Programming with Big Data

Tim Savage

February 25, 2017

New York City

De-Mystifying the Un-Mystical

Like a hammer and nails,
these are just tools

My Background

- I started in the pre-digital age using small, administrative data.
- NLSY79: 3,000 young men tracked annually for about 20 years.
- Complicated algorithm to examine whether early adverse labor market outcomes affected their “lives”.
- The data was cheap, but the computation was very expensive.

Programming Languages

- Basic
- Fortran
- SAS/Stata
- R
- Python

Core Takeaways

1. If it can be digitized, it can be analyzed. And now everything is digitized.
2. The core components of big data did not simply appear a few years ago, but their confluence created the problems (and the solutions).
3. The open source computing community defeated Gresham's "law". Without open source, there would be no big data.
4. There is a reason Google won the search engines wars. Spark has brought clustered computing to the masses.

What is Big Data?

- “Big Data” is data whose scale, distribution, diversity, and/or timeliness requires the use of new computing architectures and analytical tools that essentially did not exist 10 years ago.
- Organizations are beginning to derive benefit from analyzing ever larger and more complex data sets in real time.

Key Characteristics

- Volume, which has increased nearly 44 times since 2009.
- Variety of different data structures to mine and analyze.
  - Structured, semi-structured, and unstructured.
- Processing complexity because of changing data structures.

Business Drivers (Good and Bad)

- Optimizing operations for profitability and efficiency.
- Identifying potential risks from customer churn or fraud.
- Predicting new opportunities.
- Regulatory compliance.

The Challenge

- High-value data is hard to reach and leverage.
- Sits in fragmented “puddles” without a proper data “lake”.
- Predictive analytics are the last step in the chain.
- Breaking down ad-hoc and isolated analytics projects.

Big Data Analytics Lifecycle

1. Preparing: Is there enough good data to be potentially useful.
2. Planning: There are many ML models to choose from. Which one(s) to use?
3. Building: Is the model robust? Is it appropriate?
4. Operationalizing: Scale the model(s) for deployment using the ideas discussed in this presentation.

Common Big Data Challenges

- Volume
  - Too much data to process conventionally.
  - Reduce processing time by using distributed and parallel computing. Example: Performing OCR on thousands of articles simultaneously.
  - Too big to fit into memory when no clustered resources available. Example: Given 30GB of NYC taxi data, how would one calculate the average or median fare?
- Velocity
  - Data arrives in real-time, but it cannot be stored or processed in real time.

Coding Matters

Python Example

The Volume Problem: Parallelization

- Utilize parallel computing architectures, such as multiple cores, multiple processors or clusters of machines.
  - Multiple cores for video processing.
  - Multiple processors for web servers and video games.
  - Clusters for simulations. (What I do.)

Task Parallelism

- Distributing tasks to run simultaneously, achieving efficiency if the number of tasks is large.
  - Data cleaning.
  - Running linear regression or algorithms that can be distributed, such as decision trees/random forests.
- It is basically Henry Ford's assembly line:
  - A bunch of boxes at FedEx need both shipping labels and barcode scanning. One worker can stamp the shipping labels, while the other scans the boxes.

Pipeline Parallelism

- Explicitly allocating resources for each phase of the data processing pipeline, achieving efficiency when the number of phases is large.
- Processes must communicate throughout the pipeline, so it is a combination of task and streaming.
- The assembly line analogy: pass the boxes by the stamper and then by the scanner.

Data Parallelism

- Distributing data to different processors that run simultaneously, achieving efficiency based on the number of processor nodes.
- The assembly line analogy: leave all boxes on the ground, where each worker can both stamp and scan a single box.
  - Then get more workers.

The Velocity Problem: Scaling

- Scale up by making the process scalable on a single computer.
  - Reduce the amount of data processed or the resources needed to perform the processing.
  - Increase the computing resources using parallel processing and faster memory and/or storage.
  - Improve the efficiency and/or performance of the process by, for example, better coding.
- Scale out by adding more computing resources through the networking of multiple computers.

Too Much Data

- Scale up with “streaming”.
- An active area of research among “computing scientists.” (They are touchy about this distinction.)
- Started in the 1970’s, but now very popular because of its successful application to massive data processing.
- Big data world adopts the stream processing model.
  - Process data as soon as it arrives.
  - Continuous and incremental processing.

