

Cloud computing deployment and management

First colloquium . 29.11.2021

Efrain Fernandez Sangrador (Erasmus+ Student)

1. Introduction to Cloud Computing

Cloud Computing is an **access** model which:

- on the easy way and upon request
- gives access to sharing of configurable computer resource
- resources can be quickly assigned and free (with minimal interaction with provide server).

Like an **Architecture** promotes accessibility and consist on five basic features:

- self service on request
- wide network access
- pooling of resources (agrupación de recursos)
- fast elasticity
- measuring the services provided

NIST **service** models:

- software in the cloud as a service (SaaS)
- platform for the cloud as a service (PaaS)
- infrastructure for the cloud as a service (IaaS)

Deployment models:

- private cloud
- community cloud
- public cloud
- hybrid cloud

Technologies:

- fast broadband networks
- powerful, inexpensive server computers
- high performance virtualization for hardware

examples: Gmail, Wikipedia, Facebook, twitter...

With hardware virtualization support on commodity hardware, shared elastic distributed computing accounting re.emerged on the consumer (services) market.

Main commands in Linux

sed: Search and replace a text

awk: Filtering files

tail: Print the lastest 10 numbers

wc: word count

diff: compares the differences between two files

nc: netcat, to access to our TCP/UDP ports

ps: Process status

ls: List

mkfifo: Create a pipe FIFO

cat: Concat and print in standard output

tee: Copy the standard output to a file

echo: Printing text in screen.

N01 . Linux, OpenSSH (to make cypher communications)

Cloud computing technologies will be covered, including:

- Message Passing Interface (MPI),
- Docker,
- Hadoop, and
- Google App Engine (GAE)

These enable cloud computing stacks. Specifying intro to facilitating cloud computing will be provided:

- virtualization
- networking
- including a definition of the cloud computing term as stemming from the computer security perspective.

Then, an example will be futher elaborated for Hadoop and VirtualBox,

- demonstrating how to deploy and manage a new cloud of virtual computers
- that can be provided as a service to clients
- using the MapReduce programming model
- for parallel execution in the cloud.

2. Parallel and Distributed Computer System Models

2.1. Heterogeneous Computer System

It refers to electronic systems that use a variety of different types of computational units.

Computational Units:

- A general-purpose processor (GPP)
- a special-purpose processor
- a co-processor
- Examples: DSP (Digital signal processor), GPU (graphics processing unit), ASIC, FPGA.

In general, a heterogeneous computing platform consists of processors with different instruction set architectures (ISAs). We'll have CISC (Complex Instruction Set Computer, that normally they finish like a lot of RISC instructions) and RISC (Reduced Instruction Set Computer).

In the past, the advances in technology and frequency scaling allowed the majority of computer applications to increase in performance without requiring structural changes. While these advances continue, their effect is not dramatic as other obstacles such as the memory-wall and power-wall come into play.

Now, with these constraints, the primary method of gaining extra performance out of computing systems is to introduce additional specialized resources.

Independent computing resources, most heterogeneous systems need considered parallel computing or multi-core (computing) systems.

Hybrid computing is a form of heterogeneous computing where asymmetric computational units coexist with a commodity processor.

2.2. Parallelization of CPU Operations

Superscalarity

Some instructions can execute independently:

- arithmetic (EX)
- loading and saving (IF, ID, OF, WB)
- branching (JMP=)

Restrictions of pipelining:

- data dependency,
- control dependency,
- resource conflict,
- output dependency.

2.3. Multiprocessor Organization

- Single Instruction, Single Data Stream (SISD)
- Single Instruction, Multiple Data Stream (SIMD)
- Multiple Instruction, Single Data Stream (MISD)
- Multiple Instruction, Multiple Data Stream (MIMD): A set of general purpose processors, simultaneous execution of various instruction sequences and different sets of data.

