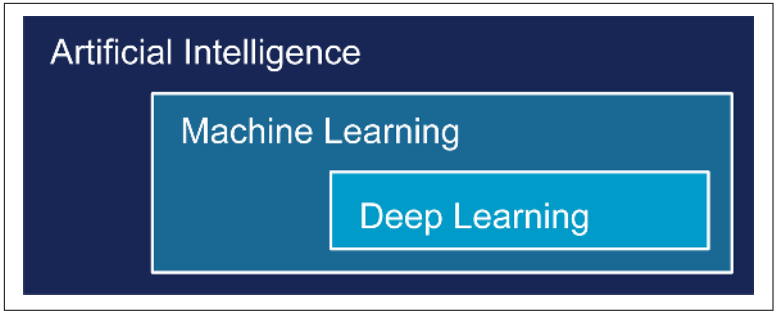

Figure 2-1. Common nesting of AI, ML, and DL

Practical Definitions of ML and Data Science

Statistics and data analysis are inherent to business in the sense that, outside of selling-water-in-hell situations, businesses that stay in business necessarily employ statistical data analysis. Capital inflow and outflow correlate with business decisions. You create value by analyzing the flows and using the analysis to improve future decisions. This is to say, in the broadest sense of the topic, there is nothing remarkable about businesses deriving value from data.

The Emergence of Professional Data Science

People began adding the term "science" more recently to refer to a broad set of techniques, tools, and practices that attempt to translate mathematical rigor into analytical results with known accuracy. There are several layers involved in the science, from cleaning and shaping data so that it can be analyzed, all the way to visually representing the results of data analysis.

Developing and Deploying Models

The distinction between development and deployment exists in any software that provides a live service. ML often introduces additional differences between the two environments because the tools a data scientist uses to develop a model tend to be fairly different from the tools powering the user-facing production system. For example, a data scientist might try out different techniques and tweak parameters using ML libraries in R or Python, but that might not be the implementation of the tool used in production, as is depicted in Figure 2-2.

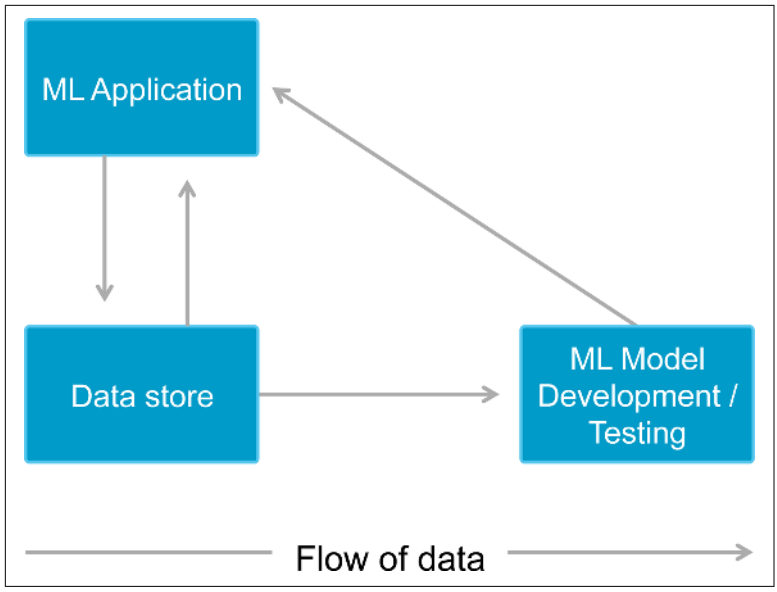

Figure 2-2. Simple development and deployment architecture

Along with professional data scientists, "Data Engineer" (or similarly titled positions) has shown up more and more on company websites in the "Now Hiring" section. These individuals work with data scientists to build and deploy production systems. Depending on the size of an organization and the way it defines roles, there might not be a strict division of labor between "data science" and "data engineering." However, there is a strict division between the development of models and deploying models as a part of live applications. After they're deployed, ML applications themselves begin to generate data that we can analyze and use to improve the models. This feedback loop between development and deployment dictates how quickly you can iterate while improving ML applications.

Automating Dynamic ML Systems

The logical extension of a tight development–deployment feedback loop is a system that improves itself. We can accomplish this in a variety of ways. One way is with "online" ML models that can update the model as new data becomes available without fully retraining the model. Another way is to automate offline retraining to be triggered by the passage of time or ingest of data, as illustrated in Figure 2-3.

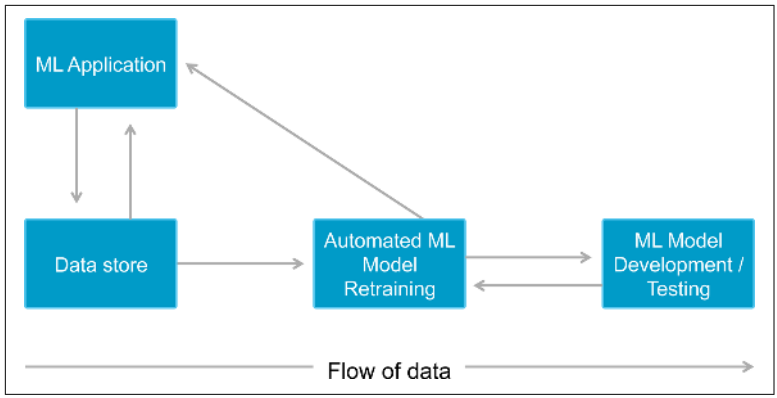

Figure 2-3. ML application with automatic retraining

Supervised ML

In supervised ML, training data is labeled. With every training record, features represent the observed measurements and they are labeled with categories in a classification model or values of an output space in a regression model, as demonstrated in Figure 2-4.

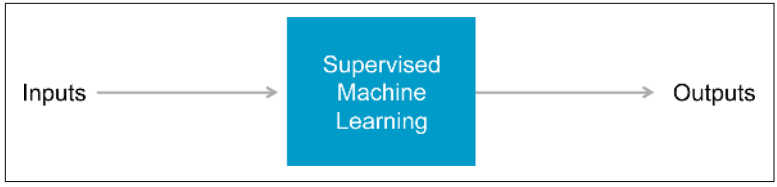

Figure 2-4. Basics of supervised ML

For example, a real estate housing assessment model would take features such as zip code, house size, number of bathrooms, and similar characteristics and then output a prediction on the house value. A regression model might deliver a range or likely range of the potential sale price. A classification model might determine whether the house is likely to sell at a price above or below the averages in its category (see Figure 2-5).

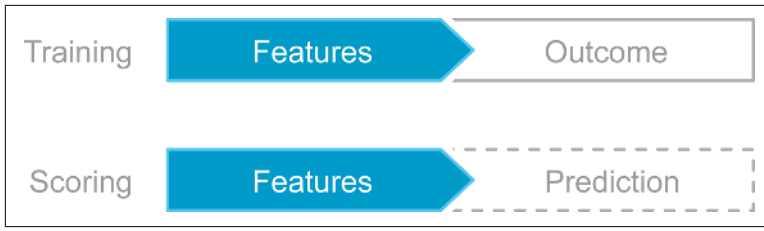

Figure 2-5. Training and scoring phases of supervised learning

A real-time use case might involve Internet of Things (IoT) sensor data from wind turbines. Each turbine would emit an electrical current that can be converted into a digital signal, which then could be analyzed and correlated with specific part failures. For example, one signal might indicate the likelihood of turbine failure, while another might indicate the likelihood of blade failure.

By gathering historical data, training it based on failures observed, and building a model, turbine operators can monitor and respond to sensor data in real time and save millions by avoiding equipment failures.

Regression

Regression models use supervised learning to output results in a continuous prediction space, as compared to classification models which output to a discrete space. The solution to a regression problem is the function that is the most accurate in identifying the relationship between features and outcomes.

In general, regression is a relatively simple way of building a model, and after the regression formula is identified, it consumes a fixed amount of compute power. DL, in contrast, can consume far larger compute resources to identify a pattern and potential outcome.

Classification

Classification models are similar to regression and can use common underlying techniques. The primary difference is that instead of a continuous output space, classification makes a prediction as to which category that record will fall. Binary classification is one example in which instead of predicting a value, the output could simply be "above average" or "below average."

Binary classifications are common in large part due to their similarity with regression techniques. Figure 2-6 presents an example of linear binary classification. There are also multiclass identifiers across more than two categories. One common example here is handwriting recognition to determine if a character is a letter, a number, or a symbol.

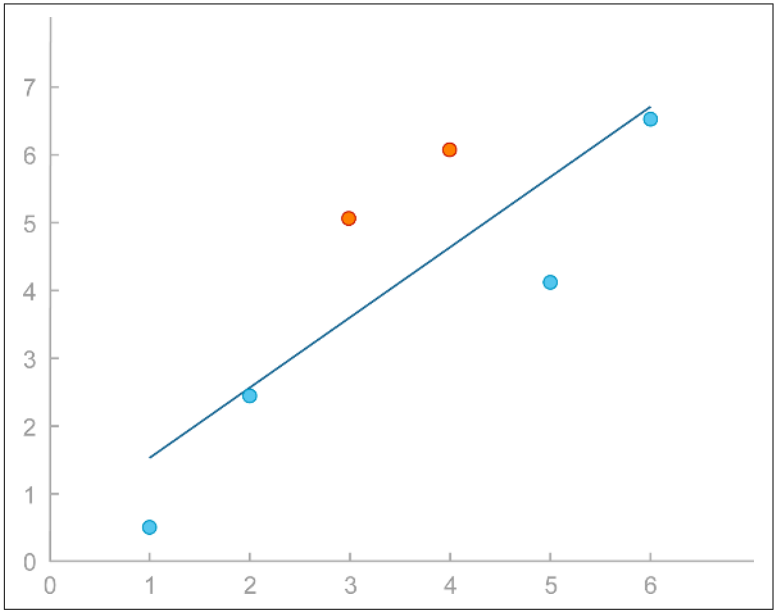

Figure 2-6. Linear binary classifier

Across all supervised learning techniques, one aspect to keep in mind is the consumption of a known amount of compute resources to calculate a result. This is different from the unsupervised techniques, which we describe in the next section.

Unsupervised ML

With unsupervised learning, there are no predefined labels upon which to base a model. So data does not have outcomes, scores, or categories as with supervised ML training data.

The main goal of unsupervised ML is to discern patterns that were not known to exist. For example, one area is the identification of "clusters" that might be easy to compute but are difficult for an individual to recognize unaided (see Figure 2-7).

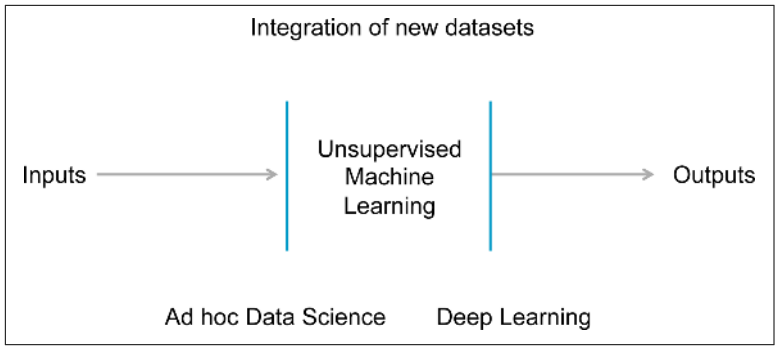

Figure 2-7. Basics of unsupervised ML

The number of clusters that exist and what they represent might be unknown; hence, the need for exploratory techniques to reach conclusions. In the context of business applications, these operations consume an unknown, and potentially uncapped, amount of compute resources putting them more into the data science category compared to operational applications.

Cluster Analysis

Cluster analysis programs detect data patterns when grouping data. In general, they measure closeness or proximity of points within a group. A common approach uses a centroid-based technique to identify clusters, wherein the clusters are defined to minimize distances from a central point, as shown in Figure 2-8.

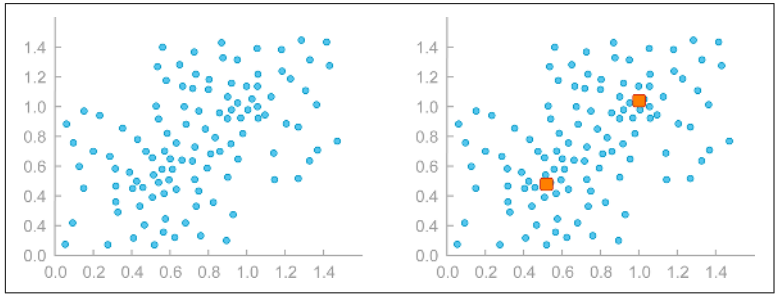

Figure 2-8. Sample clustering data with centroids determined by k-means

Online Learning

Another useful descriptor for some ML algorithms, a descriptor somewhat orthogonal to the first two, is *online learning*. An algorithm is "online" if the scoring function (predictor) can be updated as new data becomes available without a "full retrain" that would require passing over all of the original data. An online algorithm can be supervised or unsupervised, but online methods are more common in supervised learning.

Online learning is a particularly efficient way of implementing a real-time feedback loop that adjusts a model on the fly. It takes each new result—for example, "David bought a swimsuit"—and adjusts the model to make other swimsuits a more probable item to show users. Online training takes account of each new data point and adjusts the model accordingly. The results of the updated model are immediately available in the scoring environment. Over time, of course, the question becomes why not align these environments into a single system.

For businesses that operate on rapid cycles and fickle tastes, online learning adapts to changing preferences; for example, seasonal changes in retail apparel. They are quicker to adapt and less costly than out-of-band batch processing.

The Future of AI for Data Processing

For modern workloads, we have passed the monolithic and moved on to the distributed era. Looking beyond, we can see how ML and AI will affect data processing itself. We can explore these trends across database *S-curves*, as shown in Figure 2-9.

