Datenbanksysteme II: Big Data

Ulf Leser
Content of this Lecture

- Big Data Introduction
- Map Reduce Programming Paradigm
- Parallel DBMS or MapReduce?
- Extensions to MR

- [based on slides by Astrid Rheinländer, 2012]
What is Big Data?

- A buzz word
- “Collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications” [http://en.wikipedia.org/wiki/Big_data]
  - Terabytes / petabytes
  - Near future: exabytes

- Challenges
  - Storage
  - Analysis
  - Search
Example: Twitter

- 2012: 63 Billion tweets, 8.5 TB data
  - 2016: ~200 Billion tweets per year
- Business: Sell access to content
- Analysis: Find emerging trends, sentiment analysis, …
Example: Cern, LHC

- 2012: LHC experiments generated 22PB of data
  - ~99% are filtered right after creation
- Analysis runs on supercomputer: 1000+ processors
- Data stored & processed in LHC Computing Grid
  - 150 data & compute centers around the world
  - Heterogeneous architecture
Will Computers Crash Genomics?

Fast Development

1953
Double helix structure of DNA, Watson/Crick

2003
First human genome sequenced, took ~14 years, ~3 billion USD

2008
Genome of J. Watson finished, 4 Months, 1.5 Million USD

2010
1000 Genomes Project

1000GP releases more data in first 6 months than EMBL collected in the 25 years before
Next Generation Sequencing

- New generation of sequencers since ~2005
  - Illumina, Solexa, 454, Solid, ...
- Much higher throughput
  - ~15 TB raw data in 3-5 days
  - ~600 GB processed data/week
  - Cost for sequencing a genome down to ~2.000 USD
- 3rd generation sequencers
  - Single molecule sequencing
  - A (human) genome in a day
  - Sequence every human
  - Sequence different cells in every human
Sequencing becomes a commodity

- Sequencing dozens or hundreds of genomes is feasible (now!) for any mid-size research projects
Old Task: Genome Assembly
New Task: Read Mapping & SNV Detection
Example: SNV Detection

DNA/RNA

Reference sequence

- GCTGATGTGCCGCCCTGACTTCGGTGAGGT
- CTGATGTGCCGCCCA
- TGTGCCGCCCA
- CCGCTCACTTCGGTGAGGT
- GTGCCGCCCA
- CATGCGCCGCCCTGACTTCGGTGAGGT
- GCTGATGTGCCGCCCA

Reads: 1-6
SNV Detection

X00 million reads

Quality estimation

Quality filtering

Read mapper 1

Read mapper 2

Union

Local realign

Quality adaptation

Unmapped reads

Cross-species search

SNV filtering

Pileup

SNV assessment

DB 1

DB 2

Functional assessment, GWAS, …

Unmapped reads

DB 1

DB 2

Quality metrics
All Analytics is the Same?

**ROUTINELY UNIQUE**
Over 18 months, 46 data-analysis projects undertaken at the bioinformatics core of the University of Texas Health Science Center at Houston required 34 different types of analysis — most were used infrequently. Each project demanded unique combinations of analyses, demonstrating how bioinformaticians must be versatile, creative and collaborative.

Chang, Core services: Reward bioinformaticians, Nature 2015
Two Main Issues

- **Runtime**
  - Large and growing data volumes
  - HighSeq-X: 18 terabyte a week

- **Variety**
  - Many different types of pipelines
  - Dozens of tools in every pipeline
  - New tools / pipelines every day
  - No gold standards
Big Data:

- Fourth V: Veracity
  - Big data is often very noisy (unfiltered)
Is Big Data a Database Topic?

