GDSQL : un langage déclaratif pour l'auto-administration de bases de données réparties[1]

Haitang FENG Nicolas LUMINEAU Mohand-Saïd HACID

Université de Lyon, CNRS Université Lyon 1, LIRIS, UMR5205, F-69622, France

{haitang.feng, nicolas.lumineau, mohand-said.hacid}@liris.cnrs.fr

Résumé

GDSQL est une extension du langage déclaratif SQL apportant un niveau d'abstraction dans le langage de définition des données (LDD) utile pour l'optimisation de requêtes. Ce langage et son moteur de requêtes offrent aux administrateurs de bases de données réparties une solution transparente pour gérer de manière optimale les structures de données et leur allocation à travers le réseau. Ce langage simple à l'emploi répond aux besoins de simplification du processus d'administration des données réparties en facilitant leur auto-organisation tout en tenant compte des contraintes exprimées par l'administrateur. Cette démonstration dont l'objectif est de montrer la faisabilité de notre approche, s'accompagne d'un scénario montrant l'ensemble du processus de création des données avec ou sans contraintes, leur interrogation et leur évolution en termes de stockage et de placement.

Mots-clés : Optimisation ; Langage déclaratif ; Auto-organisation

I- Introduction

La génération des résultats synthétiques à partir de grands volumes de données passe généralement par l'utilisation d'algorithmes complexes, couteux en temps et en ressources. Les performances de telles applications sont généralement dégradées du fait des nombreux accès aux données et d'une stratégie d'optimisation des requêtes qui ne tient pas forcément compte de l'évolution des besoins des utilisateurs.

Parmi les nombreuses solutions envisagées pour garantir une qualité de service constante en évitant tout goulet d'étranglement, la parallélisation des accès données bien connue dans les bases de données parallèles[1], s'avère efficace mais peut rapidement être complexe à administrer. Face à l'évolution des architectures réseaux d'autres approches sont apparues dans les architectures distribuées telles que la grille[4], le P2P[7] et le *cloud computing*[12]. Dans de telles approches, l'administration des données devient un véritable challenge. Afin de garantir un traitement optimisé des requêtes portant sur des données réparties, les solutions proposées passent par un ajustement du type des objets (par exemple par matérialisation des vues)[2][3][9][11][13] représentant les données, par des stratégies de

[1] Ce travail est effectué dans le cadre d'une convention CIFRE de l'ANRT

placement des données et de leur répliques[6][10]. Avec l'évolution des besoins des utilisateurs, un maintien permanent des données réparties est lourd, complexe et quelque fois peu robuste[5][8]. Une administration optimisée consiste donc à garantir non seulement l'exécution optimale des requêtes mais aussi l'adéquation entre les besoins des utilisateurs évoluant en permanence et les données gérées par l'administrateur. Pour ce faire, il est alors indispensable de disposer d'outils efficaces et transparents pour l'administration des données.

Dans ce contexte, nous proposons l'extension du langage déclaratif SQL : '*Generic Datasource Structured Query Language*' (GDSQL). Ce langage simple à l'emploi répond aux besoins de simplification du processus d'administration des données réparties en permettant d'exprimer des contraintes (ou absence de contraintes) sur les structures de données et leur placement à travers le réseau. Dans la suite de cet article, la Section 2 introduit l'architecture adoptée et présente brièvement le niveau d'abstraction choisi pour les données. Dans la Section 3, langage est présenté à travers un scénario concret. Et une brève description de la démonstration est proposée dans la Section 4.

II- Architecture et les '*Generic Datasources*'

Dans le contexte de nos travaux, nous considérons des données homogènes et réparties sur un cluster où chaque nœud dispose d'un Système de Gestion de Base de Données. L'abstraction des données nécessaire au fonctionnement du système repose sur la notion de *Generic Datasource*, que nous appellerons *Datasource* dans la suite. Une *DataSource* est un objet générique ayant un schéma qui définit sa structure logique. Une instance de *DataSource* est un objet SQL tel qu'une table, une réplique de table, une vue virtuelle ou une vue matérialisée. Cet objet permet à l'administrateur des données de ne plus distinguer explicitement les objets SQL manipulés. Une *DataSource* peut être créée à partir d'un schéma de *DataSources* ou à partir du résultat d'une requête portant sur d'autres *DataSources*.

