Documentation APCScheduler version 0.8

Retour doc générale

Sommaire

Installation

à l'APC

APCScheduler est disponible depuis l'UI apcui01/apcui02, il suffit de modifier la variable d'environnement PYTHONPATH:

```
export PYTHONPATH=$PYTHONPATH:/Net/se/vo.apc.univ-paris7.fr/comput/GAP/python/0.8
```

Modifier également votre PATH pour les commandes unix apcgrid−xxx

```
export PATH=$PATH:/Net/se/vo.apc.univ-paris7.fr/comput/GAP/python/0.8
```

Hors de l'APC

Récupérer une archive APCScheduler version 0.8, mail à colley at apc.univ−paris7.fr. Modifier votre variable d'environnement PYTHONPATH en ajoutant le chemin où se trouve le fichier APCScheduler.py. Modifier votre variable d'environnement PATH en ajoutant le chemin où se trouvent les commandes apcgrid−xxx

Les commandes unix apcgrid−xxx

Les commandes unix apcrgrid−xxx permettent d'utiliser les fonctionnalités de base de la librairie python APCScheduler pour le scheduler grille/gLite sans connaître le langage python. Un job est soumis à la grille en une ligne avec la commande apcgrid−run. Les commandes apcgrid−xxx sont bien sûr utilisables dans un shell script. Liste des commandes disponibles:

- apcgrid−init
- apcgrid−run
- apcgrid−dotarball
- apcgrid−ls
- apcgrid−mkdir
- apcgrid−cp
- apcgrid−rm

apcgrid−init

Il est nécessaire de vérifier que la configuration de votre UI et VO sont compatibles avec la librairie python APCScheduler et initialiser quelques variables au passage. Cette commande doit être lancée avant la première utilisation des commandes apcgrid−xxx et d'APCScheduler en mode grille et à chaque fois que vous voulez changer de VO ou une option. apcgrid−init effectue les taches suivantes:

- enregistrement de la VO
- vérification de la disponibilité de WMS
- vérification du proxy
- vérification de la disponibilité d'un élément de stockage (SE) et création d'un répertoire temporaire pour APCScheduler sur le SE
- définition d'un élément de stockage par défaut
- enregistrement d'une adresse mail pour annoncer la fin des jobs

Si l'initialisation est réussie, apcgrid−init écrit 2 fichiers de configuration dans le répertoire .apcgrid dans votre home.

apcgrid−run

Hello grid

La commande unix apcgrid−run permet de lancer un job sur la grille, exemple:

```
apcgrid-run echo -p 'Hello grid from CE $HOSTNAME'
```

Par défaut vous récupérez l'output dans le répertoire où vous avez lancé la commande (voir le chapitre "Nommage des fichiers"), l'option −−ui_out permet de définir un répertoire de sortie, avec l'option −−em vous recevrez un mail dont le corps contiendra la sortie standard

```
apcgrid-run echo -p 'Hello grid from CE $HOSTNAME' --em --ui_out OutTest
```

Une description de toutes les options est disponible avec l'option −h.

Le premier argument est le nom d'un exécutable, apcgrid−run a un comportement différent s'il s'agit d'un script ou d'un éxecutable binaire compilé, voyons ces 2 modes.

Mode script

Dans ce mode le script et les données d'entrées seront transmis via l'InputSandbox qui a une taille limitée. Le script peut être par exemple un déclencheur de parachute.

Mode binaire

Dans ce mode la technique du parachute est mise en place, ie l'exécutable, ses librairies, ses fichiers et répertoires d'entrées seront réunis dans un tarball qui sera copié dans un répertoire temporaire sur le SE. apcgrid−run enverra un script déclencheur de parachute dans l'InputSandbox. De plus, si l'éxecutable ou le script principal, option −s (voir chapitre 5.3) écrivent dans le répertoire toUI du tarball, son contenu sera rappatrié vers l'UI.

Exemple :

```
apcgrid-run ExeCompile config.par -s ScriptPrincipal.sh --ui_out GridOut
-i "config.par Propagation" --ce apcce01.in2p3.fr --em
```

Dans cette exemple, le script ScriptPrincipal.sh sera exécuté sur la grille et comme l'option −−ce est présente le job sera envoyé sur une machine du Comput Element apcce01.in2p3.fr. Le fichier de paramètre doit être passé en input avec l'option −i. S'il y a plusieurs entrées, encadrés−les par des doubles cotes comme dans l'exemple. Les répertoires sont acceptés comme valeur d'entrée et copiés récursivement. La sortie standard et erreur ainsi que tous les fichiers écrits dans le répertoire toUI du tarball seront copiés dans le répertoire GridOut. Enfin l'option −−em demande l'envoie d'un mail à la fin du job.

apcgrid−dotarball

Permet de créer un tarball à partir d'un binaire et de le copier sur le SE.

```
apcui01:~>apcgrid-dotarball
usage: apcgrid-dotarball executable -t se:user/myTarBall [options]

Help:
```

```
apcgrid-dotarball -h

Comment:
apcgrid-dotarball do a tarball can be call by apcgrid-run

Example, do tarball with 'testConvolFFT' binary executable,used
'script_loop.sh' to call it and save it on SE 'user/tarballtest'
apcgrid-dotarball testConvolFFT -t se:user/tarballtest -s script_loop.sh
```

Le tarball créé ainsi est utilisable par apcgrid−run:

```
apcgrid-run se:user/tarballtest -p "10" --em --ui_out /home/colley/Grid/Run/test/03
```

apcgrid−ls

Affiche les fichiers du catalogue, exemple :

```
apcui01:~>apcgrid-ls
drwxrwxr-x   0 75       10539                    0 Dec 18  2007 atelier
drwxrwxr-x   5 155      10539                    0 Apr 29 11:58 colley
drwxrwxr-x  11 68       10539                    0 Jul 08  2008 generated
drwxrwxr-x   5 71       10539                    0 Jan 18 15:57 public

apcui01:~>apcgrid-ls colley
drwxrwxr-x   0 155      10539                    0 Apr 29 12:40 APCScheduler
-rw-rw-r--   1 155      10539                  502 Jun 09  2009 HelloGrid.py
drwxrwxr-x  10 155      10539                    0 Mar 20 19:32 exe_output
drwxrwxr-x   1 155      10539                    0 Apr 26 17:39 tarball
```

apcgrid−mkdir

apcgrid−mkdir crée un répertoire dans le catalogue:

```
apcui01:~>apcgrid-mkdir  colley/test

apcui01:~>apcgrid-ls  colley
drwxrwxr-x   0 155      10539                    0 Apr 29 12:40 APCScheduler
-rw-rw-r--   1 155      10539                  502 Jun 09  2009 HelloGrid.py
drwxrwxr-x  10 155      10539                    0 Mar 20 19:32 exe_output
drwxrwxr-x   1 155      10539                    0 Apr 26 17:39 tarball
drwxrwxr-x   0 155      10539                    0 Apr 29 13:03 test
```

