Big Data Paradigm

By: Hadachi
Outline

• Introduction
• Motivation
• Why Big Data
• Key Enablers
• Big Data
  – Technologies
  – Schema
  – Characteristics
• Tools and Techniques
Introduction
Introduction
Introduction
Introduction

Ship Transport routes
Introduction
Introduction

Facebook
Connections
Introduction

Facebook
Human Migration
Introduction

Source: OpenCellID

Mobile Network Data

Visitation frequency of each location for a single user

Home

Work

Visitation frequency

Source: OpenCellID

Location id

Visitation frequency

Location id
## Introduction

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kilobytes</td>
<td>KB</td>
<td>$10^3$</td>
</tr>
<tr>
<td>Megabyte</td>
<td>MB</td>
<td>$10^6$</td>
</tr>
<tr>
<td>Gigabyte</td>
<td>GB</td>
<td>$10^9$</td>
</tr>
<tr>
<td>Terabyte</td>
<td>TB</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>Petabyte</td>
<td>PB</td>
<td>$10^{15}$</td>
</tr>
<tr>
<td>Exabyte</td>
<td>EB</td>
<td>$10^{18}$</td>
</tr>
<tr>
<td>Zettabyte</td>
<td>ZB</td>
<td>$10^{21}$</td>
</tr>
<tr>
<td>Yottabyte</td>
<td>YB</td>
<td>$10^{24}$</td>
</tr>
</tbody>
</table>

This is Big Data?
Introduction

Data is considered to be Big Data the moment it is difficult to process it with the traditional systems.
Motivation

• Growth of digital data

THE WORLD'S CAPACITY TO STORE INFORMATION
This chart shows the world’s growth in storage capacity for both analog data (books, newspapers, videotapes, etc.) and digital (CDs, DVDs, computer hard drives, smartphone drives, etc.)

In gigabytes or estimated equivalent

1986 ANALOG 2.62 billion
1993 ANALOG STORAGE

1996 DIGITAL 0.02 billion

COMPUTING POWER
In 1986, pocket calculators accounted for much of the world's data-processing power.

Percentage of available processing power by device:

<table>
<thead>
<tr>
<th>Year</th>
<th>Pocket Calculators</th>
<th>Personal Computers</th>
<th>Video Game Consoles</th>
<th>Servers, Mainframes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986</td>
<td>41%</td>
<td>33%</td>
<td>9%</td>
<td>17%</td>
</tr>
<tr>
<td>2007</td>
<td>66%</td>
<td>25%</td>
<td>3%</td>
<td>6%</td>
</tr>
</tbody>
</table>

2007 ANALOG 18.86 billion gigabytes
- Paper, film, audiotape and vinyl: 6.2%
- Analog videotapes: 93.8%

2007 DIGITAL
- Other digital media: 0.8%
- Portable media players, flash drives: 2%
- Portable hard disks: 2.4%
- CDs and minidisks: 6.8%
- Computer servers and mainframe hard disks: 8.9%
- Digital tape: 11.8%
- DVD/Blu-ray: 22.8%
- PC hard disks: 44.5%
- Mobile phones, PDAs: 0.3%
- Supercomputers: 0.3%

2007 TOTAL 276.12 billion gigabytes

*Other includes chip cards, memory cards, floppy disks, mobile phones/PDAs, cameras/camcorders, video games

Source from the Washington Post,
http://media3.washingtonpost.com/wp-dyn/content/graphic/2011/02/11/GR2011021100614.jpg
Motivation

• Searching
• Storing
• Capturing
• Analyzing
• Transferring
• Visualizing
• Etc.

'Mining Insights from Raw Big Data'
Motivation

Data Growth
Driven by Unstructured Data

Sourced from Oracle: Big data
Motivation

- Stored Data

Source: “NO MORE SECRETS with Big Data Analytics” by Jaap Bloem et al.
Why Big Data

• Data
  – Diversity of the nature of the data
    • Social media, videos, web analytics, sensors, etc.