- We need faster algorithms: Machine learning, algorithms
- We need more scalable systems: Distributed systems
- We need more CPUs: High-Performance Computing
- We need professional services: Companies
- We need: Databases
  - **Declarativity**: Specialized, simple, powerful languages
  - Optimization: Compilation in efficient, parallel execution plans
  - Comprehensive data management: Robustness, data models, index structures, access control, ...
Big Data Landscape

Big Data Landscape (Version 2.0)

Infrastructure
- NoSQL Databases
  - 10gen DATAX
  - Couchbase
  - Cloudant
- NewSQL Databases
  - MarkLogic
  - InfiniDB
- MPP Databases
  - Vertica
  - Greenplum
- Database Management / Monitoring
  - HPCC Systems
  - Acunu
- Security
  - Stormpath
  - iMPERVA

Analytics
- Analytics Solutions
  - Palantir
  - Infobright
- Analytics Services
  - IBM
  - SAS
- Big Data Search
  - RAPID
  - HappyCow

Data Sources
- Data Marketplaces
  - Withings
  - Personal Data
- Data Sources
  - Amazon
  - Microsoft

Analytics Applications
- Ad Optimization
  - Bluekai
  - Rocketfuel
- Publisher Tools
  - Turn
  - Taboola

Social Media
- Twitter
  - Facebook
  - Google+

Cross Infrastructure / Analytics
- Open Source Projects
  - Hadoop
  - Cassandra

© Matt Turck (@mattturck) and ShivonZilis (@shivonz) Bloomberg Ventures
Content of this Lecture

• Big Data Introduction
• Map Reduce
  – Programming model
  – Framework
  – Distributed File System – HDFS
  – Error handling
• Parallel DBMS or MapReduce?
• Extensions to MR
Underlying Idea

• Observations
  – Supercomputers are expensive
  – Large commodity-clusters are error-prone
  – No simple approach to express Big Data problems and let them robustly run on a commodity cluster
  – Most parallel analysis is embarrassingly parallel
  – Functional programming has some good ideas
    • Side-effect free functions
    • 2nd order functions

• Result: MapReduce
  – Programming model: The idea
  – Software stack: A framework
  – GDFS: Google distributed file system
Some History

- Proposed by Dean & Ghemawat (Google) in 2004
  - One of the most-cited recent papers in computer science
- Yahoo adopts idea and puts code open source in 2008
  - Hadoop: Java framework for Map Reduce programming
  - HDFS: Hadoop Distributed File Systems
- Hadoop2 / Yarn in 2013
  - More flexible programming model
  - Resource management and scheduler
- Numerous companies, mostly for DWH’ish scenarios
Embarrassingly Parallel

- **Data parallel problems**: Partition the data, do partition-wise computation, then summarize results
  - Sum, min, max, mean, … of a set of values
  - Find all stars in this set of satellite sky images
  - Find all company names in this set of web pages
  - Compute this join (partitioned hash join)
- **Not emb. parallel**: Everything influences everything (a bit)
  - Median of a set of values
  - Graphs: Traversal, routing, shortest paths, …
    - But: Graph cluster coefficient
  - Simulations: Weather, molecular dynamics
  - Iterative problems: Clustering, PageRank, heuristic optimization
  - Solving equations, linear optimization
Parallel Execution of SQL Queries

• Forget joins, include grouping + aggregations
  - MR scenarios: Web log file analysis, search engines
  - No joins, just one input set (often in many files)
  - Many aggregations and groupings (by country, by day, ...)

```
SELECT lookupCountry(parse_IP(W.IP, 1)), sum(*)
FROM   weblog W
WHERE  W.cmd="GET" & W.browser="FireFox"
GROUP BY parse_IP(W.IP, 1);
```
Parallel Plan

- Partition log-file in X sets (X: Number of machines/cores?)
- Perform **parallel, partition-wise** filtering, grouping, and aggregation
- Aggregate partition-wise results (can also be done in parallel)
Why not a Parallel Database System?

- Parallel RDBMS do not scale to large clusters
  - **Synchronization**: Locks and aborts lead to contention
  - **Scalability**: Stragglers, data transport and access, central control
  - **Fault-tolerance**: Redundancy, partial restarts, central control
  - **CAP Theorem**: Could cannot have all of these three properties

- There is no open source parallel DBMS (at that time)
  - Today: Many NoSQL Systems
  - Simple data model (documents, key-value) and/or limited synchronization (eventually consistent)

- Commercial RDBMS extremely expensive
Content of this Lecture

• Big Data Introduction
• Map Reduce
  - Programming model
  - Framework
  - Distributed File System – HDFS
  - Error handling
• Parallel DBMS or MapReduce?
• Extensions to MR
Map Reduce Programming Model

