

A Brief History of Big Data

John Urbanic

Parallel Computing Scientist
Pittsburgh Supercomputing Center

Distinguished Service Professor
Carnegie Mellon University

Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate.

—*Wikipedia*

Once there was only small data...



A classic amount of "small" data

Find a tasty appetizer – Easy!

Find something to use up these oranges – grumble...

What if....



Less sophisticated is sometimes better...



“Chronologically” or “geologically” organized.
Familiar to some of you at tax time.

Get all articles from 2015.

Get all papers on “fault tolerance”
– grumble and cough

Indexing will determine your individual performance.
Teamwork can scale that up.



The culmination of centuries...



Find books on Modern Physics (DD# 539)

Find books by Wheeler

where he isn't the first author – grumble...



Your only hope...



Then data started to grow.

1956 IBM Model 350



5 MB of data!

But still pricey. \$

Better think about what
you want to save.

And finally got BIG.

10TB for \$150

Whys:

Storage got cheap

So why not keep it all?

Today data is a hot commodity \$

And we got better at generating it

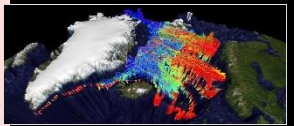
= Facebook
= Deep Learning
= IoT
= Science... **10 TB** *



Pan-STARRS



Genome sequencers
(Wikipedia Commons)



Horniman museum:
<http://www.horniman.ac.uk>
[get_involved/blog/bioblitz-insects-reviewed](http://www.horniman.ac.uk/get_involved/blog/bioblitz-insects-reviewed)

Wikipedia Commons

mentions a more accurate 208TB, and in

http://www.arctic.noaa.gov/report11/biodiv_whales_walrus.html

*Actually, a silly estimate. The original reference mentions a more accurate 208TB, and in 2013 the digital collection alone was 3PB.



A better sense of biggish

Size

- 1000 Genomes Project
 - AWS hosted
 - 260TB
- Common Crawl
 - Hosted on Bridges
 - 300-800TB+

Throughput

- Square Kilometer Array
 - Building now
 - Exabyte of raw data/day – compressed to 10PB
- Internet of Things (IoT) / motes
 - Endless streaming

Records

- GDEL (Global Database of Events, Language, and Tone) (also soon to be hosted on Bridges)
 - Only about 2.5TB per year, but...
 - 250M rows and 58 fields (BigTable)
 - *“during periods with relatively little content, maximal translation accuracy can be achieved, with accuracy linearly degraded as needed to cope with increases in volume in order to ensure that translation always finishes within the 15 minute window.... and prioritizes the highest quality material, accepting that lower-quality material may have a lower-quality translation to stay within the available time window.”*

3 V's of Big Data

- Volume
- Velocity
- Variety

Good Ol' SQL couldn't keep up.



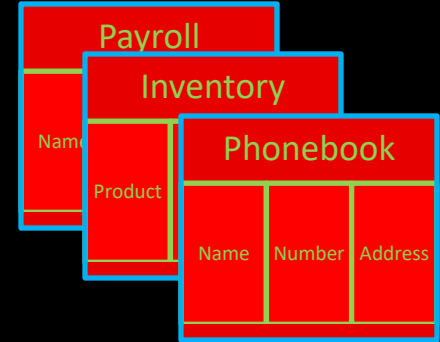
Oracle



```
SELECT NAME, NUMBER, FROM PHONEBOOK
```

Why it *wasn't* fashionable:

- Schemas set in stone:
 - Need to define before we can add data
 - Not a fit for *agile development*
"What do you mean we didn't plan to keep logs of everyone's heartbeat?"
- Queries often require accessing multiple indexes and joining and sorting multiple tables
- Sharding isn't trivial
- Caching is tough
 - ACID (Atomicity, Consistency, Isolation, Durability) in a *transaction* is costly.



So we gave up: Key-Value

Redis, Memcached, Amazon DynamoDB, Riak, Ehcache

```
GET foo
```

- Certainly agile (no schema)
- Certainly scalable (linear in most ways: hardware, storage, cost)
- Good hash might deliver fast lookup
- Sharding, backup, etc. could be simple
- Often used for “session” information: online games, shopping carts

```
GET cart:joe:15~4~7~0723
```

foo	bar
2	fast
6	0
9	0
0	9
text	pic
1055	stuff
bar	foo

How does a pile of unorganized data solve our problems?

