7th Slide Set Cloud Computing

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences
(1971–2014: Fachhochschule Frankfurt am Main)
Faculty of Computer Science and Engineering
christianbaun@fb2.fra-uas.de

Agenda for Today

- MapReduce/Hadoop
  - Basic information
  - Basics
  - Operation steps
  - Fields of application
  - Google PageRank
  - Components and extensions
    - Hadoop Distributed File System (HDFS)
    - Pig
    - Hive
    - HBase
  - Cloudera
    - Installation guide
    - Examples with the installation
  - Amazon Elastic MapReduce
  - Other MapReduce implementations

MapReduce/Hadoop

- Companies such as Google, Facebook and Twitter need to store and process several TB and PB of data every day
- Today, the processing of large amounts of data is often done via distributed computing in Clusters
- 2 fundamental requirements exist:
  1. Data must be **stored** as **efficient** as possible
  2. Data must be **processed** as **efficient** as possible

Data Storage and Processing

- In order to efficiently store data, Google developed the distributed Cluster file system **Google File Sytem** (GFS)
  - GFS operates according to the Master-Slave principle
  - Fault tolerance in case of hardware failures achieves GFS via replication

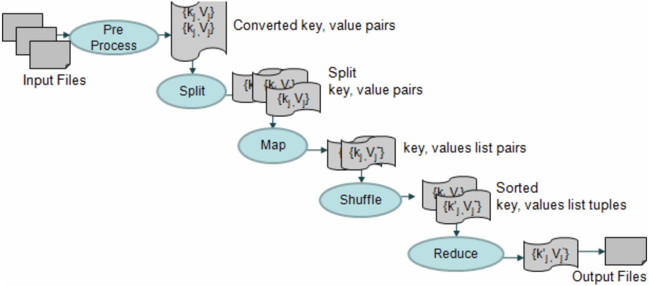
S. Ghemawat, H. Gobioff, S. Leung. **The Google File System**. Google. 2003
`http://labs.google.com/papers/gfs-sosp2003.pdf`

- The standard procedure for distributed systems is typically: Data, which needs to be processed, is transferred to the program
  - A program is executed on a computer and fetches the required input data from a source (e.g. FTP server or database)
  - This procedure is not optimal for large amounts of data, because a bottleneck occurs
  - Solution: The data processing must take place where the data is stored $\Longrightarrow$ This is possible with the **MapReduce programming model**

Two Meanings of MapReduce

Image Source: http://www.pnexpert.com

- MapReduce often means 2 things:
  1. **MapReduce programming model**
     - For parallel data processing in Clusters



  2. **MapReduce frameworks** (e.g. Hadoop)
     - Operate according to the MapReduce programming model
     - Differ in the choice of programming language and the implementation details

Basic information about MapReduce

J. Dean, S. Ghemawat. **MapReduce: Simplified Data Processing on Large Clusters**. Google 2004
http://labs.google.com/papers/mapreduce-osdi04.pdf

- The MapReduce programming model splits tasks into smaller parts and distributes them for parallel processing to different compute nodes
- The final result is created by merging the partial results

Oliver Fischer. **Verarbeiten großer verteilter Datenmengen mit Hadoop**. heise Developer. 2010
http://heise.de/-964755

- Google presented MapReduce in 2003 and the Google File System in 2004
  - The implementations of Google were never published
    - This resulted in the emergence of free (open source) re-implementations

Basic information about Hadoop

- 2005: Doug Cutting implements MapReduce for Nutch
  - Nutch is a free search engine, written in Java
  - `http://nutch.apache.org`
- Cuttings implementation was the basis of the Hadoop project
  - Hadoop is a free implementation of GFS and MapReduce
  - `http://hadoop.apache.org`
- Since 2008 coordinates the Apache Software Foundation the development
  - Hadoop is a top-level project of the Apache Software Foundation
- July 2009: A Hadoop cluster of Yahoo sorted 100 terabytes in 2 hours and 53 minutes (`http://sortbenchmark.org`)

Roots of MapReduce

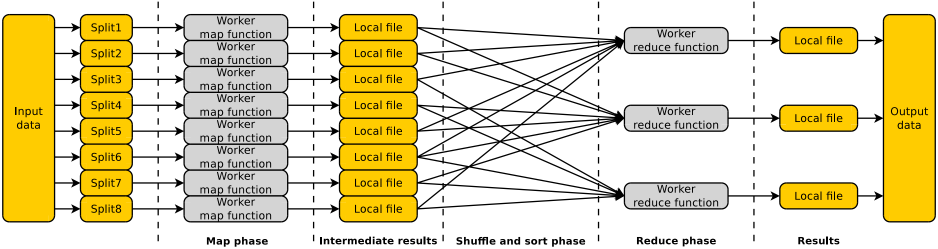
- The MapReduce concept has its roots in Lisp
  - Lisp implements the functions `map()` and `reduce()`
  - The `map`-function of Lisp is called with a list of input values and a function as parameter
    - The supplied function is applied to each value in the list
  - The `reduce`-function of Lisp is also called with a function and a list of input values
    - `reduce` operates similar to `map`, but it combines the results to a single one

MapReduce and functional Programming

- MapReduce bases of the **functional programming** principle
  - Functional programming is a programming style in which programs consist solely of functions
  - Functional programs are a set of (function)-definitions
  - (Function)-definitions implement partial mappings of input data to output data
  - The input data is never changed!
  - The functions are idempotent (free of side effects)
    - For each function call, the same result is returned
  - Only calculations are carried out with input data and then, (intermediate) result are generated
- Google uses MapReduce for the PageRank algorithm

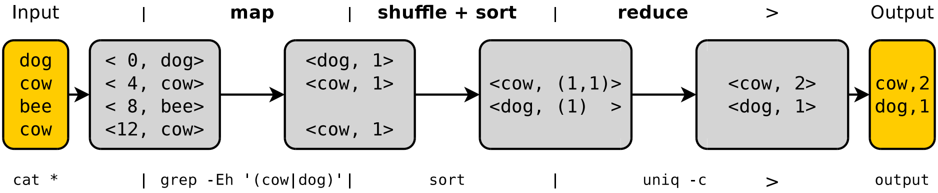
MapReduce

- MapReduce consists of a few steps:
  1. Partitioning of the initial data
  2. Mapping (*map*) the data to a data structure which consists of a key-value pair
  3. Distributing (*shuffle*) and sorting (*sort*) the key-value pairs
  4. Reducing (*reduce*) the key-value pairs to obtain the result



MapReduce – Operation Steps

- The Map phase and the Reduce phase can be executed in parallel on the nodes of a cluster
    - A master coordinates and monitors the MapReduce applications
- The diagram shows that the data processing via MapReduce is similar with the processing in a UNIX pipe



Input | **map** | **shuffle + sort** | **reduce** > Output

```
dog
cow
bee
cow
```

```
< 0, dog>
< 4, cow>
< 8, bee>
<12, cow>
```

```
<dog, 1>
<cow, 1>

<cow, 1>
```

```
<cow, (1,1)>
<dog, (1)  >
```

```
<cow, 2>
<dog, 1>
```

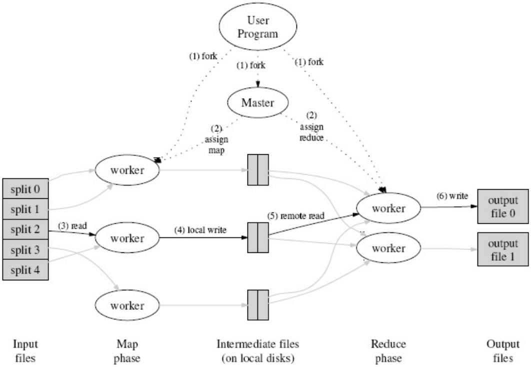
```
cow,2
dog,1
```

```
cat *         | grep -Eh '(cow|dog)'|     sort     |     uniq -c    >     output
```

Source: Christophe Bisciglia, Tob White. An Introduction to Hadoop. 2009

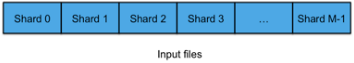
Example: Distributed Frequency Analysis with MapReduce (1/9)

- Objective: Figure out for a large text how many times which words occur



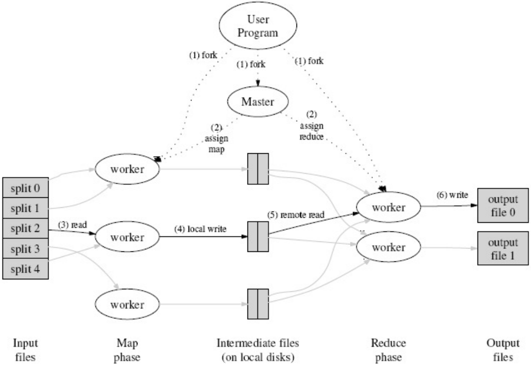
Sources: *Jeffrey Dean, Sanjay Ghemawat*. MapReduce: Simplied Data Processing on Large Clusters. 2004 and `http://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html`

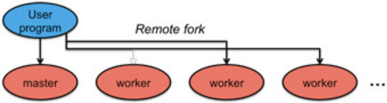
- First, the MapReduce library of the user program splits the input data into `m` parts
  - The parts are called *split* or *shard*
  - It is useful to have at least as many splits as `map` workers in the cluster exist, in order that they are all busy



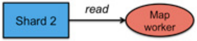
Example: Distributed Frequency Analysis with MapReduce (2/9)

- As next step, the user program creates via `fork()` copies of itself and creates this way the master and the workers





- The master assigns the `m` map tasks to the workers
- Each `map` worker reads a split part of the input data and extracts the key-value pairs



