Formation Utilisateur grille de calcul : FU

United Nations
Educational, Scientific and
Cultural Organization

FU5 : 20 et 21 Juin 2013
CNRST - Rabat

Plan de Formation

- Introduction aux grilles de calcul
- Projet MaGrid & UNESCO-HP
- Services de base de gLite
- Gestion des jobs sur la grille

Nabil Talhaoui
grid@magrid.ma
Division TIC – CNRST, Rabat

Gestion des jobs sur la grille

Gestion des jobs sur la grille

Job/Description d'un Job

- Pour exécuter un Job quelconque sur la Grille, il faut d'abord savoir le décrire en utilisant le Langage de Description de Job ou JDL.
- Décrire un Job signifie connaitre:
  - La ligne de commande ou script à exécuter
  - Les arguments à passer
  - Les inputs à utiliser
  - Les outputs à récupérer
  - Les prérequis en terme de ressources à utiliser et l'environnement à préparer pour l'exécution réussi du job
- Comprendre le workflow d'un job est nécessaire pour la bonne gestion des Jobs sur la Grille
- Les jobs peuvent être simples comme ils peuvent être complexes.
- Il faut toujours garder en tête qu'après les opérations du WMS, le job enfin s'exécute sur un WN, par conséquent l'environnement du WN c'est l'environnement de l'exécution du Job.
- WMS est le composant principale de gLite qui prenne en charge l'exécution du Job, récupération du InputSandbox du UI et l'envoie du OutputSandbox à l'UI.

Gestion des jobs sur la grille

Exemple de Job simple

- Job:

  Imprimer l'environnement du WN par la commande, printenv

- Description du Job (le fichier JDL) :

  Type = "Job"; //Il s'agit d'un job simple

  JobType = "Normal"; //type de Job

  Executable = "/usr/bin/printenv"; //excutable défini par son chemin absolu sur le WN

  StdOutput = "std.out"; //qui va contenir le resultat

  StdError = "std.err"; //qui va contenir les erreurs eventuels de l'exécution

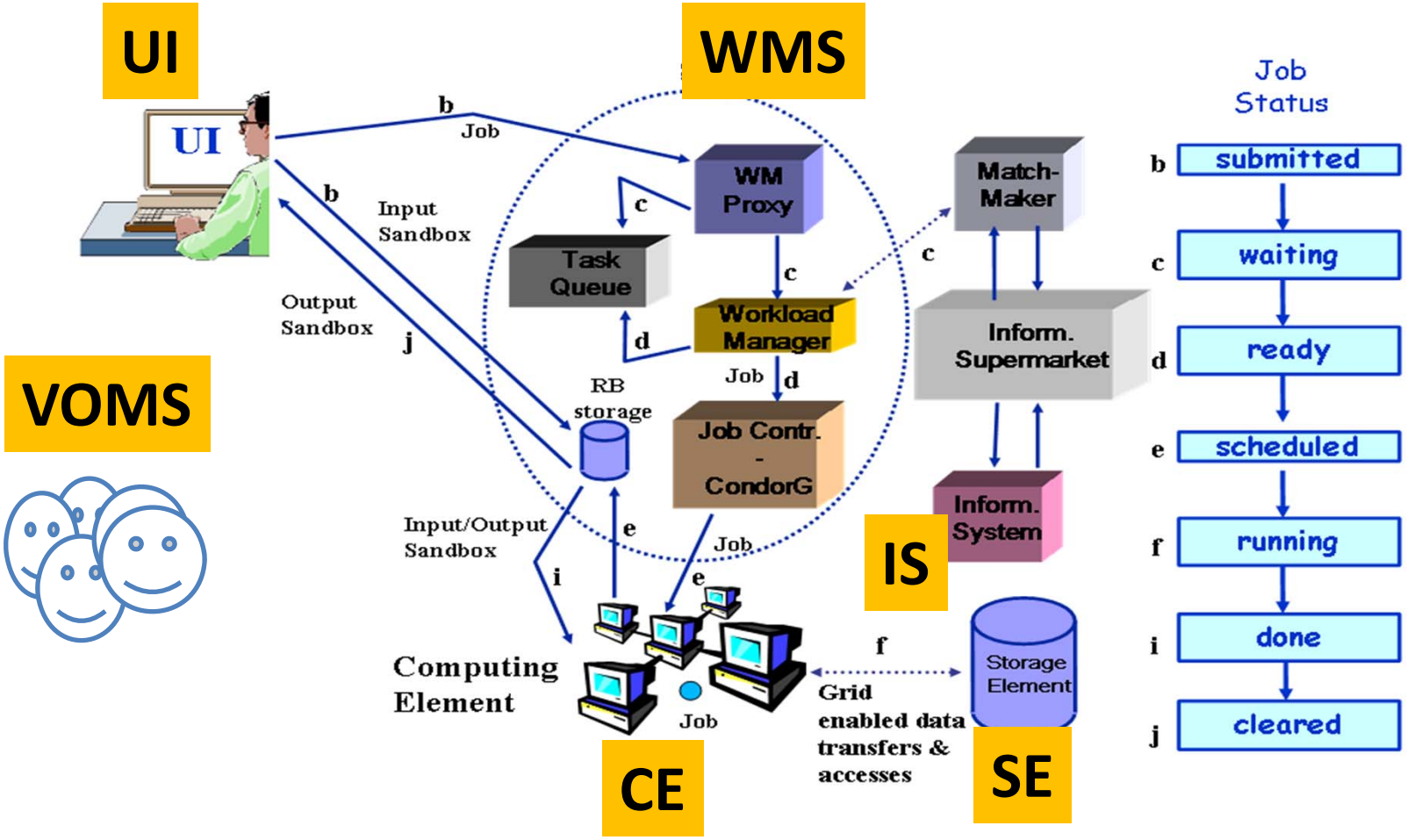
  StdOutputSandbonx = {"std.out","std.err"}; //informer le WMS des fichiers à récupérer

