UCB W205 Asynchronous Curriculum Notes

Kevin R. Crook

Updated: 6/5/2017

These notes are my personal notes that I made regarding the asynchronous curriculum to use to refresh myself before live classes.

Please note that a lot of this should be considered my personal opinions, of which, you are free to agree or disagree.

If you feel I have some misleading or incorrect information here, please feel free to let me know so I can make it better.

I also emphasize some material more than others, which is totally subjective on my part to decide what I think is more important.

Unit 1 - Course Introduction and Architecture

Hammerbacher, J, (2009), Information platforms and the rise of the data scientist. In Beautiful data: the stories behind elegant data solutions. O'Reilly

A chapter in the book detailing the history of information platforms and how we got to the data scientist. Traces the roots of Data Science through the basic needs of a business (from the author's first job at 17), to Business Intelligence Systems, to Data Warehousing (couple of rounds), to Big Data, to Data Science.

Koister, J., (2015) Dimensions for characterizing analytics data processing solutions. White paper for DATASCI W205.

The author, Jari Koister, developed the curriculum for this course as you have seen him presenting in the asynchronous videos. This paper lays the foundation work for the reference architectures and the dimensions introduced in this unit and used throughout this course. It also helps to explain how everything we will be covering in this course fits together. I would strongly recommend that you revisit this paper several times during this course. As we learn detailed information, it's good to come back and revisit how it all works together.

Han, J., Kamber, M., & Pei, J. (2012). Data mining: Concepts and techniques (3rd ed.). Morgan Kaufman, Chapter 1, pp. 1–35.

Introduces Data Mining, what it is and how it fits into KDD – Knowledge Discovery of Data. Explains the types of data that can be mined, techniques to find patterns in data, what kinds of applications can benefit from data mining, and major issues with data mining.

Patil, D.J., & Mason, H., (2015) Data driven: Creating a data culture

This is an O'Reilly book about how to create a data driven organization. More about with "soft skills" than "hard skills" (technical skills). Good information to have as you may find yourself working in a Data Science group where part of your job may be to help the company transform itself into a data drive culture.

Data-Driven Organizations - DDOs

Make decisions on data not intuition. More precise with what they want to achieve. Measure and validate using data

Some industries have been data driven: airlines, GE, etc.

DDOs collect data, develop intuition for data, prose questions to answer, run experiments, make decisions, and draw insights

Reference Architecture

We use reference architectures because there are a variety of solutions available, most architectures are best suited to specific uses, understand considerations and tradeoffs for the architectures
Reference Model – provides a framework for comparing solutions

Dimensions: Data

Characteristics of data to consider: structure, size, sink latency, source latency, quality, completeness
Structure: structured, unstructured, semi structured
Size: S – MiB, M – GiB, L- TiB, XL - 100's TiB, XXL – PiB
Sink latency – velocity of data as it arrives
Source latency – how quickly data are reflected in the indexing layer
Quality – high, medium, low
Completeness – incomplete (missing data we cannot complete), semi-complete (missing data but we can complete it), complete

Dimensions: Processing

Characteristics of processing to consider: selectivity, query time, aggregation, processing time, join, precision
Selectivity – how much data you expect to process – high >20%, medium 20-80%, low - > 80%
Query time (execution time) – expected query response time –
short ms, medium < 30 s, long minutes
Aggregation – advanced (rollups, drill down, lattices, cuboids), medium (multiple dimensions), basic (simple counters)
Processing time – short < 1 hr, medium < 12 hr, long < 24 hr
Joins – advanced, basic, none
Precision – exact, approximate, lossy

Unit 2 - Dimensions and Scaling: Understanding Trade-Offs

Krishna, S., & Tse, E. (2013). *Hadoop platform as a service in the cloud.* Netflix blog post.

Description of what is commonly called the "Netflix Architecture." They use AWS Elastic MapReduce (EMR) with S3. Great explanation of how to effectively use S3 in conjunction with Hadoop instances on EMR using Elastic Block Storage (EBS). Data are stored in S3 as the system of record, loaded from S3 into MapReduce jobs which uses HDFS on EBS, and the results are written out to S3. Their "Genie" system was built in house to automate this.

Mars, N., & Warren, J. (2015). *Big data: Principles and best practices of scalable real-time data systems.* Manning. Sections 1.4–1.10 .

Describes the wish list of desired properties of a Big Data system: robustness, fault tolerance, low latency for reads and updates, scalability, generalization, extensibility, ad hoc queries, minimal maintenance, and debug ability. Explains the problems with the fully incremental architectures (sometimes called Kappa Architectures) including an explanation for the CAP Theorem (aka Brewer's Theorem) – Consistency / Availability / Partition Tolerance. Explain the Lambda Architecture as a solution to most of these problems. Describes the Lambda Architecture's three main components: the Speed Layer, the Batch Layer, and the Serving Layer. Ends with recent trends in computer technology.

Introduction: Data Size, Transfer

Develop an intuition for data size.
Examples of type of data and how to size using charts - Tweets, Credit Card transactions, emails, web logs
Performance and Scale Bottlenecks - data size, network, storage, CPU, RAM
IOPS – Input / Output Operations per Second
Storage Performance – IOPS, Data Transfer Rate
Network Performance – Bandwidth (bps max rate), Throughput (actual rate), Latency (delays)

Introduction: Processing

Processing – compute and/or assemble a result
Computational Complexity – Ordo (Big O Notation)
Constant f(1) – always takes same time
Linear f(n) – example: twice as much data takes twice as much processing
Logarithmic f(log n) – example: twice as much data takes longer but not twice as much
Note that Log in computer science is base 2 not base 10
Exponential f(2^poly(n)) – example: twice as much data might take ten times as much processing, thrice as much data might take a thousand times as much processing

Concepts: Data Scaling

Scale Up – bigger computer in terms of RAM, CPUs, cores, etc.
Scale Out – add more computers which are connected
CAP Theorem – **Brewer's Theorem** – Consistency – Availability – Partition Tolerance
Horizontal Partitioning – Data Sharding – divide rows among nodes
Vertical Partitioning – **Columnar Database** – store tables in column major order rather than row major order – helps when a table has a lot of columns and you only want to retrieve some of them.

Concepts: Scaling Processing

Use IOPS calculations to get a high level feel for what hardware we need and where bottlenecks will happen
CPU Bound – calculation would go faster if CPU were faster –
solutions: faster CPU, distribute to many CPUs, more cores, more threads, more efficient algorithms, reduce complexity.
I/O Bound – calculation would go faster if I/O were faster –
solutions: in-memory structures, in-memory databases, reduce data size to fit memory, SSD instead of HDD, local data, faster network, reduce number of IOPS needed.

Architecture: Single Node vs. Distributed Storage

Single Node – all storage attached to a single node
Node Attached Block Storage, example: AWS EBS
Distributed Storage – storage spread among multiple nodes connected by a network
SAN – Storage Attached Network – 1K to 100K IOPS – highest cost
NAS – Network Attached Storage – low IOPS – medium cost
NFS – Network File System – very low IOPS – unpredictable performance – lowest cost
Software Defined Storage – SWIFT – AWS S3 – sweet spots: low cost, throughput, high availability, capacity – weak spots: eventual consistency
HDFS – Hadoop Distributed File System – nodes each with storage

Architecture: Single Node vs Distributed Processing

Single Node – simple, but can only scale up
Parallel Computing – multiple processors or cores, shared storage layer
Distributed Computing – scale out, multiple nodes, requires: message passing, coordination, scheduling, tolerates node failures
Cluster Computing – uniform nodes, shared distributed storage, examples: Hadoop, Spark
Grid Computing – distributed nodes, heterogeneous and physically separate nodes, examples: SETI, CERN, protein folding

Codd, E. F. (1970). A relational model of data for large shared data banks. ACM Information Retrieval, 13(6): 377–387.

A landmark theoretical paper, but dated at this point. Codd invented the Relational Database to solve the problems of the databases of the 1960s. The Relational Algebra and Relational Calculus formed the basis for the Structured Query Language (SQL) which has stood the test of time as the de facto standard. The notation is archaic at this point – even today's discrete mathematics no longer uses that notation.

Chen, P. (1976). The entity relationship model—toward a unified view of SATA. ACM Transactions on Database Systems, 1(1): 9–36.

A landmark theoretical paper, but dated at this point. Chen invented the Entity-Relationship Diagram for Data Modeling still used today to model operational databases in 3NF (Third Normal Form). The Chen notation is archaic at this point (no longer used).

Proper, H. A. (1997). Data schema design as a schema evolution process. Data & Knowledge Engineering, 22(2):159–189.

A landmark theoretical paper, but dated at this point. He proposes that data modeling should be a several step process. This idea forms the customary data modeling process used today where we start with a Conceptional Data Model (CDM), to a Logical Data Model (LDM), to a Physical Data Model (PDM).

It also planted the idea that forms the basis of what is today called Object-Relational Modeling usually using Unified Modeling Language (UML). Relational Databases in 3NF have the advantage of no duplication of data, no modification anomalies, and ease to query using SQL. However, Object Oriented programming in languages such as Java or C++ find converting between 3NF and Objects very cumbersome. Based on the foundational ideas presented in this paper, today Object-Relational Modeling using UML allows us to bridge this gap and design abstraction layers.