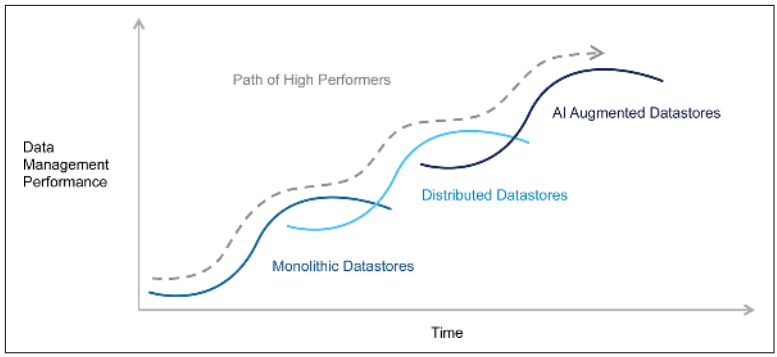

Figure 2-9. Datastore evolution S-curves

The Distributed Era

Distributed architectures use clusters of low-cost servers in concert to achieve scale and economic efficiencies not possible with monolithic systems. In the past 10 years, a range of distributed systems have emerged to power a new S-curve of business progress.

Examples of prominent technologies in the distributed era include, but are certainly not limited to, the following:

- Message queues like Apache Kafka and Amazon Web Services (AWS) Kinesis
- Transformation tiers like Apache Spark
- Orchestration systems like ZooKeeper and Kubernetes

More specifically, in the datastore arena, we have the following:

- Hadoop-inspired data lakes
- Key-value stores like Cassandra
- Relational datastores like MemSQL

Advantages of Distributed Datastores

Distributed datastores provide numerous advantages over monolithic systems, including the following:

Scale
: Aggregating servers together enables larger capacities than single node systems.

Performance
: The power of many far outpaces the power of one.

Alignment with CPU trends
: Although CPUs are gaining more cores, processing power per core has not grown nearly as much. Distributed systems are designed from the beginning to scale out to more CPUs and cores.

Numerous economic efficiencies also come into play with distributed datastores, including these:

No SANs
: Distributed systems can store data locally to make use of low-cost server resources.

No sharding
: Scaling monolithic systems requires attention to sharding. Distributed systems remove this need.

Deployment flexibility
: Well-designed distributed systems will run across bare metal, containers, virtual machines, and the cloud.

Common core team for numerous configurations
: With one type of distributed system, IT teams can configure a range of clusters for different capacities and performance requirements.

Industry standard servers
: Low-cost hardware or cloud instances provide ample resources for distributed systems. No appliances required.

Together these architectural and economic advantages mark the rationale for jumping the database S-curve.

The Future of AI Augmented Datastores

Beyond distributed datastores, the future includes more AI to streamline data management performance.

AI will appear in many ways, including the following:

Natural-language queries
: Examples include sophisticated queries expressed in business terminology using voice recognition.

Efficient data storage
: This will be done by identifying more logical patterns, compressing effectively, and creating indexes without requiring a trained database administrator.

New pattern recognition
: This will discern new trends in the data without the user having to specify a query.

Of course, AI will likely expand data management performance far beyond these examples, too. In fact, in a 2017 news release, Gartner predicted:

> More than 40 percent of data science tasks will be automated by 2020, resulting in increased productivity and broader usage of data and analytics by citizen data scientists.

CHAPTER 3

The Data Warehouse Has Changed

The Birth of the Data Warehouse

Decades ago, organizations used transactional databases to run analytics. This resulted in significant stress and hand-wringing by database administrators, who struggled to maintain performance of the application while providing worthwhile insights on the data. New techniques arose, including setting up preaggregated roll-ups or online analytical processing (OLAP) cubes during off-hours in order to accelerate report query performance. The approach was notoriously difficult to maintain and refreshed sporadically, either weekly or monthly, leaving business users in the dark on up-to-date analytics.

New Performance, Limited Flexibility

In the mid-1990s, the introduction of appliance-based data warehouse solutions (Figure 3-1) helped mitigate the performance issues for more up-to-date analytics while offloading the query load on transactional systems. These appliance solutions were optimized transactional databases using column store engines and specialized hardware. Several data warehouse solutions sprang up from Oracle, IBM Netezza, Microsoft, SAP, Teradata, and HP Vertica. However, over time, new challenges arose for appliance-based systems, such as the following:

Usability
: Each environment required specialty hardware and software services to set up, configure, and tune.

Cost
: Initial investments were high, along with adding capacity for new data sources or general growth. Adding new capacity often resulted in days or weeks of an outage to restructure and tune the environment.

Single-box scalability
: Query performance designed for a single-box configuration resulted in top-end limitations and costly data reshuffling for large data volumes or heavy user concurrency.

Batch ingestion
: Data ingestion was designed for nightly updates during off hours, affecting the analytics on the most recent data.

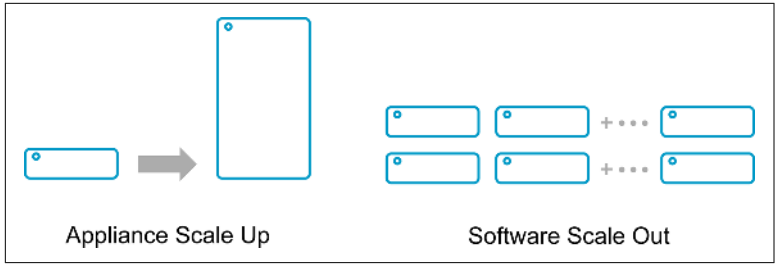

Figure 3-1. Database scale-up versus scale-out architecture

The Emergence of the Data Lake

Applications quickly evolved to collect large volumes and velocity of data driven by web and mobile technologies. These new web-scale applications tracked customer interactions, machine events, social interactions, and more. The appliance data warehouse was unable to keep up with this class of application, which resulted in new *data lake* technologies based on schema-less frameworks such as, Hadoop, HDFS, and NoSQL distributed storage systems. The benefit of these systems was the ability to store all of your data in one place.

Several analytic limitations occurred with data-lake solutions, including poor query performance and complexity for getting sophisticated insights out of an unstructured data environment.

Although SQL query layers were introduced to help simplify access to the data, the underlying data structure was not designed for fast query response to sophisticated analytic queries. It was designed to ingest a lot of variable data as quickly as possible utilizing commodity scale-out hardware.

A New Class of Data Warehousing

A new class of data warehouse has emerged to address the changes in data while simplifying the setup, management, and data accessibility. Most of these new data warehouses are cloud-only solutions designed to accelerate deployment and simplify manageability. It's based on previous generation engines, and takes advantage of columnstore table formats and industry-standard hardware.

Notable improvements to prior solutions include easy scalability for changing workloads, along with pay-as-you-go pricing. Additional innovations include the separation of storage from query compute to minimize data movement and optimization for machine utilization, as illustrated in Figure 3-2. Pricing is often tied to queries by rows scanned or the amount of time the query engine is available to the user. The new cloud data warehouses are designed for offline or ad hoc analytics for which sporadic use by a select group of analysts requires the spin up and down of system resources.

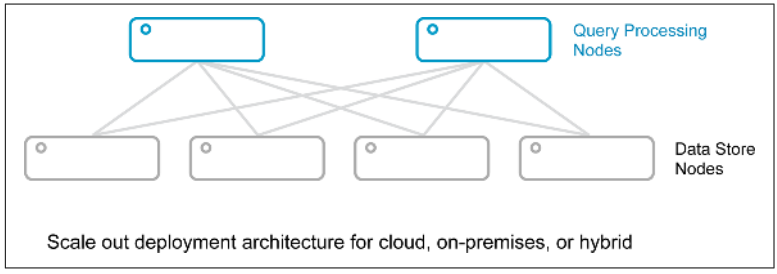

Figure 3-2. Modern distributed architecture for scalability

Notable limitations to the new class of cloud-only data warehouses are related to on-premises data analysis, optimizations for 24/7 operational analytics, and large-scale concurrency. Operational analytics can monitor and respond to a live business process requiring a continuous stream of data with subsecond query latency. Often the analysis is widely available across the enterprise or customer base,

placing additional stress on the data warehouse that has been designed for sporadic, ad hoc usage.

CHAPTER 4

The Path to the Cloud

There is no question that, whether public or private, cloud computing reigns as the new industry standard. This, of course, does not mean everything shifts overnight, but rather that data architects must ensure that their decisions fit with this path forward.

In this chapter, we take a look at the major shifts moving cloud architectures forward, and how you can best utilize them for data processing.

Cloud Is the New Datacenter

Today, cloud deployments have become the preferred method for new companies building data processing applications. The cloud has also become the dominant theme for traditional businesses as these organizations look to drive new applications and cost-optimize those already existing.

Cloud computing has essentially become the shortcut to having your own datacenter, albeit now with on-demand resources and a variety of built-in services.

Though early implementations of cloud computing came with some inherent differences compared to traditional datacenter architectures, those gaps are closing quickly.

Architectural Considerations for Cloud Computing

Understandably, cloud computing has a few architectural underpinnings different from traditional on-premises deployments. In particular, server persistence, scalability, and security need a new lens (Figure 4-1).

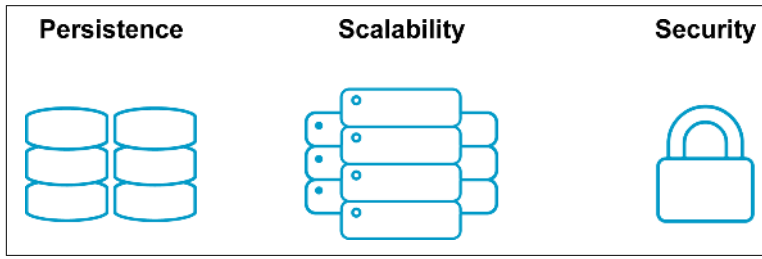

Figure 4-1. Architectural considerations for cloud computing

Persistence

Perhaps one of the most noticeable differences between traditional on-premises and cloud architectures is server or machine persistence. In the on-premises world, individual servers ran specific applications and architects worked diligently to ensure that each individual server and corresponding application had a high availability plan, typically implemented with redundancy.

In the cloud world, servers are much more ephemeral, and persistence is more often maintained outside of the server itself. For example, with the popular AWS offerings, the server might rely on storage options from S3 or Elastic Block Storage to maintain persistence. This approach understandably requires changes to conventional applications.

That said, it is and should be the new normal that, from an application perspective, cloud servers *are* persistent. That is, for the cloud to be successful, enterprises need the same reliability and availability from application servers that they saw in on-premises deployments.

Scalability

Conventional approaches also focused on scale-up computing models with even larger servers, each having a substantial compute and memory footprint. The cloud, however, represents the perfect platform to adopt distributed computing architectures, and this might be one of the most transformative aspects of the cloud.

Whereas traditional applications were often designed with a single server in mind, and an active–passive or active–active paired server for availability, new applications make use of distributed processing and frequently span tens to hundreds of servers.

Security

Across all aspects of computing, but in particular data processing, security plays a pivotal role. Today cloud architectures provide robust security mechanisms, but often with specific implementations dedicated to specific clouds or services within a designated cloud.

This dedicated security model for a single cloud or service can be challenging for companies that want to maintain multicloud architectures (something we discuss in more detail in Chapter 9).

Moving to the Cloud

Given cloud ubiquity, it is only a matter of time before more and more applications are cloud-based. Although every company has its own reasons for going to the cloud, the dominant themes revolve around cost-optimization and revenue creation, as illustrated in Figure 4-2.

Cost Optimization	Revenue Creation
Startup	Rapid prototyping
Maintenance	Temporary Deployments
Perpetual Billing	New Applications

Figure 4-2. Economic considerations for moving to the cloud

Cost Optimization

Building and operating a datacenter involves large capital expenditures that, when coupled with software and maintenance costs, force companies to explore cost-savings options.

Startup costs

Startup costs for cloud architectures can be low, given that you do not need to make an upfront investment, outside of scoping and planning.

Maintenance cost

Because many cloud offerings are "maintained" by the cloud providers, users simply consume the service without worrying about ongoing maintenance costs.

Perpetual billing costs

This area needs attention because the cloud bills continuously. In fact, an entire group of companies and services has emerged to help businesses mitigate and control cloud computing costs. Companies headed to the cloud must consider billing models and design the appropriate governance procedures in advance.

Revenue Creation

Businesses further cloud deployments with expectations to drive new revenue streams, and benefit from the following approaches.

Rapid prototyping

With virtually unlimited resources available on demand, companies can rapidly set up new application prototypes without risking significant capital expenditures.

Temporary application deployments

For cases in which a large amount of computing power is needed temporarily, the cloud fills the gap. One early cloud success story showcased how *The New York Times* converted images of its archive using hundreds of machines for 36 hours. Likely without this on-demand capability, the solution would have been economically impractical. As Derek Gottfrid explains in a *Times* blog post:

> This all adds up to terabytes of data, in a less-than-web-friendly format. So, reusing the EC2/S3/Hadoop method I discussed back in November, I got to work writing a few lines of code. Using Amazon Web Services, Hadoop and our own code, we ingested 405,000 very large TIFF images, 3.3 million articles in SGML and 405,000 XML files, mapping articles to rectangular regions in the TIFF's. This

> data was converted to a more web-friendly 810,000 PNG images (thumbnails and full images) and 405,000 JavaScript files—all of it ready to be assembled into a TimesMachine. By leveraging the power of AWS and Hadoop, we were able to utilize hundreds of machines concurrently and process all the data in less than 36 hours.[1]

New applications

Nothing gets businesses moving on new applications quite like an opportunity for revenue. One emerging area these days is Customer 360, and being able to assemble a wide variety of data from disparate sources to create a unified picture of customer activity.

Because application data in these cases comes from a variety of web and mobile sources, the cloud is a perfect aggregation point to enable high-speed collection and analysis.

Choosing the Right Path to the Cloud

When considering the right choices for cloud, data processing infrastructure remains a critical enablement decision.

