Solution compacte aux requêtes d'appartenance sur des flux de données

Yann Busnel, Noël Gillet

Solution compacte aux requêtes d'appartenance sur des flux de données †

Yann Busnel et Noël Gillet

IMT Atlantique, IRISA, UMR CNRS 6074, F-35700 Rennes, France

Le test d'appartenance est fondamental en informatique. Plus formellement, si on considère un sous-ensemble d'éléments appartenant à un univers, nous voulons pouvoir répondre à la question « étant donné tout élément de l'univers, celui-ci appartient-il au sous-ensemble en question ? ». Interroger un oracle parfait est très coûteux en pratique (temps de recherche long, latence importante, *etc.*), il est ainsi intéressant d'interroger dans un premier temps une structure compacte, comme le classique filtre de Bloom, qui représente le sous-ensemble considéré. L'oracle ne sera ainsi interrogé que dans certains cas prédéfinis. Dans cet article, nous proposons une amélioration de cette méthode dans le cadre de l'analyse de flux de données à grande échelle. L'idée consiste à construire ledit filtre en temps réel, en y stockant uniquement les éléments du flux qui appartiennent au sous-ensemble. Pour cela, l'oracle n'est interrogé que si un élément est vu pour la première fois, afin de déterminer son appartenance à l'ensemble. Nous présentons des garanties théoriques ainsi qu'un ensemble de simulation justifiant l'intérêt de notre travail.

Mots-clefs : flux de données, requête d'appartenance, filtre de Bloom

1 Introduction

Pouvoir déterminer si un élément appartient à un ensemble donné est une fonctionnalité très utile en informatique. Dans les réseaux de télécommunication, il est fréquent d'avoir à implémenter des listes noires d'adresses IP interdites, afin de sécuriser l'accès à un serveur. Dans le domaine des bases de données, déterminer si un élément est enregistré dans la base, ou non, est également une opération essentielle.

Plus généralement, une requête d'appartenance consiste à déterminer si un élément u, issu d'un univers $\mathcal{U}$, appartient ou non à un sous-ensemble $\mathcal{B} \subseteq \mathcal{U}$. Supposons que nous avons accès à un oracle, noté $O_{\mathcal{B}}$, permettant de répondre à cette requête. Une solution simple consiste alors à interroger $O_{\mathcal{B}}$ dès que nécessaire. Cependant utiliser un tel oracle a, en pratique, un coût. En effet, si $\mathcal{B}$ contient beaucoup d'éléments alors le coût d'interrogation peut se révéler prohibitif. De plus, si l'oracle est accessible via un réseau, le temps de latence peut être également très élevé en fonction de son éloignement par rapport à l'initiateur de la requête. Pour toutes ces raisons, il est souhaitable d'éviter d'utiliser $O_{\mathcal{B}}$ dans la mesure du possible. Une solution classique consiste à *résumer* l'ensemble $\mathcal{B}$ dans une structure de données locale, de taille réduite, mais capable de répondre aux requêtes d'appartenance, avec un taux d'erreur contrôlé. L'idée est donc d'utiliser prioritairement une telle structure et de n'interroger $O_{\mathcal{B}}$ que dans certains cas.

Dans cet article, nous nous intéressons à l'utilisation de résumés pour répondre aux requêtes d'appartenance dans le cadre de l'analyse de *flux de données* en temps réel. Intuitivement, nous considérons un flux de données $\sigma = s_1, s_2, \ldots$ où chaque élément s_i est choisi parmi l'univers $\mathcal{U}$ et nous voulons déterminer pour chacun d'eux s'il appartient où non à un sous-ensemble $\mathcal{B}$. Une solution bien connue consiste à utiliser une structure de données appelée *Filtre de Bloom* [1] (noté BF par la suite) qui garantit l'exactitude de la réponse si elle est négative mais peut introduire des *faux positifs*. Si un résultat exact est requis, il est toujours possible d'interroger $O_{\mathcal{B}}$, en cas de réponse positive. L'enjeu consiste à améliorer l'exactitude de cette structure afin de limiter le coût de communication. De nombreux travaux ont été réalisés pour améliorer les BF, par exemple en introduisant des faux négatifs pour diminuer le risque de faux positifs [3, 6] ou encore en adaptant dynamiquement la mémoire allouée au filtre [5]. Toutefois aucun de ces travaux ne prend en