Sources: *Jeffrey Dean, Sanjay Ghemawat*. MapReduce: Simplied Data Processing on Large Clusters. 2004 and `http://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html`

Example: Distributed Frequency Analysis with MapReduce (3/9)

Map function

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

- `map` gets a document name `key` and a document `value` provided as strings
  - `map` scans the document word by word

- The `map`-process inserts for each word `w` a `1` into the intermediate result list of the word
  - At the end of the `map` phase, for a text with `n` different words, `n` intermediate result lists exist
    - Each intermediate result list contains as many `1` entries, as the corresponding word exists in the document

Source: *Jeffrey Dean, Sanjay Ghemawat*. MapReduce: Simplied Data Processing on Large Clusters. 2004

Example: Distributed Frequency Analysis with MapReduce (4/9)

```
Text = "Fest gemauert in der Erden
        steht die Form, aus Lehm gebrannt.
        Heute muß die Glocke werden,
        frisch, Gesellen, seid zur Hand.
        Von der Stirne heiß
        rinnen muß der Schweiß,
        soll das Werk den Meister loben,
        doch der Segen kommt von oben."
```

- The text is split into sentences
- It is also useful to convert all uppercase characters to lowercase characters and to remove the punctuation symbols

```
Input_list = [ (sentence_1, "fest gemauert in der erden steht die form aus lehm gebrannt"),
               (sentence_2, "heute muß die glocke werden frisch gesellen seid zur hand"),
               (sentence_3, "von der stirne heiß rinnen muß der schweiß soll das werk den meister loben
                             doch der segen kommt von oben") ]
```

- The input list contains three key-value pairs
  - Therefore 3 map processes can be started

```
Process_1 = map(sentence_1, "fest gemauert in der erden steht die form aus lehm gebrannt")
Process_2 = map(sentence_2, "heute muß die glocke werden frisch gesellen seid zur hand")
Process_2 = map(sentence_3, "von der stirne heiß rinnen muß der schweiß soll das werk den meister loben
                             doch der segen kommt von oben") ]
```

Source of this example: http://de.m.wikipedia.org/wiki/MapReduce

Example: Distributed Frequency Analysis with MapReduce (5/9)

- The `map` processes generate lists with intermediate result pairs:

```
P1 = [ ("fest", 1), ("gemauert", 1), ("in", 1), ("der", 1), ("erden", 1), ("steht", 1), ("die", 1),
       ("form", 1), ("aus", 1), ("lehm, 1), ("gebrannt", 1) ]
P2 = [ ("heute", 1), ("muß", 1), ("die", 1), ("glocke", 1), ("werden", 1), ("frisch", 1), ("gesellen", 1),
       ("seid", 1), ("zur", 1), ("hand", 1) ]
P3 = [ ("von", 1), ("der", 1), ("stirne", 1), ("heiß", 1), ("rinnen", 1), ("muß", 1), ("der", 1),
       ("schweiß", 1), ("soll", 1), ("das", 1), ("werk", 1), ("den", 1), ("meister", 1), ("loben", 1),
       ("doch", 1), ("der", 1), ("segen", 1), ("kommt", 1), ("von", 1), ("oben", 1) ]
```

- Each `map` worker sorts its local list of the intermediate result pairs
  - This is carried out by the MapReduce framework automatically
- Next, each `map` worker groups inside its local list of intermediate result pairs those key-value pairs, which have the same key
  - This is carried out by the MapReduce framework automatically too

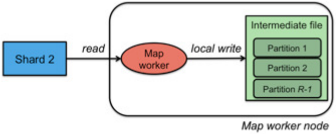
Sources: https://www.inkling.com/read/hadoop-definitive-guide-tom-white-3rd/chapter-6/shuffle-and-sort and http://de.m.wikipedia.org/wiki/MapReduce

Example: Distributed Frequency Analysis with MapReduce (6/9)

- The following output shows the result of the `map` phase

```
P1 = [ ("aus", 1), ("in", 1), ("der", 1), ("die", 1), ("erden", 1), ("fest", 1), ("form", 1),
       ("gebrannt", 1), ("gemauert", 1), ("lehm, 1), ("steht", 1) ]
P2 = [ ("die", 1), ("frisch", 1), ("gesellen", 1), ("glocke", 1), ("hand", 1), ("heute", 1), ("muß", 1),
       ("seid", 1), ("werden", 1), ("zur", 1) ]
P3 = [ ("das", 1), ("den", 1), ("der", (1, 1, 1)), ("doch", 1), ("heiß", 1), ("kommt", 1), ("loben", 1),
       ("meister", 1), ("muß", 1), ("oben", 1), ("rinnen", 1), ("schweiß", 1), ("segen", 1), ("soll", 1),
       ("stirne", 1), ("von", (1, 1)), ("werk", 1) ]
```

- The result of this phase is the intermediate result of MapReduce
  - Each `map` process stores its intermediate result to a local file
  - Each key-value pair in the intermediate result is called *partition*

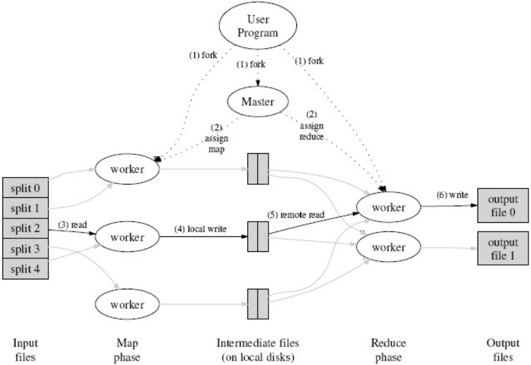


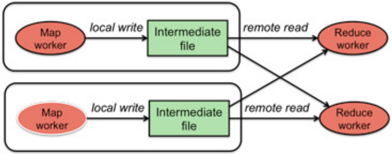
- Each `map` worker informs the master about the file name with the intermediate result and the partitions
- If all `map` processes have finished execution, the `shuffle` phase starts

Source of this example: `http://de.m.wikipedia.org/wiki/MapReduce`

Example: Distributed Frequency Analysis with MapReduce (7/9)

- The master allocates partitions and file names with intermediate results on map workers to the individual reduce workers





- Each reduce worker accesses via Remote Procedure Calls the files to receive the intermediate results
  - Next, it sorts the key-value pairs according to their keys

Sources: *Jeffrey Dean, Sanjay Ghemawat*. MapReduce: Simplied Data Processing on Large Clusters. 2004 and `http://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html`

Example: Distributed Frequency Analysis with MapReduce (8/9)

Reduce function

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
for 'word'
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

- reduce is executed for each word `key` and for the intermediate result list `values`
- reduce adds all numbers

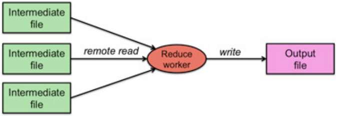
- Using the example of key `"der"`:

Input:

```
P1 = [ ("der", 1) ]
P3 = [ ("der", (1, 1, 1)) ]
```

Output:

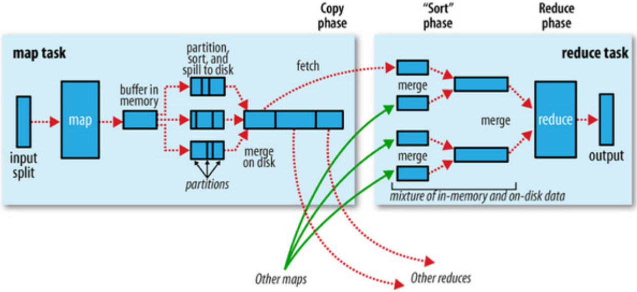
```
Output = [ ("der", 4) ]
```



Sources: *Jeffrey Dean, Sanjay Ghemawat*. MapReduce: Simplied Data Processing on Large Clusters. 2004 and `http://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html`

Example: Distributed Frequency Analysis with MapReduce (9/9)

- Finally, the master passes control back to the user program



Output of the user program

```
aus, 1
das, 1
den, 1
der, 4
die, 2
...
```

- The result of MapReduce is stored in `r` output files, which have been generated by `r` `reduce` workers
- The user program may merge the output files of the `reduce` workers to the final result

Sources: *Jeffrey Dean, Sanjay Ghemawat*. MapReduce: Simplied Data Processing on Large Clusters. 2004 and `http://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html`

Examples, where MapReduce is helpful

- **Distributed frequency analysis**
  How many times do words exist in a long text?
  - **Map-function**: Writes `<Word, 1>` into an intermediate memory
  - **Reduce-function**: Sums the values of a word to `<Word, Sum>`
- **Distributed grep**
  Which lines of text contain a search pattern?
  - **Map-function**: Writes rows detected into an intermediate memory
  - **Reduce-function**: Forwards the intermediate results for output through
- **Calculation of website requests (web access log)**
  - **Map-function**: Scans the web server log data and writes key-value pairs `<URL, 1>` into an intermediate memory
  - **Reduce-function**: Sums the values for an URL to `<URL, Sum>`
- **PageRank algorithm**

Google PageRank

- The PageRank algorithm rates linked documents (web pages)
- Developed and patented by Larry **Page** and Sergei Brin
- Basis of the Google search engine for the ranking of web pages
- Principle: The numerical weight (**PageRank**) PR_p of a web page p depends of the number and the numerical weight of the web pages, which refer to p

PageRank Algorithm

Source: Lars Kolb (Universität Leipzig) and Wikipedia

- PR_p = PageRank of a web page p
- $L_{IN}(p)$ = Set of documents, which refer to p
  $\Longrightarrow$ incoming links
- $L_{OUT}(p)$ = Set of documents, to which p refers
  $\Longrightarrow$ outgoing links