- Operates on sets of key-value pairs
- Inspired by FP 2nd order functions Map and Reduce
  - Map: Apply user function f on a key-value pair
  - Reduce: Group the following set on key and apply user function f on values of n all pairs with equal key

\[
\begin{align*}
1, x_1 & \quad 1, y_1 & \quad 1, r_1 \\
\vdots & \quad \ddots & \quad \vdots \\
n, x_n & \quad n, y_n & \quad n, r_n \\
1, a_1 & \quad 1, a_2 & \quad 1, c_1 \\
5, b_1 & \quad 3, a_2 & \quad 3, c_1 \\
1, a_2 & \quad 1, c_1 & \quad 1, a_1 \\
3, c_1 & \quad 3, r_3 & \quad 3, a_1 \\
5, c_1 & \quad 5, r_5 & \quad 5, a_2
\end{align*}
\]
Map Reduce and SQL

Partition

Map 1:0/1

Filter by browser/cmd

Filter by browser/cmd

Filter by browser/cmd

Filter by browser/cmd

Reduce n:m

Agg by country

Agg by country

Agg by country

Agg by country

Union

Agg by country

Agg by country

Agg by country
Famous None-SQL Example: Word Count

- Given a set of documents, count the **frequency** of all **unique words**
  - Important for building a search engine index
- Things we need to do
  - Break documents into their words
  - Group set of all words on word
  - Compute per-word counts

```
1, to be, or not to be, that is the question:
2, whether 'tis nobler in the mind to suffer
3, the slings and arrows of outrageous fortune,
4, or to take arms against a sea of troubles
...
```

```
to: 4
be: 2
or: 2
...

mind: 1
sea: 1
...
```
Word Count in MapReduce

• Step 1: List of all words
  - Map on set of documents
  - Foreach doc, tokenize and output one key-value pair per token

• Step 2: Group by word and count
  - Sort key-value pairs by key
  - Compute sum of values per key
Content of this Lecture

• Big Data Introduction
• Map Reduce
  – Programming model
  – Framework
  – Distributed File System – HDFS
  – Error handling
• Parallel DBMS or MapReduce?
• Extensions to MR
Operational View

• User provides
  - Map function (Java class with certain interface)
  - Reduce function (Java class with certain interface)
  - Option: Partitioner (Default: Partition by lines)
    • Default not applicable to binary files
  - System configuration: Machines, IP, access

• System provides
  - Java framework where code is integrated
  - Jobs are automatically created and distributed in cluster
  - Jobs that fail get restarted
  - Sort/combine functions for reduce
  - Distributed file system with in-build redundancy
  - Simple scheduling (greedy) and locality (unclear)
Hadoop Architecture

- **Master/Worker architecture**
  - Workers are commodity hardware
  - Masters are usually well equipped
  - Shared nothing
  - File exchange by HDFS

- Reports on installations with several thousand machines
• **Master**: Manages entire execution
  - Assigns tasks and input splits to idle workers
  - Tracks status of current jobs (waiting, running, finished)
  - Tracks status of worker nodes
Execution

• Map-Worker
  - Reads assigned splits
  - Parses key-value pairs
  - Executes map function for each pair
  - Buffered intermediate data are periodically written to local disks
  - Notify master about locations of result when finished
  - Master pushes data incrementally to reduce-workers
**Execution**

- **Reduce-Worker**
  - Read assigned splits from map workers (through HDFS)
  - Usually processes more than one key group
  - Performs intermediate sort of MAP-results
  - Executes reduce() function once for each key - aggregation
  - Result is written to HDFS

- **Master notifies host program, data accessible in HDFS**
Content of this Lecture

• Big Data Introduction
• Map Reduce
  ‐ Programming model
  ‐ Framework
  ‐ Distributed File System – HDFS
  ‐ Error handling
• Parallel DBMS or MapReduce?
• Extensions to MR
HDFS architecture

