Cryptohashing i Filtres de Bloom — GRAU-A

Eric Dacal, Joaquim Marset, Miguel Moreno — G001

Primavera 2017

1 Introducció

En aquest projecte tractarem de veure la probabilitat de fals positiu amb diverses implementacions dels filtres de Bloom amb diferents funcions de hash, tant encriptades amb SHA-256 com sense. La nostra idea era implementar i provar diferents variants dels filtres de Bloom, fer servir diferents criteris per generar les funcions de hash i els seus propis tests.

2 Filtres de Bloom

Recordem que un filtre de Bloom és una estructura de dades probabilística que serveix per comprobar si un element pertany a un conjunt donat. El filtre té com a objectiu ser eficient en espai, a costa de que és possible que el filtre admeti falsos positius, és a dir, elements que poden passar el filtre de forma erronia (en altres paraules, que no es troben al conjunt).[5]

Un esquema del seu funcionament respecte una clau i un conjunt és:

- **La clau passa el filtre** → Aquí tenim dues opcions:
  - **La clau es troba al conjunt** → Òbviament la clau formarà part del conjunt.
  - **La clau no es troba al conjunt** → Aquest element es tracta d'un fals positiu.
- **La clau no passa el filtre** → Llavors tenim garantia de que no el trobarem al conjunt.

Ara explicarem tres implementacions diferents dels filtres de Bloom:

2.1 Bàsica

El filtre bàsic[2] fa servir com a estructura de dades un vector binari de tamany |T|. Al inserir una clau a la taula, s'itera una clau sobre k funcions de hash i assignant-les a 1 (o cert) si el valor de la posició de la funció k_i a la taula no era 0 (on i $\in$ [1,|k|]). En aquest cas per construir les k funcions de hash, utilitzem el mètode de **Double Hashing** on a partir de dues funcions (h1(x), h2(x)) podem calcular-ne k. La fórmula és $hash_{\text{i}}(x) = (h1(x) + i * h2(x))$ mod m, sent m el tamany del filtre. Les dues funcions de hash utilitzades per crear les k s'expliquen més a baix.

La probabilitat d'un fals positiu és:

$$Pr[T[i] = 1] = (1 - e^{\frac{-k*n}{m}})^k,$$

on $T[i]$ és l'element i-èssim de la taula de hash, k és el número de funcions de hash, n el tamany del conjunt, i m el tamany de la taula de hash (|T|= m).

En el nostre cas la versió bàsica del filtre de Bloom es fa amb un simple vector de booleans on el tamany del filtre el variem per fer experimentació. Per tant la creació de la taula que composa el filtre té un cost assimptòtic de O(m) on m és la mida de la taula.

Llavors del fitxer de claus les introduïm una a una al filtre de manera que per cada clau calculem la posició amb k funcions de hash diferents. Per cada funció de hash apliquem el mètode explicat al paràgraf anterior de Double Hashing i obtenim el index que ocuparà al filtre. Això suposa un cost assimptòtic lineal en el nombre de caràcters alfanumèrics que té cada clau, en aquest cas serà un cost de O(d) on d serà el tamany de la clau. Un cop obtingut l'index posem el bit de la posició corresponent a 1, que suposa un cost constant. Això ho repetim per totes les k funcions de hash de cada una de les n claus que inserim al filtre. Llavors passem a fer les comprovacions amb els fitxers de test on la mecànica és la mateixa. Per cada clau i cada funció de hash de les k, obtenim l'index que ocuparia al filtre i mirem si aquella posició té el bit a 1 o no. Si el té a 1 cal mirar si realment era una clau del conjunt inicial, i això ho podem saber perquè guardem les claus en un set per poder fer-ne comprovacions. Per tant igual que abans és un cost O(d) per cada clau i cada funció de hash de calcular la posició que ocupa al filtre i llavors per comprovar si realment és un fals positiu o no recórrer el conjunt en busca de la clau és un cost logarítmic en el nombre de claus que hem inserit inicialment O($logn'$), sent n' el nombre de claus del conjunt que comprovem.

Per tant el cost de construir el filtre, inserir les claus i comprovar les claus és: O(m) de construir el filtre + O($d * k * n$) de obtenir els indexos per el conjunt de claus que inserim + O($k * n$) de posar els bits a 1 + O($d * k * n'$) de obtenir els indexos però per les claus dels tests + O($n' * k * logn'$) de comprovar els falsos positius. Com d i k seran un nombre petit el cost vindrà governat per O(m) + O($n' * logn'$) + O(n).

2.2 Conteig

Abans de res, cal comentar que el filtre funciona sobre un multiconjunt (que a diferència dels conjunts, admet repeticions), amb amplada w. Això es deu a que els filtres de Bloom no tenen en consideració ni les claus duplicades ni l'eliminació de claus sobre el conjunt; si s'elimina una clau sobre el filtre bàsic, s'hauria de recalcular el filtre de Bloom un altre cop!