Processing methods of communication:

- Symmetric MultiProcessing (SMP)
 - Two or more similar processors
 - Common memory, bus and I/O devices
 - communication through shared memory
 - symmetry: all processors can perform the same set of operations
- NUMA (Non-Uniform Memory Access) systems
 - An alternative to SMPs,
 - all processors have access to the entire memory
 - memory is divided into segments (variable access times)
 - coordination is different from SMPs and cluster architectures
- **cluster** architecture

2.4. Tasks of the Control Unit

- Concurrent execution,
- scheduling,
- synchronization,
- memory management,
- reliability (fiabilidad)
- fault-tolerance

2.5. Parallelisation of programs with a superscalar Architecture

- The application runs currently on multiple **threads** or **processing units**
- Task of the compiler:
 - Determine independent parts to run concurrently.
 - Division among processors.

- Parallel application:
 - implemented for concurrent execution
 - synchronizes between nodes
 - implementation is more advanced
- Parameterized execution: A task is repeated over different data

2.5.1. Parallel and distributed computer system models (terminology)

Distributed computing is themed in computer science, which addresses distributed systems.

Distributed system is a model of computation, which one's components are located on computers, which are linked in a common network and communicate and coordinate their actions by MESSAGE PASSING (over network).

The components have a common **goal**.

Properties of distributed systems:

- Simultaneous component execution
- don't have a shared clock (because it would be delay)
- components fail individually
- Examples: SOA (Service Oriented Architecture), MMOG (Massively Multiplayer Online Games)

A program on a distribution system is called a distributed program.

Some ways to send **messages**:

- HTTP
- RCP
- message queue

The problems to solve with distributed computing have to be divided into many task, each of which is solved by one or more computers.

Each computational entity, has its own local memory, and communicate with **message passing**.

So:

- Parallel computing: All processors access shared memory (exchange info between processors)
- Distributed computing: Each processor has its own memory (info exchange by passing messages).

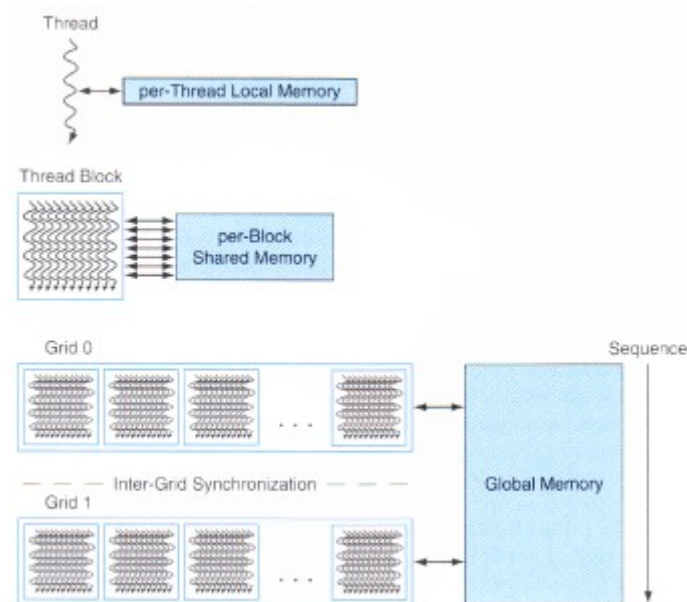
2.6. OpenMP: Multiprocessor Programming

- Parallel programming on CPU using shared memory (most CPU architectures, and Operating Systems)

- Application Programming Interface (API): set of compiler directives, libraries and variables of environment, allows writing cross-platform for multi-threaded.
- C/C++ and fortran.
- Related: MPI, PVM

2.7. CUDA

- Architecture Features
 - GPGPU: General-Purpose Computing on GPU
 - CUDA: Computer Unified Device Architecture
 - A common programming model for general-purpose computing (on the CPU or GPU)
 - NUMA: Nonuniform Memory Access: GPU access memory
 - SIMT: Single Instruction, Multiple Threads: Same instructions execute at multiple threads in a cycle
- CUDA: Thread Indexing.
 - Several hundred threads may execute and communicate (with synchro mechanisms)
 - Kernel threads are grouped in blocks
 - Host computer and CUDA device,
 - program results are same on both: GPU is much faster
 - Independent GPU grids, blocks and threads
 - synchronization
 - Access to memory: cudaMalloc(), cudaFree(), cudaMemcpy()
 - Thread → per-Thread local memory
 - Thread Block → per-Block Shared Memory
 - Grids → Global Memory



A Distributed Computing Example: Web Servers and DNS relay

- DNS resolution request to different IPs.
- Each IP traffic is processed at one computer
- Each computer runs a web server
- An individual task of request handling is a Distributed Task while the larger task of web service to any client is a clustered task.