- Commandes pour manipuler le Job sur la grille:

  glite-wms-job-*

Gestion des jobs sur la grille

Workflow d'un job



Gestion des jobs sur la grille

Job Description Language (JDL)

- Le JDL est un langage extensible de description des tâches ou Jobs à exécuter sur une grille.
- Basé sur le language de classes CLASSAD “CLASSified ADvertisement”.
- Le fichier JDL est une suite d'instructions de la forme:

<attribut> = <valeur>/{<val1>,….,<valN>};

#Commentaire: le JDL est insensible à la case, sensible au TABs

- Elles définissent un ensemble d’attributs à interpréter par le WMS groupés en 2 catégories:
  - **Attributs du job** : caractéristique du job lui même
  - **Attributs des ressources** : caractéristiques des prérequis du job
  - **Attributs des donnés** : caractéristiques spécifiques à la manipulation des données

Gestion des jobs sur la grille

Type

- L'attribut **Type** est une chaine qui indique le type du job à exécuter

✓ Syntaxe : **Type = "Job";**

✓ Valeur possibles :

- **Job** : un job simple (valeur par défaut)
- **DAG** : **D**irect **A**cyclic **G**raph pour les jobs dépendants
- **Collection** : ensemble de jobs indépendants

Gestion des jobs sur la grille

JobType

L'attribut **JobType** est une chaine ou plusieurs chaines qui représentent le type de job décrit par JDL

Syntaxe :

JobType = "Normal";

Valeur possibles :

"**Normal**" : Job simple (valeur par défaut)

"**Parametric**" : Pour soumettre plusieurs instances de job similaire avec un paramètre qui change

Cet attribut à un sens que si l'attribut : **Type = "Job"**

Gestion des jobs sur la grille

Executable

- L'attribut Executable est une chaine qui représente le nom de la commande ou de l'exécutable
- Un utilisateur peut spécifier une exécutable déjà installée sur le cluster, dans ce cas le chemin absolu doit être indiqué:

  Ex : Executable = "/usr/local/java/j2sdk1.4.0_01/bin/java";
- Une autre possibilité est de fournir une exécutable local, qui sera copié du UI aux WN lors de la soumission. Dans ce cas, le chemin absolu du fichier doit être listé parmi les valeurs de l'attribut InputSandbox

  Ex : Executable = "myExe";

  InputSandbox = {"/home/user/work/myExe", … };

Gestion des jobs sur la grille

Arguments

- L'attribut Arguments est une chaine qui contient la liste des arguments de l'executable du job

  Ex : Une exécutable native de linux "echo" qui affiche une chaine de charactère

  $ echo "Hello it\'s a test!"

