Optimisation Conjointe et Dynamiques des Requêtes Régulières et Recommandées

Mustapha Chaba Mouna *, Ladjel Bellatreche **
Narhimane Boustia *

*LRDSI Laboratory, Data Processing Department University Blida 1, Algeria
(mustapha.medea , nboustia)@gmail.com,

**LIAS/ISAE-ENSMA - Université de Poitiers,France
bellatreche@ensma.fr

Résumé. De nos jours, les applications de base de données avancées gèrent deux types principaux de requêtes: les requêtes régulières émises par les utilisateurs finaux et les requêtes émises automatiquement par des systèmes tels que les systèmes de recommandation. L'objectif principal des requêtes recommandées est de suggérer les nouveaux éléments intéressants aux utilisateurs. Du point de vue du SGBD, les requêtes recommandées représentent un surcoût important, car il doit les optimiser comme il le fait pour les requêtes régulières. Généralement, les requêtes régulières et recommandées sont actives en même temps et partagent des opérations. En conséquence, il est important de les optimiser conjointement. Motivés par cette observation, nous proposons dans cet article, une nouvelle approche qui suppose l'existence d'une charge de requêtes régulières connues à l'avance et une autre charge de requêtes recommandées arrivée dynamiquement. Les requêtes régulières sont optimisées à l'aide des vues matérialisées. Ces dernières sont alors utilisées pour optimiser les requêtes recommandées en fonction de leur similarité avec les requêtes régulières. Afin d'assurer le passage à l'échelle de notre approche et la captation de l'interaction entre les différentes requêtes, nous utilisons la structure des hypergraphes construit dynamiquement en fonction de l'arrivée des requêtes. Notre approche est validée théoriquement et réellement sur un SGBD commercial en utilisant les données du banc d'essai Star Schema Benchmark.

1 Introduction

Nous vivons dans l'ère de déluge de données, généré en permanence par plusieurs fournisseurs tels que les réseaux sociaux, les capteurs, les expérimentations scientifiques, etc. Ces données sont généralement capturées, nettoyées, stockées et analysées par les décideurs, les gestionnaires et les chercheurs. L'analyse des données est réalisée au moyen de requêtes intensives et complexes impliquant plusieurs entités stockant ces données à l'aide des opérations de jointures et d'agrégations. Optimiser ces requêtes devient un besoin crucial des chercheurs

pour satisfaire les demandes de rapidité d'exécution de requêtes des utilisateurs. Un grand nombre de recherches sur l'optimisation des requêtes existe couvrant des initiatives logicielles et matérielles Bouchakri (2015); Roukh et al. (2017). Les solutions logicielles couvrent principalement deux volets : (1) les optimisations logiques couvrant les aspects algorithmiques pour l'implémentation des opérations algébriques relationnelles telles que les jointures (boucle imbriquée, hachage, fusion de tri) et la programmation (utilisation de nouveaux paradigmes de programmation tels que Mapreduce, Spark, etc.). (2) Les optimisations physiques consistant à utiliser des méthodes d'accès telles que les index et les vues matérialisées. Les solutions matérielles concernent l'utilisation intensive de plateformes de déploiement avancées telles que les machines parallèles (clusters de données homogènes et hétérogènes, shared-nothing), le stockage de données sur Cloud, etc. et les vastes ressources offertes par les GPU, FPGA et ASIC Breß et al. (2016).

Les solutions matérielles et logicielles sont généralement combinées pour satisfaire les exigences des décideurs en termes de performances de requête. De nos jours, les systèmes de stockage de données ne traitent pas seulement des requêtes régulières émises par les utilisateurs finaux, mais aussi des requêtes émises par des systèmes de recommandation (RecSys) qui suggèrent aux utilisateurs d'autres éléments au moyen de nouvelles requêtes.

La recommandation représente une valeur ajoutée pour les entreprises. Amazon fait environ 30% des ventes grâce aux recommandations Aufaure et Zimányi (2011). Ces approches sont très pertinentes dans un contexte de base de données. Le domaine de base de données est à la fois producteur et consommateur de recommandation. Les éditeurs commerciaux comme Oracle, SQL Server, DB2 proposent des outils d'administration (Oracle SQL Tuning et DB2 Advisors) qui assistent leurs administrateurs de bases de données au cours de leurs tâches. Ces conseillers proposent des techniques d'optimisation (index, partitionnement de données, vues matérialisées) pour une charge donnée. La base de données SAP HANA propose un conseiller de stockage pour les bases de données hybrid-stores qui recommande le stockage optimal en fonction des caractéristiques des données, des requêtes et des modèles de coûts mathématiques évaluant la qualité du stockage proposé. Les efforts les plus importants sur la recommandation de requête sont effectués en dehors du SGBD et généralement basés sur les journaux de requêtes. Par exemple, le système de recommandation de QueRIE Eirinaki et al. (2014) a pour but de déterminer les requêtes dans le journal qui correspondent à la session de l'utilisateur actuel. Ainsi, la similarité de la requête est calculée en comparant les parties de la base de données adressées par les requêtes. Une large panoplie d'études sur les recommandations de requêtes OLAP a été proposée dans le contexte des entrepôts de données Aligon et al. (2015); Giacometti et al. (2009); Jerbi et al. (2009); Negre (2017). Il est profitable pour les entreprises de recommander des articles pour leur commerce, mais du point de vue de base de données et son SGBD, ces requêtes de recommandation doivent être optimisées comme les requêtes régulières dans le but d'assurer leur efficacité. Un point important qui doit être souligné est la similarité entre les requêtes régulières et recommandées. Cette similarité est matérialisée par la présence d'opérations partagées.

