

INTRO TO DATA SCIENCE

LECTURE 20: TOPICS IN DATA ENGINEERING

KEY CONCEPTS - ENSEMBLE BIG DATA

- It's a much hyped phrase 'big data', but what does it actually refer to?
- The Large Hadron Collider produces about 30 petabytes of data per year
- 1 petabyte is 10^{15} bytes, or 1 million gigabytes
- Their data center contains some 11,000 servers with approximately 100,000 processor cores
- They can process around 1 petabyte of data per day!!

KEY CONCEPTS - CHARACTERISTICS OF BIG DATA

Volume

More data than can fit in memory on your laptop e.g. Amazon user data

Velocity

Faster than standard machines can process e.g. Twitter firehose

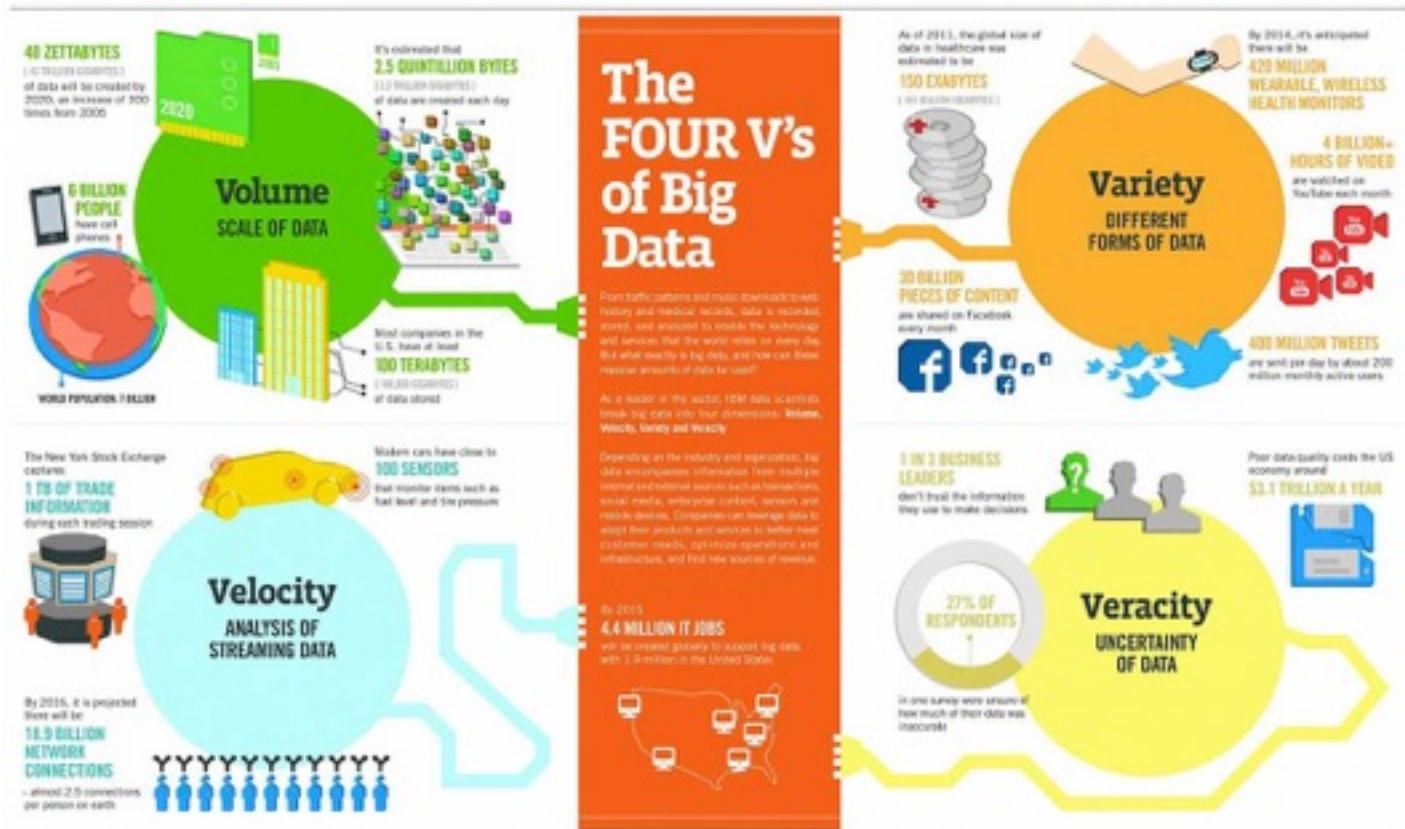
Variety

No single data type e.g. Google's cache of web pages

Veracity

Uncertainty of data e.g. Election night 2012

KEY CONCEPTS - CHARACTERISTICS OF BIG DATA



KEY CONCEPTS - CHARACTERISTICS OF BIG DATA

- Big data means working with data that doesn't 'fit' into a single computer
- We have exponential growth in data AND exponential growth in computing power

