Soumission de jobs

Tout les fichiers nécessaires se trouvent dans l'archive tutorial_materiel.tgz.

---

1. Suivi d'un job "Hello World"

1. Créez un *proxy* à l'aide de la commande **`voms-proxy-init.`**
2. Soumettez le job *HelloWorld.jdl* en utilisant la commande **`glite-wms-job-submit.`**
   Les commandes du *« workload management »* analysent les *proxies* VOMS et utilisent la VO indiquée par ces derniers.
   On peut aussi spécifier la VO avec l'option `--vo`.
3. Vérifiez le statut du job en utilisant la commande **`glite-wms-job-status.`**
4. Lorsque le job est terminé, récupérez la sortie en utilisant **`glite-wms-job-output`**.
   Si le job se termine dans l'état « Aborted », c'est qu'il ne passe pas. On peut alors trouver plus d'informations avec la commande **`glite-wms-job-logging-info`**.
5. Vérifiez que tout s'est déroulé correctement en consultant les fichiers *message.txt* et *stderror*.

2. Fichier JDL : modification et édition

1. Modifiez le fichier *HelloWorld.jdl* de manière à ce qu'il n'appelle plus `/bin/echo` mais le script *HelloWorldScript.sh.*
   Pour cela :
   - la ligne « Executable » doit être « HelloWorldScript.sh »,
   - la ligne « Arguments » peut rester avec « Hello World »,
   - vous devez de plus définir le paramètre « InputSandbox » en ajoutant la ligne :
     `InputSandbox = {"HelloWorldScript.sh"};`
     On peut utiliser n'importe quel script, cependant le shell utilisé par le script doit exister dans le WN.

2. Modifiez de nouveau *HelloWorld.jdl* de manière à ce qu'il appelle cette fois l'exécutable *myhostname*. Vous pouvez visualiser la source de cet exécutable, qui est un programme C : *myhostname.c*. Vous n'avez cette fois pas besoin de définir d'argument. Il faut modifier la ligne « InputSandbox ».

   Sur quel WN a tourné votre job ?

3. L'exécution d'un programme en C compilé n'est pas forcément pratique : l'exécutable peut être d'une grande taille, dépendre de plusieurs fichiers, ou dépendre d'un environnement d'exécution particulier. Une solution consiste à compiler le programme directement sur le CE.

   Modifier une nouvelle fois *HelloWorld.jdl* de manière à ce qu'il appelle le script *buildandrun.sh*, avec pour argument « myhostname ». Testez ce script seul pour comprendre l'argument nécessaire.

   Votre job a-t-il tourné sur le même WN que précédemment ?

3. Pour comprendre les « Requirements » and « Rank »

Il y a deux clés importantes dans les fichiers JDL : « Requirements » et « Rank ».
Les valeurs pour les clés Requirements et Rank sont des expressions. Votre job va tourner uniquement sur une ressource qui a une valeur « true » pour l'expression Requirements. S'il y a plusieurs ressources qui ont une valeur « true », le système utilise l'expression Rank pour choisir la meilleure ressource.
La ressource qui a la plus grande valeur est choisie. S'il y a plusieurs ressources avec la même valeur Rank, la ressource utilisée est choisie aléatoirement entre ces ressources de même valeur Rank.

1. Plusieurs valeurs peuvent être utilisées pour définir les expressions Requirements et Rank. Par exemple, ajoutez l'expression ci-dessous dans le fichier *HelloWorld.jdl* pour choisir tous les sites qui permettent à un job d'utiliser plus d'une heure de temps CPU :

   ```
   Requirements = (other.GlueCEPolicyMaxCPUTime > 60);
   ```

   Pour voir la liste des ressources acceptables utilisez la commande **`glite-wms-job-list-match.`**
   Combien de ressources autorisent les jobs de plus d'une heure de temps CPU ? Plus de 2 heures ? Plus de 10000 minutes ?

2. Pour choisir des sites spécifiques, on peut utiliser le nom de la ressource. Pour choisir tous les sites en France, changez la valeur de Requirements comme suit :

   ```
   Requirements = RegExp(".*.fr:.*", other.GlueCEUniqueID) ;
   ```

   Le premier argument est une expression régulière.
   Modifiez la ligne Requirements pour choisir une ressource particulière. Trouvez la syntaxe correcte pour exclure un site.
   Une option `–r` existe pour la commande `glite-wms-job-submit` qui permet de choisir une ressource spécifique. Cependant cette option évite tout le processus de *MatchMaking* du RB et n'ajoute pas le fichier nécessaire (.BrokerInfo) pour la gestion des données. La technique avec other.GlueCEUniqueID est plus flexible et plus sure.

