−Table des matières

PostgreSQL 8.3 : quoi de neuf ?

Cet article, écrit par Guillaume Lelarge, a été publié dans le magazine GNU/Linux Magazine France, numéro 103 (Mars 2008). Il est disponible maintenant sous licence Creative Commons.

Le serveur de bases de données PostgreSQL existe depuis plus de dix ans. Plus les années passent, plus le rythme de développement s'accélère. Les fonctionnalités sont de plus en plus dirigées vers l'utilisation de PostgreSQL en entreprises et dans des contextes critiques. Cette tendance est surtout visible depuis la version 8.0

Cette version 8.0, datant de début 2005, est la première version PostgreSQL native sous Windows. C'est aussi la première version à disposer des tablespaces, qui permettent de répartir facilement la charge des entrées/sorties disque sur plusieurs disques et des points de retournement (« savepoint ») qui permettent de sauvegarder l'état d'une transaction à un instant t et d'y revenir en cas de problème. Mais la fonctionnalité majeure de cette version reste à venir : la technologie PITR (« Point In Time Recovery ») est implantée. Elle améliore sensiblement la robustesse de PostgreSQL face à des crashs et va donner lieu à un grand nombre de développement dans les versions suivantes. Cette technologie permet aux administrateurs de sauvegarder les journaux de transaction (appelés XLOG sous PostgreSQL). Il est ainsi possible de les rejouer pour revenir à un instant t de l'exécution des transactions.

Après une aussi grosse version, les développeurs de PostgreSQL n'ont pas relâché leurs efforts. La version 8.1 apparue fin 2005 a permis d'intégrer la notion de rôle qui supplante utilisateur et groupe. Elle est aussi capable de gérer plusieurs types de paramètres dans les fonctions : en entrée (« IN »), en sortie (« OUT ») et en entrée/sortie (« INOUT »). Cela facilite grandement le développement de procédures stockées renvoyant une (ou plusieurs) ligne(s). Le module contrib « autovacuum » est intégré au cœur de PostgreSQL permettant une gestion plus simple et mieux contrôlée de la maintenance des tables. De gros travaux sont réalisés par Sun dans le domaine des performances, notamment en ce qui concerne les machines multi-processeurs. Une meilleure gestion du partitionnement des tables est ajoutée, amenant d'autres types de performances et permettant la gestion de bases plus conséquentes.

La dernière version stable, numérotée 8.2, sortie fin 2006, a apporté de nombreuses améliorations et fonctionnalités principalement dédiées aux administrateurs. Un premier niveau de support de SQL:2003 voit arriver les fonctions statistiques, la clause « RETURNING » dans les instructions de modifications de données, les alias dans les instructions UPDATE et DELETE, une gestion étendue de la clause VALUES. Cette version voit aussi arriver une meilleure prise en compte des extensions SQL connues : ajout de la clause « IF EXISTS », « COPY » à partir d'un « SELECT ». Cela permet un portage facilité à partir d'autres SGBD. Le support de Windows sera suffisamment amélioré pour que le portage Windows soit enfin considéré stable. L'authentification devient possible à partir d'un annuaire LDAP. La technologie PITR permet la naissance du « Log Shipping » de base sous PostgreSQL. Le « Log Shipping » est un système de réplication via la recopie et la ré-exécution immédiate des journaux de transactions. Ce système se caractérise surtout par sa grande facilité de mise en œuvre mais demande l'écriture d'un script.

Un an ayant passé depuis la version 8.2, il n'est pas étonnant de constater que la version en cours de développement, la 8.3, est en Release Candidate actuellement. Sa sortie devrait bientôt arriver mais nous pouvons déjà nous pencher sur les nombreuses nouvelles fonctionnalités. En voici une liste très résumée :

- de nouvelles fonctionnalités pour les utilisateurs :
  - support de la recherche plein texte ;
  - support du XML ;
  - support des types enum et uuid ;
- améliorations des commandes SQL (pour les utilisateurs et administrateurs) ;
- de nouveaux outils pour le développeur :
- curseurs modifiables et déplaçables ;
- fonctions PL/pgsql renvoyant des tables ;
- de nouveaux outils pour l'administrateur :
- support de l'authentification GSSAPI/SSPI ;
- journalisation applicative au format CSV ;
- configuration spécifique possible pour les fonctions ;
- autovacuum parallélisable grâce aux « autovacuum worker » ;
- de meilleures performances :
- technologie HOT ;
- enregistrement asynchrone ;
- parcours séquentiels synchronisés ;
- meilleure gestion des identifiants de transaction.