apcgrid−cp

apcgrid−cp copie de l'UI vers un SE et d'un SE vers l'UI. Un chemin relatif au SE est préfixé par 'se:'. La copie récursive est possible ainsi que le filtrage par une expression régulière.

```
apcui01:~>apcgrid-cp temp se:colley/test -r -e '.*.ps'

cp temp/Res_CorFunc.ps
cp temp/Res_ModArg.ps
cp temp/Res_Mod.ps
cp temp/Res_ReIm.ps
cp temp/simuplanck.ps
```

```
guid:c8229dd9-6e64-4da0-a9eb-ecac738e21f0
guid:231466ae-9381-490d-bdb4-a339881852d7
guid:37014b07-f324-48d2-9e67-dd108908b421
guid:8c080b75-2d67-425c-90ce-bdb14bb80cb9
guid:afc06207-ac8d-4a64-b530-0dbc625c939f

apcui01:~>apcgrid-ls  colley/test
-rw-rw-r--   1 155      10539               897971 Apr 29 13:11 Res_CorFunc.ps
-rw-rw-r--   1 155      10539               560021 Apr 29 13:11 Res_Mod.ps
-rw-rw-r--   1 155      10539               555922 Apr 29 13:11 Res_ModArg.ps
-rw-rw-r--   1 155      10539               823566 Apr 29 13:11 Res_ReIm.ps
-rw-rw-r--   1 155      10539               263809 Apr 29 13:11 simuplanck.ps
```

apcgrid–rm

Permet de supprimer des fichiers du catalogue/SE. La suppression peut se faire de manière récursive option –r et sur une expression régulière option –e:

```
apcui01:~>apcgrid-ls  colley/test
-rw-rw-r--   1 155      10539               897971 Apr 29 13:11 Res_CorFunc.ps
-rw-rw-r--   1 155      10539               560021 Apr 29 13:11 Res_Mod.ps
-rw-rw-r--   1 155      10539               555922 Apr 29 13:11 Res_ModArg.ps
-rw-rw-r--   1 155      10539               823566 Apr 29 13:11 Res_ReIm.ps
-rw-rw-r--   1 155      10539               263809 Apr 29 13:11 simuplanck.ps
apcui01:~>apcgrid-rm  colley/test -e '.*Mod.*'

rm colley/test/Res_Mod.ps
rm colley/test/Res_ModArg.ps

apcui01:~>apcgrid-ls  colley/test
-rw-rw-r--   1 155      10539               897971 Apr 29 13:11 Res_CorFunc.ps
-rw-rw-r--   1 155      10539               823566 Apr 29 13:11 Res_ReIm.ps
-rw-rw-r--   1 155      10539               263809 Apr 29 13:11 simuplanck.ps
apcui01:~>apcgrid-rm  colley/test -r

rm colley/test/Res_CorFunc.ps
rm colley/test/Res_ReIm.ps
rm colley/test/simuplanck.ps
rmdir se:colley/test

apcui01:~>apcgrid-ls  colley
drwxrwxr-x   0 155      10539                   0 Apr 29 12:40 APCScheduler
-rw-rw-r--   1 155      10539                 502 Jun 09  2009 HelloGrid.py
drwxrwxr-x  10 155      10539                   0 Mar 20 19:32 exe_output
drwxrwxr-x   1 155      10539                   0 Apr 26 17:39 tarball
```

Présentation de la librairie python APCScheduler

Principales fonctionnalités

Fonctionnalités générales

1. APCScheduler propose une interface Python pour soumettre et gérer un job via un ordonnanceur (scheduler en anglais) avec une syntaxe très proche. L'installation est très facile : un unique fichier

python !. Les ordonnanceurs disponibles sont :
    1. sur une grille de calcul utilisant GLITE
    2. sur un cluster avec BQS, scheduler du centre de calcul de l'IN2P3
    3. en local avec UNIX
2. Envoie d'un mail à la fin du job donnant son statut ainsi que le fichier stdout
3. Soumission d'un ensemble de jobs, possibilité de réguler le nombre de job RUNNING pour ménager les ressources.
4. L'ensemble des fichiers produits sont déposés dans un répertoire de travail défini par l'utilisateur
5. L'arrêt du script APCScheduler annule tous les jobs en cours (avec CRTL+C par exemple)

Ordonnanceur grille de calcul gLite

Les fonctionnalités mises en place permettent de suppléer aux difficultés suivnates d'utilisation de la grille:

- taux de réussite d'un job moins important que sur un cluster => re−soumission de jobs submit NOK, Aborted ou ne passant jamais en mode RUNNING.
- absence d'installation de librairie spécifique à un exécutable => technique du parachute et tarball automatique copiant les librairies présentes sur l'UI nécessaires à l'exécutable
- un CE ne peut pas écrire les fichiers résultats directement sur espace disque local à l'UI => système de répertoire boite aux lettres dont le contenu est copié du CE vers l'UI via le SE.

Liste des fonctionnalités:

1. La VO est configurable
2. Création automatique du fichier JDL et de configuration
3. Vérification de la validité du proxy et lancement de commande proxy−init si nécessaire
4. Rapatriement des fichiers définis comme 'output' (via OutputSandbox ou SE)
5. Utilisation de tous les WMS pour soumettre les jobs ce qui permet de diminuer le temps de soumission.
6. Possibilité d'exclure ou d'inclure des WMS.
7. Re−soumission d'un job sur un autre WMS si submit NOK ou Aborted ou s'il ne passe pas en mode RUNNING après une durée déterminée
8. Algorithme de répartition des jobs sur les CE sélectionnés via un requête lcg
9. Mise en place de la technique du parachute avec rapatriement automatique de fichiers vers l'UI
10. Classe simplifiant la manipulation des fichiers et répertoires sur un storage element
11. Délégation de proxy

Ordonnanceur BQS du centre de calcul IN2P3 à Lyon

1. Création automatique d'un script BQS de soumission sur anastasie ou pistoo
2. Prise en charge des programmes parallèles compilés avec OpenMPI
3. Prise en charge des scripts python

Conventions

Répertoire de travail

L'utilisateur définit avec la fonction suivante

```
SetRepository('/home/toto/RunGrid')
```

un répertoire de travail où tous les fichiers produits par le job ou APCScheduler seront déposés. Si le répertoire n'existe pas il sera créé. Les fichiers définis comme 'output' avec les méthodes SetOutput() et SetOutputSE() seront copiés dans ce répertoire via la récupération de l'OutputSandbox ou une copie depuis un Storage Element. Si l'utilisateur ne définit pas de répertoire de travail avec cette fonction, le répertoire de travail par défaut sera le répertoire d'où est lancé le script APCScheduler.

