Big Data
6. Massive Parallel Processing (MapReduce)
Let's begin with a field experiment
200 blocks, 8 different shapes
How many of each?
200 pieces distributed to 10 volunteers
Task 1 (10 people)
Task 1 (10 people)
Task 2 (8 people)
Task 2 (8 people) – part 1 aka "The big mess"
Task 2 (8 people) – part 2

1
2
1
3
1
Final summary

<table>
<thead>
<tr>
<th>Shape</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cylinder</td>
<td>8</td>
</tr>
<tr>
<td>Cube</td>
<td>6</td>
</tr>
<tr>
<td>Cylinder</td>
<td>4</td>
</tr>
<tr>
<td>Triangle</td>
<td>2</td>
</tr>
<tr>
<td>Cube</td>
<td>10</td>
</tr>
<tr>
<td>Rectangular</td>
<td>5</td>
</tr>
<tr>
<td>Rectangular</td>
<td>3</td>
</tr>
<tr>
<td>Semi-circle</td>
<td>7</td>
</tr>
</tbody>
</table>
Let's go!
So far, we have...

Storage as file system (HDFS)
So far, we have...

- Storage as file system (HDFS)
- Storage as tables (HBase)
Data is only useful if we can query it

- Storage as file system (HDFS)
- Storage as tables (HBase)

Querying
... in parallel

- Storage as file system (HDFS)
- Storage as tables (HBase)
- Querying
Data Processing

Input data
Data Processing

Input data

Query
Data Processing

Input data

Query

Output data
MapReduce
Data Processing: data comes in chunks

Query
Data Processing: the ideal case
Data Processing: the worst case
Data Processing: the typical case
Data Processing: Map here...
Data Processing: ... and shuffle there
A common and useful sub-case: MapReduce

Input data

Map  Map  Map  Map  Map  Map  Map  Map  Map

Shuffle

Reduce  Reduce  Reduce  Reduce  Reduce  Reduce  Reduce  Reduce  Reduce

Output data
Data Processing: Data Model

Input data

Intermediate data (shuffled)

Output data
Data Processing: Data Shape
Data Processing: Data Types

key type 1 -> value type 1

Map
Map
Map
Map
Map
Map
Map
Map

key type I -> value type I

Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce

key type A -> value type A
Data Processing: Most often

key type 1 -> value type 1

key type A -> value type A

key type A -> value type A
Splitting

Split

key 1
value

key 2
value

key 3
value

key 4
value
Mapping function

Map

key 1 | value
---|---
key I | value
key II | value
Mapping function... in parallel

Map

Map

Map

<table>
<thead>
<tr>
<th>key 1</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key 2</td>
<td>value</td>
</tr>
<tr>
<td>key 3</td>
<td>value</td>
</tr>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>
Put it all together

<table>
<thead>
<tr>
<th>key I</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key II</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key I</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key II</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key I</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>
### Sort by key

<table>
<thead>
<tr>
<th>key I</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key I</td>
<td>value</td>
</tr>
</tbody>
</table>

### Sorted Table

<table>
<thead>
<tr>
<th>key I</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>
### Partition

<table>
<thead>
<tr>
<th>key I</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key I</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key II</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key II</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key III</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>key I</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key I</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key II</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key II</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key III</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>
Reduce function

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key I</td>
<td>value</td>
</tr>
</tbody>
</table>

Reduce

key A | value
Reduce function (with identical key sets)

<table>
<thead>
<tr>
<th>key A</th>
<th>value A</th>
</tr>
</thead>
<tbody>
<tr>
<td>key A</td>
<td>value B</td>
</tr>
<tr>
<td>key A</td>
<td>value C</td>
</tr>
</tbody>
</table>

Reduce
Reduce function (most generic)

Reduce

(key A value)
(key B value)

More is fine, but uncommon
Reduce function... in parallel

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key I</td>
<td>value</td>
</tr>
<tr>
<td>key II</td>
<td>value</td>
</tr>
<tr>
<td>key III</td>
<td>value</td>
</tr>
</tbody>
</table>

Reduce

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key A</td>
<td>value</td>
</tr>
</tbody>
</table>

Reduce

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key B</td>
<td>value</td>
</tr>
</tbody>
</table>

Reduce

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key C</td>
<td>value</td>
</tr>
</tbody>
</table>
Overall

