Big Data Analytics with MapReduce

VL Implementierung von Datenbanksystemen
05-Feb-13

Astrid Rheinländer

Wissensmanagement in der Bioinformatik
What is Big Data?

• “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
  → in a few years exabytes (SKA telescope)

• Challenges
  • Storage
  • Analysis
  • Search
  • ...
Example – Twitter

2012: 63 Billion Tweets, 8.5 TB data

Business areas: The data – sell access to content
Advertiesement

Example – Cern, LHC

- 2012: LHC experiments generated 22PB of data → 99% have already been thrown away

- Analysis needs 100k modern processors to evaluate experiments

- Data is stored & processed in LHC Computing Grid
  - 150 data & compute centers around the world
  - Heterogeneous architecture

[http://lh cathome.web.cern.ch]
[http://grid-monitoring.cern.ch/myegi/gridmap/]
Big Data Landscape

© Matt Turck (@mattturck) and ShivanZilis (@shivonz) Bloomberg Ventures
Big Data Landscape

Big Data Landscape (Version 2.0)

© Matt Turck (@mattturck) and ShivonZilis (@shivonz) Bloomberg Ventures
Overview

• MapReduce
  – Programming model
  – Architecture and execution
  – Distributed File System – HDFS
  – Error handling & performance issues

• MapReduce vs. parallel Databases

• Extensions of MapReduce
Programming Model

- Inspired by functional programming concepts map and reduce
- Operates on key-value pairs

Map
- Process key-value pairs individually
  → with UDF
- Generates key/value pairs
- Example (LISP):
  \[(\text{mapcar } '1+ ' (1 2 3 4)) \Rightarrow (2 3 4 5)\]
Programming Model

- Inspired by functional programming concepts **map** and **reduce**
- Operates on key-value pairs

**Map**
- Process key-value pairs individually → with UDF
- Generates key/value pairs
- Example (LISP):
  
  \[
  \text{mapcar} \ '1+ \ '(1 \ 2 \ 3 \ 4) \Rightarrow (2 \ 3 \ 4 \ 5)
  \]

**Reduce**
- Merges intermediate key-value pairs with same key
- Example (LISP):
  
  \[
  \text{reduce} \ '+ \ '(1 \ 2 \ 3 \ 4) \Rightarrow 10
  \]
Example – Term Count

• **Input data:**
  
  Documents

  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

• **Task:**

  How often are terms contained in the set of documents?
public static class Map extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output) {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}

Example – Term Count

1, to be, or not ...
2, whether 'tis ...
3, the slings ...
4, or to take ...

to, 1
be, 1
or, 1
not, 1
whether, 1
'tis, 1
the, 1
slings, 1
or, 1
to, 1
take, 1...

Astrid Rheinländer: Big Data Analytics with MapReduce
Example – Term Count

```java
public static class Map extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
                     OutputCollector<Text, IntWritable> output) {

        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
```

<table>
<thead>
<tr>
<th>Term</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>to, 1</td>
<td>1</td>
</tr>
<tr>
<td>be, 1</td>
<td></td>
</tr>
<tr>
<td>or 1</td>
<td></td>
</tr>
<tr>
<td>not, 1</td>
<td></td>
</tr>
<tr>
<td>whether, 1</td>
<td></td>
</tr>
<tr>
<td>'tis, 1</td>
<td></td>
</tr>
<tr>
<td>the, 1</td>
<td></td>
</tr>
<tr>
<td>slings, 1</td>
<td></td>
</tr>
<tr>
<td>or, 1</td>
<td></td>
</tr>
<tr>
<td>to, 1</td>
<td></td>
</tr>
<tr>
<td>take, 1</td>
<td></td>
</tr>
</tbody>
</table>
Example – Term Count

```java
public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
                        OutputCollector<Text, IntWritable> output) {
        int sum = 0;

        while (values.hasNext()) {
            sum += values.next().get();
        }

        output.collect(key,
                        new IntWritable(sum));
    }
}
```

reduce

- not, 1
- 'tis, 1
- whether, 1
- or, 1
- or, 1
- be, 1
- to, 1
- to, 1
- the, 1
- take, 1
- slings, 1

not, 1
whether, 1
'tis, 1
the, 1
slings, 1
or, 2
be, 1
to, 2
take, 1
Example – Term Count

```java
public static class Reduce extends MapReduceBase
    implementsReducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output) {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key,
            new IntWritable(sum));
    }
}
```
Example – Term Count

- Programmer’s point of view:
  - Partition and distribute documents to mappers
  - Store and group intermediate term counts
  - Distribute intermediate data to reducers
  - Collect and return final results
  - Deal with failures and exceptions

