Cloud Computing and MapReduce
An Introduction

Thanasis Vergoulis

Knowledge and Database Systems Laboratory NTUA
&
Institute for the Management of Information Systems (IMIS)

February 4, 2011
Data & Information Explosion

- Our ability to collect data is constantly improved.
- **Rapid increase** in the amount of collected data.
- We want to **analyze** these data.
- **More computational power** is needed.

Large Hadron Collider (CERN, Zurich)

15,000,000GB/year
Data & Information Explosion, Google

Web pages indexed

Year

1 trillion
1 billion
26 million


http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html
Improving processors & Moore’s Law

- Processing speed is linked to the number of transistors on a chip.
- For decades, the previous number follows the Moore’s law.

Quotation (Gordon Moore, Electronics Magazine 1965)

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. [...] Certainly over the short term this rate can be expected to continue, if not to increase.

Definition (Moore’s Law)

The number of transistors on a chip doubles every 18 months, for the same cost.
Improving processors & Moore’s Law

CPU Transistor Counts 1971-2008 & Moore’s Law

Curve shows ‘Moore’s Law’: transistor count doubling every two years.

Date of introduction

Improving processors & Moore’s Law

- Because of problems like quantum tunnelling, Moore’s law can’t hold forever.

Quotation (Gordon Moore, 2005)

"It can’t continue forever. [...] In terms of size [of transistor] you can see that we’re approaching the size of atoms which is a fundamental barrier, but it’ll be two or three generations before we get that far - but that’s as far out as we’ve ever been able to see.”

- These problems have effects on our attempts to analyze the vast amount of data collected by the next generation sensors - machines.
- Other problems (e.g., memory bandwidth).
- Solutions??
In parallel computing, systems support “real” concurrency by using multiple processors.

**Architectures for parallel systems:**
- Shared memory, e.g., a computer with multiple processors.
- Shared disk
- Shared nothing, e.g., a cluster of computers.
- Hybrid

In “shared memory” architectures processors communicate by using the shared memory.

In “shared nothing” architectures processors communicate by using messages over the network.

Usually we refer to “shared nothing” systems as *distributed systems*. 
Cloud Computing

- In recent years ”Cloud” and ”Cloud Computing” became buzzwords.
- The definition of Cloud is still ... cloudy! (everyone has his own)
The Cloud

- A Cloud is ...
  - ... an infrastructure, transparent to the end-user, ...
  - ... which is used by a company or organization to provide services to its customers via network ...
  - ... where the infrastructure resources are used elastically and the customer is charged according to usage.

- under the hood: large clusters of commodity hardware
- terms: virtualization, elasticity, utility computing, pay-as-you-go
Commodity hardware

- **Before Cloud:**
  - “**Coarse grain**” philosophy.
  - Demanding jobs need powerful computers.
  - Powerful computers are expensive.

- **The Cloud alternative:**
  - “**Fine grain**” philosophy.
  - Large amount of commodity hardware is cheaper.
Cloud vs. Grid

- Similar terms.
  - Grid comes from academia.
  - Cloud comes from enterprise.

- Similarities:
  - Distributed computing.
  - Large scale clusters.
  - Commodity hardware.
  - Heterogeneous cluster.

- Differences:
  - Cloud: Elasticity and pay-as-you-go (if not, it is not Cloud).
  - A Grid...
    - ... can be more loosely coupled and geographically dispersed than a Cloud
    - ... may use the user computer as a part of it (volunteer computing)
    - ... may have been built for a particular purpose and then disappear
Cloud Services

- **Data Storage**
  - Examples:
    - AWS Simple Storage System (S3)

- **Infrastructure as a Service (IaaS)**
  - Provide computing instances (e.g., servers running Linux) as a service.
  - Examples:
    - AWS Elastic Computing Cloud (EC2).

