Comparaison des performances XPath avec plusieurs langages

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 17 octobre 2006. Dernière mise à jour le 18 octobre 2006

https://www.bortzmeyer.org/xpath-performance.html

J'ai un programme qui tourne beaucoup trop lentement. Il semble qu'une des tâches essentielles qu'il accomplit soit d'extraire des valeurs d'un document XML en utilisant XPath. Comment accélerer cette extraction ?

En matière de performances des systèmes informatiques, il ne sert à rien de spéculer, rares sont les informaticiens qui peuvent trouver, sans se tromper, quel est le goulet d'étranglement d'un programme. Il faut mesurer <https://www.bortzmeyer.org/mesurer-temps-execution.html>. L'informatique est une science expérimentale.

J'écris donc le même programme dans trois langages. Le programme va extraire le texte d'un fichier XML, en prenant en paramètre une expression XPath, ici `//date/text()` (le texte d'un élément XML `<date>`).

Le premier programme (en ligne sur `https://www.bortzmeyer.org/files/xpath.py`) est écrit (par moi) en Python :

```
% time  python xpath.py '//date/text()'  fichier.xml
2006-03-23
python xpath.py '//date/text()' fichier.xml  0.55s user 0.04s system 98% cpu 0.597 total
```

Son exécution complète (initialisation de Python, chargement du programme, puis du fichier XML, recherche XPath, etc) a pris plus d'une demi-seconde ! (Le temps total écoulé est affiché à la fin du résultat de `time`.) Ce temps a été presque entièrement consommé en temps utilisateur, donc à dérouler du code Python (le fichier XML fait 10 ko, donc peu d'entrées/sorties sont nécessaires).

Le second programme (en ligne sur `https://www.bortzmeyer.org/files/xpath.pl`) est écrit (il est distribué avec le module `XML::XPath`) en Perl :

```
% time perl ./xpath.pl -e '//date/text()'  fichier.xml
Found 1 nodes in fichier.xml:
-- NODE --
2006-03-23
perl ./xpath.pl -e '//date/text()' fichier.xml  0.20s user 0.02s system 97% cpu 0.220 total
```

Il ne prend que la moitié du temps consommé par son prédécesseur.

Le troisième programme (en ligne sur `https://www.bortzmeyer.org/files/xpath.c`) est écrit (par Aleksey Sanin, et il est distribué avec libxml2, mais je l'ai modifié) en C :

```
% gcc -o xpath `xml2-config --cflags` `xml2-config --libs` xpath.c
% time ./xpath  fichier.xml '//date/text()'
2006-03-23
./xpath fichier.xml '//date/text()'  0.00s user 0.00s system 91% cpu 0.004 total
```

Cette fois, le temps d'éxécution est à peine visible.

Comme tous les "*benchmarks*", celui-ci est très contestable : dans les langages comme Perl ou Python, le temps de chargement de l'interpréteur et éventuellement le temps de compilation du programme peuvent être importants et un programme qui ne fait qu'une seule recherche XPath est donc probablement peu efficace. Si on avait plusieurs recherches successives, l'écart se resserrerait sans doute. D'autre part, les performances du programme C ne doivent pas faire oublier sa longueur (cinq fois plus long que le programme Python et probablement bien plus de cinq fois plus difficile à maintenir).

D'autre part, on n'est pas obligé de faire du tout-C ou du tout-Python. Par exemple, dans ce cas, comme dans beaucoup d'autres, on peut appeler du C depuis Python (l'interface entre une bibliothèque C et un programme Python est très facile à réaliser, bien plus qu'en Perl). C'est ce que fait notre quatrième programme (en ligne sur `https://www.bortzmeyer.org/files/xpath-libxml2.py`) qui utilise les "*bindings*" Python de libxml2 :

```
%  time  python xpath-libxml2.py '//date/text()' fichier.xml
2002-03-23
python xpath-libxml2.py '//date/text()' fichier.xml  0.04s user 0.00s system 96% cpu 0.041 total
```

On le voit, il est bien plus rapide que le programme en Python pur (mais reste plus lent que le programme en C). Si on souhaite, ce qui est mon cas, réaliser la structure principale du programme en Python, cette bibliothèque permet d'accélerer les parties les plus lentes, tout en évitant de programmer soi-même dans un langage difficile comme C.

Une autre solution très courante aux problèmes de performance, lorsque la même donnée est demandée régulièrement, est l'utilisation d'un cache. Celui-ci stocke le résultat de la requête précédente, lui permettant ainsi d'être plus rapide (si, comme c'est le cas ici, le temps d'accès au cache est faible devant le temps de calcul). C'est ce que démontre notre cinquième programme (en ligne sur `https://www.bortzmeyer.org/files/xpath-cache.py`) qui utilise un fichier pour stocker le résultat précédent :

```
% time python xpath-cache.py '//date/text()' fichier.xml
2002-03-28
python xpath+cache.py '//date/text()' fichier.xml  0.29s user 0.03s system 99% cpu 0.325 total

% time python xpath-cache.py '//date/text()' fichier.xml
2002-03-28
python xpath+cache.py '//date/text()' fichier.xml  0.02s user 0.00s system 81% cpu 0.020 total

% time python xpath-cache.py '//date/text()' fichier.xml
2002-03-28
python xpath+cache.py '//date/text()' fichier.xml  0.01s user 0.01s system 99% cpu 0.020 total
```

La première requête est lente mais toutes les suivantes sont bien plus rapides. Si on pose plusieurs fois la même question, le cache peut donc apporter une solution.

Le but de cet article n'était pas de donner un conseil presse-bouton (il faut toujours réfléchir, pour le cas particulier qu'on a à traiter) mais de montrer que les différences de performances peuvent être considérables et que le choix des bonnes techniques peut sérieusement accélérer un programme.