

API DIET batch management multi-jobs system

11 novembre 2010

1 Présentation

La pile logicielle *DIET batch management multi-jobs system* est fournie avec une API permettant à un utilisateur de passer par des fonctions pour soumettre des tâches ou des lots (ensemble de jobs), consulter la liste des tâches ou lots en cours d'exécution, d'annuler un ensemble de tâches, de construire un lot à partir des tâches déjà soumises, de consulter l'état courant d'un lot et de récupérer les résultats d'une tâche terminée ou de récupérer d'éventuelles erreurs d'une tâche terminée. Ce document présente la description de chacune de ces fonctions.

2 Définitions

Afin de bien comprendre la suite de ce document, cette section présente les définitions des termes suivants : *script*, *tâche*, *lot* et *script générique en DIET*.

- **Script** : Un script est un fichier qui contient la description textuelle complète d'une tâche qui comporte, entre autres, la description des ressources (processeurs et mémoires par exemple) requises afin d'exécuter celle-ci, la durée de réquisition, une éventuelle redirection des entrées-sorties standards de la tâche, ou des variables d'environnement pour la tâche. Le format de ces descriptions peut être soit spécifique à un ordonnanceur, soit générique.
- **Tâche** : Une tâche est le travail proprement dit, décrit par un script et consommant des ressources (processeurs, mémoires) après avoir été ordonnancé sur celles-ci. Une tâche a en standard une entrée, une sortie, et une sortie d'erreur. Une tâche a également un état : soumise, en cours d'exécution, en erreur, suspendue, arrêté (avant d'avoir terminé), ou corrompue (par le système).
- **Lot** : Un lot est un ensemble de tâches. Un lot peut également être décrit par un ensemble de scripts dans un fichier. Dans un fichier le début d'un lot doit toujours commencer par le mot clé `DIET # % begin_diet_micro_jobs` et la fin doit se terminer par le mot clé `# % end_diet_micro_jobs`. Chaque ligne entre ces mots clés doit être un chemin menant vers un script.
- **Script générique** : Un script générique DIET est script qui peut être envoyé vers un ordonnanceur quelconque (Loadleveler ou Torque). Sa syntaxe est fournie par DIET. Les commentaires commencent par le caractère `#`, et les mots clés commencent par les deux caractères : `# %`. Ci-dessous la définition de quelques mots clés :
 - `# % dietGroup` : permet de spécifier le nom d'un groupe.
 - `# % dietJob_name` : permet de spécifier le nom de la tâche soumise.
 - `# % dietoutput` : permet de spécifier le chemin où mettre les résultats une fois que la tâche soumise se termine. Pour pouvoir récupérer les fichiers de données, l'utilisateur doit toujours lancer la commande `diet-OnlineOutPut-lot` en tâche de fond sur la machine cliente.
 - `# % dieterror` : permet de spécifier le chemin où mettre les erreurs une fois que la tâche soumise se termine. Pour pouvoir récupérer les fichiers d'erreurs, l'utilisateur doit toujours lancer la commande `diet-OnlineOutPut-lot` en tâche de fond sur la machine cliente.
 - `# % dietExecutable` : permet de spécifier l'exécutable de la tâche à soumettre.