†Ce travail a été partiellement financé par le projet ANR INSHARE (ANR-15-CE19-0024).

compte une propriété fondamentale : plus le nombre d'éléments ajoutés au BF est grand, plus son taux d'erreur sera important. Or, selon la distribution du flux d'entrée, il est possible que très peu d'éléments appartenant à $\mathcal{B}$ soient présents dans celui-ci. Notre proposition consiste donc à ajouter les éléments de $\mathcal{B}$ dans un *BF* à la volée, plutôt que de stocker l'intégralité de $\mathcal{B}$ *a priori*. Ainsi, $O_{\mathcal{B}}$ est interrogé lorsqu'un élément se présente pour la première fois dans le flux et ce dernier est ajouté au filtre s'il appartient à $\mathcal{B}$. Le test d'occurrence est réalisé grâce à un second BF. Pour plus de détails sur ces travaux, le lecteur intéressé pourra consulter l'article associé [2].

2 Le filtre de Bloom en détail

Avant de présenter notre solution, nous rappelons le fonctionnement de notre brique de base : le BF.

Un filtre de Bloom β est une structure de données constituée d'un tableau de bits de taille n noté A_β (initialement tous les bits valent `faux`) et de k fonctions de hachage $h_1, \ldots, h_k$ tel que $h_i(s)$ renvoie de manière aléatoire et uniforme un indice de A_β pour tout élément $s \in \mathcal{U}$. Le filtre fournit également une fonction booléenne $\beta.lookup(s)$ renvoyant `vrai` si et seulement si pour tout $i \in [k], A_\beta[h_i(s)] = \texttt{vrai}$.

Pour représenter l'ensemble $\mathcal{B}$ avec β, il suffit pour tout élément $b \in \mathcal{B}$ et tout indice $i \in [k]$ de changer la valeur de $A_\beta[h_i(b)]$ par `vrai`. Cette opération est appelée un ajout et sera notée $\beta.add(b)$. Ainsi, si $\beta.lookup(s)$ retourne `faux` pour un élément s, nous avons la garantie que s n'appartient pas à $\mathcal{B}$. Dans le cas contraire, il est possible que s n'appartienne pas à $\mathcal{B}$ et soit un faux positif. En effet, l'ensemble des cases $A_\beta[h_i(s')]_{i\in[k]}$ peuvent valoir `vrai`, suite à des collisions avec différents éléments ajoutés précédemment. La probabilité $p(n,k,m)$ de faux positif d'un BF de taille n contenant m éléments et possédant k fonctions de hachage est donnée par la formule $(1 - n_0(m)/n)^k$, où $n_0(m)$ correspond au nombre de bits ayant la valeur `faux` après ajout de m éléments. On remarquera que $n_0(m)$ est une fonction décroissante en m, impliquant que pour n et k fixées, $p(n,k,m)$ augmente avec m.

3 Notre contribution : le filtre de Bloom à ajout dynamique

Notre solution, notée DABF, combine deux BF notés β et β' de taille respectivement n et n' et possédant respectivement k et k' fonctions de hachage. Le premier filtre sera utilisé pour déterminer si un élément a déjà été vu par le passé. Le deuxième quant à lui sera utilisé pour vérifier son appartenance à $\mathcal{B}$. Lorsqu'un nouvel élément s_i du flux apparaît, nous procédons de la manière suivante :

— Si $\beta.lookup(s_i)$ renvoie `faux`

1. appliquer $\beta.add(s_i)$
2. `rep` = $O_{\mathcal{B}}.lookup(s_i)$. Si `rep` vaut `vrai`, appliquer $\beta'.add(s_i)$
3. renvoyer rep

