Big Data Analytics

Parallel Data Processing Beyond Map and Reduce

Volker Markl

Database Systems and Information Management Lab
Technische Universität Berlin

www.dima.tu-berlin.de
About Big Data Analytics

Big data: The next frontier for innovation, competition and productivity

Source: McKinsey

Source: Information Week

Source: Amazon

Source: NCSU
Agenda

- Big Data Analytics
  - Motivating Big Data Analytics
  - Examples
  - Chances and Applications
  - Challenges
  - Technologies

- Parallel Data Processing beyond map and reduce
  - Limits of Parallel Data Processing
  - Map/Reduce
  - Stratosphere
  - Nephele
  - PACTS
These data sets will fit into main memory soon!

Who needs Hadoop or NoSQL for these?
It is cheaper to resequence than to store genome data!
Other Examples of Really Big Data Sets

- Video Streams
- Smart Grids
- Audio Streams
- Web Archive
- Sensor Data
- RFID Data
- Genome Data

- RFID Data
- Genome Data
- Sensor Data
- Video Streams
- Smart Grids
- Audio Streams

- Video Streams
- Smart Grids
- Audio Streams

- Video Streams
- Smart Grids
- Audio Streams

- Video Streams
- Smart Grids
- Audio Streams

- Video Streams
- Smart Grids
- Audio Streams
Complexity

Size
Structure/Representation
Uncertainty
„Cleanliness“
Generating Process
etc.

Selection/Grouping
Relational Operators
Information Extraction & Integration
Data Mining
Predictive Models
etc.

Data

Query
Analysis Tasks on Climate Data Sets

- Validate climate models
- Locate „hot-spots“ in climate models
  - Monsoon
  - Drought
  - Flooding
- Compare climate models
  - Based on different parameter settings

Necessary Data Processing Operations

- Filter
- Aggregation (sliding window)
- Join

- Multi-dimensional sliding-window operations
- Geospatial/Temporal joins
- Uncertainty
Motivating Data Parallel Programming Models

- Need for data parallel programming
  - Increase of data complexity
  - Increase of query complexity
  - Moore’s Law: ManyCore and Cluster Computing
  - Scale-up no longer possible
  → **Scale-out is the name of the game**

- Parallel programming is not easy
  - (Network) Communication
  - Concurrent programming: Divide & Conquer
  - Synchronization as bottleneck
  - Fault tolerance

- Programming models ease development of parallel tasks
  - Abstractions hide the gory details
  - Automatic adaption to hardware
    - Parallelization and Optimization
  - Beware: Data flow and control flow dependencies!
The next generation of Business Intelligence (NGBI) will correlate data warehouses with text and other modalities from web services of information providers, corporate Intranets and the Internet.
A major new trend in information processing will be the trading of original and enriched data, effectively creating an information economy.

„When hardware became commoditized, software was valuable. Now that software is being commoditized, data is valuable.“ (Tim O’Reilly)

„The important question isn’t who owns the data. Ultimately, we all do. A better question is, who owns the means of analysis?“ (A. Croll, Mashable, 2011)
A possible Architecture for an Information Marketplace

Dataproducers

Data & Aggregation

Revenue Sharing

Algorithms

Technology

Licensing

Users

Queries

Analytical results

Marketplace

Social Media Monitoring

Media Publisher Services

SEO

Index

Massively Parallel Infrastructure

Distributed Data Storage

Trust

Infrastructure as a Service
Further Applications

- Home Automation
- Healthcare
- Water Management
- Lifecycle Management
- Traffic Management
- Energy Management
- Sales/Marketing
Some Challenges

Legal Dimension
- Copyright
- Privacy
- Digital Preservation
- Decision Making
- Verticals

Application Dimension
- Scalable Data Processing
- Signal Processing
- Statistics
- Linguistics
- HCI/Visualization

Social Dimension
- User Behaviour
- Societal Impact
- Business Models
- Collaboration
- Benchmarking
- Impact of Open Source
- Deployment
- Pricing

Technology Dimension
- Scalable Data Processing
- Signal Processing
- Statistics
- Linguistics
- HCI/Visualization

Economic Dimension
- Scalable Data Processing
- Signal Processing
- Statistics
- Linguistics
- HCI/Visualization

Some Challenges
Some Solutions

- Parallel Data Processing
- Parallel Data Management: ParStream et al.
- Map/Reduce: Hadoop, Stratosphere et al.
Parallel Speedup