El filtre de conteig (o *counting*) treballa una llista de conteig per proporcionar al filtre la opció d'esborrar claus sense tenir que refer el filtre un altre cop. Sí s'afegeixen claus, s'incrementa el conteig per aquella clau, i viceversa pels esborrats. Ara bé, esborrar comporta tenir falsos negatius. Un fals negatiu és l'oposat del fals positiu, és a dir, una clau que no passa el filtre de Bloom però que es troba dins del conjunt. La probabilitat de falsos negatius és 2^{-k} per k funcions de hash[1].
Hi han dos problemes associats amb aquest filtre, però:

- **Overflows de comptador**: Si un element s'ha comptat 2^w - 1 vegades, i si es continua comptant pot portar overflows i que $w = 0$; la alternativa seria deixar de comptar, que portaria falsos positius.

- **La** w: una w prou gran pot invalidar els aventatges (i l'objectiu) del poc espai del filtre de Bloom. A més a més, es desaprofitarà molt d'espai en forma de pocs zeros. Però una w petita pot omplir la taula de uns ràpidament! Per això caldrà decidir una w que tingui un tamany ideal per als experiments que es faràn. Amb què el nombre sigui representable amb 4 bits n'hi ha prou[1].

Pel que fa al cost, tenim que serà assimptòticament idèntic al del filtre normal ja que l'únic que varia és que en lloc de ser cada posició un bit és un comptador i que ens permet fer operacions d'eliminar que el filtre bàsic no té pel fet de ser cert o fals. Per tant amb aquestes característiques que té ara al inserir una clau igualment per cada una de les k funcions de hash calculem el index però en lloc de posar el bit corresponent a 1 incrementem el valor. Respecte a comprovar si una clau pertany o no, el mètode és el mateix que amb el filtre bàsic. Si per cada clau, la posició que retorna alguna de les k funcions de hash és 0 sabem que la clau no pertany al conjunt. En canvi si totes les funcions de hash dónen indexos on el comptador és major a 0 que llavors pot ser fals positiu o no. I per comprovar si ho és o no igualment ho mirem en el conjunt de claus que tenim guardat per saber les que hem inserit anteriorment. I les úniques coses noves que afegim és que eliminem certes claus, decrementant el valor del comptador per cada index que retornen les funcions de hash, així veiem l'efecte que suposa en el nombre de falsos positius que tenim. Però assimptòticament el cost no canviarà. I la segona és que calculem el nombre de falsos negatius que podem tenir, que seria la situació totalment oposada a un fals positiu i es fa amb una complexitat de $O(n * k * l)$, sent n el nombre de claus que inserim, k el nombre de funcions de hash i l el tamany d'un vector on tenim els que hem aplicat el remove.

3 Funcions de hash

3.1 Mètode de la divisió

La funció a calcular és relativament simple:

$ClauHash = Clau\%TamanyHashTable$[1].

És una estrategia raonable, a menys que tingui certes propietats indesitjables, com ara que totes les claus apuntin a una mateixa ClauHash. Per exemple, si la taula és de tamany T, i totes les claus són multiples de T, aquest mètode no funcionarà. De fet, és ideal per taules amb tamany igual a un nombre primer.

3.2 Folding additiu

La idea del mètode additiu de *folding*[4]. és que, donat una cadena de caràcters *str* i un numero m que es fa servir com a mòdul, anar dividint la cadena en

$|p| = \frac{|str|}{2}$ paquets formats cadascún per 2 caràcters (els de $str_i + str_{i+1}$, donada una $i : 1 \leq i \leq |p|$), obtenir la suma de tots els paquets $SumPack$, i llavors tenir la clau de hash a partir del mòdul previ; el resultat serà $ClauHash = SumPack \mod m$.

4 Programes

En el nostre projecte, tenim aquests programes:

- `Makefile`: Fitxer per generar els executables de la resta de programes.
- `main*`: Fitxers per provar els programes. Tenim 6 versions:
  - `main`: Fitxer *main* del filtre bàsic sense encriptar.
  - `main_SHA`: Fitxer *main* del filtre bàsic amb encriptació SHA-256.
  - `main_Count`: Fitxer *main* del filtre de conteig sense encriptar.
  - `main_SHA_C`: Fitxer *main* del filtre de conteig amb encriptació SHA-256.
- `Bloom`: Versió genèrica del filtre de Bloom. La resta de versions hereten d'aquesta.
- `basic_Bloom`: Versió bàsica del filtre de Bloom.
- `couting_Bloom`: Versió de compteig del filtre de Bloom.
- `keyGenerator`: Generador de claus aleatories.
- `sha-256`: Implementació lliure de la funció de hash SHA-256.[3]

