Manuel utilisateur APCScheduler v0.9

par Colley Jean-marc

Version du manuel : 1.0

1 Avertissement

La version 0.9 contient **une modification majeure dans le nommage des méthodes et des fonctions**. En effet, cette version se conforme à la règle de nommage qui veut que les méthodes et fonctions commence par une minuscule, réservant la majuscule aux noms des classes. La conséquence immédiate est que **les scripts écrits dans la version 0.8 et précédentes ne sont pas compatibles avec la 0.9**, cependant seul la casse de la première lettre change, la mise à jour est facile.

Fondamentalement l'esprit de la librairie v0.9 reste le même que pour la version v0.8, la documentation de la v0.8 reste encore la référence tant que le portage du manuel écrit dans un wiki et sauvegardée en pdf n'est pas terminé.

Ce court manuel ne présentera que les modifications et les nouvelles fonctionnalités par rapport à la v0.8

2 Modifications

2.1 Gestion du proxy avec le scheduler gLite

APCScheduler teste l'expiration du proxy avant le lancer un job. S'il reste moins de 48 heures, il propose de réinitialiser le proxy avec une nouvelle valeur supérieure à 48 heures ou de ne rien faire.

2.2 Classe Application

2.2.1 Méthode `setOutput()`

L'utilisateur peut imposer son identifiant qui rendra unique les noms des fichiers de sorties:

```
   def setOutput(self, List, AddID=True):
   """[Specific grid]
   List : Define output file List to retrieve at the end of job via OutputSandbox
and copy in job repository
   AddID : add prefixe identificator
       AddID==True : prefix is SchedulerID random number and letter
       AddID is type String : prefix is value of AddID
       else : no ID
   """
```

2.2.2 Méthode `setOutputSE()`

Même chose que pour `setOutput()`

3 Nouvelles fonctionnalités

3.1 Scheduler GE : classe `SchedulerSGE`

Le nouveau scheduler du centre de calcul de l'IN2P3 est disponible. Il s'agit du Sun Grid Engine. Le mot clef associé à la fonction `newJob()` est `GE`. Pour éviter les blocages en queue d'un job qui demande trop de ressources CPU ou mémoires, la classe SchedulerSGE effectue un test sur ces ressources avant de soumettre le job. Les limites correspondent à la classe `huge`. De plus pour les jobs parallèles APCScheduler détermine la queue associée aux ressources demandés parmi :

- `pa_short`
- `pa_medium`
- `pa_long`

Car contrairement aux jobs séquentiels le nom de la queue doit être explicitement indiqué pour un job parallèle.

3.2 Scheduler gLite : prise en charge des jobs MPI

APCScheduler mets en place le mécanisme MPI-START décrit ici:

- http://egee-uig.web.cern.ch/egee-uig/beta_pages/MPIJobs.html

On peut lister les Compute Element hébergeant cette fonctionnalité avec la commande:

```
apcui01:>lcg-info --vo vo.apc.univ-paris7.fr --list-ce --query 'Tag=*MPI-START*'
- CE: apcce01.in2p3.fr:2119/jobmanager-pbs-apc
- CE: apcce02.in2p3.fr:8443/cream-pbs-apc
- CE: grid10.lal.in2p3.fr:2119/jobmanager-pbs-apc
- CE: grid36.lal.in2p3.fr:8443/cream-pbs-apc
- CE: ipngrid04.in2p3.fr:8443/cream-pbs-apc
- CE: ipnls2001.in2p3.fr:2119/jobmanager-pbs-apc
- CE: node16.datagrid.cea.fr:2119/jobmanager-pbs-apc
```

3.2.1 Exemple avec les commandes apcgrid-xxx

On commence par créer le tarball avec les sources de l'application MPI et le script mpi-start-wrapper.sh fourni avec APCScheduler 0.9:

```
apcgrid-dotarball fakeExe -t se:colley/tarball/testmpi2 -s MainScript.sh -i "mpiHello.cpp mpi-start-wrapper.sh" -k
```

arguments:

1. fakeExe: la procédure de création de tarball n'a pas été prévu pour un code source, à terme cela disparaitra
2. -t se:colley/tarball/testmpi2: c'est le chemin du tarball sur le SE, à adapter à votre arborescence, utilisation de apcgrid-ls, apcgrid-mkdir
3. -s MainScript.sh : c'est le script principal il lance par défaut mpi-start-wrapper.sh avec l'application mpiHello avec les bonnes options
4. -i "mpiHello.cpp mpi-start-wrapper.sh" : c'est l'input du tarball , le fichier source donc et le script MPI-START : mpi-start-wrapper.sh
5. -k pour garder un tarball sur l'UI

MainScript.sh:

```
#!/bin/bash
```

```
echo "I'm main script of tarball"

if [ -r MainScriptSandbox.sh ]; then
  echo "call MainScriptSandbox.sh"
  MainScriptSandbox.sh
  exit $?
else
  # default
  echo $*
  mpi-start-wrapper.sh mpiHello OPENMPI $*
fi
```

Lancement du job :

```
apcgrid-run se:colley/tarball/testmpi2 --mpi_cpu 100 -i mpi-hooks.sh --ui_out DirOut07 --em
```

Commentaires sur les arguments:

1. se:colley/tarball/testmpi2, le tarball produit précédemment
2. –mpi_cpu 100: le nombre de CPU demandé
3. -i mpi-hooks.sh: c'est une entrée, dans ce fichier on définit comment compiler l'application avant exécution et que ce l'on fait à la fin de l'application comme posté les fichiers résultat dans le répertoire boite à lettre toUI. Cela peut être aussi une liste qui inclut un fichier de paramètres, exemple: -i "mpi-hooks.sh file.par"
4. –ui_out DirOut07: répertoire de sortie
5. –em : envoie un mail à la fin du job

Le fichier mpi-hooks.sh dédié au mpiHello.cpp:

```
#!/bin/sh
#
# This function will be called before the MPI executable is started.
# You can, for example, compile the executable itself.
#

pre_run_hook () {
  # Compile the program.
  #ls -trl
  echo $HOSTNAME
  echo "Compiling ${I2G_MPI_APPLICATION}"

  # Actually compile the program.
  cmd="mpic++ ${MPI_MPICC_OPTS} -o ${I2G_MPI_APPLICATION} mpiHello.cpp"
  echo $cmd
  $cmd
  if [ ! $? -eq 0 ]; then
    echo "Error compiling program. Exiting..."
    exit 1
  fi

  # Everything's OK.
  echo "Successfully compiled ${I2G_MPI_APPLICATION}"
  return 0
}
```

```
#
# This function will be called before the MPI executable is finished.
# A typical case for this is to upload the results to a storage
#element.
#
post_run_hook () {
  cp mpi-start-wrapper.sh toUI/
  cp mpiHello.cpp toUI/
  return 0
}
```

APCScheduler demande dans le fichier JDL un CE où se trouve la fonctionnalité MPI-START.