Map

Sort

Partition

Reduce
Input/Output formats
Input and output formats

From/to tables

From/to files
## Formats: tabular

<table>
<thead>
<tr>
<th>Row ID</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
</tr>
<tr>
<td>002</td>
<td></td>
</tr>
<tr>
<td>0A1</td>
<td></td>
</tr>
<tr>
<td>1E0</td>
<td></td>
</tr>
<tr>
<td>22A</td>
<td></td>
</tr>
<tr>
<td>4A2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row ID</th>
<th>RDBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
</tr>
<tr>
<td>002</td>
<td></td>
</tr>
<tr>
<td>0A1</td>
<td></td>
</tr>
<tr>
<td>1E0</td>
<td></td>
</tr>
<tr>
<td>22A</td>
<td></td>
</tr>
<tr>
<td>4A2</td>
<td></td>
</tr>
</tbody>
</table>
## Formats: tabular

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Arrows pointing from left to right]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Arrows pointing from right to left]
Formats: files (e.g., from HDFS)

Text  

KeyValue  

SequenceFile
Text files

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed...

<table>
<thead>
<tr>
<th>0</th>
<th>Lorem ipsum dolor</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>sit amet,</td>
</tr>
<tr>
<td>27</td>
<td>consectetur</td>
</tr>
<tr>
<td>38</td>
<td>adipiscing elit, sed</td>
</tr>
</tbody>
</table>
Text files: NLine

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed...

<table>
<thead>
<tr>
<th></th>
<th>Lorem ipsum dolor sit amet,</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>consectetur adipiscing elit, sed</td>
</tr>
</tbody>
</table>

...
Key-Value

Lorem●ipsum dolor
sit●amet,
consectetur
adipiscing●elit, sed
...

Lorem | ipsum dolor
sit | amet,
consectetur
adipiscing | elit, sed
...
Sequence files

Hadoop binary format

Stores generic key-values

| KeyLength | Key | ValueLength | Value |
Optimization
Optimization

Map

key type 1 -> value type 1

key type A -> value type A

Reducer

key type A -> value type A
How to reduce* the amount of data shuffled around?

*pun intended (Eselsbrücke)
Optimization: Combine

Mapper

Map → Map → Map → Map → Map → Map

(key type 1 -> value type 1)

Combine → Combine → Combine → Combine → Combine → Combine

(key type A -> value type A)

Reducer

Reduce → Reduce → Reduce → Reduce → Reduce → Reduce → Reduce

(key type A -> value type A)
Combine: the 90% case

Often, the **combine** function is identical to the **reduce** function.

Disclaimer: there are assumptions
Combine=Reduce: Assumption 1

Key/Value types must be **identical** for reduce input and output.
Combine=Reduce : Assumption 2

Reduce function must be

Commutative

and

Associative
Optimization: Bring the Query to the Data
MapReduce: the APIs
Supported frameworks

Hadoop MapReduce

Java

Streaming
Supported frameworks

- Hadoop MapReduce
- Java
- Streaming
Java API: Mapper

```java
import org.apache.hadoop.mapreduce.Mapper;

public class MyOwnMapper extends Mapper<K1, V1, K2, V2>{

    public void map(K1 key, V1 value, Context context)
        throws IOException, InterruptedException
    {
        ...
        K2 new-key = ...
        V2 new-value = ...
        context.write(new-key, new-value);
        ...
    }
}
```
import org.apache.hadoop.mapreduce.Reducer;

public class MyOwnReducer extends Reducer<K2, V2, K3, V3>{

    public void reduce
        (K2 key, Iterable<V2> values, Context context)
        throws IOException, InterruptedException
    {
        ...
        K3 new-key = ...
        V3 new-value = ...
        context.write(new-key, new-value);
        ...
    }
}
Java API: Job

