TP3 Grille de Calcul $3^{ème}$ année F2

Soumission de jobs avec JSAGA

Éléments à rendre avant le prochain TP :

À envoyer dans un tarball en pièce-jointe à `passerat@isima.fr` *:*
- *le code Java permettant l'envoi de jobs sur le CREAM CE de votre choix et le cluster (2 codes séparés possible)*
- *le code Java du dernier exercice ainsi que le fichier de sortie*

Utilisation de l'API JSAGA en ligne de commandes

Les développements d'applications conséquentes sur la grille ne peuvent se contenter de scripts shells. Ainsi une API comme JSAGA simplifie la tâche des développeurs en encapsulant les commandes de différents middlewares dans des objets Java. Notez que cette API dispose également d'une version en ligne de commandes que nous n'étudierons pas pour le moment.

1 Envoi de jobs simples avec JSAGA

En utilisant la présentation de l'API JSAGA qui vous a été présentée en cours, et les informations que vous pourrez trouver sur la documentation de JSAGA (http ://grid.in2p3.fr/jsaga/saga-apidocs/index.html), reprenez le dernier script du TP précédent et réalisez sa version JSAGA.

2 Les adaptateurs de JSAGA

Pour supporter la soumission sur plusieurs types d'environnements distribués, JSAGA déporte le code spécifique à ces environnements au sein de greffons appelés adaptateurs. Vous avez utilisé l'adaptateur glite-WMS dans la première partie du TP. Nous allons à présent découvrir les adaptateurs pour glite-CREAM et Torque.

2.1 CREAM

Vous avez pu constater que la soumission par les WMS s'avérait parfois lente...L'API CREAM a été mise en place pour permettre aux utilisateurs d'envoyer directement des jobs sur les CE, sans passer par le processus classique d'attribution des ressources.

1. La commande suivante (**attention aux quotes si vous copiez-collez**) :

```
lcg-infosites --vo biomed ce | grep cream \
| tr -s '\t' ' ' | cut -d ' ' -f 7
```

vous permettra de lister tous les CE accessibles supportant CREAM.

2. Choisissez un CE dans cette liste et modifier votre code JAVA en conséquence pour soumettre vos jobs sur ce CE.
3. Quel mécanisme de JSAGA permet un fonctionnement aussi simple ?

2.2 PBS/Torque

Cet adaptateur a été en partie développé par vos illustres prédécesseurs (Taha Benyezza et Yassine Bachar). Cette fois-ci une étape de configuration supplémentaire est nécessaire. Vous ajouterez donc les lignes suivantes à votre code source Java :

```
// Create PBS-SSH context and set its attributes to log in via SSH
Context ctx = ContextFactory.createContext();

ctx.setAttribute(Context.SERVER,"pbs-ssh://hpc1.clermont-universite.fr");
ctx.setAttribute(Context.TYPE, "SSH");

Session session = SessionFactory.createSession(false);
session.addContext(ctx);
```

L'instance de Session nouvellement créée devra entrer en jeu lors de la création du `JobService`. Trouvez la méthode adaptée à cette situation.

3 Pour aller plus loin...

Vous trouverez un exécutable `tinymt32dc-static` dans le répertoire : */home/jopasserat/enseignement/tp3*. Cet exécutable produit une des statuts permettant d'initialiser un générateur de nombres pseudo-aléatoires. L'idée étant de générer une grande quantité de statuts, nous souhaitons utiliser la grille pour réaliser une campagne de calcul et déterminer un maximum de statuts.

1. L'exécution se lance comme suit :

```
./tinymt32dc-static -m 0 id
```

2. *id* est le seul paramètre de cet exécutable, il varie de 0 à 2^{32}.
3. Déterminez le temps de calcul nécessaire à exécuter 1 fois ce processus.
4. Chaque binôme devra lancer les exécutions pour la plage de paramètre suivante : $[128000 * (idBinome - 1); 128000 * (idBinome)]$

5. Au final, chaque binôme devra fournir en sortie un fichier contenant les sorties concaténées de toutes les exécutions.
6. Cet exécutable étant très rapide, il n'est peut être pas judicieux d'affecter un job à chaque exécution...

   Attention ! Faîtes en sorte que les processus JSAGA que vous lancez sur l'UI ne soient pas tués si vous vous déconnectez, sinon vous ne pourriez pas récupérer les sorties de vos jobs.