Today, many cloud choices are centered on only one cloud provider, meaning that after you begin to consume the offerings of that provider, you remain relatively siloed in one cloud, as depicted in Figure 4-3.

1 For more information, see "The New York Times Archives + Amazon Web Services = TimesMachine"

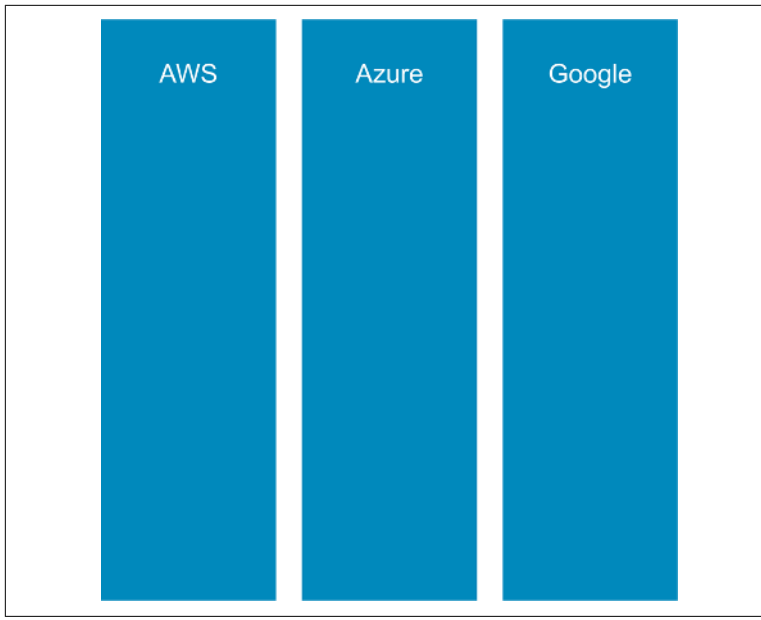

Figure 4-3. A single cloud provider approach

However, most companies are looking toward a hybrid cloud approach that covers not only public cloud providers but also enterprise datacenters and managed services, as shown in Figure 4-4.

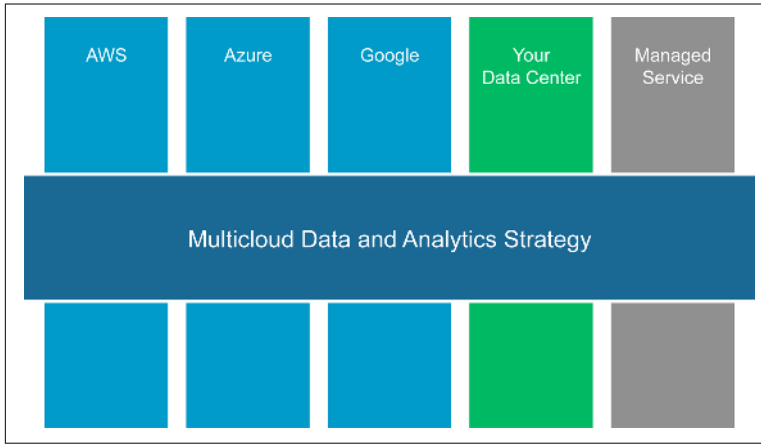

Figure 4-4. The multicloud approach

The multicloud approach for data and analytics focuses on solutions that can run anywhere; for example, in any public cloud, an enterprise datacenter, or a managed service. With this full spectrum of deployment options available, companies can take complete advantage of the cloud while retaining the flexibility and portability to move and adapt as needed.

CHAPTER 5

Historical Data

Building an effective real-time data processing and analytics platform requires that you first process and analyze your historical data. Ultimately, your goal should be to build a system that integrates real-time and historical data and makes both available for analytics. This is not the same as saying you should have only a single, monolithic datastore—for a sufficiently simple application, this might be possible, but not in general. Rather, your goal should be to provide an interface that makes both real-time and historical data accessible to applications and data scientists.

In a strict philosophical sense, all of your business's data is historical data; it represents events that happened in the past. In the context of your business operations, "real-time data" refers to the data that is sufficiently recent to where its insights can inform time-sensitive decisions. The time window that encompasses "sufficiently recent" varies across industries and applications. In digital advertising and ecommerce, the real-time window is determined by the time it takes the browser to load a web page, which is on the order of milliseconds up to around a second. Other applications, especially those monitoring physical systems such as natural resource extraction or shipping networks, can have larger real-time windows, possibly in the ballpark of seconds, minutes, or longer.

Business Intelligence on Historical Data

Business intelligence (BI) traditionally refers to analytics and visualizations on historical rather than real-time data. There is some delay

before data is loaded into the data warehouse and then loaded into the BI software's datastore, followed by reports being run. Among the challenges with this model is that multiple batched data transfer steps introduce significant latency. In addition, size might make it impractical to load the full dataset into a separate BI datastore.

Scalable BI

The size of datasets and the tools needed to process them has exceeded what can be done on a single machine. Even when operating on a dataset that can fit on a laptop hard drive, this might not be the most efficient way of accessing information. Transferring large amounts of data over a network adds latency. In addition, BI tools are generally designed to work with databases, and the BI datastores themselves might not be equipped to handle especially large datasets (Figure 5-1).

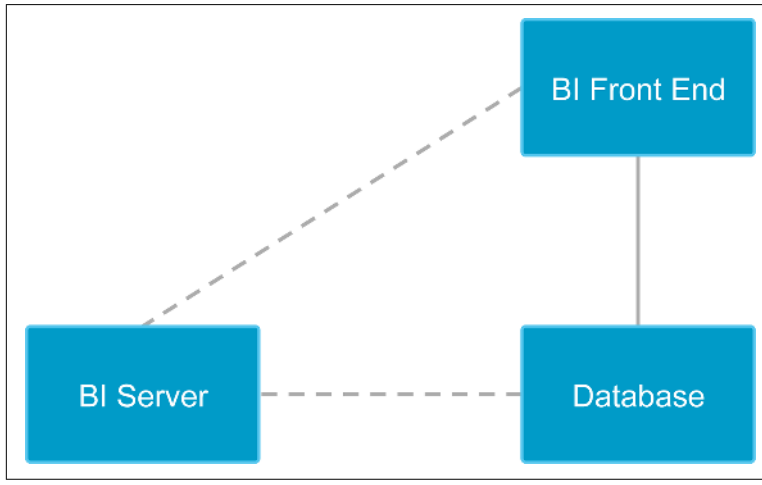

Figure 5-1. Typical BI architecture

Many modern BI tools employ a "thin" client, through which an analyst can run queries and generate diagrams and reports. Increasingly, these BI clients run in a web browser. The client is "thin" in the sense that it serves primarily as a user interface, and the user's queries "pass through" to a separate BI server or directly to a database.

Query Optimization for Distributed Data Warehouses

One of the core technologies in a distributed data warehouse is distributed query execution. How the database draws up and runs a query execution plan makes or breaks fast query response times. The plan is a sequence of suboperations that the database will go through in order to process a query as a whole and return a result.

All databases do some query planning, but it takes on much greater importance in a distributed system. The plan, for instance, determines which and how much data needs to be transferred between machines, which can be, and often is, the primary bottleneck in distributed query execution.

Example 5-1 shows a query optimization done by a distributed database. The sample query is based on one from a well-known database benchmark called TPC-H. After the initial query, which would be supplied by a user or an application, everything else happens within the database. Although this discussion is intended for a more technical audience, all readers are encouraged to at least skim the example to appreciate how much theory goes into distributed query optimization. If nothing else, this example should demonstrate the value of a database with a good optimizer and distributed query execution!

Example 5-1. Initial version of TPC-H query 17 (before query rewrite)

```
SELECT Sum(l_extendedprice) / 7.0 AS avg_yearly
FROM   lineitem,
       part
WHERE  p_partkey = l_partkey
       AND p_brand = 'Brand#43'
       AND p_container = 'LG PACK'
       AND l_quantity < (SELECT 0.2 * Avg(l_quantity)
                         FROM   lineitem
                         WHERE  l_partkey = p_partkey)
```

Example 5-1 demonstrates running the query on two tables, `part` and `lineitem`, that are partitioned along the columns `p_partkey` and `l_orderkey`, respectively.

This query computes the average annual revenue that would be lost if the company were to stop filling small orders of certain parts. The query is (arguably) written in a way that makes intuitive sense to an analyst: compute the sum of prices of parts from some brand, in

some bin, where the quantity sold is less than the result of a subquery. The subquery computes the average quantity of the part that is sold and then multiplies by .2.

This version of the query, although readable, likely will not execute efficiently without some manipulation. You may recall that `part` and `lineitem` are partitioned on `p_partkey` and `l_orderkey`, but the query is joining on `p_partkey = l_partkey`, where `l_partkey` is not a shard key. The subquery is "correlated" on `p_partkey`, meaning that the subquery does not actually select from `part`, and the `p_partkey` in the inner query refers to the same `p_partkey` from the outer query. Because the correlating condition of the inner query joins on a nonshard key, there are two (naive) options: first, you could trigger a remote query for every line of `part` that is scanned (very bad idea); or you could repartition `lineitem` on `l_partkey`. However, `lineitem` represents a large fact table, so repartitioning the entire table would be very expensive (time-consuming). Neither option is attractive.

Fortunately, there is an alternative. The optimizers of modern databases contain a component that rewrites SQL queries into other SQL queries that are logically equivalent, but easier for the database to execute in practice. Example 5-2 shows how a database might rewrite the query from Example 5-1.

Example 5-2. TPC-H query 17 rewritten

```
SELECT Sum(l_extendedprice) / 7.0 AS avg_yearly
FROM   lineitem,
       (SELECT 0.2 * Avg(l_quantity) AS s_avg,
               l_partkey             AS s_partkey
        FROM   lineitem,
               part
        WHERE  p_brand = 'Brand#43'
               AND p_container = 'LG PACK'
               AND p_partkey = l_partkey
        GROUP  BY l_partkey) sub
WHERE  s_partkey = l_partkey
       AND l_quantity < s_avg
```

This rewritten query is logically equivalent to the one shown in Example 5-1, which is to say that it can prove that the two queries will always return the same result regardless of the underlying data. The latter formulation rewrites what was originally a join between

two tables into a join between a table and a subquery, for which the outer query depends on the "inner" subquery. The advantage of this rewrite is that the inner query can execute first and return a small enough result that it can be "broadcast" across the cluster to the other nodes. The problem with the naive executions of the query from Example 5-1 is that it either requires moving too much data because `lineitem` is a fact table and would need to be repartitioned, or required executing too many subqueries, because the correlating condition did not match the sharding pattern. The formulation in Example 5-2 circumvents these problems because the subquery filters a smaller table (`part`) and then inexpensively joins it with a larger table (`lineitem`) by broadcasting the filtered results from the smaller table and only "seeking into" the larger table when necessary. In addition, the GROUP BY can be performed in parallel. The result of the subquery is itself small and can be broadcast to execute the join with `lineitem` in the outer query.

Example 5-3 explains the entire process more precisely.

Example 5-3. Distributed query execution plan for rewritten query

```
Project [s2 / 7.0 AS avg_yearly]
Aggregate [SUM(1) AS s2]
Gather partitions:all
Aggregate [SUM(lineitem_1.l_extendedprice) AS s1]
Filter [lineitem_1.l_quantity < s_avg]
NestedLoopJoin
|---IndexRangeScan lineitem AS lineitem_1,
| KEY (l_partkey) scan:[l_partkey = p_partkey]
Broadcast
HashGroupBy [AVG(l_quantity) AS s_avg]
groups:[l_partkey]
NestedLoopJoin |---IndexRangeScan lineitem,
| KEY (l_partkey) scan:[l_partkey = p_partkey]
Broadcast
Filter [p_container = 'LG PACK' AND p_brand = 'Brand#43']
TableScan part, PRIMARY KEY (p_partkey)
```

Again, there is no need to worry if you do not understand every step of this query rewriting and planning process. Distributed query optimization is an area of active research in computer science, and it is not reasonable to expect business analysts or even most software engineers to be experts in this area. One of the advantages of a distributed SQL database with a good optimizer and query planner is that users do not need to understand the finer points of distributed

query execution, but they can still reap the benefits of distributed computing.

Delivering Customer Analytics at Scale

One challenge with scaling a business is scaling your infrastructure without needing to become an "infrastructure company." Although certain opportunities or industries might provide reason to develop your own systems-level software, building scalable, customer-facing systems no longer requires developing your own infrastructure. For many businesses, building a scalable system depends more on choosing the right architecture and tools than on any limitations in existing technologies. The following section addresses several architectural design features that make it easier to scale your data processing infrastructure.

Scale-Out Architecture

As your analytics infrastructure grows more sophisticated, the amount of data that it both generates and captures will increase dramatically. For this reason, it is important to build around distributed systems with scale-out architectures.

Columnstore Query Execution

A columnstore is a class of database that employs a modern, analytics-friendly storage format that stores columns, or segments of columns, together on the storage medium. This allows for faster columns scans and aggregations. In addition, some databases will store metadata about the contents of column segments to help with even more efficient data scanning. For example, a columnstore database can store metadata like the minimum and maximum values in a column segment. Then, when computing a query with a WHERE condition involving values in that column, the database can use the column segment metadata to determine whether to "open up" a column segment to search for matching data inside, or whether to skip over the column segment entirely.

Intelligent Data Distribution

Although distributed computing can speed up data processing, a poorly designed system can negate or even invert the performance

benefits of having multiple machines. Parallelism adds CPUs, but many of those CPUs are on separate machines, connected by a network. Efficient distribution of data makes it possible to manage concurrent workloads and limit bottlenecks.

Examples of Analytics at the Largest Companies

Many of the largest companies by market capitalization, which not coincidentally happen to be technology companies, generate and capture massive volumes—in some cases petabytes per day—of data from customers who use their products and services.

