UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD
DE INGENIERÍA

Licenciatura en Ciencias de la Computación

Arquitecturas Distribuidas

Unidad 3 - Parte B

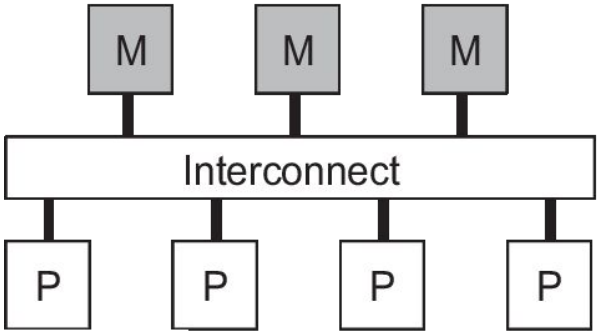
Software para arquitecturas distribuidas - Grid y Cloud Computing

Dr. Ing. Pablo Daniel Godoy (pablo.godoy@ingenieria.uncuyo.edu.ar)

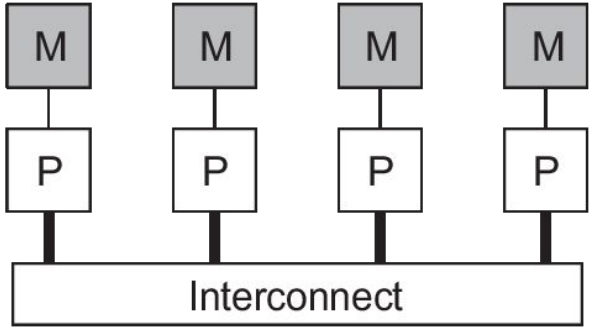
Temas

- **Software Intermedio**
  - **Nociones de MPI**
- **Computación Grid y Cloud**
  - **Arquitectura y diseño**
  - **Software intermedio. Virtualización**

Memoria compartida



Memoria Distribuida



Paralelismo en el software: Granularidad

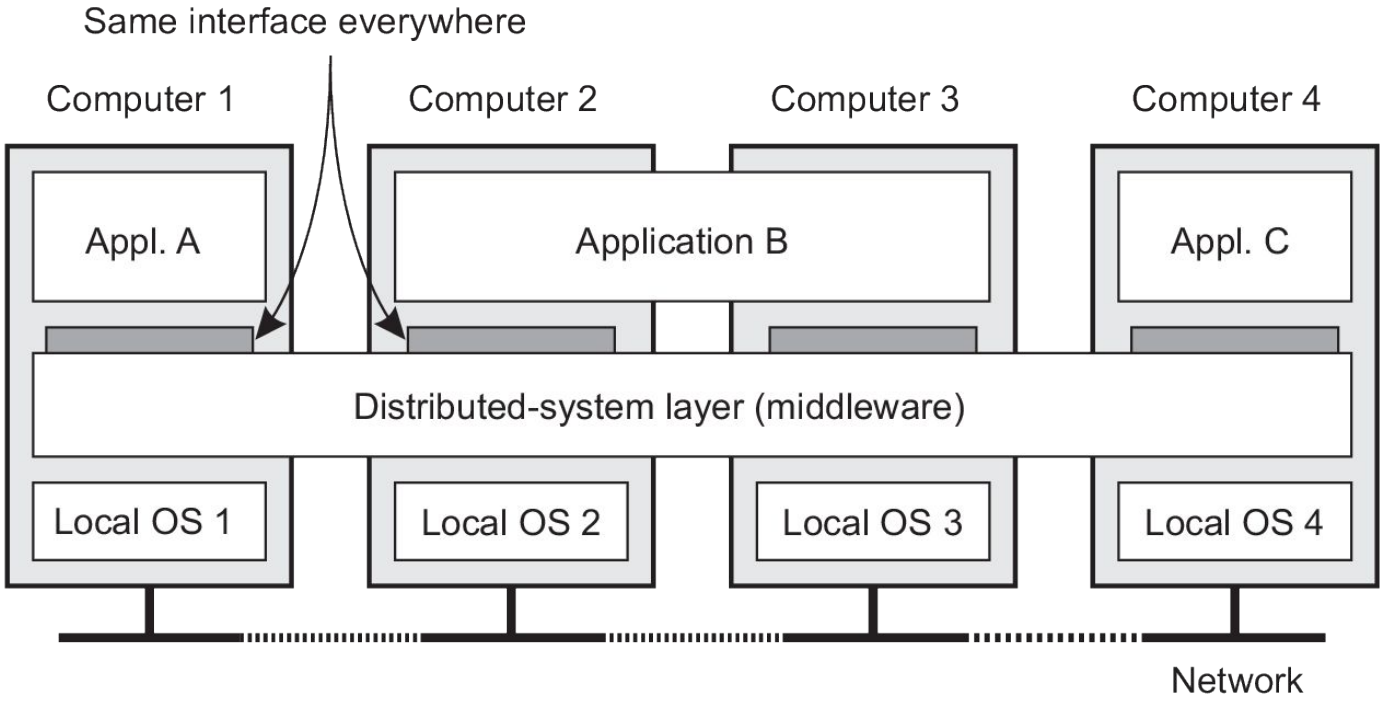
Nivel	Plataforma típica	Memoria	¿Quién decide?
Procesos independientes (programas diferentes)	Multinúcleo, multicomputadoras.	Distribuida	SO, program loader, planificador del SO
Procesos que colaboran para resolver un problema.	Multinúcleo, multicomputadoras.	Distribuida	Programador
Hilos que colaboran para resolver un problema.	Multinúcleo, proc. multihilo simultáneo.	Compartida	Programador, compilador.
Lazos no recursivos, datos, pequeños hilos independ.	Procesadores SIMD, vectoriales, GPU.	Compartida	Compilador (lazos), programador (SIMD, GPU).
Instrucción	Pipeline, superescalar, multihilo	Registros	Procesador, Compilador.