Dans le contexte des entrepôts de données relationnels, le partage de données est également élevé en raison de la structure des schémas logiques de ces applications (schéma en étoile). Le partage entre les requêtes connue sous le nom d'interaction de requêtes, qui a donné naissance à un nouveau problème appelé *multi-query optimization* (MQO). Le MQO qui vise à optimiser l'exécution globale d'une charge de requêtes en augmentant la réutilisation de résultats inter-

médiaires de requêtes T.K.Sellis (1988). Cette interaction de requête a largement contribué à l'optimisation des requêtes et la sélection des structures d'optimisation comme les vues matérialisées Goasdoué et al. (2011); Yang et al. (1997), le partitionnement Bellatreche et Kerkad (2015), la gestion du tampon A.Kerkad et al. (2012) et l'ordonnancement de requêtes Boukorca et al. (2014).

Une autre conséquence de la recommandation de requête est l'augmentation des requêtes que SGBD doit optimiser Eirinaki et al. (2014). Pour faire face au volume de requêtes, le développement de structures de données scalables qui capturent le partage des opérations de requête est largement recommandé Boukorca et al. (2015). Les hypergraphes Bretto (2013) ont été proposés et ils montrent l'efficacité et la rentabilité dans la gestion du volume de requête et l'interaction entre eux Boukorca et al. (2015); Curino et al. (2010). Ce travail a été proposé pour améliorer l'optimisation statique des requêtes régulières. Pour combler cette lacune, nous proposons d'exploiter les hypergraphes pour traiter les requêtes régulières et recommandées dans un environnement dynamique. Les requêtes de recommandations sont optimisées en se basant sur l'historique des optimisations dédiées aux requêtes régulières et de leurs similitudes. Ces optimisations sont effectuées au moyen de vues matérialisées. L'aspect dynamique de notre environnement nous oblige à construire incrémentalement notre hypergraphe. Pour ce faire, nous définissons une algèbre sur l'hypergraphe assurant cette construction. Au meilleur de nos connaissances, ce travail est le seul traitant de l'optimisation simultanée un volume important de requêtes régulières et de recommandations.

Le reste de cet article est organisé comme suit : La section 2 présente un état de l'art sur les travaux d'optimisation de requêtes régulières et recommandées. La section 3 décrit la démarche globale de notre proposition. La section 4 est dédiée à l'optimisation de requêtes recommandées. La section 5 valide nos propositions. La section 6 conclut le document.

2 Travaux connexes

Dans cette section, nous présentons les travaux les plus importants sur la recommandation de requête et l'optimisation multi-requêtes. RecSys Adomavicius et Tuzhilin (2005) est maintenant bien établi en tant que technologie de filtrage d'information, utilisée dans un large éventail de domaines (E-commerce, recherche Web et réseaux sociaux, ...). Ils sont traditionnellement classés en fonction du contenu (les suggestions sont basées sur les actions passées des utilisateurs uniquement) ou collaboratives (les suggestions sont basées sur des similitudes entre les utilisateurs) ou une combinaison hybride de celles-ci. Comme indiqué dans Eirinaki et al. (2014), la recommandation des requêtes de base de données diffère des approches classiques du système de recommandation, au moins dans deux aspects principaux : (1) les requêtes sont exprimées avec des langages de requête déclaratifs ; et (2) les requêtes recommandées devraient être intuitives afin que les utilisateurs puissent les comprendre et les affiner si nécessaire. Les études les plus importantes sur les recommandations de requêtes concernent plusieurs aspects : (a) le choix des journaux de requêtes, (b) les similarités entre requêtes utilisant des distances différentes (p. ex. cosinus, Jaccard, etc.), (c) proposer des outils aidant à formuler des requêtes en recommandant des fragments de requêtes (clauses SQL) à un utilisateur exprimant sa requête Khoussainova et al. (2010). Dans les entrepôts de données, un nombre de travaux s'est intéressés à la suggestion des requêtes Aufaure et al. (2013); Negre. (2009); Aligon (2013); Giacometti et al. (2011). Dans Aligon (2013), une approche a été proposée pour analyser les

requêtes analytiques émises et les stocker de façon structurée afin d'augmenter leur réutilisation et leur exploitation pour les systèmes de recommandation.

Le traitement et l'optimisation des requêtes ont été l'un des thèmes de recherche les plus actifs dans le domaine des bases de données et systèmes d'information. Par scholar.googling "query processing databases", et en choisissant la période 2000-2018, nous trouvons 55 900 résultats. Les requêtes peuvent être exécutées de manière isolée ou conjointe. Dans ce dernier, l'exploitation des résultats intermédiaires communs des requêtes est généralement considérée lors de la sélection des structures d'optimisation. Les problèmes liés à l'optimisation des requêtes et à la conception physique sont tous deux qualifiés de tâches difficiles. Ils intègrent plusieurs paramètres appartenant aux différentes phases du cycle de vie de la conception de base de données/entrepôt de données : conceptuel, logique et déploiement et contraintes (par exemple stockage, maintenance, énergie). Par conséquent, leurs espaces de recherche peuvent être très grands Bellatreche et al. (2009) Chaudhuri et al. (2004); Sellis et Ghosh (1990).