Rise of Data Capture for Analytics

In a short span, entire industries have been born that didn't exist previously. Each of these areas is supported by one or more of the world's largest companies:

- App stores from Apple and Google
- Online music, video, and books: Apple, Google, and Amazon
- Seller marketplaces from Amazon.com
- Social networks from Facebook

Each of these services share a couple of common characteristics that drive their data processing workloads:

- Incredibly large end-user bases, numbering hundreds of millions
- A smaller (but still large) base of creators or sellers

These businesses provide analytics for a variety of audiences including content providers, advertisers, and end users, as well as many within the company itself, including product managers, business analysts, and executives. All of these characteristics culminate in a stack that starts with the platform provider, extends up to the creators or sellers, and ends with consumers. At each level, there is a unique analytics requirement.

App Store Example

Some of the aforementioned large, successful companies do significant business through digital commerce platforms known as *app stores*. We will use app stores as an example to explore analytics architectures across this type of stack. App stores are also an ideal example of new workloads that require a fresh approach to data engineering.

The largest app stores share the following characteristics:

- Hundreds of millions of end users
- Millions of application developers
- Dozens of app segments
- One primary platform provider (e.g., Apple, Google)

In addition, the backend data systems powering the largest app stores process many types of data:

- The distribution of apps to end users
- App data coming from each app from each end user
- Transactional data
- Log data

All of these different types of data from a platform with many users results in large and rapidly growing datasets. Managing and making sense of these large data volumes requires the following capabilities:

Low-latency query execution
: Even when the data is not "new," query execution latency determines the speed of analytics applications and dramatically affects the speed and productivity of data scientists.

High concurrency
: Data systems with many users can become bogged down if it does not manage concurrency well. As with query execution, concurrency can affect the performance of both applications and data scientists.

Fast data capture
: In the interest of unifying historical with real-time analytics, it is crucial that your database have good ingest speed. Even for

systems that do not yet incorporate real-time operations, fast data ingest saves both time and compute resources that can be dedicated to other analytics processes.

The final point, fast data capture, might seem unusual in the context of a discussion about historical data. However, the capability enables you to unify analytics on historical data with the real-time systems that power your business. Whether you subscribe to the metaphors of lakes or warehouses, building intelligent real-time applications requires rethinking the role historical data plays in shaping business decisions. Historical data itself might not be changing, but the applications, powered by models built using historical data, will create data that needs to be captured and analyzed. Moreover, new historical data is created every second your business operates. With these concerns in mind, we need another data architecture metaphor that conveys constant movement of data.

CHAPTER 6

Building Real-Time Data Pipelines

Technologies and Architecture to Enable Real-Time Data Pipelines

Today's applications thrive using both real-time and historical data for fast, accurate insights. Historical data provides the background for building an initial algorithm to score real-time data as it streams into an application.

To enable real-time machine learning (ML), you will need a modern data processing architecture that takes advantage of technologies available for ingestion, analysis, and visualization. Such topics are discussed with greater depth in the book *The Path to Predictive Analytics and Machine Learning* (O'Reilly, 2016); however, the overview provided in the sections that follow offers sufficient background to understand the concepts of the next several chapters.

Building any real-time application requires infrastructure and technologies that accommodate ultra-fast data capture and processing. These high-performance technologies share the following characteristics:

1. In-memory data storage for high-speed ingest
2. Distributed architecture for horizontal scalability
3. Queryable for instantaneous, interactive data exploration.

Figure 6-1 illustrates these characteristics.

Figure 6-1. Characteristics of real-time technologies

High-Throughput Messaging Systems

The initial input for most real-time applications begins by capturing data at its source and using a high-throughput message bus to ensure that every data point is recorded in its correct place. Data can come from a variety of sources, including logs, web events, sensor data, user interactions, financial market streams, and mobile applications.

High-throughput messaging technologies like Apache Kafka are ideal for consuming streaming data at its source. Apache Kafka can handle terabytes of messages, and its effective use of memory, combined with commit log on disk, provides ideal performance for real-time pipelines and durability in the event of server failure.

Data Transformation

The next piece of a real-time data pipeline is the transformation tier. Data transformation takes raw data from a message bus, and transforms the data to a more accessible format for easier analysis. Transformers serve a number of purposes, including data enrichment, aggregation, and scoring for ML.

Similar to our message bus, it's important that you use technologies that are fast and scalable to allow for real-time data processing. Apache Python and Apache Spark are used often for data transformation and are ideal for obtaining real-time performance.

When building real-time data pipelines, the job of the transformation is to extract data from the message bus (Apache Kafka), filter down to a smaller dataset, run enrichment operations, augment data, and then push that refined dataset to an operational data warehouse. You can create pipelines by using standard SQL queries. Example 6-1 describes a pipeline used to load Twitter messages for a sentiment analysis application.

Example 6-1. Creating a real-time data pipeline from Kafka

```
sql> CREATE PIPELINE twitter_pipeline AS
 -> LOAD DATA KAFKA "public-kafka.compute.com:9092/tweets-json"
 -> INTO TABLE tweets
 -> (id, tweet);
Query OK, (0.89 sec)
sql> START PIPELINE twitter_pipeline;
Query OK, (0.01 sec)
sql> SELECT text FROM tweets ORDER BY id DESC LIMIT 5\G
```

Technologies such as Apache Spark do not include a persistent datastore; this requires that you implement an operational datastore to manage that process, which we describe in the next step of our data pipeline.

Operational Datastore

To make use of both real-time and historical data for ML, data must be saved into a permanent datastore. Although you can use unstructured systems such as Hadoop Distributed File System (HDFS) or Amazon S3 for historical data persistence, neither offers the performance for delivering real-time analytics.

This is where an operational datastore comes into play. A memory-optimized data warehouse provides both persistence for real-time and historical data as well as the ability to query streaming data in a single system. By combining transactions and analytics, data is rapidly ingested from our transformation tier and saved to a datastore. This supplies your application with the most recent data available, which you can use to power real-time dashboards, predictive analytics, and ML applications.

Data Processing Requirements

For a database management system to meet the requirements for real-time ML, a variety of criteria must be met. The following sections explore some of these criteria.

Memory Optimization

For real-time performance, in-memory data processing is a must. Taking advantage of memory for compute allows reads and writes to occur at rapid pace. In-memory operations are also necessary for

real-time analysis because no purely disk-based system can deliver the input/output (I/O) required for real-time operations.

Access to Real-Time and Historical Data

Converging traditional online transaction processing (OLTP) and online analytical processing (OLAP) systems requires the ability to compare real-time data to statistical models and aggregations of historical data. To do so, a datastore must accommodate two types of workloads (without compromising on latency): high-throughput operational transactions and fast analytical queries.

Compiled Query Execution Plans

Reducing disk I/O allows queries to execute so rapidly that dynamic SQL interpretation can become a bottleneck. To address this, some datastores use a caching layer on top of their relational database management system (RDBMS). However, this leads to cache invalidation issues resulting in minimal, if any, performance benefit. Executing a query directly in memory, against the live dataset, maintains query performance with assured data consistency, as depicted in Figure 6-2.

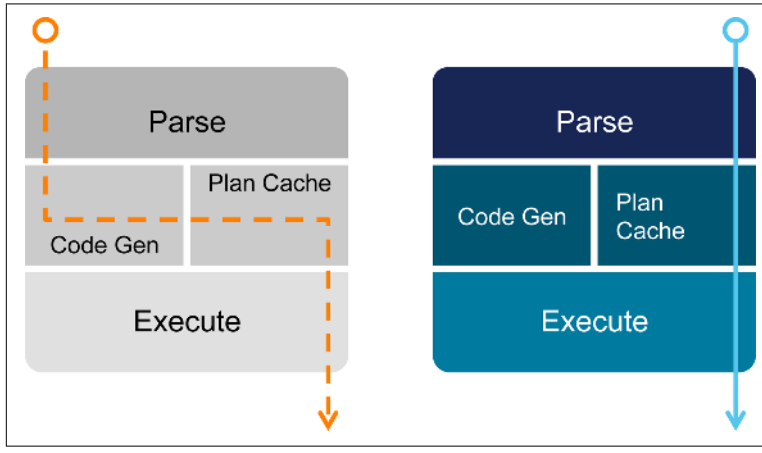

Figure 6-2. Compiled query execution plans

Multiversion Concurrency Control

You can achieve the high throughput necessary for a real-time ML engine through lock-free data structures and multiversion concurrency control (MVCC). MVCC enables data to be accessed simultaneously, avoiding locking on both reads and writes.

Fault Tolerance and ACID Compliance

Fault tolerance and Atomicity, Consistency, Isolation, and Durability (ACID) compliance are prerequisites for any operational data system, given that loss of data is unacceptable. A datastore should also support redundancy in the cluster and cross-datacenter replication for disaster recovery to ensure that data is never lost.

Benefits from Batch to Real-Time Learning

Historically, analytic computing and ML approaches have been constrained to batch processing to work around the limitations of Extract, Transfer, and Load (ETL) pipelines and traditional data warehouses. Such architectures restrict performance and limit the amount of data required for successful modeling.

Even though not every application requires real-time data, virtually every industry requires real-time solutions. For example, in real estate, transactions do not necessarily need to be logged to the millisecond. However, when every real estate transaction is logged to an operational datastore and a company wants to provide ad hoc access to that data, a real-time solution is likely required.

Here are a few benefits that real-time data pipelines offer:

Consistency with existing models
: By using existing models and bringing them into a real-time workflow, companies can maintain consistency of modeling.

Speed to production
: Using existing models means more rapid deployment and an existing knowledge base around those models.

Immediate familiarity with real-time streaming and analytics
: By not changing models, but changing the speed, companies can gain immediate familiarity with modern data pipelines.

Harness the power of distributed systems

Pipelines built with Kafka, Spark, and MemSQL harness the power of distributed systems and let companies benefit from the flexibility and performance of such systems. For example, companies can use readily available industry standard servers, or cloud instances to set up new data pipelines.

Cost savings

Most important, these real-time pipelines facilitate dramatic cost savings. In the case of energy companies drilling for reserves, they need to determine the health and efficiency of the drilling operation. Push a drill bit too far and it will break, costing millions to replace and lost time for the overall rig. Retire a drill bit too early and money is left on the table. Going to a real-time model lets companies make use of assets to their full extent without pushing too far to cause breakage or a disruption to rig operations.

CHAPTER 7

Combining Real Time with Machine Learning

Machine learning (ML) encompasses a broad class of techniques used for many purposes, and in general no two ML applications will look identical to each other. This is especially true for real-time applications, for which the application is shaped not only by the goal of the data analysis, but also by the time constraints that come with operating in a real-time window. Before delving into specific applications, it is important to understand the types of ML techniques and what they have to offer.

Real-Time ML Scenarios

We can divide real-time ML applications into two types, *continuous* or *categorical*, and the training can be supervised or unsupervised. This section provides some background on the various categories of applications.

Supervised and Unsupervised

Supervised and unsupervised ML are both useful techniques that you can use to solve ML problems (sometimes in combination with each other). You use supervised learning to train a model with *labeled* data, whereas you use unsupervised ML to train a model with *unlabeled* data. A scenario for which you could use supervised learning would be to train an ML model to predict the probability that a user would interact with a given advertisement on a web page.

You would train the model against labeled historical data which would contain a feature vector containing information about the user, the ad, and the context, as well as whether the user clicked the ad. You would then train the model to learn to correctly predict the outcome, given a feature vector. You can even train such a model in real time to fine-tune the model as new data is coming in.

Unsupervised ML, by contrast, doesn't use labeled data. Image or face recognition is a good example of unsupervised ML in action. A feature vector describing the object or the face is produced, and the model needs to identify which known face or known object is the best match. If no good match is found, the new object or new face can be added to the model in real time, and future objects can be matched against this one. This is an example of unsupervised ML because the objects being recognized are not labeled, the ML model decides by itself that an object is new and should be added, and it determines by itself how to recognize this object. This problem is closely related to *clustering*: given a large number of elements, group the elements together in a small number of classes. This way, knowledge learned about some elements of the group (e.g., shopping habits or other preferences), can be transferred to the other elements of the group.

Often, both methods are combined in a technique called *transfer learning*. Transfer learning is taking an ML model that's been trained using supervised ML, and using the model as a starting point to learn another model, using either unsupervised ML or supervised ML. Transfer learning is especially useful when using *deep neural networks*. You can train a neural network to recognize general objects or faces using labeled data (supervised learning). After the training is completed, you then can alter the neural network to be used to operate in unsupervised learning mode to recognize different types of objects that are being learned in real time.

Continuous and Categorical

The output of an ML model can either be continuous (e.g., predict the probability of a user clicking an ad on a web page), or categorical (e.g., identify an object in an image). You can use different model techniques for both types of applications.

Supervised Learning Techniques and Applications

The previous section provided an overview of various ML scenarios, and now we'll move on to cover supervised ML applications in particular.

Regression

Regression is a technique used to train an ML model using labeled data to predict a continuous variable, given an input vector. Depending on the complexity of the problem, you can use many regression techniques. You can use *linear regression* when the output is *linear* with the output. One such example would be to predict the time to transfer a file of a given size, or to predict the time an object will take to travel a certain distance, given a set of past observations.

Linear regression applications learn to model in the form $Y = AX + B$, where Y is the output, X is the input, and A and B are the coefficients. The data is usually noisy, and the typical algorithm used is least-square fitting—that is, it minimizes the sum of the square of the prediction error. Linear regression is very powerful because it can be learned fully *online*: the model can be adjusted in real time, as predictions are made. It works very well when there is a linear relationship between the input and the output, for example, in Figure 7-1, which looks at the relationship between the size of a file and the time it takes to transfer it. The chart represents various observations of file sizes and transfer times. The training data is noisy, but works very well with linear regression. You could use this type of application for online monitoring of a movie streaming application: if the transfer of the original buffer took longer than predicted based on historical data, the application could lower the quality of the video or potentially trigger a warning if the issue is widespread. The ability to update the ML model in real time is also very valuable in this type of application because it allows the model to slowly adjust itself to changes in the capacity of the infrastructure while quickly detecting anomalies.