- Le job est décrit de la façon suivante :

  Ex :

  Executable = "/usr/bin/echo";

  Arguments = "Hello, it\'s a test!";

Gestion des jobs sur la grille

InputSandbox & OuputSandbox

- InputSandbox est une chaine ou une liste de chaines indiquant les fichiers (inputs, executable etc) à transférer du UI et dont l'exécution du job a besoin.

  InputSandbox = { "/home/user/work/myExe"};

- OutputSandbox est une chaine ou une liste de chaines indiquant fichiers générés par le job et que l'utilisateur veut récupérer par la suite.

  OutputSandbox = { "myJobOutput", "myJobError", };

Gestion des jobs sur la grille

RetryCount

L'attribut RetryCount est une valeur entière qui représente le nombre maximum de resoumissions automatiques de job à l'exécution par le WMS en cas d'erreurs dues aux problèmes des composants de la grille.

RetryCount = 3;

RetryCount = -1; (pour désactiver)

Gestion des jobs sur la grille

Requirements

- Requirements spécifie les besoins du job vis à vis des ressources de calcul.
- Syntaxe : Requirements = < C-Like Boolean Expressions >

Ex1 : Demander au WMS de choisir les sites là ou les WNs doivent avoir plus de 2 CPUs

```
Requirements = other.GlueCEInfoTotalCPUs > 2;
```

Ex2: Demander au WMS de choisir un CE particulier pour exécuter le job

```
Requirements = other.GlueCEUniqueID == "lxshare0286.cern.ch:2119/jobmanager-pbs-short";
```

Ex3: Demander au WMS de choisir n'importe quel CE mais dans le domain: cern.ch

```
Requirements = RegExp("cern.ch",other.GlueCEUniqueID);
```