• Technology
  – IT solutions
  – Computer-processing power
Key Enablers: Data Availability

- Data Availability per Activity Sectors through job offers.
Key Enablers: Storage Capacity

Data storage has grown significantly, shifting markedly from analog to digital after 2000
Global installed, optimally compressed, storage

NOTE: Numbers may not sum due to rounding.
Key Enablers: Computation capacity
The Gains Behind Big Data

Some sectors are positioned for greater gains from the use of big data

Historical productivity growth in the United States, 2000–08

1 See appendix for detailed definitions and metrics used for value potential index.

Big-Data and Emerging Technologies
Big Data Schema

Big Data Production
- RFID
- Analytical DB
- Social Media
- Web Analytics
- Log Files
- CDR
- Sensors

Big Data Management Systems and Preprocessing
- BD Integration
- BD Quality

Storage
Processing
Filtering

Transforming Big Data into actionable information

Big Data Consumption
- Mining
- Analytics
- Enrichment
- Search
- ...
Big Data is a case of extreme information management

- Insights and Information
- Validation
- Analytics
- Technology
- Mining
- Classification
- Exploration
- Privacy

Volume

Velocity

 Variety

Complexity
Main Characterization of Big-Data
Characterization of Big-Data

• The four V of Big Data

- Volume: It’s estimated that 2.5 quintillion bytes of data are created each day. By 2020, an increase of 100 times from 2005.
- Velocity: Modern cars have close to 100 sensors that monitor items such as fuel level and tire pressure.
- Variety: As of 2011, the total size of data in healthcare was estimated to be 150 exabytes.
- Veracity: In one survey, 27% of respondents were unsure of how much of their data was accurate.
Popularity of Big Data on the Web

• Open Google trends
  – https://www.google.com/trends/
  – Let’s check the popularity of:
    • Big Data, Data Mining, Semantic Web, Machine Learning, Classification.
Popularity of Big Data on the Web
Popularity of Big Data on the Web
Popularity of Big Data on the Web

Google Trends
Tools and techniques
Tools usually used in Big Data

• NoSQL
  – Databases MongoDB, CouchDB, etc.
• MapReduce
  – Hadoop, Hive, etc.
• Storage
  – S3, Hadoop Distributed File System
• Servers
  – Google App engine, Heroku, etc.
• Processing
  – R, Google Search, etc.
NoSQL
MapReduce

- MapReduce is a programming model designed for large data
  - parallel,
  - distributed algorithm on a cluster.
MapReduce
Storage

Hadoop Distributed File System
Servers

• Google App Engine
  – GAE is a Platform as a Service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers.

Google App Engine lets you run web applications on Google's infrastructure.

Easy to build.
Easy to maintain.
Easy to scale as the traffic and storage needs grow.

Yes, free for up to 1 GB of storage and enough CPU and bandwidth to support 5 million page views a month. 10 Applications per Google account.
• **Google App Engine**

  » Programming languages support

  • App Engine runs JAVA apps on a JAVA 7 virtual machine (currently supports JAVA 6 as well).
  • Uses JAVA Servlet standard for web applications:
    • WAR (Web Applications ARCHive) directory structure.
    • Servlet classes
    • Java Server Pages (JSP)
  • Local development servers are available to anyone for developing and testing local applications.
  • Only whitelisted applications can be deployed on Google App Engine.

  • Uses WSGI (Web Server Gateway Interface) standard.
  • Python applications can be written using:
    • Webapp2 framework
    • Django framework
    • Any python code that uses the CGI (Common Gateway Interface) standard.

  • Go is an Google’s open source programming environment.
  • App can be written using Google’s Go SDK.
Processing

• Google Search Engine

Google-bot

---

Crawler

Storage

Web

Search Interface

Algorithms

Indexing

Ranking

Trash

Indexes

Indexes

Indexes
Processing

• Google search engine
  – Consist of a set of services
    • Crawling: to locate and retrieve the contents of the web and pass the content onto the indexing subsystem. Performed by a software called Google-bot.
    • Indexing: produce an index for the contents of the web that is similar to an index at the back of a book, but on a much larger scale. Indexing produces what is known as an inverted index mapping words appearing in web pages and other textual web resources onto the position where they occur in documents. In addition, index of links is also maintained to keep track of links to a given site.
    • Ranking: Relevance of the retrieved links. Ranking algorithm is called PageRank inspired by citation number for academic papers. A page will be viewed as important if it is linked to by a large number of other pages.
Processing

• Google Search Engine

» Figure 21.1 Outline architecture of the original Google search engine [Brin and Page 1998]
Processing

• Google as cloud provider
  – Google is now a major player in cloud computing which is defined as “a set of Internet based application, storage and computing services sufficient to support most user’s needs, thus enabling them to largely or totally dispense with local data storage and application software.
Processing