Structuring Data

Unstructured Data – limits ability to understand and operate
Naturally Structured Data – human language, requires a lot of computation to extract information
Delimited Data – records encapsulate a set of details about facts, records can be mapped to tables: record = row, detail = column in a row

Schema

Schema – (dictionary) a representation of a plan or theory in the form of an outline or model (computers) a model we apply to data at storage or retrieval to: aid understanding, speed processing, save space, and enforce policy

Self-Describing Data – packing identifiers with data aids in accessing fields on demand
Document-Object Models – DOM - examples: HTML, XML, JSON.
Document = record, Attribute = detail
Manifestations of Schema - Tables, Collections
Tables – easy to access all values for single column or row, easier to treat numerically
Collections – more amendable to mutation (regarding schema not data), easier to capture hierarchy in a record

When is Schema Applied?

Schema-on-Write – data stored only if they fit schema constraints
Examples: Oracle, Microsoft SQL Server, Teradata, MySQL, PostgreSQL, etc.
Pros: data guaranteed to be in format, speedy retrieval
Cons: schemas must be defined before data stored, schema modification can be difficult and require re-organization of stored data, no multiple interpretations of data without duplicating data

Schema-on-Read – data stored without constraint, schema applied on every read
(read the data, apply schema, and validate)
Pros: multiple interpretations of data, varying strictness of validation, schemas can mutate over time
Cons: extremely slow, high cost to read and apply schema, no guarantee data will be in right format
Examples: local disks on computer, Big Data Frameworks (Hadoop, Spark), NoSQL Databases (MongoDB, CouchDB)
NoSQL may have self-describing schema – JSON, XML, Avro (binary format)
Other Schema Encoding Interchange Formats: Google protobuf, MessagePack, Cap-n Proto, Parquet (large column oriented databases – we will use with Hive)

Hybrid Approach – support both Schema-on-Write and Schema-on-Read
Examples: BigTable: HBase, Cassandra, Hive (in Lab 3 we will do both to show the difference)

Schema as Contract

Schema Behavior as a Contract – contract between storage-retrieval system, systems writing data, systems reading data
Examples: Warehousing Contracts, Transactional Contracts
ACID – <u>A</u>tomicity, <u>C</u>onsistency, <u>I</u>solation, <u>D</u>urability
BASE – <u>B</u>asically <u>A</u>vailable, <u>S</u>oft State, <u>E</u>ventual Consistency
(BASE is not in this unit, but I mentioned it here because we mentioned ACID)

Semantic Modeling

Semantic Modeling - multiple datasets each with a schema - we need a way to manage this
Semantic Modeling of a Business Process - model the real world as it exists in our data stores
Entity-Relationship Models – Entity-Relationship Diagrams - ERDs – translate business processes into schemas

Entity – nouns – collection of records – become tables in the RDB
Relationships – verbs – relationship between two entities (tables) – bi-directional – become Foreign Keys in the RDB – Foreign Key links to Primary Key – Arity (Cardinality) – Existence (Orphan Records)
Identifying Relationship – FK is part of the PK
Non-Identifying Relationship – FK is not part of the PK

Crow's Foot Notation
Entity – rectangle – PK column(s) above line – non-key columns below the line - rounded corners means it has at least 1 identifying relationship
0 ("oh") = cardinality of zero, | ("pipe") = cardinality of 1, crow's foot = cardinality >1 ("many")
Minimum Cardinality = inside marks
Maximum Cardinality = outside marks
When both minimum and maximum cardinality are 1 we use a single | mark
Business Rules – we speak the cardinality customarily in terms of maximum cardinality

Physical Schema

Entities – become tables
Attributes – become columns
Relationships – become FK
Primary Key - PK – column(s) which uniquely identify a row
Foreign Key – FK - links with a PK in another table
Child Table has FK which links to PK in **Parent Table** – any table may be a parent any number of times and a child any number of times, even to the same table, even to itself
Orphan Record – FK is NULL – row in child table with no matching row in parent table – only allowed if minimum cardinality is zero
Normalization – remove duplication, remove modification anomalies
Modification Anomalies – insertion anomaly, update anomaly, deletion anomaly
Insertion Anomaly – occurs when extra columns values beyond target values must be added to a table
Update Anomaly – occurs when it is necessary to change multiple rows to modify a single row
Deletion Anomaly – occurs whenever deleting a row inadvertently causes other data to be deleted
Normal Forms – rules about allowable dependencies that prevent the modification anomalies – each normal form is a superset of lower normal forms
1NF – First Normal Form – attribute domain atomicity –
industry slang: "no repeating groups of columns"
2NF – Second Normal Form – 1NF and if and only if nonprime attributes aren't part of the PK – industry slang: "no repeating groups of rows"
3NF – Third Normal Form – 2NF and if and only if attributes are determined only by the PK – industry slang: "nothing but the key"
Memory Aid: A database is in 3NF, if and only if every non-key attribute is dependent on the key, the whole key, and nothing but the key, so help me Codd
Higher Normal Forms: BCNF (Boyce-Codd Normal Form aka 3.5NF), 4NF (Fourth Normal Form), 5NF (Fifth Normal Form aka PJ/NF Project Join Normal Form), DKNY (Domain Key Normal Form)

Ghemawat, S., Gobioff, H., & Leung, S. (2003). The Google file system. SOSP'03, October 19–22, Bolton Landing, New York, USA.

In this landmark paper from 2003, the Google file system (GFS) is presented. It is the precursor to the modern open source Hadoop Distributed Files System (HDFS) that we are using in this course. It's interesting to read about the original design of GFS and what went into consideration, but keep in mind that HDFS is more modern and has differences.

Kreps, J. (2013). The log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn blog.

For many years, most software systems had logs, but only during the dot com boom of the late 1990s did companies start analyzing their web logs to piece together information about what visitors were doing on visits to their websites. For the first few years the analysis was rudimentary, but in the late 2000s, as Data Science was coming of age, the sophistication of web log analytics was reaching a high level. Web logs come at an extremely high velocity and volume, so Big Data architectures were being used to load and analyze them. A lot was learned from this architecture. This paper describes how to take what was learned in processing web logs and apply it to other areas, especially traditional data warehousing and the modern data lake architecture. It is the precursor to the ideas we will be using in this course: ELT instead of ETL, Schema-on-Read instead of or in addition to Schema-on-Write, and immutability, all of which help form the architecture of the data lake.

What is a Data Lake?

Data Warehouse – a repository of data collected from multiple data sources, stored under a unified schema and residing in a central place, Scale Out SQL, Dimensional Modeling (Star Schema, Snowflake, etc) instead of 3NF, early 1990's
Characteristics: data transformed to defined schema, loaded when usage defined, allows for quick response for defined queries
Vendors: Teradata, IBM Netezza, HP Vertica, AWS RedShift

Data Lake – a storage repository that holds a vast amount of raw data in their native formation until they are needed, analogy is a lake holds unpurified water for a city supply until it's needed and purified
Characteristics: many data sources, retain all data, allows for exploration, apply transforms as needed, apply schema as needed
Vendors: Hadoop (Cloudera, Hortonworks, MapR), AWS Elastic MapReduce (EMR), Teradata Aster

High-Level Architectures

Elements of an Analytics Architecture: collect / ingest data; store raw data; clean & transform data; query data; compute, join, aggregate data; analyze & report
(see diagrams in slides)
Data Warehouse Architecture: operational systems, ERP, CRM, flat files => ETL (extract transform load) => Data Warehouse (metadata, summary data, raw data) => OLAP analytics, reporting, data mining

Lambda Architecture: Speed Layer, Batch Layer, Serving Layer
Kappa Architecture: no Batch Layer
Speed Layer: quick analytics on streaming data, "knee jerk decision",
Examples: Storm, Twitter Heron, Spark Streaming
Batch Layer: store streaming data for deeper analytics at a later time, allows us to evaluate & improve decisions made in the Speed Layer
Examples: Hadoop, Spark
Serving Layer: presentation of data via queries, reports, apps, etc.
Netflix Architecture: S3 => AWS Elastic MapReduce (EMR) => analytics results back to S3, controlled by "Genie" (Hadoop Platform as a Service)

Data Characteristics

Online Transaction Processing – OLTP – system and approach that facilitate and manager transaction-oriented applications, typically for data entry and retrieval-transactions, CRUD (create, read, update, delete) transactions, frequent updates
Online Analytics Processing – OLAP – system and approach to answering multidimensional analytical queries quickly, aggregations, drill-downs, roll-ups
Data Warehouse Model – Master Data (dimension table) Transaction Data (fact table), Analytics Data (Cuboid)
Dimensional Modeling – different from 3NF, Star Schema, Snowflake Schema, (see slides)
Big Data Architecture – Master Data (fact-based, immutable dimensions), Transaction Data (log items), Analytics Data (aggregates, roll-ups)