- # % **dietArguments** : permet de spécifier les arguments de l'exécutable de la tâche à soumettre.
- # % **dietJob_type** : permet de spécifier le type de tâche.
- # % **dietWallClockLimit** : permet de spécifier le temps nécessaire à l'exécution de la tâche.
- # % **dietInitialdir** : permet de spécifier le répertoire de soumission de la tâche.
- # % **dietNotification** : permet de préciser la notification.
- # % **dietNotify_user** : permet de préciser l'adresse où sera envoyée la notification.
- # % **dietClass** : permet de spécifier la classe de soumission.
- # % **dietjob_nodes** : permet de spécifier le nombre de noeuds.
- # % **dietQueue** : permet de spécifier la queue de soumission.
- **Syntaxes Spécifiques pour les tâches paramétriques :**
 - # % **dietArguments = GenArg** permet de préciser que les arguments seront générés par une fonction générateur fournie par l'utilisateur
(void my_generate_args(std::vector<std::string>& vec_args)).
 - # % **dietNbJobsParam** permet de préciser le nombre de jobs paramétrique que l'utilisateur veut lancer.
- **Ajout des sections spécifiques à un ordonnanceur**
 - # % **Loadleveler_sec_beg** : syntaxe permettant d'indiquer le début d'une section spécifique pour l'ordonnanceur Loadleveler.
 - # % **Loadleveler_sec_end** : syntaxe permettant d'indiquer la fin d'une section spécifique pour l'ordonnanceur Loadleveler.
 - # % **Torque_sec_beg** : syntaxe permettant d'indiquer le début d'une section spécifique pour l'ordonnanceur Torque.
 - # % **Torque_sec_end** : syntaxe permettant d'indiquer la fin d'une section spécifique pour l'ordonnanceur Torque.

Note : les paramètres # % **dietoutput** et # % **dieterror** peuvent contenir les variables suivantes :

- \$(**job_name**) : le nom du job
- \$(**lotid**) ou \$(**jobid**) : l'identifiant du lot

3 Fonction de soumission d'un script ou d'un lot

La fonction de l'API `dietSubmitLot` permet de soumettre un script ou un lot :

- `std::string dietSubmitLot(const char* path_to_lot, char*& errorMessage)`
Fonction permettant de soumettre un script ou un lot. Le chemin du fichier `path_to_lot` décrivant le script ou le lot est passé en premier paramètre. La chaîne de caractère contenant le message d'erreurs (`errorMessage`) est passé en deuxième paramètre.
Elle retourne l'identifiant global du lot ou l'identifiant de la tâche soumise décrite dans le script transmis. En cas d'échec lors de la soumission, elle retourne une chaîne vide (de taille égale à zéro).

4 Fonction de construction d'un lot à partir d'identifiants de tâches déjà soumises

La fonction de l'API `dietBuildLot` permet de construire un lot à partir d'identifiants de tâches déjà soumises :

- `std::string dietBuildLot(const char* hostname, const std::vector<std::string> listIDs, char*& errorMessage)`
Fonction permettant de construire un lot à partir d'identifiants de tâches déjà soumises précisés dans le vecteur `listIDs` passé en deuxième paramètre. Ce lot sera construit sur le serveur de nom `hostname` passé en premier paramètre. La chaîne de caractère contenant le message d'erreurs (`errorMessage`)

est passé en troisième paramètre.

Elle retourne l'identifiant global du lot formé. En cas d'échec lors de la soumission, elle retourne une chaîne vide (de taille égale à zéro).

5 Fonction d'affichage de l'état des lots

La fonction de l'API `dietListLot` permet de lister l'état des lots :

– `std::string dietListLot(const char* hostname)`

Fonction permettant d'afficher l'état des lots sur le serveur de nom `hostname` passé en paramètre. Elle retourne les informations contenant l'état ou des éventuelles erreurs.

6 Fonction d'affichage de l'état courant d'un lot

La fonction de l'API `dietStatusLot` permet d'afficher l'état courant d'un lot et les états de l'ensemble des tâches qui le constituent :

– `std::string dietStatusLot(const char* hostname, const char*ID)`

Fonction permettant d'afficher l'état courant d'un lot d'identifiants `ID` passé en deuxième paramètre sur le serveur de nom `hostname` passé en premier paramètre. Elle retourne les informations contenant l'état (états des tâches qui le constituent) courant du lot d'identifiants `ID` ou des éventuelles erreurs.

7 Fonction de suppression d'un lot

La fonction de l'API `dietCancelLot` permet de supprimer un lot en supprimant l'ensemble des tâches qui le constituent :

– `std::string dietCancelLot(const char* hostname, const char*ID)`

Fonction permettant de supprimer un lot d'identifiants `ID` passé en deuxième paramètre sur le serveur de nom `hostname` passé en premier paramètre. Elle supprime aussi l'ensemble des tâches qui le constituent. Elle retourne les informations sur les tâches supprimées du lot d'identifiants `ID` ou des éventuelles erreurs.