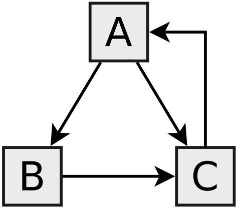
$$PR(p) = (1 - d) + d * \sum_{p_i \in L_{IN}(p)} \frac{PR(p_i)}{\text{amount } L_{OUT}(p_i)}$$

- d = damping factor between 0 and 1 (usually 0.85)
  - A small portion of the weight $(1 - d)$ is withdrawn from any web page and distributed equally among all detected web pages
    - This prevents, that the weight *flows away* to websites, which do not contain links to other websites

Source: `http://dbs.uni-leipzig.de/file/CDM_WS_2013_14_03_MapReduce.pdf`

PageRank Example

Source: Lars Kolb (Universität Leipzig)



$$PR(p) = (1-d) + d * \sum_{p_i \in L_{IN}(p)} \frac{PR(p_i)}{\text{amount } L_{OUT}(p_i)}$$

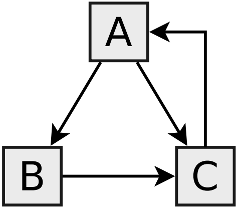
- $PR(A) = (1-d) + d * PR(C)$
- $PR(B) = (1-d) + d * \frac{PR(A)}{2}$
- $PR(C) = (1-d) + d * (\frac{PR(A)}{2} + PR(B))$

- Conversion to iteration equations with $d = 0.5$:
- $PR_{n+1}(A) = 0.5 + 0.5 * PR_n(C)$
- $PR_{n+1}(B) = 0.5 + 0.5 * \frac{PR_n(A)}{2}$
- $PR_{n+1}(C) = 0.5 + 0.5 * (\frac{PR_n(A)}{2} + PR_n(B))$
- Result of the iteration with $PR_0(A) = PR_0(B) = PR_0(C) = 1$

	0	1	2	3	4	5	6	PR
A	1							
B	1							
C	1							

PageRank Example (Result)

Source: Lars Kolb (Universität Leipzig)



- For examples with just a few documents, $<$ 10 iterations are required to compute the PageRank of the documents
- For calculating the PageRank for the entire WWW, around 100 iterations are required

- Conversion to iteration equations with $d = 0.5$:
- $PR_{n+1}(A) = 0.5 + 0.5 * PR_n(C)$
- $PR_{n+1}(B) = 0.5 + 0.5 * \frac{PR_n(A)}{2}$
- $PR_{n+1}(C) = 0.5 + 0.5 * (\frac{PR_n(A)}{2} + PR_n(B))$
- Result of the iteration with $PR_0(A) = PR_0(B) = PR_0(C) = 1$

	0	1	2	3	4	5	6	PR
A	1	1	1.125	1.0625	1.078125	1.078125	1.076171875	1.077
B	1	0.75	0.75	0.78125	0.765625	0.76953125	0.76953125	0.769
C	1	1.25	1.125	1.15625	1.15625	1.15234375	1.154296875	1.154

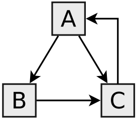
PageRank and MapReduce

Source: Lars Kolb (Universität Leipzig)

- PageRank can be parallelized because:
  - Iteration $n+1$ depends only of the values of iteration n
  - The calculations of the PR values of different documents are independent of each other
- Map phase:
  - Input: Document X, PR_n, **outgoing** links $L_{OUT}(X)$
  - Calculate for each link $X \longrightarrow Y$ the *sum component*, which X contributes for Y
    $\Longrightarrow$ Output: $Y, \frac{PR_n(X)}{\text{amount } L_{OUT}(X)}$
  - Additional output: List of outgoing links: $X, L_{OUT}(X)$
- Reduce phase:
  - Input: Document X, sum components of the **incoming** links and the list of outgoing links $X, L_{OUT}(X)$
  - Calculate PR_{n+1}
  - Output: $X, PR_{n+1}, L_{OUT}(X)$
- A predetermined number of iterations is carried out

PageRank and MapReduce – Example

Source: Lars Kolb (Universität Leipzig)



X, PR_n, list of outgoing links in X

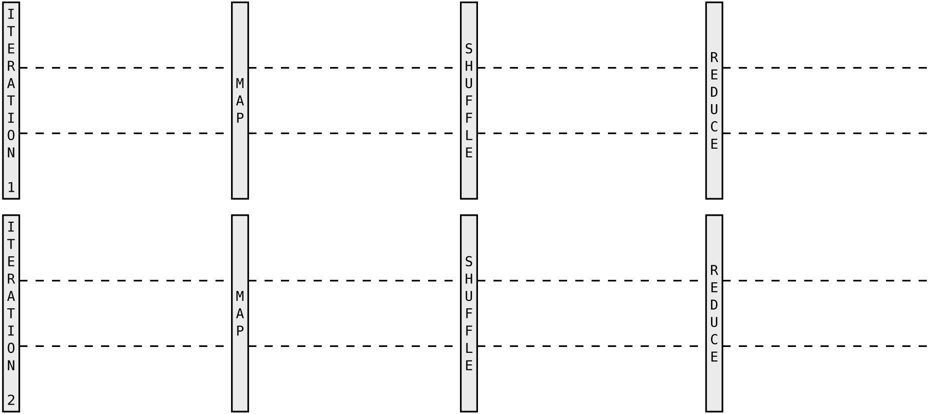
For each outgoing link X --> Y:
Y, PR(X)/amount of outgoing links in X

Additionally:
X, list of outgoing links in X

For each incoming link to X:
X, sum component of the incoming link

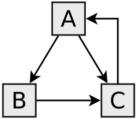
Additionally:
X, list of outgoing links in X

X, PR_n+1, list of outgoing links in X



PageRank and MapReduce – Result

Source: Lars Kolb (Universität Leipzig)



	X, PR_n, list of outgoing links in X		For each outgoing link X --> Y: Y, PR(X)/amount of outgoing links in X Additionally: X, list of outgoing links in X		For each incoming link to X: X, sum component of the incoming link Additionally: X, list of outgoing links in X		X, PR_n+1, list of outgoing links in X
ITERATION 1	A,1,(B,C)	MAP	B,0.5 C,0.5 A,B,C	SHUFFLE	A,1 A,B,C	REDUCE	A,1,(B,C)
	B,1,C		C,1 B,C		B,0.5 B,C		B,0.75,C
	C,1,A		A,1 C,A		C,0.5 C,1 C,A		C,1.25,A
ITERATION 2	A,1,(B,C)	MAP	B,0.5 C,0.5 A,B,C	SHUFFLE	A,1.25 A,B,C	REDUCE	A,1.125,(B,C)
	B,0.75,C		C,0.75 B,C		B,0.5 B,C		B,0.75,C
	C,1.25,A		A,1.25 C,A		C,0.5 C,0.75 C,A		C,1.125,A

Hadoop – Components and Extensions

- **Hadoop Distributed File System** (HDFS)
- **Pig**: Database language of Yahoo
- **Hive**: Data Warehouse of Facebook
- **HBase**: Database for managing very large amounts of data
- This is just a selection of popular components/extensions
  - Further extensions, such as Chukwa and ZooKeeper exist
    - Chukwa is used for real-time monitoring of very large distributed systems
    - ZooKeeper simplifies the configuration of distributed systems

Good introduction to Pig and Hive in German language

Ralf Falk, David Knaak, Michael Köster, Marko Salchow
`http://wiki.fh-stralsund.de/index.php/Vergleich_Hive_vs._Pig`

Hadoop Distributed File System (HDFS)

- Hadoop contains that Hadoop Distributed File System (HDFS)
  - Open-Source re-implementation of the Google File System (GFS)
  - Fault-tolerant, distributed file system

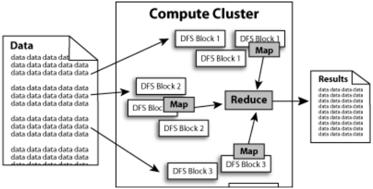

Compute Cluster
Data
data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data data
DFS Block 1
DFS Block 1
Map
DFS Block 1
DFS Block 2
DFS Bloc
Map
Reduce
DFS Block 2
Map
DFS Block 3
DFS Block 3
Results
data data data data

Image source:
`http://hadoop.apache.org`

Further details:
`http://hadoop.apache.org/hdfs/`

- The Google Clusters consist of low-cost commodity hardware
  - Failure of individual nodes is not an exception, but rather the usual case
    $\Longrightarrow$ Fault tolerance is an important goal of GFS and HDFS
  - New nodes can be added easily
  - Amounts of data in the petabyte range need to be managed

Helpful Source: Ghemawat, Gobioff, Leung. The Google file system (2003)

Architecture of HDFS (follows the Master/Slave Principle)

- **Namenode** (Master) stores the metadata, and reacts to failures
  - Exists exactly one time in a HDFS Cluster
    $\Longrightarrow$ Single Point of Failure
  - Does not store any user data, but only metadata
    - Knows all files and directories, which exist in the HDFS Cluster
    - Stores the numbers of chunks of the files, the number of copies and their locations (Datanodes)
    - Monitors the availability of the individual Datanodes, and redistributes the chunks, in case a Datanode fails
- **Datanodes** (Slaves) only store the user data
  - Special feature of HDFS and GFS: Chunk size is 64 MB
    - The resulting internal fragmentation is accepted for the benefit of a lower network overhead
- **Clients** communicate with the Namenode and the Datanodes
  - The Namenode only operates as a directory service
    - Provides the information, on which Datanodes, a searched file is stored
  - Read and write requests send Clients directly to the Datanodes