Mapping to Data Architectures

See slide diagrams which show the Lambda Architecture, Traditional Business Data Warehouse, and Kappa Architecture and how the data characteristics discussed in the previous section map

Mapping to Data Architectures: NoSQL and HDFS

HDFS - Hadoop Distributed File System – presents like a local file system, distribution mechanics handled automatically
See slide diagrams for how HDFS works
Uses: popular bulk data store, many large files where file size > block size, agnostic to file content, good as immutable data store, good for parallel reads of lots of blocks, bad for small, specific reads, bad for fast writes

NoSQL - Not Only SQL – typically stores records as Key-Value Pairs, distribution mechanics tied to record keys
See slide diagrams for how Key-Value Stores work
Vendors: Redis, BerkeleyDB, MongoDB, Cassandra (column-families imposed on value)
Uses: good for fast, key-based access, good for fast writes, bad for off key access, complicated for merging datasets

Mapping to Data Architectures: Relational / Columnar

Two kinds of Database Management Systems – DBMS – Relational Databases – RDB – Columnar Databases – CDB
Relational Databases – RDB (or RDBMS) – present via declarative query languages, organize underlying storage row-wise (sometime column-wise)
Records are stored a tables – schema validate on write, typically indexed
Records may be persisted (row-wise, column-wise)
Additional structures can be applied to enhance access: secondary indices, materialized views, stored procedures
Uses: most common data storage and retrieval system, many drivers, declarative language (SQL); good for fast inserts, good for (some) fast reads, good for sharing data among applications, limited schema-on-read support, can be costly or difficult to scale
Columnar Databases – CDB – organize underlying storage column-wise, present via API and declarative query languages
Records stored as tables
Largely for analytical workloads (read mostly, bulk insertion)
Additional access structures
Presumption: more likely to read all values of a column, optimized storage for fast column retrieval
Uses: optimized for analytical workloads, maintains relational data model, good for analytical operations, good for horizontal scaling, bad for fast writes, bad for fast row-wise reads

Mapping to Data Architectures: Software-Defined Object Storage (Swift, S3)

Object Storage – data are managed as one large logical object: consists of data and metadata, has unique identifier across system; data are an uninterpreted set of bytes
Software-Defined Storage – data replication for resilience for high availability (HA); stores anything; runs on commodity hardware; manages data automatically, blocks hidden
Solution: simplified model for managing data growth; lowers management cost; lowers hardware cost; meets resiliency and HA needs and lower cost
How it works: simple API with object abstraction; data are partitioned & partitions replicated; software layer manages replication & data placement; software layer will automatically redistribute in case of expansion, contraction, or failure
Uses: when storing an object does not require indexed access; HA, scale, cost are key; “looser” consistency is acceptable
Vendors: OpenStack Swift (open source, Nasa & Rackspace); AWS S3; Oracle Storage Services; Google Cloud Storage; Windows Azur Storage; Rackspace Files

(next)

Management, Provenance, Governance

Lineage (Data Lineage) – data's origins and where it moves over time; tracked with either tools or manually

Provenance (Data Provenance) – documents the inputs, entities, systems, and processes that influence data of interest; provides a historical record of data and its origins; lineage, correctness of source, what has been manipulated or compromised, whether calculations comply, etc.

Governance (Data Governance) – control that ensures that the data meets precise standards; standards can be internal controls, such as business rules, or external controls such as industry standards or government regulations: SOX (Sarbanes-Oxley Act), PCI (Payment Card Industry), HIPAA (Health Insurance Portability and Accountability Act), FERPA (Family Educational Rights and Privacy Act), ADA (Americans with Disabilities Act), GAAP (Generally Accepted Accounting Principles)

Vassiliadis, P. (2009). A survey of extract–transform–load technology. International Journal of Data Warehousing & Mining, 5(3), 1–27.

As the title suggests, this paper from 2009 is a survey of the ETL (Extract, Transform, Load) technologies. It first covers the conceptional and logical modeling of ETL processes (which saw the need in the last unit for a division between CDM, LDM, and PDM). It then goes through issues with each of extract, transform, and load and then problems of the ETL process as a whole. This paper is not specific to Big Data, it also covers the area of traditional Business Data Warehousing. For Big Data related items, I think the most relevant sections are: Pivoting Problem, Data Mappers, Data Lineage Problem, Data Cleaning, Resumption Problem, Near-Real-Time ETL.

Kreps, J. et al. (2011). Kafka: A distributed messaging system for log processing. NetDB'11, Athens, Greece. ACM 978-1-4503-0652-2/11/06.

As the title suggest, this paper covers Kafka. Kafka is a scale out (distributed) architecture to process messages. The article highlights its ability to consume streaming logs, but it can also work with MQ (Message Queue) products. Over the last 15 years or so, most Fortune 500 companies have implemented extensive MQ implementations, usually using Tibco, IBM MQ, or MS MQ. These are designed on the producer / consumer model of messaging with messages organized into topic and recursive subtopics with guaranteed message delivery. Kafka is often placed in front of other streaming systems, such as Storm, Heron, or Spark Streaming, because it can scale out and handle huge volumes of incoming messages.

Introduction: Data Ingestion / Loading

Needs of Data Analytics – pyramid – basic needs => understanding needs => predictive needs
Common Problems: many data sources (SQL, NoSQL, streams, logs), many data transport protocols (HTTP, JDBC, REST, proprietary), different type of data bundles (full, incremental, changes), ingest for different data processes (batch, streaming)
Many Sources – many schemas
Network – reliability, fault tolerance, bottlenecks, bursts
Data Bundles – many semantics

Traditional ETL / ELT

ETL – Extract, Transform, Load – traditional for business data warehousing, we take the hit at ETL time to make analytics time faster
ELT – Extract, Load, Transform – new for Big Data, we assume a more powerful target system, more flexibility at later stages

Ingesting of High-Velocity Logs

Large amount of data, TiBs daily, both online and offline processing (Lambda Architecture)
Vendors: Karka, Amazon Kinesis, S4, Storm, Samza

Kafka – messaging API, stores many messages for long periods, scales out (distributed), designed for large datasets, allows clients to read in different orders - scalability, network bottlenecks, consumer bottlenecks, bursts, reliability, fault tolerance – topology of producer / consumer – topic oriented, consumers can subscribe to topics (or recursive subtopics) – messages saved and replayed – MQ integration

Big Data Ingest: Logs and ETL

LinkedIn Gobblin – example of ETL to ingest logs
Source integration – out-of-the-box adaptors for data sources such as Salesforce, MySQL, Google, Kafka, Databus
Processing paradigm – support both standalone and scalable platforms, such as Hadoop and Yarn
Data quality assurance – framework exposes data metrics collectors and data quality checkers as first-class citizens
Extensibility – data pipeline developers can integrate their own adaptors
Self-service – data pipeline developers can compose ingestion and transformation flow

Moving Large Data Sets

Considerations – parallelism, kind of source, kind of sink, network bandwidth usage, different formats, different structures, long move times, handling failures
Performance Measures – bandwidth (max xfer rate), throughput (actual xfer rate), latency (delays between sender and receiver), jitter (variation in time of arrival at the receiver), error rate (percentage of corrupt bits)
Tools – Sqoop, distcp2, rsynch
Sqoop – xfer RDBMS (SQL) and Data Warehouses to Hadoop – limited parallelism – specific to SQL – limited recovery for failures
rsynch – FS (file system) to FS copy or synchronization – old – reliable – not parallel – efficient use of network – agnostic to file content – no recovery for failures, must restart
distcp2 – HDFS or S3 to HDFS copy – high parallelism – agnostic to file content – handles failure well using standard MapReduce recovery

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. HotCloud.

A Berkeley paper - as the title implies, this paper is about Spark. Spark was designed to process in memory using what is called RDDs (Resilient Distributed Datasets) and use a form of MapReduce that is not limited to chaining the Hadoop style MapReduce implementations. Obviously, this allows iterative job streams to process many times faster, which they point out is especially helpful for interactive analytics. It described the operations of Spark, such as parallelize, flatMap, map, filter, reduce, collect, foreach, etc. and explains how these are used to overcome limitations of chained MapReduce jobs. It gives practical examples of a text search, and machine learning algorithms logistic regression and alternative least squares, both of which benefit greatly from the architecture because they iterate over the same data numerous times. It also describes that Spark is designed to be able to run both inside Hadoop and independently of Hadoop (including using Mesos a cluster operating system). The importance of lineage (provenance) and how it was designed into Spark is discussed at several key points.

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1):107–113.

A 2004 paper from Google that described MapReduce as a simplified way to do data processing on large clusters. This information is a bit dated at this point, as there have been improvements to MapReduce since then and newer implementations of MapReduce such as Spark. But, it does give you the initial design details and considerations that went into building the original version of MapReduce. Their choice of algorithms is very telling as to why it was needed as it maximizes such things are word counts, sorting, aggregating counts, greps, skipping bad records, large scale indexing, etc. – all things needed for Google's work on indexing the entire Internet at scale. Fault tolerance and recovery from failures were also of high importance.

