Challenges for Data Driven Systems

Eiko Yoneki
University of Cambridge Computer Laboratory

Quick History of Data Management

- 4000 B.C. Manual recording
- From tablets to papyrus...to paper

A. Payberah'2014
1800's - 1940's

- Punched cards (no fault-tolerance)
- Binary data
- 1911: IBM appeared

1940's - 1970's

- Magnetic tapes
- Batch transaction processing
- Hierarchical DBMS
- Network DBMS
1980's

- Relational DBMS (tables) and SQL
- ACID (Atomicity Consistency Isolation Durability)
- Client-server computing
- Parallel processing

1990's - 2000's

- The Internet...
2010's

- NoSQL: BASE instead of ACID
  - Basic Availability, Soft-state, Eventual consistency
- Big Data is emerging!

2010s: Big Data

- Why Big Data now?
  - Increase of Storage Capacity
  - Increase of Processing Capacity
  - Availability of Data
  - Hardware and software technologies can manage ocean of data

  up to 2003 5 exabytes  
  → 2012 2.7 zettabytes (500 x more)  
  → 2015 ~8 zettabytes (3 x more than 2012)
Examples of Big Data

- Facebook:
  - 300 PB data warehouse (600TB/day)
  - 1 billion users

- Twitter Firehose:
  - 500 million tweet/day

- CERN
  - 15 PB/year - Stored in RDB

- Google:
  - 40,000 search queries/second

- eBay
  - 9PB of user data + >50 TB/day

- Amazon Web Services
  - Estimated ~450,000 servers for AWS
  - S3 450B objects, peak 290K request/sec

- JPMorgan Chase
  - 150PB on 50K+ servers with 15K apps running

Scale-Up vs. Scale-Out

- Popular solution for big data processing → to scale and build on distribution and combine theoretically unlimited number of machines in a single distributed storage

- Scale up: add resources to single node in a system
- Scale out: add more nodes to a system
**Challenges**

- Distribute and shard parts over machines
  - Still fast traversal and read to keep related data together
  - Scale out instead scale up

- Avoid naïve hashing for sharding
  - Do not depend on the number of node
  - But difficult add/remove nodes
  - Trade off – data locality, consistency, availability, read/write/search speed, latency etc.

- Analytics requires both real time and post fact analytics – and incremental operation

**Big Data: Technologies**

- Distributed infrastructure
  - Cloud (e.g. Infrastructure as a service, Amazon EC2, Google App Engine, Elastic, Azure)
    cf. Multi-core (parallel computing)

- Storage
  - Distributed storage (e.g. Amazon S3, Hadoop Distributed File System (HDFS), Google File System (GFS))

- Data model/indexing
  - High-performance schema-free database (e.g. NoSQL DB - Redis, BigTable, Hbase, Neo4J)

- Programming model
  - Distributed processing (e.g. MapReduce)
Big Data Analytics Stack

Query Language
- Pig, Hive, Shark
- Meteor, SCOPE
- DryadLINQ

Machine learning
- Mahout, MLBase
- SystemML, Presto

Graph Processing
- Pregel, GraphLab
- Bagel, GraphX
- Giraph
- Unicorn

Streaming Processing
- Storm, S4, SEEP
- Dstream, Naïad

Execution Engine
- MapReduce, Dryad, Spark
- Nephele/PACT, Hayracks
- Percolator

Resource Manager
- Mesos, YARN

Storage
- BigTable, Hbase, Dynamo
- Cassandra, MongoDB, Voldemort
- HDFS
- GFS Spanner Dremel

Database

Distributed Infrastructure

- Computing + Storage transparently
  - Cloud computing, Web 2.0
  - Scalability and fault tolerance

- Distributed servers
  - Amazon EC2, Google App Engine, Elastic, Azure
    - System? OS, customisations
    - Sizing? RAM/CPU based on tiered model
    - Storage? Quantity, type

- Distributed storage
  - Amazon S3
  - Hadoop Distributed File System (HDFS)
  - Google File System (GFS), BigTable...
Data Model/Indexing

- Support large data
- Fast and flexible access to data
- Operate on distributed infrastructure

Is SQL Database sufficient?

