ComPAS'2014 : Parallélisme / Architecture / Système
Neuchâtel, Suisse, du 23 au 25 avril 2014

Comment reproduire les résultats de l'article : *POP-Java : Parallélisme et distribution orienté objet*

Beat Wolf[1], Pierre Kuonen[1], Thomas Dandekar[2]
[1]iCoSys, Haute École Spécialisée de Suisse occidentale, Fribourg
[2]Biozentrum, Universität Würzburg

beat.wolf@hefr.ch, pierre.kuonen@hefr.ch, dandekar@biozentrum.uni-wuerzburg.de

---

Résumé
Cet article, qui est une extension de l'article *POP-Java : Parallélisme et distribution orienté objet*, présente la manière dont les résultats de l'article original peuvent être reproduits. L'environnement de test dans lequel les mesures ont été effectuées est présenté en détail et une marche à suivre pour mettre en place cet environnement et reproduire les résultats est donné. Ceci doit permette à toute autre chercheur de reproduire les résultats présentés dans l'article original *POP-Java : Parallélisme et distribution orienté objet*.

Mots-clés : Parallélisme, Distribution de calcul, Java, Langage de programmation OO, Reproductibilité

---

1. Introduction

La publication *POP-Java : Parallélisme et distribution orienté objet* présente POP-Java [2], une implémentation du modèle POP [1] dans le langage de programmation Java. Les performances de POP-Java sont comparées à celle de la technologie RMI [3] présente en standard dans Java depuis sa version 5. Ceci a été fait en comparant les performances de deux implémentations d'un même algorithme, une en POP-Java et une en Java pur utilisant la technologie RMI. L'algorithme utilisé est un algorithme d'alignement de séquences d'ADN humaines contre une référence. Il s'agit d'un problème courant dans le domaine de la bioinformatique. Dans le présent article, la méthode à suivre pour mettre en place l'environnement dans lequel ces deux implémentations peuvent être testées est décrite, ainsi que la manière de reproduire les résultats présentés. Le code source de l'application de test utilisé pour comparer les performances de POP-Java et RMI est mis à disposition dans une archive. Il est à noter que la licence du code source de l'algorithme d'alignement ne permet pas son utilisation en dehors du cadre des tests liés à cet article.

2. Environnement

Les tests ont été exécutés sur une grappe de machines (cluster) située à l'école d'ingénieurs de Fribourg, Suisse. Comme décrit dans l'article original, les machines de ce cluster ont des processeurs à quatre cœurs cadencées à 3.4 GHz et 8 GO de mémoire vive. L'hyperthreading a été désactivé et les machines sont connectées par un réseau de 1 Gb/s. Le système d'exploitation utilisé est "Fedora release 17", 64 bit. La version Java, utilisée par les deux implémentations,

est 1.7.0_17, 64 bit. Les machines se partagent un répertoire `home` au travers NFS (Network File System). Il est à signaler que, pour l'instant, POP-Java n'est supporté que sur des plate-formes de type UNIX.

3. Pré-requis

Les pré-requis nécessaires pour installer l'application sont : Une machine virtuelle Java (7+), ant(1.8+), et POP-Java. Les deux chapitres suivants décrivent l'installation de ces pré-requis. Dans le cadre de l'environnement de test utilisé, toutes les commandes de cet article peuvent être exécutées à distance au travers d'une connexion SSH.

3.1. Java & Ant

Pour l'installation de Java et ant, veuillez consulter la documentation de votre distribution linux. Pour la distribution Fedora, veuillez suivre les explications données dans [7] et [8].

