COMPUTACIÓN DE ALTA PERFORMANCE

Curso 2025

Sergio Nesmachnow (sergion@fing.edu.uy)

Centro de Cálculo

Tema 9: Computación grid

Grid computing

1. Introducción
2. Tipos de sistemas Grid
3. Middleware para sistemas Grid
4. Ejemplo de aplicación: Digi-Clima Grid en GISELA y Ourgrid

9.1: Introducción

Introducción

¿Qué es un Grid?

"… a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements".

– Buyya/Venugopal

R. Buyya and S. Venugopal, Market Oriented Computing and Global Grids: An Introduction, Market Oriented Grid and Utility Computing, Wiley Press, USA, 2009.

Introducción

- Desarrollado en ámbitos científicos en la década de 2000, el término computación grid refiere a una infraestructura geográficamente distribuida que permite la integración y el uso colectivo de recursos de cómputo, de almacenamiento, o recursos especializados.
- Universidades, laboratorios de investigación o empresas se asocian para formar Grids.
- Los recursos son administrados por diferentes instituciones y no están sujetos a un control centralizado.
- Los recursos compartidos pueden ser ordenadores (estaciones de trabajo, supercomputadoras, PDA, portátiles, móviles, etc.), software, datos, o intrumentos especiales (radio telescopios, equipamiento nuclear, etc.).

¿Por qué utilizar grid computing?

- Porque el poder de cómputo requerido por muchas aplicaciones ha aumentado significativamente.
- Porque la necesidad de espacio de almacenamiento también ha crecido significativamente y requiere de sistemas de almacenamiento masivos, complejos y distribuidos.
- Para facilitar la cooperación entre usuarios.
- Para utilizar y compartir algún instrumento o dispositivo que no es fácilmente accesible.

Características y desafíos

- Heterogeneidad
  - Un Grid puede contener tanto recursos de hardware como recursos de software: archivos, componentes de software, instrumentos científicos, dispositivos portátiles, PC, supercomputadoras, etc.
- Transparencia
  - Los recursos de un Grid podrán estar geográficamente distribuidos. Su distribución debe permanecer oculta al usuario. El Grid debe ser percibido como una única supercomputadora.
- Tolerancia a fallas
  - La falla total o parcial de un componente no debe resultar en la falla total del sistema. Esto implica nunca tener un único punto central de control.

Características y desafíos

- Escalabilidad
  - Debe soportar desde unos pocos recursos hasta millones.
- Administradores múltiples
  - Cada organización podrá establecer políticas administrativas y de seguridad diferentes para los recursos propios.
- Concurrencia
  - Debe permitirse el acceso compartido a recursos.
- Acceso confiable
  - Debe asegurar la prestación de servicios sujeto a requerimientos de calidad de servicio (QoS) previamente establecidos.

Características y desafíos

- Acceso ubicuo
  - Debe brindar acceso a los recursos disponibles del Grid adaptándose a un ambiente dinámico por naturaleza en donde los fallos en los recursos es la regla.
- Acceso consistente
  - Debe prestar servicios, protocolos, e interfaces estandarizadas de forma de esconder la heterogeneidad subyacente. Sin estos estándares el desarrollo de aplicaciones y su uso ubicuo no sería posible.
- Seguridad
  - El sistema debe ser usado solamente de la manera esperada.

Consideraciones

- Aspectos a considerar al utilizar un sistema Grid:
  1. Privacidad de la información.
  2. Necesidad de *gridificar* aplicaciones (migrar/desarrollar aplicaciones para el Grid).
  3. Comunicación lenta y no uniforme.
     - Adecuado para aplicaciones cuyo cómputo paralelo puede realizarse de forma independiente, sin la necesidad de comunicar resultados intermedios.
  4. Gran heterogeneidad de recursos.

9.2: Tipos de sistemas grid

Tipos de sistemas grid

- Los sistemas Grids pueden clasificarse en diferentes tipos:

  1. Peer-to-Peer (o voluntarios)
  2. Remote Procedure Call (RPC)
  3. High-Throughput Computing (HTC)
  4. High-Performance Computing (HPC)

Grids peer-to-peer

- Los grids computacionales peer-to-peer, también llamados grids voluntarios, son un tipo de sistema de computación distribuida en el que cada usuario dona algunos de sus recursos de computo.
- Se aprovechan los ciclos de CPU ociosos (CPU-scavenging) de los recursos donados para ejecutar trabajos en el Grid. Con esta técnica se aprovechan ciclos de computo que en otro caso se perderían.
- Los recursos computacionales del Grid no cuentan con puntos centrales de control (*"pure P2P"*).
- Brindan soporte para aplicaciones de tipo Bag-of-Tasks (BoT). Los usuarios suelen participar porque tienen intereses comunes.
- Uno de los mayores desafíos de los grid peer-to-peer consiste en descubrir recursos y servicios en un número potencialmente enorme de peers globalmente distribuidos.
- Middlewares: The Berkeley Open Infrastructure for Network Computing (BOINC), XtremWeb, Xgrid, Grid MP.

Grids peer-to-peer

- Einstein@Home

  Búsqueda de fuentes contínuas de ondas gravitacionales.

  https://einsteinathome.org/

- Great Internet Mersenne Prime Search (GIMPS)

  Proyecto creado en 1996; a comienzos de 2016 encontró el 49.°vo número primo de Mersenne conocido (que además es el número primo más grande conocido): (22.338.618 cifras)

  http://www.mersenne.org/

- Folding@home

  Proyecto que realiza simulaciones de dinámica molecular para el diseño de drogas y el tratamiento de enfermedades.

  http://folding.stanford.edu/

BOINC middleware

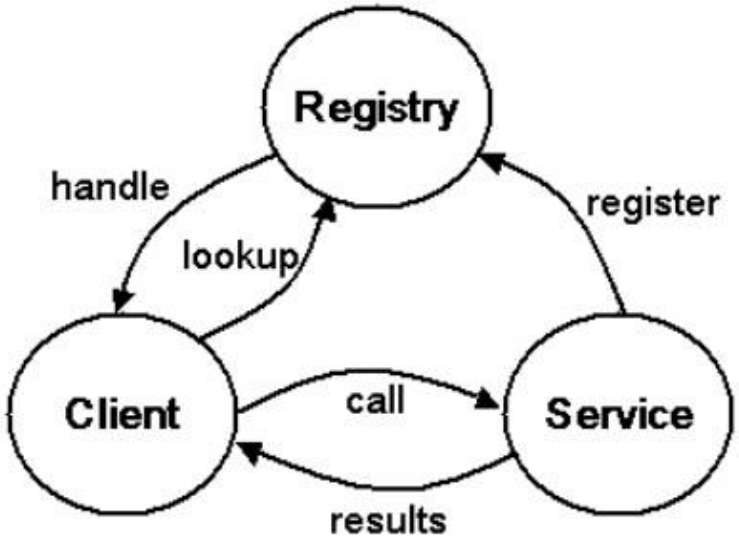
- Berkeley Open Infrastructure for Network Computing (BOINC) fue originalmente desarrollado para dar soporte al proyecto SETI@home.
- BOINC sigue una arquitectura cliente-servidor. El cliente y el servidor se comunican utilizando mecanismos de RPC para distribuir y procesar *unidades de trabajo* y retornar su resultado.
- Una *unidad de trabajo* representa la entrada para computar: una aplicación, un conjunto de archivos de entrada, un conjunto de argumentos de ejecución, y variables de entorno.
- Además contiene parámetros como requerimientos de cómputo, memoria, y almacenamiento, y un *soft deadline* para su finalización
- El resultado del cómputo consiste en una referencia a la *unidad de trabajo* y un conjunto de archivos resultado.

BOINC middleware

- Los Grids públicos deben lidiar con resultados computacionales erróneos, ya sea por hardware funcionando incorrectamente (típicamente debido a overclocking) o debido a participantes mal intencionados.
- BOINC brinda soporte para computación redundante, un mecanismo para identificar resultados erróneos.
- Puede especificarse que cada *unidad de trabajo* sea procesada N veces. Los N resultados son comparados y en caso de que no se llegue a un consenso vuelve a repetirse el proceso.
- BOINC también brinda mecanismos de "*failure and backoff*" que permite servir a millones de clientes utilizando un servidor modesto. Si todos los clientes intentan conectarse al servidor resultaría desastroso.
- En caso de falla en la conexión el tiempo de reintento se incrementa exponencialmente para evitar sobrecargar al servidor.

Grids RPC

- Los sistemas GridRPC están basados en el modelo cliente-servidor de invocación remota de procedimientos.
- GridRPC API fue propuesta y estandarizada por Open Grid Forum.
- En el estándar GridRPC el usuario desarrollador debe implementar dos aplicaciones: un cliente y un servidor.
- La aplicación servidor será registrada en los servicios del Grid y esperará por invocaciones de procesamiento.
- La aplicación cliente será ejecutada por el usuario y se contactará con la aplicación servidor para solicitarle un resultado.
- MPI es más poderoso, pero RPC es más simple
- Middlewares: Ninf-G, DIET, etc.