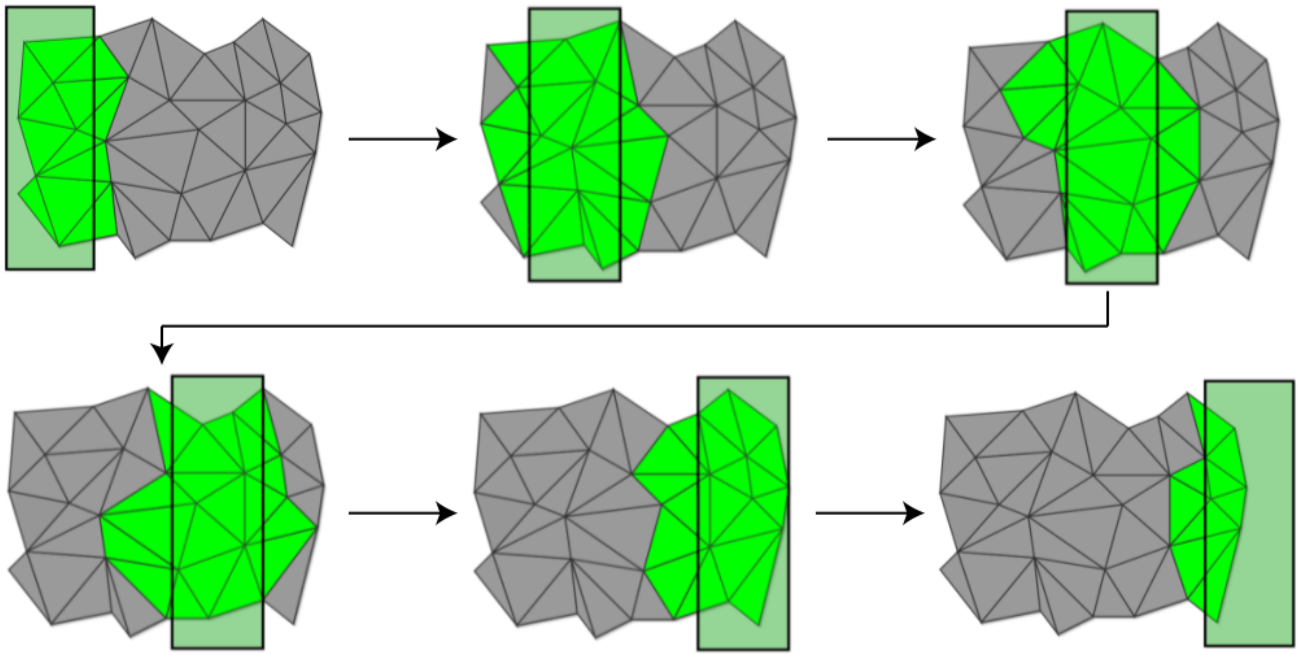
Streaming Computation

- Given a data series of n elements, [a_1, a_2, …, a_n], that can only be examined in a limited number of passes, typically one.
- Compute a function of the stream, such as an average, a median, or a histogram.
- Primary constraints:
  - Limited working memory of size m (m << n).
  - Elements are accessed sequentially.

Handling Data Streams

- There are many flavors of the streaming model:
  - Time series (HFT price data).
  - Cash register (storing incremental counts and totals).
  - Turnstile (arrival-departure summarization).
  - Sliding window (keeping a continuous but fixed subset of the input).
- There are many classes of techniques to process data elements:
  - Sampling (reduce data inputs).
  - Sketching (data aggregation).
  - Counting (data compression).

Picturing the Sliding Window



Example: Detecting Omission

- There are 11 football players, numbered 1 to 11, walking from the locker room to the field.
- Only 10 arrive: 8, 2, 6, 1, 10, 3, 5, 11, 9, and 7.
- How to determine the missing number in the "stream" of players?
- Given the constraints:
  - We can only look at one number at a time.
  - We can only store one number in our head.

What Is Omitted?