Introduction: Processing

Type of Processing Operations – filter, mutate, aggregate, combine or merge
Filter – exclude some records from the set – selection (filter rows) – projection (filter columns)
Mutate – for each record in a set, produce a new record based on some transaction
Aggregate – combine records in a dataset to produce an aggregated feature – sum, count, average, etc.
Merge – merge records from two datasets into a new record based on some condition – special case of aggregation – inner join, left (right) outer join, Cartesian Product, etc.

Methods of Processing

Serial Processing – process records one at a time in order read – similar to procedural programming – not efficient – no scale out – disk I/O costs dwarf memory costs, heavy use of disk seeks
Declarative / Functional – Functionally Drive Processing – logically consistent with serialized processing – reads and processes batches of records – applies functions to batches – scale out: multiple threads, multiple computers

Functional in Parallelism

Function passing – makes scale out easier – immutable data, portable closures – dual of the actor model
Vendors: scale out SQL, Hadoop, Spark
Common Operators: filter, map, flatMap, reduce, fold
filter – filter the collection such that only some set of records pass
map – apply the function f(x) to all members of x in the collection
flatMap – flatten the collection and apply f(x) to all resulting members
reduce – apply aggregating function f(x,y) to all members of the collection
fold – apply aggregating function f(x,y) to all members and store the result in a variable g

Processing in Stages

Break down processing into stages – complex processing broken down into smaller stages – DAG (Directed Acyclic Graph) is a plan for process (we used DAGs with SQL execution plans) – DAGs allow for optimization and fault tolerance, but present new complexities such as merging datasets
MapReduce – Map (filter data, mutate data, merge data under certain conditions) – Reduce (filter data, aggregate data, merge data)
“generic” MapReduce - a logical concept, chaining of two functions, exists in many languages (Python, Ruby, Scala)
Hadoop MapReduce – specific implementation – data-processing framework (distributed system, designed for massive data processing) – Read Data => Map => Reduce => Write Data
MapReduce limitations – program can only map and reduce between I/Os – have to chain MapReduce jobs for iterative, merge, or aggregate operations – read data in and out more than once
MapReduce => checkpoint => MapReduce => checkpoint => etc. - checkpoint requires I/O and data cannot be kept in memory for the next MapReduce job – good for fault tolerance – bad for performance
DAG – Directed Acyclic Graph – directed (operations flow one way) – acyclic (no cycles – no dependencies on future steps) – enables dataflow processing (data enters node, date leaves node)
Logical plan – separate from physical plan (choice of algorithm)
Optimized – control checkpoints – mover operators around to minimize data xfer
Distributed – some nodes occur at each member of the system – some nodes gather data from other members (shuffling)
Examples: Pig (declarative dataflow language that transforms to Hadoop MapReduce as either a Map or a Reduce), Tez (extension of Hadoop MapReduce to more generic DAG – removes checkpoints between MapReduce iternations), Spark (memory-centric dataflow framework built on functional operators), Scale Out SQL (execution plans in trees as special case of DAGs)
How to Optimize Dag – minimize materialization (read minimum amount of data necessary, filter data as early as possible, write data as late as possible), minimize data shipping (passing data between processes is expensive – serialization - shared memory or network transfer, minimize the number and size of shuffles – use combiners aka partial aggregation)

Understanding Aggregation

Important – how two datasets relate to each other, statistics about dataset (mean, median, variance, min/max, etc.)
Problems at scale – shuffles required (data shipping overhead)
Partitioning – highly dependent on partitioning – aggregations on multiple partitions must shuffle – merges on disparate partitions must be shuffled
Partial Aggregation – combiners – pre-aggregate – reduce shuffle overhead
Merging and Data Skew – partitions being joined are of uneven sizes
Solutions: Broadcast Join – copy the small partition out to the nodes, **Pre-filtering** – filter the large partitions before merging with the small partitions

Unit 7 – Querying Data

Graefe, G. (1993). Query evaluation techniques for large databases. ACM Computing Surveys (CSUR), 25(2): 73–169.

This is a long paper from 1993, over 90 pages, which is comprehensive over all major issues regarding how to take a query and create an execution plan. It starts with presenting how to structure DAGs. It covers various methods of sorting and hashing, and the appropriate uses and tradeoffs. Disk access techniques are presented, although much of this is dated at this point due to modern storage architectures. Aggregation, duplicate removal, and binary matching techniques are presented, which are still in widespread use today. The last half of the paper deals with parallel query execution plans, which I think the MapReduce material in this course supersedes.

Chaudhuri, S. (1998). An overview of query optimization in relational systems. Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.

This paper from 1998 provides the foundation for the modern cost based optimizer. Before cost based optimizers, we used rule based optimization. Rule based optimizers looked at the basics such as the table structure, the joins in the query, what indices are present, etc. and made a decision without regard to examining the actual data sizing and characteristics. Cost based optimizers gather statistics on tables, indices, IOPS of various hardware components, etc., and then generate a list of execution plans and computer a cost for each execution plan. The execution plan with the lowest cost is selected. Note that as data is added, statistics change and have to be gathered on a regular basis.

Stonebraker, M. et al. (2005). C-store: A column-oriented DBMS. Proceedings of the 31st International Conference on Very Large Databases. VLDB Endowment.

This paper from 2005 proposed a database in which tables are essentially stored in column major order instead of row major order. Today, it's called vertical partitioning or a columnar database. In support of this, it outlines: shared nothing architecture (scale out), how the query execution DAGs would be different, why compacting storage is better using columns instead of rows, how indices need to be different, how to perform distributed transactions without traditional logs and two-phase commit, and how to isolate queries.

Review: Schema, RDBMS, and DAGs

Schema – (dictionary) a representation of a plan or theory in the form of an outline or model (computers) a model we apply to data at storage or retrieval to: aid understanding, speed processing, save space, and enforce policy
We apply schema at the dataset level. >1 data set, we use an ERD to
Entity-Relationship Models – Entity-Relationship Diagrams - ERDs – translate business processes into multiple schemas
Entity – nouns – collection of records – become tables in the RDB
Relationships – verbs – relationship between two entities (tables) – bi-directional – become Foreign Keys in the RDB – Foreign Key links to Primary Key – Arity (Cardinality) – Existence (Orphan Records)
Identifying Relationship – FK is part of the PK
Non-Identifying Relationship – FK is not part of the PK

Crow's Foot Notation
Entity – rectangle – PK column(s) above line – non-key columns below the line - rounded corners means it has at least 1 identifying relationship
0 ("oh") = cardinality of zero, | ("pipe") = cardinality of 1, crow's foot = cardinality >1 ("many")
Minimum Cardinality = inside marks
Maximum Cardinality = outside marks
When both minimum and maximum cardinality are 1 we use a single | mark
Business Rules – we speak the cardinality customarily in terms of maximum cardinality

Review the ERDs from the slides for 7.3 (slides 14 through 20)

DAGs – Directed Acyclic Graphs -
Graph – collection of vertices and edges
Directed Graph – edges have a direction
DAG – a Directed Graph with no cycles – in computer science used in multi-processing to show: the processes, which processes must complete before a process can run, which processes can run at the same time. Database queries are translated into execution plans in the form of DAGs

Motivation for Declarative Languages

How to get from Query to DAG: 1) procedural language 2) declarative language
Procedural Language – specify step by step execution path – "how" – loops – examples: Python, Java, C++, Javascript. **Pros:** complete control, **Cons:** more work and responsibility
Declarative Language – aka Functional Language – specify "what" not "how" – no loops – examples: SQL, HTML, Regular Expressions, Rails **Pros:** less work, **Cons:** lack of control
Best of both worlds: often design outer layer that is Procedural (Python) with an inner layer that is Declarative (SQL). Do as much as you can in SQL, use Python when you need a loop or other procedural structure or "glue".
Oracle PL/SQL is an example a language system designed for this exact purpose.

Structured Query Language (SQL)

SQL - declarative language – de facto standard query language for RDB – stood test of time
Optimizer – generates execution plans for SQL – implements the declarative language into procedural using DAGs
Lab 5 - examples of PostgreSQL – generate execution plans (DAGs) for the SQL

Joins

FK in child table links with PK in parent table – ERD shows us how to do this – always consult ERD before writing SQL with joins
Joins: Inner, Outer (left, right, full), Cross (Cartesian Product)
Inner Join: only FK to PK matches

Outer Join: FK to PK matches, left – adds non-matching rows in left table – right adds non-matching rows in the right table (seldom used) - full adds non-matching rows in both left and right
Cross (Cartesian Product) – rows don't match FK to PK
Implementations: Nested-loop, Sort-merge, Hash

Analytical SQL and Windows

Time-series – today minus yesterday
Rolling sums – yesterday plus today
Ranking rows – whole table or by partition

Indexes (Indices?)