Nommage des fichiers

Chaque job reçoit un identificateur unique, il est composé du nom de l'application et de 6 caractères aléatoires pour garantir avec une grande probabilité son unicité, exemple : echo_jHKfJ0. Par défaut chaque fichier d'un job est préfixé par cet identificateur. Exemple:

```
apcui01:~/xxx>ls -trl ScriptTest_o8GZnn.*
-rw-r--r--  1 colley comput   659 Dec 22 17:33 ScriptTest_o8GZnn.jdl
-rw-r--r--  1 colley comput   282 Dec 22 17:33 ScriptTest_o8GZnn.conf
-rw-r--r--  1 colley comput     0 Dec 22 17:33 ScriptTest_o8GZnn.sub_err
-rw-r--r--  1 colley comput    80 Dec 22 17:33 ScriptTest_o8GZnn.id
-rw-r--r--  1 colley comput   493 Dec 22 17:33 ScriptTest_o8GZnn.sub_out
-rw-r--r--  1 colley comput    56 Dec 22 17:40 ScriptTest_o8GZnn.stdout
-rw-r--r--  1 colley comput     0 Dec 22 17:40 ScriptTest_o8GZnn.stderr
-rw-r--r--  1 colley comput    42 Dec 22 17:40 ScriptTest_o8GZnn_out0.txt
-rw-r--r--  1 colley comput    42 Dec 22 17:40 ScriptTest_o8GZnn_out1.txt
-rw-r--r--  1 colley comput 22687 Dec 22 17:40 ScriptTest_o8GZnn_graphe.jpg
```

Dans l'ordre nous avons:

- le fichier JDL
- le fichier de configuration
- le stderr retourné par la commande submit
- l'identificateur du job dans la grille
- le stdout retourné par la commande submit
- le stderr retourné par le job
- le stdout retourné par le job
- les 3 derniers sont des fichiers 'output', le nom de création avant leur recopie étaient: out0.txt, out1.txt, graphe.jpg.

L'ajout du préfixe est optionnel pour les fichiers output, voir méthode SetOutput() dans la chapitre Applications.

Conception orientée objet

APCScheduler considère un job comme l'association d'une application et d'un scheduler. La classe de base d'une application est **Application** et celle d'un scheduler est **SchedulerAbstract**. La classe définissant un job est **JobClass**, un objet job est créé en donnant un objet Application et un objet Scheduler au constructeur:

```
class JobClass:
```

```
    def __init__(self, MyAppli, MySched):
```

Cette approche garantit une certaine souplesse d'évolution des fonctionnalités et de l'interface. Voici les classes dérivées d'**Application** et **SchedulerAbstract**:

```
CLASSES
    Application
        AppliExe
            AppliParachute
            AppliPython

    SchedulerAbstract
        SchedulerCluster
            SchedulerBQS
        SchedulerGrid
            SchedulerGLITE
        SchedulerLOCAL
```

La classe JobClass mémorise l'application et le scheduler dans ces 2 attributs:

- _Appli
- _Scheduler

La création d'un job s'effectue de 2 manières différentes. La plus simple est en 2 temps:

1. Création d'un objet de la classe **AppliExe** dérivée de **Application**
2. Création d'un job avec la fonction **NewJob(MyExe,'GLITE')** avec en paramètre un objet exécutable et un mot clé qui désigne un scheduler.

```
MyExe = AppliExe('echo')
MyExe.SetArg('Hello APCgrid')
Job = NewJob(MyExe,'GLITE')
```

Ou d'une manière plus concise

```
Job = NewJob(AppliExe('echo'),'GLITE')
Job._Appli.SetArg('Hello APCgrid')
```

Dans cet exemple AppliExe est une classe dérivée de **Application** qui prend en charge les applications du type exécutable.

```
La seconde s'effectue en 3 temps
```

1. Création d'un objet application d'une classe dérivée de **Application** et sa spécialisation
2. Création d'un objet scheduler d'une classe dérivée de **SchedulerAbstract** et sa spécialisation
3. Création d'un objet job de la classe JobClass

```
MyExe = AppliExe('echo')
MyExe.SetArg('Hello APCgrid')
MySched = SchedulerGLITE()
Job = JobClass(MyExe, MySched)
```

Dans cet exemple SchedulerGLITE() est une classe dérivée de **SchedulerAbstract** qui s'occupe de soumettre le job avec les commandes GLITE. La spécialisation d'un objet application est décrite dans le chapitre *Applications* et dans le chapitre *Schedulers* pour un objet scheduler. Mais examinons déjà un exemple où l'on

voit comment définir les arguments associées à un exécutable (SetArg()) et quelques méthodes de la classe **JobClass** (Submit(), Wait(), SendMailResult()) qui seront abordées dans le chapitre *Soumettre une application*.

Exemple : Hello grid

Voici le script HelloGrid.py

```
#! /usr/bin/python

from APCScheduler import *

# 0- define some env variables
SetRepository('/home/toto/RunGrid')
SetVerboseLevel(2)

# 1- Define my exe
MyExe = AppliExe('echo')
MyExe.SetArg('Hello APCgrid demo')

# 2- exe + scheduler = job
Job = NewJob(MyExe,'GLITE')
Job.Submit()
Job.Wait()
Job.SendMailResult()
```

Pour l'exécuter il suffit de lancer

```
apcui01:~>python HelloGrid.py
```

ou bien si HelloGrid.py a les droits d'un exécutable plus simplement

```
apcui01:~>HelloGrid.py
```

puisque le script commence par :

```
#! /usr/bin/python
```

Cela va produire le log suivant:

```
apcui01:~/example>python HelloGrid.py
proxy timeleft:  0:00:00 0
Enter GRID pass phrase for this identity:
Your identity: /O=GRID-FR/C=FR/O=CNRS/OU=APC/CN=Jean-Marc Colley
Creating temporary proxy ..................................................... Done
Contacting  grid12.lal.in2p3.fr:20010 [/O=GRID-FR/C=FR/O=CNRS/OU=LAL/CN=grid12.lal.in2p3.fr]
 "vo.apc.univ-paris7.fr" Done
Creating proxy ..................................... Done
Your proxy is valid until Sat Oct 18 18:18:02 2008
proxy timeleft:  24:00:00 86400
Proxy OK
Submit echo_jHKfJ0 is ok.
echo_Qq0fTL status is Submitted
echo_Qq0fTL status is Waiting
echo_Qq0fTL status is Ready
echo_Qq0fTL status is Scheduled
echo_Qq0fTL status is Running
```

```
echo_Qq0fTL status is Done
```

Aide python

L'aide python est accessible de la manière suivante:

```
apcui01:~>python
Python 2.3.4 (#1, Jun  4 2007, 16:38:03)
[GCC 3.4.4 20050721 (Red Hat 3.4.4-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import APCScheduler
>>> help(APCScheduler)
```

Elle permet d'obtenir l'arbre d'héritage des classes, des informations sur les fonctions et des méthodes de toutes les classes présentes dans le module APCScheduler.py. Pour la description d'une classe en particulier taper

```
>>> help(APCScheduler.JobClass)
```

qui donne toutes les méthodes disponibles pour un job. **Rappel**: Pour sortir de l'help 'q', pour sortir de python 'CRTL+d'.

