



Tema 6. Programación de Aplicaciones Grid

Conceptos de la Computación en
Grid y Cloud



- Aplicar las Técnicas Más Adecuadas Para el Desarrollo de Aplicaciones que utilicen Infraestructuras Grid.
- Desarrollar e Implementar Aplicaciones Básicas Utilizando la Capacidad Computacional del Grid de globus.

- Programación Aplicaciones Grid.
 - Alternativas
 - Etapas de desarrollo
- Casos de Estudio
 - Caso Estudio 1: Filtrado de Imágenes.
 - Caso Estudio 2: Ajuste de Parámetros en Resolución de Sistemas de Ecuaciones por SOR.

- **PRÁCTICA 1:** Puesta en Marcha del Caso de Estudio 1.
- **PRÁCTICA 2:** Puesta en Marcha del Caso de Estudio 2.

- **Programación Aplicaciones Grid.**
 - Alternativas
 - Etapas de desarrollo
- **Casos de Estudio**
 - Caso Estudio 1: Filtrado de Imágenes.
 - Caso Estudio 2: Ajuste de Parámetros en Resolución de Sistemas de Ecuaciones por SOR.

- **Command Line Interface (CLI)**
 - Programación Mediante Scripts que Utilizan los Comandos de globus.
 - Menos Eficiente Pero Más Rápido de Desarrollo.
 - Mayor Portabilidad.
- **Application Program Interface (API)**
 - Enlazando con las Bibliotecas Software de globus o de Nivel Superior.
 - Más Eficiente y Compacto.
 - Más Dependiente de la Plataforma.

Programación de Aplicaciones Grid

Etapas de Desarrollo

1. Análisis de las Dependencias.
2. Diseño de la Arquitectura Global y las Sincronizaciones.
3. Diseño de los Módulos Funcionales e Interfaces.
4. Implementación de los Módulos Relativos al Proceso.
5. Despliegue de la Aplicación.
6. Producción.

Programación de Aplicaciones Grid

Etapas de Desarrollo

Análisis de las Dependencias



- Los Programas Grid Se Corresponden con un Modelo de Memoria Distribuida Poco Acoplado.
- Las Comunicaciones entre los Procesos son Poco Eficientes.
- Los Diferentes Bloques Concurrentes Deberán Requerir un Tiempo de Proceso que Compense los Retardos del Grid.

