Quiz_review_2

Xingyu Chen

March 16, 2022

The modern data stack - lecture 10

Data Warehouse - Curated analytical data
Data Mart - Subset of Data Warehouse
ETL - Methodical integration to/from data warehouse
Dimensional Model - Star-schema for data analysis
One-Big-Table - Simplified alternative for data analysis **Data Lake** - Uncurated data

The modern data stack - Database Features:

- MPP: can scale crazy
- Auto-scaling: dont have to statically purchase exactly the right amount, can do migrations
- Managed: no dbas, no sysadmins
- Pay as you go: no more waiting for a year fo a budget

Database Challenges - Snowflake, redshift, bigquery:

- Cost: annual costs are high
- Minimal Database performance-tuning features: small operation to improve

Replication features:

- Support both polling & transaction log reading
- Easy deployments
- Supports many SaaS vendors as inputs

Replication Challenges:

- Performance: easily breaks

- Availability: support sucks
- Big transactions
- Cost: can be a little expensive
- Tight Coupling on Data Models: violating their data encapsulation

Transform through SQL Layers:
From raw -> base -> staging -> intermediate -> dimensional model. Includes Quality-Control test framework

Is it the new modern and improved solution?

- Good at faster time to market and lower in labor costs;
- Bad at massive piles of SQL is a disaster.

So we should expect some non-SQL alternatives to emerge; we need to add custom code around it.

Where is it a poor fit:

- very low latency
- very high volume
- very high data quality requirements
- complex transforms required
- Lack of skilled analysts to own the modeling
- Plenty of time to do it right

Message Systems / Kafka - lecture 11

The need: the information needs easy ways to be reliably and quickly routed to multiple types of receivers. Seamless integration of information of producers and consumers.

Kakfa: persistent messaging, High throughput, Distributed, Multiple client support, real time.

Topic: Kafka maintains feeds of messages in categories.

Producers: Processes that publish messages to a Kafka topic. Logs

Consumers: Processes that subscribe to topics and process the feed of published messages. Consumer Group

Broker: Kafka is run as a cluster comprised of one or more servers.

Each partition is an **ordered, immutable sequence of messages** that is **continually appended to a commit log.**

The messages in the partitions are each assigned **a sequential id number called the offset** that uniquely identifies each message within the partition.

The Kafka cluster **retains all published messages—whether or not they have been consumed—for a configurable period of time.**

Kafka's **performance is effectively constant with respect to data size** so retaining lots of data is not a problem.

Messaging traditionally has two models:

- **queuing**: in a queue, a pool of consumers may read from a server and each message goes to one of them
- **publish-subscribe**: in publish-subscribe the message is broadcast to all consumers.

By having a notion of **parallelism—the partition—within the topics**, Kafka is able to provide both **ordering guarantees and load balancing** over a pool of consumer processes.

Messages sent by a producer to a particular topic partition will be **appended in the order they are sent.**

A consumer instance **sees messages in the order they are stored in the log.**

For a topic with replication factor N, we **will tolerate up to N-1 server failures without losing any messages committed** to the log.

Kafka Design Principles
The fundamental backbone of Kafka is message caching and storing it on the filesystem. In Kafka, data is immediately written to the OS kernel page. Caching and flushing of data to the disk is configurable.

Kafka uses a message set to group messages to allow lesser network overhead.

Unlike most of the messaging systems, where metadata of the consumed messages are kept at server level, in Kafka, the state of the consumed messages is maintained at consumer level. This also addresses issues such as:
- Losing messages due to failure
- Multiple deliveries of the same message

By default, consumers store the state in ZooKeeper, but Kafka also allows storing it within other storage systems used for Online Transaction Processing (OLTP) applications as well.

In Kafka, producers and consumers work on the traditional push-and-pull model, where producers push the message to a Kafka broker and consumers pull the message from the broker.

Message Compression
Kafka provides a message group compression feature. Here, data is compressed by the message producer using either GZIP or Snappy compression protocols and decompressed by the message consumer.