import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setMapperClass(MyOwnMapper.class);
        job.setReducerClass(MyOwnReducer.class);
        FileInputFormat.addInputPath(job, ...);
        FileOutputFormat.setOutputPath(job, ...);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
Java API: Combiner (=Reducer)

```java
import org.apache.hadoop.mapreduce.Job;

public class MyMapReduceJob {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");

        job.setMapperClass(MyOwnMapper.class);
        job.setCombinerClass(MyOwnReducer.class);
        job.setReducerClass(MyOwnReducer.class);

        FileInputFormat.addInputPath(job, ...);
        FileOutputFormat.setOutputPath(job, ...);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```
Java API: InputFormat classes

InputFormat
DBInputFormat  RDBMS
TableInputFormat  HBase
FileChooserInputFormat
KeyValueTextInputFormat
SequenceFileInputFormat
TextInputFormat
FixedLengthInputFormat
NLineInputFormat
Java API: OutputFormat classes

OutputFormat
  DBOutputFormat RDBMS
  TableOutputFormat HBase
FileoutputFormat
  SequenceFileOutputFormat Sequence file
  TextOutputFormat Text
  MapFileOutputFormat
MapReduce: the physical layer
Possible storage layers

- Local Filesystem
- HDFS
- S3
- Azure Blob Storage

Hadoop MapReduce
Possible storage layers

- Local Filesystem
- HDFS
- S3
- Azure Blob Storage
Hadoop MapReduce: Numbers

Several TBs of data

1000s of nodes
Hadoop infrastructure (version 1)
Master-slave architecture

Master

Slave  Slave  Slave  Slave  Slave  Slave  Slave
Hadoop infrastructure (version 1)

Namenode
+
JobTracker

Datanode
+
TaskTracker

Datanode
+
TaskTracker

Datanode
+
TaskTracker

Datanode
+
TaskTracker

Datanode
+
TaskTracker

Datanode
+
TaskTracker
Hadoop infrastructure (version 1)

Namenode

+ 

JobTracker

Bring the Query to the Data

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker
Tasks

Task = Map or Reduce
Splits
Splits
Splits vs. map tasks

1 split
= 1 map task
In practice

1 split = 1 block
(subject to min and max size)
Splits vs. blocks: possible confusion

Logical Level (MapReduce)

Split

Record (key/value pair)

Bit

Physical Level (HDFS)

Block
Records across blocks

Logical Level (MapReduce)

Split

Physical Level (HDFS)

Block
Records across blocks

Logical Level (MapReduce)

Split

Physical Level (HDFS)

Remote read

Block
Fine-tuning to adjust splits to blocks

Logical Level (MapReduce)

Physical Level (HDFS)
Hadoop infrastructure (version 1)

Namenode +

JobTracker

/dir/file

Datanode +

TaskTracker

Datanode +

TaskTracker

Datanode +

TaskTracker

Datanode +

TaskTracker

Datanode +

TaskTracker

Datanode +

TaskTracker
Hadoop infrastructure: map tasks

As many map tasks as splits

Namenode

JobTracker

/dir/file

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

Datanode + TaskTracker

87
Hadoop infrastructure: map tasks

As many map tasks as splits

Occasionally not possible to co-locate task and block

Namenode +

JobTracker

/dir/file

Datanode

TaskTracker

+ + + + + + + +
Hadoop infrastructure: reduce tasks

Namenode
+ JobTracker

A few reduce tasks
/dir/file

Datanode + TaskTracker
Datanode + TaskTracker
Datanode + TaskTracker
Datanode + TaskTracker
Datanode + TaskTracker
Datanode + TaskTracker
Datanode + TaskTracker
Hadoop infrastructure: shuffling (inbetween)
Issue 1: Tight coupling
Issue 2: Scalability

Namenode + JobTracker

Only one!

Datanode
+ TaskTracker
Datanode
+ TaskTracker
Datanode
+ TaskTracker
Datanode
+ TaskTracker
Datanode
+ TaskTracker
Datanode
+ TaskTracker
Shuffling phase

Mappers

Reducer
Shuffling phase

Each mapper sorts its output key-value pairs
Spilling to disk

Key-value pairs are spilled to disk if necessary
Shuffling phase

Mappers

Reducer

Gets its key value pairs over HTTP
YARN
YARN

Yet Another Resource Negotiator
YARN

Resource Management

Job Scheduling & Monitoring
YARN architecture
Remember...

It does ring a bell, doesn't it?
Master-slave architecture
HDFS server architecture
Yarn

ResourceManager

NodeManager

Container

NodeManager

Container

NodeManager

Container

NodeManager

Container

NodeManager
ResourceManager

Scheduler + Applications Manager
YARN: Client posts a job

Client posts a job to the ResourceManager. The job is then distributed to various NodeManagers, each managing a Container.
YARN: RM allocates an Application Master

Client → ResourceManager (Scheduler) → ApplicationMaster Protocol

Job

Application Master

Container

Container

NodeManager → NodeManager → NodeManager → NodeManager → NodeManager → NodeManager
YARN: Application Master asks for Containers

Client

ResourceManager (Applications Manager)

Job

ApplicationMaster Protocol

Application Master

Container

Container

NodeManager

NodeManager

NodeManager

NodeManager

NodeManager
Application Master communicates with containers
Next week outlook: forward compatibility with DAGs of tasks