Quelques fonctions utiles

Fonctions de configuration

- **SetMail(*mail*)** : la fonction SendMailFile() et les méthodes SendMailResult() envoient un mail à l'adresse passée en paramètre de cette fonction. Il est possible de mettre plusieurs adresses mail séparées par des virgules sans espace. apcgrid−init mémorise une adresse mail qui sera utilisée par défaut, mais c'est disponible uniquement sur une UI.
- **SetRepository(*rep*)** : *rep* de format chaine de caractères, défini le répertoire de travail, si nécessaire le répertoire est créé automatiquement. En absence de cette fonction le répertoire de travail par défaut est le répertoire depuis lequel est lancé le script.
- **SetVerboseLevel(*int*)** : défini le niveau de verbosité d'APCScheduler, de 0 à 20, cela permet de voir les messages de débug.
- **AddLIBRARYPATH(*cheminLibs*)** : ajout à LD_LIBRARY_PATH le chemin *cheminLibs*
- **SubmitCmd(*cmd*, *TpsMax*=3600, *verbose*=True)**: soumets la commande unix *cmd* et tue le process si nécessaire au bout d'un delais *TpsMax* de manière à ce que cette commande ne soit pas bloquante
- **SetFileConfVO(*file*)**: pour définir un fichier de configuration apcgrid/gLite autre que celui par défaut écrit dans ~/.apcgrid par apcgrid−init.

Manipulation de fichiers et de répertoires sur le storage element

La classe **SEtools** permets de copier et d'effacer fichier et répertoire entre UI et le SE

```
class SEtools
 |  Methods defined here:
 |
 |  __init__(self, NameVO, NameSE)
```

```
|
|  cp(self, src, dest)
|      File on storage element must begining by se:
|
|  cpdir(self, src, dest, prefixe='')
|      Directory on storage element must begining by se:
|
|  cprec(self, src, dest, re, rec, nbstream, check)
|      File on storage element must begining by se:
|
|  isDir(self, path)
|
|  ls(self, dir='', retry=False)
|
|  rm(self, file)
|
|  rmdir(self, dir)
|
|  sizeDir(self, dir)
|
|  sizeFile(self, file)
```

Afin de permettre l'utilisation des storage elements, ces 2 variables d'environnement doivent être présentes

- LFC_HOST
- LCG_CATALOG_TYPE

Par exemple ajouter ceci à votre script d'initialisation bash:

```
export LFC_HOST=grid14.lal.in2p3.fr
export LCG_CATALOG_TYPE=lfc
```

Autres fonctions

- **SendMailFile(*file*)** : envoie par mail le fichier texte *file*
- **NewJob(*MyAppli*, *scheduler*)**: créé un job avec un object application *MyAppli* et utilise le scheduler défini par un mot clé 'GLITE', 'BQS','LOCAL'. Le paramètre *scheduler* est optionel la valeur par défaut étant 'LOCAL'
- **GetVersion()**: donne la version de la librairie APCScheduler

Applications

Dans cette version APCScheduler seule les applications exécutables sont disponibles, mais il peut s'agir d'un exécutable compilé, d'un shell script ou bien script python.

Exécutable compilé ou shell script

La classe **AppliExe** dérive de la classe Application. Un object de ce type est utilisé dans le l'exemple *HelloGrid.py* avec l'exécutable unix echo passé en paramètre lors de la création de l'objet:

```
# 1- Define my exe
MyExe = AppliExe('echo',label='test echo')
```

```
MyExe.SetArg('Hello APCgrid demo')
```

Il n'est pas nécessaire de mettre le chemin absolu pour accéder à l'exécutable s'il est dans votre PATH. Il est possible de nommé l'application avec le paramètre optionel 'label' dans constructeur de toutes les classes dérivant d'Application

Exécutable python

Un exécutable python commence par la ligne suivante **"#! /usr/bin/python"**, exemple:

```
>more HelloGrid.py
#! /usr/bin/python

from APCScheduler import *
# 0- define some env variables
SetRepository('/home/colley/Grid/RunSubmit/run62')
...
```

et a les droits d'un exécutable

```
>ll HelloGrid.py
-rwxr-xr-x  1 colley comput 546 Jan 14 14:09 HelloGrid.py
```

La classe **AppliPython** gère les exécutables Python et elle dérive de AppliExe

```
# 1- Define my exe
MyExe = AppliPython('test/testSchedulerLOCAL.py')

# 2- exe + scheduler = job
Job = NewJob(MyExe)
```

Binaire pour la grille et technique du parachute

La classe **AppliParachute** met en place la technique du parachute pour un exécutable binaire. Cette classe est donc spécifique pour les schedulers de type grille.

Fonctionnalités

Les fonctionnalités de cette classe permet de mettre en oeuvre "facilement" la technique du parachute, à savoir:

- Création d'un répertoire tarball contenant 4 sous−répertoires : lib, python, toUI, toSE
  - dans le répertoire lib, APCScheduler mets toutes les librairies dynamiques non systèmes indiquées par la commande unix ldd appliquée au binaire
  - le répertoire python contient la librairie python APCScheduler
  - toUI est une sorte de boîte aux lettres, c'est à dire que tous les fichiers déposés lors de l'exécution seront copiés vers l'UI automatiquement à la fin de l'application
  - toSE même chose que toUI mais pour un répertoire d'un SE défini par la méthode **SetOutputDirSE()**
  - en plus de ces 4 sous−répertoires l'utilisateur peut ajouter ses fichiers et ses propres répertoires via la méthode **SetInput()**

- Au moment du submit le tarball est déplacé vers un répertoire temporaire sur un SE défini par la méthode **SetWorkDirSE()** pour qu'il puisse être accessible depuis le worker node
- Le fichier jdl associé à l'application contient un script qui recopira le tarball vers le worker node, le dezipera et lancera soit le binaire lui−même soit un MainScript (voir chapitre suivant) écrit par l'utilisateur et déclaré dans le constructeur de la classe
- à la fin de l'application le contenu des boites aux lettres toUI et toSE est copié vers les répertoires indiqués et le contenu du répertoire temporaire du SE est nettoyé

Création d'un object AppliParachute et déclaration du MainScript

Le constructeur contient 3 paramètres, le premier obligatoire est le nom du binaire ou un tarball sur le SE, le second optionnel est un script dit MainScript qui sera lancé à la place du binaire. Ce MainScript est parfois nécessaire pour :

- Positionner quelques variables d'environnement avant l'appel du binaire
- Eventuellement réaliser plusieurs fois le binaire, par exemple pour des applications du type Monte−Carlo
- Recopier des fichiers de sortie vers les boites aux lettres toUI et toSE si le binaire ne le fait pas directement

Le paramètre *KeepTarBall* est un boolean optionel (False) qui permet de conserver le tarball dans le répertoire de travail de l'UI.

La méthode suivante de la classe AppliParachute ou JobClass permet de copier le tarball sur le SE afin de réutiliser et de le réutiliser par la suite:

- **CopyTarBallOnSE(*pathSE*)** : copie le tarball sur le SE, *pathSE* est de la forme 'se:path/to/the/nametar'.

