Why Big Data, Hadoop, M-R?

- What is the connection with the things we learned?
- What about SQL?
- What about relational databases?
social networks

blogs and news

RFIDs

sensors

1TB

10TB

100TB
Challenges

- Huge amounts of data
- Most data is not well structured (does not follow a specified schema)
- Can’t scale using a bigger, more powerful machine
The NoSQL revolution
What is Big Data?
What is Big Data?

Big Data: push the limits of the technology
What is Big Data?

2 billion internet users
What is Big Data?

4.6 billion mobile phones
What is Big Data?

7TB of data processed by Twitter per day

10TB of data processed by Facebook per day
What is Hadoop?

see video at: http://www.youtube.com/watch?v=xJHv5t8jcM8

Patricia Florissi, Vice President & Global Chief Technology Officer for Sales, EMC Corp.
Motivation: Large Scale Data Processing

- Want to process lots of data, unstructured or structured

- Want to parallelize across hundreds/thousands of commodity computers
  - New definition of cluster computing: *large numbers of low-end processors working in parallel to solve a computing problem.*
  - Parallel DB: *a small number of high-end servers.*

- Want to make this easy
Programming Model

- Borrows from functional programming
- Users implement an interface of two functions:
  
  ```
  map (in_key, in_value) ->
  list(out_key, intermediate_value)
  ```

  ```
  reduce (out_key, list(intermediate_value) ->
  list(out_value)
  ```
map

- Input: a key-value pair. E.g.,
  - A line out of files (filename, line),
  - A row of a database (row_id, row),
  - A document (doc_name, document)

- `map( )` produces one or more *intermediate* values along with an output key from the input.

- `map( )` is *stateless*: one input leaves no state that would affect the processing of the next input.
reduce

- After the map phase is over, **all** the intermediate values for a given output key are collected into a list
- `reduce()` combines those intermediate values into one or more *final values* for that same output key
- `reduce()` can be **stateful**: it operates on all the intermediate values of a certain key
Why is MapReduce Successful?

- **Easy**
  - Democratization of parallel computing
  - Just two *serial* functions
  - Time to first query: a few hours (contrast with parallel DB...)

- **Flexible**
  - Schema-free, “In situ” processing
  - “First, load your data into the database...”
  - “First, convert your images to bitmaps...”
  - “First, encode your 3D mesh as triangle soup...”

- **Fault-tolerance**
Example: Count Word Occurrences

map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Emit(AsString(result));

How do we implement this using a relational DBMS? Customized data loading (data may be used only once), then Group By.
Click Stream Analysis: Page Frequencies

Clicks(time, url, referral_url, user_id, geo_info...)

map(String tuple_id, String tuple):
    EmitIntermediate(url, "1");

reduce(String url, Iterator list_tuples):
    int result = 0;
    for each t in list_tuples:
        result += ParseInt(t);
    Emit(AsString(result));

Select count(*)
From Clicks
Group By url;
Parallelism

- The map() function is stateless, so many instances can run in parallel on different splits (chunks) of input data.
- The reduce() function is stateful, but works on an output key at a time, so many copies can run in parallel on different keys (groups).
- Performance bottleneck: reduce phase can’t start until map phase is completely finished.
MapReduce vs RDBMS

- **RDBMS**
  - Declarative query languages
  - Schemas
  - Logical Data Independence
  - Indexing
  - Algebraic Optimization
  - Caching/Materialized Views
  - ACID/Transactions
  - DryadLINQ, Pig, HIVE
  - HIVE, Pig
  - Hbase

- **MapReduce**
  - High Scalability
  - Fault-tolerance
  - “One-person deployment”
Shared Nothing Parallel Databases

- Teradata
- Greenplum  EMC (July 2010)
- Netezza  IBM (Sep 2010)
- Aster Data Systems  Teradata (March 2011)
- Datallegro  Microsoft (July 2008)
- Vertica  HP (Feb 2011)
- MonetDB  Commercialized as Vectorwise