Grids RPC

- GridRPC ofrece mecanismos para realizar invocaciones remotas de forma sincrónica y asincrónica.
- Tipos de datos principales de GridRPC API
  - **grpc_function_handle_t** representa una referencia remota a una función que se encuentra en un servidor dado. Una vez instanciada puede utilizarse repetidas veces.
  - **grpc_session_t** utilizada para identificar una invocación remota asíncrona.
- Funciones principales de GridRPC API
  - **grpc_initialize() y grpc_finalize()** son funciones similares a las funciones de inicialización de MPI.
  - **grpc_function_handle_init() y grpc_function_handle_destruct()** son utilizadas para inicializar y destruir una refrencia a una función remota.

Grids RPC

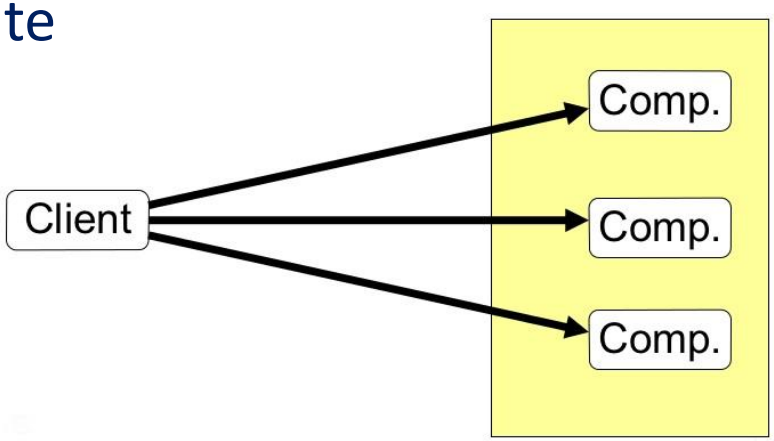
- Funciones principales del API de GridRPC:
  - **grpc_call()** y **grpc_call_async()** para realizar invocaciones remotas de forma sincrónica y asincrónica, respectivamente.
  - Se proveen un conjunto de funciones para manejar las invocaciones remotas asincrónicas: **grpc_probe()** para consultar por la finalización de una invocación; **grpc_probe_or()** para consultar por la finalización de alguna de las invocaciones previas; **grpc_cancel()** para cancelar una invocación; **grpc_wait()** para bloquear la ejecución hasta terminar una invocación; **grpc_wait_and()** para bloquear la ejecución hasta terminar un conjunto de invocaciónes; **grpc_wait_or()** para bloquear la ejecución hasta que finalize alguna invocación de un conjunto de invocaciones; **grpc_wait_all()** para bloquear la ejecución hasta que terminen todas las invocaciones pendientes; y **grpc_wait_any()** para bloquear la ejecución hasta que termina alguna de las invocaciones previas.

Grids RPC

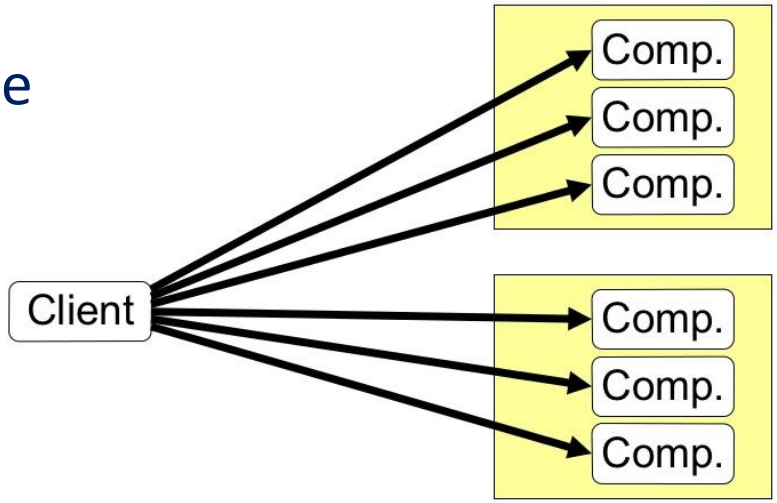
```
/* initialize function handle */
for (i=0; i<NUM_HOSTS; i++)
    grpc_function_handle_init(&handles[i],…);

/* parallel invocation */
for (i=0; i<NUM_HOSTS; i++)
    gprc_call_async(&handles[i],…);

/* wait for all the calls */
grpc_wait_all();
```

Grids RPC

Arquitectura single-site

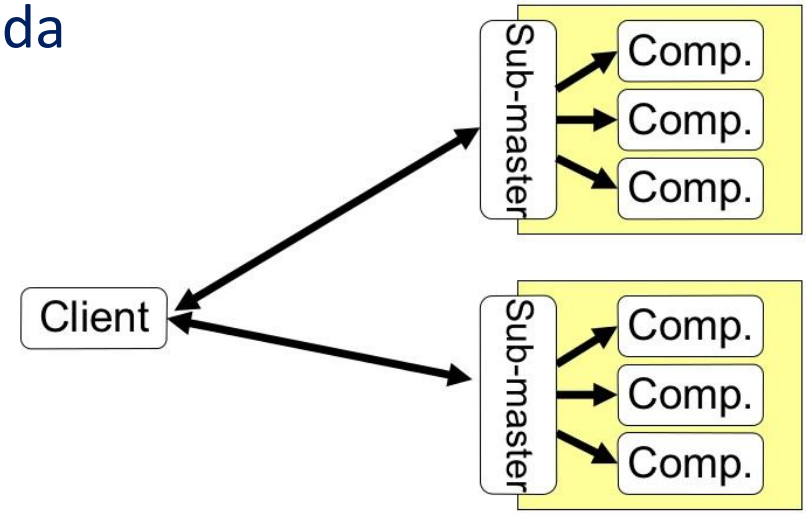


Arquitectura multi-site

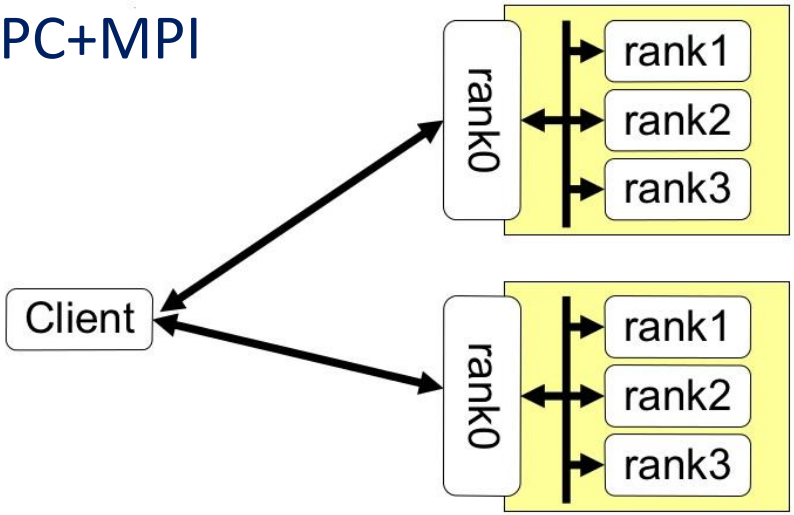


Grids RPC

Arquitectura en cascada



Arquitectura híbrida RPC+MPI



Grids HTC

- La computación de alto desempeño (HPC) intenta maximizar la cantidad de trabajos ejecutados en un período pequeño de tiempo.
- Frecuentemente, ingenieros y científicos, enfrentan problemas que requieren semanas o meses de cómputo.
- El tipo de computación *high-throughput computing* (HTC) ataca este tipo de problemas e intenta maximizar la cantidad de problemas que pueden resolverse en un período largo de tiempo.
- Un sistema HTC se encuentra optimizado para maximizar la utilización de los recursos de cómputo.
- Sus características más importantes son la tolerancia a fallas y la robustez.

HTCondor middleware

- HTCondor es un middleware para High-Throughput Computing (HTC).
- Especializado en tareas paralelas distribuidas de «grano grueso» y computacionalmente intensivas.
- Ofrece servicios de remote system calls, checkpointing y migración de procesos para algunos tipos de trabajos.
- Remote system calls permite a los procesos invocar operaciones de I/O de forma remota, p.ej. para acceder a un archivo remoto sin necesidad de contar con un filesystem compartido como NFS.
- Checkpointing almacena todo el estado del proceso en ejecución. En caso de falla de una nodo de cómputo, el estado guardado permite migrar el trabajo a otro nodo y continuar su ejecución desde el punto del último checkpoint.
- Checkpointing y migración también permiten realizar una planificación preemptiva, suspendiendo y reanudando trabajos según sea necesario.

