Rubrique Arima en pratique

Résolution efficace des requêtes Top-k par Agrégation

Zekri Lougmiri[*], Timsit Hicham[**], Megague Khadidja[***], Beldjillali Bouziane[****]

Département d'informatique
Université d'Oran, Es-sénia
BP 1524, El-M'naouer, Maraval, Oran 31000
ALGERIE

* : zekri.lougmiri@univ-oran.dz,

** : timsit.hicham@gmail.com,

*** : megague.khadidja@gmail.com,

**** : beldjillali.bouziane@univ-oran.dz

RÉSUMÉ. La résolution des requêtes Top-k est un problème important et difficile dans les systèmes large échelle. Cette résolution se fait par agrégation afin d'optimiser la recherche et de manipuler moins de quantités de données. Dans cet article, nous présentons un nouvel algorithme dans le but de résoudre ce type de problème efficacement. Notre algorithme est composé de trois phases et ne définit qu'un seul seuil, à la différence des autres algorithmes déjà existants. Les simulations sur des collections de données réelles et la comparaison avec des algorithmes très connus montrent que notre solution est très compétitive et plus rapide en temps d'exécution.

ABSTRACT. The resolution of the Top-k queries is an important and a difficult problem in the large scale systems. Aggregation is the best way to solve this type of queries. It reduces substantially the amount of exchanged data and optimizes the search task. In this paper, we present a new algorithm with the aim of solving this type of problem efficiently. Our algorithm consists of three phases and defines only one threshold, unlike the other already existing algorithms. The simulations on real datasets and the comparison with some other algorithms show that our solution is very competitive.

MOTS-CLÉS : agrégation, requêtes Top-k, fonction monotone, seuil, phase, listes triées.

KEYWORDS: aggregation, Top-k queries, monotonic function, threshold, phase, sorted lists.

1. Introduction

Les systèmes actuels tels que le web, les systèmes pair-à-pair de partage de fichiers, les systèmes de recommandation comme del.icio.us, les bases de données et les entrepôts de données sont larges et détiennent des quantités de données immenses. L'acquisition de données nécessite d'introduire des critères de sélection très performants afin de ne pas surcharger les serveurs lors de la localisation et de ne pas submerger les utilisateurs par des réponses inutiles. Le traitement des requêtes de classement Top-K par agrégation est un moyen efficace pour venir à bout de ces problèmes. L'agrégation épargne le serveur de visiter ses pairs et faire les calculs seul, alors que considérer uniquement des meilleurs objets réduit la quantité de données échangées. Les scores des réponses sont calculés selon une fonction monotone sur les valeurs [4], comme la moyenne, le max et la somme.

Dans cet article, nous présentons une version améliorée de notre algorithme TPOT (Three Phases and One Threshold) qui améliore le seuil de notre algorithme ZA[11]. Nous définissons un nouveau seuil qui rend l'exécution plus rapide et nous donnons de nouveaux résultats. A notre connaissance, c'est le seul algorithme qui ne définit qu'un seul seuil. Nous déduisons dans une seule phase (la deuxième) l'ensemble exact des objets Top-k. La troisième phase consiste à chercher l'ordre exact au sein de cet ensemble. Plus encore et à la différence des autres algorithmes, nous calculons les Top-K valeurs, car il est possible que deux objets ou plus puissent avoir la même valeur. Afin que les autres algorithmes adoptent le Top-k valeurs, il faut qu'il ajoute plus d'opérations. Nous comparons notre solution avec d'autres très connues dans le domaine et nous montrons par le biais de simulations sur des données réelles que notre solution est compétitive. Nous commençons dans la section 2 par la définition du problème. Dans la section 3, nous donnons les travaux liés à ce domaine. Nous revoyons l'algorithme TPUT[2] qui est une référence incontournable. La section 5 présente notre algorithme. La section 6 contient les résultats de l'expérimentation. La section 7 conclut ce papier.

2. Définition du problème.

Soient L_1, L_2, …, L_m m listes triées par ordre décroissant de couples $(o_j, v(o_j))$, où o_j est un objet et $v(o_j)$ est une valeur non négative. Tout objet o_j n'apparait qu'une seule fois au plus dans une liste L_i. L'ensemble de ces listes est appelé base de données [4]. D'un point de vue réseau, nous supposons qu'il existe un central manager CM auquel sont liés m pairs et chaque pair P_i manipule une et une seule liste L_i.