DATA SCIENCE

MAP REDUCE & HADOOP

KEY CONCEPTS - MAPREDUCE

- Programming model and implementation that arose out of Google as a means of handling vast quantities of data stored distributively

KEY CONCEPTS - THE MOTIVATION FOR MAPREDUCE

- Let's consider the word frequency problem
- How would you count the words in a word list containing 1000, 10,000, 1,000,000,000 words?

```
def count_words(word_list):  
    count = {}  
    for w in word_list:  
        count[w] += 1  
  
    print sort(count)
```

KEY CONCEPTS - THE MOTIVATION FOR MAPREDUCE

- Counting and sorting are fast, easily scales to ~100 million words
- LIMIT: computer memory
 - need to fit not just the list of words, but the count too
- SOLUTION: stream the words, and process them in batches
- LIMIT: computer memory & speed
 - if the words are all unique then you will still run out of memory
 - it would be rare for all the words to be unique but you just don't know
- SOLUTION: use multiple cores, with multiple streams
- LIMIT: bandwidth

KEY CONCEPTS - THE MOTIVATION FOR MAPREDUCE

- SOLUTION: compress and/or use better data structures (heaps, hashing)
- This should allow for the processing of 10 trillion words on a single computer
- LIMIT: the single computer
- SOLUTION: use 10 computers. Break the problem up so that each computer counts its share of words before sending their results to a controlling computer. The controller 'adds' the results together and solves the problem
- LIMIT: 100 trillion words
- SOLUTION: use 100 computers.

KEY CONCEPTS - THE MOTIVATION FOR MAPREDUCE

- LIMIT: the fan-in to the controller. The bandwidth will not be enough
- SOLUTION: re-organize the network shape, e.g. a tree. 10 machines report to 1 controller, 10 controllers report to a super-controller, etc
- LIMIT: hardware will fail
- SOLUTION: introduce fault tolerance. Replicate the input and introduce redundancy. Embed checksums to ensure the integrity of the data. Introduce error detection. Introduce oversight - error detection and auto-restart

The first 10 computers are easy
The first 100 computers are hard
The first 1000 computers are impossible

KEY CONCEPTS - THE MOTIVATION FOR MAPREDUCE

- 2004 paper by Sanjay Ghemawat, and Jeffrey Dean: MapReduce: Simplified Data Processing on Large Clusters.
- Both worked for Google at the time
- Allows us to use 1000 computers at one time, and handles the fault tolerance problem
- In our word count story the underlying premise was to bring the data to the algorithm
- MapReduce redefine that paradigm by allowing the data to reside on it's local machine and distributing algorithms to process it

KEY CONCEPTS - MAPREDUCE - WHAT IS IT?

- You have to write 2 functions:

1. A mapper function : filter and transforms data - takes each data point and returns an ordered (key, value) pair

A framework exists that sorts the output of the mappers via the 'shuffle'. Collates all the keys that match. Sends key piles to the reducers. 'Partitioner'

2. A reducer function: outputs a new ordered (key, new_value), where new_value is some aggregate function of the old values

KEY CONCEPTS - MAPREDUCE - SIMPLE EXAMPLE

1. Mappers (running on many machines accessing their local data)
2. red- >("red",1), red->("red",1), blue->("blue",1), red->("red", 1)....
3. These fan-in to the 'Shuffle'
4. A pile of ("red", 1)s become ("red", 1, 1, 1)
5. The output of the 'Shuffle' is then sent down to the reducers
6. ("red",1,1,1) -> ("red", 3)

KEY CONCEPTS - MAPREDUCE - SIMPLE EXAMPLE

- One reducer handles all the values for a specific key
- You can add more mappers, add more reducers
- Each computer has no (and needs no) knowledge of the other computers doing the work

KEY CONCEPTS - MAPREDUCE - SO WHAT'S THE DOWNSIDE?

- Converting an algorithm to a series of MapReduce steps is often unintuitive
- The above word count algorithm works well if there is a good distribution of different words
- Counting the same word would push the all of the work into a single 'Shuffle'. Lots of values of a single key
- Need to real-time monitor MapReduce jobs
- By monitoring performance MapReduce jobs can be optimized. Nanoseconds is significant where petabytes of data are concerned.

KEY CONCEPTS - MAPREDUCE - A BETTER EXAMPLE?

- You are running an Internet operation
- Petabytes of timestamped event data, which contains information about user's actions on the website
- {userID, IPaddress, Zip, add_seen, did_they_click}
- QUESTION:
- 1. CTR by zip

KEY CONCEPTS - MAPREDUCE - A BETTER EXAMPLE?