- MapReduce framework takes care of distributed execution

Example – Term Count

- \( \text{doc}_1 \)
- \( \ldots \)
- \( \text{doc}_n \) → \text{map} → \text{reduce} → \( \text{term}_{1r}, 1 \)
  - \( \ldots \)
  - \( \text{term}_{zr}, 1 \)
- \( \text{term}_{1r}, \text{count}_1 \)
  - \( \ldots \)
  - \( \text{term}_{zr}, \text{count}_z \)
Overview

• MapReduce
  – Programming model
  – Architecture and execution
  – Distributed File System – HDFS
  – Error handling & performance issues

• MapReduce vs. parallel Databases

• Extensions of MapReduce
Cluster architecture

- Master/Worker architecture
  - Workers are commodity hardware
  - Masters are usually well equipped
  - Shared nothing

- Examples:
  - Google
    - cluster with several thousand nodes
    - 2008: 2 x86 CPUs, 4-8GB RAM, Linux, IDE HDDs
  - Facebook
    - 2010: 2000 nodes (8/16 cores), 32 GB RAM, 21PB storage in HDFS
MapReduce - Execution

- **User**
  - Specification of MapReduce job
  - Start execution engine
  - Submit job to execution engine

- **Execution engine**
  - Fork worker nodes
  - Partition input

```
chunk_1
...
chunk_n
```

```plaintext
docs
```
**MapReduce - Execution**

- **Master**
  - Manages entire execution
  - Assigns tasks and input splits to idle workers
  - Tracks status of current job and all tasks (waiting, running, finished)
  - Tracks status of worker nodes

![Diagram of MapReduce execution showing the master, workers, and chunk distribution.](image)
MapReduce - Execution

- **Map-Worker**
  - Reads assigned splits
  - Parses key-value pairs
  - Executes map function for each pair
  - Buffers intermediate data in memory
MapReduce - Execution

- **Map-Worker**
  - Buffered intermediate data are periodically written to local disks using some partition function
  - Notify master about locations of intermediate data when map() is finished
  - Master pushes locations incrementally to reduce-workers
MapReduce - Execution

- **Reduce-Worker**
  - Read assigned splits from map workers disks (RPC)
  - Usually processes more than one key
  → Sort data by intermediate key

![MapReduce diagram](image)

- chunk<sub>1</sub>
- ...
- chunk<sub>n</sub>

- worker
- Remote read
- worker

**Execution engine**

Master
MapReduce - Execution

- Reduce-Worker
  - Executes reduce() function once for each intermediate key
  - reduce() usually iterates over entire list of values per key → aggregation
  - Result is written to distributed FS
  - Usually one file per reduce worker
MapReduce - Execution

- All Map/Reduce jobs are finished
  - Master notifies user program
  - User gets access to result
Overview

• MapReduce
  – Programming model
  – Architecture and execution
  – Distributed File System – HDFS
  – Error handling & performance issues

• MapReduce vs. parallel Databases

• Extensions of MapReduce
HDFS architecture

- HDFS: Hadoop distributed file system

- Why a special file system?
  - Different priorities/workload
    - Goal: store very large data redundantly on commodity hardware
    - manage much larger data compared to 'standard' fs

- Assumptions:
  - Nodes fail all the time
  - Mostly read/append file operations, few rewrites
    - streaming access pattern
  - Network latency should not interrupt computations
  - Relatively small number of large files
HDFS architecture

- Master/Worker architecture

- Master: Name node
  - Access to DFS
  - Replication
  - Metadata

- Workers: Data nodes
  - Local storage in cluster
  - I/O and maintenance operations

- Client talks to data nodes and name node

→ Data does not pass name node
HDFS

- Use blocks to store (parts of) files
  - Default: 64MB
  - Recommended: 128 MB
  - Unix: 4KB
  - Advantages:
    - Fixed size
    - Easy to calculate how many blocks fit on disk
    - Files can be larger than any single disk in cluster
    - Well suited for replication
HDFS

- Use blocks to store (parts of) files
  - Default: 64MB
  - Recommended: 128 MB
  - Unix: 4KB
  - Advantages:
    - Fixed size
    - Easy to calculate how many blocks fit on disk
    - Files can be larger than any single disk in cluster
    - Well suited for replication

- Runs on top of OS file system
  - OS fs blocks under the hood

→ one HDFS block consists of multiple OS fs blocks
Architecture - Summary

- **Pros:**
  - Low costs
  - Extensible
  - Nodes are easily exchangeable

- **But:**
  - Nodes fail regularly for various reasons
    - Disk failures
    - Broken controllers
    - Cable breaks
    - Network partitioning
    - ...
  - Network bandwidth is a potential bottleneck