• HDFS: Hadoop distributed file system

• Why a special file system?
  - Goal: store very large data redundantly on commodity hardware
  - Network latency should not hinder computations too much
  - Mostly read/append file operations, few rewrites
  - Relatively small number of large files
  - Manage much larger data compared to 'standard' fs

• Data distributed over commodity hardware
  - Nodes fails all the time
  - Data must be kept redundant and distributed
  - Single point of access
Some Details

- **Master/Worker architecture**
  - Master: *Access to DFS*, manages replication and metadata
  - Workers: Local storage in cluster, I/O and maintenance

- **Client talks to**
  - Master: To get file handles
  - Workers: To (directly) read/write data

- **Replication**
  - Files chopped in *chunks* (64MB), chunks are *replicated* (3 times)
  - Advantage: Fixed file size, easier calculations, files larger than disk

- **Interference with local FS**
  - HDFS workers have a reserved space in “usual” FS
  - Access only through HDFS, *no posix* 😞
Locality: Co-locate Data and Computation

- **Goals**
  - Increase speed of access
  - Avoid network congestion

- **What is local?**
  - Physically on same machine
  - Close in the network
  - Within the same rack
Error Handling

- Large commodity-hardware clusters: *Errors (node failure) are the norm, not the exception*
- Hadoop
  - Master nodes fail: *System gone*
    - Use robust machines, failover, RAID, …
  - Worker nodes fail
    - Tasks crash: Identify and restart
    - Node crashed: Identify (heart-beat) and mark as “dead”
    - Data is replicated: Identify gone chunks and *replicate to new nodes*
  - Stragglers: Tasks taking much longer than others
    - Create many more *(small) tasks* than machines
    - When few tasks left, start them multiple times and use first finished
      - Often system hangs, network bottleneck, …
Summary

- Focus on **cheap hardware yet large systems**
- Made by distributed systems people: No real data model, **ad-hoc analysis**, no indexes, no query languages, ...
- Made for programmers: **Java (not SQL)**
- Made for non-relational workloads: Files (not records)
- Does not solve many (exotic) DS problems, but focuses on average workloads
  - E.g., we didn’t mention time synchronization
- Many shortcomings: Scheduling, data placements, strict MapReduce framework, joins, ...
- **Extremely influential**, many subsequent developments
Content of this Lecture

• Big Data Introduction
• Map Reduce
  – Programming model
  – Framework
  – Distributed File System – HDFS
  – Error handling
• Parallel DBMS or MapReduce?
• Extensions to MR
“Major Step Backward”

- Many database people criticizing MapReduce
  - Pavlo, Paulson, Rasin, Abadi, DeWitt, Madden, Stonebraker. A comparison of approaches to large-scale data analysis. SIGMOD 2009
  - Nothing really new
    - Functional programming, distributed systems, Teradata
  - Functionality can be reached by UDFs in parallel DBMS
  - Parallel DBMS provide good scalability (and much more)
  - Interface too low-level and not declarative
  - Disk-based batched data exchange instead of pipelining/streaming
  - Lack of schemata obstructs performance optimizations
  - No indexing for recurring analyses
  - No statistics for cost-based optimizations
  - Frequent re-parsing of ASCII data
  - …
# Points of Views

<table>
<thead>
<tr>
<th></th>
<th>Parallel RDBMS</th>
<th>MapReduce-Style Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data sets</strong></td>
<td>Large, structured, discrete attributes, normalized, correct</td>
<td>Unstructured, binary, redundant, noisy</td>
</tr>
<tr>
<td><strong>Analysis</strong></td>
<td>Relational queries</td>
<td>Everything, domain-specific</td>
</tr>
<tr>
<td><strong>Data / queries</strong></td>
<td>Same data, many different queries</td>
<td>Same data, few queries</td>
</tr>
<tr>
<td><strong>Pre-Analysis</strong></td>
<td>Pays off</td>
<td>Does not pay-off</td>
</tr>
<tr>
<td></td>
<td>• Ingestion: Loading into DB</td>
<td>• Ad hoc analysis</td>
</tr>
<tr>
<td></td>
<td>• Parsing, statistics, indexing</td>
<td>• Data too large for pre-analysis</td>
</tr>
<tr>
<td><strong>Units of work</strong></td>
<td>Transactions</td>
<td>Read-only</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>Multi-user and synchronization</td>
<td>Read-only</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>High reliability</td>
<td>Pragmatism instead of perfect reliability</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>Industry-proven system</td>
<td>Many open source systems</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Very expensive</td>
<td>Much cheaper</td>
</tr>
</tbody>
</table>
What Most People Like