Les requêtes qui représentent l'une des entités les plus importantes de la technologie de base de données sont également concernées par le partage des opérations intermédiaires. Le partage de requêtes a suscité un intérêt particulier dans les années 80, où Timos Sellis a identifié un nouveau phénomène connu par l'interaction des requêtes et a donné lieu à un nouveau problème appelé Optimisation multi-requêtes (MQO) T.K.Sellis (1988). La résolution de ce problème vise à optimiser les performances globales d'une charge de requêtes, en augmentant la réutilisation des résultats intermédiaires des requêtes. L'une des principales caractéristiques de ce problème de recherche est qu'il a été abordé dans tous les langages de requêtes associés à toutes les générations de bases de données : bases de données traditionnelles T.K.Sellis (1988) bases de données orientées objets Zdonik et D. Maier (1990), bases sémantiques Goasdoué et al. (2011) Le et al. (2012), bases de données distribuées A. Kementsietsidis et Vansummeren (2008), bases de données de flux Dobra et al. (2016), entrepôts de données (SQL OLAP) Yang et al. (1997). Dans toutes ces générations, le volume de requêtes a été ignoré, sauf dans le contexte des entrepôts de données, où Boukorca et al. (2015) a proposé une structure d'hypergraphe pour optimiser statiquement un nombre important de requêtes.

3 Notre proposition

Avant de détailler notre proposition, nous formalisons d'abord notre problème. Étant donné un ensemble de requêtes régulières connues à l'avance, et un ensemble de requêtes recommandées arrivées dynamiquement, une contrainte de stockage pour les vues matérialisées, notre problème consiste d'abord à optimiser les requêtes régulières en utilisant les vues matérialisées, ensuite utiliser leur optimisation pour optimiser les requêtes recommandées en se basant sur la distance de similarité entre elles. La sélection des vues matérialisées se fait à l'aide d'une structure d'hypergraphe afin de gérer le volume de requêtes.

3.1 Hypergraphe au service de l'optimisation de requêtes

L'architecture générale de notre méthodologie est décrit dans la figure 1. Nos propositions d'optimisation de requêtes sont basées sur la structure de données d'hypergraphe.

Définition 1 *Un hypergraphe $H = (V, E)$ est une structure composée d'un ensemble V d'éléments appelés sommets, et d'un ensemble E de sous-ensembles de V appelés hyper-arêtes.*

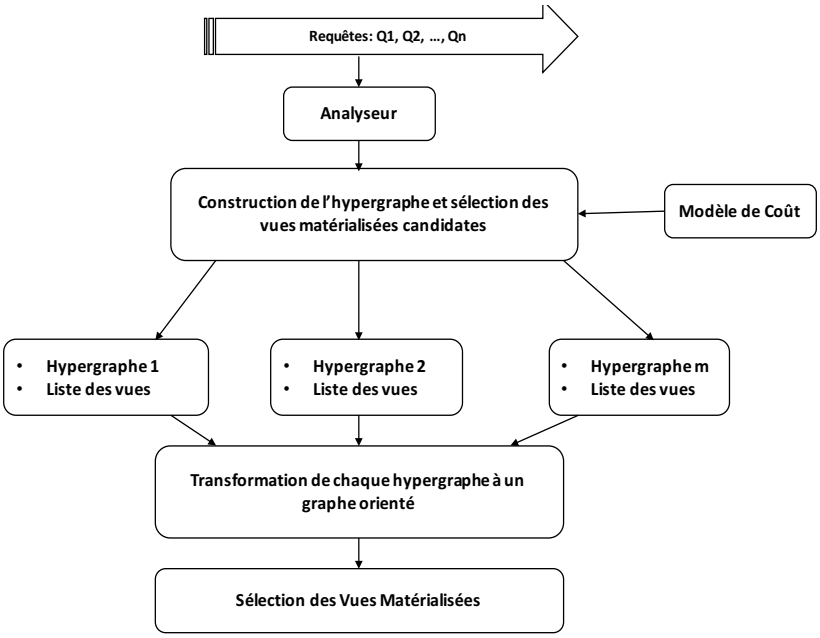

FIG. 1: Notre Approche Globale

$V(H)$ et $E(H)$ désignent respectivement l'ensemble des sommets et l'ensemble des hyper-arêtes de H. Un graphe est donc un hypergraphe dont chaque hyper-arête ne comporte que deux sommets.

Notre processus d'optimisation des requêtes régulières passe par les étapes suivantes :

Étape 1 : analyse chaque requête régulière de notre charge afin d'identifier les opérations logiques (nœuds) d'une requête (Sélection-Projection-Jointure).

Étape 2 : construit les composantes (sous-hypergraphes) de façon incrémentale. Pour se faire, il est indispensable de définir une algèbre qui manipule les hypergraphes de façon incrémentielle. Plusieurs primitives sont proposées : : *add-node(), add-hyperedge(),remove-node(),remove-hyperedge()*, etc. Les sommets et les arêtes de notre hypergraphe représentent respectivement les jointures (les opérations les plus coûteuses dans les bases de données) et les requêtes entre les sommets. Cette relation existe si les sommets partagent une ou plusieurs jointures. La construction des sous-hypergraphes se fait comme suit : lorsque la première requête $Q1$ arrive, nous construisons le premier composant (le premier hypergraphe). Ensuite, à l'arrivée de la deuxième requête $Q2$, nous testons si elle partage des nœuds de jointure avec la requête précédente $Q1$. Dans le cas où elle partage cette jointure, $Q2$ est alors placée dans le premier composant en utilisant la fonction *add-hyperedge()* ajoute un hyper-arête à un hypergraphe. Dans le cas contraire, un nouveau hypergraphe est construit associé à la $Q2$. Lorsqu'une requête $Q3$ arrive, nous calculons son partage de nœuds de jointure avec les composants existants, puis on la place dans le composant avec lequel elle partage le maximum des opérations de jointure. Ce processus est répété pour chaque requête régulière de la charge. Algorithme 1 décrit le processus incrémental pour construire des hypergraphes.