Sure, giving up ACID buys us a lot performance, but doesn't our crude organization cost us something? Yes, but remember these guys?



This is what they look like today.



Document



GET foo

- Value must be an object the DB can understand
- Common are: XML, JSON, Binary JSON and nested thereof
- This allows server side operations on the data

GET plant=daisy

- Can be quite complex: Linq query, JavaScript function
- Different DB's have different update/staleness paradigms

foo	
2	<pre><CATALOG> <PLANT> <COMMON>Bloodroot</COMMON> <BOTANICAL>Sanguinaria canadensis</BOTANICAL> <ZONE>4</ZONE> <LIGHT>Mostly Shady</LIGHT> <PRICE>\$2.44</PRICE> <AVAILABILITY>031599</AVAILABILITY> </PLANT> <PLANT> <COMMON>Columbine</COMMON> <BOTANICAL>Aquilegia canadensis</BOTANICAL> <ZONE>3</ZONE> <LIGHT>Mostly Shady</LIGHT> <PRICE>\$9.37</PRICE> <AVAILABILITY>030699</AVAILABILITY> </PLANT> </pre>
6	JSON
9	XML
0	Binary JSON
bar	JSON XML
12	XML XML

Wide Column Stores



```
SELECT Name, Occupation FROM People WHERE key IN (199, 200, 207);
```

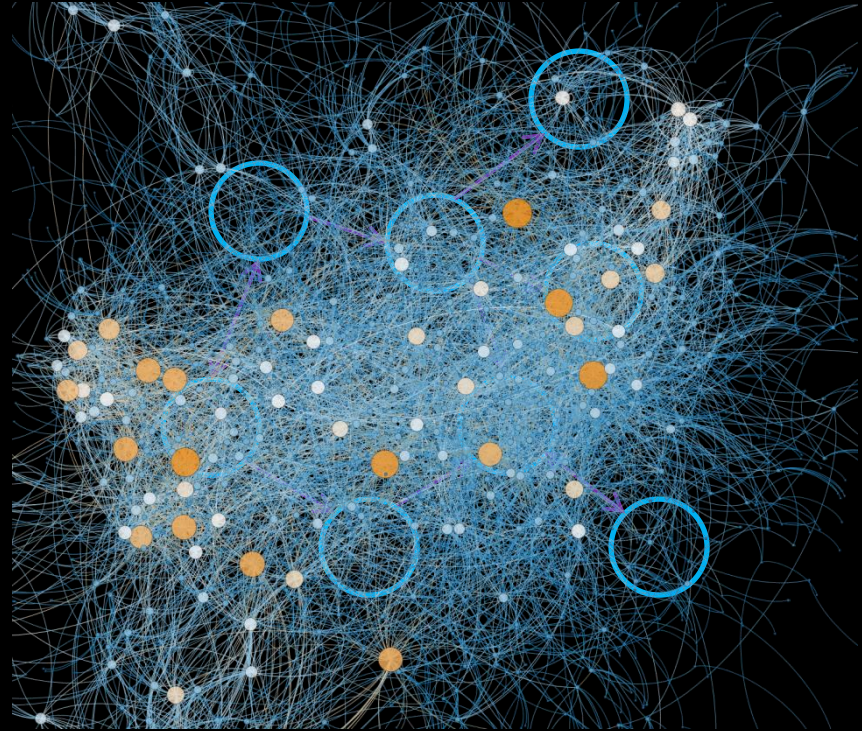
- No predefined schema
- Can think of this as a 2-D key-value store: the value may be a key-value store itself
- Different databases aggregate data differently on disk with different optimizations

Key			
Joe	Email: joe@gmail	Web: www.joe.com	
Fred	Phone: 412-555-3412	Email: fred@yahoo.com	Address: 200 S. Main Street
Julia	Email: julia@apple.com		
Mac	Phone: 214-555-5847		