Exemple

Exemple de création de tarball a partir d'un binaire et d'un script principal:

```
# -*- coding: utf-8 -*-

from APCScheduler import *

# 0- define some env variables
##############################
SetRepository('DataAniso')
SetVerboseLevel(5)
chemin='/home/dupont/code/'

# Ajout à LD_LIBRARY_PATH le chemin des librairies
# dynamique utilisées par le binaire
AddLIBRARYPATH(chemin+'Libs')

# 1- Define my exe
##################
MyExe = AppliParachute(chemin+'main',
                       'AnisoLimitsScript.sh', True)

# Ajout des données d'entrées
```

```
MyExe.SetInput([chemin+'Distances',chemin+'etc',chemin+'fonts',
                chemin+'Propagation', chemin+'config.conf',
                chemin+'/HeraldEventsAllReduced.txt'])
MyExe.SetArg('config.conf')
MyExe.SetStorageElement('node12.datagrid.cea.fr')

# définition optionnelle du répertoire où seront copié les
# fichiers présents dans le répertoire toSE du tarball
MyExe.SetOutputDirSE('colley/DataAniso')
# on donne uniquement la fin du chemin, dans ce cas les
# données seront déposées dans
# lfn:/grid/vo.apc.univ-paris7.fr/colley/DataAniso

# définition du répertoire où sera copié le tarball et
# les fichiers présents dans le répertoire toUI du tarball
# avant d'etre copies dans le répertoire de travail.A la
# fin du job ce répertoire est nettoyé
# on donne que la fin du chemin
MyExe.SetWorkDirSE('colley/APCgrid')
# Les données seront déposées dans
# lfn:/grid/vo.apc.univ-paris7.fr/colley/APCgrid

# 2- exe + scheduler = job
############################
Job = NewJob(MyExe ,'GLITE')
Job.Submit("30:0")
Job.Wait()
Job.SendMailResult()
```

Et le script AnisoLimitsScript.sh qui appelle plusieurs fois l'éxecutable et place les fichiers de sortie dans la boite aux lettres 'toUI':

```
~>more AnisoLimitsScript.sh
#!/bin/sh

export ROOTSYS=./
mkdir Outs

echo "**************** date debut job code : " `date`
for i in `seq 1 100`;do
        ./main $1 2>&1
done
echo "**************** date fin job code  : " `date`

cp Outs/* toUI/
```

Exemple d'utilisation de la méthode *CopyTarBallOnSE()*

```
from APCScheduler import *

SetVerboseLevel(10)
AddLIBRARYPATH("/home/colley/Grid/install/fftw-3.2.1/.libs:/home/colley/Grid/install/HL2_Desire/s

# 1- Define my exe
pathExe ='/home/colley/Grid/demo/apcgrid_run/testConvolFFT'
MyExe = AppliParachute(pathExe,'script_loop.sh', label='testConvolFFT',  KeepTarBall=False)
MyExe.SetArg('3')

# 2- exe + scheduler = job
Job = NewJob(MyExe ,'GLITE')
```

```
query='CEStatus=Production,OSVersion=Boron,PlatformArch=x86_64,EstRespTime=0,'
query+='WaitingJobs=0,"FreeCPUs>=20","FreeJobSlots>=20","TotalCPUs>=50","MaxCPUTime>=500"'

Job._Scheduler.LoadBalancingCE("LoadBalCE_v3",query)
Job._Scheduler.ExcludeWMS(["node27.datagrid.cea.fr","grid25.lal.in2p3.fr","marwms.in2p3.fr"])

Job.CopyTarBallOnSE('se:colley/tarball/ConvolFFT')

Job.Submit()
Job.Wait()
Job.SendMailResult()
```

Interface

Les classes **Appliexe** et **AppliPython** ont la même interface. Voici les méthodes permettant de définir et de spécialiser une application.

Fichiers d'entrées sorties

- **SetInput(*List*)** : permet de définir les fichiers à inclure dans l'InputSandbox dans le cas des scheduler de type grille. Exemple : MyExe.SetInput(['In0','In1','Parfile.txt'])
- **SetInputSE(*List*)** : permet de définir les fichiers à recopier sur le CE dans la cas d'une application de type parachute.
- **SetOutput(*List*, *AddID*=True)** : permet de définir les fichiers output à récupérer dans l'OutputSandbox dans le cas des scheduler de type grille. L'option *AddID* permet d'ajouter ou pas l'identificateur unique du job avant le nom du fichier output définit dans *List*.
- **SetOutputSE(*List*, *AddID*=True)** : permet de définir les fichiers output à récupérer depuis un Storage Element dans le cas des scheduler de type grille. L'option *AddID* permet d'ajouter ou pas l'identificateur unique du job avant le nom du fichier output définit dans *List*. Exemple :

MyExe.SetOutputSE(['lfn:/grid/vo.apc.univ−paris7.fr/colley/graphe.jpg'])

- **GetNameOutput(*idx*)** : donne le nom du *idx*−ième fichier output définit par SetOutput()
- **GetNameStdErr()** : donne le nom du fichier stderr du job
- **GetNameStdOut()** : donne le nom du fichier stdout du job

Statut de l'application

- **IsFinish()** : retour True si le job est fini.
- **IsOk()** : retour True si le job est fini et si son statut est 'FinishOK'

Ressources et arguments

- **SetArg(*Arg*)** : permet de définir les arguments associés à l'exécutable, l'option *Arg* est une chaîne de caractère.
- **SetMPI(*[Node, CPUbyNode]*)**: pour le application MPI permet de définir le nombre de Noeud et le nombre de CPU par noeud, le total de CPU est le produit de *Node* et *CPUbyNode*.
- **SetCPUTime(*String*)** : permet de définir le temps CPU nécessaire à l'éxecution de l'application, ce paramètre est fortement recommandé par exemple pour l'ordonnanceur BQS, le format est le suivant

[[hh:]mm:]ss.
- **SetCPUTimePerWeek(*String*)** : permet de définir le temps CPU par semaine nécessaire à l'éxecution de l'application, ce paramètre permet de à l'ordonnanceur BQS de mettre le job en classe V, le format est le suivant [[hh:]mm:]ss.
- **SetAccessDirectory(*String*)** : permet d'indiquer à l'ordonnanceur BQS l'accès à un ressource disque.
- **SetMemorySize(*Integer*)** : permet de définir le volume de mémoire en Mo nécessaire à l'éxecution de l'application, ce paramètre est fortement recommandé par exemple pour l'ordonnanceur BQS.

Identification

- **GetNameJobID()** : donne l'identificateur unique du job

Exemple

```
#! /usr/bin/python

from APCScheduler import *

# 1- Define my exe
MyExe = AppliExe('testOpenMPIifort')
MyExe.SetCPUTime('10:0')
MyExe.SetMemorySize(800)
MyExe.SetMPI([6,1])
MyExe.SetAccessDirectory('u_sps_planck')
```

Schedulers

Scheduler UNIX

La classe **SchedulerLOCAL** permet de soumettre un exécutable localement sur UNIX. On peut définir un job avec ce scheduler avec la fonction NewJob() en utilisant le mot clé 'LOCAL':

```
Job = NewJob(MyExe,'LOCAL')
```

Scheduler grille

Seul le protocole Glite est disponible pour soumettre un job sur la grille à travers la classe *SchedulerGLITE*. **Avant d'utiliser la librairie python avec la grille, lancer la procédure d'initialisation avec apcgrid–init**, c'est durant cette procédure qu'APCScheduler enregistera votre VO. La fonction *newjob* permet de définir un job (association d'une application et d'un scheduler) sans définir explicitement un objet scheduler par l'intermédiaire de keywords. Pour la grille employer 'GLITE':

```
Job = NewJob(MyExe,'GLITE')
```