Security against Data Loss at HDFS

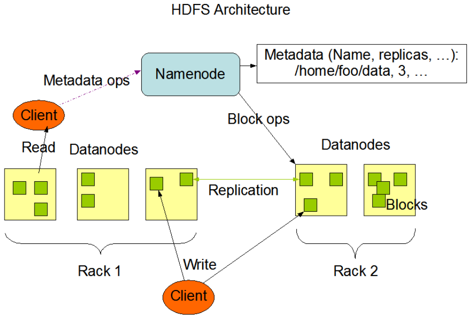


Image source: `http://hadoop.apache.org/common/docs/current/hdfs_design.html`

- User data is replicated three times on different nodes to ensure data consistency in case of frequent failures of nodes

- Datanodes inform the Namenode regularly via heartbeat about their existence
  - If a Namenode does not receive any more messages of a Datanode, the Namenode declares the Datanode as failed
  - Next, the Namenode orders the replication of the affected Chunks, in order not to fall below the minimum number of replications

Read Accesses at HDFS

- The HDFS client calls a web service (for read access) on the Namenode with the desired filename as a parameter
- The Namenode checks the namespace, to find out which Datanodes store the chunks
  - The namespace resides inside the main memory of the Namenode
- The Namenode provides the Client:
  - Unique 64-bit identifiers (*chunk handles*) of the chunks on the Datanodes
  - a list of Datanodes, which store the chunks
- The HDFS client calls the web service of one or more Datanodes, to obtain the user data
- By using the 64-bit identifiers, the Datanodes read the HDFS chunks on their HDD and transfer them as the result to the client

Write Accesses at HDFS (1/2)

- The HDFS client calls a web service (for write access) on the Namenode with the desired filename as a parameter
- The Namenode checks, if the client has write permissions and if the file already exists
  - If the verifications for the client are positive, the Namenode stores the meta-information of the file in the namespace
  - If the file already exists, or if the client does not have write permissions, the Namenode interrupts the process with an exception
- It is impossible to overwrite files in HDFS
  - Overwriting files is only possible by deleting and re-creating them
- The client splits the file to be stored into chunks and places them in a local a queue
- For each chunk in the queue, the client calls the web service interface of the Namenode, which returns a list of Datanodes, to store the chunk
  - Additionally, the client receives an identifier for the chunk

Write Accesses at HDFS (2/2)

- The choice of the Datanodes to store a chunk depends on the configuration
  - Several configuration options exist, e.g. the definition of racks to join physically neighboring servers to a virtual rack ($\Longrightarrow$ *rack-awareness*)
  - Objective: Reduce network traffic
- The HDFS client transmits the chunk and the list of Datanodes to a single Datanode, which stores them locally with the identifier
- After the successful transmission of the chunk, the Datanode forwards the chunk to another Datanode in the list, to get the chunk stored there too
  - This process is repeated with another Datanode from the list, until the specified number of chunk replications is reached

File system alternatives in Hadoop

It is not absolutely necessary to use HDFS. Alternatives are among others S3 and FTP

Secondary Namenode at HDFS

- In order to ensure data integrity and a fast restart of the Namenode after a failure, the secondary Namenode exists
  - It can not replace the Namenode in case of failure
- The Secondary Namenode never communicates with the clients
- The Namenode stores the metadata in form of an image (= namespace), and a list of transactions which need to be applied to the image
  - In case the Namenode fails, it needs to virtually carry out all the transactions on the image to obtain the latest state
    - That takes a long time for large file systems
- The Secondary Namenode stores the image (namespace) as backup in intervals
- If the the Namenode fails, during reboot, it can fetch the latest image checkpoint from the Secondary Namenode

Architecture of the Google File System (GFS)

Image Source: Google

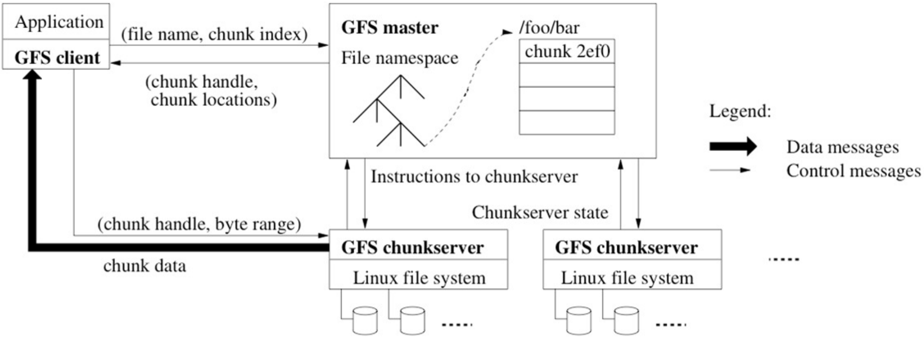


Figure 1: GFS Architecture

Pig (1/2)

- Can be used for the analysis of very large amounts of semi-structured, structured or relational data
  - Includes a programming language and a compiler for queries on data
- The programming language is Pig Latin
  - Pig Latin is called a *Dataflow Language*
  - It is used to specify sequences of individual transformations on data
  - Thus, doing ad-hoc analyzing of large amounts of data is possible
- The compiler translates Pig Latin statements into MapReduce jobs
  - Pig also orchestrates the execution of the jobs in the Hadoop Cluster
- Pig is used with the Pig shell (Grunt)
  - Grunt can also load scripts to execute commands in batch mode

Pig (2/2)

- Pig reads all data formats, regardless of their structure
  - By default, Pig expects the data as plain text and tab separated
  - For the interpretation of different formatted data, the users can specify User Defined Functions (UDF)
    - With UDF, users can integrate own code into Pig programs
    - Apache offers with Piggybank an open repository `http://svn.apache.org/repos/asf/pig/trunk/contrib/piggyban`
    - UDFs are written in Java and integrated as a JAR file into Pig
- Advantage of Pig: Reduced complexity compared to MapReduce queries

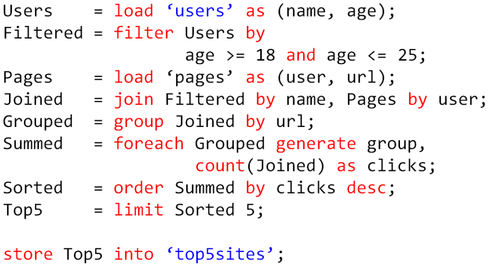
Pig Commands

Command	Meaning
`load`	Read data from the file system
`store`	Write data into the file system
`foreach`	Apply an expression to all records
`filter`	Discard all records, which do not match the filter rules
`group/cogroup`	Collect records with the same key from one or more input sources
`join`	Combine two or more input sources according to a key
`order`	Sort records according to a key
`distinct`	Erase duplicate records
`union`	Merge two records
`split`	Split data into two or more records, using filter rules
`stream`	Transfer all records to a specified binary file
`dump`	Write the output to `stdout`
`limit`	Limit the number of records

Source: Introduction to Pig. Cloudera (2009)
`http://www.cloudera.com/videos/introduction_to_pig`

Example of a Job in Pig Latin

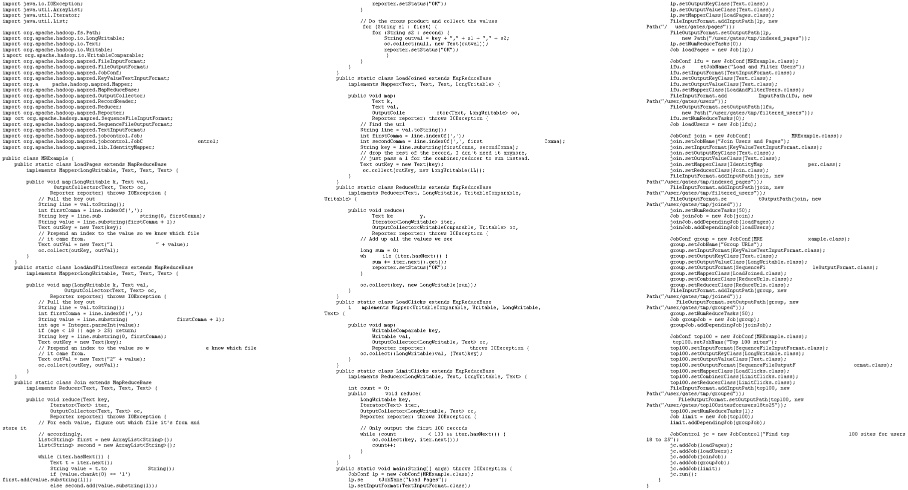
- This example shows the complexity reduction of MapReduce queries compared to Pig Latin queries
    - Query for the 5 most frequently visited web pages from people, which are 18-25 years old
    - The user information and data of the web pages are located in 2 different files



```
Users    = load ‘users’ as (name, age);
Filtered = filter Users by
                  age >= 18 and age <= 25;
Pages    = load ‘pages’ as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                   count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into ‘top5sites’;
```

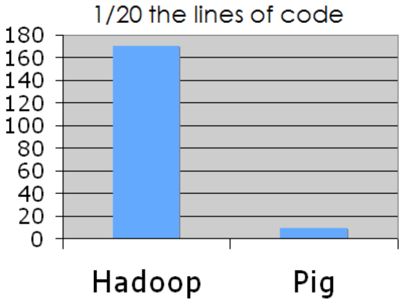
Source of the example and the images: ApacheCon Europe 2009

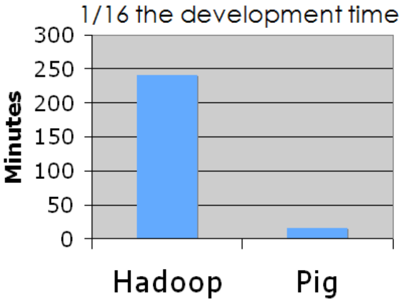
For Comparison, the Statements in MapReduce Format



