Die zahlreichen Facetten der Datenflut – Wege zum effizienteren Umgang mit Informationen

Prof. Dr. Mira Mezini
Technische Universität Darmstadt

During the first day of a baby’s life, the amount of data generated by humanity is equivalent to 70 times the information contained in the library of congress.

The Human Face of Big Data, Smolan & Erwitt
The **VALUE** of Big Data: Accenture survey 2014

More than 1,000 respondents from companies across 7 industries in 19 countries that completed at least one big data implementation.

What does the future hold?

A vast majority of users (89%) believe big data will revolutionize the way business is done in the same way the Internet did. Over the next five years, users believe big data will have the biggest impact on:

- **63%** customer relationships
- **58%** redefining product development
- **56%** changing operations

What are the challenges?

- **51%** security
- **47%** budget
- **41%** lack of talent to implement
- **37%** lack of talent to run
- **35%** integration with existing systems
Extracting value is hard ...

- Data is massive, unstructured, dirty; questions are complex
- **Sequential execution just won’t scale up;** No single computer can do it!

- Facebook: > 180,000 servers
- Microsoft: > 1 million servers
- Google: >>> 1 million servers; planning for 10 million (J. Dean)
Extracting value is hard ...

Parallel programming is hard: Race conditions, fault tolerance, data distribution, load balancing, ... Processing tools still in their “infancy”

Jeff Dean@Google on “the joys of real hardware”

Typical first year for a new cluster:
~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packet loss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures, slow disks, bad memory, misconfigured machines, ... etc.

### Parallelization: Platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud of Clouds</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Clouds</td>
<td>Data Parallel Programming</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Clusters</td>
<td>MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>GPU</td>
<td>CUDA/OpenCL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Big Data Processing: MapReduce
MapReduce

... a data-parallel **programming model** and **associated implementation** for processing **large data sets** designed for **scalability** and **fault-tolerance**

**Programming Model:**
- Read a lot of data (key-value pairs) from disk
- **Map:** Extract smth relevant from each record
- Shuffle and sort
- **Reduce:** aggregate, summarize, filter, transform
- Write results (key-value pairs) to disk

**Implementation library (runtime):**
Takes care of parallelization, fault tolerance, data distribution and load balancing
Parallelized juice production
... by Jeff Dean 😊
MapReduce@Work

Pioneered by Google popularized by Hadoop

• Google:
  – Index construction for Google Search
  – Article clustering for Google News
  – Statistical machine translation

• Yahoo!:
  – “Web map” powering Yahoo! Search
  – Spam detection for Yahoo! Mail

• Facebook:
  – Data mining
  – Ad optimization
  – Spam detection
Can you do everything in MapReduce?

Users wanted more …

- More complex, multi-stage applications: iterative algorithms and machine learning
- More interactive ad-hoc queries
- Stream, graph processing
- Higher-level abstraction

Multi-stage, interactive apps, and stream processing require faster **data sharing** across parallel jobs
Specialized big data systems

**Higher-level programming**
- Sawzall
- SCOPE
- DryadLINQ
- PIG
- Impala
- HIVE
- ...

**Streaming**
- S4
- SummingBird
- Storm
- LINQ/Rx
- ...

**CEP**
- StreamBase
- Apama
- Esper
- SQLstream
- StreamInsight

**Big Graphs**
- GraphLab
- Pregel
- Giraph
- Hama
- ...

**Iter. Algorithms**
- Dremel
- Drill
- Tez
- ...

**Incremental processing**
- Incoop
- LiveLINQ
- ...

**Unification & composability needed!**
An Analogy

First cellular phones

Specialized devices

Unified device (smartphone)

by Ion Stoica
Beyond MapReduce: Spark
Data sharing in MapReduce versus Spark

**MapReduce (MR)**

- Input
- HDFS read
- iter. 1
- HDFS write
- iter. 2
- ... 
  - query 1
  - result 1
  - query 2
  - result 2
  - query 3
  - result 3

**Spark**

- Input
- one-time processing
- iter. 1
- iter. 2
- ... 
  - query 1
  - query 2
  - query 3

**Slow** due to replication, serialization, and disk IO; Each iteration is a MR job

10-100× faster than network and disk
Spark programming model

Key idea: *resilient distributed datasets* (**RDDs**)
- **Objects collections** that can be **cached in memory** across cluster nodes
- Manipulated through various **parallel operators**:
  