- **Platform as a Service (PaaS)**
  - The delivery of a computing platform and solution stack as a service via network.
  - Examples:
    - Google AppEngine

- **Software as a Service (SaaS)**
  - Software that is deployed over network as a service.
  - Examples:
    - Google Documents
    - Google Calendar
    - Google Reader
Cloud Providers

- Amazon Web Services (AWS)
  - The most complete set of Cloud services.

- Google App Engine
  - [http://code.google.com/appengine/](http://code.google.com/appengine/)

- IBM Cloud

- Microsoft Windows Azure
Cloud File Systems

- Traditional distributed file systems (DFSs) need modifications.
- Like in traditional DFSs we need...
  - ... performance
  - ... scalability
  - ... reliability
  - ... availability
- Differences:
  - Component failures are the norm (large number of commodity machines).
  - Files are huge ($\geq 100$MB).
  - Appending new data at the end of files is better than overwriting existing data.
  - High, sustained bandwidth is more important than low latency.
- Examples of Cloud DFSs: Google File System (GFS), Amazon Simple Storage System (S3), Hadoop Distributed File System (HDFS).
GFS: The Google File System

- Familiar file system interface.
  - Files are organized hierarchically in directories.
  - Files are identified by path names.
  - File operations:
    - create
    - delete
    - open
    - close
    - read
    - write

- Some extra operations:
  - snapshot: Creates a copy of a file/directory in low cost.
  - append: Guarantees atomicity during multiple appends.

---

MapReduce

- A **programming paradigm** that comes with a **framework** to provide to
  the programmers an **easy way** for **parallel and distributed computing**.

**MapReduce facts & properties:**

- Designed by Google (published in 2004).
- Designed to scale well on large clusters > Perfect for Cloud Computing.
- Input & output data stored in a distributed file system.
- Fault tolerance, status & monitoring tools.
- It is attractive because it provides a simple model.
- More than 10,000 distinct MapReduce programs have been implemented in Google.
  
  - Graph processing, text processing, machine learning, statistical machine translation etc.

---

3 Jeffrey Dean, Sanjay Ghemawat. “MapReduce: Simplified Data Processing for Large Clusters”. OSDI 2004
Why Google introduced MapReduce?

- Information Retrieval (IR) problem: Indexing the Web\(^4\).
- Because of the explosion of data Google decided:
  - to use large clusters of commodity hardware (i.e., Cloud)
  - to introduce a framework that makes easy programming on Cloud (i.e., MapReduce)

---

\(^4\)http://www.youtube.com/watch?v=BNHR6IQJGZs
Functional Programming and MapReduce

- Functional programming:
  - Functions always the same result for the same input.
  - Functions always use call by value.
  - Easy to program, easy to parallelize.

- MapReduce borrows from functional programming.
  - map: like map function of functional programming languages.
  - reduce: like fold function of functional programming languages.

\[
S = \text{map}(f(L))
\]

\[
S = \text{foldl}(f(L,a))
\]
The MapReduce paradigm

- **MapReduce program input:** A set of files stored in the DFS.
- **MapReduce program output:** A set of files stored in the DFS.
- Each program is divided into two subprograms:
  - Map subprogram.
  - Reduce subprogram.
- MapReduce framework is responsible for executing a large number of instances of each subprogram on a computer cluster.
- Map subprogram instances:
  - Process a part of the input.
  - Produce \((\text{key}, \text{value})\) pairs.
- Reduce subprogram instances:
  - Process \((\text{key}, \text{value})\) pairs having particular keys.
  - Produce output.
- The execution of multiple map instances reminds of the map function of functional programming.
- The execution of multiple reduce instances reminds of the fold function of functional programming.
The MapReduce paradigm: The map subprogram

- Multiple map instances run on different nodes of a cluster.
- Each map instance...
  - ... gets disjoint portion of the input from the DFS (files, records of files, etc.)
  - ... processes its input to produce \((key, value)\) pairs
  - ... uses a split function to partition these pairs into \(R\) disjoint buckets according to their key
  - ... writes any completed bucket to the disk (locally)
- The input is given to the instances by the MapReduce Scheduler.
- The split function and \(R\) are given by the user.
- The output of each map instance consists of \(R\) files on the disk.
- The map subprogram is an arbitrary computation in a general-purpose language.
The MapReduce paradigm: The reduce subprogram