Paralelismo en el software: Granularidad

Nivel	comunicación	Herramienta
Procesos independientes (programas diferentes)	Mensajes	
Procesos que colaboran para resolver un problema.	Mensajes, RPC.	MPI, RPC.
Hilos que colaboran para resolver un problema.	Variables compartidas	CUDA, OpenCL, librerías multihilos
Lazos no recursivos, datos, pequeños hilos independ.	Variables compartidas en registros o memoria	CUDA, OpenCL, instrucc. SIMD, compilador.
Instrucción		

Tipos de sistemas distribuidos
Cluster computing

- Son sistemas de memoria distribuida fuertemente acoplados: cluster computing (HPC).



MPI (Message Passing Interface)

- **Especificación** para **funciones (primitivas)** y **mecanismos** para comunicación entre procesos.
  - Las funciones se implementan como **librerías** + middleware.
- Consorcio **MPI forum**: (https://www.mpi-forum.org/)
- Características:
  - Independencia del lenguaje.
  - Portabilidad del código.
  - La red subyacente no está definida en el estándar (frecuentemente es TCP/IP).
- ¿Qué hace?: **Comunicación entre procesos** a través de **mensajes** que van desde el espacio de direcciones de un proceso al espacio de direcciones del otro proceso.

Implementaciones de MPI

- Implementaciones:
  - **Open MPI**: Implementado por varias universidades de USA y Alemania. Gratuito. https://www.open-mpi.org/.
  - **MPICH**: Implementado por Mississippi State University (Argonne National Laboratory). Gratuito. https://www.mpich.org/
  - **HPE Cray MPI**: creada por Cray Inc., luego comprada por Hewlett Packard Enterprise. Implementación basada en MPICH (utilizado por Frontier y LUMI, N°1 y N°3 del top 500).
  - **Spectrum MPI**: Implementación paga de IBM (Utilizada por Summit y Sierra, Nº4 y Nº5 del Top 500 y varias otras). https://www.ibm.com/ar-es/marketplace/spectrum-mpi
    - La supercomputadora N°2 del TOP 500 (Fugaku, creda por Fujitsu) utiliza Fujitsu MPI (Based on OpenMPI).

