D-WorM : Middleware pour le traitement des requêtes massives dans les entrepôts de données massives

Rado Ratsimbazafy*, Omar Boussaid*, Fadila Bentayeb*

*Laboratoire ERIC Université Lumière Lyon 2,
5 Avenue Pierre Mendes France, 69676 Bron Cedex
prenom.nom@univ-lyon2.fr, http://www.eric.univ-lyon2.fr

Résumé. Avec l'accroissement massif des données, plusieurs problèmes connus des entrepôts de données sont à repenser à l'échelle du *Big Data* et des environnements distribués. Dans cette article nous proposons D-WorM (Data warehouse Workload Manager), une solution pour résoudre la problématique posée par le traitement des requêtes massives. Notre Middleware a pour objectif d'améliorer la gestion des montées en charge.

1 Introduction

Les entrepôts de données sont au coeur des systèmes d'informations décisionnels. Ils sont constamment interrogés par plusieurs utilisateurs avec des requêtes décisionnelles à des fins d'analyse pour la prise de décision (Inmon, 1992). Il n'est pas rare qu'ils subissent des montées en charge (requêtes massives ou *workloads* en anglais). Avec l'avènement du *Big Data* la taille des entrepôts de données devient de plus en plus grande. Le développement de nouvelles technologies type *cloud computing* nous amène à repenser les mécanismes de stockage et de traitement des très grands entrepôts de données.

Le traitement des requêtes massives n'est pas un problème récent. En effet, de nombreux chercheurs s'y sont déjà intéressés, que ce soit pour les bases de données (Sellis, 1988) ou pour les entrepôts de données (Gacem et Boukhalfa, 2013).

Des travaux de recherche récents ont été menés pour résoudre le problème, à l'échelle du *Big Data*. Des solutions ont alors été proposées sous forme de boîtes noires qui regroupent tout un écosystème qui ne laisse pas beaucoup de liberté à l'utilisateur pour personnaliser l'infrastructure selon ses besoins (Bajda-Pawlikowski et al., 2011; Giannikis et al., 2012).

Le prototype D-WorM que nous proposons dans cet article, a pour objectif de fournir une couche applicative capable de traiter les montées en charge pour réduire le temps de réponse moyen en mutualisant certaines tâches, en partageant certaines données et en utilisant des vues. L'objet de ce papier est de présenter le fonctionnement de notre solution à partir de plusieurs charges de requêtes de tailles différentes.

Cet article est organisé comme suit. Le principe de fonctionnement de D-WorM est expliqué dans la section 2. Les choix technologiques pour l'implémentation et le scénario de démonstration sont détaillés dans la section 3. Dans la section 4 nous concluons et discutons des perspectives de recherche à venir.

2 Principe de fonctionnement

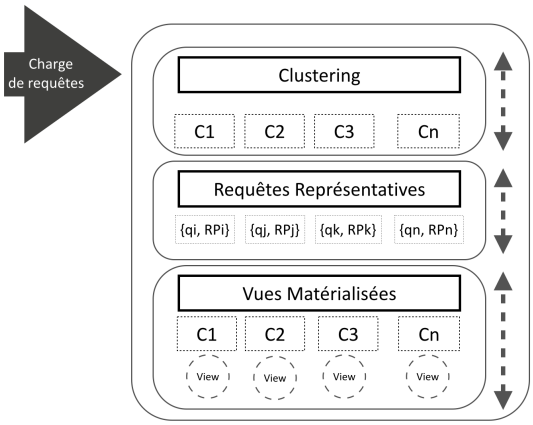

FIG. 1 – *Phase 1 : Traitement d'une charge de requêtes*

Le principe de fonctionnement de D-WorM est divisé en deux phases.

La première phase Comme montré dans la Fig. 1, cette phase consiste à traiter une charge de requêtes en entier. Cette phase est subdivisée en 3 étapes. (1) La classification des requêtes : consiste à transformer la charge de requêtes en matrice binaire de présence/absence où les lignes sont les requêtes et les colonnes sont les attributs si l'attribut est présent dans la requête on met 1 sinon 0. Avec l'aide d'un algorithme de fouille de données, les K-means, les requêtes de la charge sont regroupées en clusters. (2) La construction de la requête représentative : est obtenue à partir de chaque classe de requêtes pour répondre aux requêtes de celle-ci. Une seule requête représentative suffit-elle ou en faut-il plusieurs par classe ? (3) La création des vues : avec chaque requête représentative précédemment construite, nous disposons de requêtes qui sont utilisées pour la construction des vues regroupant les données les plus fréquemment demandées dans l'ensemble des requêtes de la charge.

La deuxième phase Cette phase correspond à l'exécution de chaque requête de la charge, comme montré dans la Fig. 2. Pour cela, on prend une classe de requêtes et on utilise la vue construite avec la requête représentative pour y répondre. Dans le cas d'une nouvelle requête, D-WorM vérifie si une demande similaire est en cours d'exécution, afin de réutiliser une partie ou la totalité des données générées par des requêtes exécutées précedemment . Si aucune requête similaire n'est en cours d'exécution, il accédera à l'entrepôt de données et créera une vue temporaire qui pourrait servir à d'autres requêtes. Des statistiques sont alors calculées pour ne garder que les vues les plus utilisées et faire une mise à jour ponctuelle des ces vues.