What is web3? - lecture 12

The internet of money. Computers can send value across a computer network. This could radically change anything in society that relies on value transfer. (banking, insurance, finance, art, jobs)

2009 - Bitcoin Whitepaper
2015 - Ethereum Whitepaper. Smart contract – Solidity; Ethereum checkpoint; Vyper smart contract; openzeppelin contracts
2016 - The DAO
2017 - ICO Boom
2018 - MakerDAO Launch
2020 – DeFi (Decentralized Finance) Defi summer
2021 - NFTs, DAOs (OpenSea)

Quadratic Funding: is the mathematically optimal way to fund public goods in a democratic community.

A **smart contract audit** is an extensive methodical examination and analysis of a smart contract's code that is used to interact with a cryptocurrency or blockchain.

Map Reduce - lecture 13

Motivation: bring the processing to the data: Map-Reduce is just a programming model that allows for parallelization. Map-Reduce frameworks do all the heavy lifting on running your code across clusters of servers and ensuring it completes (i.e. handling failure).

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a Map() procedure that performs filtering and sorting and a Reduce() procedure that performs a summary operation.

- Goal of map function is to produce key / value pairs
- Each map function is independent of others
- Allows them to be safely parallelized
- Data is co-located with processing
- Can be any arbitrarily complex function (not just selection)
- Map stage can be all you want to do (e.g. transformation)

Partitioning (aka 'Shuffling') allows reduce functions to run in parallel on different keys.

Combining reduces data before sending it across the network. The reduce function needs a special shape for this to work. Its output must match its input - **a combinable reducer**. And not all are combinable

As map-reduce calculations get more complex, it's useful to break them down into stages using a pipes-and-filters approach, with the the output of one stage serving as input to the next, like the pipelines in UNIX.

Decomposing this report into multiple map-reduce steps makes it easier to write. Like many transformation examples, once you've found a transformation framework that makes it easy to compose steps, it's easier to compose many small steps together than try to cram heaps of logic into a single step. Another advantage is that the intermediate output may be useful for different outputs too, so you can get some reuse. Reduce operations are particularly valuable to save since they often represent the heaviest amount of data access.

- Map-reduce is a batch pattern to allow computations to be parallelized over a cluster.
- The map task reads data from an aggregate and boils it down to relevant key-value pairs. Maps only read a single record at a time and can thus be parallelized and run on the node that stores the record.
- Reduce tasks take many values for a single key output from map tasks and summarize them into a single output.

- Reducers that have the same form for input and output can be combined into pipelines. This improves parallelism and reduces the amount of data to be transferred.
- Map-reduce operations can be composed into pipelines where the output of one reduce is the input to another operation's map.
- If the result of a map-reduce computation is widely used, it can be stored as a materialized view for reuse.
- Materialized views can be updated through incremental map-reduce operations that only compute changes to the view instead of recomputing everything from scratch.

Intro to Machine Learning - lecture 14

Machine Learning (ML) is the "the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead"

- Different from most algorithms, in which the steps are explicitly programmed
- ML algorithms "learn" from data
- ML is considered a (limited) form of AI

Common Applications

- Chatbots (Google Assistant, Siri, Alexa)
- Recommender Systems ("Customers Like you Bought This", Netflix Movie Recommendations)
- Forecasting ("You Should Buy This Stock", "You Should Not Buy This Cryptocurrency")
- Reinforcement Learning (AlphaGo, Self-Driving Cars)
- Image Recognition ("This Picture Contains a Cat")
- Automated Language Translation

Model: the algorithm itself (e.g. "Logistic Regression") or the object you get after training a model

Training vs. Inference:

- Training: a model is applying the learning algorithm of the model to a set of data.
- Inference: is applying a trained model to a new observation.