Programación de Aplicaciones Grid

Etapas de Desarrollo – Análisis de Análisis de Análisis de las Dependencias

● Modelos

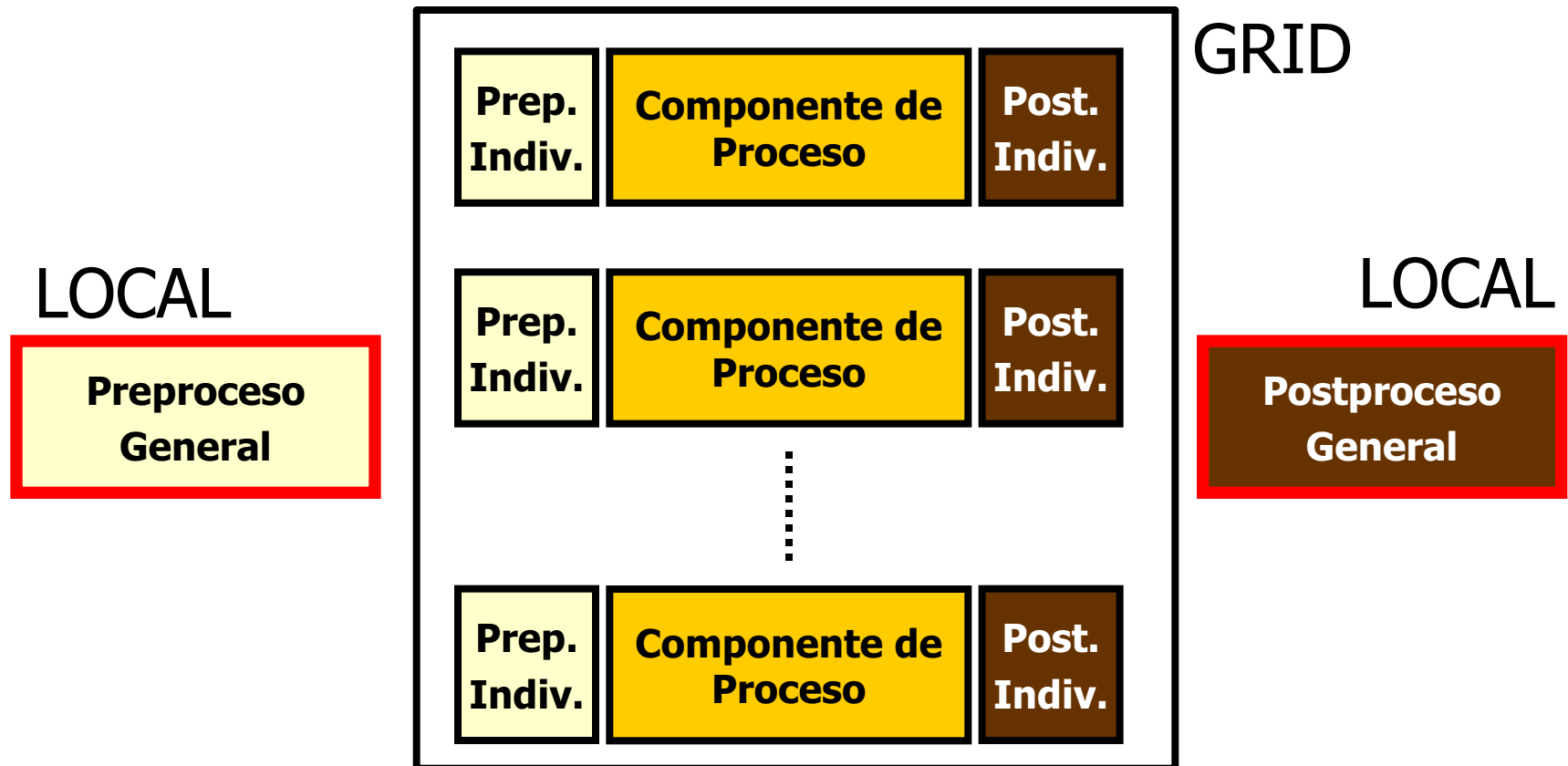
- Modelo 1: Ejecución Masiva de Proceso sobre Datos.
 - Se Debe Aplicar de Forma Independiente un Proceso a una Gran Cantidad de Datos.
 - P.e. Ecualización de un Conjunto de Ficheros de Audio.
- Modelo 2: Ejecución Multi-paramétrica.
 - Se Debe Realizar la Misma Ejecución un Número Elevado de Veces Variando el Valor de Algunos Argumentos.
 - P.e. Generación de Imágenes Fotorealistas Desde Diferentes Puntos de Vista en una Animación.
- Modelo 3: Programa Paralelo Poco Acoplado
 - Los Diferentes Procesos Trabajan de Forma Independiente sobre Datos Distribuidos.
 - P.e. Búsqueda de Homólogos en Bases de Datos de Proteínas

Programación de Aplicaciones Grid

Etapas de Desarrollo

Diseño Arquitectura Global y sincronizaciones

- Diseño de la Arquitectura Global y las Sincronizaciones



Programación de Aplicaciones Grid

Etapas de Desarrollo

Diseño de Módulos Funcionales e Interfaces

- **Ejecución Local**

- **Preproceso General**

- Generalmente Implica Particionar Datos de Entrada y Definir los Valores de los Parámetros Para Cada Trabajo Individual.
 - Puede Implicar Crear Scripts Adaptados al Vuelo.
 - Dependiendo del Caso, Puede Activar el GASS y Transferir los Ficheros Apropriados.