HTCondor middleware

- Cuenta con diferentes universos para ejecutar trabajos.
- *Standard Universe* permite la ejecución de trabajos seriales y provee a estos trabajos con una serie de servicios muy deseables
  - Checkpointing y restart transparente de procesos
  - Migración transparente de procesos
  - Remote System Calls
- MPI Universe permite la ejecución de aplicaciones MPI
- PVM Universe permite la ejecución de aplicaciones PVM

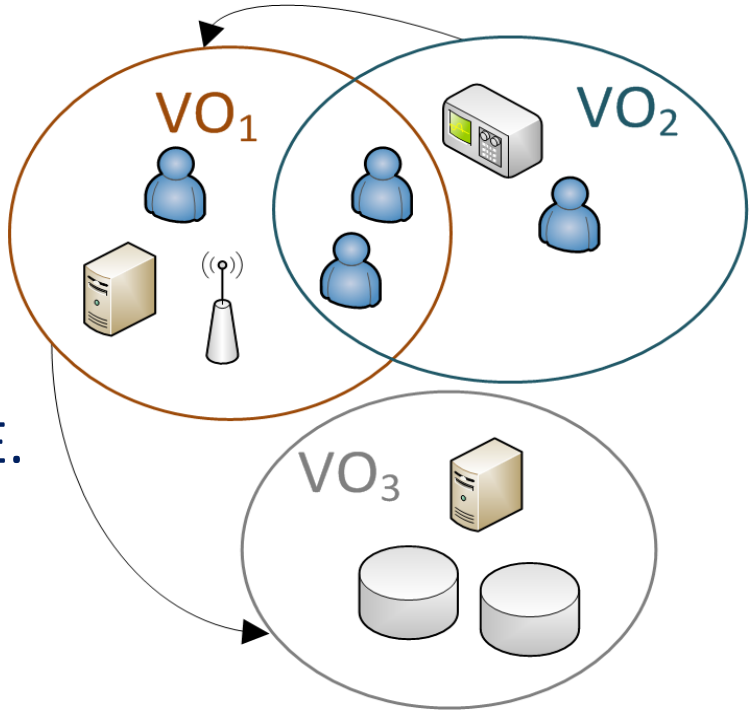
- Limitaciones de *Standard Universe*
  - No se permiten trabajos multi-proceso (p.ej. fork) ni multi-hilo.
  - IPC no esta permitido (p.ej. pipes, semáforos, o memoria compartida)
  - Comunicación con sockets debe ser breve (no es posible hacer checkpoints mientras se encuentren sockets en uso)

HTCondor middleware

- Puede utilizarse para el manejo combinado de recursos dedicados y recursos no dedicados (CPU-scavenging).
- CPU-scavenging:
  - Cuando una estación de trabajo ha estado ociosa durante una cantidad de tiempo, le es asignado un trabajo de la cola de ejecución.
  - Cuando se detecta movimiento en el mouse, se presiona una tecla o se detecta un alto uso de CPU por parte de una aplicación, el trabajo es detenido y enviado nuevamente a la cola de ejecución.

Grids HPC

- Compuesta por la agregación de recursos computacionales de institutos y universidades distribuidos geográficamente.
- Estos recursos pueden ser: supercomputadores, clusters de computadores, hardware científico especializado, etc.
- Se encuentran organizados en Virtual Organizations (VO)
  - Una VO es un conjunto dinámico de personas o recursos que trabajan de forma coordinada para alcanzar una meta común.
  - Las VO son usadas para definir políticas de seguridad entre los miembros del grid.



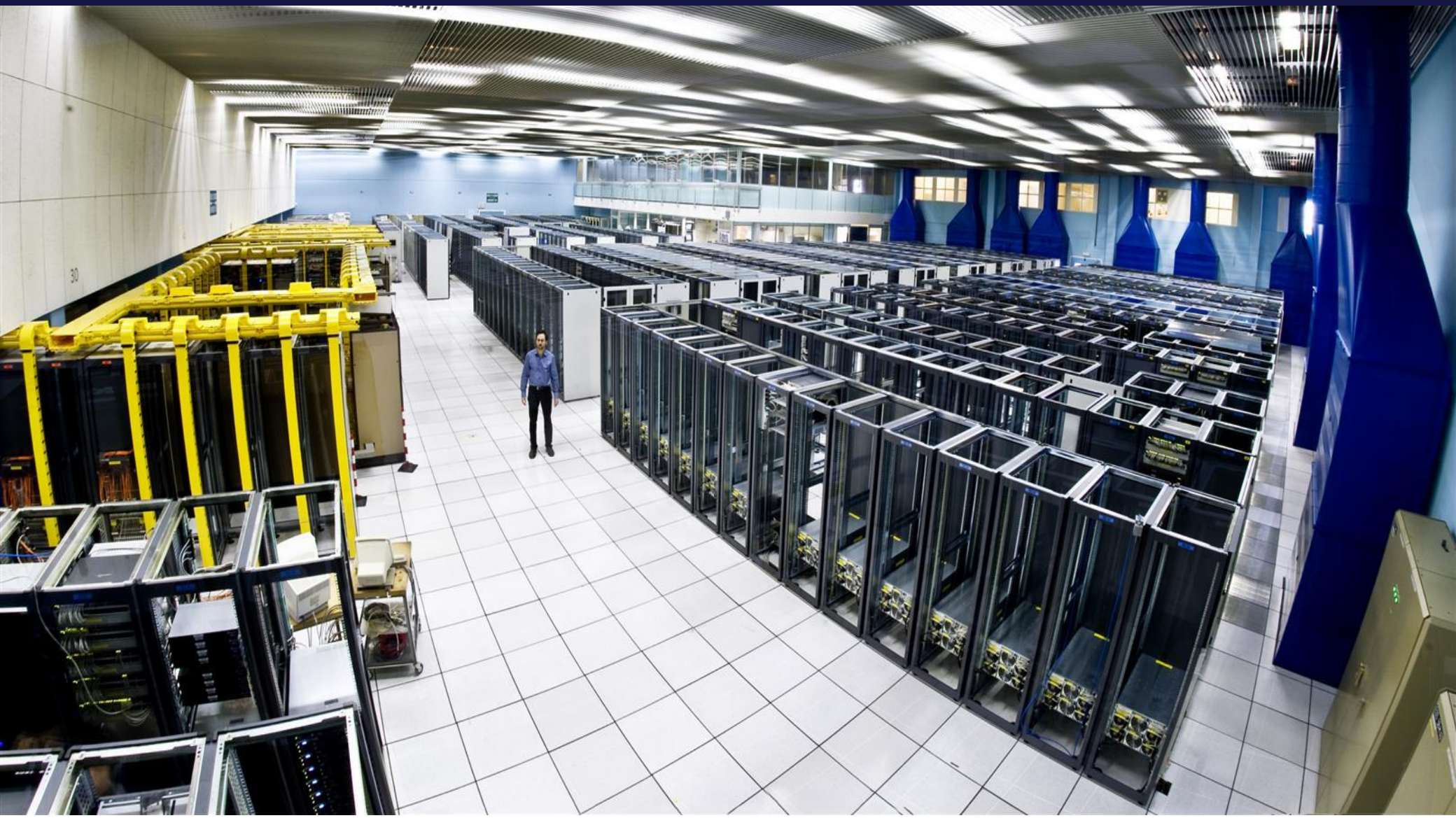
- Middlewares: Globus Toolkit, gLite, UNICORE.

LHC Computing Grid

- Worldwide LHC Computing Grid (WLCG) fue creado en 2002 para dar soporte a los experimentos realizados en el CERN Large Hadron Collider.
- LCG se encuentra formado por más de 170 centros de cómputo distribuidos en 34 países.
- Se utiliza para el análisis de colisiones entre partículas. Solo se analizan las colisiones que resultan más “promisorias”.
- Se manejan enormes cantidades de datos.
  - Aún rechazando el 99,9995% de las colisiones se generan 19 gigabytes de datos por minuto.
  - 27 terabytes de datos “crudos” por día.
  - Se estiman ~50 petabytes para 2016.
- Cuenta con aproximadamente 200.000 petabytes de capacidad de almacenamiento.

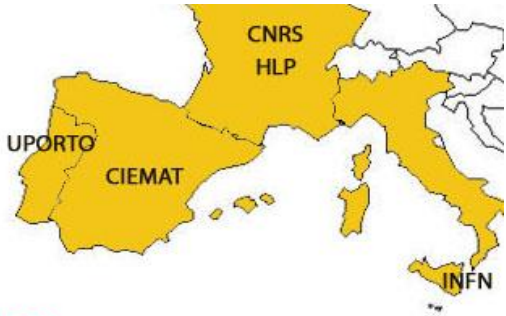
LHC Computing Grid

LHC Computing Grid



GISELA grid

- Proyecto GISELA (Grid Initiatives for e-Science virtual communities in Europe and Latin America).
- Financiado con fondos de European Grid Initiative (EGI).
- Integrado por 19 instituciones de 15 países:
- 4 de Europa
  - Francia, Italia,
  - Portugal, España.
- 11 de Latinoamérica
  - Argentina, Brasil, Chile,
  - Colombia, Cuba, Ecuador,
  - México, Panamá, Perú,
  - Uruguay y Venezuela



Europe

Italy	INFN – Catania
France	CNRS, HLP
Portugal	U.PORTO
Spain	CIEMAT (Coord. Institution)

Latin America and the Caribbean

Argentina	INNOVA-T
Brazil	UFRJ, UFCG
Chile	REUNA
Colombia	UNIANDES
Cuba	CUBAENERGIA
Ecuador	CEDIA
International	CLARA
Mexico	CUDI, UNAM
Panama	CIDETYS
Peru	RAAP
Uruguay	UdelaR
Venezuela	ULA