— Sinon, renvoyer $\beta'.lookup(s_i)$

Notons que cet algorithme peut générer occasionnellement un faux négatif. En effet, si le premier filtre β considère avoir déjà vu une occurrence d'un élément $s \in \mathcal{B}$, par la présence d'un faux positif à la première ligne, ce dernier ne sera jamais ajouté à β'. Toutefois, cette contrepartie peut être tolérer selon les cas d'application, comme par exemple pour la distribution de fichier dans des réseaux P2P ou encore dans le domaine médicale pour la détection d'effets indésirables inconnus via l'interrogation d'une base de connaissance pharmacologique. Dans ce dernier cas, un faux négatif impliquerait de surveiller un médicament ayant un effet indésirable déjà connu, ce qui ne constitue pas un problème sanitaire mais uniquement un travail de pharmaco-vigilance inutile. Pour d'autres exemples d'applications tolérant les faux négatifs, le lecteur intéressé peut par exemple se référer à [3]. Notre solution peut également être déclinée dans le contexte d'une *fenêtre glissante*. L'idée est d'utiliser deux DABF (soit quatre BF au total), qui seront alternativement réinitialisés. Chaque filtre sera peuplé avec les éléments de $\mathcal{B}$ comme vu précédemment.

4 Résultats théoriques

On rappelle que $p(n,k,m)$ correspond à la probabilité de faux positif pour un BF de taille n avec k fonctions de hachage et contenant m éléments. Notons $f_p(s)$ (respectivement $f_n(s')$) la probabilité qu'un

élément $s \notin \mathcal{B}$ ($s' \in \mathcal{B}$) soit un faux positif (faux négatif) avec notre structure de données. Enfin, nous notons $S_i(\beta)$ et $S_i(\beta')$ l'ensemble des éléments ajoutés respectivement à β et β' après réception et traitement de s_i. Notre solution permet de faire des compromis entre différentes mesures de performance, à savoir la précision, le coût de communication et la mémoire utilisée. Ce phénomène est parfaitement illustré par notre théorème principal.

Théorème 1. *Soit $s_i \notin \mathcal{B}$ un élément du flux reçu au temps i, alors*

$$f_p(s_i) = \begin{cases} p(n,k,|S_{i-1}(\beta)|)p(n',k',|S_{i-1}(\beta')|) & \textit{si } s_i \textit{ apparaît pour la première fois.} \\ p(n',k',|S_{i-1}(\beta')|) & \textit{sinon.} \end{cases} \quad (1)$$

Soit $s_i \in \mathcal{B}$ un élément du flux reçu au temps i, alors $f_n(s_i) = p(n,k,|S_{i-1}(\beta)|)(1-p(n',k',|S_{i-1}(\beta')|))$ si s_i apparaît pour la première fois. Sinon on a $f_n(s_i) \leq f_n(s_j)$ où s_j est la première occurrence de s_i.

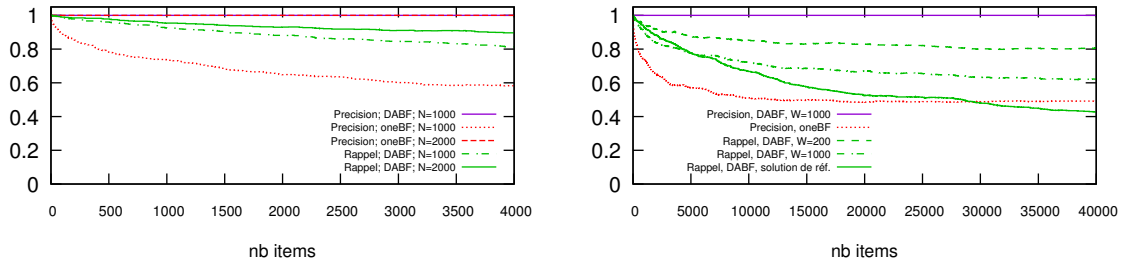
Les probabilités f_p et f_n sont inversement proportionnelles à la valeur de n. En effet par définition des BF la probabilité $p(n,k,m)$ diminue avec n (pour k et m fixées). Augmenter la valeur de n' va améliorer f_p, mais va détériorer f_n (et inversement). Enfin, le coût de communication est fortement corrélé avec la précision de β. Plus précisément, nous prouvons que le nombre de communication après avoir reçu i éléments du flux est borné inférieurement par $d_i\left(1-(kd_i/n)^k\right)$, où d_i correspond au nombre d'éléments distincts reçus. Ici $d_i\,(kd_i/n)^k$ est une borne supérieure du nombre de faux positifs produits par β.