- The Speedup is defined as: \( S_p = \frac{T_1}{T_p} \)
  - \( T_1 \): runtime of the sequential program
  - \( T_p \): runtime of the parallel program on \( p \) processors

- Amdal’s Law: „The maximal speedup is determined by the non-parallelizable part of a program“:
  - \( S_{max} = \frac{1}{(1-f) + \frac{f}{p}} \) \( f \): fraction of the program that can be parallelized
  - \( \rightarrow \) Ideal speedup: \( S=p \) for \( f=1.0 \) (linear speedup)
  - \( \rightarrow \) However - since usually \( f<1.0 \) - \( S \) is bound by a constant ! (e.g. ~10 for \( f=0.9 \))
  - \( \rightarrow \) Fixed problems can only be parallelized to a certain degree!
Parallel Speedup

Amdahl’s Law

- Parallel Portion
  - 50%
  - 75%
  - 90%
  - 95%

Number of Processors

Speedup
ParStream provides the unique combination of REAL-TIME – LOW-LATENCY – HIGH THROUGHPUT

ParStream’s key advantages:
• Unique Highly compressed Bitmap Index which can be analyzed in compressed form (patent application filed)
• Real-time analytics through Massively Parallel Processing (MPP)
• Very high throughput due to massively reduced workload (no decompression, small index, efficient algorithms)
• Low-Latency through continuous import of new data without slowing down analytics
• Columnar data / index allows very flexible analytics (multi-column, multi-value)
• Specialized data / index types and algorithms
• Shared Nothing Architecture

© Parstream GmbH, used with permission, www.parstream.com
Big Data Analytics is a game changer in every industry and is a huge market opportunity.

- eCommerce Services
  - Facetted Search
  - Web analytics
  - SEO-analytics
  - Online-Advertising
- Social Networks
  - Ad serving
  - Profiling
  - Targeting
- Telco
  - Customer attrition prevention
  - Network monitoring
  - Targeting
  - Prepaid account mgmt
- Finance
  - Trend analysis
  - Fraud detection
  - Automatic trading
  - Risk analysis
- Energy Oil and Gas
  - Smart metering
  - Smart grids
  - Wind parks
  - Mining
  - Solar Panels
- Many More
  - Production
  - Mining
  - M2M
  - Sensors
  - Genetics
  - Intelligence
  - Weather

© Parstream GmbH, used with permission
www.parstream.com
Further Players

- Hive
- HBase
- JasperReports Server
- IBM
- Sybase IQ
- In-Memory Data Management
- SAS Analytics
- SPSS Statistical Data Analysis
- Aster Data
- Greenplum
- EMC

Real-Time Analytics
- Fastest Time-to-Value
- Agile Environment
- Performance at Scale
What is Map/Reduce?

- Programming model for data-intensive programming
  - well-suited for large scale parallel execution
  - automatic parallelization & distribution of data and computational logic
  - easy to extend with fault tolerance schemes
  - clean abstraction for programmers

- Based on functional programming
  - treats computation as the evaluation of mathematical functions and avoids state and mutable data
  - no changes of states (no side effects)
  - output value of a function depends only on its arguments

- Map and Reduce are higher-order functions
  - take user-defined functions as argument
  - return a function as result
  - to define a map/reduce job, the user implements the two functions m and r
Data Flow in Map/Reduce

\[
\begin{align*}
(K_m,V_m)^* & \quad \rightarrow \quad (K_m,V_m) \\
\downarrow & \\
\text{Framework} & \\
(MAP(K_m,V_m)) & \\
\downarrow & \\
(K_m,V_m) & \\
\downarrow & \\
(K_m,V_m) & \\
\downarrow & \\
(K_m,V_m) & \\
\downarrow & \\
(\ldots) & \\
\end{align*}
\]

\[
\begin{align*}
(K_r,V_r)^* & \quad \rightarrow \quad (K_r,V_r)* \\
\downarrow & \\
\text{Framework} & \\
(\text{REDUCE}(K_r,V_r)^*) & \\
\downarrow & \\
(K_r,V_r)* & \\
\downarrow & \\
(\ldots) & \\
(\ldots) & \\
\end{align*}
\]

\[
\begin{align*}
(K_r,V_r) & \quad \rightarrow \quad (K_r,V_r) \\
\downarrow & \\
\text{Framework} & \\
(\ldots) & \\
\downarrow & \\
(\ldots) & \\
\downarrow & \\
(\ldots) & \\
\end{align*}
\]

\[
(\ldots)
\]
Problem: Counting words in a parallel fashion
- How many times different words appear in a set of files
- juliet.txt: Romeo, Romeo, wherefore art thou Romeo?
- benvolio.txt: What, art thou hurt?
- Expected output: Romeo (3), art (2), thou (2), art (2), hurt (1), wherefore (1), what (1)