- **PostProceso General**

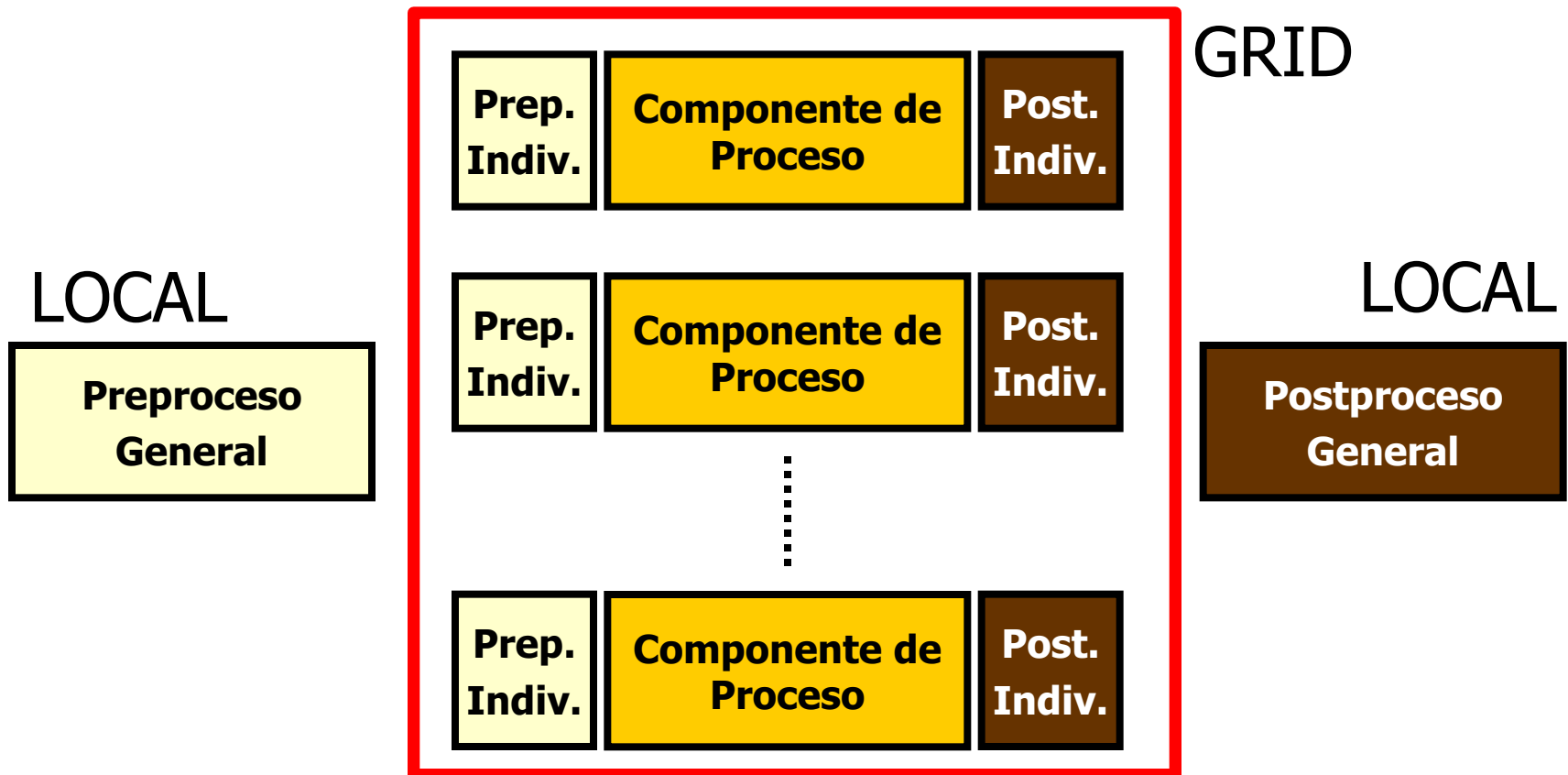
- Generalmente Implica Esperar la Finalización de los Diferentes Procesos Ejecutados en el Grid y Recoger y Consolidar su Salida.