La résolution exacte d'une requête Top-K par agrégation, dans cet article, revient à trouver l'ensemble des objets O dont le cardinal est égal à K et telle que la valeur totale

de chaque objet o_j de O est supérieure à la valeur totale de tout objet o_l n'appartenant pas à O. Le tableau 1 suivant présente un exemple de listes triées qui maintiennent des objets et leurs valeurs. Nous pouvons voir que V(o1)=15+2+9=26; V(o6)=7+4+9=20; V(o3)=9+10=19 ; Pour K=3, Top-3 est dans l'ordre o_1, o_6, o_3.

	Pair 1	**Pair 2**	**Pair 3**
1	<O_1, 15>	<O_2, 13>	<O_3, 10>
2	<O_3, 9>	<O_4, 9>	<O_1, 9>
3	<O_6, 7>	<O_6, 4>	<O_6, 9>
4	<O_2, 1>	<O_1, 2>	<O_5, 7>

Tableau 1. *Exemple de listes*

3. Travaux liés

Dans la littérature, il existe deux catégories d'algorithmes pour la résolution des requêtes Top-K. La première catégorie comporte des algorithmes qui retournent des solutions approximatives et probabilistes. Ces algorithmes se justifient par le fait que les systèmes sont dynamiques et par suite, il n'est pas nécessaire de retourner un calcul exact. [5,6] sont des exemples de tels algorithmes.

La deuxième catégorie consiste à retourner un calcul exact des K objets respectant la requête. Cette même catégorie se divise en deux parties. La première contient les algorithmes dont le nombre de phases n'est pas borné, c'est-à-dire l'exécution s'arrête lorsque les premiers k objets sont détectés. Dans chaque phase, un seuil est défini et si la valeur d'un objet est supérieure à ce seuil alors l'objet sera sélectionné et si aucun objet n'est sélectionné alors un nouveau seuil sera défini. Il résulte qu'il s'agit de seuils adaptatifs. De part leur logique, il est fort possible que le Central Manager touche à un nombre important d'objets différents avant d'atteindre la solution. On rencontre dans cette partie l'algorithme TA[4] de Fagin et al. Celui-ci est non borné et requiert un nombre important de tours pour le calcul dans chaque phase. Le nombre de messages générés est aussi important. Les Best Position Algorithms [11] qui, en principe, s'arrêtent plutôt que TA; cependant, les auteurs ne les ont pas comparés aux algorithmes efficaces déjà existants comme TPUT. [4] comporte de nouvelles versions plus généralisées pour TA et Non-Random Algorithm (NRA)[3]. Jing et al ont observé des insuffisances dans NRA et ont produit SNRA [4] pour minimiser les accès aux objets.

Dans la deuxième partie, il s'agit des algorithmes fixes en nombre d'étapes. TPUT [2] en est la référence. Cet algorithme définit deux seuils pour l'envoi des objets candidats. Son problème est que si le seuil est très petit alors les pairs vont envoyer des listes importantes aux CM. Plus encore, il exécute des opérations supplémentaires pour atteindre les *limites supérieures*. La nature des bases de données affecte aussi TPUT;

TA a été plus optimal que lui pour le dataset WorldCup98. L'algorithme Three-Phase Object Ranking (TPOR) [9] s'est présenté comme un algorithme qui corrige ce problème de seuillage de TPUT. Les concepteurs proposent que le CM retourne une liste d'objets candidats aux pairs qui, à leurs tours, vont les comparer localement, et retourneront au CM leurs listes résultats par la suite. Ici, le problème est plus important que dans TPUT, car les pairs qui ne partagent pas d'objets avec la liste du CM, alors ils déduisent que le seuil est nul et, par conséquent, ils vont retourner toutes leurs listes au CM. Plus encore, les résultats de la simulation effectuée par les concepteurs eux-mêmes montrent clairement que TPUT est meilleur. Afin de remédier aux inconvénients de TPUT et TPOR, les concepteurs de ce dernier ont combiné ces deux algorithmes pour concevoir Hybrid-Threshold Algorithm (HT). Celui-ci s'exécute pendant 3 à 4 phases. La troisième phase s'appelle *patch phase* qui s'exécute si certaines conditions ne sont pas vérifiées. Cette phase est une manière de rattraper les objets non considérés pendant la deuxième phase. L'algorithme HT est une mutation de TPUT et exécute aussi le calcul des limites supérieures, ce qui augmente le nombre d'opérations.