- **Mapper function 1:** key = {zip}, value = {clicked on ad, saw add}
- (78929, {1,1}), (78929, {0,1}), (20817, {0,1})...
- Reducer handling 78929 produces (78929, {1, 2})
- and eventually something like (78929, {700, 16337})
- You end up with {zip, clicks, impressions}

KEY CONCEPTS - MAPREDUCE - MACHINE LEARNING ALGORITHMS

- The MapReduce framework can be used to implement a variety of ML algorithms
- Linear and logistic regression, K-means
- http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2006_725.pdf
- MapReduce handles:
 - parallelization & distribution,
 - partitioning fault tolerance,
 - I/O scheduling, status, and
 - monitoring

KEY CONCEPTS - MAPREDUCE - WHAT CAN'T IT DO?

- Iterate
 - Gradient descent convergence methods
- Any other problems?
 - Speed

KEY CONCEPTS - HADOOP

- Google developed the GFS, or Global File System to support MapReduce
- Hadoop is the open source version of Google's MapReduce, and has the HDFS, or Hadoop Distributed File System
- HDFS uses large files with block sizes of 64 or 256MB. The blocks are replicated to nodes in the cluster.
- The master node is aware of the other nodes and notices if a node dies.

KEY CONCEPTS - HADOOP

- Hadoop is written in Java, Google's MapReduce is C++
- Hadoop has a streaming utility that allows client code to be written in any language (including Python)
- Cloudera is like the Red Hat for Hadoop

KEY CONCEPTS - HDFS

- It is a distributed file system - across machines
- Highly fault tolerant
- Designed to be deployed on low-cost hardware
- Can store very large files across machines
- A cluster is a group of computers that work together and can be considered as a single system
- A node is a machine in the cluster

KEY CONCEPTS - HDFS

- NameNode = Runs on a master node that tracks and directs the storage of the cluster
- DataNode = Runs on slave nodes that make up the majority of the machines in the cluster. Instructs data files to be split into blocks, each of which are replicated 3 times and stored on machines across the cluster. Fault tolerance.
- Client machine = neither a NameNode or DataNode, but has Hadoop installed. Responsible for loading data into the cluster, submitting MapReduce jobs and viewing the results once the jobs are complete

KEY CONCEPTS - HDFS

- Hadoop has been demonstrated on a cluster of up to 4000 nodes
- Sort performance on 900 nodes is considered good
- Sorting 8 TB of data on 900 nodes takes around 1.8 hours

KEY CONCEPTS - APACHE HIVE

- The open source implementation is done under the aegis of the Apache Software Foundation
- A data warehousing system sits on top of Hadoop, called Apache Hive
- Hive implements an SQL-like language which produces the mapping and reducing functions for you
- Apache Mahout is a collection of ML libraries and command-line tools that interface with Hadoop



KEY CONCEPTS - PIG

- There are also other frameworks overlying MapReduce
- Another popular program, and an alternative to Apache Hive, is Pig Latin
- Pig Latin is built on Hadoop and provides a relational data-flow language (an alternative to tradition SQL)



KEY CONCEPTS - HIVE VS PIG

- This is a good post that compares Pig and Hive:

<https://developer.yahoo.com/blogs/hadoop/comparing-pig-latin-sql-constructing-data-processing-pipelines-444.html>

- Horton Works offers an environment where you can try Pig and Hive for free
- They have good interactive tutorials that you can work your way through

<http://hortonworks.com/>

KEY CONCEPTS - WHAT'S THE MOTIVATION?

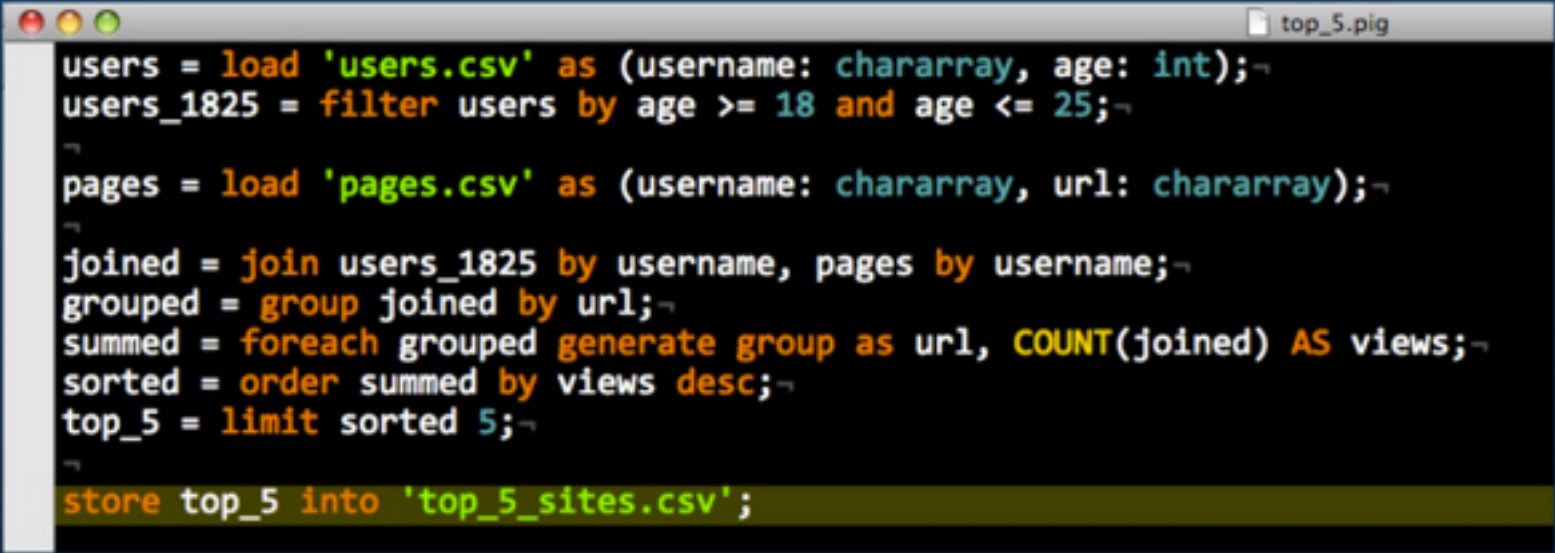
<http://www.slideshare.net/kevinweil/hadoop-pig-and-twitter-nosql-east-2009>