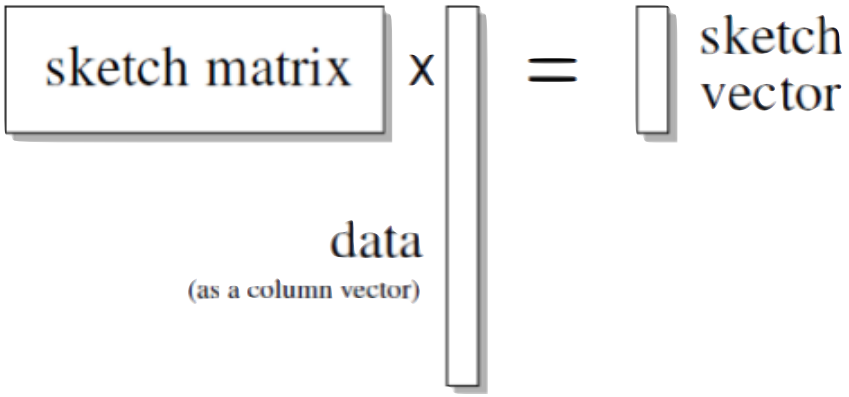
- It is 4. How?
- The sum of all numbers is fixed: (1+11)*11/2 = 66. (Simple mathematical formula of sums.)
- Record only the sum of the numbers as we scan through the stream of players.
- 8, 10, 16, 17, 27, 30, 35, 46, 55, and 62.
- Our missing number is 66 - 62 = 4.
- Simple example that highlights many important ideas about processing constraints and exploiting “lazy” knowledge.

Sampling

- Motivation: a small random sample of the data can be a good representation.
- Action: sample the data based on a probability model, which is a challenge for data streams of unknown size.
- Useful for showing patterns but not at detection deviations from central tendencies.

Sketching

- Motivation: only certain pieces of the data are needed for computation.
- Action: project data into a “sketch” space, progressively building the function of stream.
- Useful for aggregation, but the sketch vector could be very large.



Sketching Examples

- Computing a streaming average by sketching the (sum, count) pair.
  - For each element, add the incremental value to the sum and increase the count.
- Computing a streaming histogram by sketching the count per category.
  - For each element, add to the count of its category.
  - Lions, tigers, and bears.
- Mirrors simple game theory of tit for tat.

For Streaming, Use Approximation

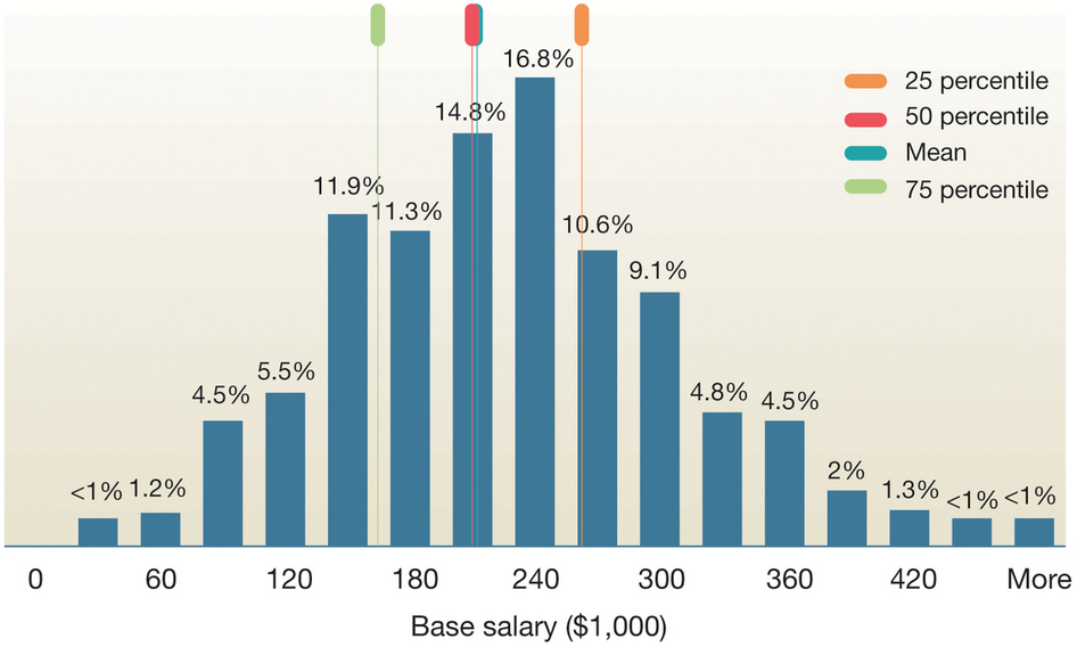
- We need to speed up the computation.
  - Brute force approach often takes a too much processing time.
  - As a result, we sacrifice accuracy or granularity for efficiency.
- Common approaches:
  - Memoization for repeated queries or computation.
  - Indexing for quickly retrieving a small subset of data.
  - Data cubes for computing aggregation.