Programación de Aplicaciones Grid

Etapas de Desarrollo

Diseño Arquitectura Global y sincronizaciones

- Diseño de la Arquitectura Global y las Sincronizaciones



Programación de Aplicaciones Grid

Etapas de Desarrollo

Diseño de Módulos Funcionales e Interfaces

- Ejecución en el Grid
 - Preproceso
 - Puede Implicar el Compilar Fuentes, Cambiar Permisos de Ejecución o Incluso Copiar Ficheros Remotos.
 - Proceso
 - Generalmente es un Ejecutable Autónomo que Realiza el Proceso de los Datos Preparados por los scripts de preproceso.
 - PostProceso
 - Generalmente Puede Implicar Copiar Resultados y Eliminar Temporales.

Programación de Aplicaciones Grid

Etapas de Desarrollo

Implementación de los Módulos Relativos al Proceso

- Ejecutables Autónomos, Preferiblemente Enlazados con Bibliotecas Estáticas y Funcionando en Modo Batch.
- Normalmente Obtienen los Datos de Entrada por Línea de Comandos o por Ficheros de Entrada y Configuración.
- La Salida Se Realiza por la Salida Estándar y/o Ficheros Específicos.
- A Veces es Necesario Compilar y Enlazar en las Máquinas Destino.

Programación de Aplicaciones Grid

Etapas de Desarrollo

Despliegue de la Aplicación



- El Despliegue Puede Implicar la Preinstalación de Software o Datos Específicos.
- Una Vez Completados, Se Activan las Credenciales (Creación del Proxy) y se Lanza el Proceso Desde el Ordenador Local.

- Programación Aplicaciones Grid.
 - Alternativas
 - Etapas de desarrollo
- **Casos de Estudio**
 - Caso Estudio 1: Filtrado de Imágenes.
 - Caso Estudio 2: Ajuste de Parámetros en Resolución de Sistemas de Ecuaciones por SOR.

- Caso Estudio 1: Filtrado de Imágenes.
(Modelo 1 Ejecución Masiva de Proceso sobre Datos)
- Caso Estudio 2: Ajuste de Parámetros en
Resolución de Sistemas de Ecuaciones por SOR.
(Modelo 2: Ejecución Multiparamétrica)

Caso de Estudio 1: Filtrado de Imágenes



Caso de Estudio 1: Descripción

- Filtrado de Imágenes.
 - Implementación de un Filtrado Artístico a una Serie de Imágenes.
 - Comando convert –paint 25 filein fileout
 - Para una Imagen de 3MP el Tiempo Invertido para su Ejecución es de 350 Segundos.
 - Probar con –paint 5
 - El Objetivo es Utilizar un Grid Para Transformar un número grande de Imágenes.
 - Aplicación poco Acoplada sin Posprocesamiento Posterior
 - Típico Caso de Ejecución Masiva de Proceso sobre Datos

● Etapas

- Selección de los Recursos Donde Ejecutar.
- Particionado de Tareas.
 - Copiado de los Ficheros de Imagen Originales (Staging).
 - Ejecución de la Transformación.
 - Copiado de los Resultados (Staging).
 - Verificación de los Errores.
- Monitorización.

- Para la selección de los Recursos Donde se van a ejecutar los trabajos, crea un fichero con el nombre de las maquinas de tus compañeros y llámalo ***recursos*** (puedes consultar */etc/hosts* para crear dicho fichero):
 - XXX.ccgc.mastercpd.upv.es
 - ...
- Uso de GASS server de mi maquina LOCAL para realizar el staging de datos.

