Pre et Post Processus de Toulouse

Les Pre/PostProcessus (PP) sont des scripts permettant d'exécuter des programmes ou des chaînes de traitements sur les données téléchargées.
Sur Toulouse, on utilise principalement 4 PP :

- formatdbTLSE.pl : Formatage des banques blast
- fastacmdTLSE.pl : Génération de fichier fasta à partir d'index blast
- indexSrsTLSE.pl : Formatage des banques pour SRS
- sendMailTLSE.pl : Envoi de mail.

Le langage de ces scripts est libre (ANT, PERL, C, …).

Via les fichiers bank.properties, on peut définir des chaînes de traitement s'exécutant en parallèle ou séquentiellement.

Ex pour les PostProcess :

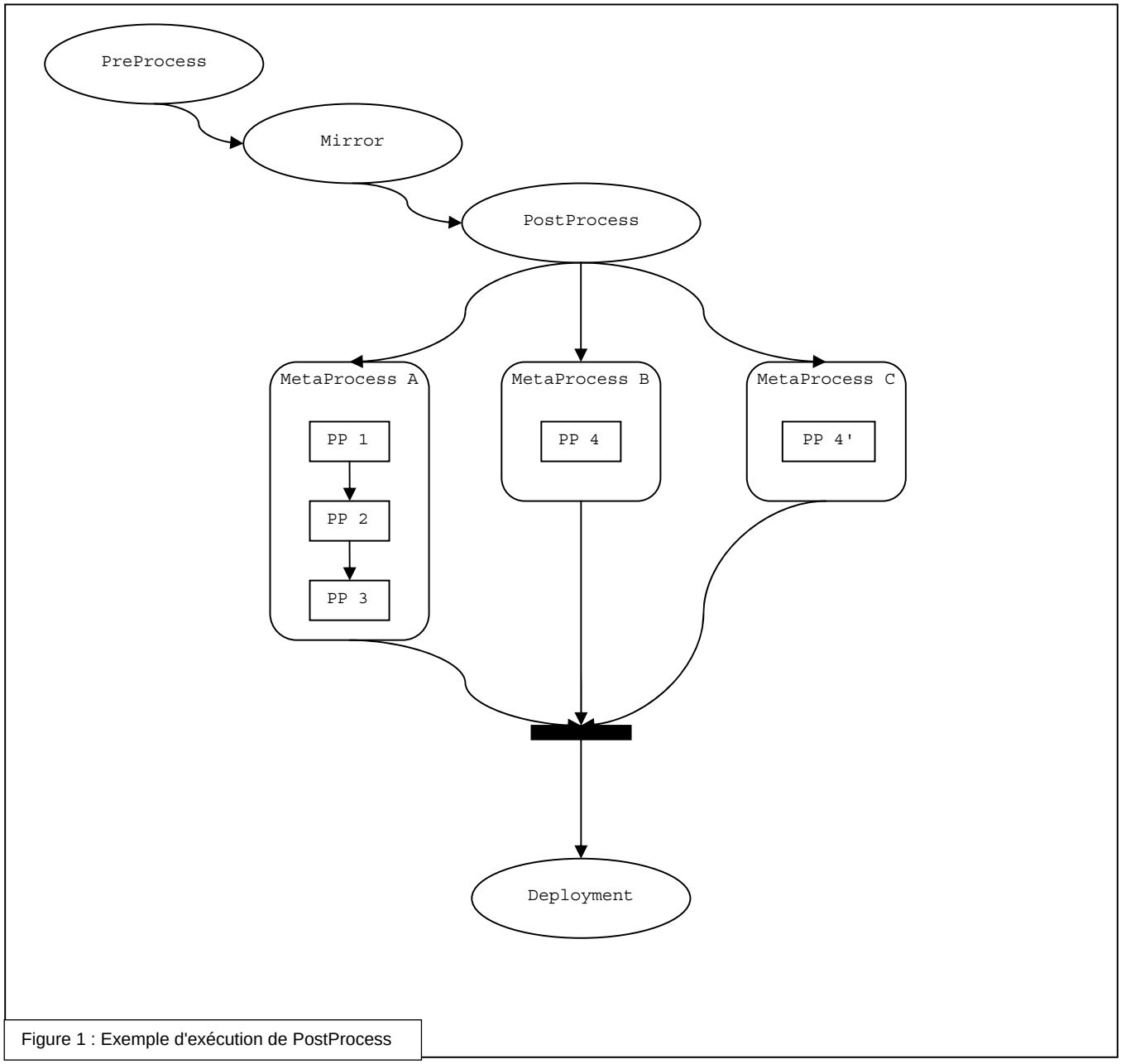

Figure 1 : Exemple d'exécution de PostProcess

Sur la figure 1 est représenté un exemple d'exécution de PostProcess. Pour cette banque, il est défini 3 MetaProcess (enchaînement de PostProcess) s'exécutant en parallèle. Dans le MetaProcess_A, on a 3 PP séquentiels et dans les MetaProcess_B et _C, il y a deux fois le PP_4 avec des paramètres différents.

A - Contraintes de développement des PP

- Les PP sont placés dans le répertoire projectfiles/process.
- Il est recommandé que les PP ne suppriment pas les données rapatriées.
- Pour que Biomaj puisse assurer l'intégrité des données pour une release, les PP doivent communiquer les fichiers produits à Biomaj (cf : Communication Biomaj / PP)

B - Communication Biomaj / PP

Dans les taches "PreProcess" et "PostProcess", Biomaj exécute, via un appel système, des scripts que les utilisateurs ont développé (ou récupéré). Pour que Biomaj puisse gérer et "logger" l'action de ces scripts, une communication entre Biomaj et les PP doit s'établir.

Cette communication se fait via STDOUT et STDERR et en utilisant des "tag" que Biomaj donne aux PP par l'intermédiaire de variables d'environnements.

- PP_WARNING
- PP_DEPENDENCE
- PP_DEPENDENCE_VOLATILE

1 - Gestion des messages

Biomaj peut "logger" des messages dans les fichiers de log spécifiques aux PP lors de l'exécution de ceux-ci. Il existe 3 niveaux de messages que Biomaj sait interpréter en provenance des PP :

1. Le message informatif : Chaîne de caractères sur STDOUT