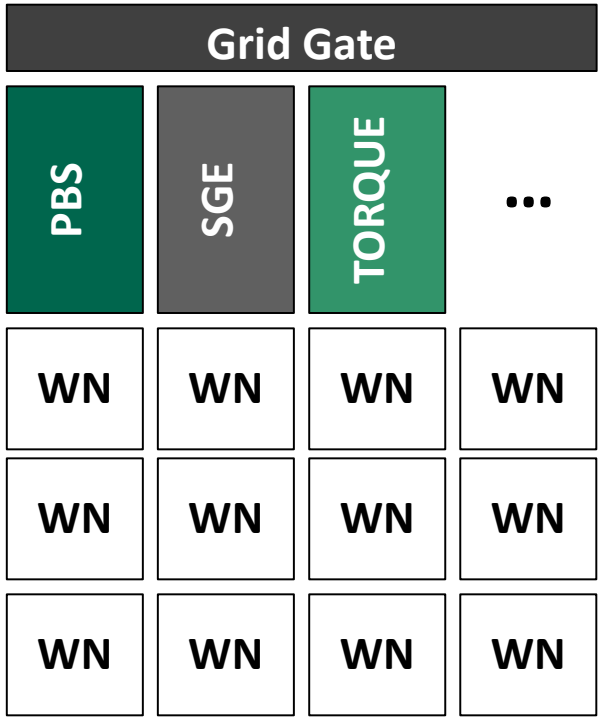
9.3: Middlewares para sistemas grid

gLite middleware

- gLite es un middleware stack para grid computing.
- Utilizado por LHC Grid.
- Seguridad.
  - Todo usuario debe pertenecer a una VO autorizada por la infraestructura para ser autorizado a utilizar el Grid.
  - Grid Security Infrastructure (GSI) basada en Public-Key Infrastructure y Secure Socket Layer (SSL) con extensiones para single sign-on y delegación.
  - Para autenticarse un usuario necesita un certificado digital emitido por una Autoridad Certificadora reconocida.
  - Este certificado digital es protegido por una contraseña y utilizado para generar un certificado temporal llamado *proxy certificate*.
  - Durante su ejecución en el Grid los trabajos mantienen una copia del *proxy certificate* con el que fueron creados para autenticarse con los servicios del Grid.

gLite: principales componentes

- User interface (UI)
  - Punto de acceso de los usuarios a la infraestructura.
  - Incluye herramientas para ejecutar trabajos, cancelar trabajos, obtener resultados, listar trabajos en ejecución, etc.
- Computing element (CE).
  - Provee la capacidad de cómputo del Grid.
  - Expone una interfaz de comunicación genérica Grid Gate (GG).
  - Cuenta con un Local Resource Management System (LRMS)
    - PBS, SGE, TORQUE, SLURM, etc.
  - Contiene una colección de Working Nodes (WN).
    - Usualmente un cluster de computadores.



gLite: principales componentes

- Storage Element (SE)
  - Acceso global y uniforme a los recursos de almacenamiento.
  - Interfaz genérica Storage Resource Manager (SRM).
    - Disk Pool Manager (DPM), dCache, CASTOR, etc.
  - Transmisión de archivos mediante GSI-FTP.
  - Tolerancia a fallos y balanceo de carga mediante replicas de archivos.
  - Define una representación lógica de los archivos (LFN) separada de su representación física (SURL). Por ejemplo:
    - `lfn://grid/gilda/users/mario/myfile.dat`
    - `srm://aliserv6.ct.infn.it/dpm/home/gilda/mario/myfile.dat`
  - El servicio *Logical File Catalogue* (LFC) mantiene la correspondencia entre el LFN de cada archivo y todos sus SURL.
  - Los archivos siguen un modelo “write-once, read-many”.

gLite: principales componentes

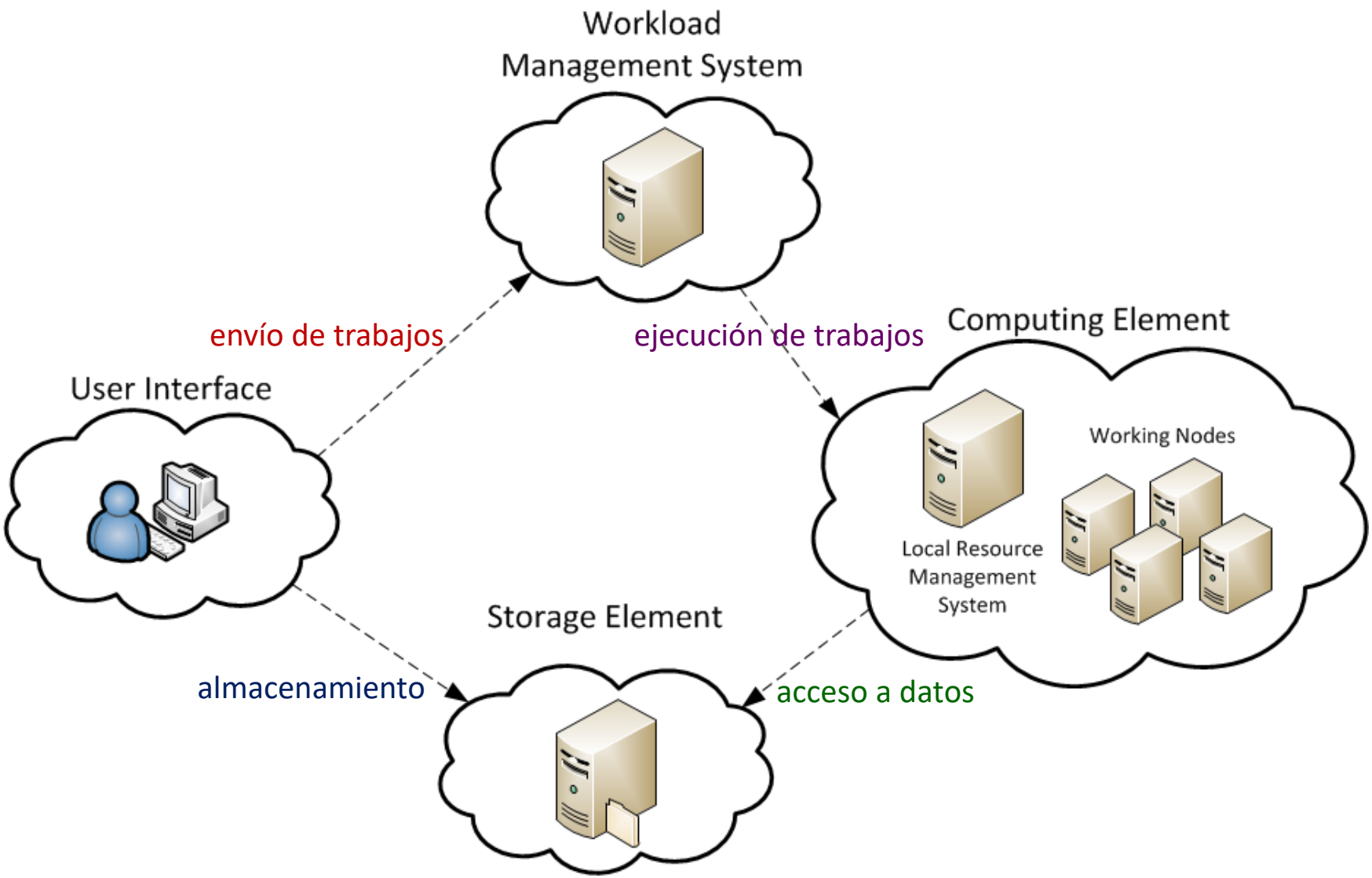
- Workload Management System (WMS)
  - Acepta trabajos de los usuarios y los asigna al CE más apropiado.
  - Descripción de trabajos mediante *Job Description Language* (JDL).
    - Especifica qué ejecutable correr, sus parámetros, archivos de entrada/salida, requerimientos de cómputo, etc.
  - Input Sandbox: conjunto de archivos de entrada de un trabajo.
    - Estos archivos son copiados al WN seleccionado antes de comenzar a correr el ejecutable.
  - Output Sandbox: conjunto de archivos de salida de un trabajo.
    - Estos archivos son retornados al usuario como parte del resultado de ejecución.

gLite: principales componentes

- Workload Management System (WMS)
  - Ejemplo de archivo JDL:

```
Executable = "job.sh";
StdOutput = "stdout.log";
StdError = "stderr.log";
InputSandbox = {"job.sh"};
OutputSandbox = {"stdout.log", "stderr.log",
                 "testfile.txt"};
RetryCount = 0;
```

gLite: principales componentes



gLite: principales componentes

- Information Service (IS)
  - Globus Monitoring and Discovery Service (MDS), utilizado para descubrir y publicar el estado de recursos.
  - Relational Grid Monitoring Architecture (R-GMA), utilizado para contabilidad, monitoreo, y publicación de información a nivel de usuarios.
- ARDA Metadata Grid Application (AMGA)
  - Servicio de base de datos que permite agregar metadatos a archivos.
- Science Gateway (SG)
  - Servicio Web que permite a los usuarios administrar trabajos y datos desde un navegador Web.
  - SG utiliza el portal Liferay para permitir a los usuarios implementar UI de sus aplicaciones en el Grid.
    - Liferay es un portal open source implementado en Java que soporta la especificación de Portlet de Java.