Solution: Map-Reduce Job

```java
m (filename, line) {
    foreach (word in line)
        emit(word, 1);
}

r (word, numbers) {
    int sum = 0;
    foreach (value in numbers) {
        sum += value;
    }
    emit(word, sum);
}
```
Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

map

reduce

map

reduce

Romeo, 1
Romeo, 1
wherefore, 1
art, 1
thou, 1
Romeo, 1

art, (1, 1)
hurt (1),
thou (1, 1)

art, 2
hurt, 1
thou, 2

What, 1
art, 1
thou, 1
hurt, 1

Romeo, (1, 1, 1)
wherefore, (1)
what, (1)

Romeo, 3
wherefore, 1
what, 1
Relational Operators as Map/Reduce jobs

- Selection / projection / aggregation

- SQL Query:
  
  ```sql
  SELECT year, SUM(price)
  FROM sales
  WHERE area_code = "US"
  GROUP BY year
  ```

- Map/Reduce job:
  
  ```java
  map(key, tuple) {
    int year = YEAR(tuple.date);
    if (tuple.area_code == "US")
      emit(year, {'year' => year, 'price' => tuple.price });
  }

  reduce(key, tuples) {
    double sum_price = 0;
    foreach (tuple in tuples) {
      sum_price += tuple.price;
    }
    emit(key, sum_price);
  }
  ```
Relational Operators as Map/Reduce jobs

- **Sorting**
  - SQL Query:
    ```sql
    SELECT *
    FROM sales
    ORDER BY year
    ```
  - Map/Reduce job:
    ```java
    map(key, tuple) {
      emit(YEAR(tuple.date) DIV 10, tuple);
    }
    
    reduce(key, tuples) {
      emit(key, sort(tuples));
    }
    ```
Symmetric Fragment-and-Replicate Join

Nodes in the Cluster

Requires Tricks in map/reduce
Programmer has to worry about parallelization and/or handcode an optimizer

Important generic example. Specialized parallel joins may exploit data locality
Stratosphere

- Infrastructure for Big Data Analytics
- Collaborative research group
  - 3 Universities in greater Berlin area (TUB, HUB, HPI)
  - 5 research groups (DIMA, CIT, DBIS, WBI, IS)
  - 5 professors, 2 postdocs, 9+ PhD students, 11+ MSc students
- Flagship project at DIMA, TU Berlin
- Open-source platform
  - www.stratosphere.eu
  - Used for teaching and research by (among others) UCSD, RWTH, INRIA, KTH, UCI
Stratosphere Research Agenda

- Programming models for Big Data
  - Programming model based on second-order functions
  - Query languages for complex data
  - Intersection with functional programming

- Re-architecting data management systems in massively parallel scale
  - Robust and adaptive query optimization
  - Parallel execution
  - Fault tolerance
  - Resource management

- Use cases on scientific data management, data cleansing and text mining
$res = \text{filter} \; \$e \; \text{in} \; \$\text{emp} \; \text{where} \; \$e\.income > 30000$;
The Stratosphere Stack

Scientific Data  Life Sciences  Linked Data

Stratosphere
Above the Clouds

Query Processor

Compiler

PACT Optimizer

Nephele

$\text{res} = \text{filter } \$e \text{ in } \$\text{emp}$
$\text{where } \$e.\text{income} > 30000;$
Stratosphere Use Cases

10TB

950km, 2km resolution

1100km, 2km resolution

3 months, 1 km resolution

24/05/12

DIMA – TU Berlin
The Stratosphere Stack

$\text{res} = \text{filter} \, \text{e} \, \text{in} \, \text{emp} \, \text{where} \, \text{e.income} > 30000;
Currently a few high-level language efforts
  □ Inspired by Jaql, XQuery (@INRIA), Pig (@KTH), SQL (@TUB)
  □ “Official”: Jaql-inspired SIMPLE language (HPI)

JSON data model, efficient packing to records

Set of common operators
  □ Read, write, filter, transform, intersect, replace, group, join, ...
  □ Support for Java UDFs
  □ Extensible: libraries for domain-specific operators, e.g., data cleansing, text mining, ...