3. Ajoutez les lignes suivantes pour utiliser la ressource avec le plus grand nombre de CPU

libres :

```
Requirements = other.GlueCEStateFreeCPUs;
```

Utilisez la commande `glite-wms-job-list-match` pour visualiser le résultat. (L'ordre indique le Rank. La ressource la plus intéressante est la première)
Si on utilise la valeur -other.GlueCEStateFreeCPUs, quelle ressource le RB va-t-il choisir ?
Que fait le RB si on utilise la valeur rank = 1 ?

4. Environnement d'exécution sur le WN

Chaque utilisateur de la grille est mappé dans un compte local pour chaque site. Maintenant l'accès aux ressources locales est contrôlé par les droits de ce compte.

1. Visualisez le contenu du fichier JDL *whoami.jdl.* Lancez le job et récupérez la sortie. Visualisez le fichier *std.out.* Sur quel compte êtes-vous mappé ?
2. Visualisez le contenu du script *envvar.jdl.* Soumettez un job qui lance ce script sur la grille. Regardez la liste des variables. Combien de variables concernent la grille ?
3. Ecrivez un job qui liste les versions des logiciels disponibles dans le WN. On peut utiliser la commande **`rpm`** pour le faire.

5. Soumission d'un job MPI

Beaucoup des disciplines utilisent des jobs parallèles. MPI (Message Passing Interface) est un protocole qui permet la communication entre les tâches parallèles. Les jobs MPIs sont supportés dans la grille.

1. Regardez dans le fichier MPIHelloWorld.jdl. Il y deux paramètres spéciaux pour les jobs MPI : "JobType" et "NodeNumber". Le NodeNumber est le nombre de CPUs réellement utilisés par le job.
2. Regardez le script MPIHelloWorld.sh. Le script permet la compilation du code MPI, la vérification de l'existence de la liste des machines et le lancement du job en parallèle.
3. Lancez ce job MPI et vérifiez que le résultat dans std.out est bon.

Pour les jobs de production, on doit ajouter dans le code du script de quoi vérifier que le binaire est visible par tous les esclaves. Cela est nécessaire car le système de fichier partagé n'existe pas dans tous les sites. Il est nécessaire de faire un script plus élaboré pour utiliser le maximum de ressources.
La grille supporte les jobs MPI dans un seul site. MPI avec esclaves sur plusieurs sites n'est pas supporté.

6. Outils / Aide

6.1. Utilisation du paramètre "requirements" dans le JDL

Pour spécifier dans le JDL un site particulier pour la soumission :

```
Requirements=other.GlueCEUniqueID=="<CEHostname>:2119/<QueueName>"
```

6.2. Soumission d'un job en ligne de commande

```
$ glite-wms-job-submit --vo <voname> -a <file.jdl>
```

Si la soumission se passe bien, la sortie doit ressembler à :

```
Connecting to the service https://wms110.cern.ch:7443/glite_wms_wmproxy_server


====================== glite-wms-job-submit Success ======================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://lb106.cern.ch:9000/_ztr0Typ3CWv4HEdUekBpA

The job identifier has been saved in the following file:
/afs/in2p3.fr/home/d/dbouvet/public/tutorial_materiel/JobID

==========================================================================
```

6.3. Récupération du statut d'un job

```
~> glite-wms-job-status <jobID>
```

Selon le statut du job, la sortie doit ressembler à :

```
**************************************************************************

BOOKKEEPING INFORMATION:

Status info for the Job :
https://grid004.ct.infn.it:9000/ybnyOZ9O-5A-kanmY2zyEA
Current Status:     Submitted
reached on:         Thu Nov  4 08:19:36 2004
**************************************************************************
```

6.4. Récupération de la sortie d'un job

```
~> glite-job-output <jobID>
```

6.5. Liste des sites sur lesquels un job peut être soumis

```
~> glite-wms-job-list-match --vo vo.rocfr.in2p3.fr -a <file.jdl>
```