Globus Toolkit

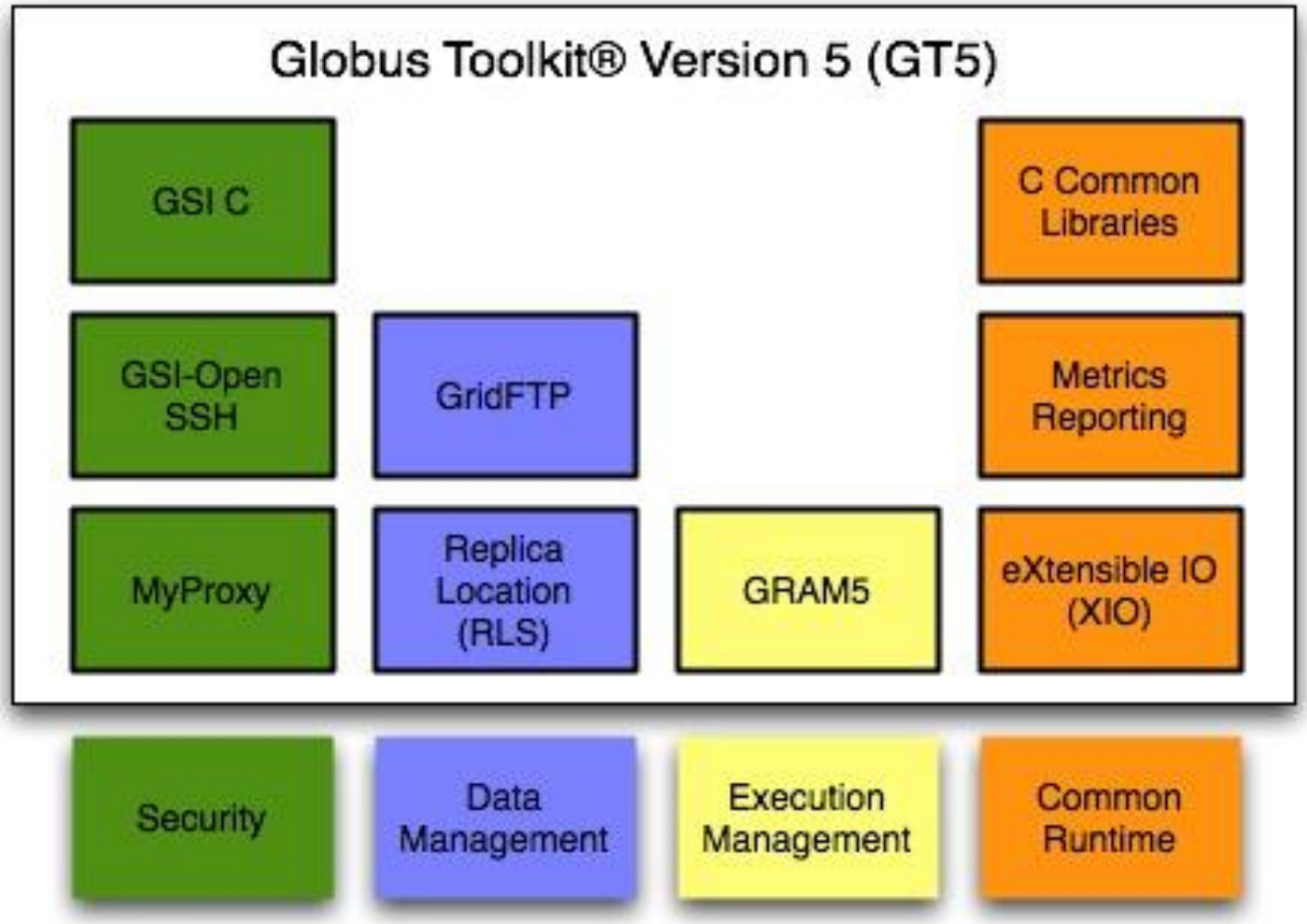
- Globus Toolkit es un middleware de código abierto para la implementación segura de Grids.
- Es desarrollado por instuticiones académicas y gubernamentales de diferentes países (Estados Unidos, Irlanda y Suecia) y ofrece componentes de software para:
  - Seguridad
  - Control de infraestructura
  - Control de recursos
  - Gestión de datos
  - Comunicación
  - Tolerancia a fallos
  - Portabilidad
- El desarrollo de Globus Toolkit empezó en 1998.

the globus® toolkit
www.globustoolkit.org

Globus Toolkit: componentes

- Los principales componentes de Globus son:
  - GridFTP – trasferencia segura de archivos; está basado en el protcolo FTP y fue optimizado para ofrecer transferencia efeciente de grandes conjuntos de datos (GB's y TB's)
  - GRAM5 – gestor de recursos; permite interoperabilidad entre diferentes Local Resource Managers (LRM's), tales como Condor, PBS/Torque, GridEndige, SLURM y Fork
  - MyProxy – servidor para manejo de credenciales
  - SimpleCA – implementación de una Autoridad Certificadora sencilla
- Otros componentes disponibles incluyen una API para Java, bibliotecas para C de propósito general, bibliotecas para C con soporte a Grid Security Infrastructure (GSI C) y servicios GSI sobre OpenSSH.

Globus Toolkit: componentes



- Los componentes de Globus Toolkit están divididos en 4 conjuntos.

Globus Toolkit: GRAM5

- GRAM5 es un gestor de recursos, no es un gestor de trabajos; ofrece servicios y clientes para la comunicación con otros LRM´s.
- GRAM5 usará el LRM disponible en un sitio para la ejecución del trabajo; existen adaptadores para varios LRM´s (Condor, PBS/Torque, GridEngine, SLURM y Fork).
- El adaptador Fork siempre está disponible y equivale a hacer un fork de un proceso en el nodo principal de la instalación de GRAM5.
- El administrador de GRAM5 debe instalar y configurar otros adaptadores y el usuario puede elegir un adaptador en particular.

Globus Toolkit: ejemplo

- Para ejecutar un comando en un grid que use Globus Toolkit el usuario debe usar el comando `globus-job-run`:

```
$ globus-job-run globus.fing.edu.uy /bin/date
Fri Sep 18 23:01:18 UYT 2015
$ globus-job-run globus.fing.edu.uy /bin/hostname
globus.fing.edu.uy
```

- Globus soporta la ejecución de trabajos en modo batch y en modo interactivo.
- Si hay archivos que son necesarios para la ejecución de un trabajo, ésos deben ser copiados (stage_in) previamente y estar disponibles antes de la ejecución del trabajo.

Globus Toolkit: componentes

- El lenguage RSL (Resource Specification Language) provee una manera uniforme de describir recursos.
- Usando RSL es posible indicar los archivos que son copiados para la ejecución de un trabajo:

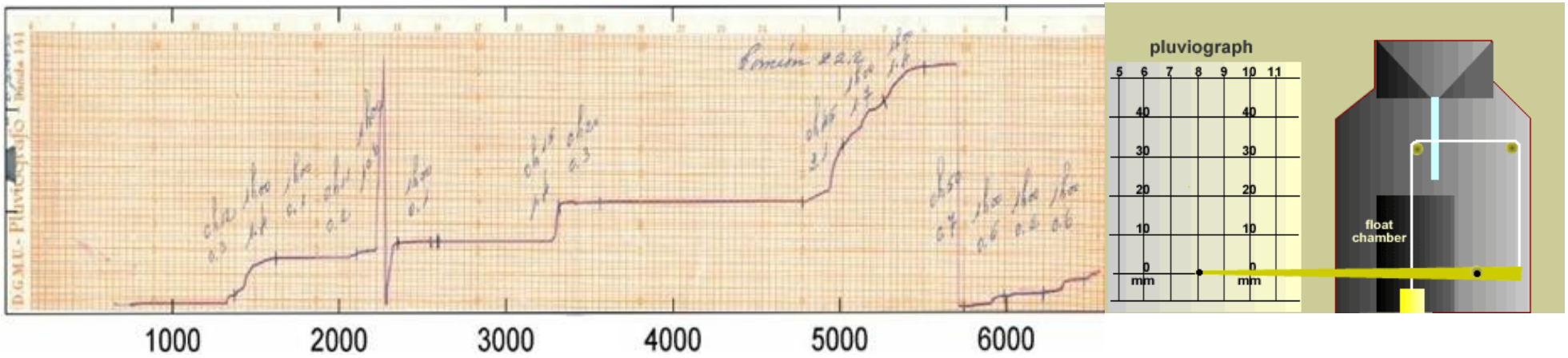
```
$ globus-job-run cliente.globus –x \
      '(file_stage_in=(gsiftp://cliente.globus/bin/echo
/tmp/echo)) (file_clean_up=/tmp/echo)' /bin/ls -l
/tmp/echo
-rw-r--r-- 1 user user 27120 Sep 18 23:05 /tmp/echo
```

- La transferencia de archivos entre cliente y servidor se hace a través de GridFTP y la autenticación está a cargo de MyProxy.