Avertissement sur le proxy

Cas 1 : proxy valide mais obtenu lors d'une autre session UNIX

APCScheduler teste la validité du proxy au lancement du premier job sur la grille avec la fonction

```
apcui01:~/xxx>glite-voms-proxy-info
subject   : /O=GRID-FR/C=FR/O=CNRS/OU=APC/CN=Jean-Marc Colley/CN=proxy
issuer    : /O=GRID-FR/C=FR/O=CNRS/OU=APC/CN=Jean-Marc Colley
identity  : /O=GRID-FR/C=FR/O=CNRS/OU=APC/CN=Jean-Marc Colley
type      : proxy
strength  : 1024 bits
path      : /tmp/x509up_u709
timeleft  : 6:32:17
```

APCScheduler examine le champ **timeleft**, s'il est inférieur à 48 heures il lance la procédure d'initialisation du proxy suivante

```
glite-voms-proxy-init -voms <MyVO> -hours 24
```

où <MyVO> est votre VO de travail. Cependant lorsque l'on a un **timeleft** suffissant mais issu d'un glite−voms−proxy−init d'un session de la veille la soumission du job échoue (parfois?) avec le message d'erreur du type :

```
Warning - Unable to delegate the credential to the endpoint: https://node27.datagrid.cea.fr:7443/
User not authorized:
unable to check credential permission  (/opt/glite/etc/glite_wms_wmproxy.gacl)
(credential entry not found)
credential type: person
input dn: /O=GRID-FR/C=FR/O=CNRS/OU=APC/CN=Jean-Marc Colley



Error - Operation failed
Unable to find any endpoint where to perform service request
```

Pour résoudre le problème il est préférable de détruire le proxy

```
apcui01:~/xxx>glite-voms-proxy-destroy
```

Cas 2 : proxy valide mais obtenu pour une autre VO

Si vous changez de VO, il est préférable de détruire le proxy

ensuite APCScheduler lancera glite−voms−proxy−init avec les bons paramètres.

Personnaliser les paramètres de soumission

Changer le Rank

Le rank est stocké dans l'attribut _Rank de la classe **SchedulerGLITE**:

```
>>> from  APCScheduler  import *
>>> a=SchedulerGLITE()
>>> a._Rank
'Rank = ( other.GlueCEStateWaitingJobs == 0 ) ?
((other.GlueCEStateFreeCPUs==0)?-2222:other.GlueCEStateFreeCPUs) : -

other.GlueCEStateWaitingJobs * 10 / (other.GlueCEStateRunningJobs + 1)
 * ( (other.GlueCEStateFreeCPUs == 0)?500:1 ) ;'
```

Voici comment changer le rank d'un objet de la classe **SchedulerGLITE** et l'utiliser pour scheduler un job

```
MyExe = AppliExe('echo')
MySched = SchedulerGLITE()
MySched._Rank = 'Mon rank'
Job = JobClass(MyExe, MySched)
```

Changer le Requirements

Le requirement est stocké dans l'attribut _Req de la classe **SchedulerGLITE**:

```
>>> from  APCScheduler  import *
>>> a=SchedulerGLITE()
>>> a._Req
'Requirements = other.GlueCEStateStatus == "Production" && ( ! ( RegExp(".*node16.*",other.CEId)
```

Voici comment changer le requirement d'un objet de la classe **SchedulerGLITE** et l'utiliser pour scheduler un job

```
MyExe = AppliExe('echo')
MySched = SchedulerGLITE()
MySched._Req = 'Mon requirement'
Job = JobClass(MyExe, MySched)
```

Définir le Compute Element

Avec le requirements par défaut, le Compute Element (CE) est défini dynamiquement par la grille. Il est possible de préciser le Compute Element sur lequel vous souhaitez que votre job s'exécute. La classe **SchedulerGrid** possède 2 méthodes pour cela:

- **SetLocalCE**(): Sans argument, la valeur du CE sera celle choisie durant la procédure d'initialisation apcgrid−init.
- **SetCE(*String*)**: Où *String* est une chaîne de caractère indiquant un CE ie un nom de machine valide pour votre VO.

Pour connaître la lsite de CE de votre VO:

```
apcui01:~> lcg-infosites --vo vo.apc.univ-paris7.fr ce
#CPU    Free    Total Jobs      Running Waiting ComputingElement
----------------------------------------------------------
 392     392       1              0        1    ipnls2001.in2p3.fr:2119/jobmanager-pbs-apc
```

```
 360       0       0              0        0    polgrid1.in2p3.fr:2119/jobmanager-pbs-apc
 352     352       0              0        0    node07.datagrid.cea.fr:2119/jobmanager-lcgpbs-apc
 242     242       0              0        0    lpnce.in2p3.fr:2119/jobmanager-pbs-apc
  12      12       0              0        0    node16.datagrid.cea.fr:2119/jobmanager-pbs-apc
  88      87       1              1        0    apcce01.in2p3.fr:2119/jobmanager-pbs-apc
  64      64       0              0        0    ipngrid12.in2p3.fr:2119/jobmanager-pbs-apc
1168    1168       0              0        0    grid10.lal.in2p3.fr:2119/jobmanager-pbs-apc
```

Exemple:

```
from APCScheduler import *

# 1- Define my exe
MyExe = AppliExe('echo')
MyExe.SetArg('Hello APCgrid demo GAP')

# 2- exe + scheduler = job
Job = NewJob(MyExe,'GLITE')
Job._Scheduler.SetCE('grid10.lal.in2p3.fr')
Job.Submit()
```

Il est également possible d'utiliser un algorithme qui choisit les CE suivant une requête via la méthode:

- **LoadBalancingCE(*Algo*, *query*)**: *query* est optionel c'est le paramètre −−query de la commande lcg−info qui permet de sélectionner les CE valides pour l'exécutable, exemple de la syntaxe de cette requête:
  - *CEStatus=Production,PlatformArch=x86_64,EstRespTime=0* c'est la requête par défaut
  - *CEStatus=Production,OSVersion=Boron,PlatformArch=x86_64, EstRespTime=0,WaitingJobs=0,"FreeCPUs>=20","FreeJobSlots>=20","TotalCPUs>=50", "MaxCPUTime>=500*

Ensuite l'algorithme de choix des CEs sélectionnés par la requête lcg−info peut se faire selon 3 méthodes que l'on défini via le paramètre *Algo* avec les mots clés suivants:

- 'LoadBalCE_v1' utilise uniquement le CE qui présente le plus de slot 'Free', une fois tous les slots épuissés utilisation du CE par défaut
- 'LoadBalCE_v2' choisit le CE présentant le plus de slot 'Free', l'algo décrémente ce CE d'un slot puis retrie l'ensemble des CE, c'est l'algo par défaut
- 'LoadBalCE_v3' choisit au hasard un CE, c'est l'algo par défaut pour l'option −−lbce de la commande apcgrid−run

