Security of Cloud Computing

Fabrizio Baiardi
f.baiardi@unipi.it

Syllabus

- Cloud Computing Introduction
  - Definitions
  - Economic Reasons
  - Service Model
  - Deployment Model
- → Supporting Technologies
  - Virtualization Technology
  - Scalable Computing = Elasticity
- Security
  - New Threat Model
  - New Attacks
  - Countermeasures

Elastic, scalable, parallel

- All these term implies that not all the computations can exploit at best a cloud architecture
- If an application running on a single machine is migrated to a single VM on a cloud
  - Benefits in term of availability and reliability because the whole VM can be migrated to another machine if the current one crashes
  - Crash cannot be masked
  - No benefit in terms of performance even if a large number of physical hosts is available
- An application should be conceived from the beginning as a huge number of processess mapped onto a large number of VMs
- The final performance depends on the mapping of the VMs onto the physical hosts under the assumption that there is a minimal performance loss with respect to the sequential case when several VMs are mapped onto the same node

Elastic, Scalable, Parallel Computing

This point discusses

a) how can we decompose an application into a large number of concurrent processes

b) how to structure the software to support an application that has been decomposed into a large number of concurrent processes

Goal

a) simple decomposition

b) linear speed up: the time to implement a computation decreases as 1/n where n is the number of processes resulting from the decomposition

Parallel and Distributed Computing

Parallel computing can apply distinct strategies :

Vector processing of data (SIMD)

Multiple CPUs in a single computer (MIMD)

Distributed computing is multiple CPUs across many computers (MIMD distributed memory)

Example of distributed computing

- Weather prediction
- Indexing the web (Google)
- Simulating an Internet-sized network for networking experiments (PlanetLab)
- Speeding up content delivery (Akamai)

Elasticity = fundamental cloud property

- Elasticity is the power to scale computing resources up and down easily and with minimal friction.
- It is important to understand that elasticity will ultimately drive most of the benefits of the cloud.

- A cloud architect, has to internalize this concept and work it into an application architecture in order to take maximum benefit of the cloud.

Elasticity = fundamental cloud property

- Traditionally, applications have been built for fixed, rigid and pre provisioned infrastructure. Companies never provision and install servers on daily basis. As a result, most software architectures do not address the rapid deployment or reduction of hardware.
- Since the provisioning time and upfront investment for acquiring new resources was too high, software architects never invested time and resources in optimizing for hardware utilization. It was acceptable that the hardware running the application was under-utilized
- The notion of “elasticity” within an architecture was overlooked becausehaving new resources in minutes was impossible

Elasticity = fundamental cloud property

- With the cloud, this mindset needs to change. Cloud computing streamlines the process of acquiring the necessary resources
- No longer any need to place orders ahead of time and to hold unused hardware captive
- Cloud architects can request what they need mere minutes before they need it or automate the procurement process, taking advantage of the vast scale and rapid response time of the cloud. The same is applicable to releasing tunneeded or under-utilized resources when you don’t need them

Elasticity = fundamental cloud property

- A cloud architect should think creatively implement elasticity to fully exploit the cloud
- Infrastructure to run daily nightly builds and perform regression and unit tests every night at 2:00 AM for two hours is sitting idle for rest of the day. With elastic infrastructure, one can run nightly builds on boxes that are “alive” and being paid for only for 2 hours in the night.
- An internal trouble ticketing web application that always used to run on peak capacity (5 servers 24x7x365) to meet the demand during the day can now be provisioned to run on-demand (5 servers) from 9AM to 5 PM and 2 servers for (5 PM to 9 AM) based on the traffic pattern.

MapReduce Design Goals

Scalability to large data volumes:

1000's of machines, 10,000's of disks

Cost-efficiency:

Commodity machines (cheap, but unreliable)

Commodity network

Automatic fault-tolerance (fewer administrators)

Easy to use (fewer programmers)

Challenges

Cheap nodes fail, especially if you have many

Mean time between failures for 1 node = 3 years

Mean time between failures for 1000 nodes = 1 day

Solution: Build fault-tolerance into system

Commodity network = low bandwidth

Solution: Push computation to the data

Programming distributed systems is hard

Solution: Data-parallel programming model: users write “map” & “reduce” functions, system distributes work and handles faults

Fault Tolerance

3. If a task is going slowly (straggler):
   - Launch second copy of task on another node ("speculative execution")
   - Take the output of whichever copy finishes first, and kill the other

