Optimisation de requêtes dans un environnement de grille de données

Ayouni Houssam Eddine* - Belbachir Hafida*

*Département d'informatique
Université des Sciences et de la Technologie Mohamed Boudiaf Oran
Oran, Algérie
ayouni.houssam@gmail.com, h_belbach@yahoo.fr

RÉSUMÉ. Malgré l'évolution rapide des systèmes réseaux à grande échelle, les grilles de données affrontent de nouveaux problèmes pendant la phase de traitement de requêtes distribuées. Nous proposons dans cet article, une nouvelle stratégie permettant d'optimiser l'exécution d'une requête en tenant compte les problèmes de découverte de nœuds, et l'allocation efficace des répliques. Ainsi, nous proposons d'exécuter chaque opérateur relationnel d'un plan d'exécution par deux agents mobiles, en attachant chaque agent au nœud stockant la réplique intervenant dans l'opérateur. Enfin, un modèle de coût est déterminé afin de calculer le coût d'exécution d'une requête dans un environnement de grille.

ABSTRACT. Despite the rapid evolution of large scale network systems, data grids are facing new problems during the distributed queries processing. We propose in this paper, a new strategy for optimizing the query execution, taking into account the problems of discovery of nodes, and the efficient allocation of replicas. Thus, we propose to execute each relational operator of an execution plan by two mobile agents, by attaching each agent to the node storing the replica involved in the operator. A mobile agent can migrate to another site to improve the query execution cost. This cost is calculated using a cost model that we have determined.

MOTS-CLÉS : grille de données, optimisation de requête, base de données, model de cout, agents mobiles.

KEYWORDS: data grid, query optimization, database, cost model, mobile agents.

1. Introduction

Avec l'évolution rapide des systèmes réseaux à grande échelle tel que l'internet, les utilisateurs peuvent interroger des sources de données autonomes, hétérogènes et distribuées sur un réseau à grande échelle. Les grilles de données sont l'un des systèmes à grande échelle les plus utilisé pour exploiter ces sources de données. Les utilisateurs d'une grille de données peuvent soumettre leurs requêtes d'une façon autonome, hétérogènes et qui fait intervenir des sources de données distribuées provenant de plusieurs sites reliés par un réseau de faible débit, et de forte latence. Ces caractéristiques posent de nouveaux problèmes qui empêchent la bonne exécution de la requête. La réplication des données est l'un des problèmes majeurs (la découverte, et l'allocation des répliques).

Notre contribution est composée de plusieurs solutions, afin d'assurer la bonne exécution de la requête : définir une méthode de découverte de ressources (sites et répliques), en se basant su le service d'information grille (GIS), la description d'un algorithme pour l'allocation efficace de répliques intervenantes dans la requête, nous utilisons ensuite un modèle d'exécution à base d'agents mobiles pour l'exécution efficace des opérateurs relationnels binaires (jointures) de la requête, et enfin, un modèle de cout est déterminé pour calculer le cout d'exécution de requêtes.

2. Travaux lies

Dans les derniers décennies, de nombreux chercheurs ont consacrés leurs recherches dans le cadre de traitement de requêtes dans les environnements de grille [1, 2, 3, 4, 5, 6]. Dans ce contexte, la conception et l'implémentation d'une technique efficaces d'optimisation de requêtes pour l'environnement de grille a pris une très grande importance. Prenant en compte les contraintes de la grille, un modèle de coût pour le calcul du coût d'exécution d'une requête, a été introduit dans [1]. Dans [2], un autre modèle de coûts est définit pour les bases de données des environnements de grille dynamique, un algorithme d'optimisation de requête dynamique est aussi utilisé pour les plans de requête, qui sont adaptés aux fluctuations de l'environnement de grille. Les auteurs de [3,4] proposent un nouveau modèle d'optimisation des requêtes distribuées qui intègre trois phases d'optimisation de requêtes : la création du plan d'exécution, la génération d'un plan parallèle, et le choix optimal des sites pour l'exécution de ce plan. Dans [5], un optimiseur de requête sémantique pour un environnement de grille est proposé, ce dernier traite essentiellement la phase d'optimisation de requête selon trois modules : l'extension sémantique du requête de l'utilisateur, la sélection de ressources, et le traitement parallèle. Dans [6], l'équipe de Hameurlain a défini un modèle d'exécution à base d'agents mobiles pour l'optimisation dynamique de requêtes