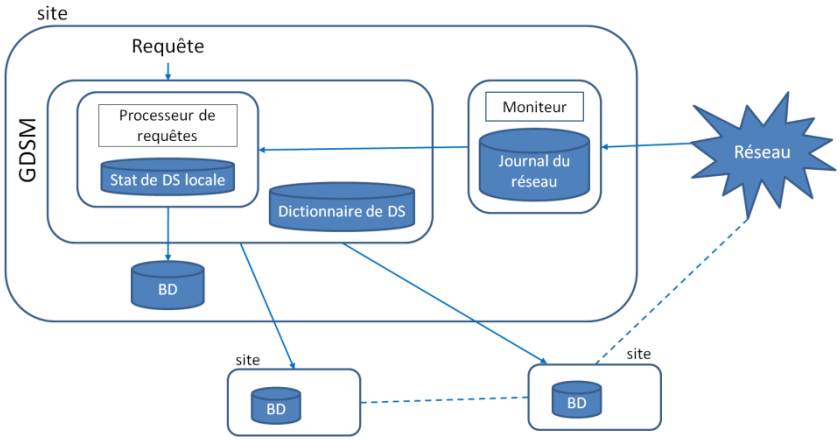

Figure 1 : Architecture globale

L'architecture que nous proposons pour pouvoir gérer les *Generic Datasources* est illustrée par la figure 1.

Les données sont stockées et réparties sur les différents SGBD des différents sites. Chaque SGBD dispose d'une surcouche logicielle que nous avons appelés *GDSM* (*Generic DataSource Manager*). Le GDSM prend en entrée les requêtes qu'il analyse et traite en fonction des informations fournies par le moniteur. Le moniteur a pour fonction de surveiller et de sauvegarder les informations concernant l'évolution du réseau et en particuliers des données. La requête est ensuite transférée au SGBD pertinent pour être ensuite exécutée et le résultat sera retourné à l'application appelante. Une fois ce processus terminé, les statistiques dans le *Access Manager* seront mises à jour. En fonction de ces statistiques collectées et de leur évolution, le système lancera les audits nécessaires pour statuer sur l'efficacité des structures données fortement sollicitées. Ces audits aboutiront selon le cas à matérialiser des vues virtuelles fortement sollicitées, à dématérialiser des vues faiblement sollicitées, à répliquer ou migrer des tables en fonction de leur usage.

III- Scénario

L'objectif de ce scénario est de montrer les différentes étapes de gestion des *DataSources* : création, interrogation et gestion.

a) Création des *DataSources*

Nous considérons qu'un groupe de chercheurs souhaite mettre en place une application pour interroger le classement mondial des établissements académiques en fonction de leurs publications. L'administrateur des données a la possibilité de créer des *DataSources* selon le modèle prédéfinie ou à partir de requêtes considérées comme coûteuses et fréquentes. Dans ce scénario, on considère que les données seront réparties sur trois sites différents : Lyon, Paris et Shanghai. Dans tous le cas, il est nécessaire de déclarer les paramètres de connexion à ces différentes bases :

```
INSERT INTO SITES VALUES ('Lyon', 'b710ntb.univ-lyon1.fr', '1521', 'ora10g');
INSERT INTO SITES VALUES ('Paris', 'b001.univ-paris6.fr', '1521', 'ora10g');
INSERT INTO SITES VALUES ('Shanghai', 'cs21.sjtu.edu.cn', '1521', 'ora10g');
```

i. Création de *DataSources* à partir du modèle de données

L'administrateur souhaite créer des *DataSources* concernant les Universités, les régions et les pays stockés respectivement sur les sites de Lyon, Paris et Shanghai.

```
CREATE DATASOURCE Universite (idu number(5), nomu varchar(5), idreg number(5)) ON 'Lyon';
CREATE DATASOURCE Region (idr number(5), nomr varchar(5), idp number(5)) ON 'Paris';
CREATE DATASOURCE Pays (idp number(5), nomp varchar(5)) ON 'Shanghai';
```

L'administrateur souhaite ensuite créer la *DataSource* 'Auteur' sans contrainte de placement de l'administrateur mais avec apprentissage par le système GDSM.

```
CREATE DATASOURCE Auteur (ida number(5), noma varchar(5), iduniv number(5)) WITH USAGE
LEARNING;
```

Dans ce scénario, on suppose que la *DataSource* 'Auteur' est placée sur le site 'Paris'.

ii. Création de *DataSources* à partir de requêtes

La *DataSource* 'ArticlePublie' est créée à partir de l'union de deux *DataSources* 'ArticleRevue' et 'ArticleConf' se trouvant respectivement sur les sites de 'Paris' et 'Shanghai'. 'ArticlePublie ' est sans contrainte de placement par l'administrateur et avec apprentissage par le système GSDM. Ce qui est le cas également pour la *DataSource* 'AuteurArticlePublie' qui permet de faire l'association entre les auteurs et les articles qu'ils ont (co-)rédigés.

```
CREATE DATASOURCE ArticlePublie (idar, titrear, anneear) AS ((SELECT idar, titrear, anar FROM
ArticleRevue) union (SELECT idarc, titrearc, annee FROM ArticleConf)) WITH USAGE LEARNING;

CREATE DATASOURCE AuteurArticlePublie (idart, idaut) WITH USAGE LEARNING;
```