5 Expérimentations

Nous avons réalisé des simulations afin de valider expérimentalement notre solution. Nous nous intéressons particulièrement à déterminer la précision et le rappel au cours du temps. Intuitivement, la précision permet de mesurer la proportion de réponses positives exactes parmi les réponses positives (qualité des réponses). Le rappel quand à lui permet de mesurer l'exhaustivité (*i.e.*, la proportion de réponses positives exactes parmi l'ensemble des éléments de σ appartenant à $\mathcal{B}$). Formellement, notons σ_i le sous-flux formé par les i premiers éléments du flux, $\mathcal{D}_i$ les éléments distincts de σ_i et $\mathcal{P}_i$ les éléments distincts de σ_i pour lesquels DABF renvoie au moins une fois `vrai`. La précision est donnée par $\frac{|\mathcal{P}_i \cap \mathcal{B}|}{|\mathcal{P}_i|}$ et le rappel par $\frac{|\mathcal{P}_i \cap \mathcal{B}|}{|\mathcal{D}_i \cap \mathcal{B}|}$. Nous nous intéressons aussi au taux de faux positifs (resp. de faux négatifs) défini comme le rapport entre le nombre d'éléments (pas nécessairement distincts) du flux qui sont des faux positifs (resp. faux négatifs) et le nombre d'éléments (pas forcément distincts) du flux qui n'appartiennent pas (resp. appartiennent) à $\mathcal{B}$.

Protocole expérimental L'univers $\mathcal{U}$ est représenté par l'ensemble des entiers $[O, U-1]$, avec $U = 10^4$. Nous construisons la base $\mathcal{B}$ aléatoirement, par un échantillonnage uniforme de $\mathcal{U}$, avec $|\mathcal{B}| \in \{100, 1000\}$. Enfin, 500 flux différents sont générés suivant une loi de Zipf de paramètre $\alpha \in \{0.5, 2.0\}$, notée `zipf(`α`)`. Les résultats présentés ci-dessous correspondent à la moyenne des mesures sur ces 500 flux. Concernant l'implémentation des BF, nous fixons $k = \lfloor \frac{n}{10^3} \ln 2 \rfloor$ et $k' = \lfloor \frac{n'}{|\mathcal{B}|} \ln 2 \rfloor$, en faisant varier la taille des BF selon les scénarios. Nous comparons notre solution avec une solution simple noté `oneBF`, qui consiste à utiliser un seul BF peuplé au départ avec l'intégralité des éléments de $\mathcal{B}$. Soit n le nombre de bits de `oneBF`, alors on lui associe $k = \lfloor \frac{n}{|\mathcal{B}|} \ln 2 \rfloor$ fonctions de hachage, suivant les recommandations de [4]. Nous notons N la mémoire totale utilisée par notre structure, à savoir la taille des deux tableaux de bits de nos filtres.