réparties à grande échelle. Bien que ces techniques d'optimisation des requêtes, soient puissantes, elles aussi souffrent d'un certain nombre de limites. La première limite est dû au fait que ces approches ne prend pas en considération les paramètres de hétérogénéité des nœuds de la grille (la mémoire disponible, la vitesse de traitement etc.), alors que le deuxième problème est que ces approches ne traitent pas le problème de la réplication des données sur les différents nœuds de la grille. Ces limites influencent négativement sur les performances et le temps de réponse global de la requête.

3. La structure Traitement de requête dans un environnement de grille de données

Dans cette section, nous présentons notre architecture d'évaluateur de requêtes en environnement de grille de données, en expliquant les différentes phases.

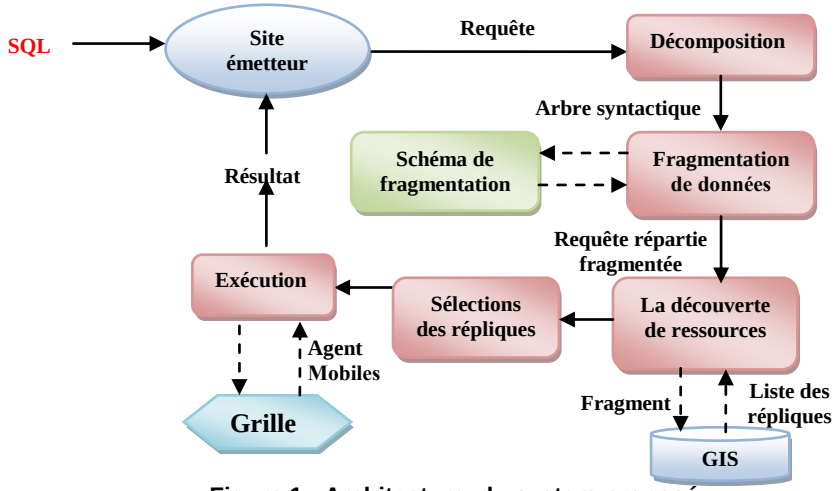

Figure 1 : Architecture du system proposé.

1. La décomposition et la fragmentation de données : À ce niveau, la requête est réécrite sous forme normalisée, puis l'arbre syntaxique (plan de la requête) est généré.

2. La découverte de ressources (sites stockant les répliques) : dans cette phase, il s'agit de la découverte d'un sous-ensemble de sites disponibles pour l'exécution de la requête. Dans ce contexte, une stratégie simple consiste à découvrir les ressources de calcul. La stratégie de découverte se base sur le service d'information grille (GIS) [7], qui contient des métadonnées statiques décrivant les sites: Time_I/O (temps nécessaire pour lire et écrire une page), Time_CPU (temps pour effectuer une opération, ex : temps de comparaison de deux attributs), Size_MEMO (taille mémoire en octet i.e. la taille

d'une page |p| = 4 Ko), et Trans(S_i, S_j) (temps moyen pour envoyer une page du site Si vers Sj, avec i≠j). Cette phase consiste à découvrir pour chaque réplique R référencée dans la requête, l'ensemble des sites E={S_i,…,S_m} stockant R. A chaque site S_k, est associé un temps Time(S_{emet} , S_k) représentant le temps de réponse du site S_k au site émetteur S_{emet} , avec : $Time(S_{emet}, S_k) = Time\_I/O(S_k) + Time\_CPU(S_k) + Trans(S_k, S_{emet})$ (1)

3. Sélection des répliques : Dans cette phase, on va sélectionner les sites qui vont minimiser le coût d'exécution de la requête. L'idée est de choisir si possible pour chaque opérateur les sites qui sont communs à ses opérandes (op_i). Dans ce contexte, on a proposé la stratégie suivante : (1) tout d'abord, le plan de la requête est divisé en plusieurs niveaux (voir exemple Figure 2). La figure 2 donne un exemple d'un plan d'exécution de requête, ou les nœuds sont les opérateurs, et les feuilles (R1,R2,R3 ,R4,R5,R6) sont des fragments. A chaque fragment est associé à l'ensemble de sites ou se trouve une copie du fragment.