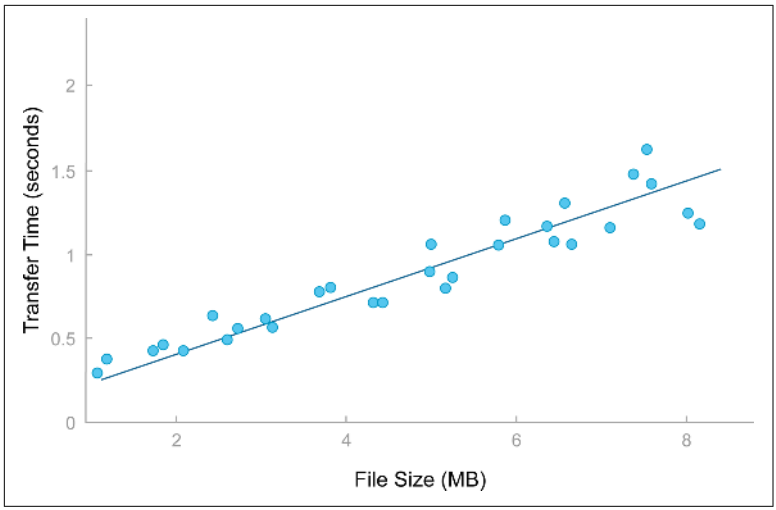

Figure 7-1. Detecting anomalies in the ML model

To train a linear regression ML model in SQL, you can use the following formula:

```
SELECT
covariance/variance as A,
mean_y - covariance/variance*mean_x AS B
FROM
(select sum(x*y)/count(*) - sum(x)/count(*)*sum(y)/count(*))
  as covariance,
(sum(x*x)/count(*) - sum(x)/count(*)*sum(x)/count(*))
  as variance,
sum(x)/count(*) as mean_x,
sum(y)/count(*) as mean_y from training_data) as t;
```

In this formula, `x` is the independent variable, and `y` is the variable we are trying to predict from `x`.

The statement outputs something similar to the following:

```
+--------------------+--------------------+
| A                  | B                  |
+--------------------+--------------------+
| 1.6000000000000012 | 0.6999999999999966 |
+--------------------+--------------------+
```

The values then can be used to predict another value of `y` from a value of `x`.

```
SELECT x, 1.6 * x + 0.69999
FROM data_to_predict;
```

It does not work well when the relationship is not linear between the inputs and the output. For example, the trajectory of airplanes after takeoff is nonlinear. The airplane accelerates, and the acceleration causes a nonlinear relationship between the input and the output variable (the prediction). For this type of model, you can use *Polynomial regression* to more accurately learn. Polynomial regression is a technique that learns the coefficients of a polynomial in the form $y = ax + bx^2 + cx^3 + \ldots$ using labeled data. Polynomial regression can capture more complex behavior than linear regression. When the behavior the model is trying to predict becomes very complex, you can use *regression trees*.

Regression trees are a class of decision trees that produces a continuous output variable. They work by separating the *domain*, or the total possible range of inputs, into smaller domains. The inputs to a regression tree can be a mix of categorical and continuous variables, which then are combined.

Categorical: Classification

It is very often useful to determine if a data point can be matched to a known category, or *class*. Applications are very wide and range from rate prediction to face recognition. You can use multiple techniques to build ML models producing categorical variables.

K-NN, or *K-Nearest-Neighbor*, is a technique by which data is classified by directly comparing it with the labeled data. When a new data point needs to be classified, the new data point is compared with every labeled data point. The category from the labeled data points closest to the new data point is used as the output. In a real-time ML application, the newly classified point can be added to the labeled data in order to refine the model in real time.

To classify a point ($x = 4$, $y = 2$) using K-NN on a dataset in a table in a SQL database, you can use the following query:

```
SELECT
x, y, category,
sqrt( (x - 4)*(x-4) + (y-2)*(y-2) ) as distance
FROM
training_data_2
ORDER BY distance ASC
LIMIT 3;
+------+------+----------+--------------------+
| x    | y    | category | distance           |
+------+------+----------+--------------------+
```

```
|    2 |    4 | C        | 2.8284271247461903 |
|    1 |    2 | C        |                  3 |
|    3 |    5 | A        | 3.1622776601683795 |
+------+------+----------+--------------------+
```

In this case, two of the closest points have the category `C`, so picking `C` as the category would be a good choice.

When the data points have a large number of dimensions, it is more efficient to encode the data as vectors and to use a database that supports vector arithmetic. In a well-implemented database, vector arithmetic will be implemented using modern single instruction, multiple data (SIMD) instructions, allowing much more efficient processing. In this case, the sample query would look like this:

```
CREATE TABLE training_data_3 (category VARCHAR(5) PRIMARY KEY,
  vector VARBINARY(4096));

INSERT INTO training_data_3 (category, vector) SELECT 'A',
    json_array_pack('[3, 5, ...]');

SELECT
category,
EUCLEDIAN_DISTANCE(vector, json_array_pack('[4, 5, 23, ...]')
    as distance
FROM
training_data_3
ORDER BY distance ASC
LIMIT 3;

+----------+--------------------+
| category | distance           |
+----------+--------------------+
| C        | 2.8284271247461903 |
| C        |                  3 |
| A        | 3.1622776601683795 |
+----------+--------------------+
```

Determining Whether a Data Point Belongs to a Class by Using Logistic Regression

Logistic regression is a very common technique for categorical classification when the output can take only two values, "0" or "1." It is commonly employed to predict whether an event will occur; for example, if a visitor to a web page will click an advertisement, or whether a piece of equipment is damaged, based on its sensor data. Logistic regression effectively learns a "threshold" in the input domain. The output value is 0 below the threshold and 1 above the

threshold. You can train logistics regression online, allowing the model to be refined as predictions are made and as feedback comes in. When you need multiple thresholds to identify the category (e.g., if the input is between the values A and B, then "0"; otherwise "1"), you can use more powerful models such as decision trees and neural networks.

In practice, logistic regression often is used with large feature vectors that are most effectively represented as vectors, or strings of numbers. In the following example, the training of a logistic regression learns a weight vector `w`, and an intercept `w_0`. The trained model can be stored in a table in a database to allow online classification using logistic regression:

```
SELECT
 1/(1 + exp(-w_0 + dot_product(w,
  json_array_pack('[0, 0.1, 0.8, ...]')) as logistic_regression
FROM weight;
```

Unsupervised Learning Applications

This section covers models and examples using unsupervised learning to solve ML problems.

Continuous: Real-Time Clustering

Clustering is the task of grouping a set of data points into a group (called a cluster) such that each group consists of data points that are similar. A *distance function* determines the similarity between data points.

You can use the centroid-based clustering (k-means) technique to learn the center position of each cluster, called a centroid. To classify a data point, you compare it to each centroid, and assign it to the most similar centroid. When a data point is matched to a centroid, the centroid moves slightly toward the new data point. You can apply this technique to continuously update the clusters in real time, which is useful because it allows the model to constantly be updated to reflect new data points.

An example application of real-time clustering is to provide customized content recommendation, such as news articles or videos. Users and their viewing history are grouped into clusters. Recommendations can be given to a user by inspecting which videos the cluster

watched that the user has not yet watched. Updating the clusters in real time allows those applications to react to changes in trends and provide up-to-date recommendations:

```
SELECT
cluster_name,
euclidean_distance(vector, json_array_pack('[1, 3, 5, ...]')
  as distance
FROM
clusters
ORDER BY distance ASC
LIMIT 1;
+--------------+--------------------+
| cluster_name | distance           |
+--------------+--------------------+
| cluster_1    | 2.8284271247461903 |
+--------------+--------------------+
```

Categorical: Real-Time Unsupervised Classification with Neural Networks

Neural networks are a very powerful tool to solve classification problems. They are composed of *neurons*, divided into layers. Typically, a neural network consists of an input layer, one or more *hidden* layers, and an output layer. Each layer uses the previous layer as an input. The final output layer would contain one neuron per possible category. The value of each output neuron determines whether an input belongs to a specific category. Image recognition is a good example of an application for neural networks. In those networks, the pixels of the image are fed in the neural network, and there is one output neuron for each object to be recognized. In the context of real-time ML, neural networks can be used to classify data points in real time. Neurons in the output layer can also be added in real time, in the context of unsupervised learning, to give the network the capacity to recognize an object it had not seen before, as illustrated in Figure 7-2.

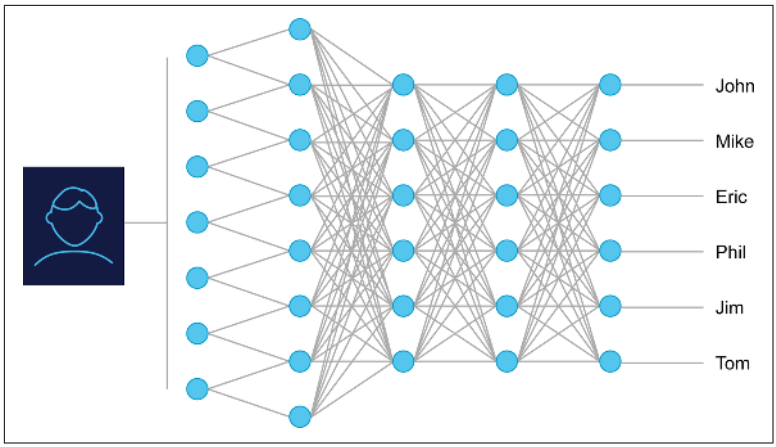

Figure 7-2. A neural network

The neurons of the output layer can be stored as vectors (one row/vector per output neuron) in a table. The following example illustrates how to evaluate the output layer of a neural network in a real-time database for facial recognition. We need to send a query to the database to determine which neurons in the output layer have a high output. The neurons with a high output determine which faces would match the input.

In the code example that follows, the output of the last hidden layer of the facial recognition neural network is passed in as a vector using *json_array_pack*:

```
SELECT
dot_product(neuron_vector,
  json_array_pack('[0, 0.4, 0.6 0,...]') as dot,
id
FROM
neurons
HAVING dot > 0.75;
```

You can use publicly available neural networks for facial recognition (one such example can be found at *http://www.robots.ox.ac.uk/~vgg/software/vgg_face/*).

Using a database to store the neurons provides the strong advantage that the model can be incrementally and transactionally updated concurrently with evaluation of the model. If no good match was found in the preceding example, the following code illustrates how

to insert a neuron in the output layer to recognize this pattern if it were to come again:

```
INSERT INTO neurons (id, neuron_vector)
SELECT 'descriptor for this face',
  json_array_pack('[0, 0.4, 0.6 0,...]');
```

If the algorithm was to see the face again, it would know it has seen it before. This example demonstrates the role that infrastructure plays in allowing ML applications to operate in real time. There are countless ways in which you might implement a neural network for facial recognition, or any of the ML techniques discussed in this section. The examples have focused on pushing computation to a database in order to take advantage of distributed computation, data locality, and the power of SQL. The next chapter discusses considerations for building your application and data analytics stack on top of a distributed database.

CHAPTER 8

Building the Ideal Stack for Machine Learning

Building an effective machine learning (ML) pipeline requires balance between two natural but competing impulses. On one hand, you want to use existing software systems and libraries in order to move quickly and maximize performance. However, most likely there is not a preexisting piece of software that provides the exact utility you require. The competing impulse, then, is to eschew existing software tools and to build your system from scratch. The challenge is that this option requires greater expertise, is more time-consuming, and can ultimately be more expensive.

This tension echoes the classic "build versus buy" debate, which comes up at some stage of every business venture. ML is unique in that both extremes of the spectrum can seem reasonable in the appropriate light. For example, there are plenty of powerful analytics solutions on the market and you can probably get something off the ground without writing too much code. Conversely, at the end of the day ML is just math, not magic, and there is nothing stopping you from cracking a linear algebra textbook and starting from scratch. However, in the context of nearly any other engineering problem, both extremes raise obvious red flags. For example, if you are building a bridge, you would be skeptical if someone tried to sell you a one-size-fits-all bridge in a box. Similarly, it probably does not bode well for any construction project if you also plan to source and manufacture all the raw materials.

ML pipelines are not bridges, but should not be treated as totally dissimilar. As businesses further realize the promises of "big data," ML pipelines take on a more central role as critical business infrastructure. As a result, ML pipelines demonstrate value not only by the accuracy of the models they employ, but the reliability, speed, and versatility of the system as a whole.

Example of an ML Data Pipeline

There is no limit to the number of different permutations of ML pipelines that a business can employ (not to mention different ways of rendering them graphically). This example focuses on a simple pipeline that employs a supervised learning model (classification or regression).

New Data and Historical Data

Figure 8-1 shows an ML pipeline applied to a real-time business problem. The top row of the diagram represents the operational component of the application, where the model is applied to automate real-time decision making. For instance, a user accesses a web page and the application must choose a targeted advertisement to display in the time it takes the page to load. When applied to real-time business problems, the operational component of the application always has restrictive latency requirement.

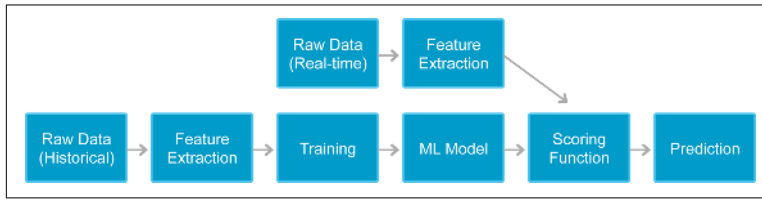

Figure 8-1. Data pipeline leading to real-time scoring

Model Training

The bottom row in Figure 8-1 represents the learning component of the application. It creates the model that the operational component applies to make decisions. Depending on the application, the training component might have less-stringent latency requirements because training is a one-time cost—scoring multiple data points does not require retraining. That being said, many applications can

benefit from reduced training latency by enabling more frequent retraining as the dynamics of the underlying system change.