Impact de la mémoire sur l'exactitude Nous avons tout d'abord déterminé expérimentalement, pour différentes valeurs de $|B|$ et N, que donner plus de mémoire à β' plutôt qu'à β permet d'optimiser la précision. Nous choisissons le ratio $\theta = n/n' = 1/9$, ce qui revient à allouer 9 fois plus de mémoire à β' qu'à β. Nous présentons dans la Figure 1a une expérimentation avec $|\mathcal{B}| = 100$ et des flux de 4000 éléments suivant une distribution `zipf(2.0)` . La précision est environ deux fois meilleure avec DABF par rapport à `oneBF` lorsque $N = 1000$. Il faut doubler la mémoire pour obtenir une précision équivalente. D'autres expériences présentées dans [2] montre des comportements similaires pour $|\mathcal{B}| = 1000$ et/ou `zipf(0.5)` . De plus, le rappel diminue plus lentement lorsque la distribution est hétérogène.



(a) Impact de la mémoire sur le rappel et la précision

(b) Résultats en fenêtre glissante

FIGURE 1: Résultats des simulations

Etude de la version ≪ en fenêtre glissante ≫ Dans cette expérimentation, nous étudions la version en fenêtre glissante décrite en Section 3. Nous nous intéressons au cas $|\mathcal{B}| = 100$, $N = 1\,000$ et à la distribution `zipf(2.0)`. Nous considérons des flux de 40 000 éléments. Les filtres sont réinitialisés par fenêtre de taille W, c'est-à-dire quand W nouveaux éléments ont été reçus. Nous présentons dans la figure 1b les résultats pour $W \in \{200, 1000\}$. Nous nous comparons avec `oneBF` et avec DABF sans fenêtre glissante (solution de référence). La précision est déjà à près de 100% avec $W = 1000$. Logiquement, le rappel est inversement proportionnel à W. Il apparaît également que cette version améliore significativement le rappel par rapport à notre solution de référence pour un flux assez grand. Par manque de place, nous ne présentons pas les résultats concernant le taux de faux positifs/faux négatifs mais il ressort des expérimentations que le taux de faux positif est nul et que le taux de faux négatifs reste stable au cours du temps (*cf.* [2]). Concernant le coût de communication, il est également inversement proportionnel à W. En effet, l'oracle est interrogé plus souvent après une réinitialisation car cela revient à ≪ oublier ≫ d'avoir vu un élément par le passé.

6 Conclusion et perspectives

Nous avons présenté une solution permettant d'améliorer la précision du filtre de Bloom pour répondre à des requêtes d'appartenance à un sous-ensemble $\mathcal{B}$ sur les flux de données. L'idée est d'ajouter à la volée dans le filtre les éléments de $\mathcal{B}$ qui apparaissent dans le flux. Pour cela, un oracle est interrogé uniquement si l'élément est vu pour la première fois. La solution est particulièrement intéressante lorsque la distribution du flux est hétérogène. Une extension naturelle serait d'améliorer le test de première occurence en utilisant d'autres structures de données (par exemple la version dynamique de BF [5]). Il serait également intéressant d'étudier la généralisation de notre solution à toutes structures de type filtre de Bloom. Enfin, une analyse théorique de la version fenêtre glissante permettrait de compléter judicieusement ce travail.

Références

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7) :422–426, 1970.

[2] Yann Busnel and Noël Gillet. A Cost-effective and Lightweight Membership Assessment for Large-scale Data Stream. In *NCA*. IEEE, 2019.

[3] Benoit Donnet, Bruno Baynat, and Timur Friedman. Retouched bloom filters : allowing networked applications to trade off selected false positives against false negatives. In *CoNEXT*. ACM, 2006.

[4] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache : a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3) :281–293, 2000.

[5] Deke Guo, Jie Wu, Honghui Chen, Ye Yuan, and Xueshan Luo. The dynamic bloom filters. *IEEE Trans. Knowl. Data Eng.*, 22(1) :120–133, 2010.

[6] Rafael P. Laufer, Pedro B. Velloso, and Otto Carlos Muniz Bandeira Duarte. A generalized bloom filter to secure distributed network applications. *Computer Networks*, 55(8) :1804–1819, 2011.