- Google as cloud provider
Processing

• Google as a cloud provider
  – Software as a service: offering application-level software over the Internet as web application.
  – Platform as a service: concerned with offering distributed system APIs and services across the Internet, with these APIs used to support the development and hosting of web applications. With the launch of Google App Engine, Google went beyond software as a service and now offers it distributed system infrastructure as a cloud service. Other organizations to run their own web applications on the Google platform.
Big Data
Quiz

You have 5 minutes to answer.
Password: ELGOOG
Figure 21.5
The overall Google systems architecture

Google applications and services

Google infrastructure (middleware)

Google platform
Google infrastructure

- **Distributed computation**
  - MapReduce
  - Sawzall

- **Data and coordination**
  - GFS
  - Chubby
  - Bigtable

- **Communication paradigms**
  - Protocol buffers
  - Publish-subscribe
Google Infrastructure

- **The underlying communication paradigms, including services for both remote invocation and indirect communication.**
  - The protocol buffers offers a common serialization format including the serialization of requests and replies in remote invocation.
  - The publish-subscribe supports the efficient dissemination of events to large numbers of subscribers.

- **Data and coordination services providing unstructured and semi-structured abstractions for the storage of data coupled with services to support access to the data.**
  - GFS offers a distributed file system optimized for Google application and services like large file storage.
  - Chubby supports coordination services and the ability to store small volumes of data
  - BigTable provides a distributed database offering access to semi-structure data.

- **Distributed computation services providing means for carrying out parallel and distributed computation over the physical infrastructure.**
  - MapReduce supports distributed computation over potentially very large datasets for example stored in Bigtable.
  - Sawzall provides a higher-level language for the execution of such distributed computation.
message Book {
  required string title = 1;
  repeated string author = 2;
  enum Status {
    IN_PRESS = 0;
    PUBLISHED = 1;
    OUT_OF_PRINT = 2;
  }
}
message BookStats {
  required int32 sales = 1;
  optional int32 citations = 2;
  optional Status bookstatus = 3 [default = PUBLISHED];
}
optional BookStats statistics = 3;
repeated string keyword = 4;
**Figure 21.8a**  
*Summary of design choices related to communication paradigms - part 1*

<table>
<thead>
<tr>
<th>Element</th>
<th>Design choice</th>
<th>Rationale</th>
<th>Trade-offs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol buffers</td>
<td>The use of a language for specifying data formats</td>
<td>Flexible in that the same language can be used for serializing data for storage or communication</td>
<td></td>
</tr>
<tr>
<td>Simplicity of the language</td>
<td>Efficient implementation</td>
<td>Lack of expressiveness when compared, for example, with XML</td>
<td></td>
</tr>
<tr>
<td>Support for a style of RPC</td>
<td>More efficient, extensible and supports service evolution</td>
<td>Lack of expressiveness when compared with other RPC or RMI packages</td>
<td></td>
</tr>
<tr>
<td>(taking a single message as a parameter and returning a single message as result)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol-agnostic design</td>
<td>Different RPC implementations can be used</td>
<td>No common semantics for RPC exchanges</td>
<td></td>
</tr>
</tbody>
</table>
Figure 21.8b
Summary of design choices related to communication paradigms - part 2

<table>
<thead>
<tr>
<th>Publish-subscribe</th>
<th>Topic-based approach</th>
<th>Supports efficient implementation</th>
<th>Less expressive than content-based approaches (mitigated by the additional filtering capabilities)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real-time and reliability guarantees</td>
<td>Supports maintenance of consistent views in a timely manner</td>
<td>Additional algorithmic support required with associated overhead</td>
</tr>
</tbody>
</table>
NFS and AFS are general-purpose distributed file system offering file and directory abstraction. The GFS offers similar abstractions but is specialized for storage and access to very large quantities of data (not huge number of files but each file is massive 100Mega or 1Giga) and sequential reads and sequential write as opposed to random reads and writes. Must also run reliably in the face of any failure condition.
Four distinct capabilities:
1. Distribute locks to synchronize distributed activities in a large-scale asynchronous environment.
2. File system offering reliable storage of small files complementing the service offered by GFS.
3. Support the election of a primary in a set of replicas.
4. Used as a name service within Google.

It might appear to contradict the over design principle of simplicity doing one thing and doing it well. However, we will see that its heart is one core service that is offering a solution to distributed consensus and other facets emerge from this core service.