Dans ce scénario, on suppose que les requêtes soumises au système portent principalement sur le nombre moyen d'articles publiés par Région en 2010. L'administrateur, prévenu par le moniteur, crée la *DataSource* 'PubliantRegion' suivante :

```
CREATE DATASOURCE PubliantRegion (idr, nomr, nbmoyart) AS (SELECT idr, nomr, avg FROM Region
r, Universite u, Auteur a, ArticlePublie ap, AuteurArticlePublie aap WHERE u.idreg=r.idr AND
a.iduniv=u.idu AND aap.idaut=a.ida AND aap.idart=ap.idar AND ap.anneear=2010 GROUP BY idr,
nomr) WITH USAGE LEARNING;
```

iii. Bilan

Concrètement les *DataSources* créées ont généré des tables pour Université, Région, Pays, Auteur, AuteurArticlePublie, ArticleRevue et ArticleConf; des vues virtuelles pour les *DataSources* : PubliantRegion, ArticlePublie; et aucune vue matérialisée (pour l'instant).

b) Interrogation et gestion des *DataSources*

On suppose que de nombreux utilisateurs ont interrogé les informations des *DataSources*. Les nombreuses requêtes qui ont été traitées par le système GDSM ont engendré une forte sollicitation de 'PubliantRegion' et 'Auteur'. Une phase d'audit est lancée afin :

- de calculer le bénéfice et le coût de la matérialisation de la vue 'PubliantRegion',
- de calculer le bénéfice et le coût de la réplication de la table 'Auteur'.

Dans ce scénario, nous supposons que le système a pris la décision de matérialiser la vue 'PubliantRegion' en exécutant la commande :

```
TRANSFORM DATASOURCE PubliantRegion TO MATERIALIZED VIEW;
```

Nous supposons également que le système a pris la décision de répliquer la *DataSource* sur 'Shanghai' et 'Lyon'.

Au bout d'un certain temps, l'intérêt pour la vue a évolué et l'absence de sollicitations de la vue 'PubliantRegion', déclenche un audit par le système pour calculer le bénéfice et le coût de la dématérialisation de la vue 'PubliantRegion', en exécutant la transformation.

```
TRANSFORM DATASOURCE PubliantRegion TO VIEW;
```

Finalement, l'administrateur souhaite s'assurer que la *DataSource* 'ArticleRevue' soit sur le site 'Shanghai' (ou dans le cas de réplication, que la copie primaire 'ArticleRevue' soit sur le

site de ‘Shanghai’). Il va changer la localisation de cette *DataSource* en exécutant la commande de migration.

```
MOVE DATASOURCE ArticleRevue TO ’Shanghai’;
```

IV- Démonstration

La démonstration proposée consiste à présenter le scenario précédent *via* le prototype ‘AutoGDS’ pour montrer la faisabilité de l’approche (prototype a été développé en Java sous Eclipse SDK 3.5.2 avec l’extension JSqlParser et sous Oracle) et une simulation sera présentée pour valider les algorithmes liés à l’auto-administration des *DataSources via* l’application ‘Salsa’ (Simulateur de réseau à large échelle développé en Java).

Références

[1] DeWitt D., Gray J., 1992. *Parallel database systems: the future of high performance database systems*. Communications of the ACM, 35(6):85-98.

[2] Harinarayan V., Rajaraman A., Ullman J.D., 1996. *Implementing Data Cubes Efficiently*. In SIGMOD.

[3] Gupta A., Mumick I.S., 1999. *Maintenance of materialized views: problems, techniques, and applications*. MIT Press, Cambridge, MA, 145-157.

[4] Chervenak A., Foster I., Kesselman C., *et al*., 2000. *The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets*, Journal of Network and Computer Applications, 23(3):187-200.

[5] Afrati F.N., Li C., Ullman J.D., 2001: *Generating efficient plans for queries using views*. In SIGMOD.

[6] Ranganathan K., Foster, I.T., 2001. *Identifying Dynamic Replication Strategies for a High-Performance Data Grid*. LNCS 2242:75-86.

[7] Stoica I., Morris R., Karger D., *et al.*, 2001. *Chord: A scalable peer-to-peer lookup service for internet applications*. SIGCOMM Computer Communication Review, 31(4):149-160.

[8] Galanis L., Wang Y., Jeffery S., *et al.*, 2003. *Locating data sources in large distributed systems*. In VLDB.

[9] Arion A., Benzwken V., Manolescu I. *et al.*, 2007. *Structured materialized views for XML queries*. In VLDB: 87-98.

[10] Rasool Q., Li J., Oreku G.S. *et al.*, 2007. *A Comparative Study of Replica Placement Strategies in Data Grids*. LNCS 4537:135-143.

[11] Agrawal P., Silberstein A., Cooper B.F., 2009. *Asynchronous View Maintenance for VLSD Databases*. In SIGMOD.

[12] Armbrust M., Fox A., Griffith R. et al., 2009. *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, Feb. 10.

[13] Manolescu I., Zoupanos P., 2009. *XML materialized views in P2P networks*. Workshop DataX in conjunction with EDBT 2009 and ICDT 2009.