- Multiple reduce instances run on different nodes of a cluster.
- Each reduce instance...
  - ... gets those \((key, value)\) pairs which have particular keys
  - ... processes its input to produce the output
  - ... writes the output to the GFS (not locally!)
- The fetching of \((key, value)\) pairs is done by remote reads (use of pull instead of push).
- The output of each reduce instance is part of the output (i.e., the output is distributed in many files).
- The map subprogram is an arbitrary computation in a general-purpose language.
The user starts the MapReduce program on master node...

The input is on the DFS...
The Overall Flow of a MapReduce Program

The user program creates many copies of itself on the cluster...

The copy on the master node is special...

user
master node
fork
worker
worker
worker
worker

Cluster

Files:
- files on the DFS
- local files
- active node

Thanasis Vergoulis (Knowledge and Database Systems Laboratory NTUA & Institute for the Management of Information Systems)
The Overall Flow of a MapReduce Program

Master assigns map tasks to workers...

user

Cluster

master node

assign map

worker

assign

worker

worker

worker

# files on the DFS

# local files

# active node

Thanasis Vergoulis (Knowledge and Database Systems Laboratory NTUA & Institute for the Management of Information Systems)
The Overall Flow of a MapReduce Program

Cluster

user

master
node

Map-workers read their input splits from DFS...

worker

worker

worker

worker

i

i

i

read

files on the DFS: local files

: active node

: files on the DFS

: local files

: active node
The Overall Flow of a MapReduce Program

Map-workers produce the (key,value) pairs, write them in the bucket files, and send the locations to the master...
The Overall Flow of a MapReduce Program

Master assigns reduce tasks to workers and inform them about the locations of the bucket files...

- Master node
- Worker nodes
- Files on the DFS
- Local files
- Active node

Thanasis Vergoulis (Knowledge and Database Systems Laboratory NTUA & Institute for the Management of Information Systems (IMIS))
February 4, 2011
The Overall Flow of a MapReduce Program

Reduce-workers read the bucket files from the local disks of the map workers using remote procedure calls...
The Overall Flow of a MapReduce Program

Reduce-workers sort (key,value) pairs according to key, do processing, and write the results on DFS...
The Overall Flow of a MapReduce Program

Reduce-workers notify the master and master returns to the user program...

- **user**
- **Cluster**
- **master node**
- **worker**

- **Files on the DFS**: active node
- **Local files**

**Legend**:
- □: files on the DFS
- □: local files
- □: active node
The WordCount MapReduce example

**map** (String input_key, String input_value):
for each word w in input_value:
EmitIntermediate(w, "1");

- **input_key**: document name
- **input_value**: document contents

**reduce** (String output_key, Iterator intermediate_values):
int result = 0;
for each v in intermediate_values:
result += parseInt(v);
Emit(AsString(result));

- **output_key**: a word
- **output_values**: a list of counts
Some Details

- **Locality is good.**
  - Master assigns to Map-workers data chunks that are stored on their disk or on disks of machines which are near to them (e.g., on the same rack).

- **Fault Tolerance**
  - Master detects worker failures and then orders another worker to re-execute.
  - If particular \((key, value)\) pairs create problems, skips them during re-execution.
Optimizations

- **Redundant execution of map tasks.**
  - A slow map-worker can be a bottleneck.
  - Pushing the \((key, value)\) pairs to the reduce-workers would complicate fault tolerance if the map-worker crashes at the middle of a task.
  - Idle map-workers are used to help slow workers, thus, using them as reduce-workers slows down map phase.
  - Master orders a map-worker who finished his task to execute also the task of the slow map-worker.
  - If two map-workers complete the same task, the results of the second are skipped.