Exemple de l'utilisation du load balancing sur un ensemble de job réutilisant un tarball déjà existant. **Il est important que tous les jobs utilisent le même objet SchedulerGLITE()**:

```
from APCScheduler import *

mywp='/home/colley/Grid/Run/test/31'
SetRepository(mywp)

# 1 define scheduler
MyGLITE = SchedulerGLITE()
MyGLITE.LoadBalancingCE()

Run = MultiJobsClass('testConvolFFT')
for i in range(100):
```

```
    MyExe = AppliParachute('se:colley/tarball/ConvolFFT', label="testConvolFFT_%i"%i)
    MyExe.SetArg('2')
    job = JobClass(MyExe, MyGLITE)
    Run.Append(job)

Run.TimerUpdateStatus('30')
Run.SubmitAndWaitAll('1:0:0')
Run.SendMailResult()

Run.ConcatStdOut(mywp+'allstd.txt')
SendMailFile(mywp+'allstd.txt')
```

Définir les WMS

Par défaut APCScheduler utilise tous les WMS, mais en pratique un WMS défectueux peut être présent dans la liste, il est donc pratique de l'exclure. La classe **SchedulerGLITE** possède les deux méthodes suivante pour définir la liste des WMS valide :

- **SetWMS(*ListWMS*)** : où *ListWMS* est une liste de chaines de caractère indiquant les WMS sur lesquels vous souhaitez soumettre vos jobs.
- **ExcludeWMS(*ListWMS*)** : où *ListWMS* est une liste de chaines de caractère indiquant les WMS sur lesquels vous ne souhaitez pas soumettre vos jobs.

Pour connaître la liste des WMS associé à votre VO, utiliser la commande:

```
apcui01:~> lcg-infosites --vo vo.apc.univ-paris7.fr wms
https://grid25.lal.in2p3.fr:7443/glite_wms_wmproxy_server
https://marwms.in2p3.fr:7443/glite_wms_wmproxy_server
https://node04.datagrid.cea.fr:7443/glite_wms_wmproxy_server
https://node27.datagrid.cea.fr:7443/glite_wms_wmproxy_server
```

Exemple:

```
from APCScheduler import *

# 1- Define my exe
MyExe = AppliExe('echo')
MyExe.SetArg('Hello APCgrid demo GAP')

# 2- exe + scheduler = job
Job = NewJob(MyExe,'GLITE')
Job._Scheduler.SetWMS(['node04.datagrid.cea.fr','marwms.in2p3.fr'])
Job.Submit()
```

Délégation de proxy

Méthode de **APCSchedulerGLITE** permet de mettre en place la délégation de proxy:

- **delegateProxy()**

Scheduler cluster

BQS

La classe **SchedulerBQS** permet de soumettre un exécutable sur les fermes de calcul au centre de calcul IN2P3 via le scheduler BQS/SL5. On peut définir un job avec ce scheduler avec la fonction NewJob() en utilisant le mot clé 'BQS':

```
Job = NewJob(MyExe,'BQS')
```

Définir les ressources

Les ressources mémoires, temps de calculs et accès disque doivent être déclarées pour déterminer la classe dans laquelle l'application va être lancée. Les méthodes **SetMemorySize()**, **SetCPUTime()**, **SetAccessDirectory()** ne sont pas obligatoires car des valeurs pas défaut sont positionnnées, mais tout dépassement de ces ressources par défaut sera sanctionné par BQS par l'arrêt du job, il est donc recommandé de les définir.

Applications MPI

APCScheduler gère les applications MPI compilées avec OpenMPI avec l'ordonnanceur BQS. Trouver ici [1] la procédure de compilation d'une application OpenMPI au CCIN2P3. Une application est considérée comme MPI par APCScheduler si le nombre de processeurs est précisé par la méthode **SetMPI()** de la classe **AppliExe**.

Exemple

Voici un exemple pour soumettre une application FORTRAN90 MPI, qui a besoin de 800Mo de mémoire, 6 CPUs sur 6 noeuds différents pour un temps de calcul de 10 minutes CPUtime, enfin cet exécutable accédera à l'espace disque GPFS /sps/planck/...

```
#! /usr/bin/python

from APCScheduler import *

# 0- define some env variables
SetRepository('testBQS')
SetMail('dupond@xxx.fr')
SetVerboseLevel(15)

# 1- Define my exe
MyExe = AppliExe('testOpenMPIifort')
MyExe.SetCPUTime('10:0')
MyExe.SetMemorySize(800)
MyExe.SetMPI([6,1])
MyExe.SetAccessDirectory('u_sps_planck')

# 2- exe + scheduler = job
Job = NewJob(MyExe,'BQS')

Job.Submit("10:0")
Job.Wait()
```

```
Job.SendMailResult()
```

Soumettre une application

Description de la classe JobClass

Nous avons vu comment définir un job soit en déclarant directement un objet de la classe **JobClass** soit à travers la fonction *newjob()*. Ses deux principaux attributs sont donc un objet application et un objet scheduler. Voici la liste des méthodes de la classe.

Soumettre et attendre la fin du job

- **Submit(*TimeOutToStart*='15:00')** : soumet un job, l'argument optionnel *TimeOutToStart* définie le durée après laquelle le job est re−soumis ou abandonné s'il n'est pas passé en mode RUNNING, 15:00 correspond à 15 minutes, format [[hh:]mm:]ss.
- **Wait()** : attend la fin du job
- **SubmitAndWait(*TimeOutToStart*='15:00')** : soumet un job et attend sa fin, l'argument optionnel *TimeOutToStart* définie le durée après laquelle le job est re−soumis ou abandonné s'il n'est pas passé en mode RUNNING, 15:00 correspond à 15 minutes, format [[hh:]mm:]ss.
- **TimerUpdateStatus(*timeUpdate*='2:0')** : définie le temps entre deux mises à jour du statut du job sur la grille, format [[hh:]mm:]ss.

Connaître le statut du job

- **IsFinish()** : retour True si le job est fini
- **IsOk()** : retour True si le job est fini et si son statut est 'FinishOK'
- **Status()** : retourne le statut du job, liste des statuts:
    - NotSubmit : *le job n'a pas encore été soumis, c'est le statut intial*
    - SubmitQuery : *la demande de soumission du job est en cours*
    - SubmitScheduled : *le job a été accepté par le scheduler, mais il n'est pas encore exécuté*
    - SubmitRunning : *le job est en cours d'exécution*
    - SubmitRetrieveSE : *le job est terminé mais dans le cas du scheduler grid il n'est pas considéré comme totalement fini si l'outputSandbox n'a pas été récupéré ainsi que d'autres données sur le SE*
    - FinishOK : *le job est terminé et le statut de sortie de l'exécutable est ok*
    - FinishNOK : *le job est terminé et le statut de sortie de l'exécutable est Nok*
    - FinishCancelled : *le job a été abandonné par l'utilisateur*
    - FinishSubmitNOK : *le job a échoué à la soumission, problème dans le script cluster ou jdl, ou pas de resource permettant de recevoir cette demande. Le répertoire de travail doit contenir un fichier IDjob.sub_err avec le message d'erreur de la commande submit*
    - FinishLostIDgrid : *l'identificateur grid du job a été perdu, il est stocké dans un fichier*
    - FinishUnknow : *le job est terminé mais pas d'information sur le statut final*
    - FinishAborted : *le job a été arrêté pendant son exécution du certainement à un problème sur le worker*
    - FinishNOKretrievePB : *le job est terminé mais le processus de récupération des données (outputSandbox) a échoué*

    - ♦ FinishTimeOut : *le job n'est jamais passé en mode RUNNING dans le délais fixé par l'utilisateur (paramètre**TimeOutToStart** de submit())*
- **SendMailResult()** : envoie un mail avec le contenu du stdout à l'adresse définie par la fonction SetMail('dupond@xxx.fr'). Il est nécessaire de précéder cette méthode d'un Wait() pour attendre la fin du job pour connaître son issu, sinon nous aurons son statut courant et le fichier stdout sera inexistant.