9.4: Ejemplo de aplicación: digi-clima grid en GISELA y Ourgrid

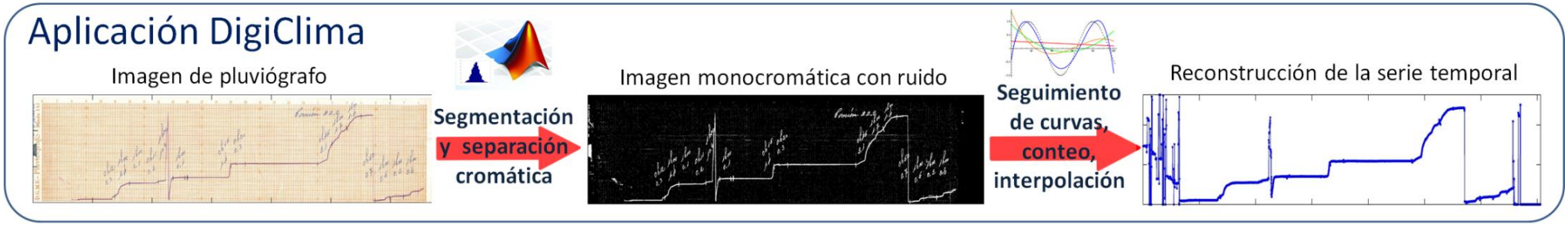
Ejemplo: aplicación digi-clima grid

- Recuperar datos climáticos recabados a través de los años para el estudio de variables climáticas.
- Desarrollar una aplicación eficiente capaz de digitalizar la salida histórica de pluviógrafos y almacenarlas en una base de datos de registros histórico-climáticos.



Ejemplo: aplicación digi-clima grid

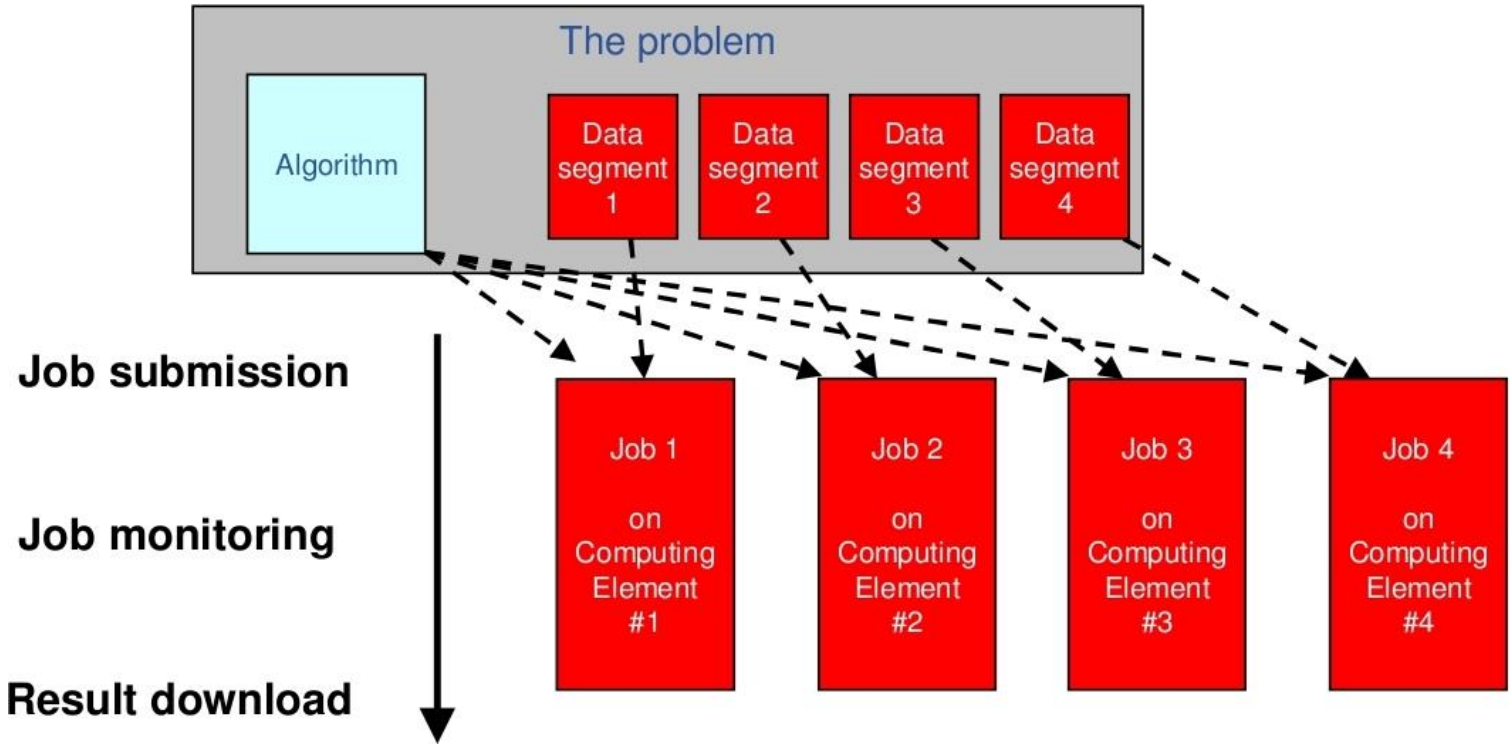
- Para la digitalización se utilizan técnicas de segmentación y separación cromática, seguimiento de curvas, conteo e interpolación.



- En un computador estándar, se requieren 10-15 minutos para procesar una imagen correspondiente a un día de registro pluviográfico.
- Se dispone de más de un siglo de registros para estaciones meteorológicas en varios puntos de Uruguay.
- Es necesario aplicar técnicas de computación de alto desempeño para poder obtener los resultados en tiempos razonables.

Digi-clima grid: modelo de paralelismo

- El procesamiento de cada imagen es independiente por lo que se aplicó un modelo de paralelismo de datos con esquema de caja negra.
- Múltiples instancias de la aplicación ejecutan simultáneamente, para diferentes imágenes. Comunicaciones sólo para mantener información sobre el estado de procesamiento.



Digi-clima en una infraestructura cluster

- En una infraestructura local (cluster), la existencia de un sistema de archivos compartido o una base de datos local resuelve los problemas de almacenamiento y acceso a datos.
- Se puede utilizar una implementación basada en *pilot jobs* independientes, que iteran los siguientes pasos:
  1. obtener una imagen sin procesar de la base de datos
  2. marcarla “en proceso”
  3. ejecutar el proceso Digi-Clima para la imagen
  4. registrar la imagen “procesada” en la base de datos
- La implementación indicada resuelve el balance de cargas e incrementa la escalabilidad, y proporciona un nivel básico de tolerancia a fallos.

Digi-clima en una infraestructura grid

- En una infraestructura grid, existen varios aspectos a resolver:
  1. Distribución de datos: distribuir el banco de imágenes a procesar en el grid
  2. Sincronización y cooperación: evitar procesar la misma imagen por más de una instancia de Digi-Clima
  3. Recuperación correcta de resultados: se deben recuperar los resultados y mantener la referencia correcta de la imagen de los cuales proceden.

- Solución:
  - Para distribuir los datos: script que sube imágenes.
  - Para el procesamiento: scripts independientes para control de ejecución.

DIGI-CLIMA GRID con PILOTS JOBS en GISELA

- Script para subir imagenes al grid:
  1. Crea un directorio en el servicio de metadatos AMGA y agrega atributos:
     i. id en el SE del grid
     ii. estado de procesamiento
     iii. nombre del archivo
     iv. marca de tiempo (última actualización)
     v. jobid del trabajo que procesa
- Se envían las imágenes al grid, usando un servicio de catalogo de archivos (LFC) y generando réplicas en el SE.

Digi-clima grid con pilots jobs en GISELA

- Pilot job para procesamiento en GISELA:
  1. archivo de descripción del trabajo (JDL) para su envío al WMS
     - Especifica parámetros, datos y requerimientos
  2. script para ejecutar Digi-Clima en un WN
     - Inicializa LCG y descarga archivos para ejecutar (intenta localizar la mejor réplica)
     - Itera ejecutando Digi-Clima para un conjunto de imágenes