Indexes – physical data structures we create to make retrieval faster – but slower insertions and use space
B+ Tree – most common index – similar to binary tree data structures you coded in the Python class
Bitmap Index – low cardinality of values – example: 1 million customer of 5 types
Hash Index – retrieve 1 row by a unique key value

View / Partitions

View – virtual table based on an SQL select statement – no space – CREATE my_view as SELECT …
Hide complex logic such as joins – subsets of tables – security – encryption layer
Materialized View – caches query results – uses space – defined refresh rate
Partition (horizontal) – divide rows by a List of values, Range of values, Hash

Approximate SQL

Approximate SQL - Table may be too large to query and a sample of data may yield meaningful analytical value
Data Sketching – query counts, groupings, minimums, maximums, averages, etc.
Uniform and Stratified Sampling - obtain sample of data that represents the entire data using a combination of taking a uniform sample and reducing it by stratification
Uniform Sample – pull a uniform number of random records, example: every 10th record
Stratified Sample – pull a sample that proportionally matches the data,
example: if 25% of our customers are male and between 15 and 25 make sure 25% of our sample is male and between 15 and 25

Unit 8 – Exploring Data

Stevens, S. S. (1946). On theory of scales and measurement. Science, 103(2684).

A scientific paper from 1946, before electronic computers, discussing scales for measurement, which are the same ones in use today and used in the videos / slides for this unit: nominal, ordinal, interval, and ratio. Nominal scale allows determination of equality. Ordinal scale allows determination of greater or less. Interval scale allows determination of equality of intervals or differences. Ratio scale allows determination of equality of ratios. For each scale, a set of permissible statistics is listed.

Tukey, J. W. (1980). We need both exploratory and confirmatory. The American Statistician, 34(1): 23–25.

In this paper from 1980, a statistics professor makes a case that universities at the time were currently teaching only Confirmatory Data Analysis (CDA) and not Exploratory Data Analysis (EDA). CDA is based on the traditional scientific method of making a hypothesis and using data and statistics to prove or disprove it. He describes EDA as "an attitude … AND a flexibility… AND some graph paper…" to use statistics. The "graph paper" today would be called data visualization. He closes by stressing that results from EDA should be confirmed with CDA – that both are needed.

Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T., (2010). Dremel: Interactive analysis of web-scale datasets. Proceedings of the VLDB Endowment, 3(1).

This paper from 2010 describes Google's internal software systems called Dremel which they release d in 2014 as open source Apache Drill. Dremel scales and excels at very large datasets (> trillion rows) using a nested tree columnar data storage structure. It is especially well suited for nested dataset formats such as JSON. The paper covers the details of the nested columnar storage format, the query language and execution plans, how execution trees that are commonly used in web search engines can be applied to database schema, and results of experiments of running queries on trillion row datasets. It can be used with MapReduce with either feeding data to the other. It can also read and process JSON files in S3, which I find very interesting.

Introduction to Data Discovery

- **Data Discovery**
  - **Data Characteristics** – what are the distribution and ranges? How big are the data? What are the missing values? Are there outliers? Are data skewed? What are the type, scale, and measures?
  - **Insights**
    - **Exploratory Analysis – EDA** – explore data, look for new insights
      - Structure Learning
      - Faceted Search
      - Visualization
      - Pattern Detection
      - Query and Filter
    - **Confirmatory Analysis – CDA** – define a hypothesis, try to prove or disprove it
      - Statistical Analysis

Understanding Your Data

Categorical variable – a variable that can take on one of a limited number of possible values, thus assigning each individual to a particular category
Quantative / Continuous Data – data where the values can change continuously, and you cannot count the number of different values
Type of Data / Variables – Binary (nominal scale), categorical (nominal scale), ordinal (ordinal scale), absolute (count, ratio scale), real-valued (additive, interval scale), real-valued (multiplicative, ratio scale)
Scales – nominal, ordinal, interval, ratio
Managing Missing Value (Imputation) – data deletion (delete the record), hot-deck imputation (replace with last value seen), mean imputation (replace with the mean), median imputation (replace with the mean), class imputation (use class mean or median), "most probable" imputation (use regression or inference to predict likely value)

Exploratory Data Analysis – EDA

Inductive – look for ways to examine data without preconceptions, evaluate assumptions, use data visualizations, let data suggest questions, focus on indications and approximate error magnitudes
Advantages – flexible way to generate hypothesis, more realistic statements of accuracy, does not require more data than data can support, promotes deeper understanding of processes
Disadvantages – usually does not provide definitive answers, difficult to avoid optimistic bias produced by overfitting, requires judgement and artistry – can't be cook booked
Search – relevance, precision / recall, taxonomies, ontologies, ranking
Faceted Search – technique for accessing information organized according to a faceted classification system, allowing users to explore a collection of information by applying multiple filters
Parametric Search – Boolean interface for combining faceted content using AND / OR combinations
Faceted Classification – uses a set of semantically cohesive categories that are combined as needed to create an expression of a concept, in this way, it is unlimited in the concepts it can express (what kind of wine are you looking for?)
Faceted Search – facets + text search (narrow selection by clicking on web links)
SQL – obviously good to use for exploring data by query
Data Cubes – facts (quantitative data) inside the cube, dimensions (measures) go on edges, drill up, drill down, roll up, slicing, dicing, pivot, etc.
Big Data Examples: Hive, Spark SQL, BigQuery, BlinkDB, Impala, AWS Redshift, etc.

Visualization and Its Realization

Slides have various examples of data visualizations: bar charts, histograms, scatterplot, boxplot, geomap, bubble chart, parallel coordinates, Gantt chart, stream graph, etc.
Vendors: Oracle Big Data Discovery, Tableau

Confirmatory Data Analysis - CDA

Statistical CDA – state null hypothesis => state alternative hypothesis => set alpha (significance level) => collect data => calculate a test statistic => construct acceptance / rejection criteria => draw conclusion on hypothesis
Basic CDA – state question => hypothesis => query data => analyze results

Sampling, Enriching, Merging

Sampling for discovery – way to manage very large datasets, can provide for faster and more interactive handling, allows exploration of datasets whose complete form is unmanageable on a platform
Basic Sampling – Set D contains N tuples; sample size is s
Simple random sample without replacement - draw s of N tuples with probability of 1/N
Simple random sample with replacement - previous, but put tuples back in for possible redraw
Cluster sampling - D divided in M disjoint groups; apply SRS at cluster level
Stratified sampling - previous, but apply SRS at the tuples level proportional to cluster size
BlinkDB – scale out SQL with approximate query engine for running SQL on large volumes of data trading off query accuracy for response time with meaningful error bars
Transformation – get data to right type and form, calculate derived data, split data
Enrichment – from another data source, add information that is valuable on discovery such as table join or lookup from external source

Clustering

Segmentation [DEFINE LABELS] - user defines the border between segments, examples: gender, age, income range
Clustering [DISCOVERY LABELS] – algorithmically detect groupings based on some features, examples: find clusters and their defining features
Classification [GROUP TO KNOWN LABELS] – bucket items on some predefined set of attributes
Cluster Analysis – the activity of finding clusters and their defining attributes
Clustering Techniques – different characteristics and limitations – some expect a guess of number of clusters; some are nondeterministic; some do not scale well
Implementations – R (hierarchical agglomerative, K-means), Spark-MLlib (K-means), Mahout (K-means)
Storage and Computing Needs – very computationally intensive; sometimes hard to distribute; difficult to make interactive

Classification

Classification [GROUP TO KNOWN LABELS] – bucket items on some predefined set of attributes – understand the structure of the data, use for decision making, score new incoming data
Implementations – Decision Trees (R, MLlib, Weka), Rule Based (R, Mahout, MLlib), Support Vector Machines (R, MLlib, Weka), Naïve Bayes (R, Mahout, MLlib, Weka)

Toshniwal, A. et al. (2014). Storm@Twitter. Proceedings of SIGMOD Conference.

This first paper from 2014, describes how Storm was being used at Twitter at the time, and mentions that Storm is growing in use at Twitter, which is understandable given that they produce GiB's of streaming data each minute. Parts 1 and 2 are similar to the Storm information presented in this unit's video material. Part 3 is of most interest here, as it goes over the operational difficulties that Twitter has experienced using Storm, including Zookeeper overloads, overheads in Storm being much greater than Kafka creating back pressure on spouts, difficulty in tuning spouts and bolts due to deep layers of code, and difficulty in taking action to release back pressure at one point creating back pressure elsewhere.

Kulkarni, S. et al. (2015). Twitter Heron: Streaming at scale. Proceedings of SIGMOD Conference.

This paper from 2015, just a year after the previous paper, describes how Twitter is replacing Storm with Heron. Parts 1 and 2 go over Heron, which is similar to material in the unit videos. Part 3 is of interest, as it details the motivation for Heron, which is mainly the shortcoming of Storm: Worker limitations, Nimbus issues, Zookeeper overloads, backpressure, and efficiency issues. Part 4 tells us that there really wasn't a good alternative, so they had to develop something new. Part 5 described the design of Heron and how it fixes the shortcomings of Storm. The remaining sections of the paper go through production, metrics, future directions, etc.