Graph

Neo4j Titan, GEMS

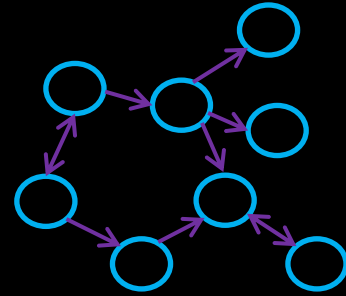
- Great for semantic web
- Great for graphs 😊
- Can be hard to visualize
- Serialization can be difficult
- Queries more complicated



From [PDX Graph Meetup](#)

Queries

SPARQL, Cypher



SPARQL (W3C Standard)

- Uses Resource Description Framework format
 - triple store
- RDF Limitations
 - No named graphs
 - No quantifiers or general statements
 - “Every page was created by some author”
 - “Cats meow”
- Requires a schema or *ontology* (RDFS) to define rules
 - "The object of 'homepage' must be a Document."
 - "Link from an actor to a movie must connect an object of type Person to an object of type Movie."

```
SELECT ?name ?email
WHERE {
    ?person a foaf:Person.
    ?person foaf:name ?name.
    ?person foaf:mbox ?email. }
```

Cypher (Neo4J only)

- No longer proprietary
- Stores whole graph, not just triples
- Allows for named graphs
- ...and general Property Graphs (edges and nodes may have values)


```
SMATCH (Jack:Person
        { name:'Jack Nicolson' })-[ :ACTED_IN ]-(movie:Movie)
RETURN movie
```

Graph Databases

- These are not curiosities, but are behind some of the most high-profile pieces of Web infrastructure.
- They are definitely *big* data.

Microsoft Bing Knowledge Graph	Search and conversations.	~2 billion primary entries ~55 billion facts
Facebook		~50 million primary entries ~500 million assertions
Google Knowledge Graph	Search and conversations.	~1 billion entries ~55 billion facts
LinkedIn graph		590 million members 30 million companies

Hadoop & Spark



What kind
of databases
are they?

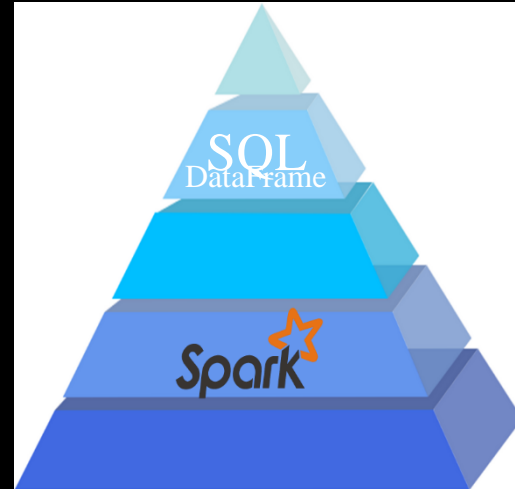
Frameworks for Data

These are both frameworks for distributing and retrieving data. Hadoop is focused on disk based data and a basic map-reduce scheme, and Spark evolves that in several directions that we will get in to. Both can accommodate multiple types of databases and *achieve their performance gains by using parallel workers.*



The mother of Hadoop was necessity. It is trendy to ridicule its primitive design, but it was the first step.

We have repurposed many of these blocks to build a better framework.