Example: Calculating a Median

- A median is the middle mark and can differ wildly from an average, but tends to be stable. Example: median income of your customer.
  - Well-studied problem in streaming algorithms.
  - Lots of methods to approximate the median value within tolerable error bounds.
  - But computationally expensive, needing multiple passes through the data.
- Most of these methods rely on assuming that data are continuous and of large range (such as income).

Example: Calculating a Median

- Build a histogram of values based on bucketed ranges.
- Select the 50th percentile bucket.
- Choose the first bucket that passes the that mark.
- In this case, $210,000.



Streaming and Big Data Queries

- Exact answers are not required for real-time decision making, and results can be pre-computed. This can be done in a streaming fashion (as the data arrives).
  - Yields more accurate answers than sampling.
  - Storage is not expensive.
- This approach is used at Google, Twitter, and Facebook.
- Twitter's open source "Summingbird" toolkit:
  - HyperLogLog (count distinction): the number of unique users who perform a certain action.
  - Count-Min Sketch (hashing): the number of times a particular query is requested.
  - Bloom Filters (membership test): keep track of users who have been exposed to an event.

MapReduce
Hadoop

Divide and Conquer

Why MapReduce?

- Big data is too large to handle using conventional means.
- Sooner or later, there are energy limits on scaling up a given machine. But we can add more machines by scaling out.
- MapReduce paradigm allows us to scale out.
  - Issue is: what's going to herd all these cats?
  - Google's paradigm won them the search-engine wars.

What is MapReduce?

- A programming paradigm for big data processing.
  - Data is split into distributed chunks.
  - Transformations are performed on the chunks, running in parallel.
- MapReduce is scalable by adding more machines to process distributed chunks.
- It is the foundation for "Hadoop," which is a specific implementation of MapReduce.

Map(and)Reduce

- A programming paradigm that processes data in two phases/operations: map() and then reduce().
  - instead of map(), filter(), reduce() in Python, we are limited to only map() and reduce().
- In a nutshell, that is all there is:
  - User provides a data collection of separable records.
  - User applies a map function to each data record, such as a count.
  - User reduces the mapped output with another user-defined function, if needed.