Introduction to Streaming

Analytics – Discovery (ability to identify patterns in data), Communications (provThiide insights in a meaningful way)
Types of Analytics – Cube analytics, Predictive analytics
Dimensions of Analytics – Streaming, Interactive, Batch
Streaming – ability to analyze data immediately after it is produced
Stream Processing – analyze and act on streaming data using continuous queries
Streaming Analytics – continuously calculate mathematical or statistical analysis on the fly
Interactive – ability to provide results instantly when a query is posed
Batch – ability to provide insights after several hours/days when a query is posed
Examples: network monitoring, intelligence and surveillance, risk management, e-commerce, m-commerce, transaction cost analysis, fraud detection, algorithmic trading, live datamart
Stream System Requirements – processing massive amounts of streaming events, performance and scalability as data volume increases, real-time responsiveness to changing conditions, rapid integration with existing data sources, push-based visualization
Stream Processing = Data Flow Through a DAG
First Generation – SQL based – NIAGARA Query Engine, Stanford Stream Data Manager, Aurora Stream Processing Engine, Borealis Distributed Stream Processing Engine, Cayuga Stateful Event Monitoring
Next-Generation – Storm, Spark Streaming, Twitter Heron, S4, Samza, Pulsar

Storm Overview

Storm – streaming platform for analyzing real-time data as it arrives, so you can react to data as it happens: guaranteed message processing, horizontal scalability, robust fault tolerance, concise code-focus on login
Storm Data Model – Topology, Spouts, Bolts
Topology – DAG; vertices = computation; edges = streams of data tuples
Spouts – sources of data tuples for the topology; examples: Kafka, Kestrel, MySQL, PostgreSQL
Bolts – process incoming tuples, emit outgoing tuples; examples: filtering, aggregation, join, arbitrary function

Storm Example – see Lab 6 – Apache Storm Introduction

Storm Architecture

(see slide diagrams)
Master Node (Nimbus), Zookeeper (ZK), Slave Nodes (Supervisor process plus Worker processes)

Storm Deployment

Mesos – distributed systems kernel, scales out – Storm can run multiple instances on same cluster using Mesos – (Spark also can use Mesos) – can run outside of Hadoop
Topology Isolation – shared pool (multiple topologies can run on the same host), isolated pool (dedicated set of hosts to run a single topology)

Storm Issues

Master Node (Nimbus) is a single point of failure
Zookeeper (ZK) is easily susceptible to storage contention, constantly spawning and killing processes
Storm Workers – complex hierarchy – hard to debug (you will experience debugging in Lab 6 and Exercise 2) – hard to tune
Queue contention between Spouts and/or Bolts
Evolution vs revolution – fix storm or develop new system?

Twitter Heron

Full API compatible with Storm (DAG, topologies, spouts, bolts) and well known languages (C++, Java, Python, NO Clojure)
Heron Design Choices
Off-the-shelf scheduler – unmanaged (YARN, Mesos), managed (Aurora, ECS)
Containers for resource reservation – plays nicely with Linux C group container, limit CPU and RAM
Separation of data routing and data processing – allows multi-language data processing
Back pressure – self-adjust the topology and provide predictability
Batching – moves tuple faster for higher throughput

Topology Architecture – see slide diagrams
Topology Master – solely responsible for entire topology (assigns roles, monitoring, metrics)
Stream Manager – routing engine (routes tuples, back pressure, ack mgmt)
Back Pressure Advantages – predictability (tuple failures are more deterministic), self-adjusts (topology goes as fast as the slowest component)
Heron Instance Architecture – does the real work (runs one task, exposes api, collect metrics)

Twitter Heron Performance / Operational Experiences – see slide diagrams

Unit 10 – Cleaning Data, Entity Linkage, String Clustering, and Fuzzy Methods for Merging Datasets

Elmagarmid, A., Ipeirotis, P., & Verykios, V. (2007). Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering, 19(1): 1–16. link Read the following sections: 1,2 3 {3.1.1,3.1.2,3.1.4,3.3.1} 4 {4.1,4.3,4.5,4.6,4.8},5{5.1,5.2},7. The rest is optional.

> This paper discusses methods for detecting duplicate records, but it starts by mentioning that often duplicate records don't share a common primary key and often bad data is present in duplicates. The next section goes through a list of the all the common methods for fixing bad data, with various methods to match on similarity and differences, several of which are covered in the videos for this unit. Having dealt with bad data, the final part of the paper goes through with various methods of how to detect duplicates, and closes with how to improve the efficiency of duplicate detection.

Rahm, E., & Do, H. H. (2000). Data cleaning: Problems and current approaches. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. link Read the following sections: 1,2 3 {3.1.1,3.1.2,3.1.4,3.3.1} 4 {4.1,4.3,4.5,4.6,4.8},5{5.1,5.2},7. The rest is optional.

> As the name implies, this paper goes through the various methods for data cleansing as it related to ETL in the traditional data warehouse, although the same techniques apply to big data. The hierarch of data quality problems, single source problems, and multisource problems match the material presented in the videos for this unit. The remainder of the paper presents data cleansing techniques and tools, which are similar to the modern tools, but dated at this point.

Wang, R. Y., &. Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems, 12(4): 5–33. link Read the following sections: 1,2 3 5. The rest is optional. Read the following sections: Introduction, Preliminary Conceptual Framework ,Toward a Hierarchical Framework of Data Quality. The rest is optional.

> This paper from 1996 highlights the fact that at the time the general feeling in IT was that as long as data is accurate, that was all that mattered with regards to data quality. This paper presents the results of research conducted by surveys of data consumers asking them what data quality attributes are most important to them. The first survey asked them to list quality attributes. The second survey complied and organized the results from the first survey into more pointed questions. The findings were compiled and a hierarchical framework for data quality was proposed.

Han, J., Kamber, M., & Pei, J. (2012). Data mining: Concepts and techniques, (3rd ed.). Morgan Kaufman, Chapter 3, pp. 83–120.Read the following sections: 3.1, 3.2, 3.3.1, 3.4.8-3.4.9 Optional : 3.3.2-3.4.7, 3.5

> This chapter deals with data preprocessing: data cleaning, data integration, data reduction, data transformation, and data discretization. The material is very timely and relevant and is similar to the material presented in the unit videos.

Introduction

Bad Data – misspellings, outliers, incorrect values, missing values, incorrect values
Entity Linkage - entity and its associated attributes
Type of Problems

- Single-Source Problems
  - Schema Level – lack of integrity constraints, poor schema design, uniqueness, referential integrity
  - Instance Level – data entry errors, misspellings, redundancy, duplicates, contradictory values
- Multi-Source Problems
  - Schema Level – heterogeneous data models and schema designs, naming conflicts, structural conflicts
  - Instance Level – overlapping, contradicting, and inconsistent data, inconsistent aggregating, inconsistent timing

Defining Data Quality

Entry Quality – is the record entered correctly? – check in single record sets and data streams, fixes can be automatic and independent
Process Quality – was the record correct throughout the system? – check in single record sets and data streams, fixes can be automatic and independent

Identity Quality – are similar entities resolved to be the same? – check across many datasets, fixes may need extra intelligence
Integration Quality – are the data sufficiently integrated to answer relevant questions? – check in architecture, fixes can be costly
Usage Quality – are the data being used correctly? - check in the organization, fixes may be nontechnical
Age Quality – are the data new enough to be trusted?
Data Quality as a Hierarchy – Entry => Process => Identity => Integration => Usage

Single Stream Issues

Entry Quality – incorrect data at source
Process Quality – incorrect data at destination
Single Stream Quality Evaluation – SSQE – framework – determine what issues might occur in the data, weigh the criticality of the issues, profile the data to score quality – allows quality to be ever-increasing standard – allows us to prioritize data-cleaning activities
Quality Issues for Single Streams – Definitional, Incorrect, Too Soon to Tell
Definitional – constants, definition mismatches, filler containing data, inconsistent cases, inconsistent data types, inconsistent null rules
Incorrect – invalid keys, invalid values, miscellaneous, missing values, orphans, out of range

Too Soon to Tell – pattern exceptions, potential constants, potential defaults, potential duplicates, potential invalids, potential redundant values, potential unused fields, rule exceptions, unused fields
Comparing Quality Effects – Weighting Issues – issues should be weighted in the context of the system, invalid or missing data are most problematic, potential issues can be weighted lower
Accessing Quality – any issue has a possible maximum, possibilities will be cardinalities of rows, keys, constants, etc., discovered are results of digging through dataset looking for rule violations
Scoring Quality – weighting issues allow us 2 views of data: raw and weighted, process helps us understand where issues are occurring, impact helps us understand the effect to downstream customers
Ensuring Stream Quality –
integrate quality check throughout data pipelines (pass records once and only once, enable the replay of records lost in transmission),
enforce schema (applying and enforcing schema at the RPC layer can radically improve quality, "transaction contract" for all senders and receivers, flexible schema RPC formats help enable this: Avra, protocol buffers, MessagePack, etc)