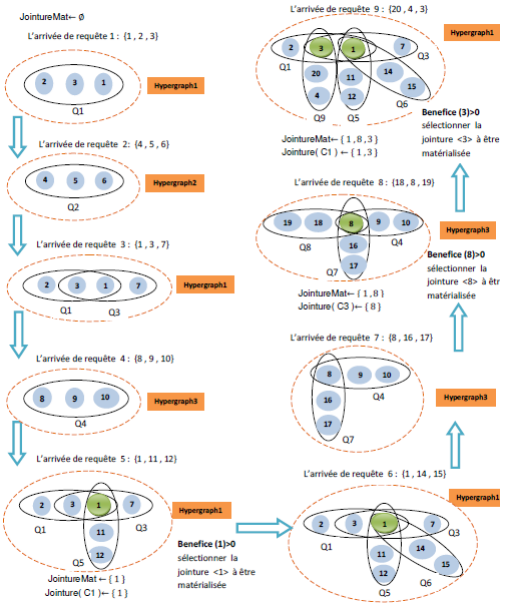

FIG. 2: Exemple de construction incrémentale des hypergraphes et la sélection des vues matérialisées

La figure 2 montre un exemple pour l'étape de la construction incrémentielle des hypergraphes et la sélection dynamique des vues matérialisées.

La Figure 2 montre un exemple du résultat de la construction incrémentielle des composants (hypergraphes) et de la sélection des vues matérialisées pour un jeu de 9 requêtes régulières, trois composantes ont été créées incrémentalement (hypergraphe1, hypergraphe 2 et hypergraphe3) et trois jointures avec des bénéfices positifs ont été sélectionnées comme candidates à la matérialisation nœud1, nœud2 et nœud3) pour le composant 1 et nœud8 pour le composant 3.

Étape 3 : la sélection des vues matérialisées se fait de façon incrémentale et simultanée avec la phase de construction des composants (sous-hypergraphes), où nous pouvons passer d'un composant à un autre pour sélectionner de nouvelles vues matérialisées, Nous pouvons également revenir à un composant pour sélectionner d'autres vues matérialisées. Pour ce faire, lorsqu'une nouvelle requête arrive à un composant, nous testons s'il y a de nouveaux nœuds de jointure avec un bénéfice positif (voir équation 1). Dans ce cas, nous appelons l'algorithme *updatematerilizedviews* qui met à jour la liste des vues matérialisées sélectionnées pour ce composant en ajoutant des jointures à bénéfice positif en tant que candidates à la matérialisation jusqu'à la saturation de l'espace de stockage. Si l'espace disque de stockage est saturé, nous supprimons les vues matérialisées les moins bénéfiques pour ajouter le plus bénéfique. Par conséquent, tous les nœuds qui ont un bénéfice positif sont considérées comme des vues matérialisées potentielles. Nous définissons le bénéfice d'un nœud n par $benefit(n_i)$ par :

$$benefit(n_i) = (nbr - 1) \times processing\_cost(n_i) - mat\_cost(n_i) \quad (1)$$

Algorithm 1 la construction incrémental de l'hypergraphe et la sélection des vues

Input W : charge de requêtes ; S : espace de stockage
output C : ensemble de composants de l'hypergraphe ; $jointure(C)$: l'ensemble de jointures de tous les composants ; $JointureMat(C)$: la liste globale des jointures candidates à la matérialisation
$jointure(C) \leftarrow \emptyset$
$F \leftarrow \emptyset$
for all $Q_i \in W$ **do**
    parse-query(Q_i)
    if F = 0 **then**
        CreateNewComponent(C)
        add-hyperedge (C , Q_i, set of join nodes of Q_i)
        F.add(C)
    else
        for all $C \in F$ **do**
            Calculate-sharing-joinoperation(Q_i, C)
        $Max \leftarrow \{Calculate - theMaximum - sharing(F, Q_i)\}$
        if Max = 0 **then**
            CreateNewComponent(C)
            add-hyperedge (C, Q_i, les jointures de Q_i)
            F.add(C)
        else
            $pos \leftarrow \{GetPosofComponentWithMaxSharing(F, Max)\}$
            add-hyperedge (F.get(pos) , Q_i, les nœuds de jointure de Q_i)
            UpdateMaterializedViews (F.get(pos), Q_i, S, jointure(C), JointureMat(C))
end

où nbr, $processing\_cost(n_i)$ et $mat\_cost(n_i)$ représentent respectivement le nombre de requêtes qui utilisent le nœud de jointure n_i, le coût de traitement de n_i et le coût de matérialisation de n_i Boukorca et al. (2015). Algorithme 1 décrit le processus de mise à jour de la liste des jointures à matérialiser pour un composant donné.

Algorithm 2 UpdateMaterializedViews

Input NReq (nouvelle requête), C_i : un composant i de l'hypergraphe, $Jointure(C_i)$: l'ensemble de jointures candidates à la matérialisation de composant C_i, $JointureMat$: la liste globale des jointures candidates à la matérialisation
output $Jointure(C_i)$, $JointureMat$;
$Joins \leftarrow \{RetournerlesJointures(NReq)\}$
calculer Benefice (joins)
$L \leftarrow \{RetournerlesJointureCandidats(Joins)\}$
ordre décroissant (L)
for all $nud \in L$ **do**
    if ($nud \notin Jointure(C_i)$) and $diskSpace \leq DS$ **then**
        add (nœud , Jointure(Ci)
        add (nœud ,Jointure(Ci), JointureMat)
        $diskSpace \leftarrow \{diskSpace - size(nud)\}$]
        ordre décroissant (Jointure(Ci))
    else
        if ($nud \notin Jointure(C_i)$) and $diskSpace > DS$ **then**
            $JointureMat \leftarrow RetournerToutesJointuresCandaidates()$
            tri croissant (JointureMat);
            if $benefit(nud) > benefit(JointureMat[0])$ **then**
                Delete JointureMat [0]
                $diskSpace \leftarrow \{diskSpace - size(JointureMat[0])\}$]
                add (nœud ,Jointure(Ci))
                $diskSpace \leftarrow \{diskSpace + size(nud)\}$]
                tri décroissant (Jointure(Ci))
end