KDNuggets 2014 Poll: Languages used for Analytics/Data Mining

- Python: 35%
- R & Python: 20%
- R: 49%
- R & SQL: 22%
- SQL: 30.6%
- Python & SQL: 13%
Content of this Lecture

- Big Data Introduction
- Map Reduce
- Parallel DBMS or MapReduce?
- Extensions to MR
  - Stratosphere
  - Scientific Workflow Systems und SaasFee
Programming Model has Many Limitations

- Always scan all input data
- No support for schemata
- Rigid schema: map & reduce
- No joins, no heterogeneous inputs
- Low-level, imperative access
- Only Java
- No iterative workflows (recursion etc.)
- Naïve scheduler
- Too much IO, too much network traffic
- …
- [Google abandoned MR in ~2012]
Stratosphere

- Research project
  - TU, HU, and HPI
- “Web-scale” distributed data analytics system
- Database-inspired
  - Semi-structured data model
  - Analytics as queries
  - Declarative languages
  - Optimization
  - Extensible by domain-specific operations
- More flexible programming model
Declarative language with domain-specific predicates

Logical optimizer to generate optimal execution plans

Cloud-enabled processing engine for fully parallel execution

```
T = read from file...;
D = annotate $T$ using...;
G = annotate $T$ using...;
R = merge $D$, $G$ on...;
```
METEOR: A Dataflow Specification Language

1 $texts = \textbf{read from} ,articles.json';
2 $recent = \textbf{filter} \text{ }$text in $texts$
3 \hspace{1em} where $text.year \geq 2000 ;$
4 $\textbf{write} \text{ }$recent to ,recentTexts.json';

- JAQL style
- Semi-structured data model
- Designed for \textit{extensibility}
- UDFs in domain-specific packages
- Root package = relational operators
- \textbf{Uniform optimization} framework across all packages
More Complex Example

1 using ie;
2 $texts = read from 'pubmed.json';
3 $protDB = read from 'uniprotKB.json';
4 $texts = annotate sentences in $texts using medPost;
5 $texts = annotate entities in $texts using type.protein and regex ,UPDict';
6 $pText = pivot $texts around
7 $ent = $text.annotations[*].entity
8 into { protein:$ent, art:$arts};
9 $refs = join $p in $pText,$prot in $protDB
10 where $p.protein.id=$prot.id
11 into { protein:$prot, reference:$p.art};
12 write $refs to 'uniprotWithRefs.json';
## Logical Operators

<table>
<thead>
<tr>
<th>Relational</th>
<th>Information Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>All: Filter, project, join, intersect, union, …</td>
<td>Data scrubbing</td>
</tr>
<tr>
<td><strong>Information Extraction</strong></td>
<td>Duplicate detection</td>
</tr>
<tr>
<td>Sentence splitting</td>
<td>Entity mapping</td>
</tr>
<tr>
<td>Tokenization</td>
<td>Record linkage</td>
</tr>
<tr>
<td>N-Gram extraction</td>
<td></td>
</tr>
<tr>
<td>Part-of-speech tagging</td>
<td>Web Extraction</td>
</tr>
<tr>
<td>Dependency parsing</td>
<td>HTML-scrubbing</td>
</tr>
<tr>
<td>Entity annotation</td>
<td>Metadata extraction</td>
</tr>
<tr>
<td>Relationship extraction</td>
<td>Boilerplate detection</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Optimization: Plan Reordering

1. POS → Gene NER → Drug NER → RE
2. POS → Gene NER → Filter → Drug NER → Filter → RE
3. POS → Drug NER → Filter → Gene NER → Filter → RE
4. Med Pos → Drug RegEx → Filter → Gene CRF → Filter → RE
5. Drug RegEx → Filter → Med Pos → Gene CRF → Filter → RE