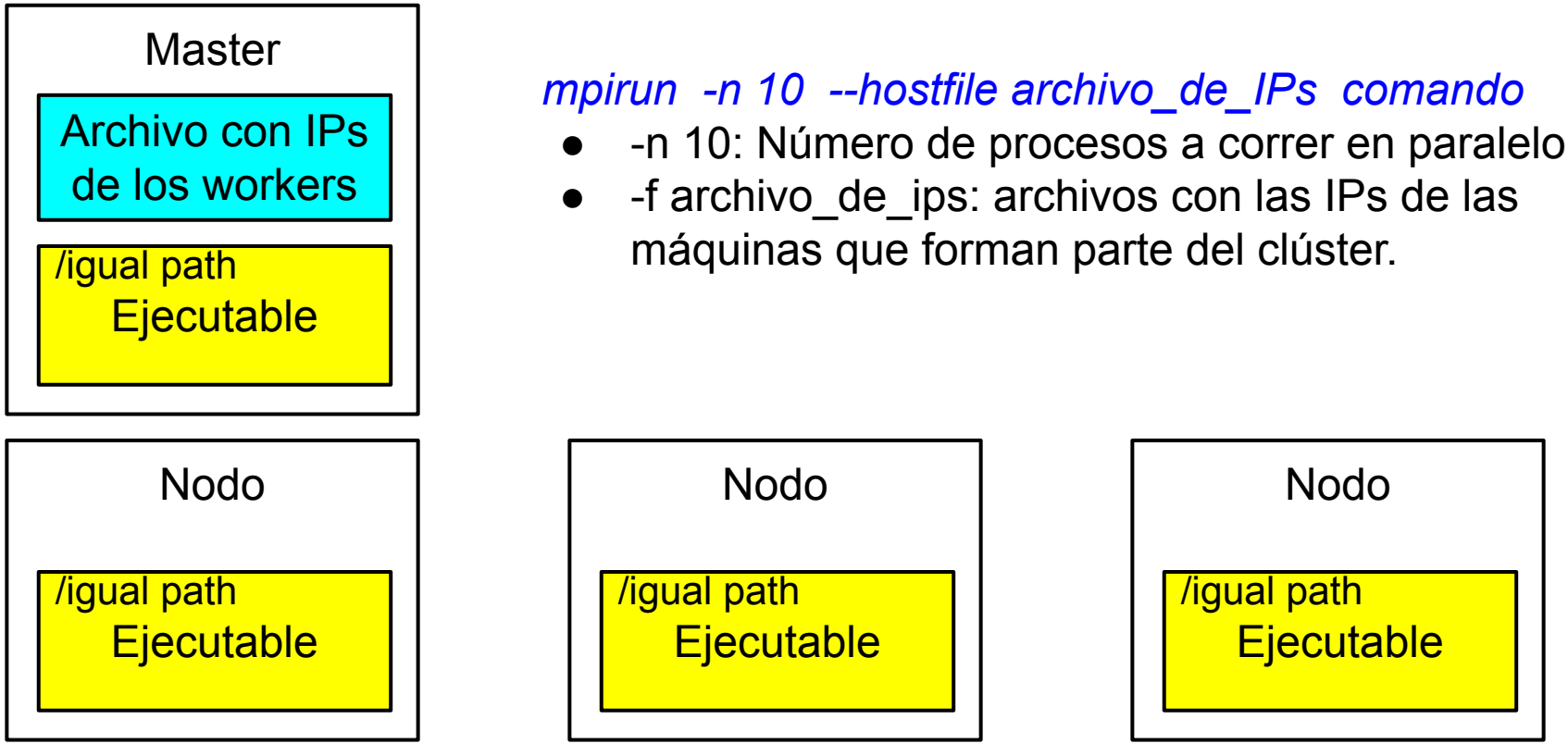
Implementación OpenMPI y MPICH

- Middleware que trabaja sobre SSH.
  - Es necesario tener ssh trabajando sin necesidad de ingresar usuario y contraseña en todas las máquinas del cluster:
    - Cada nodo y el master deben compartir una clave pública para poder comunicarse.
- Trabajar sobre C, C++ o Fortran.
  - Se debe tener instalado un compilador de esos lenguajes (g++ para Linux).
- Existen librerías o wrappers para otros lenguajes:
  - Por ejemplo: MPI4PY para Python.

Implementación OpenMPI y MPICH

- Ejecución de un programa paralelo:
  - Configurar archivo que contenga las IP de las máquinas que forman parte del Cluster en el nodo master.
  - Copiar los programas en todos los nodos en el mismo directorio.
  - Ejecutar los programas paralelizables que hayamos creado en el nodo master: *mpirun -n 10 --hostfile archivo_de_ips comando*
    - -n 10: Número de procesos a correr en paralelo.
    - -f archivo_de_ips: archivos con las IPs de las máquinas que forman parte del clúster.

Ejecución de un programa sobre Open MPI y MPICH



mpirun -n 10 --hostfile archivo_de_IPs comando

- -n 10: Número de procesos a correr en paralelo.
- -f archivo_de_ips: archivos con las IPs de las máquinas que forman parte del clúster.

Variables de un programa MPI

- **Comunicador:** Conjunto de procesos que resuelven una tarea comunicándose mediante MPI.
- **rank**: entero positivo que identifica a cada proceso.
- **size**: número total de procesos.

Estructura básica de un programa en C++ y Python

```cpp
#include<stdio.h>
#include <mpi.h>
using namespace std;
int main()
{
  int rank, size, length;
  char name[80];
  if(MPI_Init(NULL, NULL)!=MPI_SUCCESS){
    cout<<"Error iniciando MPI"<<endl;
    exit(1);
  }
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Get_processor_name(name,&length);

 /*Código del programa*/

  if(MPI_Finalize()!=MPI_SUCCESS){
    cout<<"Error finalizando MPI"<<endl;
    exit(1);
  }
}
```

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
name = MPI.Get_processor_name()

#Código del programa
```

Primitivas Send y Recv en C++

- **int MPI_Send(const void *buffer, int count, MPI_Datatype, int rank_dest, int tag, MPI_Comm comm)**
- **int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int rank_source, int tag, MPI_Comm comm, MPI_Status *status);**
  - buffer: buffer con datos a enviar (MPI_Send) o recibir (MPI_Recv).
  - Count: Número de datos a enviar.
  - MPI_Datatype (MPI_INT, MPI_BYTE).
  - rank_dest: proceso destino.
  - tag: etiqueta que permite distinguir distintos envíos.
  - rank_source: proceso fuente.
  - MPI_Comm: comunicador.
  - status: estructura donde se indica el estado de la recepción.
    - status.MPI_SOURCE
    - status.MPI_TAG
    - status.MPI_ERROR

MPI: Primitivas Send y Recv en C++

```
int numero;
if(rank==0){
   /*Otro código*/
   numero=88;
   MPI_Send(&numero,1,MPI_INT,1,12,MPI_COMM_WORLD);
}
if(rank==1){
   /*Otro código*/
   MPI_Status status;
   MPI_Recv(&numero, 1, MPI_INT, 0, 12, MPI_COMM_WORLD, &status);
   cout<<"Se recibió "<<numero<<" desde proceso " <<status.MPI_SOURCE<<endl;
}
```

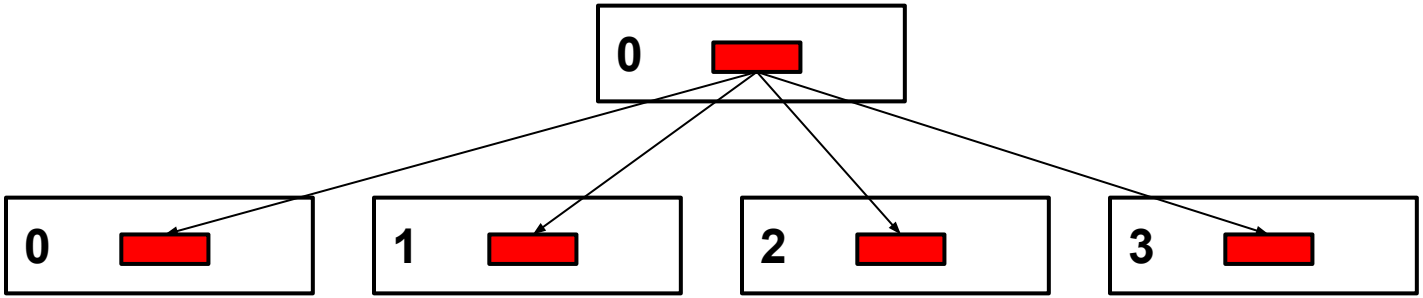
MPI: Primitivas Send y Recv en Python

comm.send(dato,dest=i,tag=11)
dato=comm.recv(source=i,tag=11)