Accès aux fichiers résultats

- **Output(*Idx*)** : ouvre en lecture le *idx*–ième fichier output. *Idx* commence à 0 et utilise l'ordre de défintion des fichiers output de la méthode SetOutput() de l'objet application
- **SdtErr()** : donne le contenu de stderr
- **StdOut()** : donne le contenu de stdout

Méthodes diverses

- **FullNameOutput(*Idx*)** : donne le nom avec le chemin absolue du fichier *idx*–ième fichier output
- **FullNameStdErr()** : donne le nom avec le chemin absolue du fichier stderr du job
- **FullNameStdOut()** : donne le nom avec le chemin absolue du fichier stdout du job
- **Name()** : retourne le nom du job

Exemple

```
#! /usr/bin/python

from APCScheduler import *

# 0- define some env variables
SetRepository('testBQS')
SetMail('dupond@xxx.fr')
SetVerboseLevel(15)

# 1- Define my exe
MyExe = AppliExe('testOpenMPIifort')
MyExe.SetCPUTime('10:0')
MyExe.SetMemorySize(800)
MyExe.SetMPI([6,1])
MyExe.SetAccessDirectory('u_sps_planck')

# 2- exe + scheduler = job
Job = NewJob(MyExe,'BQS')

Job.Submit("10:0")
Job.Wait()
Job.SendMailResult()
```

Soumettre plusieurs applications

Description de la classe MultiJobsClass

La classe **MultiJobsClass** permet de soumettre et d'attendre plusieurs jobs en même temps et de concaténer les fichiers de sortie. Son attribut principal est une liste de job, donc des objets JobClass. On ajoute un job à la liste avec cette méthode:

- **Append(job)** : ajoute un job à la liste des jobs à exécuter

Soumettre et attendre la fin de tous les jobs

- **SubmitAndWaitAll(*TimeOutToStart*="24:0:0", *MaxRunning*=0)** : soumet et attend que tous les jobs se terminent, l'argument optionnel *TimeOutToStart* définit le durée après laquelle le job est re−soumis ou abandonné s'il n'est pas passé en mode RUNNING, 24:0:0 correspond à 24 heures, format [[hh:]mm:]ss, *MaxRunning* indique le nombre maximum de jobs qui peuvent être RUNNING simultanément, 0 indique qu'il n'y a pas de limite
- **SubmitAll(*TimeOutToStart*="15:00")** : soumet tous les jobs, l'argument optionnel *TimeOutToStart* définit le durée après laquelle le job est re−soumis ou abandonné s'il n'est pas passé en mode RUNNING, 15:00 correspond à 15 minutes, format [[hh:]mm:]ss.
- **WaitAll()** : attend la fin de tous les jobs
- **TimerUpdateStatus(*timeUpdate*='2:0')** : définie le temps entre deux mise à jour du status des jobs sur la grille, format [[hh:]mm:]ss.

Mail avertissant de la fin des jobs

- **SendMailResult()** : envoie un mail donnant le taux de réussite des jobs et la liste de leur statut à l'adresse définie par la fonction SetMail('dupond@xxx.fr'). Il est nécessaire de précéder cette méthode d'un Wait() pour attendre la fin des jobs pour connaître son issu, sinon nous aurons leur statut courant.

Accès et manipulation des fichiers résultats

- **ConcatOutput(*Idx*, *FileConcat*)** : concatène de tous les *idx*−ième fichier output des jobs dans *FileConcat*. *Idx* commence à 0 et utilise l'ordre de défintion des fichiers output de la méthode SetOutput de l'objet application
- **ConcatEachOutput(*PrefixeFileConcat*)**: concatène tous les fichiers output de même rang
- **ConcatStdOut(*FileConcat*)** : concatène tous les stdout des jobs dans *FileConcat*
- **ConcatStdErr(*FileConcat*)** : concatène tous les stderr des jobs dans *FileConcat*

Exemple

```
#! /usr/bin/python
from APCScheduler import *

# 0- define some env variables
WorkDir = '/home/colley/Grid/RunSubmit/run31/'
SetRepository(WorkDir)
SetMail('dupond@xxx.fr')
SetVerboseLevel(2)

Run = MultiJobsClass()
```

```
for i in range(3):
    MyExe = AppliExe('./ScriptTest')
    MyExe.SetArg('%d out0.txt out1.txt'%i)
    MyExe.SetOutput(['out0.txt','out1.txt'], AddID=True)
    MyExe.SetOutputSE(['lfn:/grid/vo.apc.univ-paris7.fr/colley/graphe.jpg'], AddID=True)

    Job = NewJob(MyExe,'GLITE')
    Run.Append(Job)

Run.SubmitAndWaitAll()
Run.SendMailResult()

# concat out0.txt
MyFile = WorkDir+'concat_out0.txt'
Run.ConcatOutput(0, MyFile)
SendMailFile(MyFile)
```

Avec ScriptTest

```
#!/bin/bash

myid=$1
shift
echo "Je suis le job " $myid
for myarg in $*
do
  echo "Ceci est le fichier "$myarg" lie au job "$myid|  cat > $myarg
  echo "create file " $myarg
done
```

Annexe

python

Documentation

La version 2.3.4 (May 2004) est disponible sur apcui01.in2p3.fr

- liste de tous les documents : http://www.python.org/doc/2.3.4/index.html
- la plus utile : http://www.python.org/doc/2.3.4/lib/lib.html

Lancer un exécutable/script sous python et récupérer son résultat

Utiliser le module popen2 (process open) et sa méthode popen3

```
import popen2

(o,i,e) = popen2.popen3('/chemin/vers/mon/script')
# on récupère dans une chaine de caractères le stdout en lisant le fichier 'o'
output = o.read()

# on récupère dans une chaine de caractères le sdterr en lisant le fichier 'e'
```

```
error = e.read()
```

Exemple, attention si on lit 2 fois le stdout la deuxième fois il est vide, il est conseillé de stocker le résultat dans une variable:

```
apcpcs5:~/Grid/projet/APCScheduler/trunk/src>python
Python 2.3.4 (#1, Dec 10 2007, 15:05:56)
[GCC 3.4.5 20051201 (Red Hat 3.4.5-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import popen2
>>> (o,i,e) = popen2.popen3('ls')
>>> print o.read()
APCgrid.py
APCgrid.pyc
APCgrid.py~

>>> print o.read()

>>> (o,i,e) = popen2.popen3('ls')
>>> out= o.read()
>>> print out
APCgrid.py
APCgrid.pyc
APCgrid.py~
>>> print out.split()
['APCgrid.py', 'APCgrid.pyc', 'APCgrid.py~']
>>> print out.split()[0]
APCgrid.py
```

Voir ici [2] les puissante méthodes permettant de manipuler les chaines de caractères