Étape 4 : génère le plan unifié de requête (UQP) pour chaque composant en transformant chaque sous-hypergraphe en un graphe orienté en utilisant un modèle de coût mathématique A.Kerkad et al. (2012) et des algorithmes d'implémentation pour ordonner les nœuds. Cette étape est nécessaire pour ordonner les nœuds de jointure sélectionnés auparavant comme candidats à la matérialisation (à l'étape 3). L'ajout d'un arc au graphe orienté correspond à l'établissement d'un ordre entre deux nœuds de jointure. La transformation a trois étapes : (1) choisir le nœud pivot (pivot), le nœud pivot correspond au nœud qui a le meilleur bénéfice possible de la réutilisation des résultats intermédiaires. (2) transformer le nœud pivot de l'hypergraphe au graphe orienté. (3) retirer le nœud pivot de l'hypergraphe. Nous mentionnons que nous nous sommes inspirés de travail de Boukorca et al. (2015) pour générer l'UQP qui a assuré la scalabilité de notre approche. La figure 3 montre un exemple pour l'étape de transformation d'un hypergraphe en un graphique orienté. Nous mentionnons qu'on a gardé une copier des sous-

hypergraphes crées car nous aurons besoin d'eux plus tard pour capter le partage des jointures dans la phase d'optimisation des requêtes recommandées. .

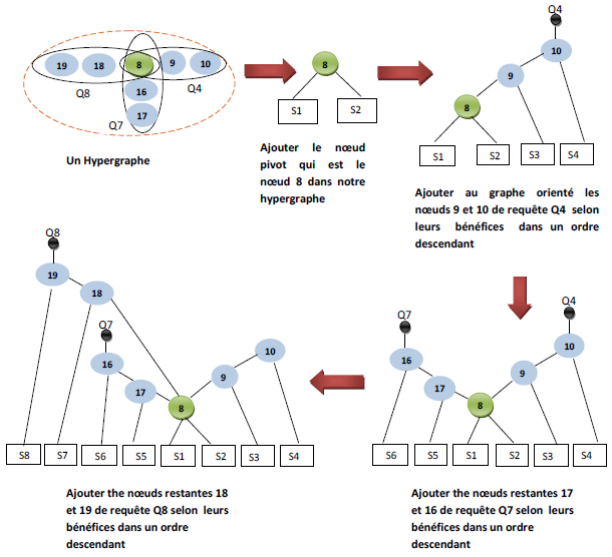

FIG. 3: l'étape de transformation de l'hypergraphe en graphe orienté UQP

Étape 5 : optimise les requêtes régulières de chaque composant. Nous matérialisons les jointures précédemment sélectionnées comme candidates à la matérialisation(étape 3) en fonction de leur ordre dans chaque plan unifié (UQP). Pour chaque UQP nous matérialisons le nœud pivot de l'UQP, puis nous matérialisons les autres jointures en fonction de leur bénéfice en prenant en compte les jointures matérialisées précédemment.

4 Réutilisation des optimisations des requêtes régulières pour les recommandées

Dans cette section, nous montrons comment les optimisations de requêtes régulières peuvent être réutilisées pour les requêtes recommandées. Le processus de réutilisation se passe par les étapes suivantes :

- **Étape 1 :** analyse la requête recommandée en extrayant ses opérations algébriques (jointures, sélections, projections).
- **Étape 2 :** place la requête recommandée dans le composant approprié (sous-hypergraphe). Le processus de placement de la requête recommandée est basé sur une distance définie entre la requête recommandée et les composants des requêtes existants. La requête de recommandation est alors placée dans le composant qui partage avec elle le maximum d'opérations de jointure de ce composant (potentiellement matérialisables).
- **Étape 3 :** optimise la requête recommandée en utilisant l'historique d'optimisation des requêtes régulières. Pour ce faire, nous réécrivons la requête recommandée en fonction

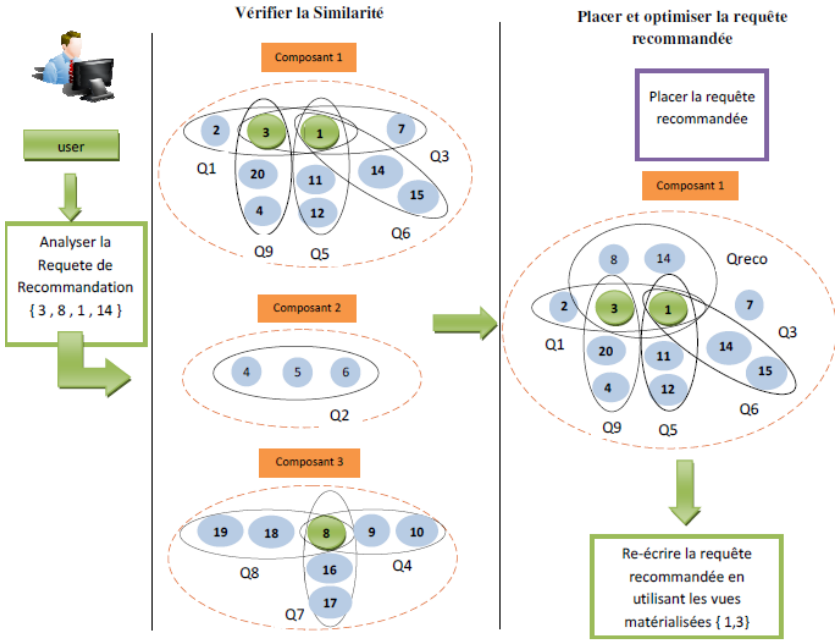

FIG. 4: Un exemple sur notre proposition pour optimiser les requêtes recommandées

des vues matérialisées sélectionnées précédemment des composants. Le processus de réécriture utilise le graphe unifié de requêtes (UQP) générés précédemment.
La figure 5 montre un exemple pour notre proposition d'optimiser les requêtes de recommandation en utilisant l'historique des optimisations des requêtes régulières, où la requête recommandée est placée dans le composant 1 qui partage avec elle le maximum de jointures matérialisées précédemment pour optimiser les requêtes régulières.