Cet article va parcourir rapidement ces fonctionnalités pour vous donner un avant goût de ce qui vous attend avec la version 8.3.

Pour les utilisateurs

Dès la version 7.4, PostgreSQL dispose d'un module de recherche plein texte. Ce dernier, nommé Tsearch2, est rapidement adopté par de nombreux utilisateurs car il comble un manque dans PostgreSQL. Les recherches de texte du type LIKE, ILIKE (un LIKE insensible à la casse), ~ (opérateur sur des expressions rationnelles) et ~* (le même opérateur, mais insensible à la casse) ne suffisent pas pour des recherches en langue naturelle. L'idée qui tourne autour de cette recherche est qu'une personne fait généralement une recherche sur ce que signifie un mot, et non pas sur cet ensemble particulier de lettres assemblés dans ce sens. Une recherche standard du terme « rechercheront » ramène tous les documents qui contiennent ce mot précis. Une recherche plein texte ramène tous les documents contenant la racine de « rechercheront », donc « recherch ». Cela permet de trouver « rechercheront » mais aussi « rechercher », « rechercha », etc. Ce système de recherche plein texte est essentiel pour les nouveaux systèmes d'information. Il est devenu indispensable pour les moteurs de bases de données. De plus, sa disponibilité en module contrib a gêné son utilisation, beaucoup d'utilisateurs pensant qu'un module contrib est un morceau de code insuffisamment testé pour faire partie du cœur de PostgreSQL. Rien n'était plus faux. Du coup, le code a été intégré au moteur. Cela va faciliter son utilisation. Deux nouveaux types sont disponibles, sans parler d'opérateurs et de fonctions permettant de manipuler ces types. Les opérateurs sont utilisés principalement pour la recherche alors que les fonctions permettent de gérer la conversion des nouveaux types et de récupérer des informations comme le score d'un document par rapport à une recherche, un surlignage des termes trouvés dans un document d'après une recherche.

Un module XML est aussi disponible depuis quelques versions, tout à fait comme le module Tsearch2. Il est intégré au cœur du moteur pour la version 8.3. Cela s'est fait en ajoutant un nouveau type de données, xml, et les fonctions et opérateurs relatifs à ce type. Le but est de pouvoir créer, modifier un document XML valide. Cela passe aussi par des fonctions d'encodage pour les données binaires. Enfin, les fonctionnalité les plus intéressantes concernent l'utilisation d'XPath pour les traitements de valeurs de type XML et la transformation d'objets de bases de données, de requêtes et de curseurs en XML.

D'autres nouveaux types sont apparus, dont le tant attendu ENUM. Ce type est fréquemment demandé, notamment par les utilisateurs migrant de MySQL vers PostgreSQL et par les développeurs voulant créer un produit pouvant utiliser plusieurs moteurs de bases de données. Un type ENUM est un type de données comprenant un nombre limité et statique de valeurs citées dans un ordre précis. Les jours de la semaine ou le sexe d'une personne sont de bons exemples. La gestion interne de ce type permet de s'assurer que seuls les éléments d'un type ENUM seront utilisés dans les colonnes ENUM. Le tri est aussi géré suivant l'ordre de citation des éléments à la création du type. Le type UUID a été ajouté pour permettre la gestion d'un identifiant universel (donc à priori valide sur tous les clusters PostgreSQL, contrairement aux OID qui étaient autrefois utilisés pour compenser ce manque). Ce type a été créé en suivant la RFC 4122.

Certaines commandes SQL ont été améliorées. Pour des raisons de compatibilité avec d'autres serveurs de bases de données, la modification la plus importante se situe au niveau du tri. Auparavant, un tri sur une colonne plaçait automatiquement les valeurs NULL en fin de table. Avec la version 8.3, les valeurs NULL peuvent se trouver en début ou en fin de table. Il suffit d'indiquer sa préférence avec la clause « NULLS { FIRST | LAST } ». Ces options sont aussi disponibles pour la création d'un index. Toujours pour les index, les clauses ASC et DESC sont utilisables pour chaque colonne indexée. Cela permet l'utilisation des index lors d'un tri et cela permet aussi qu'une clause ORDER BY … LIMIT s'exécute sans tri. Le parcours de l'index suffit.