- **Combiners.**
  - Each combine instance works like reducer instances on the output data of one map-worker.
  - Example: Use a combiner in the Word Count program to count words in the input of each mapper. Then, the reducer just adds the resulting number of appearances.
Hadoop\(^5\)

- Open-source implementation of MapReduce.
- Written in Java.
- It is a top-level Apache project.
- Large number of contributors, like Yahoo!.
- Cloud providers, like Amazon, provide MapReduce to their customers by using Hadoop.
- Hadoop comes with its own distributed file system: HDFS (it is like GFS).

\(^5\)http://hadoop.apache.org/
MapReduce and Parallel Execution of Aggregation Queries

- Aggregation queries can be parallelized easily by using MapReduce:

```
SELECT department_id, AVG(salary)
FROM employee
GROUP BY department_id
```
MapReduce and Parallel Execution of Aggregation Queries

- Aggregation queries can be parallelized easily by using MapReduce:

```sql
SELECT department_id, AVG(salary)
FROM employee
GROUP BY department_id
```

- **map phase**: Execute the group-by clause.
- **reduce phase**: Execute the aggregate function (AVG).
Aggregation queries can be parallelized easily by using MapReduce:

```
SELECT department_id, AVG(salary)
FROM employee
GROUP BY department_id
```

- **map phase**: Execute the group-by clause.
- **reduce phase**: Execute the aggregate function (AVG).

In general, Analytical Data Management consists of operations which are fairly easy to parallelize using shared nothing parallelization (e.g., scans, multidimensional aggregations, star schema joins).
Parallel Databases & Analytical Data Management

- Analytical Databases:
  - DBMSs for Analytical Data Management
  - Many companies: Teradata, Aster Data, Netezza, Vertica, DATAllegro, Greenplum, Infobright, Exadata (Oracle) etc.
  - Market (2009): $3.98 billion (27% of database software market)

- Parallel Databases:
  - Analytical DBMS systems that deploy on a shared-nothing architectures for parallelization.
  - Scale well for tens of nodes.
  - Do not scale well into the hundreds or thousands of nodes:
    - Designed assuming that failures are a rare event.
    - Designed assuming homogeneous machines in general.
    - Not tested for such large clusters because there was no need.

- Why not to use MapReduce instead?
  - Designed for environments where failures are common.
  - Designed for heterogeneous clusters.
  - MapReduce installations run on clusters that contain thousands of nodes.
MapReduce vs Parallel Databases

- The ... “Cloud Wars”:
  - MapReduce Supporters: Jeffrey Dean & Sanjay Ghemawat.
“MapReduce is a major step backwards”

- Criticism by the team of DeWitt & Stonebraker:
  - 17/1/2008, “MapReduce: A major step backwards”\(^6\).
  - 25/1/2008, “MapReduce II”\(^7\).
  - SIGMOD 2009, “A Comparison of Approaches to Large-Scale Data Analysis”.
  - CACM 01/2010, “MapReduce and Parallel DBMSs: Friends or Foes?”.

- Basic arguments:
  - MapReduce\(^6\) does not support schema for data.
  - MapReduce programming incorporates low-level data manipulation.
  - MapReduce\(^6\) abandons well-known database techniques (e.g., indexing).
  - MapReduce\(^6\) it is not novel.
  - MapReduce\(^7\) is missing features and it is incompatible with DBMS tools.
  - There is no real need for scaling to hundreds and thousands of machines for analytical data management.

---

\(^6\) http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/

\(^7\) http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/
Criticism: Schema Support

- **Schema for data:**
  - The fields and their data types are recorded in storage (system catalogs).
    - Moreover, the system catalogs can be queried by SQL.
  - Ensures that input records obey these restrictions.
  - Helps the separation of data from the application.