Ex4: Demander au WMS de choisir un site avec un PBS comme système de queue et avec des WNs qui ont au moins 2 CPU:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" && other.GlueCEInfoTotalCPUs >= 2;
```

Gestion des jobs sur la grille

Liste des attributs

- Se référencier à la documentation du JDL pour la liste descriptive des attributs.
- Certains attributs sont mandataires (Ex: Executable), les autres non (Ex: StdOutputSandbox).
- Si les attributs ne sont pas spécifiés explicitement sur le JDL, alors les valeurs par défaut sont considérées par par le WMS.

Gestion des jobs sur la grille

Commandes pour gérer un job

glite-wms-job-submit

Commande principale de soumission du job vers le WMS

glite-wms-job-list-match

Liste les ressources correspondantes à la description du job

Permet de connaître le résultat de l'ordonnancement sans soumettre le job.

glite-wms-job-cancel

Annuler le job

glite-wms-job-status

Afficher l'état du job.

glite-wms-job-output

Récupèrer le contenu de OutputSandbox

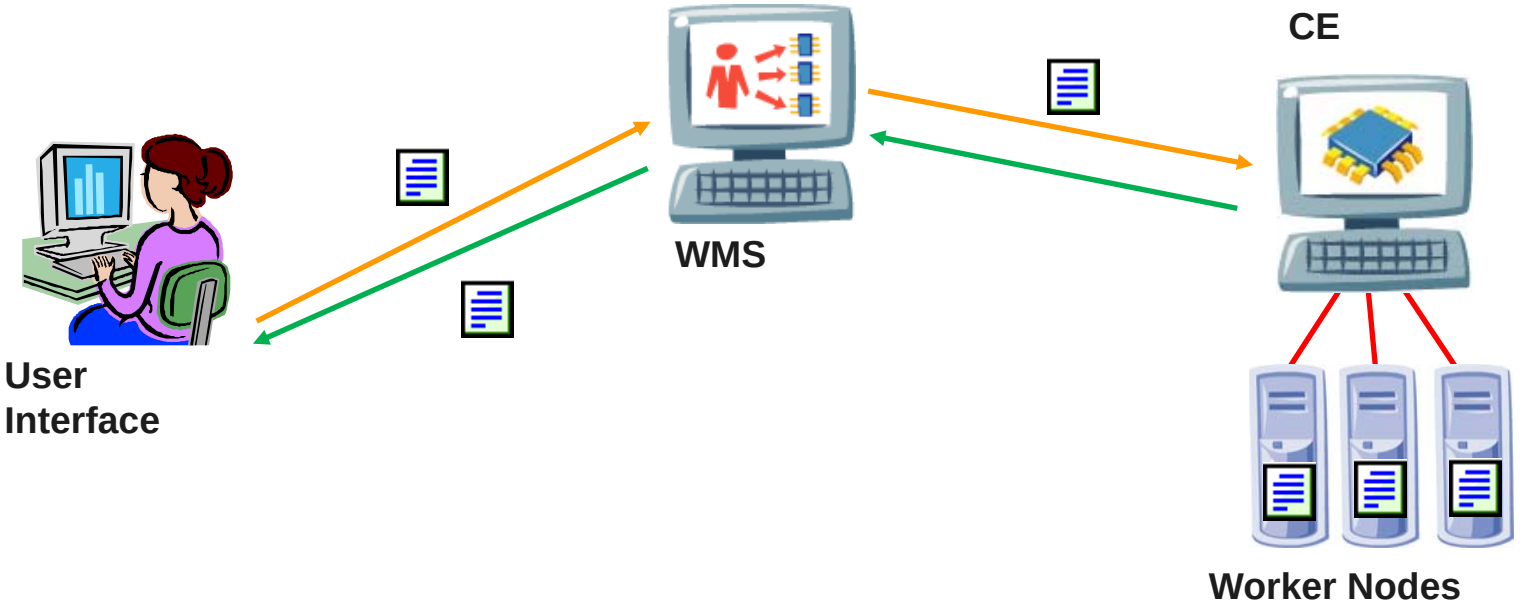
glite-wms-job-logging-info

Afficher des informations sur les différents états pris par le job tout au long de son existence (Utilisé essentiellement pour le debuging).

Gestion des jobs sur la grille

Soumission d'un job – cas 1

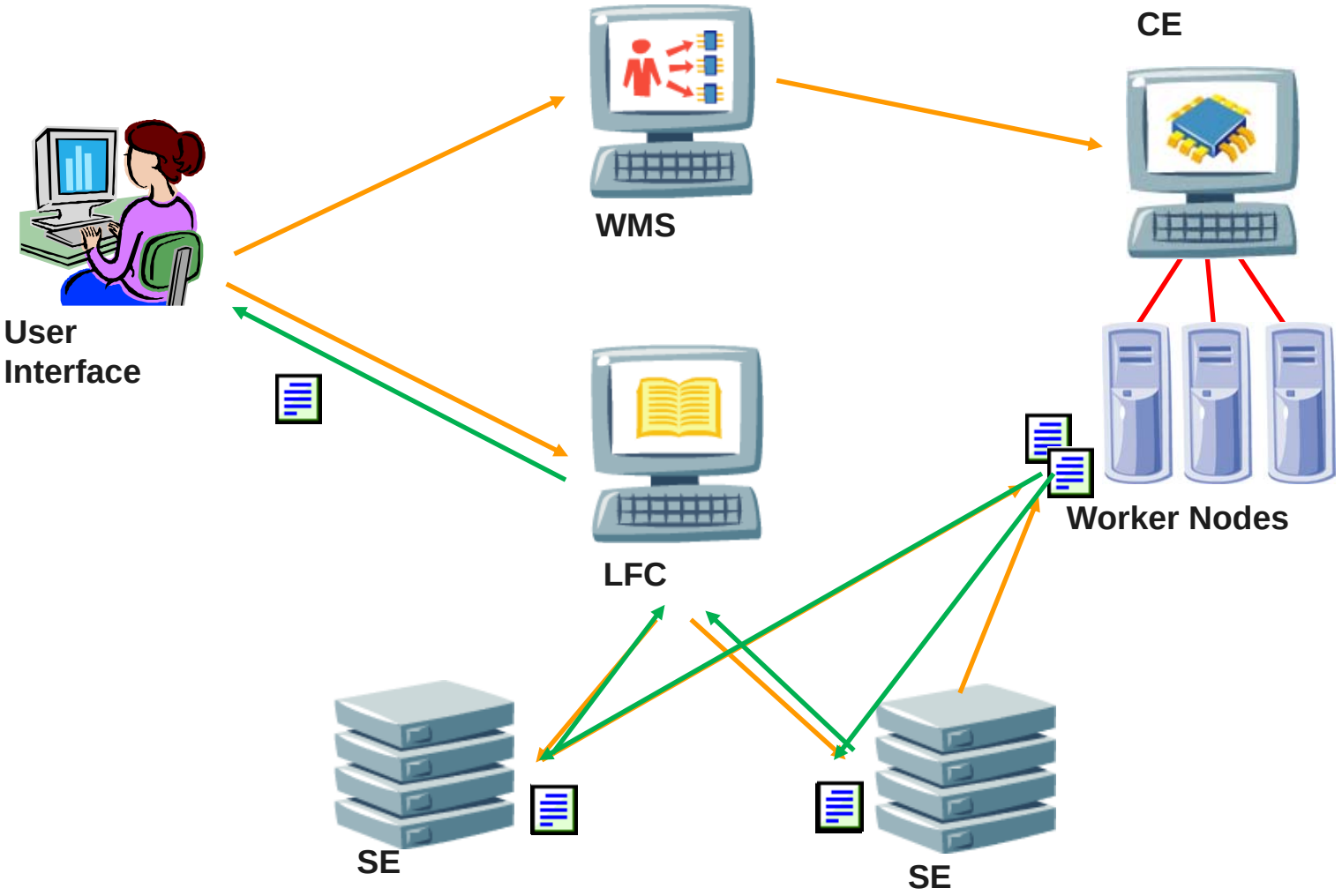
Input /Output du job sont de petite taille (max 10 MB)



Gestion des jobs sur la grille

Soumission d'un job – cas 2

Input /Output du job sont de grand taille (> 10 MB)



Gestion des jobs sur la grille

Préparation du job

- ✓ Se connecter à l'UI
- ✓ Créer un proxy (12H par défaut)