Ej. [profesor]\$ globus-gass-server -r -w -t
https://xxx.ccgc.mastercpd.upv.es:47301

● Particionado de Tareas

- Cada Imagen (puedes coger las de poliformaT input_images) se Procesará de Forma Independiente
 - `convert -paint 15 img0x.jpg img0x_f.jpg`
- Las Imágenes de Entrada y Salida se Copiarán por Staging.
- Se Generará un Fichero "RSL" por Cada Proceso.

```
&
(count = 1)
(executable = /usr/bin/convert)
(arguments = -paint 15 img00.jpg img00_f.jpg )
(rsl_substitution = (GASS_URL
  https://xxx.ccgc.mastercpd.upv.es:XXXXX/home/ccgc/Evidencias/Grid/04_P
  rogramacion/P1))
(stdout = $(GASS_URL)/miStdout)
(stderr = $(GASS_URL)/miStderr)
(file_stage_in = ( $(GASS_URL)/input_images/img00.jpg img00.jpg ) )
(file_stage_out = (img00_f.jpg $(GASS_URL)/output_images/img00_f.jpg ) )
```

- Particionado de Tareas
 - Un **Script Generará los N Ficheros RSL** (uno por cada fichero imagen) y Lanzará los N Trabajos.
 - Cada Script Se Adaptará a un Fichero Concreto y Producirá un Resultado Diferente.
 - Los datos de entrada y salida se enviarán por Staging.
 - Un **Script para el lanzamiento** de los trabajos, que se Enviarán Mediante Staging.
 - Los Procesos se Someterán en Background y se Capturará el Identificador del Proceso.
 - El **Script Deberá Monitorizar** la Terminación de los Procesos Sometidos Comprobando de Forma Periódica el Estado de los Procesos.
 - Un **Script para recuperar los Resultados**, se recuperarán Mediante Staging. una vez finalizados todos los trabajos

Casos de Estudio 1

- Creación de los RSL (gen_rsl.sh) con un argumento de entrada que indica el numero de imágenes a procesar.

```
#!/bin/bash
GASS_PORT=YYYY
LOCAL_HOST=XXX.ccgc.mastercpd.upv.es
NUMBER_IMAGES=$1
# Creacion de los RSL

i=0

while [ $i -lt $NUMBER_IMAGES ]
do
    echo "&" >script_${i}.rsl
    echo "(count = 1)" >>script_${i}.rsl
    echo "(executable = /usr/bin/convert)" >>script_${i}.rsl
    echo "(arguments = -paint 5 img0${i}.jpg img0${i}_f.jpg )" >>script_${i}.rsl
    echo "(rsl_substitution = (GASS_URL
        https://$LOCAL_HOST:${GASS_PORT}/home/ccgc/Evidencias/Grid/04_Programacion/P1))"
        >>script_${i}.rsl
    echo "(stdout = \$(GASS_URL)/miStdout)" >>script_${i}.rsl
    echo "(stderr = \$(GASS_URL)/miStderr)" >>script_${i}.rsl
    echo "(file_stage_in=( \$(GASS_URL)/input_images/img0${i}.jpg img0${i}.jpg ) )" >>script_${i}.rsl
    echo "(file_stage_out=(img0${i}_f.jpg \$(GASS_URL)/output_images/img0${i}_f.jpg) )"
        >>script_${i}.rsl
    i=$(( i+1 ))
done
```


- Lanzamiento (sub_jobs.sh)
 - Copiamos en un Fichero los Identificadores de Todos los Trabajos Generados.
 - Mostramos por Pantalla Cada Identificador para su Verificación.

```
#!/bin/bash
# Lanzamiento de los trabajos
rm -r proc_ids.txt
NUM_JOBS=$1
JOB=0
LINE=1

while [ $JOB -lt $NUM_JOBS ]
do
    RES=`sed -n ${LINE}p recursos`      #capturo el nombre del recurso
    echo globusrun -b -r ${RES} -f script_${JOB}.rsl
    globusrun -b -r $RES -f script_${JOB}.rsl | grep http >> proc_ids.txt
    JOB=$(( JOB+1 ))
    LINE=$(( LINE+1 ))
done
```