- **No support for data schema means...**
  - ... that we cannot keep garbage out of the datasets
  - ... the programmer must discover the structure of the data by examining the code
    - Difficult.
    - Impossible if not open-source.
  - ... the programmer must write a custom parser.
  - ... different programmers on the same data must agree on a single structure.
Criticism: Low-level data manipulation

- Data management in MapReduce goes low-level.
- 1970s debate:
  - Relational: State what you want to get (high-level languages).
  - Codasyl: Present an algorithm for how to get what you want.
- Relational view was better:
  - easier to write
  - easier to modify
  - easier for a new person to understand
Criticism: No use of Database techniques

- MapReduce doesn’t support:
  - Indexing.
    - Full scan of the data is needed.
  - Data skew problem solutions.
  - Sockets to push the split files.
    - MapReduce materializes the split files and use pull.
    - This may slow down the disk transfer rate.

Quotation (from “MapReduce: A major step backwards”)

“[…] we have serious doubts about how well MapReduce applications can scale.”

- Irony: Google indexes the whole Web using MapReduce…
Criticism: Missing features & incompatibility

- **Missing features:**
  - Bulk loader (to make easier data loading)
  - Indexing
  - Easy Updates
  - Transactions
  - Integrity constraints (no schema!)
  - Referencial integrity (no schema!)
  - Views (with views, the schema can change without having to rewrite the application)

- **MapReduce is incompatible with existing:**
  - Report writers (to prepare reports for human visualization)
  - Business intelligence tools (to enable ad-hoc querying)
  - Data mining tools (to discover structure in data)
  - Replication tools (to replicate data from one DBMS to another)
  - Database design tools (to assist user in constructing a database)

- **Comment:** MapReduce community is going to provide in the future some of the previous features and tools.
Criticism: No need for hundreds of machines

- MapReduce is capable of scaling up to 1000s of nodes.
- The superior efficiency of modern DBMSs alleviates the need to use such massive hardware.
- Even the largest warehouses need tens of machines:
  - eBay (by Teradata, 2009):
    - 2.4 PB relational data
    - 72 nodes
    - 2 quad-core CPUs, 32 GB RAM, 104,300 GB disk
  - Fox Interactive Media (by Greenplum, 2009):
    - 40 nodes
    - Sun X4500 with 2 dual-core CPUs, 16 GB RAM, 48,500 GB disk

- Comment: What if you need to use commodity hardware?
The MapReduce strikes back-1

- The response of the team of Dean & Ghemawat
  - CACM 01/2010, “MapReduce: A Flexible Data Processing Tool”
- Many of the arguments were based on implementation and evaluation shortcomings not fundamental to the MapReduce model.
- No indices:
  - If a work must be done on a part of data that satisfy a particular condition:
    - We can have these data organized in an index (e.g., in a relational database)
    - We executed the query on the index
    - The result is given to MapReduce to do the work
  - Use of BigTable
    - We can read data from specific rows or columns
  - Data within certain date range
    - Nearly every logging system rolls over a new log file periodically and embeds the rollover time in the name of each log file.
    - Read only the needed log files.
The MapReduce strikes back-2

- Complex functions:
  - In many cases the function is too complicated to be expressed easily in an SQL query.
    - Extracting the set of outgoing links from a collection of HTML documents and aggregating by target document.
    - Fault-tolerant parallel execution of programs written in higher-level languages across a collection of input data
  - UDFs can be combined with SQL queries
    - UDF support in parallel databases is either buggy or missing.

- Heterogeneous Systems:
  - Many production environments contain mix of storage systems.
  - MapReduce provides a simple model for analyzing data in such heterogenous systems.
  - There are several supported storage systems for input in MapReduce.
  - New storage systems can be added by end users.
  - Parallel databases need a program to load the data (maybe a MapReduce program! :P).
The MapReduce strikes back-3

- No schema
  - Google does not use inefficient text inputs and custom parsers.
  - Google uses Protocol Buffer format.
    - Compiler-generated code is used to hide details of encoding & decoding from application code.
    - The Protocol Buffers use an optimized binary representation that is more compact and much faster to encode and decode than the textual formats.