2.8 Clusters

Is a set of computer resources in numerous locations, related in a common goal. Node allows performing different task or applications.

A cluster can be from a few computers to several thousand and more.

2.9. Grids

A grid computing computers are connected in a network and share sub-tasks. We can create one or more clusters of virtual computers (connected computers that work on the same task together)

Ex: BOINC, [SETI@Home](http://setiathome.berkeley.edu)

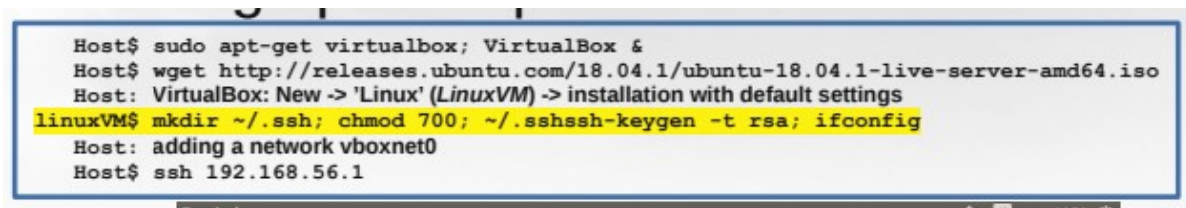
2.10. Peer-to-peer (P2P)

- Any node can be connected to each other
- Nodes act as a client – server at the same time
- Each message can travel through multiple nodes to arrive
- Nodes create virtual connections, and network nodes are announced
- Communication by TCP/IP at the application layer
- Development (desarrollo):

- USENET
- Email: MUA, MSA, **MTA**, MDA
- File sharing: before FTP (File Transfer protocol)+
- Distributed hashing tables (for sharing): in pair the table keeps file ownership for nodes and the nodes keep lists of neighbors (example: **BitTorrent**)

2.11. Clouds

- Instead of sending data for processing, we are sending entire programs (and in doing so we rent hardware, data or software platforms)
- Virtualization of resources (the cloud can host a variety of tasks)
- Rapid renting of clouds
- Redundancy, automatic recovery upon failure
- High scalable, with real-time monitoring
- Setting up a simple virtual enviroment:



```
Host$ sudo apt-get virtualbox; VirtualBox &
Host$ wget http://releases.ubuntu.com/18.04.1/ubuntu-18.04.1-live-server-amd64.iso
Host: VirtualBox: New -> 'Linux' (LinuxVM) -> installation with default settings
linuxVM$ mkdir ~/.ssh; chmod 700; ~/.sshssh-keygen -t rsa; ifconfig
Host: adding a network vboxnet0
Host$ ssh 192.168.56.1
```

We can use Virtual machines for make more PCs in only one.

3. Clustering

3.1. MPP Architectures (Massively parallel processing)

- A task is divided into several subtask that run in parallel.
- Each processor uses its own operating system and main memory
- MPP communicate via **messaging interface**-
- Examples: decision systems, data warehouses

3.2. OpenMPI: Multiprocessor Programming (CPU)

OpenMP enables distributed programming on extension and **OpenMPI** is directed towards distributed programming.

- Acts on sending and receiving messages.
- Node organization is chosen by the program
- Ex: client-server, tree, p2p

3.2. MPI: Message Passing Interface

- Is **distributed**: multiple computers at the same time by pieces works on one task
- MPI work is divided into several processes
- Each node is driven by one (or more) processes
- Each MPI process is run by the same program but with different rank
- The ranks (id) divide the work between nodes