- Monitorización (stat_jobs.sh)
 - Comprobamos que los 7 Procesos Hayan Acabado como "Done".

```
NUM_JOBS=$1
procesos=$1
JOB=0
LINE=1

while [ $procesos -gt 0 ]; do
    while [ $JOB -lt $NUM_JOBS ]
    do
        id=`head -${LINE} proc_ids.txt | tail -1`
        echo 'Analizando proceso' ${id}
        if [ `globus-job-status ${id}` = "DONE" ]; then
            procesos=$((procesos-1))
        fi
        LINE=$(( LINE+1 ))
        JOB=$(( JOB+1 ))
    done
    echo .Pendientes: $procesos esperamos 30 segundos
    sleep 30
done
```

P1

P1

Caso de Estudio 2: Resolución de Sistemas de Ecuaciones por SOR

- **Ajuste de Parámetros en Resolución de Sistemas de Ecuaciones por SOR**
 - La Resolución de un Sistema de Ecuaciones Mediante el Método Iterativo de SOR Implica el Ajuste de un Parámetro de Convergencia (w) que Puede Tomar Valores entre 1 y 2.
 - Se Debería Repetir la Ejecución Para Distintos Valores de w y Escoger Aquél que Requiera el Mínimo Número de Iteraciones (Siempre y Cuando Converja).
 - Aplicación Multiparamétrica Típica. Se Debe Realizar la Misma Ejecución un Número Elevado de Veces Variando el Valor de Algunos Argumentos.

- Utilizaremos Octave para la Ejecución

- `octave -q test_sor.m <dim> <w>`

```
d = int32(str2num(argv(){1}));
w = double(str2num(argv(){2}));

n = d*d;
mu = 0.01;
v1 = ones(d*d-1,1);
v2 = ones(d*(d-1),1);
A = eye(n)+eye(n)*4*mu + diag(v1,1)*mu
    + diag(v1,-1)*mu + diag(v2,d)*mu +
    diag(v2,-d)*mu;
b = zeros(n,1); b(n/2,1)=1;
x0 = zeros(n,1);
it = 0;
salir = 0;

Maxits = 1000;
tol = 1e-8;
```

```
while (it<Maxits) & (salir==0)
    for i=1:n
        x(i) = b(i);
        for j=1:i-1
            x(i) = x(i) - A(i,j)*x(j);
        end
        for j=i+1:n
            x(i) = x(i) - A(i,j)*x0(j);
        end
        x(i) = x(i)/A(i,i);
        x(i) = w*x(i) + (1-w)*x0(i);
    end
    if (norm(x(:)-x0(:))< tol)
        salir = 1;
    end
    it = it +1;
    x0 = x(:);
end
it
```

● Etapas

- Creación de un Script que Construya los RSLs y Lance los Trabajos.
- Lanzamiento de los Trabajos y Monitorización.
- Recogida de Resultados y Postproceso.
- Selección del Valor Óptimo de 'w'.

Casos de Estudio 2

- Etapas
 - Puesta en marcha del GASS (globus-gass-server) LOCAL.
 - Creación de un Script que Construya los RSLs (rsl.sh).
 - El fichero *test_sort.m* se debe de pasar por stage_in

● Etapas

- Lanzamiento de los Trabajos (sub_jobs.sh de la P1)
 - globusrun -r XXX.ccgc.mastercpd.upv.es -b -f script_XX.rsl
- Recogida de Resultados por el Staging.
- Selección del Valor Óptimo (post_out.sh).

```
tail -1 miStdout_0 >> aux.txt
maximo=`sed -e 's/it =//' aux.txt`
echo max=${maximo}
w=0
for id in 0 1 2 3 4 5 6 7 8 9 ; do
    rm aux.txt
    tail -1 miStdout_${id} >> aux.txt
    valor=`sed -e 's/it =//' aux.txt`
    echo max=${maximo}
    echo val=${valor}
    if [ ${valor} -lt ${maximo} ]; then
        maximo=${valor}
        w=${id}
        echo 'cambio:' ${maximo}
    fi
done
echo 'optimo = ' ${w}
echo '    con ' ${maximo} ' iteraciones '
rm aux.txt
```

P2