Python Example

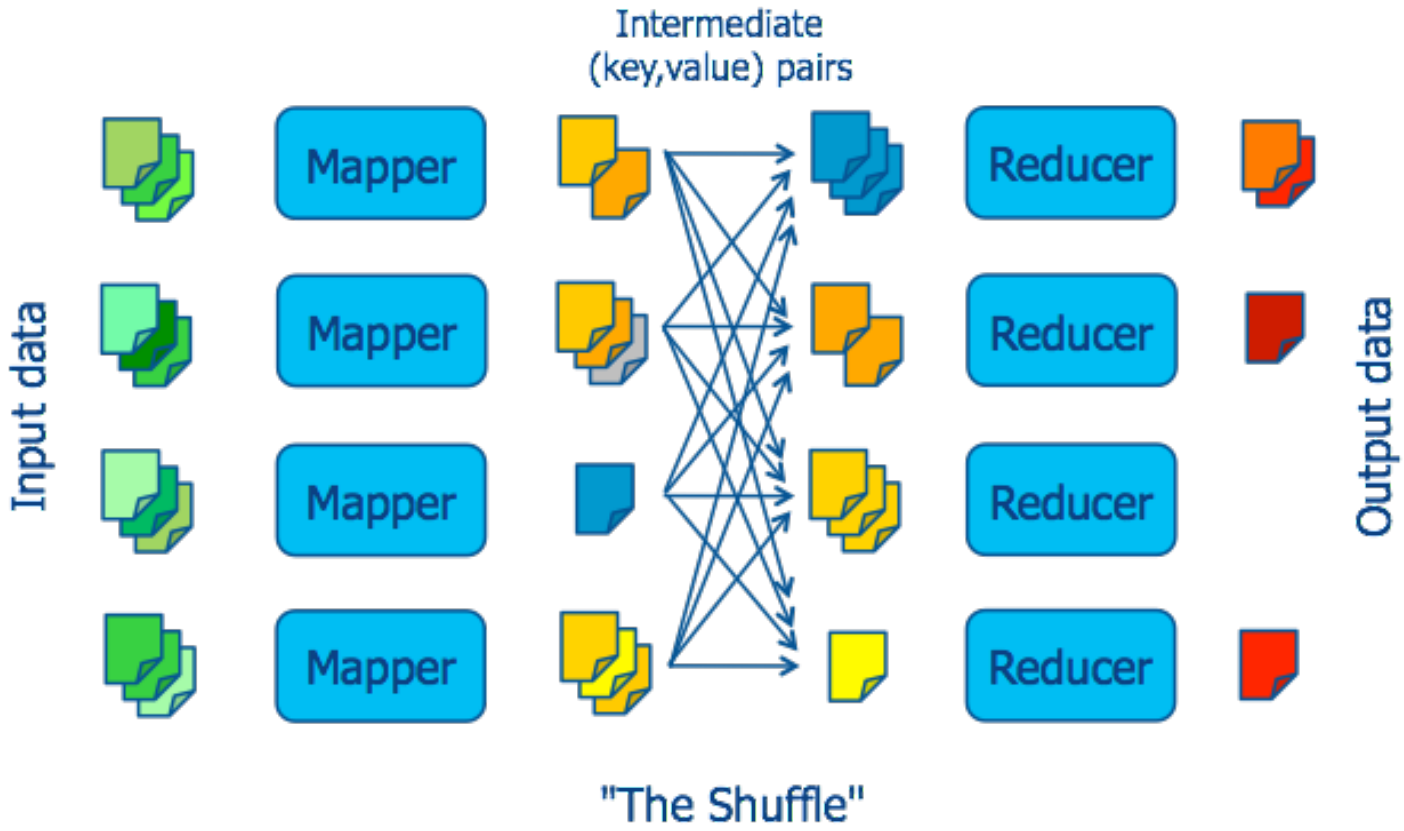
Input

- Data must be separable into records.
  - Lines of text or rows of a spreadsheet.
  - CSV works easily, but not true for other data types, such as JSON and XML.
- Key/value pairs
  - Key = line number or record index.
  - Value = text string or row data.

Phases

- Map phase: transform each input record with a user-defined function.
- Shuffle (and sort) phase: complicated but it amounts to ensuring stuff lines up properly. (This is herding cats).
- Reduce phase: Transform the output of the shuffle phase with another user-defined function. (May or may not be necessary.)

MapReduce Dataflow



When to use MapReduce?

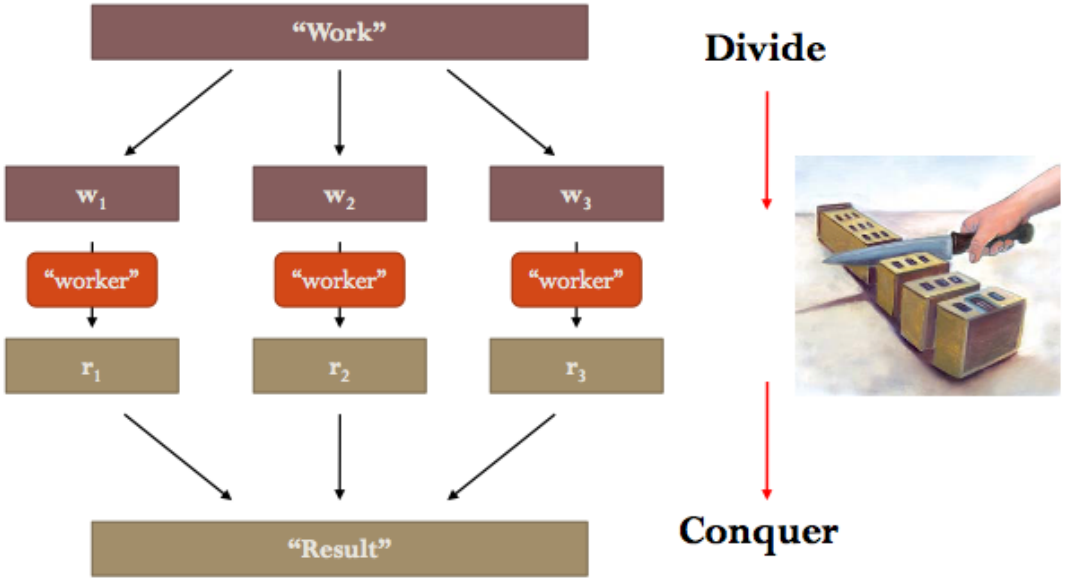
- When there are efficiencies to be gained from the types of parallelization we discussed earlier.
- In other words, whenever you have big data.
- Data must be “split-able” into chunks and records.

Designing MapReduce Algorithms

- User must decide what is to be done by map and separately by reduce.
  - Map can act on individual key-value pairs, but it cannot look at other key-value pairs.
  - Reduce can aggregate data by looking at multiple values, as long as map has properly mapped them.
  - If reduce needs to look at several values together, map must emit them using the same key.
  - Never easy to herd cats.