NoSQL (Schema Free) Database

- NoSQL database
  - Operate on distributed infrastructure
  - Based on key-value pairs (no predefined schema)
  - Fast and flexible

Pros: Scalable and fast
Cons: Fewer consistency/concurrency guarantees and weaker queries support

- Implementations
  - MongoDB, CouchDB, Cassandra, Redis, BigTable, Hbase ...
MapReduce Programming

- Target problem needs to be parallelisable
- Split into a set of smaller code (map)
- Next small piece of code executed in parallel
- Results from map operation get synthesised into a result of original problem (reduce)

Data Flow Programming

- Non standard programming models
- Data (flow) parallel programming
  - e.g. MapReduce, Dryad/LINQ, NAIAD, Spark

MapReduce: Hadoop

DAG (Directed Acyclic Graph) based: Dryad/Spark/Tez

Two-Stage fixed dataflow

More flexible dataflow model
Easy Cases

- Sorting
  - Google 1 trillion items (1PB) sorted in 6 Hours

- Searching
  - Hashing and distributed search

→ Random split of data to feed M/R operation

- BUT Not all algorithms are parallelisable

Streaming Data

- Departure from traditional static web pages
- New time-sensitive data is generated continuously
- Rich connections between entities

- Challenges:
  - High rate of updates
  - Continuous data mining - Incremental data processing
  - Data consistency
**Techniques for Analysis**

- Applying these techniques: larger and more diverse datasets can be used to generate more numerous and insightful results than smaller, less diverse ones.

- Classification
- Cluster analysis
- Crowd sourcing
- Data fusion/integration
- Data mining
- Ensemble learning
- Genetic algorithms
- Machine learning
- NLP
- Neural networks
- Network analysis
- Optimisation
- Pattern recognition
- Predictive modelling
- Regression
- Sentiment analysis
- Signal processing
- Spatial analysis
- Statistics
- Supervised learning
- Simulation
- Time series analysis
- Unsupervised learning
- Visualisation

---

**Do we need new types of algorithms?**

- Cannot always store all data
  - Online/streaming algorithms
    - Have we seen x before?
    - Rolling average of previous K items
  - Incremental updating

- Memory vs. disk becomes critical
  - Algorithms with limited passes

- N^2 is impossible and fast data processing
  - Approximate algorithms, sampling

- Iterative operation (e.g. machine learning)

- Data has different relations to other data
  - Algorithms for high-dimensional data (efficient multidimensional indexing)
Typical Operation with Big Data

- Scalable clustering for parallel execution
- Smart sampling of data
- Find similar items ➔ efficient multidimensional indexing
- Incremental updating of models ➔ support streaming
- Distributed linear algebra ➔ dealing with large sparse matrices
- Plus usual data mining, machine learning and statistics
  - Supervised (e.g. classification, regression)
  - Non-supervised (e.g. clustering..)

How about Graph (Network) Data?

- Protein Interactions [genomebiology.com]
- Bipartite graph of phrases in documents
- Gene expression data
- Social media data
- Brain Networks: 100B neurons (700T links) requires 100s GB memory
- Airline Graphs
- Web 1.4B pages (6.6B links)
How about Graph Data?

- Bipartite graph of phrases in documents
- Protein Interactions [genomebiology.com]
- Gene expression data
- Social media data
- Brain Networks: 100B neurons (700T links) require 100s GB memory
- Web: 1.4B pages (6.6B links)

Data-Parallel vs. Graph-Parallel

- **Data-Parallel** for all? **Graph-Parallel** is hard!
  - Data-Parallel (sort/search - randomly split data to feed MapReduce)
  - Not every graph algorithm is parallelisable (interdependent computation)
  - Not much data access locality
  - High data access to computation ratio

![Data-Parallel vs. Graph-Parallel Diagram]
Graph-Parallel

- Graph-Parallel (Graph Specific Data Parallel)
  - Vertex-based iterative computation model
  - Use of iterative Bulk Synchronous Parallel Model
    - Pregel (Google), Giraph (Apache), Graphlab, GraphChi (CMU)
  - Optimisation over data parallel
    - GraphX/Spark (U.C. Berkeley)
  - Data-flow programming – more general framework
    - NAIAD (MSR)

BSP Example

- Finding the largest value in a connected graph

Local Computation
Communication
Local Computation
Communication
...