Scoring in Production

When we talk about supervised learning, we need to distinguish the way we think about scoring in development versus in production. Scoring latency in a development environment is generally just an annoyance. In production, the reduction or elimination of latency is a source of competitive advantage. This means that, when considering techniques and algorithms for predictive analytics, you need to evaluate not only the bias and variance of the model, but how the model will actually be used for scoring in production.

Scoring with supervised learning models presents another opportunity to push computation to a real-time database (Figure 8-2).

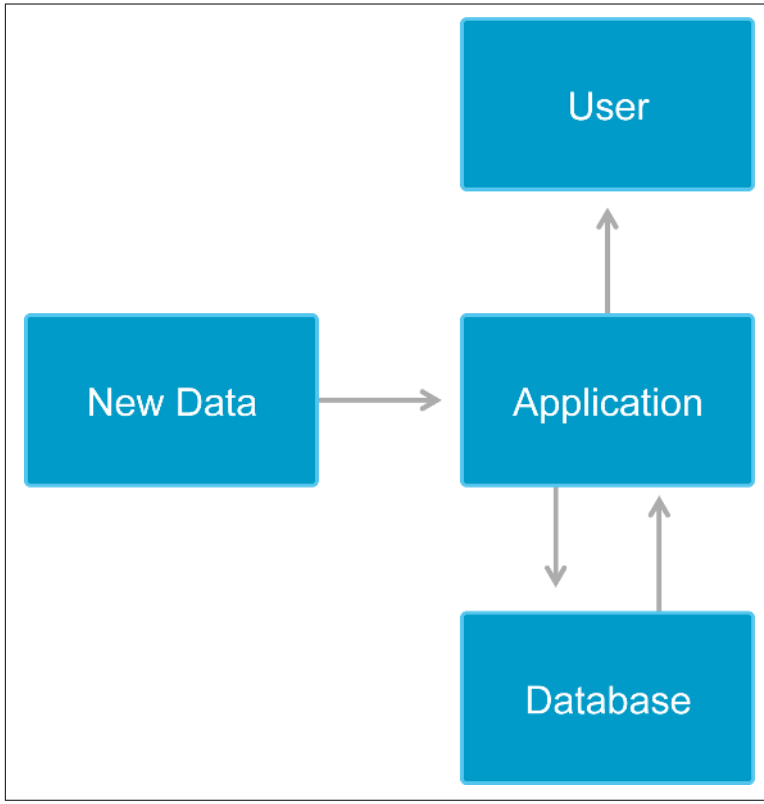

Figure 8-2. ML model expressed in SQL, computation pushed directly to database

Even though it can be tempting to perform scoring in the same interactive environment in which you developed the model, in most cases this will not satisfy the latency requirements of real-time use cases. Depending on the type of model, often the scoring function can be implemented in pure SQL. Then, instead of using separate systems for scoring and transaction processing, you do both in the database (Figure 8-3).

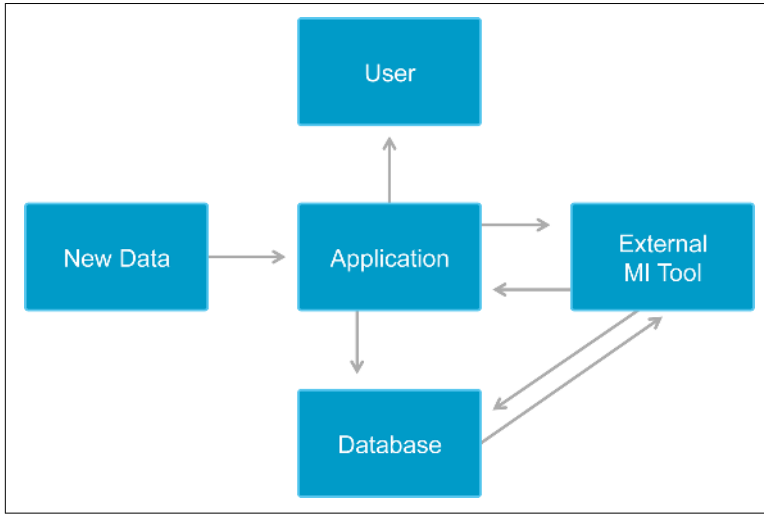

Figure 8-3. ML model expressed in an external tool, more data transfer and computation

Technologies That Power ML

The notion of a "stack" refers to the fact that there are generally several layers to any production software system. At the very least, there is an interface layer with which the users interact, and then there is the layer "below" that one that stores and manipulates the data that users access through the interface layer. In practice, an ML "stack" is not one dimensional. There is a production system that predicts, recommends, or detects anomalies, and makes this information available to end users, who might be customers or decision makers within an organization.

Adding data scientists, who build and modify the models that power the user-facing production system, adds another dimension to the "stack." Depending on your interpretation of the stack metaphor, the tools used by a data scientist on a given day likely does not fit any-

where on the spectrum from "frontend" to "backend," at least with respect to the user-facing production stack. However, there can, and often should, be overlap between the two toolchains.

Programming Stack: R, Matlab, Python, and Scala

When developing a model, data scientists generally work in a development environment tailored for statistics and ML like Python, R, Matlab, or Scala. These tools allow data scientists to train and test models all in one place and offer powerful abstractions for building models and making predictions while writing relatively little code. Every tool offers different features and performance characteristics suited to different tasks, but the good news is that your choice of development language likely will not make or break your project. Each of the tools discussed is extremely powerful and continually being improved. Figure 8-4 shows interactive data analysis in Python.

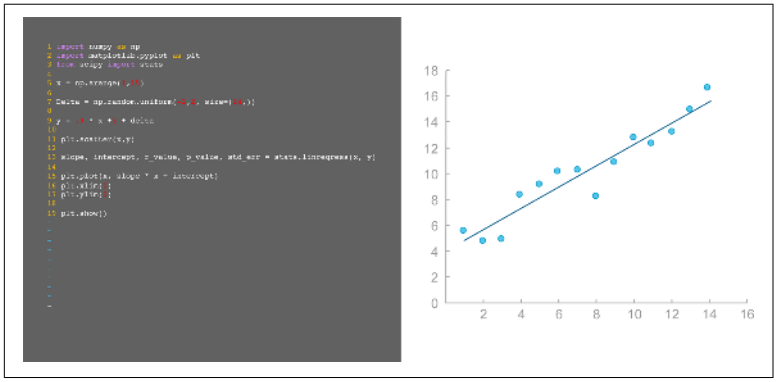

Figure 8-4. Interactive data analysis in Python

The programming languages and environments mentioned so far share (at least) one common design feature: interactivity. Each of these languages have Read-Evaluate-Print Loop (REPL) shell environments that allow the user to manipulate variables and functions interactively.

Beyond what they share, these languages fall into two general categories. R and Matlab (as well as SAS, Octave, and many others) are designed for numerical and statistical computation, modeling, data visualization, and ease of use for scientists and analysts who might not have a background in software engineering. The trade-off in

design choices is that these are not general-purpose languages, and you generally do not write user-facing applications in them.

Python and Scala are more general-purpose languages, commonly used to write user-facing applications, in addition to being used interactively for data science. They share further characteristics in common, including support for both functional and object-oriented programming. While Python and Scala are relatively "high-level" languages, in that they do not require the programmer to delve too deeply into tricky areas like memory management and concurrency, both languages offer sufficiently good performance for many types of services and applications.

The popularity of infrastructure like Apache Kafka and Apache Spark contributed significantly to Scala's growing prominence in recent years, whereas Python has been widely used at leading technology companies, including Google, for everything from web services to applications for managing distributed systems and datacenter resources.

Analytics Stack: Numpy/Scipy, TensorFlow, Theano, and MLlib

When building a new ML application, you should almost never start from scratch. If you wanted, you could spend a lifetime just building and optimizing linear algebra operators and completely lose sight of ML altogether, let alone any concrete business objective. For example, most ML libraries use at least some functionality from LAPACK, a numerical linear algebra library written in Fortran that has been used and continually improved since the 1970s.

The popular Python libraries Numpy and Scipy, for instance, both make calls out to LAPACK functions. In fact, these libraries make calls out to many external (non-Python) functions from libraries written in lower-level languages like Fortran or C. This is part of what make Python so versatile: in addition to offering an interpreter with good performance, it is relatively easy to call out to external libraries (and with relatively little overhead), so you get the usability of Python but the scientific computing performance of Fortran and C.

On the Scala side, adoption of Apache Spark has made the library MLlib widely used, as well. One of the biggest advantages of MLlib,

and Spark more generally, is that programs execute in a distributed environment. Where many ML libraries run on a single machine, MLlib was always designed to run in a distributed environment.

Visualization Tools: Business Intelligence, Graphing Libraries, and Custom Dashboards

If it seems to you like there is an "abundance" of tools, systems, and libraries for data processing, there also might appear to be an overabundance of data visualization options. Lots of data visualization software, especially traditional business intelligence (BI), comes packaged with its own internal tools for data storage and processing. The trend more recently, as discussed in Chapter 5, is toward "thinner" clients for visualization and pushing more computation to databases and other "backend" systems. Many of the most widely used BI tools now provide a web interface.

Widespread adoption of HTML5, with features like Canvas and SVG support, and powerful JavaScript engines like Google's V8 and Mozilla's SpiderMonkey have transformed the notion of what can and should run in the browser. Emscripten, an LLVM-to-JavaScript, enables you to translate, for instance, C and C++ programs into extremely optimized JavaScript programs. WebAssembly is a low-level bytecode that makes it possible to create more expressive programming and more efficient execution of JavaScript now, and whatever JavaScript becomes or is replaced by in the future.

There is an interesting, and not coincidental, parallel between the advances in web and data processing technologies: both have been changed dramatically by low-level languages that enable new, faster methods of code compilation that preserve or even improve the efficiency of the compiled program versus old methods. When your database writes and compiles distributed query execution plans, and you can crunch the final numbers with low-level, optimized bytecode in the browser, there is less need for additional steps between the browser and your analytics backend.

Top Considerations

As with any production software system, success in deploying ML models depends, over time, on know-how and familiarity with your own system. The following sections will cover specific areas to focus

on when designing and monitoring ML and data processing pipelines.

Ingest Performance

It might seem odd to mention ingest as a top consideration given that training ML models is frequently done offline. However, there are many reasons to care about ingest performance, and you want to avoid choosing a tool early on in your building process that prevents you from building a real-time system later on. For instance, Hadoop Distributed File System (HDFS) is great for cheaply storing enormous volumes of data, and it works just fine as a data source for a data scientist developing a model. That being said, it is not designed for executing queries and ingesting data simultaneously and would likely not be the ideal datastore for a real-time analytics platform. By starting out with the proper database or data warehouse, you prepare yourself to build a versatile platform that powers applications, serves data scientists, and can mix analytics with ingest and data processing workloads.

Analytics Performance

In addition to considering performance while ingesting data, you also should think about "pure" analytics performance, namely how quickly can the system execute a query or job. Analytics performance is a function of the system's ability to generate and optimize a plan, and then to execute that plan efficiently. A good execution plan will minimize the amount of data that needs to be scanned and transferred and might be the product of logical query "rewrites" that transform the user's query into a logically equivalent query that reduces computation or another resource.

In addition, databases designed for analytics performance implement a number of optimizations allowing them to speed up analytical queries. Modern databases implement a column-store engine, which lays out the data one column at a time rather than one row at a time. This has a number of benefits: first, it allows the data to be efficiently compressed by using the properties of the column. For example, if a VARCHAR(255) column only takes a few distinct values, the database will compress the column by using dictionary encoding and will only store the index of the value in the dictionary rather than the full value. This reduces the size of the data and reduces the input/output (I/O) bandwidth required to process data

in real time. Further, this data layout allows a well-implemented columnar engine to take advantage of modern single instruction, multiple data (SIMD) instructions sets. This enables the database to process multiple rows per clock cycles of the microprocessor, as demonstrated in Figure 8-5.

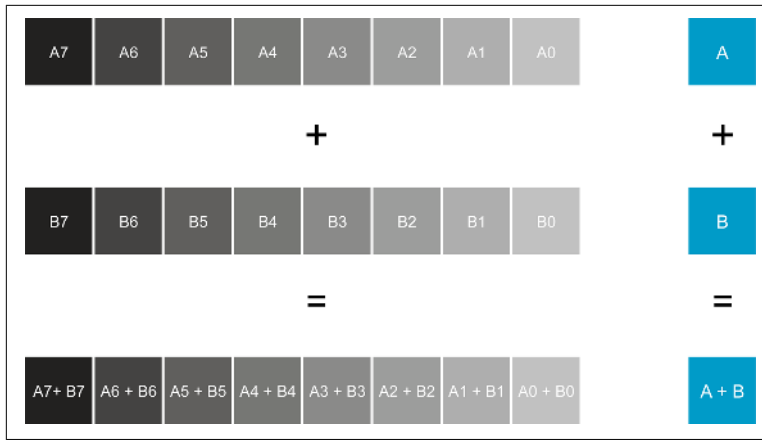

Figure 8-5. Scalar versus SIMD computation

The AVX2 instruction set, present on most modern Intel microprocessors, is used by modern databases to speed up the scanning of data, the filtering of data, and the aggregation of data. The following query is a good example of a query benefiting heavily from a well-optimized columnar engine running on a microprocessor supporting the AVX2 instruction set:

```
SELECT key, COUNT(*) as count FROM table t GROUP BY key;
```

When the number of distinct values of "key" is small, a modern data warehouse needs as little as two clock cycles for each record processed. As such, it is recommended to use microprocessors supporting those instructions sets in your real-time analytics stack.

Distributed Data Processing

This chapter's final topic has been touched on previously, but bears repeating. In many cases, distributed data processing is crucial to building real-time applications. Although many ML algorithms are not (or not easily) parallelizable, this is only a single step in your pipeline. Distributed message queues, databases, data warehouses, and many other data processing systems are now widely available

and, in many cases, the industry standard. If there is any chance that you will need to scale a particular step in a data pipeline, it is probably worth it to begin by using a distributed, fault-tolerant system for that step.