```
voms-proxy-init --voms <VO>
```

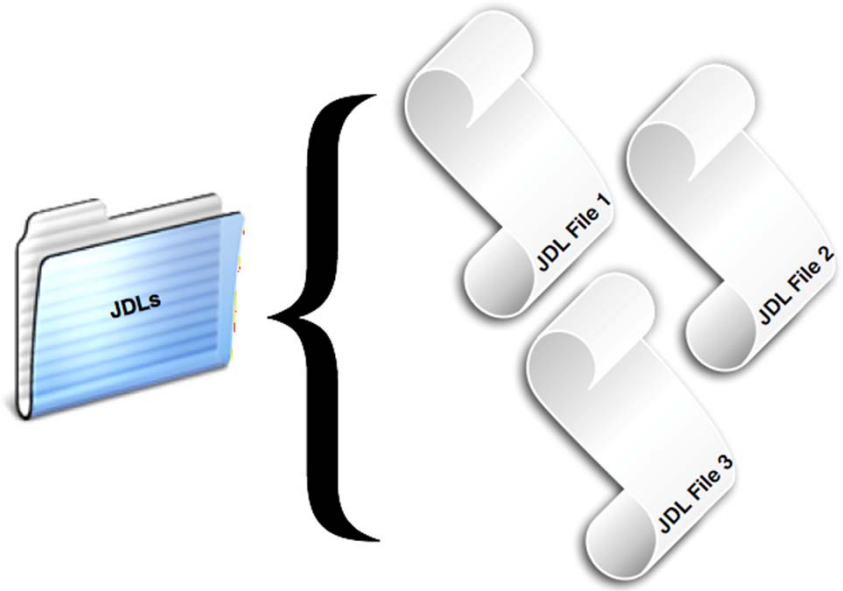
- ✓ Préparer les ingrédients pour l'exécution du Job :
  - Exécutable
  - Inputs
  - Le JDL avec tous les prérequis du Job sur les ressources
- ✓ Soumettre le Job
- ✓ Suivre l'état du Job
- ✓ Récupérer les résultat

Gestion des jobs sur la grille

Collection de jobs

glite-wms-job-submit -a --collection -o JobID JDLs/

- Soumissions simultanées
- Exécutions asynchrones
- Nombre quelconque de jobs

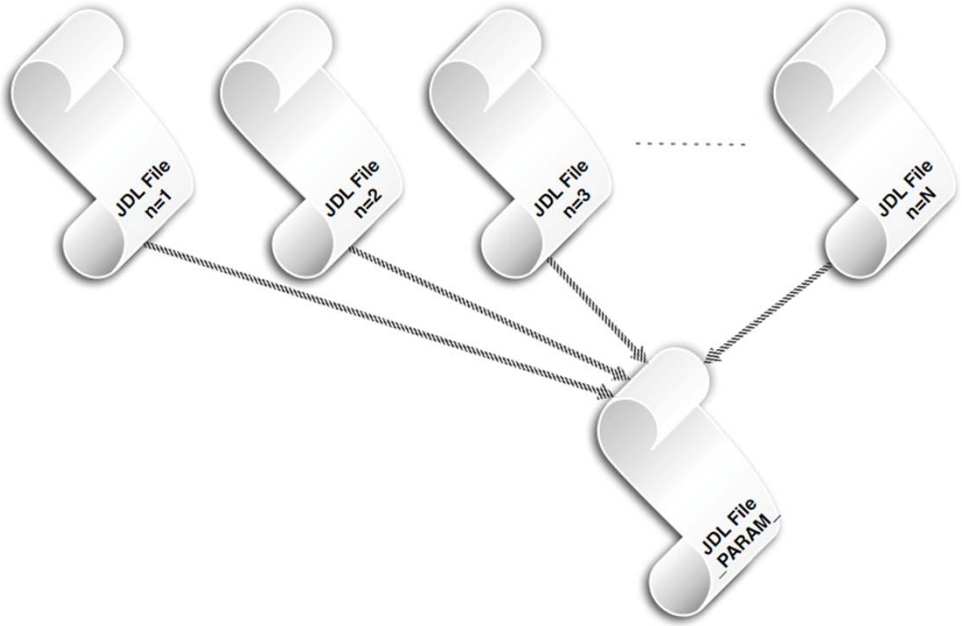


Gestion des jobs sur la grille

Job paramétrique

- Soumission de job multiple avec un **paramètres qui change** à chaque fois.

- Les valeurs du paramètres est défini:
  - Processus d'incrémentation
  - Une liste
- Les paramètres peuvent être:
  - Entiers
  - Réels
  - caractères or
  - Chaine de caractère
- Soumissions simultanées
- Exécutions asynchrones
- Excellente pour une étude à un seul paramètre



Gestion des jobs sur la grille

Job paramétrique

JDL 1