Message

3 6 2 1
6 6 2 6
6 6 6 6
6 6 6 6
6 6 6 6
6 6 6 6
6 6 6 6
6 6 6 6
6 6 6 6
Graph Computation

- Two characteristic patterns: traversal and fixed-point iteration

- Breadth-first search (weakly-connected components)
  - Search proceeds in a frontier
  - 90% computation, 10% communication

- PageRank
  - All vertices active in each iteration
  - 50% computation, 50% communication

(* based on Pannotia benchmark suite)

Big Data Analytics Stack

<table>
<thead>
<tr>
<th>Query Language</th>
<th>Machine learning</th>
<th>Graph Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pig, Hive, Shark, Meteor, SCOPEDryadLINQ</td>
<td>Mahout, MLBase, SystemML, Presto</td>
<td>Pregel, GraphLab, Bagel, GraphX, Giraph, Unicorn</td>
</tr>
<tr>
<td>Streaming Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storm, S4, SEEP, Dstream, Naiad</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Execution Engine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MapReduce, Dryad, Spark, Nephele/PACT, Hayracks, Percolator</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Manager</td>
<td>Mesos, YARN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BigTable, Hbase, Dynamo, Cassandra, MongoDB, Voldemort</td>
<td>HDFS, GFS, Spanner, Dremel</td>
<td></td>
</tr>
</tbody>
</table>
Hadoop Big Data Analytics Stack

Spark Big Data Analytics Stack
Do we really need a large cluster?

- A laptop can perform sufficiently

<table>
<thead>
<tr>
<th>System</th>
<th>cores</th>
<th>twitter_rv</th>
<th>uk_2007_05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>128</td>
<td>857s</td>
<td>1759s</td>
</tr>
<tr>
<td>Giraph</td>
<td>128</td>
<td>596s</td>
<td>1235s</td>
</tr>
<tr>
<td>GraphLab</td>
<td>128</td>
<td>249s</td>
<td>833s</td>
</tr>
<tr>
<td>GraphX</td>
<td>128</td>
<td>419s</td>
<td>462s</td>
</tr>
<tr>
<td>Single thread</td>
<td>1</td>
<td>300s</td>
<td>651s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>cores</th>
<th>twitter_rv</th>
<th>uk_2007_05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>128</td>
<td>1784s</td>
<td>8000s+</td>
</tr>
<tr>
<td>Giraph</td>
<td>128</td>
<td>200s</td>
<td>8000s+</td>
</tr>
<tr>
<td>GraphLab</td>
<td>128</td>
<td>242s</td>
<td>714s</td>
</tr>
<tr>
<td>GraphX</td>
<td>128</td>
<td>251s</td>
<td>800s</td>
</tr>
<tr>
<td>Single thread</td>
<td>1</td>
<td>153s</td>
<td>417s</td>
</tr>
</tbody>
</table>

Single Computer?

- Use of powerful HW/SW parallelism
  - SSDs as external memory
  - GPU for massive parallelism
  - Exploit graph structure/algorithm for processing
**Big Data Analytics Stack**

![Diagram of Big Data Analytics Stack]

**Topic Areas**

Session 1: Introduction

Session 2: Programming in Data Centric Environment

Session 3: Processing Models of Large-Scale Graph Data

Session 4: Data Flow Programming Hands-on Tutorial with EC2

Session 6: Stream Data Processing + Guest lecture

Session 5: Optimisation in Data Processing

Session 7: Machine Learning for Computer System's Optimisation

Session 8: Project Study Presentation
Summary

- R212 course web page:
  www.cl.cam.ac.uk/~ey204/teaching/ACS/R212_2015_2016

- Enjoy the course!