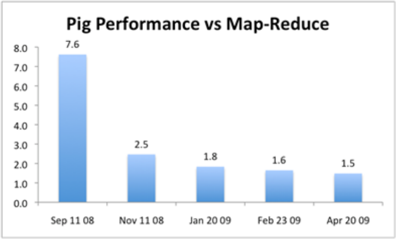
```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
i mport org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.a     pache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
imp ort org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC                  ontrol;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                 OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.sub             string(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1              " + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                    OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(                firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so w                   e know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to             String();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
             for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                     }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputColle          ctor<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first                    Comma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
             oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text ke        y,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            wh     ile (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        i   mplements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter)              throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public      void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count          < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.se     tJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/  user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.s     etJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add          InputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(             MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMap              per.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.se          tOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRE              xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi              leOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
          FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
         top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputF                   ormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
          FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top                   100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Savings because of using Pig

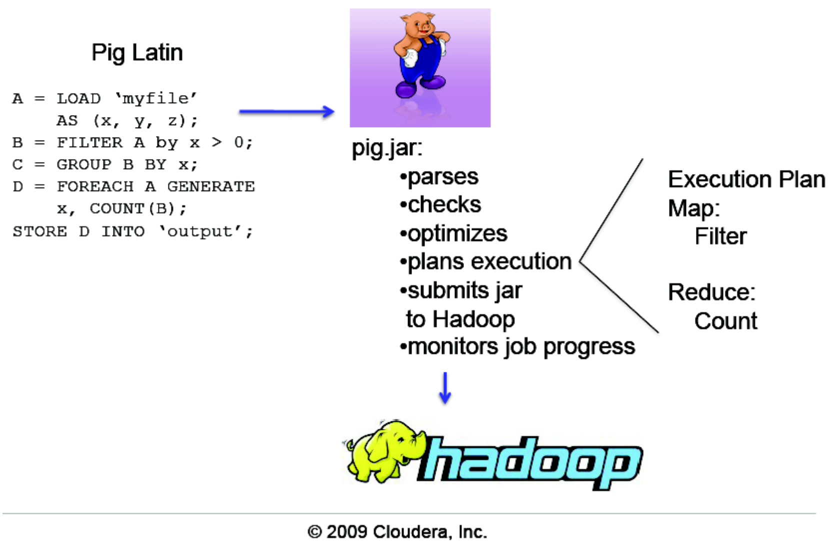




Performance:
1.5x Hadoop



Pig – Helpful Summary of Cloudera



Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```

pig.jar:

- •parses
- •checks
- •optimizes
- •plans execution
- •submits jar to Hadoop
- •monitors job progress

Execution Plan
Map:
Filter

Reduce:
Count

Hive

- Data warehouse system on the basis of Hadoop
- A data warehouse. . .
  - is a data collection site in form of a database
  - obtains data from different sources (e.g. other databases)
- Data model is analogous to relational database systems with tables
  - Payload stores Hive in HDFS
  - Tables are represented by folder in HDFS
  - The data inside the tables are stored serialized in files inside the folders
  - Metadata is stored in the relational database Metastore
- Supports different column types (e.g. integer, string, date, boolean)
- For Queries, the declarative language HiveQL is used
  - Query language which provides a SQL-like syntax
  - Hive translates HiveQL statements into MapReduce jobs
    - Hive also orchestrates the execution of the jobs in the Hadoop Clusters
- Controlled via a command line interface, web interface or JDBC/ODBC interface

Load Text Data into Hive Tables and analyze them

- For each access to a web server, these information is recorded:
  - Hostname or IP address of the accessing client
  - Date and time
  - Time zone
  - File
  - Result of the access (HTTP status message)
  - Bytes transferred

```
client.anbieter.de - - [08/Oct/2010:22:35:51 -0100] "GET /pfad/index.html HTTP/1.1" 200 1832
```

- Import log data from `access.log` into a table:

```
LOAD DATA LOCAL INPATH 'access.log' OVERWRITE INTO TABLE apachelog;
```

- Print the first 20 rows of the tables, sorted according the IP addresses:

```
SELECT * FROM apachelog SORT BY ipaddress LIMIT 20;
```

- Print all records, which contain the IP address 84.171.184.103:

```
SELECT * FROM apachelog WHERE ipaddress = '84.171.184.103';
```

Source: Ramin Wartala. Analyse großer Datenmengen mit Hive. iX 12/2010

Hive Examples

(Source: http://wiki.apache.org/hadoop/Hive/LanguageManual/DDL)

- Create table page_view:

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
     page_url STRING, referrer_url STRING,
     ip STRING COMMENT ’IP Address of the User’)
 COMMENT ’This is the page view table’
 PARTITIONED BY(dt STRING, country STRING)
 ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ’\001’
   LINES TERMINATED BY ’\012’
STORED AS SEQUENCEFILE;
```

- Erase table:

```
DROP TABLE [IF EXISTS] table_name
```

- Print table name:

```
SHOW TABLES identifier_with_wildcards
```

- Print partitions of a table:

```
SHOW PARTITIONS table_name
```

- Rename table:

```
ALTER TABLE table_name RENAME TO new_table_name
```

- Add or replace columns:

```
ALTER TABLE table_name ADD|REPLACE COLUMNS (col_name data_type [COMMENT col_comment], ...)
```

Distinction between Pig and Hive

Criterion	Pig	Hive
Typical application scenarios	logfile analysis	logfile analysis, data mining, web analytics in real time, data warehousing
Objectives	simplification of MapReduce queries with a scripting language	simplification of MapReduce with a SQL style
Query language	Pig Latin (procedural)	HiveQL9 (declarative)
Metadata	none	stored in Metastore
User interfaces	command line interface (Grunt)	command line interface, web interface
Export interfaces	none	ODBC/JDBC
Input data structure	unstructured	structured
Input data formats	raw data	raw data
Output data formats	raw data	raw data
Main developer	Yahoo	Facebook

Source: `http://wiki.fh-stralsund.de/index.php/Vergleich_Hive_vs._Pig`

HBase

http://hbase.apache.org

- Column-oriented database to manage very large amounts of data in Hadoop Clusters
  - Suited for large amounts of data, which are rarely changed, but often added with additional data
  - Suited for billions of rows and millions of columns, distributed over many servers from commodity hardware
- Free re-implementation of Google BigTable
  - Googles BigTable runs on top of the GFS
  - HBase runs on top of HDFS (free re-implementation of the GFS)

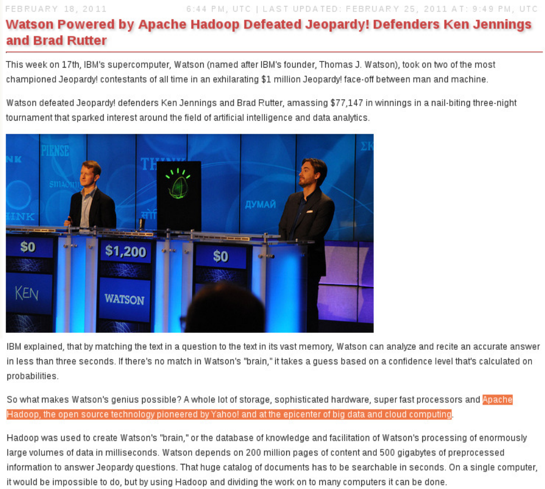
Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, Gruber. Bigtable: A Distributed Storage System for Structured Data. Google (2006)
http://labs.google.com/papers/bigtable-osdi06.pdf

Hadoop Success Stories

(http://wiki.apache.org/hadoop/PoweredBy)

- **EBay**
  - Clusters with 532 nodes (4,256 CPU cores, 5.3 PB memory)
  - Task: Optimization of the search functionality
- **Facebook**
  - Clusters with 1,100 nodes (8,800 CPU cores, 12 PB memory)
  - Clusters with 300 nodes (2,400 CPU cores, 3 PB memory)
  - Task: Log data storage and analysis
- **Last.fm**
  - Clusters with 44 nodes (352 CPU cores, 176 PB memory)
  - Task: Log data storage and analysis, calculation of charts
- **Twitter**
  - Task: Log data storage and analysis, storing the Tweets
- **Yahoo**
  - Multiple Clusters, together with $>$ 40,000 nodes and $>$ 100,000 CPUs
  - Largest Cluster: 4,500 nodes (each with 8 CPUs and 4 TB Storage)
  - Task: Web search and advertising
  - Further information: `http://developer.yahoo.com/blogs/hadoop/`

Hadoop and IBM Watson vs. Mankind



FEBRUARY 18, 2011 6:44 PM, UTC | LAST UPDATED: FEBRUARY 25, 2011 AT: 9:49 PM, UTC

Watson Powered by Apache Hadoop Defeated Jeopardy! Defenders Ken Jennings and Brad Rutter

This week on 17th, IBM's supercomputer, Watson (named after IBM's founder, Thomas J. Watson), took on two of the most championed Jeopardy! contestants of all time in an exhilarating $1 million Jeopardy! face-off between man and machine.

Watson defeated Jeopardy! defenders Ken Jennings and Brad Rutter, amassing $77,147 in winnings in a nail-biting three-night tournament that sparked interest around the field of artificial intelligence and data analytics.

IBM explained, that by matching the text in a question to the text in its vast memory, Watson can analyze and recite an accurate answer in less than three seconds. If there's no match in Watson's "brain," it takes a guess based on a confidence level that's calculated on probabilities.