→ Performance optimization and error handling needed
Overview

- MapReduce
  - Programming model
  - Architecture and execution
  - Distributed File System – HDFS
    - Error handling & performance issues

- MapReduce vs. parallel Databases

- Current trends
Types of errors

- HDFS data node fails
- Worker node fails
- Master / Name node fails
- (Performance bottlenecks)
HDFS – Replication

- **Goal:** Reliability by replication

- blocks are replicated on $\geq 3$ data nodes

- **Placement**
  - 1 copy on local node
  - 1 copy on remote node
  - 1 copy on different node in same remote rack
  - Additional copies randomly placed

- Client reads from closest copy
- Integrity via checksums
HDFS – Replication

- Replication engine on name node

- Name node detects data node failures
  → heartbeat messages

- Data node failure:
  → move tasks to intact replicas

→ Moving computation is cheaper than moving data
Worker fails

- Master pings all workers regularly (heartbeat messages)
- Worker answer within timeout

- If worker does not answer, redistribute tasks to other workers
  - Map/Reduce tasks with status “assigned” or “running” get status “waiting for execution”
  - Finished Map tasks need to be executed again
  - Finished Reduce tasks do not need to be re-run
Master or Name node fails

→ single points of failure
  • Execution is aborted and needs to be restarted
    • Solutions:
      → Shadow master / secondary name node
      → Checkpoints
Master or Name node fails

→ single points of failure
  • Execution is aborted and needs to be restarted
    • Solutions:
      → Shadow master / secondary name node
      → Checkpoints
Performance Bottlenecks

- „Stragglers“
  - Tasks which run much longer than others

  ➔ Backup-Tasks
Backup tasks

• Problem: stragglers significantly slow down whole MapReduce job

• Reasons:
  • Other jobs might consume resources on a machine
  • Bad disks with correctable errors transfer data very slowly
  • Not enough RAM – machine starts swapping
  • ...

• Solution:
  • When MapReduce job is close to finishing
  • Spawn copies of remaining in-progress tasks
  \( \rightarrow \) Keep results from task that finishes first
  • Takes a few percent resource overhead
Performance Bottlenecks

- „Stragglers“
- Network traffic

→ Locality
Locality

- Goal: conserve bandwidth
- Schedule map tasks to locations of processed chunks
  - physically on same machine
  - many machines can read data with local disk speed
Locality

- Goal: conserve bandwidth
- Schedule map tasks to locations of processed chunks
- If same machine not possible: \(\rightarrow\) same rack / switch
Overview

• MapReduce
  – Programming model
  – Architecture and execution
  – Distributed File System – HDFS
  – Error handling & performance issues

• MapReduce vs. parallel Databases

• Current trends
MapReduce vs. parallel DBMS

- DeWitt & Stonebraker:

  „A major step backwards“
MapReduce vs. parallel DBMS

- DeWitt & Stonebraker:
  
  „A major step backwards“

- Criticism: „Teradata has done this 20 years ago“

  - Not really new
    - Functionality can be reached by UDFs in parallel DBMS
    - Parallel DBMS provide good scalability

  - Interface too low-level
  - Write intermediate data to disk instead of pipelining/streaming
  - Lack of schemata obstructs performance optimizations
MapReduce != parallel DBMS

Focus is different!

- **Parallel DBMS:**
  - Query very large data sets
  - Transactions
  - User management
  - High reliability
  - Very expensive

- **MapReduce:**
  - ETL tasks
  - Complex analytics
  - Pragmatism instead of perfect reliability
  - Much cheaper
  - Open source systems
Extensions of MapReduce

- Data flow languages on top of MapReduce
  → Jaql, Hive, Pig, ...

- Record-based data model

- Optimizers

- Extensions of programming model
  → Example: Stratosphere
Stratosphere research unit

Use-Cases
- Scientific Data
- Life Sciences
- Linked Data

Jointly carried out by TU, HU, and HPI

→ web-scale distributed data analytics system
→ Database-inspired approach
→ Analytical workload
Stratosphere research unit

Use-Cases

Scientific Data  Life Sciences  Linked Data

Stratosphere system

→ web-scale distributed data analytics system

→ Database-inspired approach

→ Analytical workload
Stratosphere programming model

- **PArallelization ConTracts (PACTs)**
  → Generalization of MapReduce
• Parallelization Contracts
  → Generalization of MapReduce

Two inputs

Matches tuples that share the same key

Equi-join on key

Each pair handed to UDF
Stratosphere programming model

- Parallelization Contracts
  → Generalization of MapReduce

Two inputs
Cartesian product of both data sets
Each pair handed to UDF
Stratosphere programming model

- Parallelization Contracts
  → Generalization of MapReduce

Reduce on two inputs
Each key group is handed to UDF