Des améliorations ont été apportés à plusieurs commandes SQL d'administration. La création d'une table par clonage d'une autre table peut demander aussi le clonage des index. Il s'agit de la variante CREATE TABLE LIKE … INCLUDING INDEXES. La manipulation des séquences et des vues se faisait auparavant avec la commande ALTER TABLE (pour les renommer par exemple). Elles disposent maintenant de leur propre instruction, respectivement ALTER SEQUENCE et ALTER VIEW, bien que l'ancienne instruction fonctionne toujours.

Pour les développeurs

Les développeurs de fonctions PL/pgsql vont voir leur travail facilité grâce à trois améliorations importantes.

Dans les versions antérieures, le plan d'exécution d'une fonction était mis en cache. Par exemple, les identifiants système (appelé OID) des tables se trouvent en cache après la première exécution d'une fonction. À la deuxième exécution de la fonction, PostgreSQL utilisait cet OID directement pour gagner en temps d'exécution. La suppression d'une table dont l'OID se trouvait en cache n'invalidait pas le cache pour autant. Du coup, à la prochaine exécution de la fonction, une erreur était invariablement renvoyée indiquant que l'objet correspondant à l'OID était introuvable. Ce problème a été corrigé. Les données placées en cache pour cette fonction sont enfin invalidées si un OID compris dans le cache est supprimé. Elles seront aussi invalidées dans le cas où les statistiques sur un objet du cache sont modifiées.

La deuxième amélioration concerne les curseurs. Ils sont utilisables depuis bien longtemps avec PostgreSQL. Cependant, il était impossible de supprimer ou de modifier la ligne actuellement pointée par un curseur sans passer par la clé primaire. Ce manque a été comblé avec l'ajout des clauses WHERE CURRENT OF pour les instructions UPDATE et DELETE. Seule restriction, le curseur doit concerner une requête sans jointure ni groupement. Dans les instructions de déplacement, on peut noter l'ajout de l'instruction MOVE qui permet de déplacer le curseur sans récupérer les lignes. De plus, FETCH a été amélioré pour que l'utilisateur puisse indiquer la direction de la récupération. Les options de direction couvrent tous les cas possibles : NEXT, PRIOR, FIRST, LAST, ABSOLUTE, RELATIVE, FORWARD, BACKWARD.

PostgreSQL sait utiliser des fonctions qui renvoient un grand nombre de lignes. Ces fonctions, appelés SRF pour « Set Returning Functions », utilisent l'instruction RETURN NEXT généralement dans une boucle, ce qui peut se révéler être assez lent. Si les données sont déjà disponibles dans une table temporaire ou comme résultat d'une instruction SQL, l'instruction RETURN QUERY permet de renvoyer directement les lignes résultant de l'exécution de cette requête. C'est bien plus rapide mais c'est aussi plus compréhensible.

Dernier point, la déclaration d'une fonction permet maintenant de préciser au planificateur le nombre de lignes qui va être renvoyé et le coût d'exécution de la fonction. Pour cela, il existe les clauses ROWS et COST. Par défaut, le planificateur pense que la fonction renvoie 1000 lignes, ce qui est souvent loin de la vérité. Du coup, le planificateur construit un plan de requête à partir de mauvaises informations. En ajoutant la clause ROWS, le planificateur aura une idée plus précise du nombre de lignes renvoyées et pourra construire un plan de requête plus adéquat. De même pour la clause COST.

Pour les administrateurs

Les administrateurs ne sont pas en reste grâce notamment à une nouvelle méthode d'authentification : GSSAPI/SSPI. GSSAPI permet une authentification automatique, appelée aussi « Single Sign-On », pour les systèmes qui la supportent.

Les développeurs de PostgreSQL ont aussi ajouté une nouvelle méthode de journalisation des messages du moteur. Pour faciliter la récupération des informations dans une table, la journalisation applicative peut maintenant se faire dans un format CSV. Une fois le journal intégré dans une table, il est

possible de filtrer et de trier tous les messages grâce aux instructions SQL habituelles. Les applications sont nombreuses et il ne serait pas étonnant de voir de nouveaux outils arriver pour faciliter le traitement des journaux applicatifs avec une interface adéquate.