```
JobType = "Parametric";
Executable = ”run.sh";
Arguments = "_PARAM_";
InputSandbox = ”run.sh";
Parameters = 8;
ParameterStep = 2;
ParameterStart = 0;
StdOutput = "std_PARAM_.out”;
StdError = "std.err";
OutputSandbox  =
       {"std_PARAM_.out", "std.err”};
```

JDL 2

```
JobType = "Parametric";
Executable = ”run.sh";
Arguments = "_PARAM_";
InputSandbox = ”run.sh";
Parameters = {0,2,4,6,8,10,12,14};
StdOutput = "std_PARAM_.out”;
StdError = "std.err";
OutputSandbox  =
       {"std_PARAM_.out", "std.err”};
```

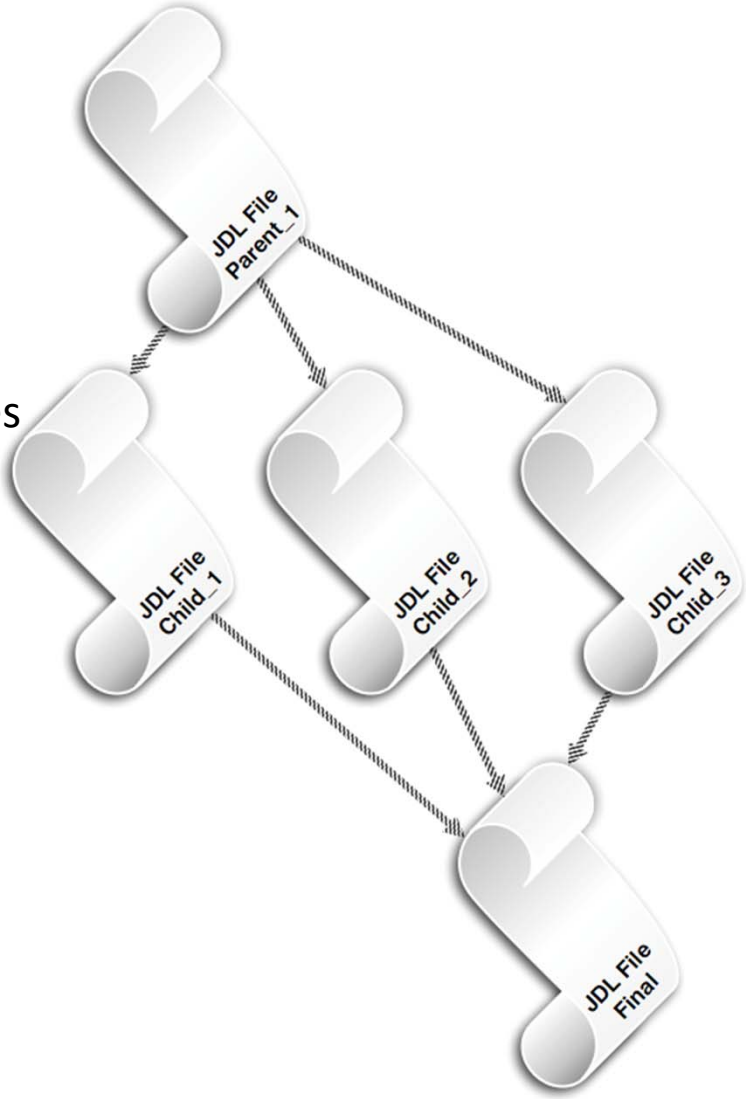
run.sh

```
#!/bin/bash
param=$1
echo $param
```

Gestion des jobs sur la grille

Job DAG

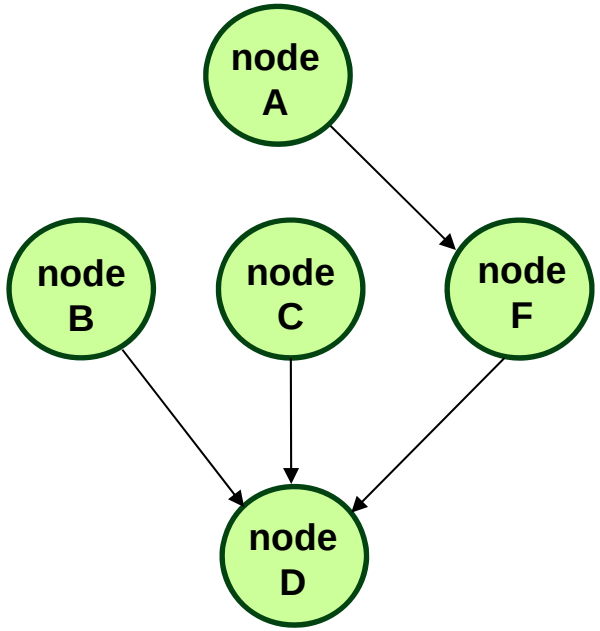
- Le job DAG (Directed Acyclic Graph) est un ensemble de **jobs interdépendants**.
- Les dépendances sont représentées par un graphe (DAG), où les nœuds représentent les jobs et les liaisons représentent les dépendances
- L'utilisateur doit mettre l'attribut **Type** à **dag**, l'attribut **Nodes** doit contenir la description des nœuds (les JDL) et l'attribut **Dependencies** doit définir les dépendances.
- Soumissions asynchrones
- Exécutions asynchrones



Gestion des jobs sur la grille

Job DAG