```
if rank==0:
    dato=88
    comm.send(dato,dest=1,tag=11)
if rank==1:
    dato=comm.recv(source=0,tag=11)
    print("Se recibió " + str(dato))
```

MPI: Primitiva bcast (broadcast)

- **C++: MPI_Bcast(void* buffer, count, MPI_Datatype, root, MPI_Comm comm);**
  - buffer: buffer donde están los datos a ser enviados o donde se depositarán los datos recibidos.
  - dato_env= dato que será enviado por broadcast
  - root=0: ID del proceso que enviará el dato por broadcast
- **Python: dato_a_recibir = comm.bcast(dato_a_enviar, root=i)**



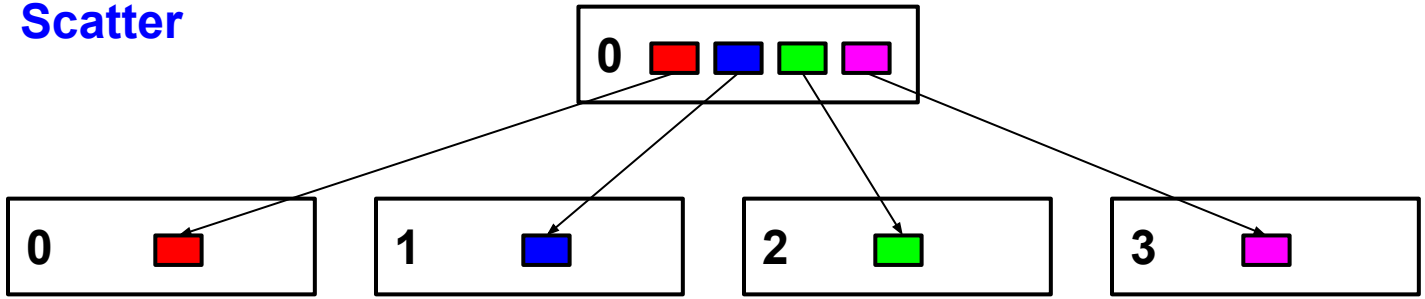
MPI: Primitiva bcast

```
long double numero;
if(rank==0){
  cout<<"Ingrese el número: "<<endl;
  cin>>numero;
}

if(MPI_Bcast(&numero, 1, MPI_LONG_DOUBLE, 0, MPI_COMM_WORLD)!=MPI_SUCCESS){
  cout<<"Error ejecutando MPI_Bcast"<<endl;
  exit(1);
}
```

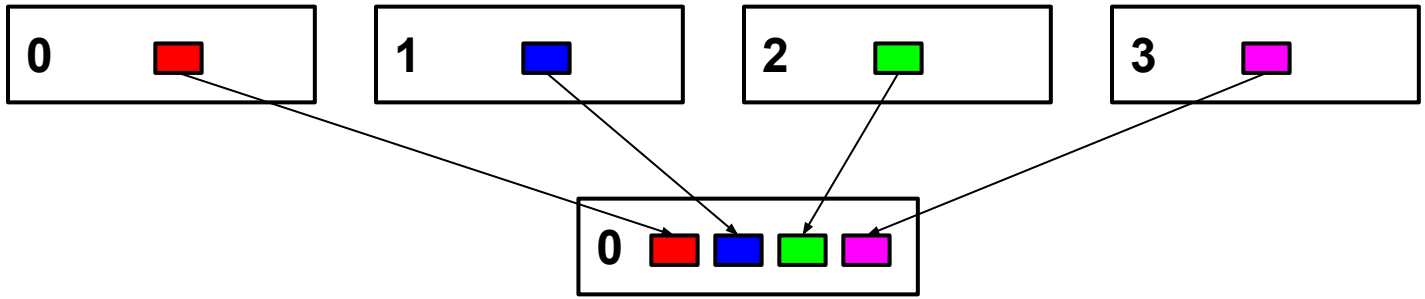
```
numero=0.0
if rank==0:
  numero=float(input("Ingrese el numero: "))

numero = comm.bcast(numero, root=0)
```

MPI: Primitivas scatter y gather (esparcir y reunir)

- **Scatter**



- **Gather**



MPI: Primitivas scatter y gather (esparcir y reunir)

C++