En dehors de cette nouvelle méthode de journalisation, les développeurs PostgreSQL ont incorporé de nombreuses traces supplémentaires :

- log_autovacuum_min_duration, pour tracer les actions effectuées par tous les processus autovacuum dont le temps d'exécution dépasse ce nombre de millisecondes ;
- log_lock_waits, pour tracer si une session attend un verrou depuis un certain nombre de millisecondes (ce nombre dépend du paramètre deadlock_timeout) ;
- log_checkpoint, pour tracer l'action des points de vérification (nombre de tampons écrits sur disque, temps passé à les écrire, etc.) ;
- log_temp_files, pour tracer la création de fichiers temporaires dont la taille dépasse ce nombre de Ko.

Pour une meilleure surveillance de l'état du cluster, une vue système a été ajoutée. pg_stat_bgwriter permet de suivre le fonctionnement du processus d'écriture en tâche de fond. Des colonnes ont été ajoutées à des vues systèmes déjà existantes (pg_stat_activity, pg_stat_database, pg_stat_all_tables, etc.) permettant un meilleur contrôle du système.

PostgreSQL dispose d'une configuration générale pour le cluster complet. Il était déjà possible d'avoir une configuration personnalisée pour une base de données ou un utilisateur. C'est maintenant possible pour une fonction. Cela permet d'éviter l'écriture du code qui consiste, à l'entrée dans la fonction, à enregistrer l'état du paramètre, à l'initialiser à la bonne valeur, puis, à la fin du script, à le configurer tel qu'il était initialement. Tout ce travail est maintenant réalisé par PostgreSQL.

Grâce au nouveau paramètre « autovacuum worker », l'autovacuum peut exécuter plusieurs VACUUM en même temps. De cette façon, l'exécution d'un VACUUM sur une très grosse table n'empêchera pas d'exécuter un VACUUM sur des tables plus petites mais qui en ont autant besoin.

Et les performances ?

Là-aussi, les développeurs PostgreSQL ont apporté des nouveautés significatives.

L'enregistrement asynchrone permet d'ajouter un délai après un COMMIT pour enregistrer les données modifiées sur les journaux de transaction. Cette fonctionnalité permet d'améliorer considérablement les performances pour les petites transactions au prix de la perte des dernières transactions en cas de crash du serveur. Cela étant dit, la sécurité des données et leur cohérence ne sont pas mise en jeu. Ce n'est donc pas équivalent à la désactivation du paramètre fsync. Pour éviter une trop grosse perte de transactions, le paramètre wal_writer_delay doit être finement configuré.

Toujours dans cette idée de diluer les écritures dans le temps, un nouveau paramètre permet cela pour les points de vérification (Checkpoint). Évidemment, les points de vérification manuels (donc issus de la commande CHECKPOINT) et ceux exécutés à l'arrêt du serveur sont exécutés le plus rapidement possible.

Concernant les journaux de transaction, un processus dédié, wal_writer, s'occupe de les écrire. Les écritures des journaux de transaction sont évitées dans certains cas spécifiques (tables temporaires, utilisation de CLUSTER et COPY) à condition que l'archivage des journaux soit désactivé.

La technologie HOT accélère la réutilisation de l'espace libre pour la majorité des UPDATE et DELETE. Auparavant, seul VACUUM pouvait récupérer l'espace pris par des lignes mortes. Avec HOT, cet espace peut aussi être récupéré lors d'un INSERT ou d'un DELETE si aucune modification ne se fait sur les colonnes indexées. Cela va en plus permettre d'éviter l'ajout d'entrées dupliquées dans les index.

Des parcours séquentiels exécutés en parallèle sont synchronisés. Autrement dit, si le parcours d'index de la table t en est à 20% de la table et qu'un deuxième parcours séquentiel est lancé en parallèle pour la même table, ce deuxième parcours va commencer son parcours séquentiel à l'endroit où se trouve le premier processus. Ils partagent ainsi la lecture de la table pour une bonne partie. Si la table t fait 5 Go, chaque parcours séquentiel aurait lu les 5 Go pour les versions antérieures (soit 10 Go lu). En version 8.3, si le deuxième processus débute alors que le second en est à 20%, les deux processus liront 6 Go (5 Go + 20% de 5 Go). Du coup, plus le lancement des processus est rapproché, plus le gain est important. De même, plus vous avez de parcours séquentiel en parallèle sur la même table, plus vous gagnez en performance. Cette fonctionnalité affecte évidemment l'ordre des lignes renvoyées s'il n'y a pas de clause ORDER BY. Cependant, PostgreSQL a toujours indiqué que l'ordre des lignes renvoyées n'était pas garantie quand cette clause n'était pas précisée. Le gros intérêt de cette fonctionnalité, en dehors de la diminution des lectures/écritures, est d'éviter au maximum de remplir totalement le cache shared_buffers avec la table en cours de lecture.