4. Revue de l'algorithme TPUT

L'algorithme TPUT s'exécute en trois phases. Dans la première phase, le CM demande aux pairs de lui retourner leurs k premiers couples (objet, valeur). Le CM calcule les sommes partielles et les trie puis il prend la $k^{ième}$ valeur et la divise par *m* comme étant le seuil de cette phase. Ensuite, il élague les objets dont les valeurs sont inférieures à ce seuil. Dans la deuxième phase, le CM envoie le seuil aux pairs et leur demande de lui retourner les objets dont les valeurs ne sont pas inférieures à ce seuil. Les pairs répondent et le CM calcule les nouvelles sommes partielles et les trie. Il prend la $k^{ième}$ valeur et la divise par *m*. Ceci définira le nouveau seuil. Maintenant le CM augmente la candidature des objets en calculant les limites supérieures. Il s'agit de recalculer les sommes partielles en prenant la valeur du seuil pour les objets non retournés au CM. Puis il élague les objets dont les limites supérieures sont inférieures au deuxième seuil. L'ensemble d'objets ainsi déduit est appelé S. Dans la troisième phase, le CM demande aux pairs de lui compléter la liste S. Une fois qu'il reçoive ces objets, le CM calcule les sommes totales, les trie et prend les k premiers objets.

Malgré son efficacité, TPUT souffre de certains problèmes parmi lesquels : si le seuil est petit alors les pairs retourneront des quantités énormes d'objets. Le calcul des limites supérieures est un calcul supplémentaire qui élargit l'ensemble des objets candidats. Il s'agit ici de forcer les valeurs des objets alors qu'il est sûr que certains seront élagués dans la phase suivante, cet ensemble S est maximal. Donc cet algorithme élague les objets pendant toutes ses étapes. Plus encore et selon les auteurs eux-mêmes [2], TPUT peut aller jusqu'à ré-exécuter la deuxième phase à cause des cas pathologiques où les pairs ne peuvent pas envoyer de grandes quantités de donnés.

5. L'algorithme TPOT

Notre algorithme s'exécute en trois phases sur une architecture centralisée telle que définie dans la section 2. Les étapes de déroulement de TPOT sont les suivantes :

– **Phase1 : calcul du seuil**

- Le CM demande à tous les *m* pairs de lui retourner leurs $K^{ièmes}$ valeurs. Ayant reçu ces valeurs, le CM calcule le Seuil par l'équation (1) suivante :

$$\text{Seuil} = \left(\frac{Moyenne\ des\ Kièmes\ valeurs\ reçués}{15\%m}\right) \quad (1)$$

–**Phase 2 : réception des listes et calcul des sommes partielles**

- Le CM envoie Seuil à tous les pairs et leur demande de lui retourner leurs couples (objet, valeur) avec valeur≥ Seuil,
- Le CM calcule les sommes partielles de la façon suivante : Soit Oj un objet envoyé par les pairs et soit Vi(Oj) la valeur de Oj que le pair Pi maintient pour cet objet. La somme partielle SPOj est donnée par l'équation (2) suivante :

$$\text{SPOj} = \sum_{i=1}^{m} Vi(Oj) \quad (2)$$

- Le CM Trie les sommes partielles par ordre décroissant et met les K premiers couples (Oj, SPOj) selon leurs sommes partielles dans une liste LP.

–**Phase 3 : Finalisation**

- Le CM demande aux pairs de lui compléter LP par les couples (objet, valeur) qu'ils n'avaient pas envoyés dans la phase précédente,
- Le CM ajoute ces valeurs reçues aux sommes partielles: ce sont donc les valeurs totales VTOj (Valeur Totale de Oj) et trie les listes (Oj, VTOj), selon VTO_j
- Le CM déduit les K premiers couples (Oj, VTOj) comme Top-K réponses.

15%m se justifie par les statistiques qui ont montré que la couche supérieure des systèmes p2p hybrides détient *15%* de la taille totale du réseau [1]. Dans la version hybride de notre solution, nous avons déduit que ce seuil donne les meilleurs.

L'étude de la complexité est très difficile. Dans [2], on a fait des approximations en considérant une certaine fonction C(m) croissante sans rentrer dans les détails, quoiqu'ils préfèrent une fonction Zipfienne. Nous donnons ici les nombres d'accès aux objets des algorithmes TPOT et TPUT ainsi que les nombres d'opérations qu'ils effectuent. Dans la partie expérimentation, nous avions comparé les algorithmes selon les nombres d'opérations en utilisant le 'profiling java'. Le tableau suivant donne une comparaison les deux algorithmes.