2. Le message warning: Chaîne de caractères sur STDOUT, préfixée par la variable d'environnement PP_WARNING
3. Le message d'erreur : Chaîne de caractères sur STDERR

En général (et selon les bonnes pratiques de développement des PP), un message d'erreur doit être suivie une commande "exit()", avec un code de retour autre que zéro. Un code de retour zéro indique à Biomaj que le PP c'est bien déroulé. Tout autre code de retour provoquera l'arrêt du cycle de mise à jour de Biomaj.

2 - Gestion des fichiers des PP

On peut distinguer 3 types de fichiers qu'un PP peut être amené à produire :

1. Les fichiers contenant le résultat final du traitement. Ces fichiers font partie intégrante des données de la banque. Leur existence est primordiale (ex : fichiers d'index blast).

2. Les fichiers contenant un résultat intermédiaire (pouvant servir d'entrée à un autre PP, mais ne correspond pas à un résultat final). Ces fichiers ont un rôle transitoire et uniquement lors du processus de mise à jour de la banque. Ils peuvent être effacés une fois la banque mise en production.

3. Tous les fichiers ne contenant pas de données sensibles. La présence de ces fichiers pendant et après la mise à jour de la banque n'est indispensable (ex : fichier de log comme formatdb.log)

Pour que Biomaj intègre correctement les fichiers des PP à la release en cours de mise à jour de la banque, les PP doivent indiquer à Biomaj le statut des fichiers qu'ils produisent.

Pour les fichiers du premier groupe (1), il faut écrire sur la sortie standard (STDOUT) le chemin complet du fichier, préfixé par la variable d'environnement PP_DEPENDENCE. Ainsi, Biomaj considèrera ces fichiers comme faisant partie intégrante de la release de la banque (indispensable pour assurer l'intégrité des données).

Pour les fichiers du deuxième groupe (2), il faut écrire sur la sortie standard (STDOUT) le chemin complet du fichier, préfixé par la variable d'environnement PP_DEPENDENCE_VOLATILE. Ces fichiers seront conservés sur le système de fichiers jusqu'à la mise en production de la banque. Ainsi si une erreur intervient avant cette étape, les résultats intermédiaires ne seront pas recalculés. Une fois la banque mise ne production, Biomaj supprime ces fichiers.