5 Validation de notre approche

Pour évaluer l'efficacité et la qualité de notre approche, nous effectuons une série de tests. Dans un premier temps, nous menons une étude expérimentale pour analyser les performances de notre proposition de construction incrémentale d'hypergraphes (composants) en comparaison avec le travail de Boukorca et al. (2015). Ensuite, nous évaluons la performance de la charge des requêtes régulières en utilisant notre approche dynamique. Enfin, nous évaluons notre proposition d'optimisation des requêtes recommandées gérées à la demande des utilisateurs et basée sur l'historique de l'optimisation dédiée aux requêtes régulières. Nos expériences ont été réalisées sur un serveur HP avec 8 Go de RAM et 1 To de disque dur. Nous considérons le Benchmark du schéma en étoile (SSB) qui a une table de faits *Lineorder* et quatre tables de dimension : *Customer*, *Supplier*, *Part* et *Dates* avec une taille de 30 Go. Ce schéma est interrogé par différentes charges de requêtes.

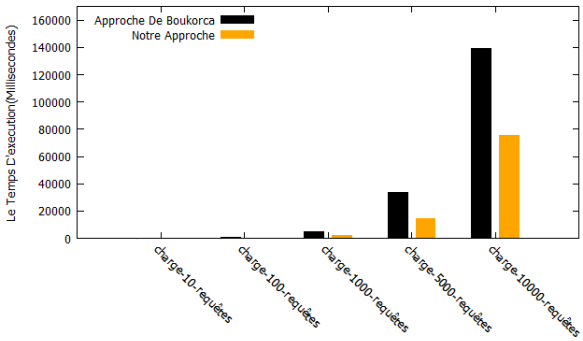

FIG. 5: Le temps d'exécution pour générer les composants connectés (sub-hypergraphes)

Nous avons effectué une série de tests pour comparer notre approche et celle de Boukorca et al. (2015) en termes de temps d'exécution de plusieurs charges de requêtes de différentes tailles (générées aléatoirement en utilisant le générateur de requêtes de notre banc d'essai SSB). La Figure 5 résume les résultats obtenus qui montrent que notre approche surpasse largement l'algorithme de Boukorca et al. (2015). L'efficacité de notre approche est principalement due à la génération des composants connectés de l'hypergraphe. Table 1 donne une comparaison de notre approche avec de Boukorca et al. (2015), en prenant en compte le temps d'exécution pour générer les composants connectés tout en variant la charge de requêtes.

charge de requêtes	notre programme	le programme de Boukorca
10	26 ms	330 ms
100	240 ms	510 ms
1000	2154 ms	4691 ms
5000	14682 ms	33891 ms
10000	75748 ms	139310 ms

TAB. 1: Temps d'exécution de notre approche et celle de Boukorca

Pour évaluer l'impact de notre algorithme incrémentiel de sélection des vues matérialisées, nous avons utilisé un modèle de coût développé dans A.Kerkad et al. (2012) pour estimer le coût de traitement de la charge avec et sans notre algorithme. Les résultats obtenus sont décrits dans la figure 6 montrant l'impact positif de notre approche de sélection des vues.

Afin de confirmer les résultats théoriques, nous avons exécuté notre programme pour différentes charges de requêtes régulières, ensuite nous avons déployé les résultats dans un SGBD Oracle 12c. Les figures 7 et 8 comparent les coûts réels de traitement et le temps d'exécution de différentes charges de requêtes régulières en utilisant notre approche. Les résultats obtenus coïncident avec ceux obtenus théoriquement.

Afin de tester l'efficacité de notre approche pour un grand nombre de requêtes. Nous avons effectué des expérimentations pour évaluer le cout d'exécution de différentes charges de requêtes avec et sans utiliser notre approche, et nous calculons le taux d'optimisation *(1-coût*

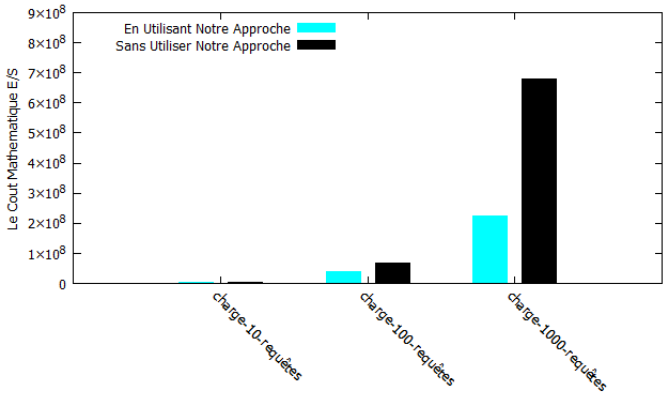

FIG. 6: Le Coût Mathématique de traitement de différentes charge de requêtes avec et sans notre sélection des vues