Surprisingly important in large clusters
   - Stragglers occur frequently due to failing hardware, software bugs, misconfiguration, etc
   - Single straggler may noticeably slow down a job

"Fault" Tolerance

- Nodes fail
  - Re-run tasks
- Nodes are slow (stragglers)
  - Run backup tasks (speculative execution)
  - To minimize job's response time
    - Important for short jobs

Speculative execution

The scheduler schedules backup executions of the remaining *in-progress tasks*

The task is marked as completed whenever either the primary or the backup execution completes

Improve job response time by 44% according Google's experiments

Seems a simple problem, but

Resource for speculative tasks is not free

- How to choose nodes to run speculative tasks?
- How to distinguish "stragglers" from nodes that are slightly slower?

Stragglers should be found out early

Speculative execution vs Monitoring

- How to choose nodes to run speculative tasks?
- How to distinguish "stragglers" from nodes that are slightly slower?

Stragglers should be found out early

All these problems require the adoption of a proper execution monitoring

An execution monitoring is a tool to analyze actual resource usage to discover problems in program execution

Fault Tolerance

High availability

fast recovery : master and chunkservers restartable in a few seconds

chunk replication: default: 3 replicas.

shadow masters

Data integrity

checksum every 64KB block in each chunk

What's

- **Framework for running applications on large clusters of commodity hardware**
- **Scale: petabytes of data on thousands of nodes**
  - **Storage: Hadoop Distributed FS**
  - **Processing: MapReduce Requirements**
  - **Economy: use cluster of comodity computers**
  - **Easy to use**
    - **Users: no need to deal with the complexity of distributed computing**
  - **Reliable: can handle node failures automatically**

MapReduce Programming Model

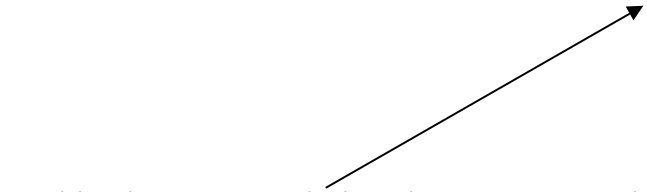
Data type: key-value *records*

Map function:

$$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$$

Reduce function:

$$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$$

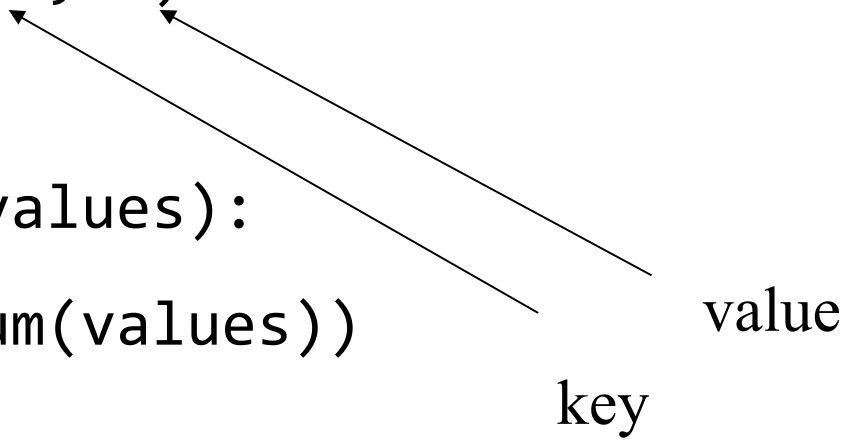


All those with the same key, K_{inter} in the example

Example: Word Count

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
```



Each word in the document is a key

Hadoop: Motivation

- Previous discussion shows that several parallel algorithms can be expressed by a series of MapReduce jobs
- But MapReduce is fairly low-level: must think about keys, values, partitioning, etc
- Can we capture common “job building blocks”?
- Hadoop was inspired by
  - MapReduce
  - Google File System

Hadoop Components

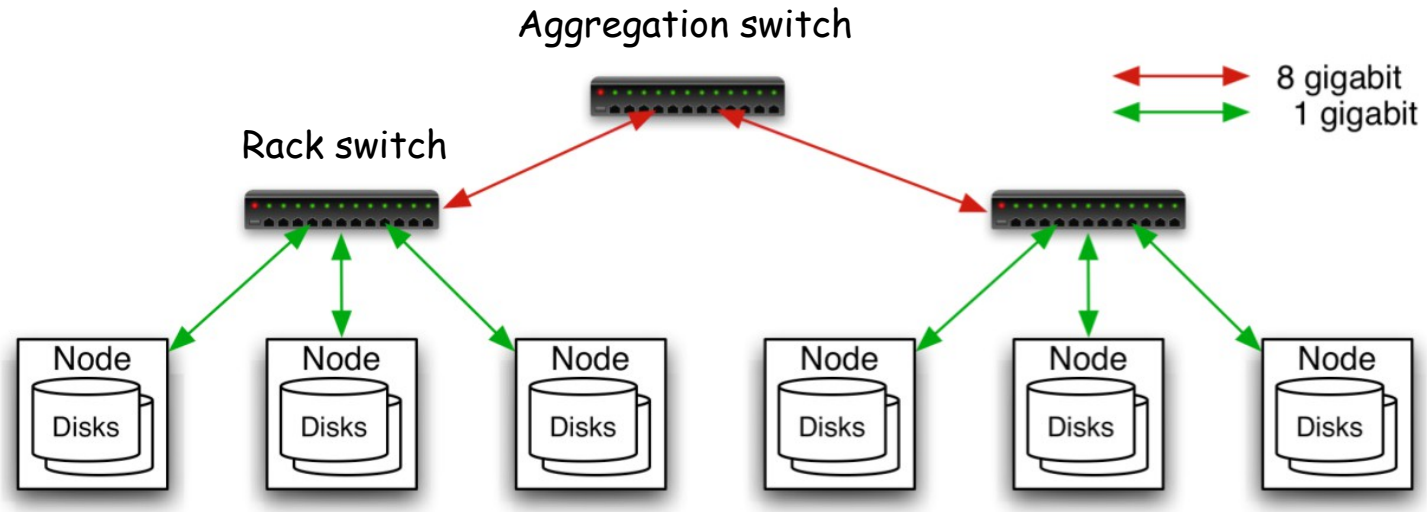
Distributed file system (HDFS)

- Single namespace for entire cluster
- Replicates data 3x for fault-tolerance

MapReduce framework

- Executes user jobs specified as “map” and “reduce” functions
- Manages work distribution & fault-tolerance

Typical Hadoop Cluster



40 nodes/rack, 1000-4000 nodes in cluster

1 Gbps bandwidth within rack, 8 Gbps out of rack

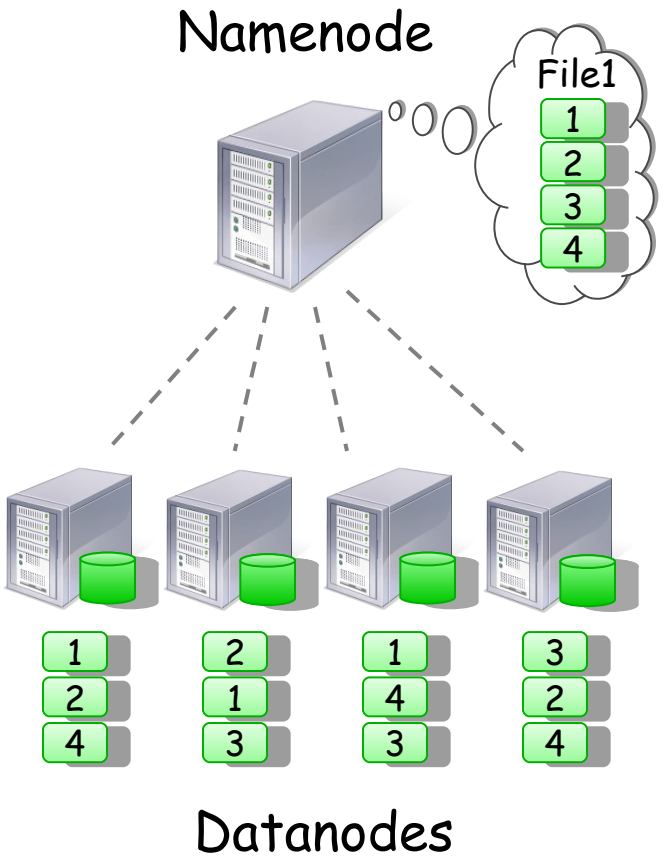
Node specs (Yahoo terasort):

8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