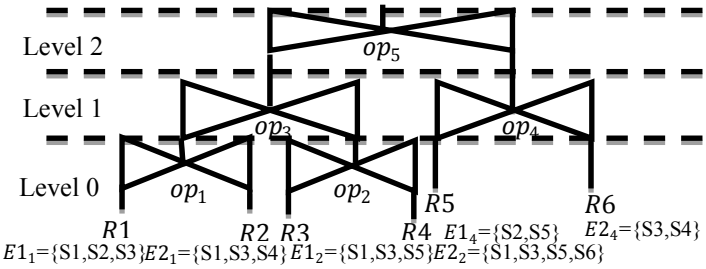

Figure 2 : Exemple de plan de requête avec niveaux.

Dans l'étape (2), on applique l'algorithme suivant :

i : indice des niveaux
j : indice des opérateurs du même niveau
$E1_j$: L'ensemble des sites stockant les répliques du 1er opérande de l'opérateur op_j
$E2_j$: L'ensemble des sites stockant les répliques du 2eme opérande de l'opérateur op_j
i := 0 ; j :=1 ;
pour** chaque niveau i **faire
pour** chaque operateur j (op_j) **faire
- *trouver les sites en commun stockant les opérandes : $E_{opj} = \{E1_j \cap E2_j\}$*
- ***Si** $E_{opj} = \emptyset$ **alors** : **Pour** chaque $E1_j$ et $E2_j$ **faire** : choisir le site S_k avec min (Time(S_{emet} , S_k))*

Fin
- *$E = \{E_{op1} \cap \ldots \cap E_{opj}\}$ avec $E_{opj} \neq \emptyset$ /*trouver les sites en commun entre les opérateurs du même niveau */*
- ***Si** $E \neq \emptyset$ **alors** : **Pour** chaque E_{opj} **faire** :*
  - ***Si** $S_{emet} \in E$ **alors** choisir S_{emet} **sinon** choisir S_k tel que $S_k \in E$ et min(Time(S_{emet} , S_k))*
  - ***Sinon** (**Si** $E = \emptyset$) **alors** :**pour** chaque operateur E_{opj} **faire** :*
    - ***Si** $S_{emet} \in E_{opj}$ alors choisir S_{emet} **sinon** choisir S_k tel que $S_k \in E_{opj}$ et min(Time(S_{emet} , S_k)*

Fin

Algorithme 1 : Allocation des répliques.

4. Exécution : On a défini dans la partie précédente le choix initial des sites qui contiennent les répliques nécessaires pour l'exécution de la requête. Dans ce contexte, on va définir une méthode d'exécution des opérateurs binaires OPj (jointure) du niveau en cours, en se basant sur un modèle d'agents mobiles. Un agent mobile est une entité logicielle autonome et adaptable, capable de se déplacer dynamiquement (code, données) pour accéder à des données ou à des ressources distantes. Notre stratégie d'exécution va suivre deux cas :

1er cas : si les deux opérandes (répliques) alloués précédemment se trouvent dans le même site alors : Exécution immédiate de l'opérateur par le même site en utilisant la jointure simple (pas du transfert de données)

2eme cas : si les deux opérandes (R et S) alloués se trouvent dans deux sites différents (S1 et S2) alors : On associe un agent mobile (AG1 et AG2) pour chaque site (S1 et S2 respectivement), et chaque agent vérifie :

1) L'état local du site Si en se basant sur les paramètres machine (la mémoire disponible, la mémoire utilisée, le nombre d'E/S par seconde, le nombre de processus actifs, le nombre des processus suspendus [8]) fournit par le GIS.

2) L'état de la bande passante entre son binôme en basant sur les paramètres réseaux (BW(S1,S2) [1]: la charge de la bande passante entre S1 et S2) fournit par le GIS.

a) Si un site S_i est saturé alors l'agent mobile (AG) associé doit :

1) Migrer vers un autre site S_k parmi les sites qui contient la même réplique (R), découvertes précédemment s'il existe avec min($CoutMigration_{Si,Sk}(AG)$[2] + *CoutProd(R)*)[3] , sinon on passe vers l'étape 2:

2) Migrer avec la réplique vers un autre site S_k avec min($CoutMigration_{Si,Sk}(AG)$).

b) Sinon on applique l'algorithme de semi jointure simple en permettant à l'agent de choisir le site d'exécution de la jointure dans un environnement de grille de données.