<table>
<thead>
<tr>
<th>Role</th>
<th>Operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Open</td>
<td>Opens a given named file or directory and returns a handle</td>
</tr>
<tr>
<td></td>
<td>Close</td>
<td>Closes the file associated with the handle</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
<td>Deletes the file or directory</td>
</tr>
<tr>
<td>File</td>
<td>GetContentsAndStat</td>
<td>Returns (atomically) the whole file contents and metadata associated with the file</td>
</tr>
<tr>
<td></td>
<td>GetStat</td>
<td>Returns just the metadata</td>
</tr>
<tr>
<td></td>
<td>ReadDir</td>
<td>Returns the contents of a directory – that is, the names and metadata of any children</td>
</tr>
<tr>
<td></td>
<td>SetContents</td>
<td>Writes the whole contents of a file (atomically)</td>
</tr>
<tr>
<td></td>
<td>SetACL</td>
<td>Writes new access control list information</td>
</tr>
<tr>
<td>Lock</td>
<td>Acquire</td>
<td>Acquires a lock on a file</td>
</tr>
<tr>
<td></td>
<td>TryAcquire</td>
<td>Tries to acquire a lock on a file</td>
</tr>
<tr>
<td></td>
<td>Release</td>
<td>Releases a lock</td>
</tr>
</tbody>
</table>
Figure 21.11
Overall architecture of Chubby

* denotes current master
Figure 21.12
Message exchanges in Paxos (in absence of failures) - step 1

Step 1: electing a coordinator

Coordinator → Propose (seq_number) → Replicas

Replicas → Promise → Coordinator
Figure 21.12
Message exchanges in Paxos (in absence of failures) - step 2

Step 2: seeking consensus

Coordinator

Accept (value)

Acknowledgement

Replicas
Figure 21.12
Message exchanges in Paxos (in absence of failures) - step 3

Step 3: achieving consensus

Coordinator

Commit

Replicas
GFS offers storing and accessing large flat file which is accessed relative to byte offsets within a file. It is efficient to store large quantities of data and perform sequential read and write (append) operations. However, there is a strong need for a distributed storage system that provide access to data that is indexed in more sophisticated ways related to its content and structure.

Instead of using an existing relational database with a full set of relational operators (union, selection, projection, intersection and join). However, the performance and scalability is a problem. So Google uses BigTable in 2008 which retains the table model but with a much simpler interface.

Given table is a three-dimensional structure containing cells indexed by a row key, a column key and a timestamp to save multiple versions.

For example, web pages uses rows to represent individual web pages, and the columns to represent data and metadata associated with that given web page.

For example, Google earth uses rows to represent geographical segments and columns to represent different images available for that segment.
A Bigtable is broken up into tablets, with a given tablet being approximately 100 to 200 megabytes in size. It uses both GFS and Chubby for data storage and distributed coordination.

Three major components:
- A library component on the client side
- A master server
- A potential large number of tablet servers
Figure 21.15
The storage architecture in Bigtable

Held in main memory

Memtable

Merge

Read

Write through

Held in GFS

Persistent log

Write

SSTable files
A Bigtable client seeking the location of a tablet starts the search by looking up a particular file in Chubby that is known to hold the location of a root tablet (containing the root index of the tree structure).

The root contains metadata about other tablets specifically about other metadata tablets, which in turn contain the location of the actual data tablets.
Figure 21.17
Summary of design choices related to data storage and coordination

<table>
<thead>
<tr>
<th>Element</th>
<th>Design choice</th>
<th>Rationale</th>
<th>Trade-offs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GFS</strong></td>
<td>The use of a large chunk size (64 megabytes)</td>
<td>Suited to the size of files in GFS; efficient for large sequential reads and appends; minimizes the amount of metadata</td>
<td>Would be very inefficient for random access to small parts of files</td>
</tr>
<tr>
<td></td>
<td>The use of a centralized master</td>
<td>The master maintains a global view that informs management decisions; simpler to implement</td>
<td>Single point of failure (mitigated by maintaining replicas of operations logs)</td>
</tr>
<tr>
<td></td>
<td>Separation of control and data flows</td>
<td>High-performance file access with minimal master involvement</td>
<td>Complicates the client library as it must deal with both the master and chunk servers</td>
</tr>
<tr>
<td></td>
<td>Relaxed consistency model</td>
<td>High performance, exploiting semantics of the GFS operations</td>
<td>Data may be inconsistent, in particular duplicated</td>
</tr>
<tr>
<td><strong>Chubby</strong></td>
<td>Combined lock and file abstraction</td>
<td>Multipurpose, for example supporting elections</td>
<td>Need to understand and differentiate between different facets</td>
</tr>
<tr>
<td></td>
<td>Whole-file reading and writing</td>
<td>Very efficient for small files</td>
<td>Inappropriate for large files</td>
</tr>
<tr>
<td></td>
<td>Client caching with strict consistency</td>
<td>Deterministic semantics</td>
<td>Overhead of maintaining strict consistency</td>
</tr>
<tr>
<td><strong>Bigtable</strong></td>
<td>The use of a table abstraction</td>
<td>Supports structured data efficiently</td>
<td>Less expressive than a relational database</td>
</tr>
<tr>
<td></td>
<td>The use of a centralized master</td>
<td>As above, master has a global view; simpler to implement</td>
<td>Single point of failure; possible bottleneck</td>
</tr>
<tr>
<td></td>
<td>Separation of control and data flows</td>
<td>High-performance data access with minimal master involvement</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Emphasis on monitoring and load balancing</td>
<td>Ability to support very large numbers of parallel clients</td>
<td>Overhead associated with maintaining global states</td>
</tr>
</tbody>
</table>
Figure 21.18
Examples of the use of MapReduce