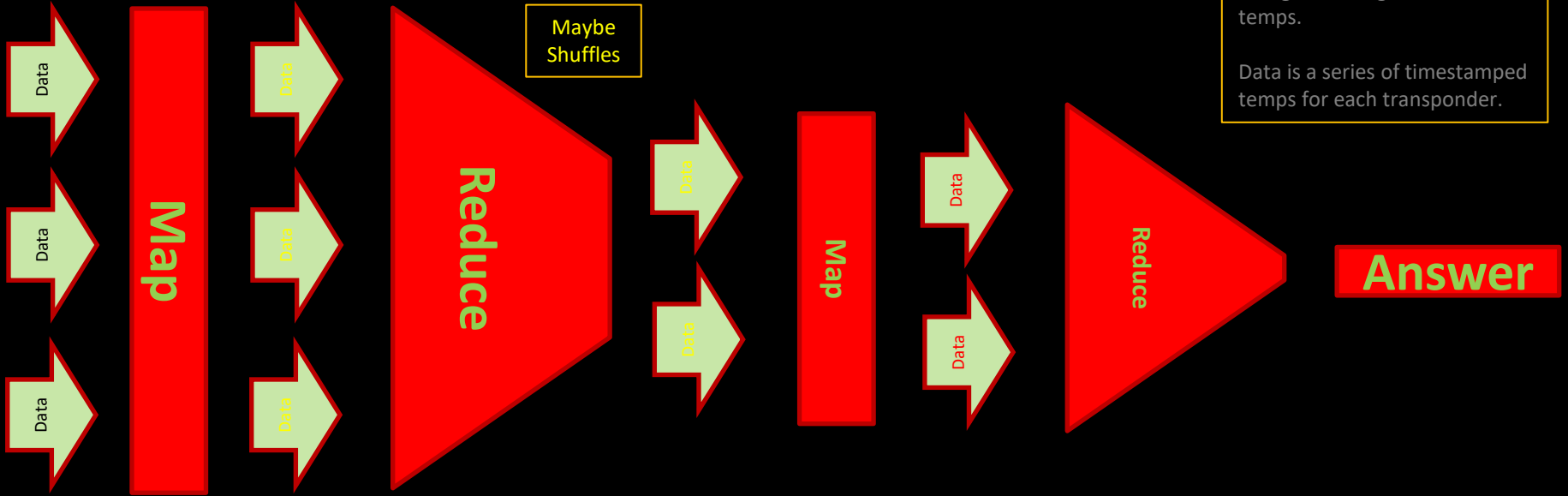


What exactly is this Hadoop "framework"?

- Programming platform
- Distributed filesystem
- Parallel execution environment
- Software ecosystem

Programming = MapReduce

In Place



Transponder ID -> Geo Coordinates
00154301 -> 59.33, 177.60
04435354 -> 56.71, 171.73
04539340 -> 25.18, -118.89

Only keep data for Bering Sea
00154301 -> 59.33, 177.60
04435354 -> 56.71, 171.73

Find biggest change
at each transponder
in last 24h

00154301 -> 30
04435354 -> 5

Keep any over 20
degrees

00154301 -> 30

Ex: Need to find any recent big swings in Bering Sea surface temps.

Data is a series of timestamped temps for each transponder.

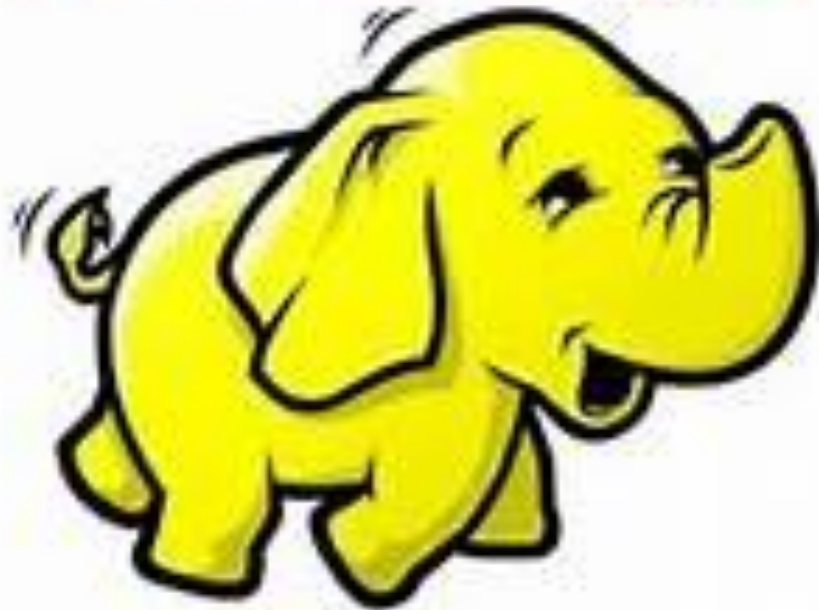
Answer

HDFS: Hadoop Distributed File System

- Replication
- Write Once Read Many (WORM)
- Requires underlying filesystem
- But very friendly to set up

Hadoop Ecosystem Lives On

hadoop



And lots
more...