```
MPI_Scatter(void* sendbuf, sendcount, MPI_Datatype, void* recvbuf,
recvcount, MPI_Datatype, root, MPI_Comm);
MPI_Gather(void* sendbuf, sendcount, MPI_Datatype, void* recvbuf,
recvcount, MPI_Datatype, root, MPI_Comm);
```

- senfbuf: buffer con los datos a enviar.
- sendcount: cantidad de datos a enviar por proceso.
- Datatype: tipo de datos del buffer (MPI_INT, MPI_LONG).
- recvbuf: buffer de recepción.
- recvcount: cantidad de datos a recibir por proceso.
- root: proceso que esparce los datos.

Python

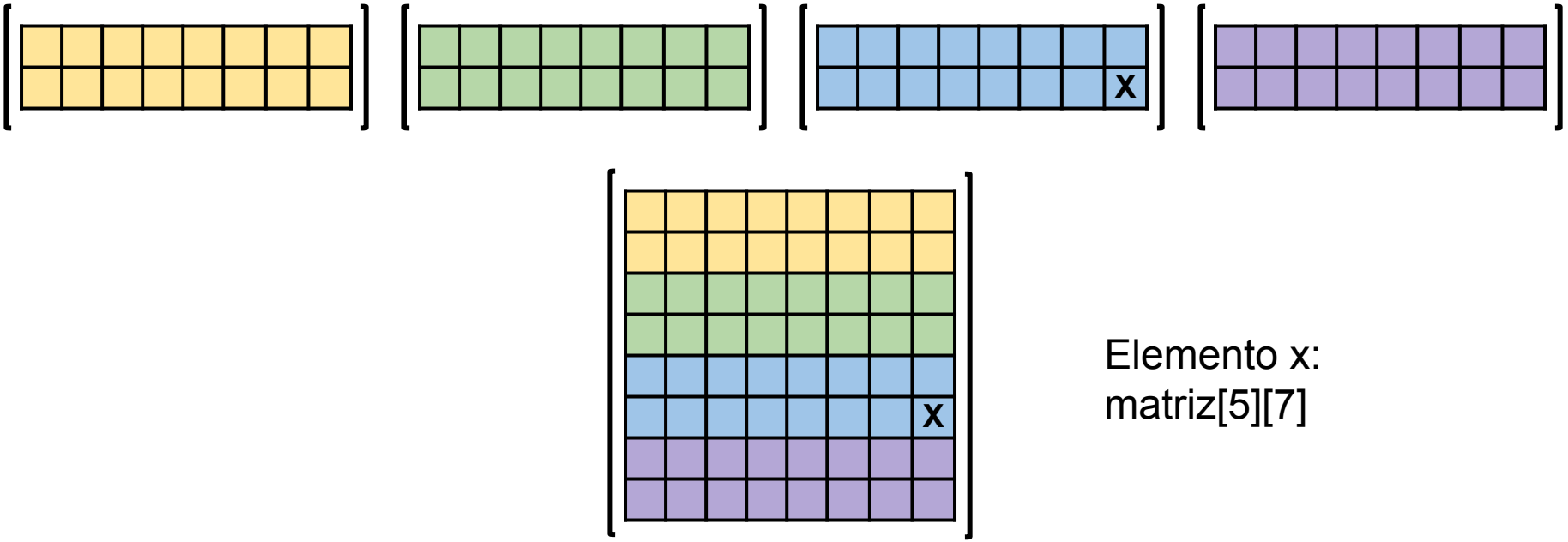
```
data = comm.scatter(arreglo, root=i)
arreglo = comm.gather(data,root=i)
```

MPI: Ejemplos de uso de gather con C++ y Python

```
long double *matriz_resultados=new long double[size];
if(MPI_Gather(&ln_por_proceso, 1, MPI_LONG_DOUBLE, &matriz_resultados[0],
   1,MPI_LONG_DOUBLE, 0, MPI_COMM_WORLD)!=MPI_SUCCESS){
  cout<<"Error ejecutando MPI_Gather"<<endl;
  exit(1);
}
```

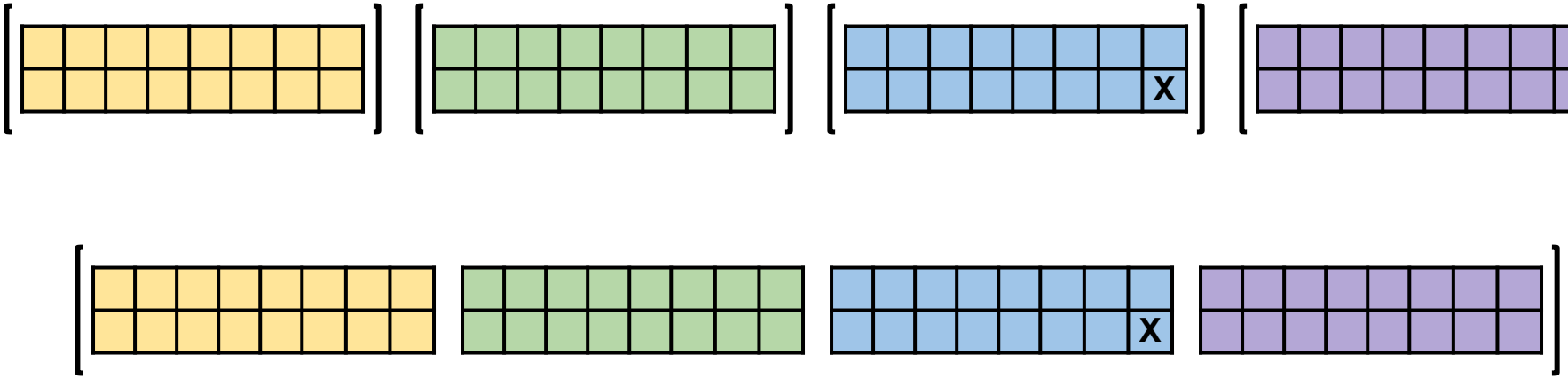
```
sumatoria=0.0
min=rank*cantidad_por_proceso
max=(rank+1)*cantidad_por_proceso
for i in range(min,max):
  sumatoria=sumatoria+(1.0/(2.0*float(i)+1.0))*(((numero-1.0)/-
(numero+1.0))**(2.0*float(i)+1.0))