Digi-clima grid: implementación

- Archivo JDL del pilot job Digi-Clima en GISELA

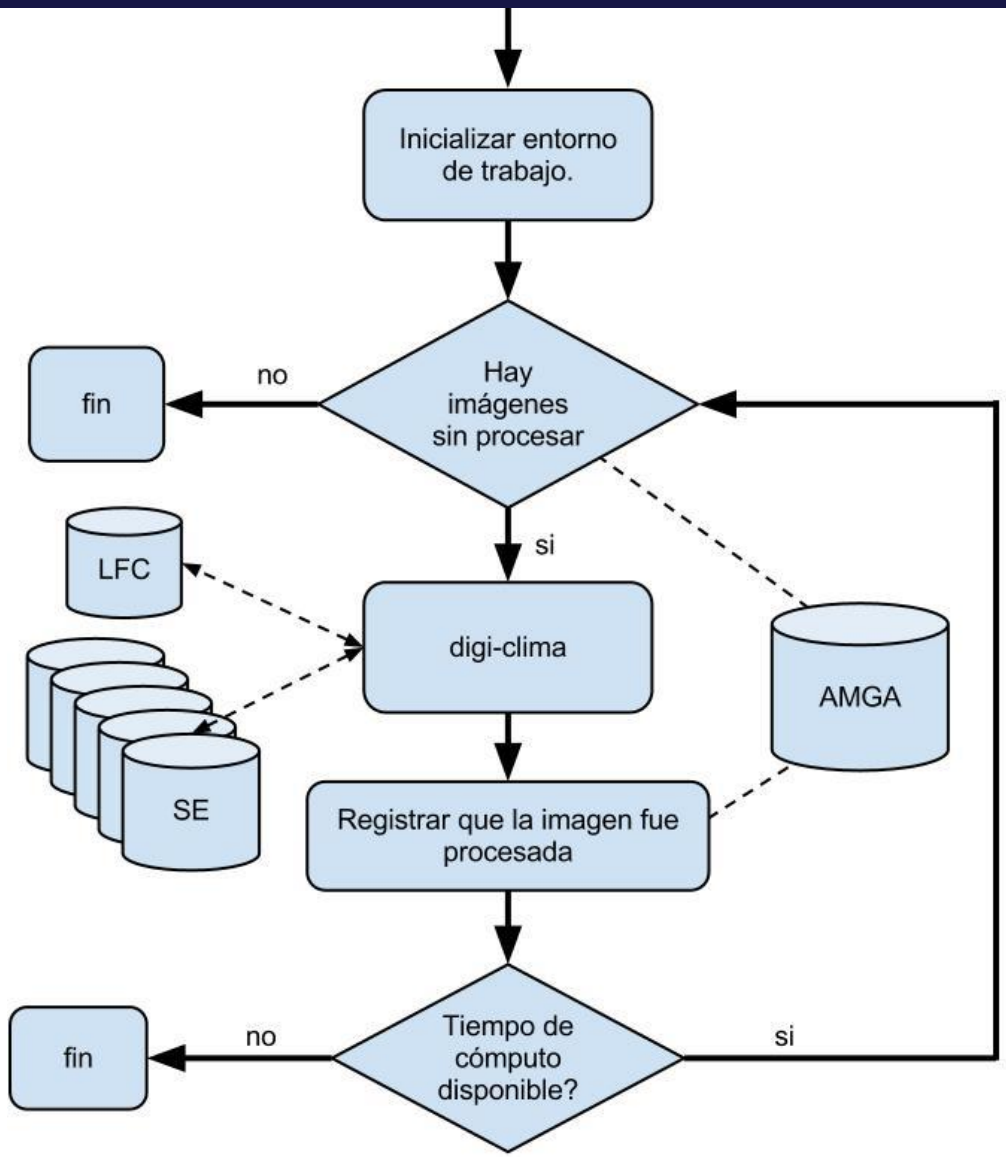
```
1  [ Executable = "/bin/sh";
2    Arguments = "Digi-clima.pilot.sh";
3    StdError = "stderr.err";
4    StdOutput = "stdout.out";
5    InputSandbox={"Digi-clima.sh","pilot.sh","mdclient.config"};
6    OutputSandbox = {"stderr.err", "stdout.out"};
7    RetryCount = 3;
8    Requirements=(other.GlueHostArchitecturePlatformType =="x86_64");]
```

- línea 1: nombre del archivo a ejecutar (shell de Linux)
- línea 2: atributo para pasar parámetros al ejecutable (nombre del script)
- líneas 3 y 4: nombres para la entrada y salida estándar, por OutputSandbox
- línea 5: archivos que serán enviados junto con el trabajo al WMS
  - mdclient.config contiene la configuración del servicio de metadatos AMGA.
- línea 7: cantidad de veces a reintentar la ejecución en caso que falle.
- línea 8: requerimientos, en este caso arquitectura de los nodos de 64 bits.

Digi-clima grid: implementación

- Script para ejecutar Digi-Clima en un WN: utiliza AMGA para mantener información sobre el estado de cada imagen.

1. Consulta a AMGA, ubica una imagen sin procesar.
2. Actualiza el estado de la imagen (“*en proceso*”).
3. Obtiene imagen desde réplica más cercana.
4. Ejecuta la aplicación Digi-Clima sobre la imagen.
5. Almacena el resultado en un SE.
6. Actualiza los metadatos en AMGA (imagen “*procesada*”).



Digi-clima grid: implementación

- Script bash del pilot job en GISELA

```
# Inicializar el entorno LFC y LCG
export vo=prod.vo.eu-eela.eu
export LCG_LOCATION=/opt/grid/ui/lcg
export LCG_GFAL_INFOSYS=bdii-eela.ceta-ciemat.es:2170
for each file do # Descargar achivos
  SEL_REPL= file_uri
  # Intentar obtener la mejor réplica
  REPL=$(lcg-lr file_uri | grep "${$VO_PROD_VO_EU_EELA_EU_DEFAULT_SE}")
  if [ "$REPL" != "" ]; then
    # utilizar replica encontrada en DEFAULT_SE
    SEL_REPL=$REPL
  else
    # Intentar utilizar "se01-tic.ciemat.es"
    REPL=$(lcg-lr file_uri | grep "se01-tic.ciemat.es")
    if [ "$REPL" != "" ]; then
      # se01-tic.ciemat.es no disponible, intentar LFN
      SEL_REPL=$REPL
  lcg-cp -checksum -checksum-type md5 -verbose $SEL_REPL file_name
# Ejecutar Digi-Clima
END_EXECUTION=0
while [ $END_EXECUTION -eq 0 ] ; do
  /bin/sh ${JOB_DIRECTORY}/Digi-clima.sh ${JOB_DIRECTORY}
  if [ $? == $ERROR_ON_ASSIGN -o $EXIT_STATUS == $OK ] ; then
    END_EXECUTION= 0 # Imagen ya procesada, continuar con la siguiente.
  else
    END_EXECUTION= 1 # Error en la ejecución, finalizar pilot job.
```

Digi-clima grid: implementación

- Script bash de la aplicación Digi-Clima en GISELA

```
# Obtener una imagen no procesada.
query_result=$(mdcli -c mdclient.config "SELECT FILE FROM $path/images WHERE status = '$AVAILABLE' LIMIT 1")
if [ "$query_result" != "" ] ; then
 # Marcar imagen como asignada
 mdcli -c mdclient.config "updateattr ${path}/images/${query_result} status $ASSIGNED last_update $(date +\ %s)
        last_update_job \"$JOB_ID\" 'status=$AVAILABLE'"
 if [ $? != 0 ] ; then
    # Otro job tomó la imagen entre el select y el assign (es raro que suceda)
    exit $ERROR_ON_ASSIGN
 # Hallar la mejor réplica y descargar la imagen, como en el script bash del pilot job
 find_best_replica()
 lcg-cp --verbose --checksum --checksum-type md5 $BEST_REPLICA ${image_local_path}
 # Procesar imagen y subir resultados
 ./Digi-Clima $image_local_path
 lcg-cr --verbose --checksum --checksum-type md5 --vo prod.vo.eu-eela.eu -l ${path_LFN}/results/$query_result -d
        $DEFAULT_SE $RESULT
 # Marcar imagen como procesada
 mdcli -c mdclient.config "updateattr ${path}/images/${query_result} status $PROCESSED last_update $(date +\ %s)
        last_update_job \"$JOB_ID\" 'status=$ASSIGNED"
else # No hay más imágenes para procesar, finalizar
 exit $END
```

Digi-clima grid: resultados

- El algoritmo fue evaluado utilizando una base de datos de 150 imágenes (500 MB), y un prototipo liviano del algoritmo Digi-Clima.
- Se compararon los tiempos de procesamiento del algoritmo secuencial, utilizando 4 WN y utilizando 20 WN.

escenario	imágenes	# jobs	tiempo	est.	speedup
secuencial	150	1	73.0m	20.0h	1.00
$grid_4$	150	4	32.0m	5.3h	3.77
$grid_{20}$	150	20	12.0m	1.3h	15.05

- 9 minutos de espera para ejecutar los trabajos en el Grid desde que fueron enviados.
- El banco completo de 30000 imágenes se procesó en 202 horas (8,5 días) utilizando 20 pilot jobs, en lugar de las 4000 horas (166,6 días) que demandaría secuencialmente.