ML Evaluation: Measuring how well a model is performing

Overfitting: Inadvertently learning patterns that do not generalize

Features: The inputs to the model (akin to "independent variables" in linear regression)

Supervised Learning: Training a model to predict a label or a value, based on a labeled training set (Classification and Regression are most common types of Supervised Learning)

Unsupervised Learning: Training a model without labels (Clustering and Dimensionality Reduction are examples)

Reinforcement Learning: Training a model to maximize some "reward" by taking actions in some environment (AlphaGo is an example)

Linear Regression: not exactly ML, but shares some characteristics

Logistic Regression: like linear regression for classification

KNN Classifier: look for the k training examples that are closest to the new sample

Decision Trees (and Random Forest): greedily generate a tree (or many trees) that represent the patterns in the training data

K-Means Clustering: a simple, fast, scalable clustering algorithm

Artificial Neural Network: fits the data according to a network of artificial "neurons"

Deep Learning is the use of Artificial Neural Networks (ANNs) with many hidden layers

One Learning Algorithm Hypothesis: The theory that the brain uses the same neural learning architecture and algorithm to learn a variety of things

Deep learning neural nets perform well on **image recognition, speech recognition, text processing, language translation, sequence prediction, etc**.

How a Neural network works

1. Inputs enter first hidden layer "neurons", and **are transformed by activation function**, then passed to second hidden layer, and transformed again by activation function, then passed to output layer (like function composition)
2. **Cost function** calculated based on outputs
3. Parameters of **activation function** adjusted to reduce value of cost function. This is called **gradient descent**.
4. Process repeated until gradient reaches small enough tolerance (ie. cost function stabilizes)

Activation Function (one at each node of each hidden layer):

- **logistic function**: f(x) = 1/(1 + exp(-x))
- **ReLU (rectified linear units)**: approx. linear combo of logistic functions, prevents "problem of vanishing gradient" in gradient descent

Cost Function: The function that measures how "bad" the algorithm is with a certain set of parameters

Gradient Descent: a method that attempts to minimize the cost function using the gradient of the cost function with respect to the parameters

Training a neural network is about minimizing the cost function

REST - lecture 14

REST Stands for Representational State Transfer, which is an **architectural style** for web services. A service based on REST is called a RESTful service.

REST is primarily used to build Web services that are lightweight, maintainable, and scalable.

A set of constraints that predictably help service achieve benefits such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

Client: the service making the request. This might be a web or mobile client, or may be another server application.

Server: The service fulfilling the request and responsible for the resources. This might be a server you own, or this might be the Twitter API.

Stateless services are easier to maintain and scale, since the server doesn't have to keep track of N client's application state.

Cacheability: Each response should include whether the response is cacheable or not and for how long the response should be cached by the client

Layered system: If the server authenticates requests, stores data, and hosts APIs on different services, the client should be agnostic of those underlying concerns.

Uniform interface: A server should have a uniform way of interacting with all of its clients.

- Resources: The unit of data requested
- Representations: A grouping of information about the resource that the server exposes.

HTTP response format: The server returns , which contains the status of the request. This response code is generally the 3-digit HTTP status code. contains the metadata and settings about the response message. contains the representation if the request was successful. http://MyService/Persons/1 Protocol://ServiceName/ResourceType/ResourceID

Versioning will make your life easier

Consistency is key. It becomes easier for your clients to construct the URIs programmatically if they are aware of the resource hierarchy and the URI convention you follow.

Avoid verbs for your resource names until your resource is actually an operation or a process: http://MyService/FetchPerson/1

The basic purpose of **query parameters** is to provide parameters to an operation that needs the data items: http://MyService/Persons/1?format=compact

Most commonly a **POST** is used for creation and a **PUT** is used for an update

GraphQL:
Advantages:

- Send and receive less unused data over the network
- Simplify client transformations of data

- Get many resources in a single request
- Evolve API with less hassle / versioning

Disadvantages:

- Pushes complexity to the server
- Queries can be unexpectedly expensive
- Error handling can be more difficult