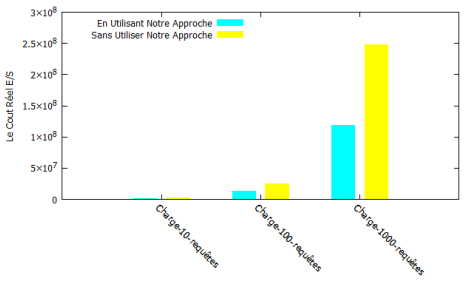

FIG. 7: Validation sous Oracle 12c : Le cout de traitement

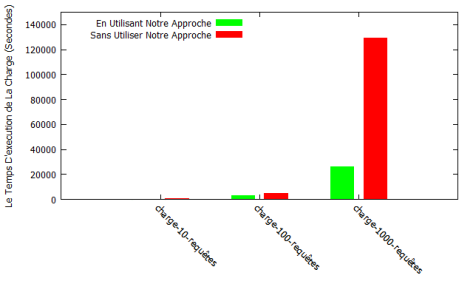

FIG. 8: Validation sous Oracle 12c : le temps d'exécution

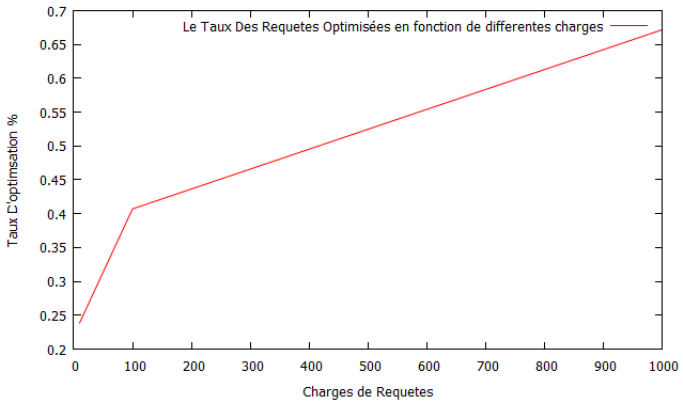

FIG. 9: Le taux d'optimisation du coût d'exécution de différentes charges

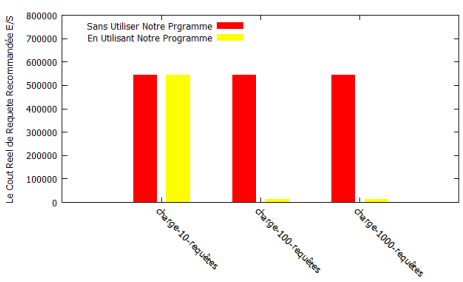

FIG. 10: Le coût de traitement Oracle pour la requête recommandée

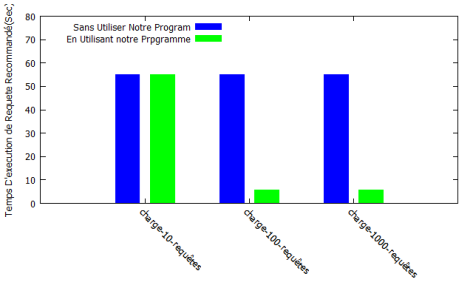

FIG. 11: Temps d'exécution pour la requête recommandée

avec / coût sans). La figure 9 montre les résultats obtenus qui montrent que notre approche devient plus intéressante quand le nombre de requêtes augmente (taux d'optimisation de 70 % pour une charge de 1 000 requêtes), ce qui prouve l'efficacité de notre approche.

Nous avons effectué une expérience pour évaluer les performances de notre système qui prend en compte les requêtes recommandées dans le processus d'optimisation. Pour ce faire, une batterie de tests a été effectuée afin d'évaluer l'effet des vues matérialisées précédemment sélectionnées et dédiées aux requêtes régulières sur la performance de la requête recommandée en changeant la charge des requêtes régulières. Dans cette expérience, nous recommandons une requête généré aléatoirement par le générateur aléatoire de requêtes *Qgen* du banc d'essai SSB, et nous déployons les résultats dans un SGBD Oracle 12c. Les figures 10 et 11 comparent les coûts de traitement et le temps d'exécution de la requête recommandée, en utilisant et sans utiliser notre approche. Le coût réel a été estimé par l'outil d'Oracle *EXPLAIN PLAN*.

En analysant les résultats obtenus de ces expériences, nous constatons que la performance des requêtes recommandées peut être améliorée en se basant sur l'historique d'optimisation dédiés aux requêtes régulières, ce qui confirme l'efficacité de notre proposition.

6 Conclusion

Dans cet article, nous avons présenté une approche dynamique pour optimiser simultanément des requêtes régulières et recommandées en considérant un volume important de requêtes. Pour faire face à ce volume, nous avons utilisé la structure des hypergraphes, où ils ont prouvé une efficacité dans la gestion du volume de requêtes et l'interaction entre eux dans plusieurs domaines (VLSI, réseaux sociaux, etc.). Notre approche considère un ensemble de requêtes régulières connues à l'avance et un ensemble de requêtes recommandées arrivant d'une manière dynamique et non connues. D'abord, nous optimisons les requêtes régulières en exploitant leur interaction qui permet d'identifier des vues matérialisées candidates. En se basant sur l'historique d'optimisation des requêtes régulières, les requêtes recommandées sont optimisées sur la base de cet historique. Nos propositions sont validées théoriquement à l'aide d'un modèle de coût et réellement sur le SGBD Oracle. Les résultats obtenus sont encourageants. Actuellement, nous nous intéressons à rendre dynamique l'arrivée des requêtes régulières.

Références

A. Kementsietsidis, F. Neven, D. V. d. C. et S. Vansummeren (2008). Scalable multi-query optimization for exploratory queries over federated scientific databases. *PVLDB*, 16–27.