  ```
  map, filter, groupBy, sort, join, leftOuterJoin, rightOuterJoin, reduce, count, reduceByKey, groupByKey, first, union, cross, ...
  ```
- Automatically **rebuilt on failure**: RDDs track the series of transformations used to build them (their *lineage*) to re-compute lost data

Interface
- Clean **language-integrated API** in Scala
- Can be used *interactively* from Scala console
Higher-level programming model

```cpp
#include "mapreduce/mapreduce.h"

// User's map function
class SplitWords: public Mapper {
public:
    virtual void Map(const MapInput& input) {
        const string& text = input.value();
        const int n = text.size();
        for (int i = 0; i < n; ) {
            // Skip past leading whitespace
            while (i < n && isspace(text[i])) i++;
            // Find word end
            int start = i;
            while (i < n && !isspace(text[i])) i++;
            if (start < i)
                Emit(text.substr(start, i - start), "1");
        }
    }
}

REGISTER_MAPPER(SplitWords);

// User's reduce function
class Sum: public Reducer {
public:
    virtual void Reduce(ReduceInput* input) {
        int64 value = 0;
        while (!input->done()) {
            value += StringToInt(input->value());
            input->NextValue();
        }
        // Emit sum for input->key()
        Emit(IntToString(value));
    }
}

REGISTER_REDUCTER(Sum);

int main(int argc, char** argv) {
    ParseCommandLineFlags(argc, argv);
    MapReduceSpecification spec;
    for (int i = 1; i < argc; i++) {
        MapReduceInput* input = spec.add_input();
        input->set_format("text");
        input->set_filepattern(argv[i]);
        input->set_mapper_class("SplitWords");
    }
    // Specify the output files
    MapReduceOutput* out = spec.output();
    out->set_filebase("/gfs/test/freq");
    out->set_num_tasks(100);
    out->set_format("text");
    out->set_reducer_class("Sum");
    // Do partial sums within map
    out->set_combiner_class("Sum");
    // Tuning parameters
    spec.set_machines(2000);
    spec.set_map_megabytes(100);
    spec.set_reduce_megabytes(100);
    // Now run it
    MapReduceResult result;
    if (!MapReduce(spec, &result)) abort();
    return 0;
}

// Spark code
val file = spark.textFile("hdfs://...");
val counts = file.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)

counts.save("out.txt")
```
Unification and seamless integration

```
$ ./spark-shell
scala> val file = sc.hadoopFile("smallLogs")
... 
scala> val filtered = file.filter(_.contains("ERROR"))
... 
scala> val mapped = filtered.map(...)
...
```

Explore data interactively

```
object ProcessProductionData {
  def main(args: Array[String]) {
    val sc = new SparkContext(...) 
    val file = sc.hadoopFile("productionLogs")
    val filtered = file.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

Similar code for batch processing of large logs

```
object ProcessLiveStream {
  def main(args: Array[String]) {
    val sc = new StreamingContext(...) 
    val stream = sc.kafkaStream(...) 
    val filtered = stream.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

Similar code for stream processing

- **Streaming** runs as a series of small (~1s) batch jobs, keeping state in memory as fault-tolerant RDDs
- **Intermix seamlessly with batch and ad-hoc queries**
Spark ecosystem

- BlinkDB (Approximate SQL)
- Shark (SQL)
- Spark Streaming (Streaming)
- MLLib (Machine learning)
- GraphX (Graph Computation)
- SparkR (R on Spark)

Spark Core Engine

Alpha / Pre-alpha
Big Data for Accelerated Software Development: Code Recommenders
What does the developer need, i.e., which methods should the code completion present to the user?
this is what you get today …
... and, it can be worse ...
Intelligent code completion

```java
@Override
protected Control createDialogArea(Composite parent) {
    Composite container = (Composite) super.createDialogArea(parent);
    SWTTextWidget = new Text(container, SWT.BORDER);
    SWTTextWidget.
}
```

... just the three relevant ones
Data driven software development

interact with

usage data clicks, selections
contribute, search, query, vote

interact with

suggest, notify

Knowledge Engine
Concluding Remarks
The future of Big Data

- Unified processing stacks for reduced complexity
- Secure and privacy preserving big data analytics
- Scale-invariant data analytics algorithms
- Collaborative predictive/prescriptive analytics
- Real-time interactive big data analytics and visualization
- Collecting massive data via crowd-sourcing

Cognitive environments

Neurosynaptic chips

http://www.research.ibm.com/cognitive-computing/#fbid=OutzStPPVbp
One more thing ...

The iPhone would probably not exist if Apple had made it a “data driven decision”... 😊