NER: Name Entity Recognition
RE: Rule Engine
POS: Part of Speech
Med Pos: Medical Position
RegEx: Regular Expression
Optimization: Plan Reordering

- Data-specific preconditions allow early filtering
- **Cost estimates** allow operator reordering
- Instantiation: Logical to physical operators
- Dependency resolution allows operator reordering
Parallelization

POS → Gene NER → Filter → Drug NER → Filter → RE

POS → Filter → Join → RE

POS → Gene NER → Filter → Join → RE

POS → Drug NER → Filter → Join → RE
Stratosphere programming model

- PArallelization ConTracts (PACTs)
  → Generalization of MapReduce
Many Other Variations / Improvements

- **Hbase**: ist eine skalierbare, einfache Datenbank zur Verwaltung sehr großer Datenmengen innerhalb eines Hadoop-Clusters.

- **Hive**: erweitert Hadoop um Data-Warehouse-Funktionalitäten, namentlich die Anfragesprache *HiveQL* und Indizes. ... Seit Hive 2.0 wird Hybrid Procedural SQL On Hadoop (HPL/SQL) unterstützt
  - Im Sommer 2008 stellte Facebook, der ursprüngliche Entwickler von Hive, das Projekt der Open-Source-Gemeinde zur Verfügung.[14] Die von Facebook verwendete Hadoop-Datenbank gehört mit etwas mehr als 100 Petabyte (Stand: August 2012) zu den größten der Welt.[15] Dieses Warehouse wuchs bis 2014 auf 300 PB an.[16]

- **Pig**: kann Hadoop MapReduce-Programme in der High-Level-Sprache Pig Latin erstellen. Einfach, erweiterbar, optimiert

- **Spark**: ist eine in-memory Batch Processing Engine, welche vornehmlich für Machine-Learning-Anwendungen entwickelt wurde


*Source: Wikipedia*
Content of this Lecture

- Big Data Introduction
- Map Reduce
- Parallel DBMS or MapReduce?
- Extensions to MR
  - Stratosphere
  - Scientific Workflow Systems und SaasFee
Recall

X00 million reads

Quality estimation

Quality filtering

Read mapper 1

Read mapper 2

Union

Local realign

Unmapped reads

Functional assessment, GWAS, …

SNV filtering

Union

SNV filtering

DB 1

DB 2

Very few relational operations

Most operations are pre-built

Optimization by reordering mostly impossible
Other Approaches to Big Data Analytics: Scientific Workflows

Links
- Connect input and output ports
- Implemented as files, memory, network
- Determine data dependencies / orders of execution

Tasks
- Input and output ports
- Executables or web services
- Typically black box implementation
Features

- Controlled assembly of black box scripts
- Execution monitoring and **failure recovery**
- **Parallelization** and scheduling
- Workflows: Understandable & sharable
- Limited expressivity: *Easier to read* and develop
- Often with graphical user interfaces for wf composition
  - SWMF for end users / for developers
- **Reproducibility**

- Requires **Scientific Workflow Management System**
SaasFee Software Stack

www.saasfee.io (video tutorials available)
• Light-weight statically typed functional dataflow language
• Compiles into dynamic pipelines of black-box tools
• Make foreign code integration as easy as possible
• Allow complex, iterative workflows
Foreign Code Interface

• **Directly integrates** BASH, LISP, R, MatLab, Python ...
• No wrapping, no data (un)marshalling, no API
• Communication via variables or files
• Mixing of several languages
• Snippets are shipped and executed by Hi-Way

```bash
deftask greet( out : person ) in bash *
  out="Hello $person"
}*
deftask greet( out : person ) in r *
  out = paste("Hello", person )
}*
```
Hi-Way

- **Hi-Way Workflow Application Master for YARN**
- Executes workflows on Hadoop YARN
  - Scalability, maintenance, fault tolerance, …
Achieving Parallelism

Task Parallelism

• Data dependencies

Data Parallelism

• Custom partitioning
  - Default for record-oriented files
Scalability

- Cluster with >500 cores
- Variant calling, unsplit reference, split data
- Almost perfect scalability