resultados_list=comm.gather(sumatoria,root=0)
```

Función Gather en C++



Elemento x:
matriz[5][7]

Función Gather en Python



Elemento x:
Matriz[2][1,7] (suponiendo que usamos numpy)

MPI: Otras Funciones

- Primitivas bloqueantes (las vistas anteriormente son bloqueantes).
- Primitivas que permiten enviar datos de distinta longitud (MPI_Gatherv).
- Primitivas no bloqueantes (asíncronas):
  - Send and Receive bloqueante o síncrono:
    - La siguiente instrucción no se ejecuta hasta que send o receive terminaron su trabajo.
  - Send and Receive no bloqueante o asíncrono:
    - El programa sigue su ejecución mientras los datos se envían o reciben.
      - **Mayor performance.**
      - **Posibilidad de errores** (sobreescribir el buffer antes de que los datos terminen de enviarse).

Fuente: https://www.open-mpi.org/doc/v4.1/

Cloud Computing

- Pool de **recursos virtualizados** accesibles a través de una red o Internet.
  - Recursos:
    - Tiempo (en segundos) y poder (en GHz) de procesamiento.
    - Almacenamiento.
    - Servidores.
    - Máquinas virtuales
      - Características específicas
      - Sin características específicas.
    - Software.
    - Plataformas de ejecución de aplicaciones.
- Nace a partir de datacenters (Amazon, Google, etc.) que ponen a disposición sus recursos para que clientes puedan accederlos en base a un sistema de “pago por uso”.

Cloud Computing

- Características (definidas por NIST: National Institute of Standards Technology):
  - Autoservicio: Los clientes pueden proveerse unilateralmente (automáticamente) de servicios.
  - Servicio bajo demanda: Cada cliente es provisto con las capacidades computacionales que necesite.
  - Acceso a través de la red disponible por el cliente.
  - Acceso ubicuo: Podemos acceder a nuestra máquina virtual o aplicación en cualquier parte del mundo con acceso a Internet.
  - Modelo multi-cliente: Los proveedores ofrecen servicios a múltiples clientes.
  - Los recursos pueden ser provistos o liberados en forma rápida y dinámica.
  - Los servicios son medidos.

Cloud Computing

- Arquitectura de 4 capas:
  - Hardware.
  - Infraestructura: Capa de middleware de virtualización. Provee recursos virtualizados, como procesamiento, almacenamiento, etc. (por ejemplo, una máquina virtual).
    - Provee el servicio de Infrastructure as a Service (IaaS)
  - Plataforma: Plataforma para ejecutar aplicaciones del cliente (similar a una máquina con un sistema operativo sobre la cual las aplicaciones pueden correr). Provee alta abstracción de los recursos.
    - Provee el servicio de Platform as a Service (PaaS)
  - Aplicación: Software provisto como servicio.
    - Provee el servicio de Software as a Service (SaaS).

Cloud Computing

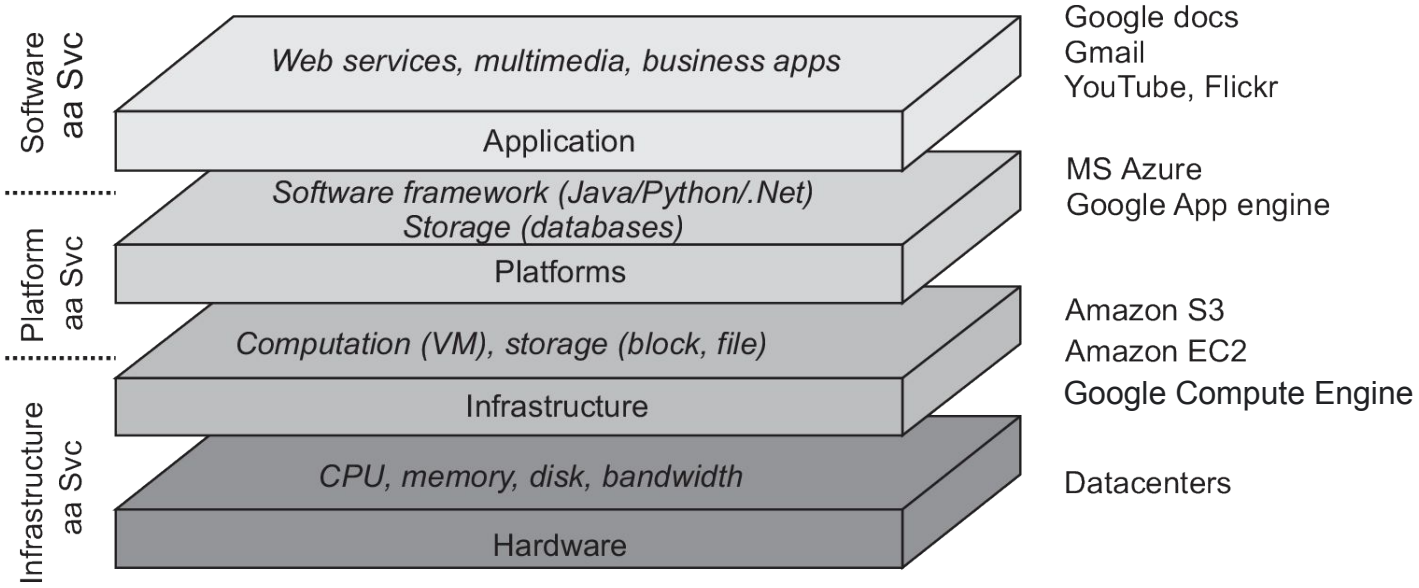


Figura obtenida de van Steen and Tanenbaum, "Distributed Systems", Third edition, v 3.03 (2020), pag 30.

Cloud Computing

Modelos básicos de servicios y arquitectura:

- **Software as a Service (SaaS)**: El cliente puede ejecutar las **aplicaciones del proveedor** sobre la infraestructura Cloud.
- **Platform as a Service (PaaS)**: El cliente puede ejecutar las **aplicaciones del cliente** sobre la infraestructura Cloud (las librerías, servicios, lenguajes, etc. deben ser soportados por el proveedor).
- **Infrastructure as a Service (IaaS)**: El proveedor ofrece recursos computacionales virtualizados (poder de procesamiento, almacenamiento, redes, otros dispositivos específicos, etc.). El cliente puede instalar sistemas operativos y aplicaciones que desee.