So what makes Watson's genius possible? A whole lot of storage, sophisticated hardware, super fast processors and Apache Hadoop, the open source technology pioneered by Yahoo! and at the epicenter of big data and cloud computing.

Hadoop was used to create Watson's "brain," or the database of knowledge and facilitation of Watson's processing of enormously large volumes of data in milliseconds. Watson depends on 200 million pages of content and 500 gigabytes of preprocessed information to answer Jeopardy questions. That huge catalog of documents has to be searchable in seconds. On a single computer, it would be impossible to do, but by using Hadoop and dividing the work on to many computers it can be done.

http://www.ditii.com/2011/02/18/watson-powered-by-apache-hadoop-defeated-jeopardy-defenders-ken-jennings-and-brad-rutter/

Cloudera

http://www.cloudera.com

- Cloudera is a powerful Hadoop distribution
  - Contains Hadoop and among others the programming language Pig, the SQL database Hive, the columns-oriented database HBase, Apache Zookeeper and Hadoop browser frontend Hue
  - Packages for Debian, Ubuntu, RedHat and SuSE are available

- 03/2009: Cloudera's Distribution for Hadoop (CDH1)
- 03/2009: $5 Millions venture capital from Accel Partners
- 06/2009: $11 Millions venture capital from Greylock Partners
- 08/2009: Doug Cutting leaves Yahoo and becomes an employee of Cloudera
- 10/2010: $25 Millions venture capital from Meritech Capital Partners
- 03/2013: Intel invests $740 Millions for an 18% investment

Current version of Cloudera's Distribution for Hadoop (State: January 2018)

https://www.cloudera.com/downloads/cdh/5-13-1.html

Install Cloudera (CDH3) in Ubunutu 10.10 (1/2)

- These instructions install a Cluster on a single node
  - **Pseudo Distributed Mode**
- These instructions base of `http://cloudera-tutorial.blogspot.com`
- Start an instance with Ubuntu 10.10 (`ami-08f40561`) in US-East
- Allow access via the ports 22, 50030 und 50070 in the security group
  - DNS: `ec2-50-17-58-144.compute-1.amazonaws.com`
- Insert the package sources in `/etc/apt/sources.list.d/cloudera.list`

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ubuntu maverick partner"
$ sudo add-apt-repository "deb http://archive.cloudera.com/debian maverick-cdh3 contrib"
```

- Import the key of the Cloudera repository

```
$ sudo curl -s http://archive.cloudera.com/debian/archive.key | sudo apt-key add -
```

- Install packages

```
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk
$ sudo apt-get install hadoop-0.20-conf-pseudo
```

- Start Cloudera services

```
$ for service in /etc/init.d/hadoop-0.20-*; do sudo $service start; done
```

Install Cloudera (CDH3) in Ubunutu 10.10 (2/2)

- Check, which files Cloudera did install

```
$ dpkg -L hadoop-0.20-conf-pseudo
```

- Check, if the Cloudera services are running

```
$ sudo jps
2232 SecondaryNameNode
2539 Jps
1994 DataNode
2074 JobTracker
2154 NameNode
2317 TaskTracker
```

- If the list of services is complete, the installation was successful!
- Web interface of the Namenode

```
http://ec2-50-17-58-144.compute-1.amazonaws.com:50070
```

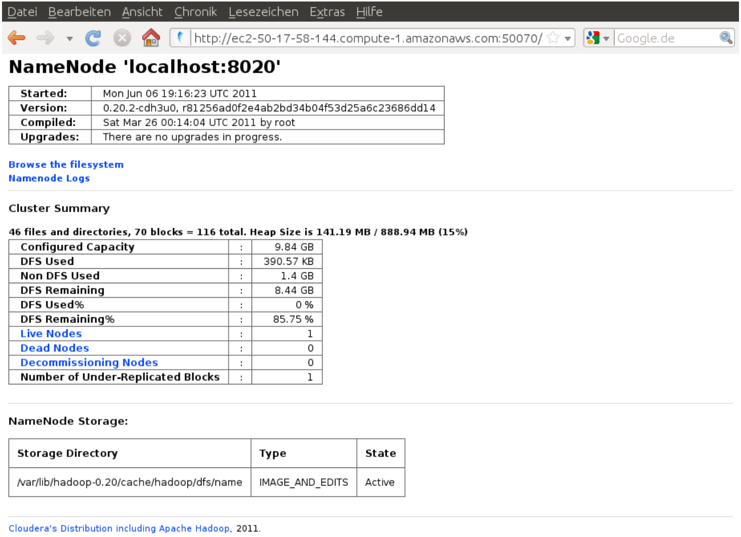
- Web interface of the Job Trackers

```
http://ec2-50-17-58-144.compute-1.amazonaws.com:50030
```

- Terminate the Cloudera services

```
$ for x in /etc/init.d/hadoop-* ; do sudo $x stop ; done
```

Web Interface of the Namenode



NameNode 'localhost:8020'

Started:	Mon Jun 06 19:16:23 UTC 2011
Version:	0.20.2-cdh3u0, r81256ad0f2e4ab2bd34b04f53d25a6c23686dd14
Compiled:	Sat Mar 26 00:14:04 UTC 2011 by root
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

46 files and directories, 70 blocks = 116 total. Heap Size is 141.19 MB / 888.94 MB (15%)

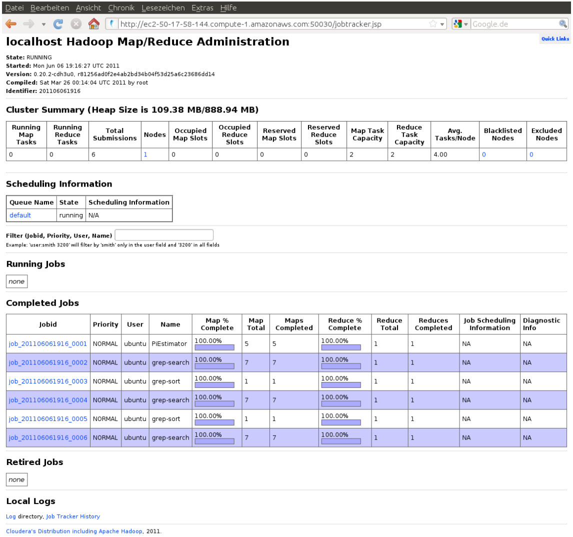
Configured Capacity	:	9.84 GB
DFS Used	:	390.57 KB
Non DFS Used	:	1.4 GB
DFS Remaining	:	8.44 GB
DFS Used%	:	0 %
DFS Remaining%	:	85.75 %
Live Nodes	:	1
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	1

NameNode Storage:

Storage Directory	**Type**	**State**
/var/lib/hadoop-0.20/cache/hadoop/dfs/name	IMAGE_AND_EDITS	Active

Cloudera's Distribution including Apache Hadoop, 2011.

Web Interface of the Job Tracker


Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://ec2-50-17-58-144.compute-1.amazonaws.com:50030/jobtracker.jsp
Google.de
Quick Links

localhost Hadoop Map/Reduce Administration

State: RUNNING
Started: Mon Jun 06 19:16:27 UTC 2011
Version: 0.20.2-cdh3u0, r81256ad0f2e4ab2bd34b04f53d25a6c23686dd14
Compiled: Sat Mar 26 00:14:04 UTC 2011 by root
Identifier: 201106061916

Cluster Summary (Heap Size is 109.38 MB/888.94 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Excluded Nodes
0	0	6	1	0	0	0	0	2	2	4.00	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201106061916_0001	NORMAL	ubuntu	PiEstimator	100.00%	5	5	100.00%	1	1	NA	NA
job_201106061916_0002	NORMAL	ubuntu	grep-search	100.00%	7	7	100.00%	1	1	NA	NA
job_201106061916_0003	NORMAL	ubuntu	grep-sort	100.00%	1	1	100.00%	1	1	NA	NA
job_201106061916_0004	NORMAL	ubuntu	grep-search	100.00%	7	7	100.00%	1	1	NA	NA
job_201106061916_0005	NORMAL	ubuntu	grep-sort	100.00%	1	1	100.00%	1	1	NA	NA
job_201106061916_0006	NORMAL	ubuntu	grep-search	100.00%	7	7	100.00%	1	1	NA	NA

Retired Jobs

none

Local Logs

Log directory, Job Tracker History

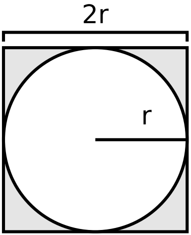
Cloudera's Distribution including Apache Hadoop, 2011.

Simple Examples with the Cloudera Installation (1/5)

- Example for the calculation of π