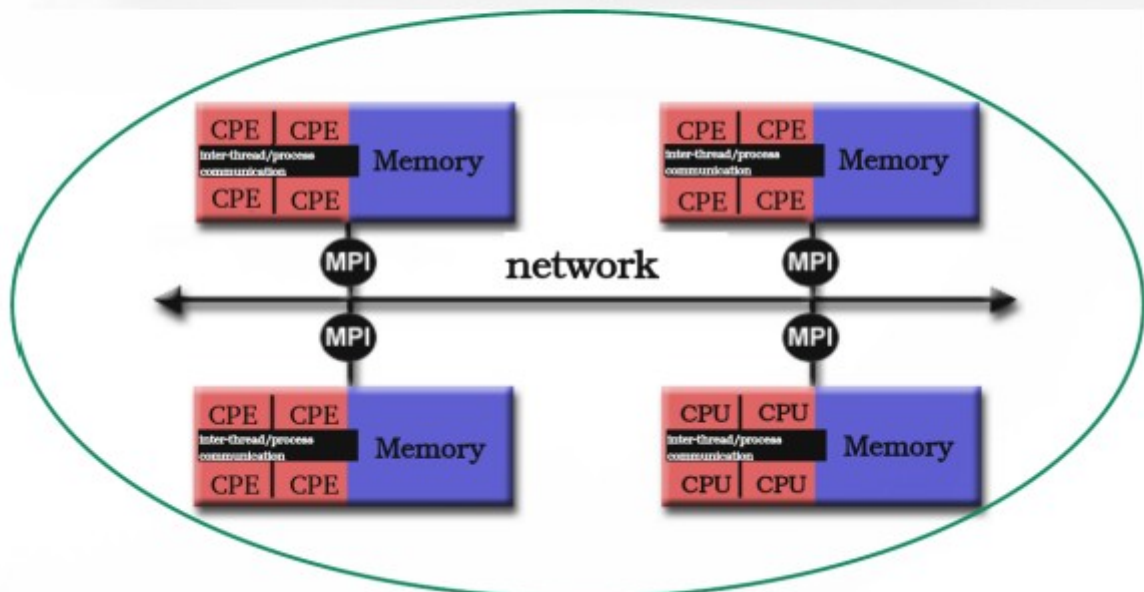
3.3. Comparison: Parallel vs distributed

- **Serially**: One computer processes program piece by piece
- **In parallel**: Multiple processors or computers at the same time by pieces works on one task
- **Distributed**: Multiple computers at the same time by pieces works on one task. More speedup and we solve problems of capacity.
- Comparison **shared** vs **distributed** memory:
 - shared has fastest communication (handled by OS/HW) → OpenMP (Multi-threading)
 - Distributed: Programmer takes care of communication, slower, with delay → **MPI**
- **Pitfalls** of parallelization:
 - Memory access patterns change

- Concurrent (I/O) request
- Additional overhead in the code

MPI → Parallelization with distributed memory

MPI – parallelization with distributed memory



MPI – *Message Passing Interface*

- MPI work distributed into several processes
- Each node is driven by one (or more) processes

4. Virtualization

Enables shared use of expensive machine resources, by defining:

- composed Virtual machines
- virtualization levels,
- VM architectures
- virtual networks
- virtual clusters
- virtual data repositories
- dynamic structures for clusters, networks and clouds.

Three requirements for VMM:

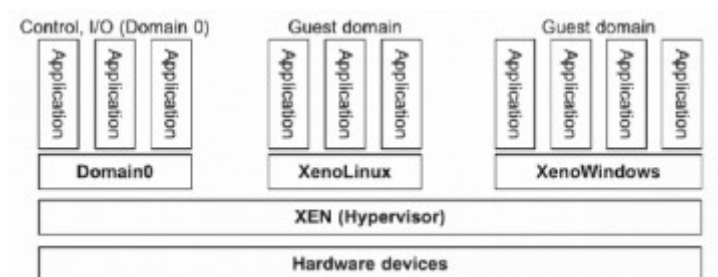
- Provides an environment equal
- Speed is slower
- VMM has full control

Implementation Levels:

- 1. ISA (instruction set architecture), with dynamic ISA translation
- 2. HAL (Hardware Abstraction Layer)
- 3. OS VE: Deployment of a Virtual Execution
- 4 & 5. Process

VM architectures:

- **hypervisor**: Between the hardware level and the OS



- **binary instruction translation**: Complete virtualization
- **para-virtualization**: Changes the OS kernel and **hypercalls**