- Fault tolerance
  - In a push model, failure of a reducer would force re-execution of all map tasks.
The Facebook MapReduce Data Warehouse System\(^8\)

- Facebook was one of the first enterprises that implemented a large data warehouse system using MapReduce technology rather than a DBMS.
- 2.5PB of data managed by Hadoop (04/2009).
- Facebook’s Hadoop/Hive system ingests 15 TB of new data per day.
- Facebook has the second largest Hadoop installation: 610 nodes in one cluster (now maybe more than 1,000 nodes).

\(^8\)http://www.dbms2.com/2009/04/15/cloudera-presents-the-mapreduce-bull-case/
Why Facebook has chosen Hadoop?

- License/maintenance costs (Hadoop & Hive are free).
- Flexibility of modifying open-source code.
- Ability to run on cheap hardware.
- Scaling to lots of nodes.
- Facebook programmers believe that Hadoop has performance advantages over DBMS (no overhead with transactions etc.)
Consistency on the Cloud

- The nature of consistency problems in distributed systems differs from this of consistency problems in traditional databases.
  - Replication may create inconsistencies.
  - Inconsistencies appear during reads.

What replication means:
  - Storage: We hold \(N\) replicas of each piece of data.
  - Writes: Each write operation waits for update confirmation from \(W (\leq N)\) replicas.
  - Reads: Each read operation gets the value of \(R (\leq N)\) replicas.

Scenario 1:
  - \(W=N, R=1\)
    - When I write, I ensure that all replicas are up-to-date.
    - When I read, I read just one of them.
    - Impossible to have inconsistencies.

Scenario 2:
  - \(W<N, R=1\)
    - Maybe the one replica I read haven’t been updated. (why?)
Consistency on the Cloud: Network partitions

- The reason why the update of some replicas may fail is network partitions.
  - I.e., (for some reason) some nodes are (temporarily) unreachable.

- If later someone read for the previously unreachable node, inconsistency occurs.
Strong consistency: After an update, any subsequent access will return the updated value.

Iff $W + R > N$, then we have strong consistency.

Example 1:
- $N=4$, $W=2$, $R=3$
- At least $2/4$ replicas are up-to-date.
- Each read gets $3$ replicas $> At least $1$ is up-to-date $> No$ inconsistencies

Example 2:
- $N=4$, $W=3$, $R=2$
- At least $3/4$ replicas are up-to-date.
- Each read gets $2$ replicas $> At least $1$ is up-to-date $> No$ inconsistencies

Why not to follow this simple rule?
Consistency on the Cloud: Availability

- If during write $X > N - W$ replicas are down, then write fails.
- If during read $X > N - R$ replicas are down, then read fails.
- The previous are two cases where the system is not available.
- An ideal system would be always available.
- Recall that to achieve consistency we must increase $W$ and $R$ values.
- Large values of $W$ and $R$ decrease availability of writes and reads, respectively.
Consistency on the Cloud: The CAP theorem

Definition (CAP theorem)
Among the three desired properties of distributed systems:
- Consistency
- Availability
- Partition tolerance
only two can be achieved at any given time.

- Previously known as Brewer’s Conjecture (PODC 2000 keynote).
- Proof: Seth Gilbert and Nancy Lynch (ACM SIGACT 2002).
- Due to replication, all large scale distributed systems are tolerant to network partitions.
  - Choice 1: Drop Availability
  - Choice 2: Drop Consistency
- For Analytical Data Management usually availability is more important than consistency.
Consistency on the Cloud: Eventual Consistency

- When $W + R \leq N$ weak consistency arises.
  - The system does not guarantee that subsequent accesses will return the updated value.

- A special case of weak consistency is eventual consistency.
  - The system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value.
  - DNS (Domain Name System) uses eventual consistency.

- The system heals inconsistencies alone, if $R \geq 2$:
  - During read, if one replica is up-to-day, the others can be updated to this value.