```
$ hadoop jar /usr/lib/hadoop/hadoop-*-examples.jar pi 10 100
Number of Maps  = 5
Samples per Map = 1000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Starting Job
11/06/06 19:18:15 INFO mapred.FileInputFormat: Total input paths to process : 5
11/06/06 19:18:16 INFO mapred.JobClient: Running job: job_201106061916_0001
11/06/06 19:18:17 INFO mapred.JobClient:  map 0% reduce 0%
11/06/06 19:18:23 INFO mapred.JobClient:  map 40% reduce 0%
11/06/06 19:18:27 INFO mapred.JobClient:  map 60% reduce 0%
11/06/06 19:18:28 INFO mapred.JobClient:  map 80% reduce 0%
11/06/06 19:18:29 INFO mapred.JobClient:  map 100% reduce 0%
11/06/06 19:18:36 INFO mapred.JobClient:  map 100% reduce 100%
11/06/06 19:18:36 INFO mapred.JobClient: Job complete: job_201106061916_0001
...
11/06/06 19:18:36 INFO mapred.JobClient:     Launched reduce tasks=1
...
11/06/06 19:18:36 INFO mapred.JobClient:     Launched map tasks=5
...
Job Finished in 20.638 seconds
Estimated value of Pi is 3.14160000000000000000
```

Calculation of π via Monte Carlo Simulation



A	=	Surface ratio
r	=	Radius
C	=	Circle
S	=	Square

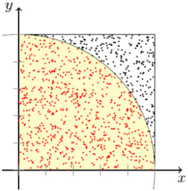


Image source: Wikipedia

- π can be approximated via Monte Carlo simulation
- Inscribe a circle of radius r inside a square with side length $2r$
- $A_S = (2r)^2 = 4r^2$
- $A_C = \pi r^2 \implies \pi = \frac{A_C}{r^2}$
- How can we approximate π?
  1. Generate random dots in the square
  2. The number of dots in A_C in relation to the number of dots in A_S is equal to the surface ratio

$$\frac{A_C}{A_S} = \frac{\pi r^2}{4r^2} \implies \frac{A_C}{A_S} = \frac{\pi}{4} \implies 4 * \frac{A_C}{A_S} = \pi$$

- The dots can be generated in parallel by the workers
- The master receives the dots and calculates π

Calculation of π via MapReduce

```
NUMPOINTS = 100000; // some large number - the bigger, the closer the approximation

p = number of WORKERS;
numPerWorker = NUMPOINTS / p;
countCircle = 0; // one of these for each WORKER

// each WORKER does the following:
for (i = 0; i < numPerWorker; i++) {
  generate 2 random numbers that lie inside the square;
  xcoord = first random number;
  ycoord = second random number;
  if (xcoord, ycoord) lies inside the circle
  countCircle++;
}

MASTER:
  receives from WORKERS their countCircle values
  computes PI from these values: PI = 4.0 * countCircle / NUMPOINTS;
```

Source: **Introduction to Parallel Programming and MapReduce**
http://code.google.com/edu/parallel/mapreduce-tutorial.html

Simple Examples with the Cloudera Installation (2/5)

- grep example

```
$ hadoop-0.20 fs -mkdir input
$ hadoop-0.20 fs -put /etc/hadoop-0.20/conf/*.xml input
$ hadoop-0.20 jar /usr/lib/hadoop-0.20/hadoop-*-examples.jar grep input output 'dfs[a-z.]+'
11/06/06 20:05:49 INFO mapred.FileInputFormat: Total input paths to process : 7
11/06/06 20:05:49 INFO mapred.JobClient: Running job: job_201106061916_0010
11/06/06 20:05:50 INFO mapred.JobClient:  map 0% reduce 0%
11/06/06 20:05:55 INFO mapred.JobClient:  map 28% reduce 0%
11/06/06 20:05:59 INFO mapred.JobClient:  map 42% reduce 0%
11/06/06 20:06:00 INFO mapred.JobClient:  map 57% reduce 0%
11/06/06 20:06:02 INFO mapred.JobClient:  map 71% reduce 0%
11/06/06 20:06:03 INFO mapred.JobClient:  map 85% reduce 0%
11/06/06 20:06:05 INFO mapred.JobClient:  map 100% reduce 0%
11/06/06 20:06:10 INFO mapred.JobClient:  map 100% reduce 28%
11/06/06 20:06:11 INFO mapred.JobClient:  map 100% reduce 100%
11/06/06 20:06:12 INFO mapred.JobClient: Job complete: job_201106061916_0010
...
11/06/06 20:06:12 INFO mapred.JobClient:     Launched reduce tasks=1
...
11/06/06 20:06:12 INFO mapred.JobClient:     Launched map tasks=7
...
```

Simple Examples with the Cloudera Installation (3/5)

- Output of the grep example

```
$ hadoop fs -ls output
Found 3 items
-rw-r--r--   1 ubuntu supergroup          0 2011-06-06 19:33 /user/ubuntu/output/_SUCCESS
drwxr-xr-x   - ubuntu supergroup          0 2011-06-06 19:32 /user/ubuntu/output/_logs
-rw-r--r--   1 ubuntu supergroup        129 2011-06-06 19:33 /user/ubuntu/output/part-00000
```

- Result of the grep example

```
$ hadoop-0.20 fs -cat output/part-00000
1       dfs.datanode.plugins
1       dfs.name.dir
1       dfs.namenode.plugins
1       dfs.permissions
1       dfs.replication
1       dfs.thrift.address
1       dfsadmin
```

- For control...

```
$ grep dfs[a-z.] /etc/hadoop-0.20/conf/*.xml
/etc/hadoop-0.20/conf/hadoop-policy.xml:    dfsadmin and mradmin commands to refresh the security...
/etc/hadoop-0.20/conf/hdfs-site.xml:    <name>dfs.replication</name>
/etc/hadoop-0.20/conf/hdfs-site.xml:     <name>dfs.permissions</name>
/etc/hadoop-0.20/conf/hdfs-site.xml:     <name>dfs.name.dir</name>
/etc/hadoop-0.20/conf/hdfs-site.xml:    <name>dfs.namenode.plugins</name>
/etc/hadoop-0.20/conf/hdfs-site.xml:    <name>dfs.datanode.plugins</name>
/etc/hadoop-0.20/conf/hdfs-site.xml:    <name>dfs.thrift.address</name>
```

Simple Examples with the Cloudera Installation (4/5)

- Word count example

```
$ hadoop-0.20 fs -mkdir inputwords
$ hadoop-0.20 fs -put /etc/hadoop-0.20/conf/*.xml inputwords
$ hadoop-0.20 jar /usr/lib/hadoop-0.20/hadoop-*-examples.jar wordcount inputwords outputwords
11/06/06 20:46:59 INFO input.FileInputFormat: Total input paths to process : 7
11/06/06 20:46:59 INFO mapred.JobClient: Running job: job_201106061916_0014
11/06/06 20:47:00 INFO mapred.JobClient:  map 0% reduce 0%
11/06/06 20:47:05 INFO mapred.JobClient:  map 28% reduce 0%
11/06/06 20:47:08 INFO mapred.JobClient:  map 42% reduce 0%
11/06/06 20:47:10 INFO mapred.JobClient:  map 57% reduce 0%
11/06/06 20:47:11 INFO mapred.JobClient:  map 71% reduce 0%
11/06/06 20:47:13 INFO mapred.JobClient:  map 85% reduce 0%
11/06/06 20:47:14 INFO mapred.JobClient:  map 100% reduce 0%
11/06/06 20:47:17 INFO mapred.JobClient:  map 100% reduce 100%
11/06/06 20:47:17 INFO mapred.JobClient: Job complete: job_201106061916_0014
...
11/06/06 20:18:20 INFO mapred.JobClient:     Launched reduce tasks=1
...
11/06/06 20:18:20 INFO mapred.JobClient:     Launched map tasks=7
...
```

Simple Examples with the Cloudera Installation (5/5)

- Output of the word count example

```
$ hadoop-0.20 fs -ls outputwords
Found 3 items
-rw-r--r--   1 ubuntu supergroup          0 2011-06-06 20:47 /user/ubuntu/outputwords/_SUCCESS
drwxr-xr-x   - ubuntu supergroup          0 2011-06-06 20:46 /user/ubuntu/outputwords/_logs
-rw-r--r--   1 ubuntu supergroup       7913 2011-06-06 20:47 /user/ubuntu/outputwords/part-00000
```

- Result of the word count example

```
$ hadoop-0.20 fs -cat outputwords/part-00000
...
based   1
be      20
being   1
below   3
below   2
between 1
beyond  1
blank   12
block   1
by      26
...
```

Setting up a Hadoop Cluster with Cloudera CDH3 (1/5)

- This installation guide installs a distributed Hadoop Cluster
  - **Distributed Mode (Multi Node Cluster)**
- These instructions base of `http://cloudera-tutorial.blogspot.com`
- Stop running Cloudera services

```
$ for x in /etc/init.d/hadoop-* ; do sudo $x stop ; done
```

- List alternative Hadoop configurations

```
$ update-alternatives --display hadoop-0.20-conf
```

- Copy the default configuration

```
$ sudo cp -r /etc/hadoop-0.20/conf.empty /etc/hadoop-0.20/conf.cluster
```

- Activate the new configuration

```
$ sudo update-alternatives --install /etc/hadoop-0.20/conf hadoop-0.20-conf
  /etc/hadoop-0.20/conf.cluster 50
```

- Check the new configuration

```
$ update-alternatives --display hadoop-0.20-conf
hadoop-0.20-conf - auto mode
  link currently points to /etc/hadoop-0.20/conf.cluster
/etc/hadoop-0.20/conf.cluster - priority 50
/etc/hadoop-0.20/conf.empty - priority 10
/etc/hadoop-0.20/conf.pseudo - priority 30
Current 'best' version is '/etc/hadoop-0.20/conf.cluster'.
```

Setting up a Hadoop Cluster with Cloudera CDH3 (2/5)

- Start an additional instance (`ami-08f40561`) (⟹ Slave)
  - DNS: `ec2-50-17-77-111.compute-1.amazonaws.com`
- Insert alias entries for the nodes in `/etc/hosts`

```
10.122.67.221 ec2-50-17-58-144.compute-1.amazonaws.com master
10.120.69.158 ec2-50-17-77-111.compute-1.amazonaws.com slave1
```

- Install SSH client and server

```
$ sudo apt-get install openssh-server openssh-client
```

- Generate ssh keys to login without a password