Algorithmes	Nombre d'accès aux objets		Nombre d'opérations
TPUT	Phase 1	*mk*	*mk*
	Phase 2	*ml*	*2ml*
	Phase 3	$m \mid S \mid$	$m \mid S \mid$
TPOT	Phase 1	*m*	*m*

	Phase 2	*ml'*	*ml'*
	Phase 3	$m\|LP\|$	$m\|LP\|$

Tableau 2. *Comparaison entre TPUT et TPOT*

En considérant le QuickSort pour tous les algorithmes, TPUT exécute un tri par étape. D'où, le coût du tri, selon [10], est $C(Tri\text{-}TPUT)=2m(kLn(mk)+l\ Ln(ml)+|S|\ Ln(m|S|))$. Alors que TPOT exécute deux tris, dont le coût est $C(Tri\text{-}TPOT)=2(ml'Ln(ml')+|LP|\ Ln(|LP|))$. Nous montrerons par des expérimentations qu'en général $|LP|=k$. Pendant la deuxième phase, TPUT exécute des tests pour déduire l'ensemble S sur les objets qu'il détient. Le coût de tests devient alors $C(Test\text{-}TPUT)=ml$. En totalité, TPUT touche à $m(k+l+|S|)$ objets et exécute $m(k+2l+|S|)+C(Tri\text{-}TPUT)+C(Test)$ opérations. Alors que TPOT touche à $m(1+l'+|LP|)$ objets et exécute $m(1+l'+|LP|)+C(Tri\text{-}TPOT)$ opérations.

6. Expérimentation

Données. Pour étudier les performances de notre algorithme, nous avons pris 15629 objets de la 51ième journée du dataset WorldCup98_51. Le dataset WorldCup98 a été créé en 1998. Il contient toutes les 1,352,804,107 requêtes qui ont été soumises à 4 régions (Paris en France; Plano au Texas ; Herndon en Virginie, et Santa Clara en Californie) qui contrôlent 29 serveurs dans le monde entre le 30/04/ et le 26/07/1998.

Réseaux. Nous avons créé un réseau centralisé auquel sont liés les m=29 pairs.

Métriques. Le tableau 2 suivant donne les métriques que nous avons considérées.

Métrique	Description
Durée	Le temps écoulé pour trouver la solution en millisecondes
Nbr opérations	Le nombre d'opérations véritablement effectué en millions
Nbr objets manipulés	Le nombre global de fois où on a touché à un objet

Tableau 2. *Métriques utilisées*

La figure 1 montre que TPOR a pris beaucoup de temps car, dans sa logique, lorsque le CM calcule une liste candidate, il l'enverra aux autres pairs et lorsqu'un pair reçoit cette liste et s'il ne partage pas d'objet avec elle alors il retournera au CM tous ses objets. De cette manière, le CM passera des temps d'attente énormes. Nous remarquons qu'à partir de k=20, TPUT est plus rapide que HT ; il est sûr que HT a exécuté la patch phase qui le rend plus lent que TPUT, alors que TPOT a été le plus rapide, car le temps d'exécution est très liés aux nombres d'opérations en premier lieu et aux nombres d'accès aux objets en second lieu. La figure 2 donne la comparaison en nombres d'opérations. Sur la figure 2a, on remarque que TPOR et HT ont exécuté des nombres opérations élevés. Il est sûr que HT a exécuté la patch phase, ce qui augmente le nombre d'opérations. HT pourrait être meilleur que TPUT s'il n'exécute pas la patch phase. La comparaison de listes sans introduction de seuil alourdit l'exécution de TPOR. La figure

2b donne la comparaison entre TPOT et TPUT. Ce dernier est pénalisé par le calcul des limites supérieures. Sur la figure3, nous remarquons que dans l'intervalle [21,36], HT a touché à moins d'objets que les autres algorithmes, alors que pour le reste des valeurs de k, TPOT a touché à moins d'objets.