Missing Values

We may never fully know why values are missing – we need to know how to deal with missing data
We need to understand: relation of missing to a field (are most values missing), relation of missing to a group (are most missing values in column y when column x = 1?)
Understanding missing values helps us model substitutes or unbiased estimates
Nuclear Option: Listwise Removal – simplest approach – if record has missing value, ignore the entire record in analyses – only acceptable if remaining statistical power and sample size are large enough and removal doesn't bias any models - system of record, not acceptable, rest of record is useful, must decide a priori what is acceptable as missing, constraints important
Softer Removal: **Pairwise Removal** – less extreme – if record has a missing value, ignore the record if and only if the missing value is in the analyzed set – in analysis this can be acceptable, use as many values as possible
Inferring Values – fill in missing values – time-ordered, rules can be applied, must understand the potential impact – more scientific approaches that can be brought to bear
Interpolation for Numeric Data – predict new data points in the range of a discrete set (model the existing values in one column as a response of other columns, for each value, predict it based on the available data) – Simple Linear, Splines, Kringing
Imputation for General Data – impute using a good model (replacement of missing data with substitute values, unit imputation – adding a whole observation, item imputation – adding a feature of an observation (a column in a row)) latent variables (don't know what mechanism generated the data, only know the data points it generated)
Expectation Maximization – EM – Gaussian Mixture Models (GMMs) (all data points including missing values are drawn from a mixture of normal distributions with unknown parameters, find the optimal mixing of distributions to generate the observed data) – Two-Step Approach (E-Step – assigns points to the model that best fits, M-Step – updates model parameters using assigned points, repeat until converged)

Single and Multisource

Entity Resolution / Record Linkage – Match records from multiple data sources that belong to the same entity – problems: attributes / records not matching 100%, how to apply to different / changing datasets, missing values, data quality, errors, semantic relations – core problems: normalize data, link records in a table, link records across tables
Single Source Approach – normalize attribute data (find matching, replace with a normalized value), match records on one or more attributes (match and annotate with an identifier)
Deterministic Method – rule-based approach: rules and record level match – scale issues: bad data in many rules
Fuzzy Method – match similar values and records and attempt to determine if match, no match, or possible match – similarity of attribute, distance of record, match thresholds, learning
Precision – how many selected items are relevant?
Recall – how many relevant items are selected?
Matching / Linkage – Structural (hierarchy or similar linkage) – Lexical (text matching)
Structural Heterogeneity – issues: different attributes / columns, split in different tables, different level or normalization, nesting levels (JSON), attribute versus element (XML)
Lexical Heterogeneity – issues: abbreviations, alternative spellings – common operations to solve: match, trim, split, join, slice, partition, etc.
Edit Distance – way to measure the similarity of string attributes, quantifies how dissimilar two strings are to one another by counting the minimum number of operations required to transform one string into the other
Levenshtein Distance – minimum number of single character edits (insert, delete, substitute) – see examples in the slides
Jaro-Winkler Distance (Jaro Distance) – measure of the similarity between two strings – between 0 and 1, must decide an appropriate threshold, 1 is exact match, 0 is no similarity – see examples in slides
Other Algorithms – Hamming Distance, Damerau-Levenshtein Distance, longest common substring distance, cosine distance, Jaccard Distance

Additional Explanation of Edit Distance and Levenshtein – see slides for additional examples

Record Linkage

Problems with Record Linkage – records in different data sources, no common record key, want to find similar records, assess similarity, create linkage using fuzzy and deterministic rules, how to compare whole record
Record Linkage – normalize data, link records on a table, link records across tables
Defining Similarity – application specific, different attributes, different angles, methods: cosine similarity for numeric / continuous attributes, edit distances for strings, Jaccard for binary or categorical values
Processing and Storing – cleaning and identifying similar records within / across data sources can be computationally expensive, calculations are usually subjective and difficult to tune (store original data

if possible, can be costly, recalculate frequently, Hadoop and Spark have dramatically reduced processing times)

Scaling Record Linkage

Entity matching can be very computationally intensive – we need to scale
Scale / Efficiency Issues – polynomial complexity, NM record comparisons, expensive comparison operations, multiple passes over data likely needed – **Concerns:** precision and recall, efficiency, processing times
Efficiency Improvements:
Reduce number of comparisons – blocking, sliding window, clustering, canopy
Reduce comparison complexity – filtering: decision tree, feature selection
Blocking – define blocking key, create blocks, compare within blocks – pros: fewer comparisons – cons: missed matches – option: block on multiple different attributes
Sliding Window – create key from relevant data, sort on key, compare records in sliding window – pros: few comparisons – cons: sensitive to key selection – option: run multiple sliding windows in parallel
Clustering – assumes similarity is transitive a=>b=>c, union-find algorithm - pros: fewer comparisons – cons: potentially complex computation, slightly more unpredictable
Canopy – cluster into canopies using "cheap" algorithm, run comparison inside each canopy with more "expensive" algorithm, preselection – pros: fewer comparisons – cons: sensitivity to key selection, missed matches
Big Data Solutions – scale out, process large datasets faster, save original data which makes algorithms better

Ontologies and Semantics

Ontologies – modeling of a system (entities, relationships, events that may change entities or relationships, axioms or rules the define what can happen) – Greek "the study of being or reality" – models can be: small, limited domains (ERDs) or large, global semantics (Semantic Web)
Semantic Web – seeks to link all entities and relationships on the web to a global ontology
Web Ontology Language – OWL – markup language, applies ontology onto any object with a URI
Resource Description Framework – RDF – framework for constructing an ontology as a graph
Knowledge Bases – attempts at capturing some semantic model of the world, in limited form with limited data, examples: one stop semantic models: Google Knowledge Graph API, Wikidata, DBPedia – can provide a common root for entity resolution since construction of a semantic web would amount to a significant data quality problem

Unit 11 – Graph Models and Analysis

Amaral, L. A. N., Scala, A., Barthelemy, M., & Stanley, H. E. (2000). Classes of small-world networks. Procedures of the National Academy of Science, 97, 11149–11152.

This paper looks at a diversity of real-world networks, including such things as an electric power grid, movie actor collaborations, neurons in a worm, world wide web, scientific papers, air traffic routes, a religious group, junior high school friendships, etc. The research led them to conclude that networks can be grouped into three classes: scale-free networks, broad-scale networks, and single-scale networks. The grouping consider such things as how edges decay over time, addition of new e dges, etc. Power law is the relationship between two quantities such that one is proportional to a fixed power of the other.

Liljeros, F., Edling, C. R., Amaral, L. A. N., Stanley, H. E., & Aberg, Y. (2001). The web of human sexual contacts. Nature, 411: 907–908.

No comment

Newman, M. E. J. (2001). The structure of scientific collaboration networks. Proceedings of the National Academy of Science, 98: 404–409.

This paper is similar to the first paper, but it limited its investigation to the study of s cientific published papers and their authors and collaborators. Each scientist would be a vertex and edges would be added for every scientist who was a co-author or collaborator. The graph analysis grouped scientists by discipline and compared and contrasted.

Introduction: Defining Graphs

Data that defines a graph (or network) presents special challenges: data are inherently related, structural measures are of particular interest, inference or path finding is also of interest – tabular questions well suited to SQL like tools, connectivity questions more suited to a graph
Graph G (V, E) consists of:
V – a set of vertices
E – a set of edges, which must connect two vertices, may be directed or undirected

Graph Measures

Size of a Graph - Number of Edges
Order of a Graph - Number of Vertices
Diameter of a Graph - longest shortest path in the graph
Components – subgraphs not connected to the rest of the super graph
Degree of a Vertex – edge count: in-degree -number of incoming edges, out-degree – number of outgoing edges
Eccentricity – longest shortest path from any node to all other nodes
Connected Components – undirected graph (connected components), directed graph (weakly connected components), strongly connected components – all nodes in the component must be

reachable by all other nodes – most graph analyses focus on the “giant” component (largest weakly connected component)

Path Finding

Path Finding – find shortest (weighted) path between two nodes, fundamental to variety of measures (eccentricity, betweenness, diameter), all shortest paths from given node make a spanning tree
Dijkstra’s Algorithm –

- Assign initial distance to all nodes
    - 0 to source
    - Infinity to all others
- Until all paths to destination have been explored
    - Create set of unvisited nodes
    - For all neighbors of particular node, compute new distance
    - Remove them from set of unvisited nodes
    - Compute new shortest path

Dijkstra vs. Bellman-Ford
Dijkstra – naively O(/V/^2) – with min-heap O(/E/ + /V/ log /V/) – fastest algorithm for non-negative edge weights, basis of link-state routing protocols
Bellman-Ford – O (/E/ + /V/) – handles negative weights – scales poorly – basis of distance-vector routing protocols
Spanning Trees – given an undirected graph G(V,E) – spanning tree T is a subgraph G such that all vertices are reachable for any other vertex and there exists only one path between 2 vertices
Minimum Spanning Tree – Spanning Tree with minimum total distance – analogous to routing table

Ranking and Centralities

Rank of a Graph
undirected = /V/ - /components/
directed = cycle rank or distance form directed acyclic graph
PageRank of a Node - Measure of relative importance of nodes, Metric is heart of web search, though much as changed – low cost proxy for centrality – vertex importance is influenced by the importance of its neighbors
Centrality – how important is a vertex to a graph – kinds: degree, closeness, betweenness, Eigenvector
Degree Centrality – how many edges a given vertex has
Closeness Centrality – inverse distance between a vertex and all other vertices
Betweenness Centrality – how many shortest paths include a given vertex
Eigenvector Centrality – components of largest Eigenvectors of G’s adjacency matrix – valuable but expensive