CHAPTER 9

Strategies for Ubiquitous Deployment

In today's market, enterprises need the flexibility to migrate, expand, and burst their storage and compute capacity both on-premises and in the cloud. As companies move to the cloud, the ability to span on-premises and cloud deployments remains a critical enterprise enabler. In this chapter, we take a look at hybrid cloud strategies.

Introduction to the Hybrid Cloud Model

Hybrid cloud generally refers to a combination of deployments across traditional on-premises infrastructure and public cloud infrastructure. Given the relatively new deployment options available, this can lead to a host of questions about more specific definitions. We will use the following broad terms to delineate.

Single Application Stack

In this model, different parts of the entire architecture will span across on-premises and cloud into a single application stack, as depicted in Figure 9-1. For example, the database might reside in a corporate datacenter, but the business intelligence (BI) layer might reside on the cloud. The BI application in the cloud will issue queries to the on-premises database, which will return results back to the cloud.

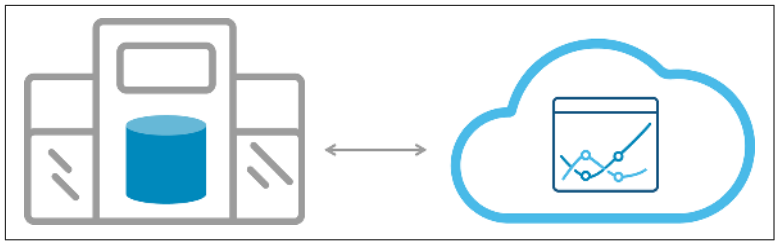

Figure 9-1. Single application stack hybrid deployment

Use Case-Centric

Figure 9-2 shows a use case-centric scenario in which different elements of the application might reside across on-premises and the cloud, depending on the need. Common examples are having the primary database on-premises and a backup or disaster recovery replica, located in the cloud. Other options in this model include test and development, which might reside in the cloud for quick spin up and adjustments, but later production deployments reside on-premises.

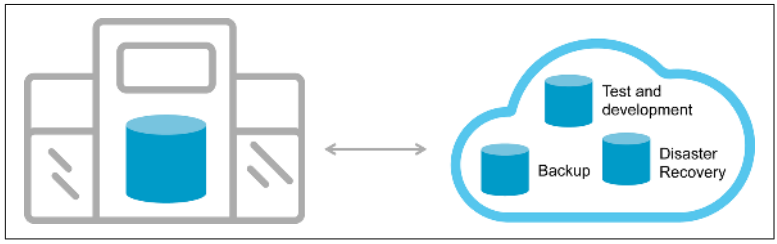

Figure 9-2. Use-case centric hybrid cloud deployment

Multicloud

The multicloud approach (Figure 9-3) takes the two prior deployment models and extends them to multiple clouds. This is likely the direction for a majority of the industry, because few companies will want to put all of their efforts into one cloud provider. With a combination of on-premises and multicloud flexibility, businesses have all the necessary tools at their disposal.

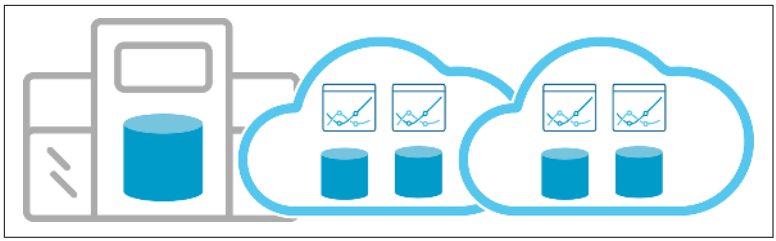

Figure 9-3. Multicloud deployment approach

On-Premises Flexibility

Many sensitive workloads require an on-premises deployment. When large amounts of data are produced locally, such as for manufacturing analytics, an on-premises model often makes sense.

However, on-premises deployments must remain compatible with cloud deployments, therefore requiring similar architectural foundations.

When considering an on-premises deployment, your solutions also need to be able to do the following:

- Run on any industry standard hardware, allowing architects to choose the CPU, memory, flash, and disk options that suit their needs.
- Run on any standard Linux platform for flexibility across environments in large organizations.
- Enable NUMA optimization, so scale-up, multicore servers can be put to full use.
- Use Advanced Vector Extensions (AVX) optimizations, to take advantage of the latest Intel and AMD features.
- Be virtual machine–ready, and be able to run with local or remote storage.
- Be container-ready, which is ideal for development and testing. Running stateful systems like databases within containers for production workloads is still an emerging area.
- Suited to using orchestration tools for smooth deployments.

Together these characteristics make datastore choices suitable for both on-premises and cloud.

Hybrid Cloud Deployments

Most new datastore deployments will need to span from on-premises to the cloud. Making use of hybrid architectures begins with the simplicity of the replicate database command. Specifically the ability to replicate a database from one cluster to another, preferably with a single command. In more flexible implementations, the two clusters do not need to be the same unit count. However, the receiving cluster must have enough capacity to absorb the data being replicated to it.

High Availability and Disaster Recovery in the Cloud

Robust datastores replicate data locally within a single cluster across multiple nodes for instant high availability, but additional layers of availability, including disaster recovery, can be deployed by replicating the database to the cloud.

Test and Development

Other cloud uses include test and development where data architects can rapidly deploy clusters to test a variety of workloads. Experiments can be run to determine the advantages of one instance type over another. Ultimately those deployments can migrate on-premises with much of the heavy lifting already accomplished.

Multicloud

The multicloud approach combines elements of on-premises and hybrid cloud, and extend those models to multiple public cloud providers. For example, you might use Amazon QuickSight, the Amazon Web Services BI layer, to query data in another database on Microsoft Azure or Google Compute Platform. Although this is just a hypothetical example, the prevailing direction is independence to use a variety of services and infrastructure layers across public cloud providers.

Today's cloud directions are dominated by on-premises-to-cloud discussions; multicloud is the next evolution as companies seek to prevent themselves from being stuck in one "silo."

When companies are able to move fluidly between cloud providers, they derive the following benefits:

- Pricing leverage
- No single source reliance
- An expanded potential footprint across more datacenters
- Better reach internationally

Multicloud also presents challenges when the intent is a single application stack. Automated deployments as well as security might differ between providers. Further, the incentives for cloud providers to work together on interoperability are not present, making technical challenges more difficult to solve.

Charting an On-Premises-to-Cloud Security Plan

One oft-cited reason against moving to the cloud is the lack of an appropriate security model. Understandably, companies want to have this model in place before moving to the cloud.

Common Security Requirements

Following are some of the common security-related issues that you must address:

- Encryption
- Audit logging
- Role-based access control
- Protection against insider threat

Encryption

Encryption can be implemented at different layers, but common solutions in the data processing world include offerings like Navencrypt or Vormetric.

Audit logging

Audit logging takes all database activity and writes the generated logs to an external location. Many logging levels are available, and each level provides limited or exhaustive information about user actions and database responses. This capability is useful for per-

forming common information security tasks such as auditing, investigating suspicious activity, and validating access control policies.

Role-based access control is essential to database security

Role-based access control (RBAC) provides a methodology for regulating an individual user's access to information systems or network resources based on their role in an organization. RBAC enables security teams to efficiently create, change, or discontinue roles as the unique needs of an organization evolve over time, without having to endure the hardship of updating the privileges of individual users.

Protecting against insider threat

Whether a large government agency or commercial company, protecting your data is critical to successful operations. A data breach can cause significant amounts of lost revenue, a tarnished brand, as well as diminished customer loyalty. For government agencies, the consequences can be more severe.

Here are three critical pillars to securing your data in a database:

- Separation of administrative duties
- Data confidentially, integrity, and availability
- 360-degree view of all database activity

Separation of administrative duties

The primary goal here is to disintermediate the database administrator (DBA) from the data. Central to this is to not allow a DBA to grant themself privileges without approval by a second administrator. There also should be special application-specific DBAs separate from operations and maintenance administrators. The developers and users should not be able to execute DBA tasks. This can all be done by setting up the following recommended roles.

At the organization level:

Compliance officer
- Manages all role permissions
- Most activity occurs at the beginning project stages
- Shared resource across the organization

Security officer
- Manages groups, users, passwords in the datastore
- Most activity occurs at the beginning project stages
- Shared resource across the organization

At the project level:

Datastore maintenance and operations DBA
- Minimal privileges to operate, maintain, and run the datastore
- Can be shared resource across projects

DBA per database application (application DBA)
- Database and schema owner
- Does not have permissions to view the data

Developer/user per database application
- Read and write data as permitted by the application DBA
- Does not have permission to modify database

After the roles and groups are established, you assign users to these groups. You can then set up row-level table access filtered by the user's identity to restrict content access at the row level. For example, a agency might want to restrict user access to data marked at higher government classification levels (e.g., Top Secret) than their clearance level allows.

Too frequently, data architects have had to compromise on security for select applications. With the proper architecture, you can achieve real-time performance and distributed SQL operations while maintaining the utmost in security controls.

CHAPTER 10

Real-Time Machine Learning Use Cases

Overview of Use Cases

Real-time data warehouses help companies take advantage of modern technology and are critical to the growth of machine learning (ML), big data, and artificial intelligence (AI). Companies looking to stay current need a data warehouse to support them.

Choosing the Correct Data Warehouse

If your company is looking to benefit from ML and needs data analytics in real time, choosing the correct data warehouse is critical to success. When deciding which data warehouse is best for your workload, here are a series of questions to ask yourself:

- Do you need to ingest data quickly?
- Do you need to run fast queries?
- Do you need to have concurrent workloads?

The use cases discussed in this chapter highlight how the combination of these needs lead to amazing business results.

Energy Sector

In this section, you will learn about two different companies in the energy sector and the work they're doing to drive equipment efficiency and pass along savings to customers.

Goal: Anomaly Detection for the Internet of Things

At a leading independent energy company during its drilling explorations, drill operators are constantly making decisions for where, when, and in what direction to adjust, with the goal of decreasing the time it takes to drill a well.

Approach: Real-Time Sensor Data to Manage Risk

Each drill bit on a drilling rig can cost millions of dollars, and the entire operation can cost millions per day, operating over the course of several months. As a result, there is a delicate balance between pushing the equipment far enough to get the full value from it, but not pushing it too hard and breaking it too soon. To better manage this operation, the company needed to combine multiple data types and third-party sources, including geospatial and weather data, into one solution, as illustrated in Figure 10-1.

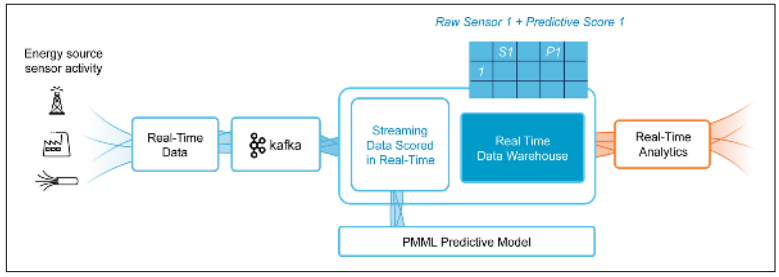

Figure 10-1. Sample real-time sensor pipeline in the energy sector

Goal: Take Control of Metering Equipment

Smart gas and electric meters produce huge volumes of data. A leading gas and electric utility needed to better manage meter readings and provide analytics to its data teams.

Approach: Use Predictive Analytics to Drive Efficiencies

The company had more than 200,000 meter readings per second load into its database while users simultaneously processed queries against that data. Additionally, it had millions of meters sending between 10 and 30 sensor readings every hour, leading to billions of rows of data. Just an initial part of a year contained more than 72 billion meter reads, which measured up to six terabytes of raw data.

To handle this amount of data, the company needed an affordable analytics platform to meet its goals and support data scientists using predictive analytics.

Implementation Outcomes

This leading independent energy production company was able to save millions of dollars based on high data ingest from its equipment, receiving about a half a million data points per second.

The gas and electric utility was able to compress its data by 10 times and reduce its storage on disk. The query time against the data dropped from 20 hours to 20 seconds.

Thorn

Thorn is a nonprofit that focuses on defending children from sexual exploitation on the internet.

Goal: Use Technology to Help End Child Sexual Exploitation

To achieve the goal of ending childhood sexual exploitation, Thorn realized it needed to create a digital defense to solve this growing problem.

Approach: ML Image Recognition to Identify Victims

Currently, roughly 100,000 ads of children are posted online daily. Matching these images to missing child photos is akin to finding a needle in a haystack. To comb through these images, Thorn implemented the use of ML and facial recognition technology.

The technology allowed the nonprofit to develop a point map of a given face, with thousands of points, and use those points to assign a

searchable number sequence to the face. This process lets the technology individually classify the image and match it to a larger database of images, as shown in Figure 10-2.

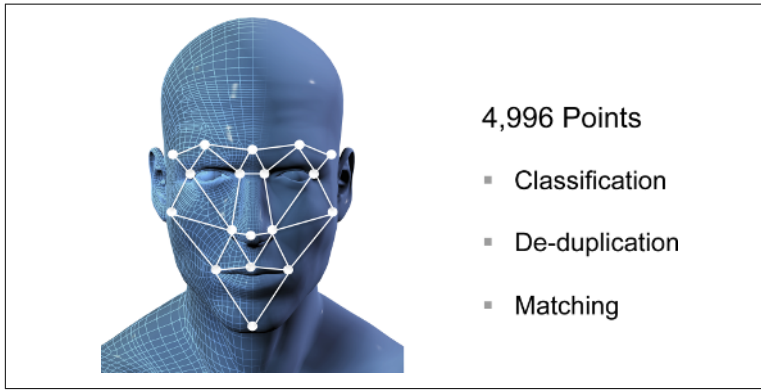

Figure 10-2. Facial-mapping diagram

To solve this problem, Thorn took advantage of the Dot Product function within its database as well as Intel's AVX2 SIMD instruction set to quickly analyze the points on a given face.