Image from http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/YahooHadoopIntro-apachecon-us-2008.pdf

Hadoop Distributed File System

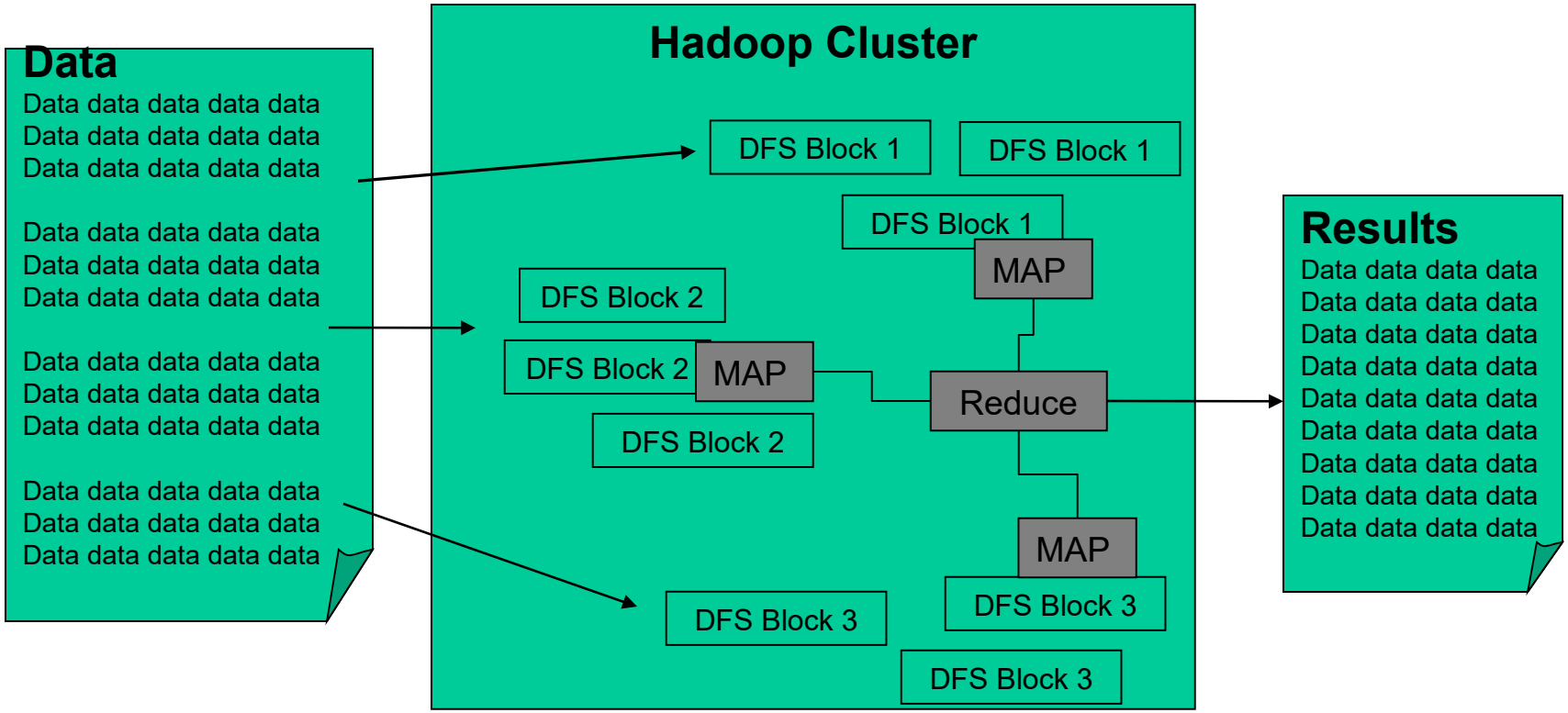
- Files split into 128MB *blocks*
- Blocks replicated across several *datanodes* (usually 3)
- Single *namenode* stores metadata (file names, block locations, etc)
- Optimized for large files, sequential reads
- Files are append-only



HDFS

- Hadoop implements MapReduce using the Hadoop Distributed File System (HDFS)
- MapReduce divides applications into many small blocks of work = HDFS creates multiple replicas of data blocks for reliability, placing them on compute nodes around the cluster so that MapReduce can then process the data where it is located.
- Hadoop has been demonstrated on clusters with 2000 nodes. The current design target is 10,000 node clusters.

Hadoop Architecture



HDFS

- HDFS assumes that hardware is unreliable and will eventually fail.
- Similar to RAID level except HDFS can replicate data across several machines
- Provides Fault tolerance
- Extremely high capacity storage
- Moving Computation is cheaper than moving data= HDFS is said to be rack aware.