```
$ ssh-keygen -t rsa -P ""
```

- Copy SSH key in `$HOME/.ssh/id_rsa.pub` to the Slave node into `$HOME/.ssh/authorized_keys`
- `/etc/hadoop-0.20/conf.cluster/masters`
  - One line with the public DNS or alias for each Master (Namenode)
  - If multiple Masters exist (⟹ adjust the file `masters`)
  - In this example, the file `masters` contains only `master`

Setting up a Hadoop Cluster with Cloudera CDH3 (3/5)

- /etc/hadoop-0.20/conf.cluster/slaves
  - One line with the public DNS or alias for each Slave node
  - Slaves are nodes which run the Datanode and/or Tasktracker services
  - In this example, the file slaves contains only slave1
- /etc/hadoop-0.20/conf.cluster/core-site.xml

```
<configuration>
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:54310</value>
</property>
</configuration>
```

- /etc/hadoop-0.20/conf.cluster/mapred-site.xml

```
<configuration>
<property>
  <name>mapred.job.tracker</name>
  <value>master:54311</value>
</property>
</configuration>
```

Setting up a Hadoop Cluster with Cloudera CDH3 (4/5)

- `/etc/hadoop-0.20/conf.cluster/hdfs-site.xml`
  - `<value>1</value>` is the number of Slaves

```
<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
</configuration>
```

- Import the package sources and keys on the Slave and install the Hadoop packages

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ubuntu maverick partner"
$ sudo add-apt-repository "deb http://archive.cloudera.com/debian maverick-cdh3 contrib"
$ sudo curl -s http://archive.cloudera.com/debian/archive.key | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk
$ sudo apt-get install hadoop-0.20-conf-pseudo
```

- Copy the directory `/etc/hadoop-0.20/conf.cluster` to the Slave
  - On the Master:

```
$ sudo apt-get install zip
$ sudo zip -r conf.cluster.zip  /etc/hadoop-0.20/conf.cluster
$ scp conf.cluster.zip slave1:~
```

  - On the Slave:

```
$ sudo apt-get install zip
$ sudo unzip -d / conf.cluster.zip
```

Setting up a Hadoop Cluster with Cloudera CDH3 (5/5)

- Insert alias entries for the nodes into `/etc/hosts` on the Slave

```
10.122.67.221 ec2-50-17-58-144.compute-1.amazonaws.com master
10.120.69.158 ec2-50-17-77-111.compute-1.amazonaws.com slave1
```

- Activate new configuration on the slave

```
$ sudo update-alternatives --install /etc/hadoop-0.20/conf hadoop-0.20-conf
  /etc/hadoop-0.20/conf.cluster 50
```

- Start the services on all nodes to activate the configuration

```
$ for x in /etc/init.d/hadoop-0.20-*; do sudo $x start; done
```

- Stop the services on all nodes

```
$ for x in /etc/init.d/hadoop-0.20-*; do sudo $x stop ; done
```

- Format the Namenode

```
$ sudo -u hdfs hadoop namenode -format
```

- Start the services on the Master (Namenode)

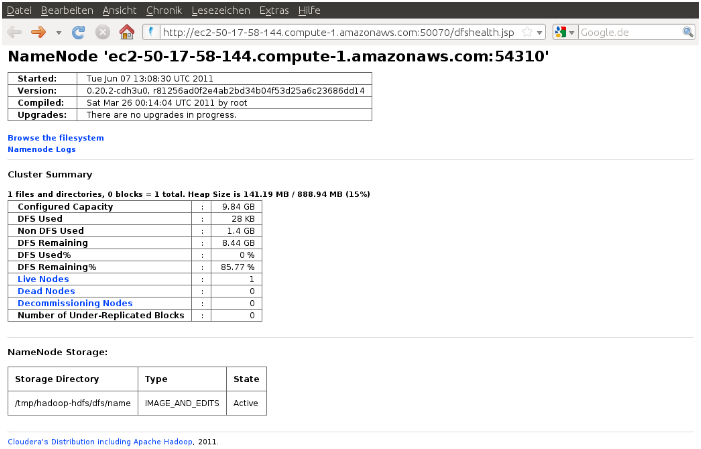
```
$ sudo /etc/init.d/hadoop-0.20-namenode start
$ sudo /etc/init.d/hadoop-0.20-secondarynamenode start
$ sudo /etc/init.d/hadoop-0.20-jobtracker start
```

- Start the services on the Slave (Datanode)

```
$ sudo /etc/init.d/hadoop-0.20-datanode start
$ sudo /etc/init.d/hadoop-0.20-tasktracker start
```

- If all services start, the installation of the Cluster was successful

Web interface of the Namenode


Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://ec2-50-17-58-144.compute-1.amazonaws.com:50070/dfshealth.jsp
Google.de

NameNode 'ec2-50-17-58-144.compute-1.amazonaws.com:54310'

Started:	Tue Jun 07 13:08:30 UTC 2011
Version:	0.20.2-cdh3u0, r81256ad0f2e4ab2bd34b04f53d25a6c23686dd14
Compiled:	Sat Mar 26 00:14:04 UTC 2011 by root
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

1 files and directories, 0 blocks = 1 total. Heap Size is 141.19 MB / 888.94 MB (15%)

Configured Capacity	:	9.84 GB
DFS Used	:	28 KB
Non DFS Used	:	1.4 GB
DFS Remaining	:	8.44 GB
DFS Used%	:	0 %
DFS Remaining%	:	85.77 %
Live Nodes	:	1
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

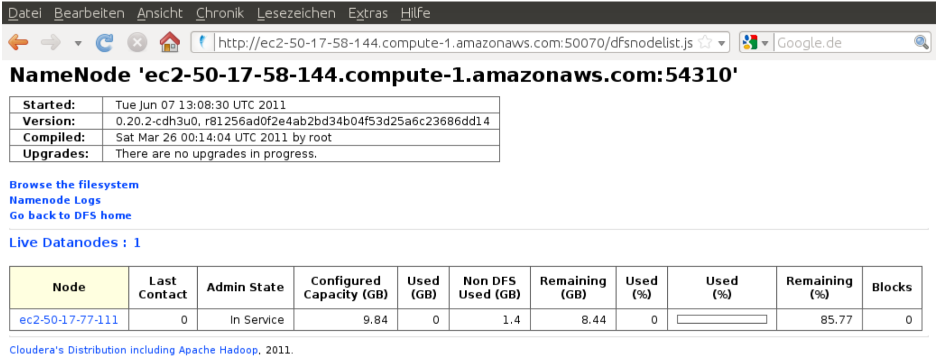
NameNode Storage:

Storage Directory	**Type**	**State**
/tmp/hadoop-hdfs/dfs/name	IMAGE_AND_EDITS	Active

Cloudera's Distribution including Apache Hadoop, 2011.

- The Datenode was detected

Web interface of the Namenode – Detail View Datanodes


Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://ec2-50-17-58-144.compute-1.amazonaws.com:50070/dfsnodelist.js
Google.de

NameNode 'ec2-50-17-58-144.compute-1.amazonaws.com:54310'

Started:	Tue Jun 07 13:08:30 UTC 2011
Version:	0.20.2-cdh3u0, r81256ad0f2e4ab2bd34b04f53d25a6c23686dd14
Compiled:	Sat Mar 26 00:14:04 UTC 2011 by root
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

Live Datanodes : 1

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
ec2-50-17-77-111	0	In Service	9.84	0	1.4	8.44	0		85.77	0

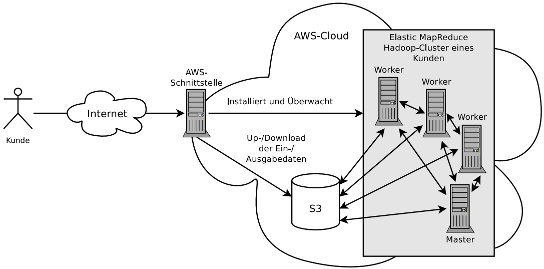
Cloudera's Distribution including Apache Hadoop, 2011.

- Detail view of the Datanodes

Amazon Elastic MapReduce

- Elastic MapReduce (EMR) is a service for virtual Hadoop Clusters
- It's easier/faster to start with EMR MapReduce jobs compared with manually creating a Hadoop Cluster in EC2

- Input data and results are stored inside S3
- Information about the current state of the Hadoop jobs are stored inside SimpleDB



Maximilian Hoecker. Hadoop as a Service (HaaaS) auf Basis von Eucalyptus und Cloudera. Bachelorthesis. HS-Mannheim (2011)

Amazon Elastic MapReduce (EMR)

- For starting a MapReduce application, a **Job-Flow** must be specified
  - A Job-Flow is a configuration of a Hadoop Cluster (in EC2)
  - The configuration contains among others the instance type and MapReduce parameters
- Each Job-Flow is split into **Steps**
  - A step is either a **MapReduce-Step** (MapReduce application) or a **Configuration-Step** (configure script or configuration command to configure the EC2 instances)
- EMR executes all steps in sequential order
- First, EMR executes the configuration steps to configure the Cluster and next executes the MapReduce applications
- Job-Flows can be created and executed either via command-line tools, via the web interface, or via the SOAP and REST interfaces

Other MapReduce implementations

- Besides Hadoop, other MapReduce implementations exist
- Examples:
  - **Quizmt** from MySpace
    - Framework, developed with .NET
    - Free software (GPLv3)
    - `http://qizmt.myspace.com`
  - **Disco**
    - Framework, developed with Erlang and Python
    - Free software (BSD License) of the Nokia Research Center
    - `http://discoproject.org`
  - **Skynet**
    - Framework, developed with Ruby
    - Free software (MIT License)
    - `http://skynet.rubyforge.org`
  - **Plasma**
    - Framework, developed with Ocaml
    - Uses the distributed filesystem PlasmaFS
    - Free software (GPL)
    - `http://plasma.camlcity.org/plasma/`