3.2. POP-java

L'installation de POP-Java se fait en téléchargeant la version utilisée pour l'écriture de cet article. L'archive mise à disposition contient les codes sources de POP-Java, qui doivent être compilés. Il faut extraire l'archive, ouvrir le répertoire ainsi créé puis compiler le code. Pour ceci exécutez les commandes suivantes :

```
wget http://beat.wolf.home.hefr.ch/realis/popjava.tar.gz
tar -zxvf popjava.tar.gz
cd popjava
ant
```

Après avoir compilé les codes sources, lancez la commande :

```
./install
```

et suivez les instructions affichées à l'écran. POP-Java doit être installé dans le répertoire `home` pour être accessible depuis toutes les machines utilisées pour le test. Pour ceci, spécifiez le répertoire `/etuhome/ggroup8/popjavainstall` comme répertoire d'installation. A la fin de l'installation il est demandé de copier des lignes dans le fichier `.bashrc`. En plus de la copie de ces lignes, il faut, dans ce même fichier, mettre en commentaire la ligne suivante : `[ -z "$PS1" ] && return` qui se trouve au début du fichier `.bashrc`. Pour que les changements apportés soient pris en compte, il faut fermer la sessions SSH et se reconnecter à nouveau.

4. Compilation de l'application

Le code source des deux implémentations est réuni dans une même archive, car beaucoup de code est commun aux deux implémentations. Pour compiler les deux implémentations il faut télécharger le code source. Dans notre exemple nous avons un répertoire nommé `realis` dans lequel nous téléchargeons les codes sources. Voici les commandes permettant de créer le répertoire et télécharger le code (exécuté depuis le répertoire `home` de l'utilisateur) :

```
cd
mkdir realis
cd realis
wget http://beat.wolf.home.hefr.ch/realis/caligner.tar.gz
tar -zxvf caligner.tar.gz
```

Dans le répertoire ainsi créé `/realis/caligner`, il faut compiler les codes sources. Ceci se fait avec la commande :

```
cd ~/realis/caligner
ant
```

5. Données

Les deux applications de test utilisent le même jeu de données. Comme indiqué dans l'article original, les données consistent en 16 millions de séquences d'ADN au format *FASTQ* [5]. Elle sont alignées contre la référence du chromosome humain 7 au format *FASTA* [4]. Le fichier *FASTQ* est compressé avec *gzip*. Pour télécharger le fichier de référence, un répertoire `Data` est créé dans le répertoire `home`, puis le téléchargement est lancé par la commande :

```
cd
mkdir Data
cd Data
wget http://beat.wolf.home.hefr.ch/realis/chr7_hg19.fasta
```

Dans le même répertoire on télécharge le fichier des données brutes :

```
wget http://beat.wolf.home.hefr.ch/realis/chr7Long.fastq.gz
```

6. Tests de performances

Pour exécuter les tests de performance, des scripts sont mis à disposition pour lancer de manière automatisée les tests sur 1 à 6 machines en mesurant les temps de calcul. Dans le cadre de ces tests, plusieurs particularités sont à mentionner. Les machines du cluster de test partagent tous le même répertoire `home` au travers d'un disque réseau NFS. Étant donné que les performances de ce disque réseau ne peuvent pas être garanti, le disque dur local est utilisé pour stocker les données critiques pour la performance. C'est pourquoi, le fichier de données brutes doit être copié dans le répertoire `tmp` de la machine sur laquelle les tests sont effectués. Le fichier de résultats est aussi écrit dans le répertoire `tmp`, ceci pour garantir une vitesse d'accès relativement constante. Le fichier de référence du chromosome 7 peut lui être stocké sur le disque réseau NFS, car il ne sera lu qu'une seule fois au début de l'application. Afin d'avoir des résultats cohérents et reproductibles, les tests sont lancés sur une machine dédié à la lecture et à l'écriture. Le calcul lui-même est fait sur d'autres machines. Dans notre environnement de test, la machine 160.98.22.10 est la machine utilisée pour lancer les tests, et les machines 160.98.22.11 à 160.98.22.16 sont les machines qui font effectivement le travail d'alignement.