Compiler translates query to *PACT program*, applies logical optimizations
The Stratosphere Stack

$\text{res} = \text{filter } e \in \text{emp} \text{ where } e.\text{income} > 30000$;
The Nephele Execution Engine

- Executes Nephele schedules
  - DAGs of already parallelized operator instances
  - Parallelization already done by PACT optimizer
- Design decisions
  - Designed to run on top of an IaaS cloud
  - Predictable performance
  - Scalability to 1000+ nodes with flexible fault-tolerance
- Permits network, in-memory (both pipelined), file (materialization) channels

*D. Warneke, O. Kao: Nephele: Efficient Parallel Data Processing in the Cloud. SC-MTAGS 2009*
Nephele Execution Engine

- Executes Nephele schedules
  - compiled from PACT programs

- Design goals
  - Exploit scalability/flexibility of clouds
  - Provide predictable performance
  - Efficient execution on 1000+ nodes
  - Introduce flexible fault tolerance mechanisms

- Inherently designed to run on top of an IaaS Cloud
  - Can exploit on-demand resource allocation
  - Heterogeneity through different types of VMs possible
  - Knows Cloud’s pricing model
Nephele Architecture

- Standard master worker pattern
- Workers can be allocated on demand

![Diagram showing the Nephele Architecture](image)

- **Client** connects to the **Compute Cloud** through the **Public Network (Internet)**.
- The **Compute Cloud** consists of a **Master** and multiple **Workers** connected by a **Private / Virtualized Network**.
- **Persistent Storage** is also part of the system.

Workload over time chart is shown in the top right corner of the image.
Structure of a Nephele Schedule

- Nephele Schedule is represented as DAG
  - Vertices represent tasks
  - Edges denote communication channels

- Mandatory information for each vertex
  - Task program
  - Input/output data location (I/O vertices only)

- Optional information for each vertex
  - Number of subtasks (degree of parallelism)
  - Number of subtasks per virtual machine
  - Type of virtual machine (#CPU cores, RAM...)
  - Channel types
  - Sharing virtual machines among tasks
Internal Schedule Representation

- Nephele schedule is converted into internal representation

- Explicit parallelization
  - Parallelization range (mpl) derived from PACT
  - Wiring of subtasks derived from PACT

- Explicit assignment to virtual machines
  - Specified by ID and type
  - Type refers to hardware profile
Issues with on-demand allocation:
- When to allocate virtual machines?
- When to deallocate virtual machines?
- No guarantee of resource availability!

Stages ensure three properties:
- VMs of upcoming stage are available
- All workers are set up and ready
- Data of previous stages is stored in persistent manner
Channel Types

- **Network channels (pipeline)**
  - Vertices must be in *same* stage

- **In-memory channels (pipeline)**
  - Vertices must run on same VM
  - Vertices must be in *same* stage

- **File channels**
  - Vertices must run on same VM
  - Vertices must be in *different* stages
Demonstrates benefits of dynamic resource allocation

Challenge: Sort and Aggregate
- Sort 100 GB of integer numbers (from GraySort benchmark)
- Aggregate TOP 20% of these numbers (exact result!)

First execution as map/reduce jobs with Hadoop
- Three map/reduce jobs on 6 VMs (each with 8 CPU cores, 24 GB RAM)
- TeraSort code used for sorting
- Custom code for aggregation

Second execution as map/reduce jobs with Nephele
- Map/reduce compatibility layer allows to run Hadoop M/R programs
- Nephele controls resource allocation
- Idea: Adapt allocated resources to required processing power
The Stratosphere Stack

Scientific Data  Life Sciences  Linked Data

StratoSphere
Above the Clouds

Query Processor

Compiler

PACT Optimizer

Nephele

$\text{res} = \text{filter } \text{e in } \text{emp } \text{where } \text{e.income} > 30000;$
- **PACT program** compiler from higher-level language, or written directly by the programmer.

- Program is a hard-wired data flow DAG of PACT (Parallelization ConTracts) operators.