4. Le model de coûts

Dans cette section, nous décrivons les différentes parties du modèle de coût qui interviennent dans la phase d'estimation du coût des plans d'exécution de requêtes. Nous distinguons deux parties: la partie résidante dans le site, et la partie intégrée dans l'agent

[1] BW(Si,Sj) : la charge de la bande passante entre Si et Sj avec i≠j, fournit par le GIS
[2] $CoutMigration_{Si,Sj}(AG)$: le coût de migration de l'agent mobile AG.
[3] CoutProd(Rel) : le coût nécessaire pour que l'agent mobile reçoive l'opérande Rel.

mobile. Ce modèle de coût est utilisé pour évaluer le coût d'exécution des plans des requêtes soumises.

4.1. Modèle de Coûts de Site

Le modèle du coût d'un site ***Si*** est composé de quatre unités: les profils des sources de données connues par ***Si***, les caractéristiques de ***Si*** (paramètres machine + paramètres réseaux), les caractéristiques des autres sites interagissant avec ***Si***, et les formules des coûts.

4.2. Modèle de Coûts d'Agent :

Le modèle de coûts embarqué dans un agent est fourni à l'agent pendant la phase d'optimisation. Le modèle de coûts d'un agent est composé de deux unités : (i) l'espace de migration (i.e. liste des sites détient la même source de donnée), (ii) localisation du deuxième agent mobile (l'agent mobile distant qui s'occupe de la deuxième source de donnée). Un agent mobile peut consulter le modèle de couts du site ou se trouve, pour estimer ses paramètres (le cout de migration).

4.3. Les formules de cout :

Dans un environnement de grille, les ressources qui calcul les tâches nécessaires sont les ressources informatiques telles que la mémoire, CPU, etc. Par conséquent, la tâche de calcul peut être assignée à n'importe quel nœud qui a des ressources de calcul suffisant pour cette tâche.

a) Les formules de cout des nœuds:

$$\text{Coût\_lecture}(R) = SS0 + SS1 * \|R\| + SS2 * \|R\| * FS(p) \quad (2)$$

Avec SSO coût initial pour la lecture séquentielle estimé en fonction de la charge du disque, SS1 coût unitaire pour retrouver un tuple estimé en fonction de la charge CPU d'un nœud, SS2 coût pour traiter un tuple résultat estimé en fonction de la charge de mémoire d'un nœud, FS (p) représente le facteur de sélectivité dans [9].

b) Les formules de cout des sites: Ces formules sont utilisées pour estimer le temps de réponse des opérateurs exécutés sur le site. Selon l'algorithme de jointure, la formule d'estimation du temps de réponse de la semi jointure basé sur les agents mobiles de deux relations de base T = Join (R, S) exécuté par deux agents mobiles AG1 et AG2, est la suivante:

$$Cout\text{-}Semi\text{-}Join_{R,S} = \text{Projection-Cost}(R) + \text{Join-Cost}(S, temp1) + Join\text{-}Cost(R, temp2) \quad (3)$$
$$+ \sum Migration\text{-}Cost(AG1) + \sum Migration\text{-}Cost(AG2) + CoutTrans_{S1,S2}(temp1) + CoutTrans_{S2,S1}(temp2)$$

avec : $Projection\text{-}Cost(R) = SS0 + SS1 * CARD(R) + SS2 * CARD(R)$ (4)

$Join\text{-}Cost(R, S) = Scan\_Cos(R) + CARD(R:P) * FS(p) * Scan\_Cost(S)$ (5)

$FS(p) = 1.5/maw(CARD(R), CARD(S))$ [10]

$$CoutMigration_{Si,Sj}(AG) = \begin{cases} 0 \quad \textbf{If AG does not move} \\ CoutSer + CoutDeser \quad \textbf{If AG moves without relation from } S_i \textbf{ to } S_j \\ CoutSer + CoutDeser + CoutTrans_{Si,Sj}(Rel) \quad \textbf{If AG moves with relation from } S_i \textbf{ to } S_j \end{cases} \quad (6)$$

$$CoutTrans_{Si,Sj}(Rel) = Initial(Si, Sj) + \left(\frac{|Rel|}{|p|}\right) * Trans(Si, Sj) \quad (7)$$