8 Fonctions de récupération des résultats et erreurs d'un lot

La fonction de l'API `dietGetOutPutLot` permet de récupérer les résultats et erreurs d'un lot terminé :

– `void dietGetOutPutLot(const char* hostname, const char*ID, char*& output, char*& error)`

Fonction permettant de récupérer les résultats et erreurs du lot d'identifiants `ID` passé en deuxième paramètre sur le serveur de nom `hostname` passé en premier paramètre. La chaîne de caractère `output` passé en troisième argument contiendra les résultats et la chaîne de caractère `error` passé en quatrième argument contiendra les erreurs.

9 Exemples d'utilisation de l'API

Dans cette section nous présentons un programme d'exemple d'utilisation de l'API. Ce programme prend en paramètre un fichier de configuration DIET, le nom abrégé de la fonction à utiliser. Afin de simplifier sa lecture, il est conçu de manière très simple et n'effectue notamment pas tous les contrôles qu'il serait nécessaire d'effectuer. Le fichier de configuration DIET est passé en premier paramètre, le nombre abrégé de

la fonction à utiliser est passé en deux deuxième paramètre. Les arguments suivants diffèrent selon le nom abrégé de la fonction :

- Si le nom abrégé de la commande est **SUBMIT** : Le troisième et dernier argument désigne le chemin menant au script à soumettre ou au lot à soumettre.
- Si le nom abrégé de la commande est **BUILD** : Le troisième argument désigne le nom du serveur sur lequel on veut construire le lot et les arguments suivants désignent les identifiants à partir desquels on veut construire le lot.
- Si le nom abrégé de la commande est **LIST** : Le troisième et dernier argument désigne le nom du serveur sur lequel on veut lister l'état des lots.
- Si le nom abrégé de la commande est **STATUS** : Le troisième argument désigne le nom du serveur sur lequel on veut afficher l'état courant du lot d'identifiants ayant pour valeur le quatrième et dernier argument.
- Si le nom abrégé de la commande est **CANCEL** : Le troisième argument désigne le nom du serveur sur lequel on veut supprimer le lot d'identifiants ayant pour valeur le quatrième et dernier argument.
- Si le nom abrégé de la commande est **OUTPUT** : Le troisième argument désigne le nom du serveur sur lequel on veut récupérer les résultats ou erreurs du lot d'identifiants ayant pour valeur le quatrième et dernier argument.

```
#include <iostream>
#include <string>
#include <sstream>
#include "DIET_client.h"
#include "diet-bmms-lib.h"

int main(int argc, char* argv[]){

    diet_initialize(argv[1], argc, argv);

    char* errorMessage;

    if(strcmp(argv[2], "SUBMIT")==0){

        char* my_lot = argv[3];

        std::string job_id = dietSubmitLot(my_lot, errorMessage);

        if(job_id.size()!=0) std::cout << job_id << std::endl;
        else std::cout << "SUBMIT : " << errorMessage << std::endl;
    }

    if(strcmp(argv[2], "BUILD")==0){

        std::vector<std::string> listIDs;
        for(int i=4; i < argc; i++)
            listIDs.push_back(argv[i]);

        std::string job_id = dietBuildLot(argv[3], listIDs, errorMessage);

        std::cout << "BUILD : " << job_id << std::endl;
    }

    if(strcmp(argv[2], "LIST")==0){
```

```

    std::cout << dietListLot(argv[3]) << std::endl;
}

if(strcmp(argv[2], "STATUS")==0){
    std::cout << dietStatusLot(argv[3], argv[4]) << std::endl;
}

if(strcmp(argv[2], "CANCEL")==0){
    std::cout << dietCancelLot(argv[3], argv[4]) << std::endl;
}

if(strcmp(argv[2], "OUTPUT")==0){

    char* output=NULL;
    char* error=NULL;
    char* output_stream=NULL;

    dietGetOutPutLot(argv[3], argv[4], output, error);

    std::cout << "output message is " << output << std::endl;
    std::cout << "error message is " << error << std::endl;
}

diet_finalize();
}

```