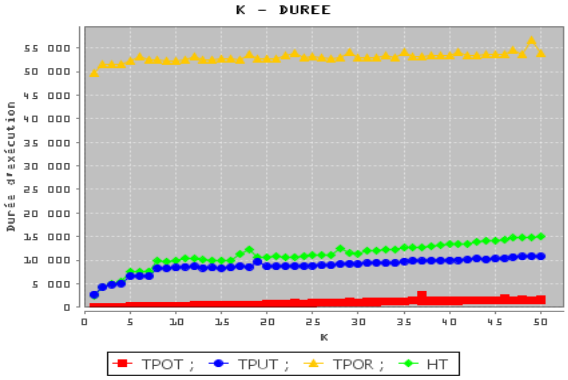

Figure 1. *Temps d'exécution effectués par les algorithmes*

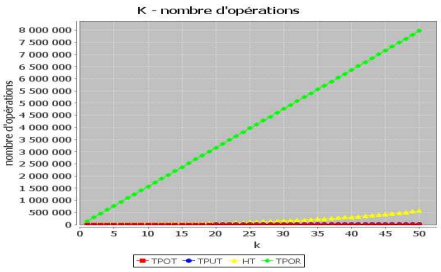

Figure2a- *Nombre d'opérations effectuées*

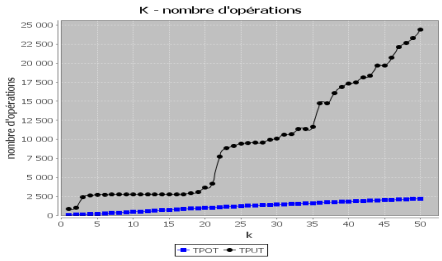

Figure 2b. *Un zoom sur la figure 2a*

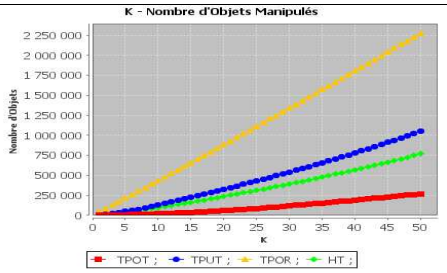

Figure 3.*Nombres d'objets manipulés*

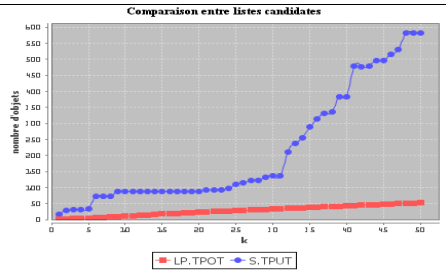

Figure 4. *Comparaison entre S et LP*

Nous avons comparé les deux listes candidates LP de TPOT et S de TPUT. Sur la figure 4, nous remarquons que le cardinal LP est de l'ordre de k alors que le cardinal de

S est très élevé. Pour k= 50, TPUT demande aux pairs de lui compléter une liste de 582 objets et éliminera 532 objets après réception de ces objets et calcul des valeurs totales.

7. Conclusion

La résolution des requêtes Top-K est un domaine très intéressant et comporte des défis considérables à relever. La réalisation d'une solution exige de rendre la réponse avec des coûts minimaux en matière de communication et d'accès aux objets. Dans cet article, nous avons présenté un nouvel algorithme basé sur trois phases, et à notre connaissance c'est le seul algorithme qui ne définit qu'un seul seuil. Nous déduisons très tôt la liste Top-K. Enfin, nous avons montré par le biais d'expérimentations, sur les mêmes datasets réels utilisés dans les autres travaux, que notre solution est très performante. Actuellement, nous étendons TPOT pour le calcul des skylines.

8. Bibliographie et biographie

[1] Boukhatemi, AEK, (2009). Crawling des Systèmes P2P pour la Détection de Structures et de Communautés. Mémoire de fin d'études d'Ingéniorat, Département d'informatique, université d'Oran, Es-sénia. Code : 16/2009.

[2] Cao, P., Wang, Z. (2004). Efficient Top-k Query Calculation in Distributed Networks. In Proceedings of the 23rd PODC

[3] Fagin R. Lotem, A. Naor, M.(2001). Optimal aggregation algorithms for middleware. In Symposium on Principles of Database Systems.

[4] Ge, T. Zdonik, P. Madden, S. (2009). Top-k Queries on Uncertain Data: On Score Distribution and Typical Answers, SIGMOD'09, Providence, RI, USA.

[5] Haghani, P. Michel, S. Aberer, K. (2009). Evaluating Top-k Queries over Incomplete Data Streams, CIKM'09, November 2–6, Hong Kong, China.

[6] Jing, Y., Guang-Zhong, S., Ye, T., Guoliang, C., Zhi, L., Selective-NRA Algorithms for Top-k Queries, APWeb/WAIM 2009, Suzhou, China

[7] Kumar, R. Punera, K. Suel, T. Vassilvitskii, S.(2009). Top-k Aggregation Using Intersections of Ranked Inputs, WSDM'09, February 9–12, Barcelona, Spain

[8] Reza,A(a). Esther,P.P. Patrick,V.(2007). Best Position Algorithms for Top-k Queries, VLDB'07, September 23-28, Vienna, Austria

[9]Yu, H. Li, H. Wu, P. Agrawal, D., Abbadi, A. E(2005). Efficient Processing of Distributed Top-k Queries. In Proceedings of the 16th DEXA

[10] Robert Sadgewick, Algorithms, Addison Wesley 1983,

[11] Zekri, L. Allal, L. Megague K, Timsit, H, Un nouvel algorithme pour le calcul des requêtes top-k par agrégation. EGC-M 2011, Tanger Maroc