Why Pig?

- Because I bet you can read the following script.

KEY CONCEPTS - WHAT'S THE MOTIVATION?

A Real Pig Script



```
users = load 'users.csv' as (username: chararray, age: int);  
users_1825 = filter users by age >= 18 and age <= 25;  
  
pages = load 'pages.csv' as (username: chararray, url: chararray);  
  
joined = join users_1825 by username, pages by username;  
grouped = group joined by url;  
summed = foreach grouped generate group as url, COUNT(joined) AS views;  
sorted = order summed by views desc;  
top_5 = limit sorted 5;  
  
store top_5 into 'top_5_sites.csv';
```

- ▶ Now, just for fun... the same calculation in vanilla Hadoop MapReduce.

KEY CONCEPTS - WHAT'S THE MOTIVATION?

No, seriously.

[illegible]

DATA SCIENCE

SPARK

KEY CONCEPTS - SPARK

- With distributed systems like Hadoop we have tools for collecting, storing and processing information at scale
- Having access to all this data is one thing, being able to do something useful with it is something entirely different
- Doing something useful means placing a schema over the data and using, say, SQL, to answer questions like: “of the millions of users who made it to the third page of our registration process, how many were over 25?”

KEY CONCEPTS - SPARK

- In point of fact “doing something useful” means more than just SQL
- We want richer functionality in areas like ML and statistics

KEY CONCEPTS - SPARK

- The sorts of things we have been doing with data in our iPython notebooks are the sorts of things we would like to do to leverage the huge data sets residing on our HDFS, which is spread over our computing cluster
- One idea might be to push our ML framework out onto the machines, having rewritten them to work well in the distributed setting
- The problem is that many algorithm have wide data dependencies, and network transfer rates are considerably slower than memory access

KEY CONCEPTS - SPARK

- There is a need, therefore, for a programming paradigm that is, by its very nature, highly parallel
- Genomics has had HPC, or high-performance computing for decades.
- The problems with HPC is in it's difficulty in use

KEY CONCEPTS - SPARK

The Hadoop ecosystem gives the mainstream user many of the underlying features of HPC at a far cheaper cost

So how do we do Data Science?

KEY CONCEPTS - SPARK

1. Most of our work is in pre-processing data.

- You will have to pre-process data that you cannot inspect directly.
- Pre-processing becomes even more computationally dependent

2. Iteration is fundamental to Data Science.

- There is a requirement for multiple passes over the data.
- Experimentation is part of the deal - feature selection, algorithm selection, hyper-parameter optimization all require investigation.

3. The task isn't over when a well performing model has been built.

- Models become part of a production service, and as such, may need to be rebuilt both initially and periodically.
- The model's performance over time must be monitored, and tweaked.
- Production code is usually Java or C++.
- If the original modeling code could be actually used in the app that would ideal.
- There is a need for a REPL, or Read-Evaluate-Print Loop environment.

KEY CONCEPTS - APACHE SPARK

- open source framework
- combines an engine to distribute programs across clusters of machines with an elegant model for writing programs atop of it.

Arguably the first open source software that makes distributed programming truly accessible to data scientists!!

KEY CONCEPTS - SPARK - CORE COMPONENTS

- **Spark Core:**

- Contains the basic functionality of Spark; in particular the APIs that define Resilient Distributed Datasets (RDDs) and the operations and actions that can be undertaken upon them.
- The rest of Spark's libraries are built on top of the RDD and Spark Core.

KEY CONCEPTS - SPARK - CORE COMPONENTS

- **Spark SQL:**

- Provides APIs for interacting with Spark via the Apache Hive variant of SQL called Hive Query Language (HiveQL).
- Every database table is represented as an RDD and Spark SQL queries are transformed into Spark operations.
- For those that are familiar with Hive and HiveQL, Spark can act as a drop-in replacement.