7. Mise en place

Avant de lancer les tests, il faut copier le fichier de données brutes dans le répertoire `tmp`. Ceci est fait avec la commande suivante :

```
cp ~/Data/chr7Long.fastq.gz /tmp/
```

Il faut ensuite faire une phase de mise en place pour l'implémentation en RMI. Pour ceci le script `setupRMI.sh` doit être lancé dans le répertoire où se trouve l'aligneur (par défaut `/realis/caligner`) en utilisant la commande suivante :

```
cd ~/realis/caligner/
./setupRMI.sh
```

8. Réalisation des tests

Il y a deux scripts pour lancer les tests de performance, un pour l'implémentation en POP-Java et un pour l'implémentation en RMI. Les deux scripts prennent comme paramètre le nombre de fois que les tests doivent être exécutés. Pour lancer les tests pour les deux implémentations, voici les commandes à exécuter dans le répertoire des sources :

```
./benchmarkPOPJava.sh 3
./benchmarkRMI.sh 3
```

Dans cet exemple, les tests sont effectués 3 fois en utilisant de 1 à 6 machines. Les résultats de ces tests se trouvent dans les fichiers nommés `popjavaResultsX.txt` et `rmiResultsX.txt`, ou `X` est un chiffre entre 1 et 6. Les résultats qui se trouvent dans ces fichiers peuvent être copiés dans une feuille Excel préparé pour l'analyse. Cela permet de calculer rapidement la courbe d'accélération (speedup) pour les deux implémentations. Une telle feuille Excel est disponible à l'URL suivante :
`http://beat.wolf.home.hefr.ch/realis/RealisStats.xls`
Le temps de référence de chaque implémentation sur un certain nombre de machines est calculé comme la médiane des valeurs mesurées, afin de lisser d'éventuelles perturbations ponctuelles.

9. Validation des résultats

Pour vérifier que les deux implémentations produisent bien le même résultat, les alignements générés doivent être comparés. Les alignements sont enregistrés dans un fichier au format SAM non ordré. Pour comparer les résultats, il les faut convertir au format BAM ordré. Le format BAM est la version binaire du format SAM [6]. Pour convertir les fichiers, on peut utiliser le script de conversion qui se trouve dans le répertoire des sources de l'aligneur. Il prend comme premier paramètre le fichier d'entrée (SAM) et comme deuxième paramètre le fichier de sortie (BAM). Voici la commande pour convertir les deux fichiers générés :

```
./samCheck.sh /tmp/pop.sam /tmp/pop.bam
./samCheck.sh /tmp/rmi.sam /tmp/rmi.bam
```

Dans notre exemple, l'implémentation en POP-Java a créé le fichier `/tmp/pop.sam` et l'implémentation RMI le fichier `/tmp/rmi.sam`.
Après avoir converti les résultats nous pouvons vérifier, que les deux fichiers générés sont identiques. Ceci peut être fait en vérifiant leurs tailles et leurs hash md5, calculés avec les commandes suivantes :

```
md5sum /tmp/pop.bam
md5sum /tmp/rmi.bam
```

10. Conclusion

Dans ce document nous avons documenté les étapes nécessaires pour reproduire les résultats présentés dans l'article *POP-Java : Parallélisme et distribution orienté objet* publié dans le track *Parallélisme* de la conférence *ComPAS'2014*.

Bibliographie

1. T. A. Nguyen, P. Kuonen, A Model of Dynamic Parallel Objects for Metacomputing. The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications, 2002, Las Vegas, Nevada, USA.
2. POP-Java, http ://gridgroup.hefr.ch/popc/doku.php/popjava
3. RMI, Remote Method Invocation, http ://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html
4. FASTA fileformat, http ://en.wikipedia.org/wiki/FASTA_format
5. Peter J. A. Cock et al., The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucl. Acids Res. (2010) 38 (6) : 1767-1771.
6. Li H.*, Handsaker B.*, Wysoker A., Fennell T., Ruan J., Homer N., Marth G., Abecasis G., Durbin R. and 1000 Genome Project Data Processing Subgroup (2009) The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics
7. https ://fedoraproject.org/wiki/Java
8. https ://access.redhat.com/site/documentation/en-US/JBoss_Enterprise_SOA_Platform/4.3/html/Getting_S install_ant.html