- PACT operator consists of:
  - A signature taken from a fixed set of second-order functions.
  - A user-defined first-order function written in Java.

```java
Record mapStub (Record i) {
    Record o = i.copy();
    if (i.getField(0) < 3.0) {
        o.setField(2,"aab");
        return o;
    }
}
```

![Diagram of PACT Programming Model]

**Map**

```
Sink
Record mapStub (Record i) {
    Record o = i.copy();
    if (i.getField(0) < 3.0) {
        o.setField(2,"aab");
        return o;
    }
}
```
Parallelization Contracts

- Describe how input is partitioned in groups
  - “What is processed together”
- First-order UDF called once per input group
- Map PACT
  - Each input record forms a group,
  - Each record is independently processed by UDF
- Reduce PACT
  - One attribute is the designated key
  - All records with same key value form a group
Cross PACT
Each pair of input records forms a group
Distributed Cartesian product

Match PACT
Each pair with equal key values forms a group
Distributed equi-join

CoGroup PACT
All pairs with equal key values form a group
2D Reduce

More PACTs currently under development

For similarity operators, stream processing, etc
Parallelization of PACT programs

- Knowledge of PACT signature permits *automatic parallelization*
- Parallelizing and executing Match
  - Broadcast or partition
  - Same as parallel joins
  - Sort-merge or hash
- Volcano-style optimizer
  - Interesting properties
  - Need output contracts
Reordering of PACT Programs

- Algebraic reordering as in relational DBMSs
- **Problem:** UDFs are black-box Java functions – semantics unknown
- Derive conflicts on data flow attributes using static code analysis of UDFs
- Prove and apply reordering rules
- Can emulate most relational optimizations
  - Selection and join reordering
  - Limited aggregation push-down
Reordering of PACT Programs

- Novel notion of Read and Write Sets
  - Read set: Attributes that may affect output of operator
  - Write set: Attributes with different values in output
- Given R/W sets, formally prove reordering conditions
  - Map: Only read conflicts
  - Reduce: In addition, key groups are preserved
  - Cross/Match/CoGroup are extensions of Map/Reduce
- Automatically derive R/W sets by analyzing operator code
  - Shallow static code analysis using Use-Def and Def-Use chains
  - Difficulty comes from different code paths – guarantee safety
Many non-trivial data analytics applications involve some form of iteration or recursion

- E.g., ML and graph analysis
- Iterations realized as fixpoints over PACT programs

Several implications
- Execution, optimization, fault tolerance, etc

Differential iterations
- Ala datalog semi-naïve evaluation
- Permit asynchronous execution
The Power of Optimization

The diagram illustrates the process of database optimization, specifically focusing on the reduction and matching steps. The process involves:

1. **Reduce (on tid)**: Reduces the data by summing up partial ranks.
2. **Match (on pid)**: Matches the data based on a condition.
3. **Join**: Joins the reduced and matched data.
4. **ProbeHT (pid)**: Performs a hash table lookup.
5. **BuildHT (pid)**: Builds a hash table.
6. **Cache**: Utilizes cache for efficient data access.
7. **Partition (pid)**: Divides the data for further processing.
8. **Part./sort**: Sorts the partitioned data.

Each step is represented by a node in the diagram, connected by arrows indicating the flow of data and operations.
Each iteration changes only few elements
E.g., the state change of a graph vertex triggered by state changes at the neighborhood

Encapsulate in an operator with two functions
- $\Delta$ updates the workset: elements that may update
- $u$ updates current solution
- Solution indexed and implicitly combined with changed elements

Can emulate Pregel
Summary

- **Stratosphere**
  - A modern, open-source platform for Big Data Analytics

- **Research themes in Stratosphere**
  - Automatic optimization of UDFs
  - Robust query optimization
  - Iteration and recursion
  - Flexible fault-tolerance mechanisms
  - Adaptive optimization and execution
  - Concrete applications: text mining, data cleansing

- **Open-source system, used externally for teaching and research**
  - Version 0.2 coming soon
The data management landscape is changing

- Hadoop made open-source popular, embraced by big players, startups. New systems are emerging

**Retain key attractive aspects of MapReduce**

- Arbitrary Java UDFs
- Massive parallelism, fault tolerance
- Analysis of “in situ” data, “no knobs”

**Stratosphere pushes paradigm forward**

- Data independence aspects from RDBMs
- Query optimization, indexing
- Iterative/recursive algorithms
- Benchmarking (with IBM CAS)
Screenshot of PACT Data Flow Visualizer

Screenshot of Nephele Parallel Execution Visualizer