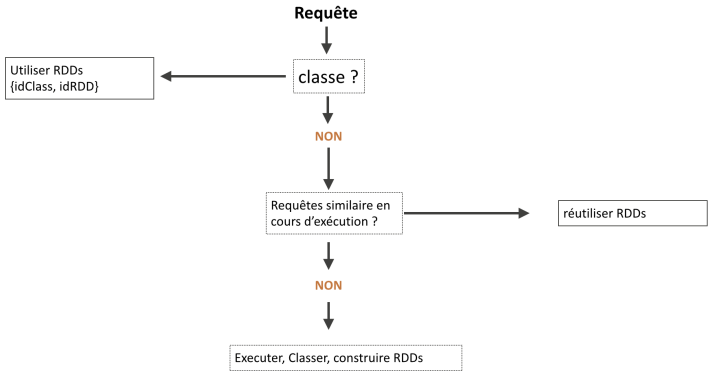

FIG. 2 – *Phase 2 : Traitement d'une requête de la charge*

3 Implémentation et scénario de démonstration

Dans cette section nous détaillons l'implémentation de D-WorM qui est développé en tant qu'intergiciel utilisé pour être le serveur de communication entre les utilisateurs et l'entrepôt de données (distribué on non).

L'environnement L'implémentation de D-Worm peut se faire dans un environnement distribué ou non distribué. Dans un environnement distribué D-Worm est implémentée sur Hadoop [1] avec l'utilisation de Spark [2] pour l'interrogation de l'entrepôt de données et le stockage des vues matérialisées est réalisé sous forme de Resilient Distributed Datasets (RDD). D-WorM est intégré au sein du SGBD *Oracle 11g*, dans un environnement non distribué, comme étant l'interface d'écoute qui reçoit les requêtes et qui renvoit les résultats. D-WorM est développé en Python 2.7 pour faciliter le déploiement multi-plateformes et le passage à l'échelle dans un environnement distribué. L'utilisation de Python nous a aussi permis d'utiliser plusieurs librairies ouvertes (SQLite 3, Scipy 0.14) pour accélérer le développement et le déploiement de notre solution. Le choix de SQLite [3] pour la création des vues est motivé par le fait que l'intégralité de la base de données est intégrée dans un seul fichier sans le schéma habituel client-serveur, ce qui permet à D-WorM d'interroger la vue en mémoire sans accéder au serveur de l'entrepôt de données.

Scénario de démonstration Dans un environnement non distribué, nous avons créé sous Oracle, à partir du benchmark TPC-DS [4], un entrepôt de données en constellation proche de la réalité, avec lequel nous avons généré un Téraoctet de données artificielles. Nous avons démultiplié des requêtes décisionnelles, d'analyse de vente, pour évaluer notre approche.

Les premiers tests ont été réalisés sur un serveur dans un environnement non distribué avec Intel Xeon e3 (3.1Ghz x 4) à 16 GB de RAM utilisant Linux CentOS 6.5. Nos premières expérimentations sont très encourageantes car avec deux charges de 500 et 1000 requêtes nous

1. `http://hadoop.apache.org`
2. `https://spark.apache.org`
3. `http://www.sqlite.org`
4. `http://www.tpc.org/tpcds/`

avons pu obtenir un temps de réponse moyen 1.5 fois plus rapide que l'exécution normal sous Oracle en incluant certaines interventions manuelles.

Pour ces premières expérimentations, la classification des requêtes à été faite avec k=5 car nous avons démultiplié cinq requêtes en modifiant aléatoirement les paramètres. Même si les résultats sont encourageants, nous pensons utiliser d'autres méthodes de clustering mieux adaptées à notre problématique et à la nature des données à classer telle que la méthode présentée par Li (2005).

4 Conclusion

D-WorM est encore en cours de développement, plusieurs fonctionnalités sont encore à améliorer, comme le traitement des requêtes sur un intervalle de temps ; l'algorithme de clustering qui reste encore à optimiser. Par ailleurs une application cliente et une interface de programmation sont prévues pour fonctionner de paire avec notre solution. Nous travaillons actuellement sur l'évaluation de la qualité de notre approche par rapport à l'existant, Certaines solutions disponibles sur le marché ne pourront pas être évaluées faute de licence. Nous continuons à creuser d'autres pistes d'optimisation que nous pourrions intégrer à notre solution.

Références

Bajda-Pawlikowski, K., D. J. Abadi, A. Silberschatz, et E. Paulson (2011). Efficient processing of data warehousing queries in a split execution environment. In *Proc. of the SIGMOD Conference*, pp. 1165–1176.

Gacem, A. et K. Boukhalfa (2013). Very large workloads based approach to efficiently partition data warehouses. In *Modeling Approaches and Algorithms for Advanced Computer Applications*, pp. 285–294.

Giannikis, G., G. Alonso, et D. Kossmann (2012). SharedDB : Killing one thousand queries with one stone. *PVLDB 5*(6), 526–537.

Inmon, W. H. (1992). *Building the Data Warehouse.* John Wiley and Sons.

Li, T. (2005). A general model for clustering binary data. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 188–197.

Sellis, T. K. (1988). Multiple-query optimization. *ACM Trans. Database Syst. 13*(1), 23–52.

Summary

With the massive growth of data, several known issues of data warehouses must be scaled up for Big Data and cloud computing. In this paper we propose D-WorM (Data warehouse Workload Manager) a software to solve one of those problems which is the massive query handling. To this end, we propose a Middleware which aims to improve the workload management especially in data warehouses environments.