<table>
<thead>
<tr>
<th>Function</th>
<th>Initial step</th>
<th>Map phase</th>
<th>Intermediate step</th>
<th>Reduce phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word count</td>
<td>For each occurrence of word in data partition, emit <code>&lt;word, 1&gt;</code></td>
<td>Output a line if it matches a given pattern</td>
<td>For each word in the intermediary set, count the number of 1s</td>
<td>Null</td>
</tr>
<tr>
<td>Grep</td>
<td>Output a line if it matches a given pattern</td>
<td>For each entry in the input data, output the key-value pairs to be sorted</td>
<td>Null</td>
<td>Null</td>
</tr>
<tr>
<td>N.B. This relies heavily on the intermediate step</td>
<td>Parse the associated documents and output a `&lt;word, document ID&gt; pair wherever that word exists</td>
<td>Merge/sort all key-value keys according to their intermediary key</td>
<td>For each word, produce a list of (sorted) document IDs</td>
<td></td>
</tr>
<tr>
<td>Inverted index</td>
<td>Partition data into fixed-size chunks for processing</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The overall execution of a MapReduce program

- The first stage is to split the input file into M pieces, with each piece being typically 16-64 megabytes in size (no bigger than a single chunk in GFS). The intermediary results is also partitioned into R pieces. So M map and R reduce.
- The library then starts a set of worker machines from the pool available in the cluster with one being designed as the master and other being used for executing map or reduce steps.
- A worker that has been assigned a map task will first read the contents of the input file allocated to that map task, extract the key-value pairs and supply them as input to the map function. The output of the map function is a processed set of key/value pairs that are held in an intermediary buffer.
- The intermediary buffers are periodically written to a file local to the map computation. At this stage, the data are partitioned resulting in R regions. Unusually apply hash function to key then modulo R to the hashed value to produce R partitions.
- When a worker is assigned to carry out a reduce function, it reads its corresponding partition from the local disk of the map workers using RPC. The data will be sorted and reduce worker will step through key-value pairs in the partition applying the reduce function to produce an accumulated result set which will be written to an output file. This continues until all keys in the partition are processed.
Figure 21.20
The overall execution of a Sawzall program
### Figure 21.21
Summary of design choices related to distributed computation

<table>
<thead>
<tr>
<th>Element</th>
<th>Design choice</th>
<th>Rationale</th>
<th>Trade-offs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapReduce</td>
<td>The use of a common framework</td>
<td>Hides details of parallelization and distribution from the programmer; improvements to the infrastructure immediately exploited by all MapReduce applications</td>
<td>Design choices within the framework may not be appropriate for all styles of distributed computation</td>
</tr>
<tr>
<td>Programming of system via two operations, <code>map</code> and <code>reduce</code></td>
<td>Very simple programming model allowing rapid development of complex distributed computations</td>
<td></td>
<td>Again, may not be appropriate for all problem domains</td>
</tr>
<tr>
<td>Inherent support for fault-tolerant distributed computations</td>
<td>Programmer does not need to worry about dealing with faults (particularly important for long-running tasks running over a physical infrastructure where failures are expected)</td>
<td></td>
<td>Overhead associated with fault-recovery strategies</td>
</tr>
<tr>
<td>Sawzall</td>
<td>Provision of a specialized programming language for distributed computation</td>
<td>Again, support for rapid development of often complex distributed computations with complexity hidden from the programmer (even more so than with MapReduce)</td>
<td>Assumes that programs can be written in the style supported (in terms of filters and aggregators)</td>
</tr>
</tbody>
</table>