Adomavicius, G. et A. Tuzhilin (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng. 17*(6), 734–749.

A.Kerkad, L.Bellatreche, et D. Geniet (2012). Queen-bee:query interaction aware for buffer allocation and scheduling problem. *DaWaK*, 156–167.

Aligon, J. (2013). *Similarity Based Recommendation of OLAP Sessions*. Phd thesis, Universite Francois Rabelais Tours.

Aligon, J., E. Gallinucci, M. Golfarelli, P. Marcel, et S. Rizzi (2015). A collaborative filtering approach for recommending OLAP sessions. *Decision Support Systems 69*, 20–30.

Aufaure, A.Beauger, K. Marcel, P. Rizzi, S. Vanrompay, et al. (2013). Predicting your next olap query based on recent analytical sessions. *In Proceedings DaWaK*.

Aufaure, M.-A. et E. Zimányi (2011). Business intelligence. *Springer*.

Bellatreche, L., K. Boukhalfa, P. Richard, et K. Y.Woameno. (2009). Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. *IJDWM*, 1–23.

Bellatreche, L. et A. Kerkad (2015). Query interaction based approach for horizontal data partitioning. *IJDWM 11*(2), 44–61.

Bouchakri, R. (2015). *Conception physique statique et dynamique des entrepôts de donnees*. Phd thesis, ESI and DE ISAE-ENSMA.

Boukorca, A., L. Bellatreche, et A. Cuzzocrea (2014). SLEMAS : an approach for selecting materialized views under query scheduling constraints. In *COMAD*, pp. 66–73. Computer Society of India.

Boukorca, A., L. Bellatreche, S. B. Senouci, et Z. Faget (2015). Coupling materialized view selection to multi query optimization : Hyper graph approach. *IJDWM 11*(2), 62–84.

Breß, S., H. Funke, et J. Teubner (2016). Robust query processing in co-processor-accelerated databases. In *ACM SIGMOD*, pp. 1891–1906.

Bretto, A. (2013). *Hypergraph Theory : An Introduction*. Springer Publishing Company, Incorporated.

Chaudhuri, S., M. Datar, et V. Narasayya (2004). Index selection for databases: A hardness study and a principled heuristic solution. *IEEE TKDE*, 1313–1323.

Curino, C., Y. Zhang, E. P. C. Jones, et S. Madden (2010). Schism: a workload-driven approach to database replication and partitioning. *PVLDB 3*(1), 48–57.

Dobra, A., M. Garofalakis, J. Gehrke, et R. Rastogi (2016). Sketch-based multi-query processing over data streams. *In Data Stream Management , springer*, 241–261.

Eirinaki, M., S. Abraham, N. Polyzotis, et N. Shaikh (2014). Querie: Collaborative database exploration. *IEEE TKDE 26*(7), 1778–1790.

Giacometti et al. (2011). Query recommendations for olap discovery-driven analysis. *IJDWM*.

Giacometti, Marcel, et Negre (2009). Recommending multidimensional queries. *Dawak*, 453–466.

Goasdoué, F., K. Karanasos, J. Leblay, et I. Manolescu (2011). View selection in semantic web databases. *PVLDB 5*(2), 97–108.

Jerbi, H., F. Ravat, O. Teste, et G. Zurfluh (2009). Preference-based recommendations for OLAP analysis. In *DaWaK*, pp. 467–478.

Khoussainova, Kwon, Balazinska, et Suciu. (2010). Snipsuggest: Context-aware autocompletion for sql. *PVLDB*.

Le, W., A. Kementsietsidis, S. Duan, et F. Li (2012). Scalable multi-query optimization for SPARQL. In *ICDE*, pp. 666–677. IEEE Computer Society.

Negre. (2009). *Exploration collaborative de cubes de données*. Thèse de doctorat, Universite Francois Rabelais Tours.

Negre, E. (2017). Prise en compte du contexte dans les systèmes de recommandations de requêtes OLAP. In *EDA*, Volume B-13 of *RNTI*, pp. 1–10. Éditions RNTI.

Roukh, A., L.Bellatreche, S.Bouarar, et A.Boukorca (2017). Eco-physic: Eco-physical design initiative for very large databases. *Information Systems*, 44–63.

Sellis, T. et S. Ghosh (1990). On the multiple query optimization problem. *IEEE Transactions on Knowledge and Data Engineering*, 262–266.

T.K.Sellis (1988). Multiple-query optimization. *ACM Transactions on Database System. (TODS)*.

Yang, J., K. Karlapalem, et Q. Li (1997). Algorithms for materialized view design in data warehousing environment. In *VLDB*, pp. 136–145. Morgan Kaufmann.

Zdonik, S. B. et e. D. Maier (1990). Readings in object-oriented database systems. *Morgan Kaufmann*.

Summary

Nowadays, advanced database applications manage two main types of queries: regular queries issued by end users and queries issued automatically by systems such as recommendation systems. The main objective of recommended queries is to suggest new interesting items to users for a large pool of items that may include movies, books, friends, articles, optimization structures, etc. From a DBMS perspective, recommended queries represent an important overhead, since it shall optimize them as it does for regular queries. Usually, queries belonging to these two types are active at the same time. As a consequence, multiple queries can be evaluated more efficiently together than independently, because it is often possible to share state and computation. Motivated by this observation, we propose in this paper, a new approach that dynamically optimizes these two types of queries using materialized views. Our approach first optimizes regular queries according their arrival, and when a recommended query arrives it is optimized based on the optimization history of the regular queries by the means of similarity measures. To ensure the scalability of our approach, we use a hyper-graph structure that capture the interaction between queries regardless of their types. Our approach is evaluated using Star Schema Benchmark and the obtained results are deployed in a commercial DBMS.