Implementation Outcomes

This process allowed Thorn to reduce the time it took to make a positive photo match from minutes down to milliseconds (see Figure 10-3).

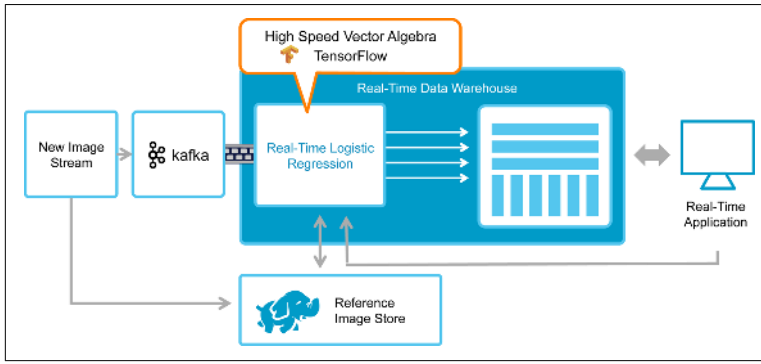

Figure 10-3. Reference architecture for building image recognition pipelines

Tapjoy

Tapjoy is a mobile advertising company that works with mobile applications to drive revenue.

Goal: Determine the Best Ads to Serve Based on Previous Behavior and Segmentation

Tapjoy helps address app economics, given that mobile application developers want to pay on installations of the applications, as opposed to advertising impressions. This is where Tapjoy comes in. It uses technology to deliver the best ad to the person using the application.

Approach: Real-Time Ad Optimization to Boost Revenue

Tapjoy wanted a solution that could combine many functions and eliminate the need for unnecessary processes. The original architecture would have required streaming from Kafka and Spark to update the application numbers—conversion rate, spending history, and total views. As the application grew, this process became more complicated with the addition of new features. One downside to this model was that streaming failed, which was problematic because the data needs to be real time and it's difficult to catch up when streaming stops working. To fix this problem, most companies would add a Lambda process. However, none of that architecture is easy to maintain or scale.

The team needed a better way. With the proper data warehouse, they were able to put raw data in, with the condition that the data could be rapidly ingested, and do the aggregation and serve it out, as depicted in Figure 10-4.

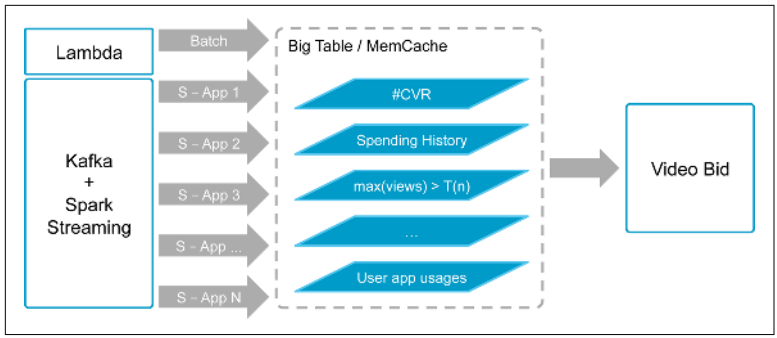

Figure 10-4. Reference architecture for building ad request pipelines

Implementation Outcomes

The new configuration allowed Tapjoy to run a query 217,000 times per minute on the device level, and achieve an average response time of .766 milliseconds. This is about two terabytes of data over 30 days. Additionally, the Tapjoy team learned that it could use a real-time data warehouse to support a variety of different tasks to reduce its tech stack footprint, which you can see in Figure 10-5.

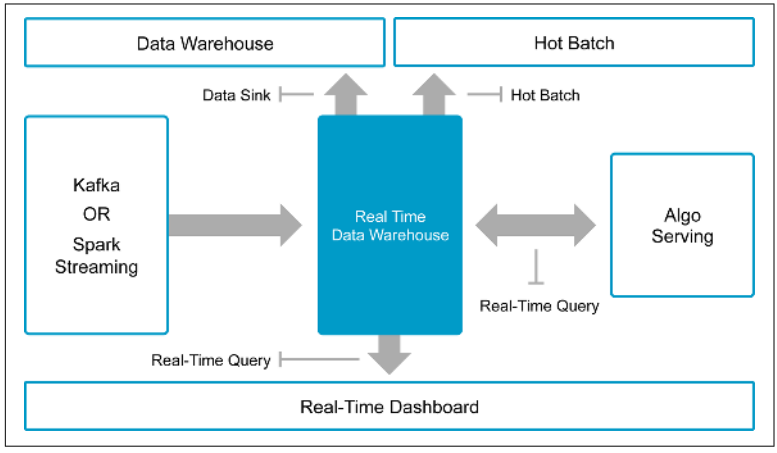

Figure 10-5. Example of uses for data warehouses

Reference Architecture

If you're looking to do real-time ML, Figure 10-6 gives a snapshot of what the architecture could look like.

First you begin with raw data. Then, you need to ingest that data into the data warehouse. You could use applications such as Kafka or Spark to transform the data. Or use Hadoop or Amazon S3 to handle your historical data.

After you've ingested the data, you can analyze it and manipulate it with tools such as Looker and Tableau. Figure 10-6 provides an overview of the environment.

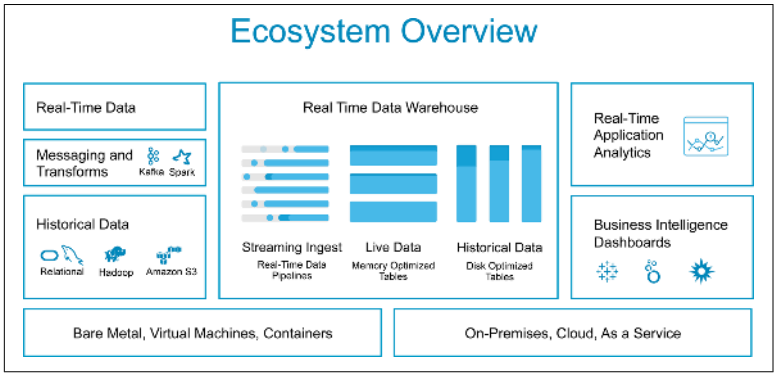

Figure 10-6. Reference architecture for data warehouse ecosystems

Datasets and Sample Queries

To begin transforming data, you can check out the following free datasets available online:

- Data.gov
- Amazon Web Services public datasets
- Facebook Graph
- Google Trends
- Pew Research Center
- World Health Organization

CHAPTER 11

The Future of Data Processing for Artificial Intelligence

As discussed in prior chapters, one major thread in designing machine learning (ML) pipelines and data processing systems more generally is pushing computation "down" whenever possible.

Data Warehouses Support More and More ML Primitives

Increasingly, databases provide the foundations to implement ML algorithms that run efficiently. As databases begin to support different data models and offer built-in operators and functions that are required for training and scoring, more machine learning computation can execute directly in the database.

Expressing More of ML Models in SQL, Pushing More Computation to the Database

As database management software grows more versatile and sophisticated, it has become feasible to push more and more ML computation to a database. As discussed in Chapter 7, modern databases already offer tools for efficiently storing and manipulating vectors. When data is stored in a database, there simply is no faster way of manipulating ML data than with single instruction, multiple data (SIMD) vector operations directly where the data resides. This elim-

inates data transfer and computation to change data types, and executes extremely efficiently using low-level vector operations.

When designing an analytics application, do not assume that all computation needs to happen directly in your application. Rather, think of your application as the top-level actor, delegating as much computation as possible to the database.

External ML Libraries/Frameworks Could Push Down Computation

When choosing analytics tools like ML libraries or business intelligence (BI) software, one thing to look for is the ability to "push down" parts of a query to the underlying database. A good example of this is a JOIN in a SQL query. Depending on the type of JOIN, the results returned might be significantly fewer than the total number of rows that were scanned to produce that result. By performing the join in the database, you can avoid transferring arbitrarily many rows that will just be "thrown out" when they don't match the join condition (see Figures 11-1 and 11-2).

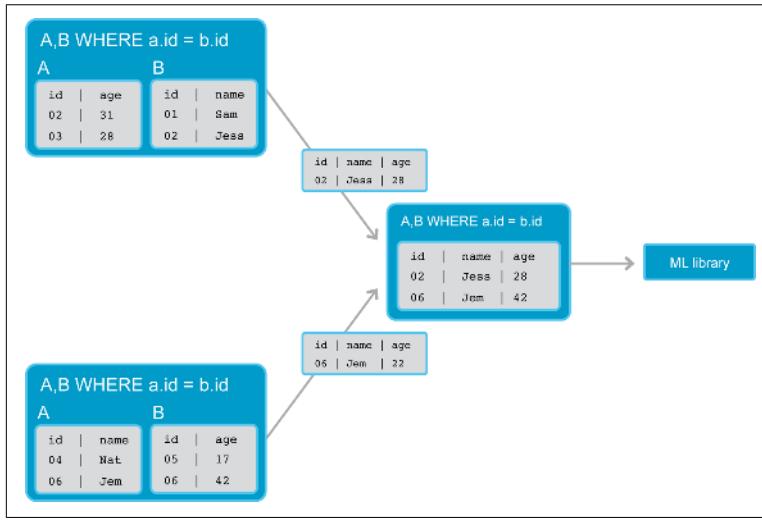

Figure 11-1. An ML library pushing JOIN to a distributed database

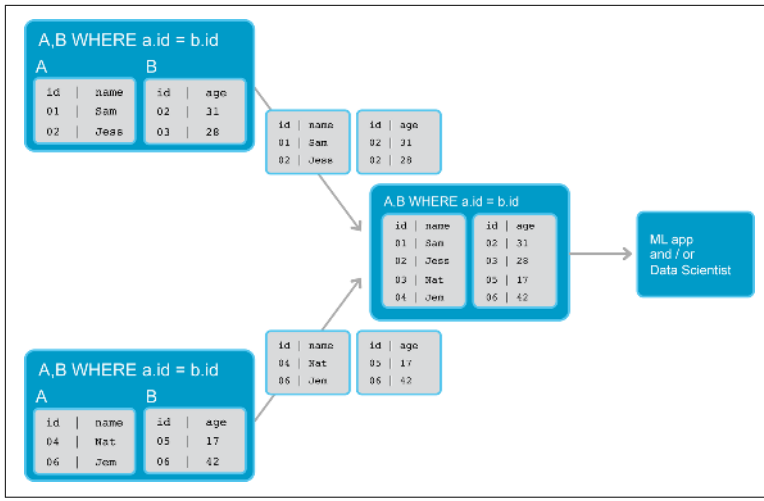

Figure 11-2. Efficiently combining data for a distributed join

ML in Distributed Systems

There are several ways in which distributed data processing can expedite ML. For one, some ML algorithms can be implemented to execute in parallel. Some ML libraries provide interfaces for distributed computing, and others do not. One of the benefits of expressing all or part of a model in SQL and using a distributed database is that you get parallelized execution for "free." For instance, if run on a modern, distributed, SQL database, the sample query in Chapter 7 that trains a linear regression model is naturally parallelized. A database with a good query optimizer will send queries out to the worker machines that compute the linear functions over the data on that machine, and send back the results of the computation rather than all of the underlying data. In effect, each worker machine trains the model on its own data, and each of these intermediate models is weighted and combined into a single, unified model.

Toward Intelligent, Dynamic ML Systems

Maximizing value from ML applications hinges not only on having good models, but on having a system in which the models can continuously be made better. The reason to employ data scientists is because there is no such thing as a self-contained and complete ML solution. In the same way that the work at a growing business is

never done, intelligent companies are always improving their analytics infrastructure.

The days of single-purpose infrastructure and narrowly defined organizational roles is over. In practice, most people working with data play many roles. In the real world, data scientists write software, software engineers administer systems, and systems administrators collect and analyze data (one might even claim that they do so "scientifically").

The need for cross-functionality put a premium on choosing tools that are powerful but familiar. Most data scientists and software engineers do not know how to optimize a query for execution in a distributed system, but they all know how to write a SQL query.

Along the same lines, collaboration between data scientists, engineers, and systems administrators will be smoothest when the data processing architecture as a whole is kept simple wherever possible, enabling ML models to go from development to production faster. Not only will models become more sophisticated and accurate, businesses will be able to extract more value from them with faster training and more frequent deployment. It is an exciting time for businesses that take advantage of the ML and distributed data processing technology that already exists, waiting to be harnessed.

About the Authors

Gary Orenstein is the Chief Marketing Officer at MemSQL and leads marketing strategy, product management, communications, and customer engagement. Prior to MemSQL, Gary was the Chief Marketing Officer at Fusion-io, and he also served as Senior Vice President of Products during the company's expansion to multiple product lines. Prior to Fusion-io, Gary worked at infrastructure companies across file systems, caching, and high-speed networking.

Conor Doherty is a technical marketing engineer at MemSQL, responsible for creating content around database innovation, analytics, and distributed systems. While Conor is most comfortable working on the command line, he occasionally takes time to write blog posts (and books) about databases and data processing.

Mike Boyarski is Senior Director of Product Marketing at MemSQL, responsible for writing and discussing topics such as analytics, data warehousing, and real-time decision making systems. Prior to MemSQL, Mike was Sr. Director of Product Marketing at TIBCO, where he was responsible for the analytics portfolio. Mike has held product management and marketing roles for databases and analytics at Oracle, Jaspersoft, and Ingres.

Eric Boutin is a Director of Engineering at MemSQL and leads distributed storage and transaction processing. Prior to MemSQL, Eric worked as a Senior Software Engineer at Microsoft where he worked on Cosmos, a distributed data processing framework, and designed novel resource management techniques to scale big data processing systems to upwards of 20,000 servers per cluster. He published influential academic papers on the subject. Eric is passionate about Machine Learning and has worked with Thorn to help them design scalable infrastructure that supports face matching in real time to fight child sexual exploitation.