Cloud Computing

- Ventajas de Cloud Computing:
  - **Robustez**: Los proveedores de servicios se encargan de la instalación, mantenimiento y actualización de software y hardware, respaldo y seguridad de los datos y aplicaciones almacenadas y calidad de servicio.
  - **Disponibilidad**: El proveedor asegura la disponibilidad deseada por el cliente (por ejemplo: 24 hs todo el año), utilizando infraestructura de respaldo.
    - Tolerancia a fallos: mecanismos de failover, recursos distribuidos en diferentes partes del mundo, etc.
  - **Escalabilidad y elasticidad**: Los recursos pueden escalar para adaptarse a las demandas del cliente de manera automática (para el cliente los recursos parecen ilimitados)

Cloud Computing

Otros servicios:

- DaaS (Database as a Service)
- FaaS (Framework as a Service)
- DaaS (Desktop as a Service)
- Virtual Private Cloud: Red privada virtual en la nube.
- Device Farm: Dispositivos móviles reales para probar código en ellos.

Frameworks para Cloud Privados

- Eucalyptus (compatible con AWS)
- OpenNebula

Cloud Computing

- Modelos de despliegue:
  - Clouds privados: La infraestructura es provista por una organización privada para ser usada por integrantes de dicha organización.
  - Clouds comunitarios: La infraestructura es montada por un grupo de organizaciones para ser empleadas por los miembros de dichas organizaciones.
  - Clouds públicos: La infraestructura es provista para ser utilizada por el público en general.
  - Cloud híbridos: El Cloud es una combinación de varios modelos de despliegue.

Ejemplos de Cloud Computing

Proveedores de servicios de Cloud Computing:

- Google Cloud Platform: (https://cloud.google.com/):
  - PaaS: App Engine (algunos recursos gratis).
  - IaaS: Compute Engine (algunos recursos gratis).
- Amazon Web Services:
  - PaaS: AWS Lambda (algunos recursos gratis).
  - IaaS:
    - EC2 (Elastic Compute Cloud).
    - Amazon S3 (Simple Storage Service)
- Windows Azure:
  - PaaS: App Service.
  - IaaS: virtual machines.

Ejemplos de Cloud Computing

IaaS Cloud computing: computacion

- Ejemplo: Amazon EC2 (Elastic Compute Cloud).
- Ejemplo: Google Compute Engine.

- Se proveen varias imágenes preconfiguradas de SO con varias herramientas (Amazon les llama AMI (Amazon Machine Image), Google Instancias).
- Los tipos de imágenes se diferencian en:
  - CPU, número y tipos de núcleos, y tipo de GPU.
  - Memoria.
  - Almacenamiento no volátil.
  - Plataforma (32 bits o 64 bits).
  - Networking: Capacidades de red (ancho de banda).
  - Aplicaciones y herramientas preinstaladas (Base de datos, servidores web,)

https://cloud.google.com/compute/docs/machine-types

Ejemplos de Cloud Computing
IaaS Cloud computing: almacenamiento

- Ejemplo: Elastic Block Store (Amazon EBS).
  - Puede ser utilizado como un disco duro (virtual).

- Google Cloud storage: Almacenamiento de objetos.
- Persistent disk: Almacenamiento como un disco duro (virtual).

- Ejemplo de instancia: Instancia e2-micro de Google Cloud Platform (instancia gratuita sin necesidad de indicar tarjeta de crédito).
  - Procesador Intel Xeon 2.20 GHz 64 bits de 4 núcleos (dos núcleos físicos, 2 hilos por núcleo), 15 GB RAM, 60 GB (características con mucha variación).
  - Subir archivos (almacenamiento persistente).
  - Interfaz a través de consola de comandos Linux.