KEY CONCEPTS - SPARK - CORE COMPONENTS

- **Spark Streaming:**

- Enables the processing and manipulation of live streams of data in real time.
- Many streaming data libraries (such as Apache Storm) exist for handling real-time data.
- Spark Streaming enables programs to leverage this data similar to how you would interact with a normal RDD as data is flowing in.

KEY CONCEPTS - SPARK - CORE COMPONENTS

- **MLlib:**

- A library of common machine learning algorithms implemented as Spark operations on RDDs.
- This library contains scalable learning algorithms like classifications, regressions, etc. that require iterative operations across large data sets.
- The Mahout library, formerly the Big Data machine learning library of choice, will move to Spark for its implementations in the future.

KEY CONCEPTS - SPARK - CORE COMPONENTS

- **GraphX:**

- A collection of algorithms and tools for manipulating graphs and performing parallel graph operations and computations.
- GraphX extends the RDD API to include operations for manipulating graphs, creating subgraphs, or accessing all vertices in a path.

KEY CONCEPTS - SPARK

- Spark maintains MapReduce's linear scalability (as data increases, you add more hardware, jobs complete in roughly the same amount of time), and fault tolerance, but extends it in 3 ways:
 1. Does not rely on a rigid map-then-reduce paradigm. It's engine can execute a directed acyclic graph (DAG) of operators
 2. It complements 1. with a rich set of transformations that allow users to express computation more naturally - has a strong developer focus
 3. It has in-memory processing. It has RDD or Resilient Distributed Data abstraction, which means that future steps requiring the same data do NOT need to recompute or reload from disk

KEY CONCEPTS - SPARK

- Highly suited for iterative algorithms and algorithms requiring multiple data passes.
- The overall aim is to bring data science into a single programming environment.
- It aims to be the Python of big data!!
- It's native language is Scala, but it has a well developed Python API, called Pyspark
- Using Scala you can do the entire data science pipeline. You do not need to query in SQL, then analyze in R or Python. From a user perspective you it is invisible as to where the data resides or is being processed!

KEY CONCEPTS - SPARK FEATURES

- Read and write data in all of the data formats supported by MapReduce - Avro, Parquet, and CSV
- Read and write to NoSQL dBs like HBase and Cassandra.
- Spark streaming can ingest data continuously from systems like Flume, and Kafka.
- SparkSQL can interact with the Hive Metastore, and will be able to use the underlying execution engine of Hive (as an alternative to MapReduce)
- It can run inside Yarn, Hadoop's scheduler and resource manager

KEY CONCEPTS - SPARK - A WORK IN PROGRESS

- BUT it is still a work in progress.
- It hasn't surpassed MapReduce as the workhorse of batch processing
- Stream processing, SQL, ML, and graph processing all undergoing continuous development
- Has nowhere near the R, Python Stack statistics and modeling functionality
- SQL is rich, but still lags behind Hive

KEY CONCEPTS - SPARK PROGRAMMING MODEL

- Writing a spark program usually consists of:
 1. Defining a set of transformations on input data sets
 2. Invoking actions that output the transformed data sets into persistent storage or return results to the driver's local memory
 3. Running local computations that operate on the results computed in a distributed fashion. These can help you decide what transformations and actions to undertake next

DATA SCIENCE

DATABASES

KEY CONCEPTS - DATABASES

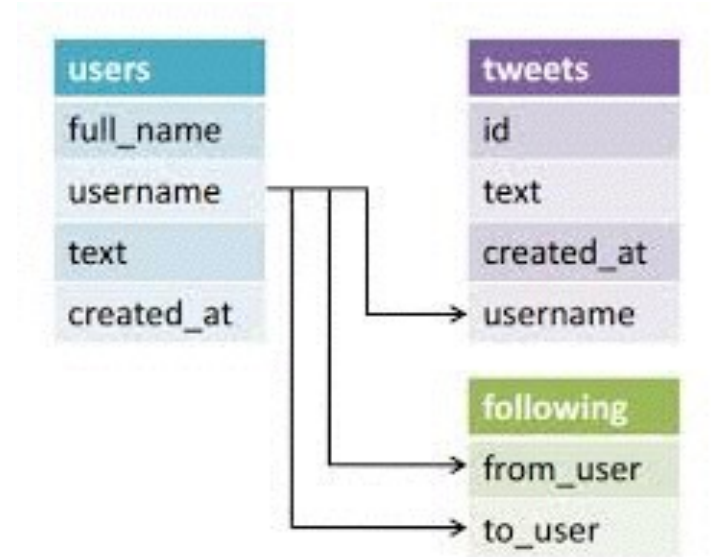
- As data scientists you need to know something about databases
- SQL knowledge is almost mandatory
- <http://sql.learncodethehardway.org/>