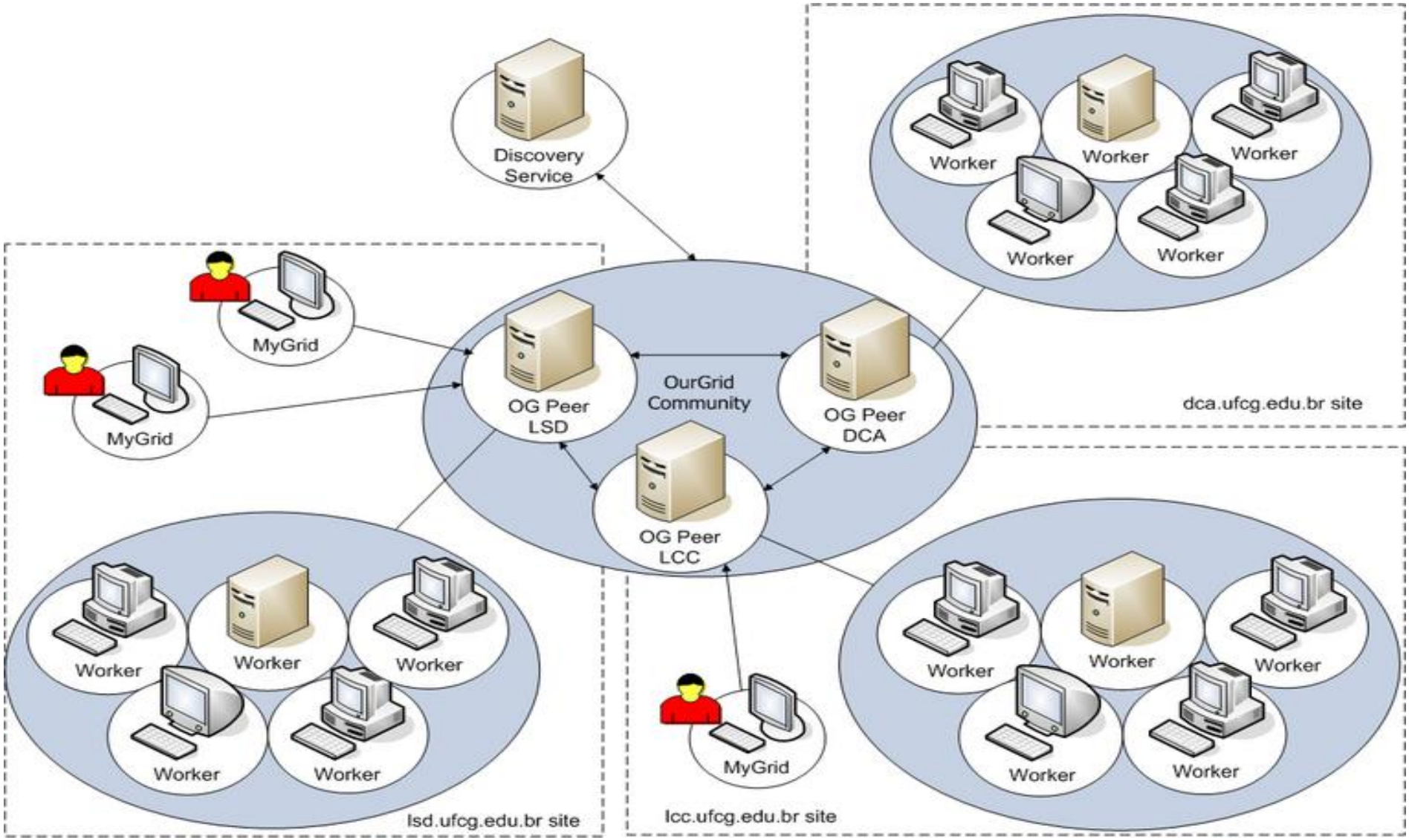
Ourgrid middleware

- OurGrid es un Grid y también un middleware open source que permite la creación de Grids Peer-to-Peer. Fue desarrollado en 2004 en la Universidade Federal de Campina Grande, Brasil.
- Grid voluntario: cualquiera puede fácilmente unirse, donar recursos, y ganar acceso a una gran cantidad de poder computacional para ejecutar aplicaciones paralelas.
- El poder computacional de OurGrid se encuentra provisto por el poder computacional ocioso de todos los participantes.
- Quienes contribuyen con más recursos a la infraestructura tienen preferencia al momento de ejecutar trabajos.
- Actualmente la infraestructura puede ser utilizada para ejecutar trabajos paralelos que trabajen de forma independiente y que no requieran comunicación de resultados intermedios.

Ourgrid: componentes

- Broker
  - Componente encargado de la planificación de trabajos en el Grid. Durante la ejecución de trabajos actúa como coordinador, planificador, y encargado de la transferencia de datos desde y hacia los recursos de cómputo.
  - El Broker también es la interfaz de usuario para ejecutar y monitorear trabajos en el Grid.
- Peer
  - El componente Peer organiza y provee las recursos Worker dentro de un dominio administrativo. Desde el punto de vista del usuario, el Peer es el encargado de obtener dinámicamente y suministrar Workers para la ejecución de un trabajo.
- Worker
  - Componente que ejecuta en toda recurso del Grid disponible para la ejecución de un trabajo.

Ourgrid: componentes



Ourgrid: componentes

- OurGrid utiliza un mecanismo para incentivar a compartir voluntariamente recursos computacionales, llamado *Network of Favours*.
- Network of Favours es un esquema de reputación autónomo que premia a los Peers que contribuyen con más recursos. De esta manera hay un incentivo a contribuir la mayor cantidad de recursos al Grid.
- La asignación de recursos se realiza considerando la Network of favours.
- La comunidad de OurGrid debe compartir sus recursos siguiendo el modelo de Network of Favours. En este modelo, cada Peer ofrece sus recursos ociosos a la comunidad.
- A cambio, cuando los Brokers de un Peer inician trabajos que exceden la capacidad local, el Peer local intenta satisfacer estas solicitudes obteniendo recursos ociosos de otros Peers.

Ejemplo: digi-clima en ourgrid

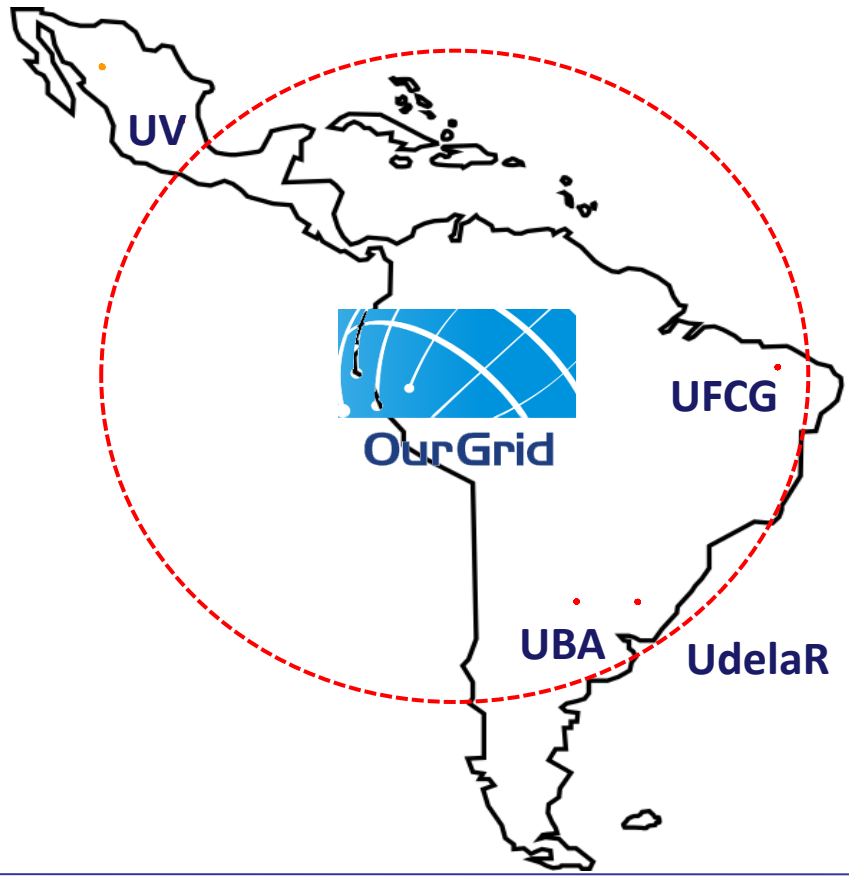
- Ourgrid: WN sobre máquinas virtuales $\rightarrow$ proporciona mayor flexibilidad y escalabilidad para la ejecución de aplicaciones.
- Se utiliza un archivo de descripción del trabajo (job description file, JDF) que permite especificar tareas a ejecutar en paralelo
- Para el caso de Digi-Clima grid, el comando a ejecutar en el worker involucra descomprimir el paquete con el binario de Digi-Clima y el runtime de Matlab, y ejecutar Digi-Clima.
- Se disponen de funciones get y store para copiar archivos desde el broker al worker.

Digi-clima en Ourgrid

- Ejemplo de archivo JDF para Digi-Clima en Ourgrid (procesa una imagen).

```
1   job:
2   label: job-test
3
4   task:
5   init: put img20110214_13484.jpg img20110214_13484.jpg  store
          digiclima.zip digiclima.zip
6   remote: unzip \$STORAGE/digiclima.zip;./run_RunAll.sh
          ./MATLAB_Compiler_Runtime/v714 img20110214_13484.jpg
          img20110214_13484.jpg
7   final: get img20110214_13484.jpg.out img20110214_13484.jpg.out
```

- líneas 1-2 identifican al trabajo
- líneas 4-8 describen una tarea
- Ourgrid permite indicar la ejecución de tareas en paralelo, y el JDF admite cláusulas para indicar requerimientos (SO, RAM, etc.).

Digi-clima en Ourgrid: resultados

- Trabajo sobre 4 sitios OurGrid
  - Univ. de la República (Uruguay), Univ. de Buenos Aires (Argentina), Univ. Federal de Campina Grande (Brasil), y Univ. Veracruzana (México)
- Dos enfoques:
  1. Distribución de aplicación y runtime Matlab: eficiencia reducida (≈0.5) a causa del mecanismo lento de transferencia Java utilizado en Ourgrid
  2. Distribución externa de aplicación, runtime e imágenes, eficiencia correspondiente a speedup casi lineal (20.6 en 24 workers, eficiencia 0.86)