Communities

Community – group of closely connected nodes
Nonoverlapping Community - set of nodes with dense intergroup connections, sparse intragroup connections
Community structures suggest real-world significance – comic book universes or superhero teams – metabolic networks, evolutionally conserved functions, citation networks, research publications
Community Analysis – methods: hierarchical clustering, min-cut, random partitioning to modularity max, label propagation – not all approaches will scale, not all are stable, choose method based on speed vs fidelity, graph size
Bipartite Graphs (k-Partite) – graphs with multiple vertex types – allow us to address more complex questions (example: shortest path from user to product)

Storing Graphs

Markup languages: Graph ML, GML, GraphSON
Adjacency Matrix – make a V x V size matrix, set the cell to non-zero if an edge exists, zero if no edge
Sparse Matrix – matrix with a lot of zeros or empty cells
RDBMS – storing a graph in RDBMS not very efficient on path finding as it requires recursive lookups – make a table with 3 columns and only insert a row if an edge exists
Columnar Database – better for storing sparse matrix

Unit 12 – Serving Data

Han, J., Kamber, M., & Pei, J. (2012). Data mining: Concepts and techniques, (3rd ed.). Morgan Kaufman, Chapter 5, pp. 187–194, 210-218.Read the following sections: 5.1 Optional : 5.2- 5.5

This chapter covers the Data Cube aka OLAP Cubes. It is very relevant and timely information. We actually cover data cubes in the next unit, but the readings are assigned on unit 12 to he lp free up student time at the end of the semester when project work is busy.

Allen, B., Bresnahan, J., Childers, L., Foster, I., Kandaswamy, G., Kettimuthu, R., Kordas, J., Link, M., Martin, S., Pickett, K., & Tuecke, S. (2012). Software as a service for data scientists. Communications of the ACM, 55(2).

This article explains the benefits in general of Software as a Service for Data Scientists. It then goes into the specifics of Globus Online (GO) which is a service to automate and scale transfers of la rge datasets to cloud services such as Amazon Web Services (AWS).

Why Serve Data?

We need to serve data because data in isolation does not propel research or the business forward
Business stakeholders – dashboards, reports
Partner organizations – dashboards or APIs
Production applications – APIs, models, result sets
Other data scientists – Web, Jupyter Notebooks, etc.
Who is the consumer? Human – web, download, mobile, static or dynamic – Machine – fixed or ad hoc? What API? What format? What is the scale? Simultaneous consumers? How much data is processed? How much data is served per request?

Reporting

Storage and retrieval system facilitate two things: application persistence and reporting
Reporting - the presentation of data to support business decision making – more accounting than science - accounting can be complicated – Business Intelligence (BI) tools
Business Intelligence Tools - BI Tools – used by non-data scientists to report to non-data scientists – BI servers
Visual Reporting Tools: Tableau, MicroStrategy, Cognos, Pentaho, Spotfire
Do it yourself: HTML5, d3, APIs

Datasets as a Service

Datasets are increasingly provided as a service to other systems
Driver-based serving – example ODBC – Pros: leverages known drivers, consumer access control can be fine grained – Cons: consumer access control may require lots of attention, consumer may do unexpected things to the system

RESTful serving – useful in service oriented architectures – Pros: API serves only known shapes of data (simpler) – Cons: may require building serving layers on top of storage layer –
example: OData – request a whole dataset, request a specific record, query a dataset, write data, call functions

In-Application Analytics

Functions of In-Application Analytics – present statistics, analytics, recommend content, adapt features to behavior, detect patterns, recognize information, auto-label
Examples: Consumer – Twitter, Uber, Facebook, Shopping, eBay
B2B – marketing, sales, support, advertising
New Features and User Experiences – enabled by analytics and machine learning, to improve user experience through personalization
Engagement – personalized content, timely new content every time a user engages, creating interest to pull users in, presenting results to end users at scale
Layers – Analytics Processing => Serving Layer => Applications

Serving at Scale

Scaling the serving layer is very different from scaling the analytics layer – beneficial to think of them and architect them differently
Measures – transactions per second, queries per second (QPS), latency, response time, throughput, concurrency
Distributing Data Storage – Data Sharding (horizontal partitioning, distributing rows among nodes), Vertical Partitioning (dividing columns among nodes)
Caching Data – storing data in memory – application will check cache for data first, if not found, will retrieve from database and store in cache – issues: personalization (each users has unique set of data), streaming/real-time updates (data updated continuously, caches frequently invalidated)
Distributing Applications – application servers distributed among nodes
Distributing Static Content – Content Delivery Network – CDN – currently used by all major websites, large distributed system of servers deployed in multiple data centers across the world, improves availability and performance by pushing data to the edges (close to end users)
Load Balancers – evenly distribute a work load among nodes

ML Pipeline

Pipelines allow us to focus on model building and evaluation – ML tasks often involve significant data transformation – pipelines are a good candidate for dataflow languages
Transformations are often not well represented in general purpose domain specific languages (DSLs) – vectorization, normalizing, whitening, nonstandard transformations, imputation, interpolation
Feature Extraction – consider the standard process for building bag of words model – nontrivial: language specific processing, tuple extraction, vectorization – needs to be repeatable: many different dataset, corpuses, modeling methods
Feature Unionization – consider a high-dimensional dataset – good features may come from different extraction methods (PCA, maximum linear contrast) – we would like to quickly assemble features from many methods
Grid Searches – often need to cross-validate and to search a space of hyperparameters (time consuming when done by hand) – ML pipelines allow us to search more effectively (assemble a pipeline, apply a grid search to the parameter space) – simpler more consistent cross-validation – more automated approach to finding best fits

Mining Streams

Background Streams vs Database – stream data arrives on its own schedule, database processes know what data they have, different streams arrive at different rates, main constraint with streams is that they need to fit into main memory (how can we filter and sample within memory constraints)
Sampling Streams – example: large data stream that cannot permanently be kept in memory (must be able to query representative sample of stream for accurate answers), stream of even with tuples {user, event, timestamp} (there can be multiple occurrences of the same event, we like to query the fraction of events repeated over the last month, we can keep only 10% of stream's data in memory)
Straightforward Approach – randomly sample 1 of 10 events – does not work – we cannot take a sample of each user's queries, we have to sample users and get each user's complete history – problem: if we have a lot of users, filtering may take up a lot of memory – solution: use hashing, have username to sample and no sample buckets
Bloom Filter
Array of m bits (all set to 0, must be k different hash functions define, each which maps or hashes some set element to one of the m array positions)
To add an element, apply each of the k hash function to get k array positions. Set the bits at all these positions to 1
To query an element (test whether it is in the set), apply each of the k hash functions to get k array positions (if any of the bits are these positions is 0, the element is not set – if all are 1, the element is in the set or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive – false negatives are not possible)

Data Cubes

Business problem – aggregate business data so that it can be analyzed and queried, get overview, drill down for detail, etc.
(see slide diagrams of cubes and operations)
Data Cube – facts (quantitative data) goes inside the cube, dimensions go on the edges –
Slicing – pick one value alone one dimension, creating a cube with one fewer dimensions
Dicing – pick specific values along multiple dimensions, creating a smaller cube (slice is a special case of a dice)
Drill Down, Drill Down, Roll Up – change the level of granularity along a dimension
Pivoting – rotation of a cube for presentation of different views of the data
Technical Problems
If we have time series data, how can we quickly query and explore multi-dimensional aggregates?
How do you compute cubes in an efficient way? (applications can take > 24 hours, not normally distributed, no scale out)
What are the scale issues with data cubes? (processing time, memory limitations)
MOLAP – data stored in multidimensional array, good performance, precomputed, proprietary query language and structures
ROLAP – data stored in relational database, performance depends on underlying query, generally slower than MOLAP, can be partially materialized and partially dynamic computation
Cuboid – a data cube can be viewed as a lattice of cuboids
Computation of Data Cubes - full cube computation of n-dimension cube requires 2^n cuboids thus very expensive
Solutions: only compute cuboids that will be used, only compute cuboids satisfying a defined threshold, compute a set of cuboids from which you can create other cuboids in real time
Iceberg Cubes – contain only those cells of the data cube that meet an aggregate condition, aggregate condition could be: minimum support or lower bound on average, min, or max, purpose is to identify and compute only values that will most likely be required for decision support queries
Pros: computation can be reduced, storage can be reduced
Cons: some queries cannot be answered, difficult to find and identify right threshold, incremental updates are costly (re-compute required)
Cube Shell and Shell Fragments
Assumption – most queries are in a subset of the dimensions
Idea – compute a cube shell of all cuboids of a dimension less than the full cube
Shell Fragments - Cube shells still very expensive, many cuboids to calculate, narrow down to most likely cuboids, certain dimensions never used together, shell fragment is a shell cuboid with fixed dimensions – have a fragment-aware query engine that disallows unsupported queries or generates background processing – common queries answered quickly, uncommon queries must be calculated
Pros: can trade offline and online processing
Cons: identify the right fragments