  - En ejecución solo cuando está la interfaz web (cliente) corriendo.

Ejemplos de máquinas virtuales de Google Cloud Computing

E2-micro: 2 núcleos; 8 GB de memoria; 10 GB disco (7.82 USD por mes).
C2-standard-60: 60 núcleos; 240 GB de memoria; 128 discos; ancho de banda de salida: 32 Gbps (1829.62 USD por mes).

Ejemplos de máquinas virtuales (IaaS) de Google

Máquinas de uso general: https://cloud.google.com/compute/docs/general-purpose-machines
Máquinas optimizada para procesamiento: https://cloud.google.com/compute/docs/compute-optimized-machines
Máquinas con optimización de memoria: https://cloud.google.com/compute/docs/memory-optimized-machines
Con optimización de acelerador: https://cloud.google.com/compute/docs/accelerator-optimized-machines
Tablas de precios: https://cloud.google.com/compute/vm-instance-pricing
Calculadora de precios: https://cloud.google.com/products/calculator
Productos gratuitos: https://cloud.google.com/free/

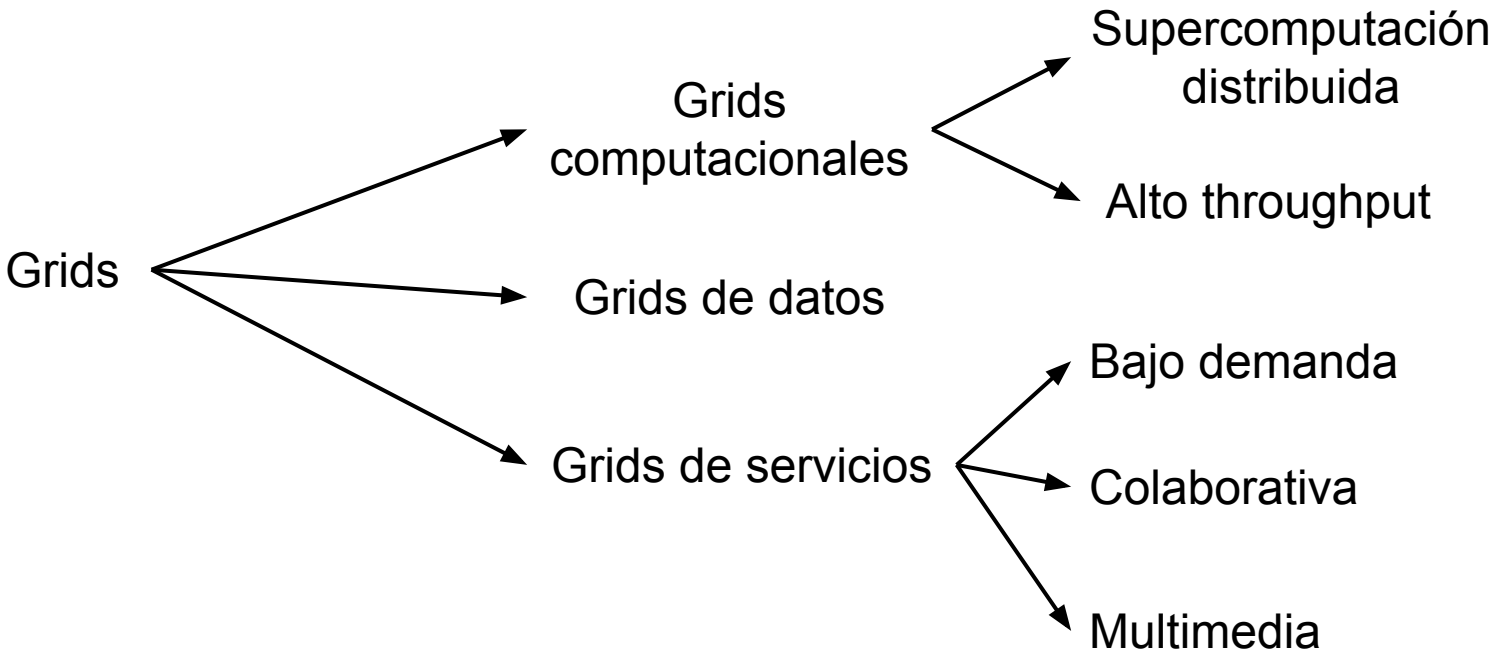
Grid Computing

- Infraestructura para compartir (y agregar) recursos geográficos distribuidos y diversos (heterogéneos) que son propiedad de diferentes organizaciones y compartidos por las mismas.
  - Supercomputación.
  - Sistemas de almacenamiento.
  - Fuentes de datos.
  - Dispositivos especializados.
- Objetivos:
  - Resolver problemas de gran escala que requieren elevado poder de procesamiento o almacenamiento.

Grid Computing

- **Middleware de gestión** de recursos disponibles en la grid (RMS: Resource Management System):
  - Scheduling de procesadores disponibles.
  - Ancho de banda de redes.
  - Almacenamiento.
- Federación de Grids: conjunto de grids interconectadas, cada una con su middleware.
- Jobs (tarea): trabajo a realizar sobre la grid, incluye una petición de recursos. El middleware puede emplear diferentes políticas para satisfacer las necesidades de recursos (mejor esfuerzo, calidad de servicio, etc.)

Grid Computing



Clasificación tomada de: Krauter, Buyya, Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing"

Ejemplos de Grids

- SETI@home: Grid construida para buscar patrones de vida extraterrestre. Computadoras hogareñas comparten voluntariamente recursos computacionales.
- EDG (European Data Grid): Grid del CERN (Centro Europeo Investigación Nuclear).

Middleware

- EGEE: para grandes proyectos
- Globus: Solutions para proyectos medios
- Grid Engine: para pequeñas implantaciones.

Bibliografía:

- William Stallings, "Computer Organization and Architecture”, 10º edición, editorial Pearson, año 2016.
- Tanenbaum and Bos, “Modern Operating Systems”, 4° edición, editorial Pearson, año 2015.
- Kai Hwang, “Advanced Computer Architecture, Parallelism, Scalability, Programmability”, 2º edición, editorial Mc Graw-Hill, año 2011.
- Hesham and Mostafa, “Advanced Computer Architecture and Parallel Processing”, 1° edición, editorial Wiley, año 2005.
- ARM, "ARM Cortex-A Series Programmer’s Guide for ARMv8-A", Version 1.0, año 2015
- MPICH Documentation (https://www.mpich.org/documentation/guides/)
- mpi4py Tutorial (https://mpi4py.readthedocs.io/en/stable/tutorial.html)

Bibliografía:

- Buyya, Yeo, Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities", 2008.
- NITS (National Institute of Standards Technology), “The NIST Definition of Cloud Computing”.
- Bhaskar Prasad Rimal, Eunmi Choi, Ian Lumb, "A Taxonomy and Survey of Cloud Computing Systems".