KEY CONCEPTS - DATABASES

<i>Relational, or SQL</i>	<i>PostgreSQL, SQLite, MySQL</i>
<i>Non-Relational, or NoSQL</i>	
<i>Key-Value Pair (KVP)</i>	<i>Riak, Redis, MemcacheDB</i>
<i>Document</i>	<i>MongoDB, CouchDB, RavenDB</i>
<i>Columnar</i>	<i>HBase, Cassandra, DynamoDB</i>
<i>Graph</i>	<i>Neo4J, OrientDB, Apache Giraph, Titan</i>
<i>Spatial</i>	<i>OpenGEO, PortGIS, ArcSDE</i>
<i>Cloud</i>	<i>IAAS, VM Image, DAAS</i>

KEY CONCEPTS - DATABASES - RELATIONAL (SQL) DATABASES

- Data is stored in tables
- A 'schema' defines the tables, the fields or variables within a table, and the relationships between tables
- SQL or Structured Query Language is used to interrogate the database
- Most commonly used language for creating, retrieving, updating and deleting data in a relational database (CRUD = create, retrieve, update, delete)



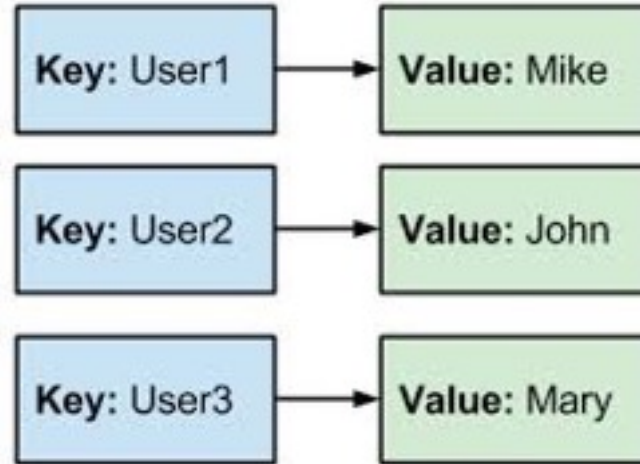
KEY CONCEPTS - DATABASES - RELATIONAL (SQL) DATABASES

		<i>Pros</i>	<i>Cons</i>
<u>SQLite</u>	<ul style="list-style-type: none">• <i>powerful, embedded RDBMS</i>	<ul style="list-style-type: none">• <i>lightweight</i>• <i>mostly fully-functional DB</i>• <i>good for prototyping, testing</i>	<ul style="list-style-type: none">• <i>lightweight- size restricted to 2GB</i>• <i>can be slow</i>
<u>MySQL</u>	<ul style="list-style-type: none">• <i>most popular and commonly used open-source RDBMS</i>	<ul style="list-style-type: none">• <i>good for read-heavy applications</i>• <i>easy to work with</i>• <i>good security</i>• <i>scalable</i>	<ul style="list-style-type: none">• <i>not 100% SQL compliant</i>• <i>not suited for high-concurrent (high read/write) apps</i>
<u>PostgreSQL</u>	<ul style="list-style-type: none">• <i>most advanced, SQL-compliant and open-source objective RDBMS</i>	<ul style="list-style-type: none">• <i>big community</i>• <i>extensible</i>• <i>can handle complex procedures</i>	<ul style="list-style-type: none">• <i>not suited for read-heavy operations</i>• <i>not as easy as MySQL to administer</i>• <i>not easy to set up replication</i>

KEY CONCEPTS - DATABASES - SHARDING

- Sharding is a design principle where the rows of a database table are held separately (also referred to as horizontal partitioning), usually on different servers
- Each server has the same table structure
- The database is logically split up into shards
- But each complete record exists only in the one shard, which allows for full CRUD operations within the shard
- Reduces the number of rows, which can improve performance
- Usually try to shard on a real-world aspect of the data, e.g. Australian vs European customers

