Le protocole d'accès au serveur de PostgreSQL

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 octobre 2010

https://www.bortzmeyer.org/protocole-postgresql.html

Alors que tout l'Internet repose sur des protocoles normalisés, décrits en détail dans des documents stables et accessibles publiquement (les RFC), il existe un domaine où il n'y a pas de protocole réseau standard : l'accès aux serveur de base de données. Que les logiciels soient libres comme MySQL ou PostgreSQL ou bien privateurs comme Oracle, le protocole de communication entre le client et le serveur est complètement spécifique. À quoi ressemble celui de PostgreSQL ?

Certains points sont plus ou moins normalisés. Par exemple, le langage des requêtes, SQL a fait l'objet de normes, malheureusement non accessibles publiquement, ce qui supprime une bonne partie de son intérêt (et aide à comprendre la sérieuse divergence des mises en œuvre de SQL). En outre, cette norme est très difficile à déchiffrer <https://www.bortzmeyer.org/sql-standard.html>.

De même, pour une communication entre une application et le client qui tourne sur la même machine, il existe des solutions plus ou moins standardisées comme ODBC ou JDBC. Mais sur le câble, sur le vrai fil où passe les paquets, quel est le protocole ? Eh bien, cela dépend du SGBD.

Il y a des gens qui trouvent que c'est mieux comme cela. Je me souviens d'avoir entendu un commercial Oracle, il y a quelques années, expliquer que le fait qu'Oracle utilise un protocole non-standard protégeait contre les "*sniffers*" qui, autrement, auraient pu capturer le mot de passe. Pour voir à quel point c'est idiot, il suffit d'essayer l'excellent dSniff, qui décode le protocole d'Oracle (le source est dans le fichier `decode_oracle.c`), Net8 (autrefois appelé SQL*Net). (Un petit piège est documenté dans la FAQ <http://monkey.org/~dugsong/dsniff/faq.html#Why%20isn\%27t%20dsniff%20capturing%20Oracle%20logins>.)

Bon, et PostgreSQL, comment fait-il ? Il a aussi un protocole privé, mais très bien documenté <http://www.postgresql.org/docs/current/interactive/protocol.html>. Si vous voulez un exemple réel pour vous instruire, vous pouvez analyser une session PostgreSQL avec Wireshark (qui a un excellent décodeur PostgreSQL) ou bien tout simplement regarder en ligne avec les bons moyens d'affichage de pcapr <https://www.bortzmeyer.org/pcapr.html> une session PostgreSQL complète <http://www.pcapr.net/view/bortzmeyer+pcapr/2010/9/5/5/postgresql.pcap.html>.

Bon, le protocole est documenté. Mais est-il suffisamment stable pour qu'on puisse créer des clients natifs, plutôt que de passer par la bibliothèque libpq <http://www.postgresql.org/docs/current/interactive/libpq.html> voire carrément par le client en ligne de commande psql <http://www.postgresql.org/docs/current/static/app-psql.html>, avant analyse du texte produit ? Les différents mises en œuvre de bibliothèque PostgreSQL ont suivi des chemins différents. Par exemple, pour Emacs, il existait un excellent mode psql-mode <http://www.hgsc.bcm.tmc.edu/~harley/elisp/> (qui semble avoir disparu de son site originel mais dont on trouve encore des copies par-ci par-là), qui fonctionne en lançant la commande psql puis en analysant le résultat. Ou bien il existe une bibliothèque native, pg.el <http://www.online-marketwatch.com/pgel/pg.html>, entièrement en Emacs Lisp.

De même, pour tout nouveau langage de programmation, se pose la question de la réalisation d'une bibliothèque PostgreSQL. Appeler libpq, ce qui oblige à dépendre de C ? C'est ce que fait, pour Python, la bibliothèque psycopg <http://initd.org/psycopg/>. Ou bien tout faire soi-même ? Pour Go, le premier effort de création d'une telle bibliothèque, go-pg <http://github.com/oibore/go-pg>, utilisait la libpq (ce qui faisait perdre un certain nombre de propriétés de Go comme la gestion complète de la mémoire). Le deuxième projet, bien plus perfectionné, go-pgsql <http://github.com/lxn/go-pgsql> a choisi la voie native et semble très bien fonctionner.

En parlant de Go <https://www.bortzmeyer.org/go-langage.html>, j'ai fait moi-même, pour apprendre le protocole, un petit programme qui se connecte à un serveur distant, effectue une requête simple et récupère les résultats. Le protocole est un protocole binaire plutôt simple, avec des règles très cohérentes partout. Mais j'ai découvert que le protocole n'était pas complètement spécifié, notamment pour les méthodes d'authentification. Ainsi, la documentation de l'authentification MD5 <http://www.postgresql.org/docs/current/interactive/protocol-flow.html> ne précise pas du tout qu'il faut concaténer le mot de passe avec le nom, ni qu'il faut faire deux hachages successifs. Cela, on ne l'apprend qu'en lisant le code du serveur ou de la libpq (ou, dans mon cas, en lisant le source Lisp de la bibliothèque Emacs citée plus haut). Bref, si cela vous amuse, mon petit programme est en (en ligne sur https://www.bortzmeyer.org/files/test-protocol-postgresql.go). Ce n'est qu'un programme d'exploration (pas de factorisation du code, pas de machine à états, etc), n'en attendez pas plus.

Vous pouvez aussi lire un article expliquant pourquoi il est recommandé d'utiliser la libpq. <http://momjian.us/main/blogs/pgblog/2012.html#March_21_2012>.