Dans PostgreSQL, toute transaction se voit affecter un identifiant de transaction. Comme le compteur de transaction est fini, un administrateur doit réaliser des opérations de maintenance pour s'assurer que les identifiants de transaction ne sont jamais en conflit. Une modification a permis de n'ajouter un identifiant de transaction que pour les instructions de modification de schémas (DDL, pour « Data Definition Language ») et pour celles de modification des données (DML, pour « Data Manipulation Language ») qui sont les seules à avoir un besoin réel de l'identifiant de transaction. Tout le monde profite de cette fonctionnalité car l'utilisation des identifiants va décroître, permettant moins d'écriture dans le répertoire pg_clog. Mais évidemment, cela intéresse en premier lieu les administrateurs de bases de données qui sont principalement en lecture seule.

Enfin, en dernier point, il est à noter que le stockage des méta-informations des fichiers de données prend moins de place, entre 2 et 5% de différence. Cela peut paraître anecdoctique mais pour une base de données de 5 Go, l'administrateur gagne entre 100 et 250 Mo. Pour la très grosse base de Météo France (3,5 To), le gain est bien plus significatif : entre 71 et 180 Go.

Incompatibilité notables

Commençons par celle dont l'impact sera vraiment important. Les types de données autres que les types caractères ne sont plus automatiquement convertis en texte. Il va donc être nécessaire de vérifier toutes les conversions implicites vers le type text pour s'assurer que ces conversion sont toujours fonctionnelles. Dans le cas contraire, il est nécessaire d'ajouter une conversion explicite grâce à l'opérateur :: ou à la fonction CAST.

Beaucoup de paramètres ont changé de noms, quelques-uns ont été supprimés ou fusionnés. Il n'est donc pas possible de remplacer le fichier de configuration par l'ancien.

Autre grosse différence par rapport à la version précédente, mais cette différence se situe au niveau du comportement. Mettre un paramètre en commentaire dans le fichier de configuration postgresql.conf fait que ce paramètre retrouve sa valeur par défaut.

Et au niveau des applications ?

Il y a eu peu de modifications sur les outils PostgreSQL. Néanmoins, notons l'option –roles-only de pg_dumpall pour ne sauvegarder que les rôles et –

tablespaces-only pour ne sauvegarder que les tablespaces.

Quant à initdb, il est enfin possible de lui demander de créer les journaux de transaction ailleurs que dans le répertoire des données de PostgreSQL. Avec l'option -X (suivie du nouveau répertoire de stockage), un lien symbolique pg_xlog sera créé pour pointer vers le répertoire indiqué.

Enfin, autre changement de comportement, les outils createdb, createuser, dropdb et dropuser n'affichent plus le message habituel (par exemple « CREATE DATABASE » pour l'outil createdb). Sauf affichage d'un message d'erreur, il faut considérer que l'opération s'est déroulée correctement.

Modules contrib

Au niveau des modules contrib, deux nouveaux sont particulièrement intéressants.

pageinspect est un outil permettant d'inspecter les pages d'une table ou d'un index. Principalement utilisé par les développeurs PostgreSQL, ce module pourrait être très intéressant en cas de problème de corruption des fichiers d'une base.

pg_standby est un outil à utiliser sur le serveur esclave dans une configuration Warm Server Standby (aussi appelé Log Shipping). Cette configuration, disponible depuis la version 8.2, permet de restaurer un serveur secondaire en continue avec chaque journal de transaction terminé au moment où ce journal est archivé. Cependant, pour cela, il fallait écrire un script ou un programme qui se charge de l'attente et de la copie du journal. En 8.3, ce n'est plus nécessaire car ce programme est fourni en module contrib. Il est possible d'utiliser directement pg_standby ou de modifier son source pour le personnaliser (une section de son code est d'ailleurs clairement indiquée comme personnalisable).

Conclusion

Cette version, comme les précédentes, ajoute un nombre conséquent de nouvelles fonctionnalités sans perdre en performance. Bien au contraire, un travail considérable est réalisé pour augmenter la rapidité et l'interactivité des bases de données. Il faut donc s'attendre à une version majeure très intéressante. Tout adimnistrateur devrait tester la dernière Release Candidate pour s'assurer que tout fonctionne correctement et pour se familiariser avec les nouveautés.