KEY CONCEPTS - DATABASES - KEY-VALUE STORAGE

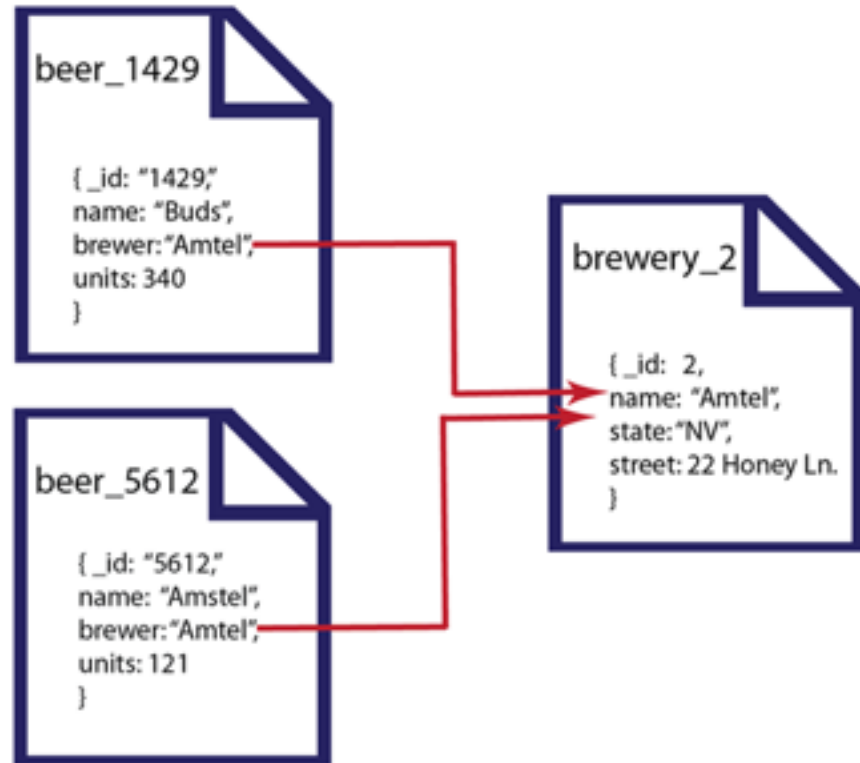


KEY CONCEPTS - DATABASES - KEY-VALUE STORAGE

		<i>Pros</i>	<i>Cons</i>
<u>Redis</u>	<ul style="list-style-type: none">• <i>Fast, in-memory</i>• <i>Good for caching</i>• <i>Partition Tolerance</i>• <i>Master-Slave Replication</i>	<ul style="list-style-type: none">• <i>fast!!</i>• <i>lots of great built-in data structures</i>• <i>easy to administer</i>• <i>good community support</i>	<ul style="list-style-type: none">• <i>sharding, failover not yet fully realized</i>
<u>Riak</u>	<ul style="list-style-type: none">• <i>Primarily used for extreme high availability</i>	<ul style="list-style-type: none">• <i>fault tolerant</i>• <i>good at replication (DR)</i>• <i>good support from Basho</i>• <i>open-source edition</i>• <i>tunable trade-offs for distribution and replication</i>	<ul style="list-style-type: none">• <i>be ready to pay for cross data center replication (enterprise)</i>

KEY CONCEPTS - DATABASES - DOCUMENT ORIENTED

- Really a sub-class of key-value storage paradigm
- XML databases are a specific sub-class of document oriented databases that are optimized to extract their metadata from XML documents



KEY CONCEPTS - DATABASES - DOCUMENT ORIENTED

		<i>Pros</i>	<i>Cons</i>
<u>MongoDB</u>	<ul style="list-style-type: none">• <i>Arguably most popular of NoSQL DBs</i>• <i>Schema-free document store</i>	<ul style="list-style-type: none">• <i>master/slave replication with failover</i>• <i>large community of users</i>• <i>good support from MongoDB (10gen)</i>• <i>auto-sharding</i>	<ul style="list-style-type: none">• <i>scaling with write-heavy applications can get tricky</i>• <i>don't use built-in map-reduce!</i>
<u>Couchbase</u>	<ul style="list-style-type: none">• <i>High performance key-value/document store with flexible, but slow, indexes</i>	<ul style="list-style-type: none">• <i>master-master replication</i>• <i>replication supports filtering or selective replication</i>• <i>write operations do not block reads</i>	<ul style="list-style-type: none">• <i>not as popular as Mongo,</i>• <i>smaller community of support</i>

KEY CONCEPTS - DATABASES - COLUMNAR DATABASES

- In a relational database we store in rows
- It is easy to find information about any given row, but hard to answer queries about an aggregate
- In this example we could find an individual salary very easily and quickly
- But it is more difficult to answer questions about salaries alone, e.g. how many people have salaries between 50 and 75 thousand dollars?

<i>User_ID</i>	<i>Name</i>	<i>Salary</i>
<i>101</i>	<i>Smith</i>	<i>100000</i>
<i>102</i>	<i>Jones</i>	<i>75000</i>
<i>103</i>	<i>Barnes</i>	<i>55000</i>
<i>104</i>	<i>Jacobs</i>	<i>110000</i>

KEY CONCEPTS - DATABASES - COLUMNAR DATABASES

- In a columnar database each column forms a storage unit
- Performing aggregated calculations on Salary now become very efficient, because we not longer have to scan over the rows to find the data in the salary column
- The relationship between columns is maintained, i.e. the 3rd entry in User_ID goes with the 3rd entry in the Salary and Name columns

