A Brief History of Big Data

John Urbanic
Parallel Computing Scientist
Pittsburgh Supercomputing Center
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...

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 2007.

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.

8TB for $130

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 *

*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
- GDELT (Global Database of Events, Language, and Tone) (also soon to be hosted on Bridges)
  - Only about 2.5TB per year, but...
  - 250M rows and 59 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.

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

- 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 foo
```

```
<table>
<thead>
<tr>
<th>foo</th>
<th>bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>fast</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>text</td>
<td>pic</td>
</tr>
<tr>
<td>1055</td>
<td>stuff</td>
</tr>
<tr>
<td>bar</td>
<td>foo</td>
</tr>
</tbody>
</table>
```
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.
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
Wide Column Stores

- 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

<table>
<thead>
<tr>
<th>Key</th>
<th>Email</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>joe@gmail</td>
<td><a href="http://www.joe.com">www.joe.com</a></td>
</tr>
<tr>
<td>Fred</td>
<td>Phone: 412-555-3412</td>
<td>Email: <a href="mailto:fred@yahoo.com">fred@yahoo.com</a></td>
</tr>
<tr>
<td>Julia</td>
<td>Email: <a href="mailto:julia@apple.com">julia@apple.com</a></td>
<td></td>
</tr>
<tr>
<td>Mac</td>
<td>Phone: 214-555-5847</td>
<td></td>
</tr>
</tbody>
</table>
Graph

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

From *PDX Graph Meetup*
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."

\[
\text{SELECT } ?\text{name} ?\text{email} \\
\text{WHERE} \{ \\
\text{?person a foaf:Person.} \\
\text{?person foaf:name } ?\text{name.} \\
\text{?person foaf:mbox } ?\text{email.} \} \\
\]

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.

|                        | Search and conversations. | ~2 billion primary entries
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Bing KG</td>
<td></td>
<td>~55 billion facts</td>
</tr>
</tbody>
</table>
| Facebook               |                           | ~50 million primary entries
|                        |                           | ~500 million assertions  |
| Google KG              | Search and conversations. | ~1 billion entries       |
|                        |                           | ~55 billion facts        |
| LinkedIn               |                           | 590 million members      |
|                        |                           | 30 million companies     |

Noy, Goa, Jain. Communications of the ACM, August 2019
What kind of databases are they?
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 into. 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
Ex: Need to find any recent big swings in Bering Sea surface temps.

Data is a series of timestamped temps for each transponder.

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
HDFS: Hadoop Distributed File System

- Replication
  - Failsafe
  - Predistribution
- Write Once Read Many (WORM)
  - No Random Access (contrast with RDBMS)
- Requires underlying filesystem
Using Hadoop

1. Load data to HDFS
2. Write a program
   - Java (compile/jar/run)
   - Hadoop Streaming
     - Mapper and reducer scripts read/write stdin/stdout
     - Use built-in utilities (wc, grep, cat)
     - Write in any language (python)
3. Submit a job
Shakespeare Using Hadoop
(from the future)

Uses *cat* as mapper and *wc* as reducer:

```
hadoop jar \\
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \\
    -input /datasets/plays/ -output streaming-out \\
    -mapper '/bin/cat' -reducer '/usr/bin/wc -l'
```
Hadoop Ecosystem Lives On

And lots more...