MapReduce for Big Data

- Data parallelism: by scaling out:
  - Divide and Conquer
- Distributed computing:
  - Data on different machines



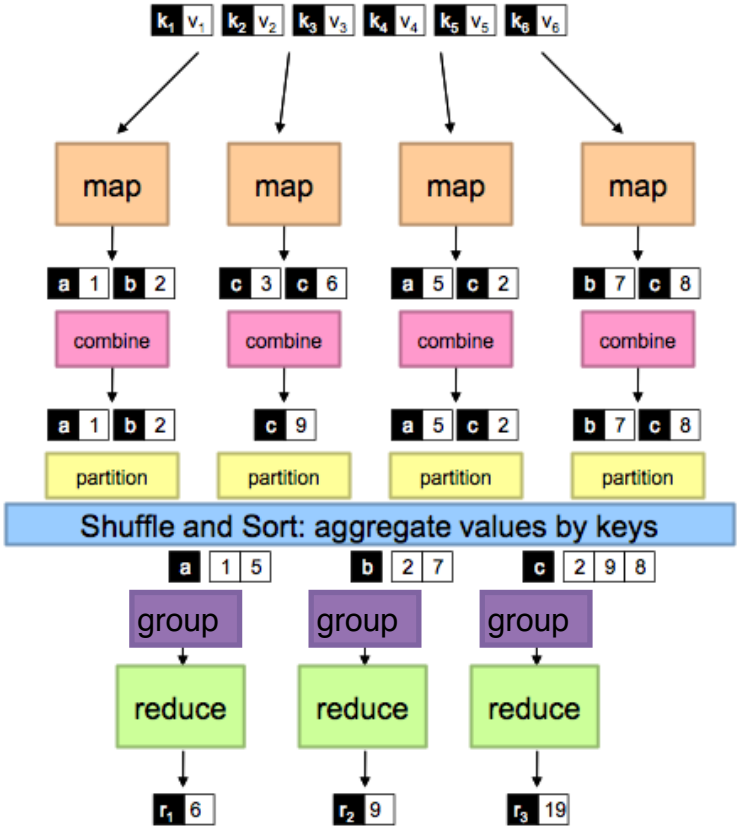
What is Hadoop?

- Hadoop is used in common parlance to describe:
  1. The MapReduce Paradigm.
  2. Massive unstructured data storage.
  3. HDFS: the Hadoop distributed file system.
- In other words:
  - Hadoop = HDFS + MapReduce.
  - Hadoop = Big Data + Analytics.

(H)DFS

- Files are divided into chunks (typically 64MB in size).
- Chunks are replicated at different compute nodes.
- Chunk size and the degree of replication are chosen by the user.
- A special file (the master node) stores, for each file, the positions of its chunks.
- So-called “master/slave” architecture.
- This is an important element of properly herding the cats.

Hadoop and MapReduce



Hadoop Runtime Tasks

- Handles scheduling.
  - Assigns workers to map and reduce tasks.
- Handles data distribution.
  - Gets data to the workers.
- Handles synchronization.
  - Gathers, sorts, and shuffles intermediate data.
- Handles errors and faults.
  - Detects worker failures and restarts.
- Everything happens on top of a distributed file sharing system.

Hadoop Operation Modes

- Java MapReduce Mode.
  - Write Mapper, Combiner, Reducer functions in Java using Hadoop Java APIs.
  - Read records one at a time.
- Streaming Mode.
  - Any statistical computing language, such as Python.
  - Input can be a line at a time or a stream at a time.

How I Use It: Simulation

- One of the most powerful simulation tools is the Markov chain Monte Carlo (MCMC), a technique that can be massively scaled.
- Re-ignited Bayesian approaches to analysis and inference, rapidly displacing frequentist approaches based on the non-existent idea of “in repeated samples.”
- We will start with a calendar to motivate MCMC.
- We will use a drunken sailor to highlight its properties.

Where I Started: Spark

- Brings clustered computing to the masses.
- Native APIs for R, Python and SQL.
- Massively extended the MapReduce paradigm so that folks like me can use it in everyday practice. (And so can you.)
- The distributed computing environment is ideal for simulation, including MCMC.
- A simple example.

Some Musings

- In the old days, data were relatively cheap, but the computation was expensive.
- The reverse is now true. Good data are relatively very expensive, but computation is pocket change.
- All of this is developing very rapidly, and a sleeping giant has shaken the big data space.
- (But everyone talks their book.)

Thank You