<i>User_ID</i>
<i>101</i>
<i>102</i>
<i>103</i>
<i>104</i>

<i>Salary</i>
<i>100000</i>
<i>75000</i>
<i>55000</i>
<i>110000</i>

<i>Name</i>
<i>Smith</i>
<i>Jones</i>
<i>Barnes</i>
<i>Jacobs</i>

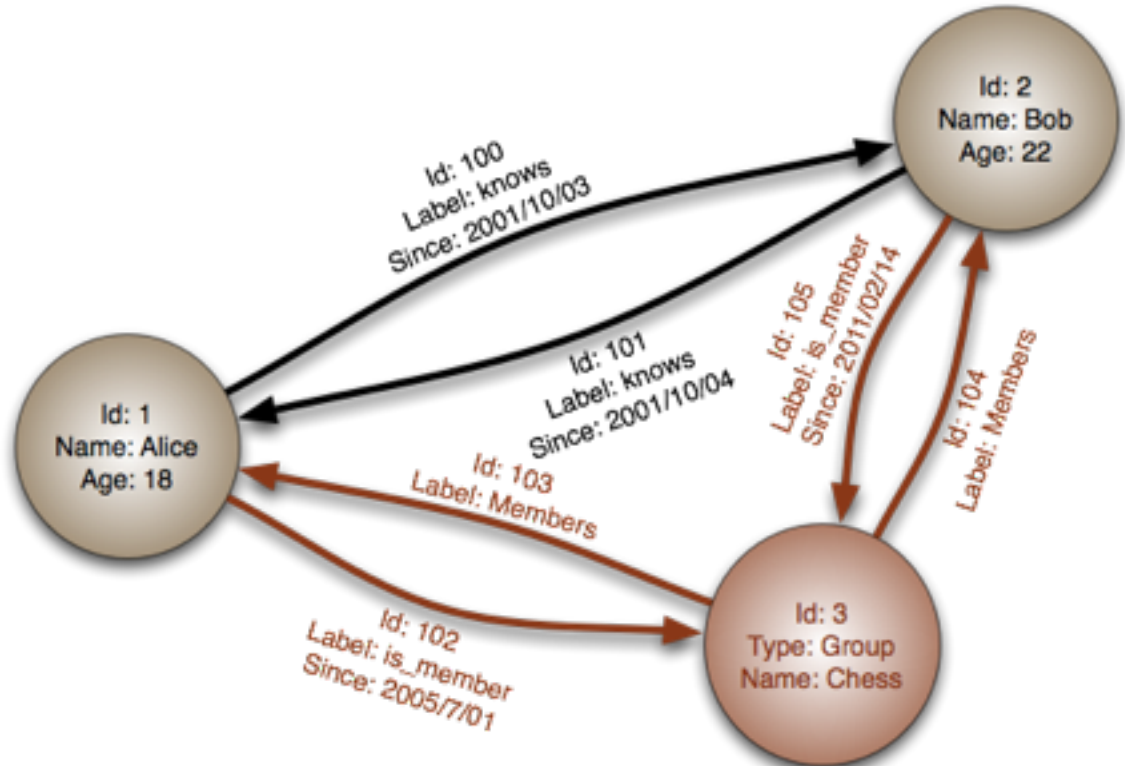
KEY CONCEPTS - DATABASES - COLUMNAR DATABASES

		<i>Pros</i>	<i>Cons</i>
<u>Cassandra</u>	<ul style="list-style-type: none">• <i>Store huge datasets in "almost" SQL</i>• <i>Based on DynamoDB</i>	<ul style="list-style-type: none">• <i>tunable trade-offs for distribution and replication</i>• <i><u>excellent</u> at cross datacenter replication</i>• <i>good for write-heavy applications</i>	<ul style="list-style-type: none">• <i>not suited for read-heavy applications</i>• <i>learning curve for efficient usage</i>• <i>best to have enterprise support</i>
<u>HBase</u>	<ul style="list-style-type: none">• <i>Based on Google's BigTable</i>• <i>Billions of rows by millions of columns</i>	<ul style="list-style-type: none">• <i>uses HDFS as storage</i>• <i>map/reduce with Hadoop</i>• <i>best if you use the Hadoop/HDFS stack already</i>	<ul style="list-style-type: none">• <i>lots of "moving parts" (e.g. zookeeper)</i>• <i>dependent on HDFS, Hadoop</i>• <i>complex to administer</i>

KEY CONCEPTS - DATABASES - GRAPH DATABASES

Graph databases

- everything is stored as a node, an edge or an attribute



KEY CONCEPTS - DATABASES - GRAPH DATABASES

		<i>Pros</i>	<i>Cons</i>
<u>Neo4j</u>	<ul style="list-style-type: none">• <i>“World’s leading graph database”</i>• <i>Native graph processing</i>• <i>open source</i>• <i>java</i>	<ul style="list-style-type: none">• <i>good at graph-style interconnected data</i>• <i>path finding</i>• <i>optimized for reads</i>• <i>clustering, replication, caching, online backup</i>	<ul style="list-style-type: none">• <i>graph DBs still relatively immature</i>• <i>query-syntax has a learning curve</i>• <i>relatively small user community</i>