Ou les termes utilisés dans les formules précédentes ont la signification suivante: *Projection-Cost(R)* (le cout de projection de R), *Join-Cost(R, S)* (le cout de la jointure naturelle entre R et S), $CARD(R)$(le nombre de tuples de R), *CARD(R:P):*Le nombre de tuples de R qui satisfaire la condition de jointure P, $Migration - Cost_{Si,Sj}(AG)$ (le coût de migration de l'agent mobile AG). Le coût de migration d'un agent [10] est la somme du coût de sérialisation ($CoutSer$), du coût de dé-sérialisation ($CoutDeser$) et du coût de transfert d'un agent ($CoutTrans$), $Initial(Si, Sj)$ (le temps pour établir une communication entre deux sites Si et Sj estimé en fonction de la valeur de la latence entre Si et Sj), $CoutTrans_{Si,Sj}(Rel)$ (le temps pour envoyer une page de Si vers Sj estimé en fonction de la valeur de la bande passante entre Si et Sj), $|Rel|$est la taille de Rel, $|p|$est la taille de la page p. Dans ce cadre, nous concluons le cout total de la requête, qui représente la somme des couts des opérateurs de la requête :

$$Cost_{query} = \sum Cout\text{-}Semi\text{-}Join_{R,S} \quad (8)$$

5. Conclusion

Dans cet article, nous sommes parvenus à situer le problème et à clarifier les objectifs. On a proposé une solution pour l'optimisation de requête dans un environnement de grille de donnés en tenant en compte les contraintes de grille tels que l'hétérogénéité, la réplication, et les comportements dynamiques des ressources. Dans notre technique d'optimisation, un modèle d'agents mobiles est utilisé afin de réduire le coût de communication pour l'exécution de requêtes distribuées. Enfin, un modèle de cout pour l'estimation de cout de requête est proposé. Notre contribution future consistera à définir un modèle de simulation de grille de donnée. Enfin, on va valider notre approche finale en comparant les résultats finals.

6. Bibliographie

[1] Chhanda Ray, Nilava Guha ”Determination of Cost Model for Constraint-based Query Optimization in Data Grids “Proceeding of International Conference on Advances in Computer Science ACEEE 2010.

[2] N. Hu, Y. Luo, Y. Wang, “Adaptive Evolvement of Query Plan based on low cost in Dynamic Grid Database”, proceedings of 9th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp 411-415, 2008.

[3] Krishnamoorthy, S., Saple, A.K., and Achutharao, P.H. *An integrated query optimization system for data grids*. In Proceedings of Bangalore Compute Conference 2008.

[4] Luo Y H, Chen T F, Zhang Y S. Distributed query optimization model based on data grid. Journal of Computer Applications, 28(10): 2553-2557 (in Chinese), 2008.

[5] Luo Y H, Chen T F, Zhang Y S. Study on semantic query optimization of grid data. Computer Engineering and Applications, 45(2):16-20,2009.

[6] Franck Morvan, Abdelkader Hameurlain. *A Mobile Relational Algebra.* in : *International Journal of Mobile Information Systems*, IOS Press, Vol. 7 N. 1, p. 1-19, january 2011.

[7] B. Plale, P. Dinda, M. Helm, G. von Laszewski, and J. McGee. Key concepts and services of a grid information service. 2002.

[8] Q. Zhu, S. Motheramgari, Y. Sun: Cost Estimation for Queries Experiencing Multiple Contention States in Dynamic Multidatabase Environments. Knowl. Inf Syst.5(1): 26-49,2003.

[9] M. T. Ozsu, P. Valduriez: Principles of Distributed Database Systems, Second Edition. Prentice-Hall 1999.

[10] E.J.Shekita, H.C.Young and K.L.Tan, MultiJoin Optimization for Symmetric Multiprocessors, Proc. of the 19th International Conference on Very Large Data Bases, Dublin, Ireland (1993), 479–492.