Pour le troisième groupe de fichiers (3), il n'y a rien à faire de particulier. Ce sont les PP qui gèrent ces fichiers. Biomaj n'aura pas de contrôle sur eux.

C - Librairie ProcessBiomajLib.pm

La librairie "ProcessBiomajLib.pm" propose une liste de fonctions pour le développement des Process (PERL). (Pour plus de détails, voir directement de code source ou le fichier ProcessBiomajLib.html)

1. Configuration

- **checkBiomajEnvironment() :** Vérifie l'existence des variables d'environnement nécessaire aux Process.
- **readCfgFile() :** Lit le fichier de conf $UNIX_COMMAND_SYSTEM_CFG.
  Place une Clé/Valeur dans $rh_cmd pour chaque ligne CLE=valeur du fichier
  Définit des valeurs par défaut pour les commandes système non définit dans le fichier de conf.

2. Les messages

- **Info($msg)** : Pour envoyer un message informatif
- **Warning($msg)** : Pour un message d'alerte
- **Error($msg)** : Pour un message d'erreur. Cette fonction se termine par **exit(-1);** ce qui arrêtera l'exécution du MetaProcess et provoquera l'arrêt sur erreur de Biomaj

3. Les fichiers

3 fonctions sont proposés pour la gestion des fichiers de sortie du Process.

- **clearOutputFiles($tag) :** Vide la liste de fichiers de sortie. $tag correspond à la liste à vider.
  "dependence"-> vide la liste des fichiers résultat
  "volatile" -> vide la liste des fichiers volatiles
  "all" -> vide les deux listes (valeur par défaut si $tag = "")
- **outputFile($file,$tag) :** Ajoute $file a la liste selon $tag (par défaut, $tag="dependence").
- **printOutputFile($tag) :** Affiche sur STDOUT la liste des fichiers selon $tag (par défaut, $tag="all")

4. Répertoires

- **getFlatDir :** Retourne le nom du répertoire "flat"
- **getLogDir :** Retourne le nom du répertoire "log"
- **getCurrentLink :** Retourne le nom du lien "current"
- **getFuturReleaseLink :** Retourne le nom du lien "futur_release"
- **getRemoteRelease :** Retourne la release de la banque en cour de mise a jour
- **getBlastDbDir :** Retourne le nom du répertoire par défaut pour "blastdb"
- **getPathBlastDbDir :** Retourne le chemin absolu du répertoire pour les alias blast
  Retourne ENV{BLASTDB} si elle est renseignée
  Retourne ENV{'datadir'}/".&getBlastDbDir() sinon
- **getPathDirVersion :** Retourne le chemin absolu du répertoire de production de la banque
- **getPathFuturReleaseLink :** Retourne le chemin absolu du lien "futur_release"
- **getPathFuturReleaseDirName :** Retourne le chemin absolu du répertoire pointe par le lien futur_release
- **getPathFuturReleaseFlatDir :** Retourne le chemin absolu du répertoire "flat" de la future release
- **getPathFuturReleaseLogDir :** Retourne le chemin absolu du répertoire "log" de la future release
- **function getPathFuturReleaseMyDir($bank) :** Concatène $myDir au chemin absolu du répertoire contenant la release au cours de mise à jour.

5. **Exécution de commandes systèmes**

- **executeCmdSystem :** Exécute la ligne de commande et vérifie que la valeur de retour égale 0. Sinon &Error().
- **executeBatch :** Exécute une suite de commandes système via différentes méthodes (sh ou cluster)
- **executeGetz :** Exécute une requête ("$request") getz. Le résultat est dans le fichier $outputfile passé en argument.

6. **Utils**

- **getSequenceType :** Détermine le type de séquence d'un fichier fasta.

D – Les Process utilisés sur Toulouse

Pour la documentation sur les process utilisés à Toulouse, voir les commentaires dans le code source ou les fichiers .html associés à chaque process.

formatdbTLSE.pl
fastacmdTLSE.pl
indexSrsTLSE.pl
sendMailTLSE.pl
diffDbestSurf.pl