```
Type = "dag";
Nodes = [ nodeA = [
            file = "nodeA.jdl" ;
          ];
          nodeB = [
           file = "nodeB.jdl" ;
          ];
           ……
          nodeF = [
           file = "nodeF.jdl" ;
          ];
Dependencies = {{nodeA, nodeF},{{nodeF, nodeB, nodeC}, nodeD}}; ];
```



Gestion des jobs sur la grille

Job MPI

- **MPI** (Message Passing Interface), librairie uilisée pour gérer la communication entre processus parallèls
- MPI est devenu une approche très populaire pour écrire des applications parallèles
- Il existe plusieurs implémentation du MPI : MPICH2, openMPI,..etc
- gLite est compatible avec plusieurs implémentations MPI grace au mécanisme **MPI-START**
  - "Bootstrapping" d'un processus MPI est fait via un script (communément appelé **mpi-start.sh**).
  - Un autre script (communément appelé **mpi-hooks.sh**) peut être utilisé pour un **pre** ou un **post traitement** dans le même job.
  - Dans le fichier JDL confirmer l'usage du mécanisme **mpi-start** dans le job

Gestion des jobs sur la grille

Job MPI

Exemple: Compiler et executer un program hello.cpp sur 4 CPUs

hello.cpp

```
#include <iostream>
#include <mpi.h>

using namespace std;

int main(int argc, char* argv[])
{
        int rank,size;
        MPI_Init( &argc, &argv );
        MPI_Comm_rank( MPI_COMM_WORLD, &rank );
        MPI_Comm_size( MPI_COMM_WORLD, &size );
        cout << "Hello World! I am process " << rank
             << " out of " << size << "." << endl;
        MPI_Finalize();
        return 0;
}
```

Gestion des jobs sur la grille

Job MPI

mpi-start.sh

```
#!/bin/bash
MY_EXECUTABLE=`pwd`/$1
MPI_FLAVOR=$2

MPI_FLAVOR_LOWER=`echo $MPI_FLAVOR | tr '[:upper:]' '[:lower:]'`
eval MPI_PATH=`printenv MPI_${MPI_FLAVOR}_PATH`
eval I2G_${MPI_FLAVOR}_PREFIX=$MPI_PATH
export I2G_${MPI_FLAVOR}_PREFIX
touch $MY_EXECUTABLE
export I2G_MPI_APPLICATION=$MY_EXECUTABLE
export I2G_MPI_APPLICATION_ARGS=
export I2G_MPI_TYPE=$MPI_FLAVOR_LOWER
export I2G_MPI_PRE_RUN_HOOK=mpi-hooks.sh
export I2G_MPI_POST_RUN_HOOK=mpi-hooks.sh
chmod +x $MY_EXECUTABLE

$I2G_MPI_START
```

mpi-hooks.sh

```
pre_run_hook () {
   mpicxx -o   ${I2G_MPI_APPLICATION}   ${I2G_MPI_APPLICATION}.cpp
   return 0
}
post_run_hook () {
}
```

Gestion des jobs sur la grille

Job MPI

hello.jdl

```
JobType = "Normal";
CpuNumber = 4;
Executable = "mpi-start.sh";
Arguments = "hello MPICH2";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"mpi-start.sh","mpi-hooks.sh","src/hello.cpp"};
OutputSandbox = {"std.err","std.out"};
Requirements =
  Member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment)
   && Member("MPICH2", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Arguments are picked up by the wrapper script(s)

Gestion des jobs sur la grille

Sessions pratiques sur les jobs

http://wiki.marwan.ma

- **Authentification**

  Placer une demande de certificat électronique personnel au : ca.magrid.ma

- **Autorisation**

  Une fois le certificat est délivré et installé et importé dans le navigateur demander l'adhésion à une Organisation Virtuelle

  Exemple : VO magrid
  https://voms.magrid.ma:8443/voms/magrid/user/home.action

- **Compte sur une Interface Utilisateur**

  Une fois l'adhésion au VO est validée, un compte sur le UI vous sera crée par l'administrateur: ui.magrid.ma

- **Importer le certificat sur le UI**
- **Tester la création d'un VOMS proxy :** **voms-proxy-init --voms magrid**
- **Interroger le SI**
- **Soumettre les Jobs sur la Grille**