5 Experiments

En aquest apartat realitzarem una sèrie d'experiments per tal de comprovar el nombre de falsos positius que es produeixen depenen dels paràmetres utilitzats a l'hora de generar un filtre de Bloom. Per a fer-ho hem creat un conjunt de 9 arxius que contenen 500 claus aleatòries amb diferents mides de clau que van des de claus de longitud 2 fins claus de longitud 10. Amb cadascun d'aquests fitxers de claus hem creat 4 subconjunts de cadascun d'aquests conjunts anomenats tests, que contenen un percentatge d'elements de les claus i la resta d'elements aleatoris sumant també 500 claus per fitxer de test. Els percentatges que contenen son 0%, 33%, 66% i 100%.

Hem considerat els següents paràmetres alhora de generar els nostres experiments, la mida del filtre, el nombre de funcions hash utilitzades, el percentatge de claus pertanyents al filtre de Bloom dels tests, si està encriptat o no en SHA256 i el nombre de falsos positius que es produeixen.

Els experiments han consistit en una cerca exhaustiva *(backtracking)* amb tots els parametres dels experiments, partir d'aquest hem anat extraient una

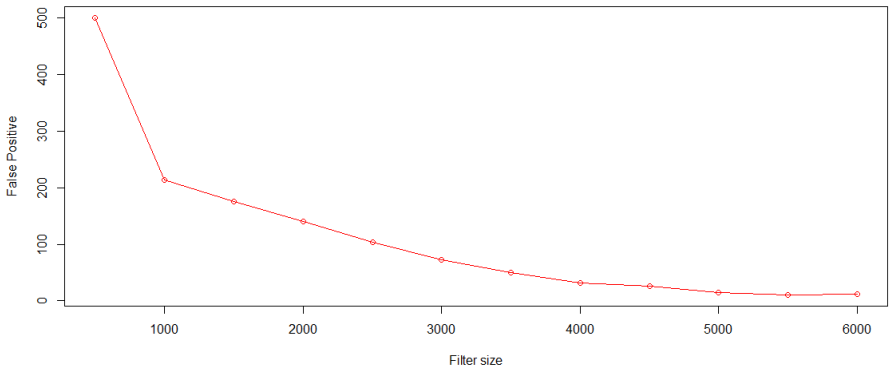

Figure 1: *Plot* del filtre de Bloom bàsic

sèrie de conclusions. Per tal d'observar l'efecte de la mida del filtre de Bloom respecte al nombre de falsos positius hem realitzat amb les dades obtingudes anteriorment el següent plot, el qual relaciona el nombre de falsos positius realitzats en la mitjana de tots els testos amb el tamany del filtre corresponent:

Es pot observar que l'augment de la mida del filtre de Bloom produeix un decrement en el nombre de falsos positius el que resulta lògic, ja que això disminueix la possibilitat que es produeixin col·lisions en les mateixes posicions del filtre.

També hem realitzat el mateix experiment amb la família de funcions hash de codificació SHA256, en aquest cas hem obtingut els següent resultats:

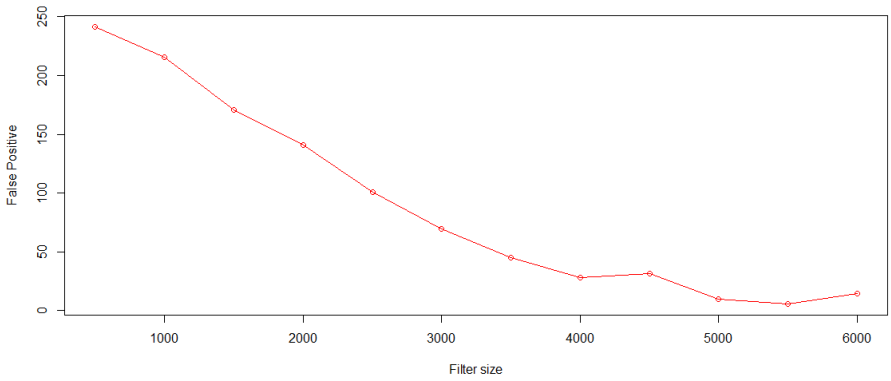

Figure 2: *Plot* del filtre de Bloom bàsic amb SHA-256

Els resultats han estat els mateixos que en el cas anterior tot i que es pot

observar que inicialment la mida del filtre afecta mes en el cas del filtre sense codificar.

Ara anem a veure com afecten el nombre de funcions de hash al nombre de falsos positius, per a fer-ho realitzarem la mitjana del nombre de falsos positius per als diferents testos i per a diferents mides.

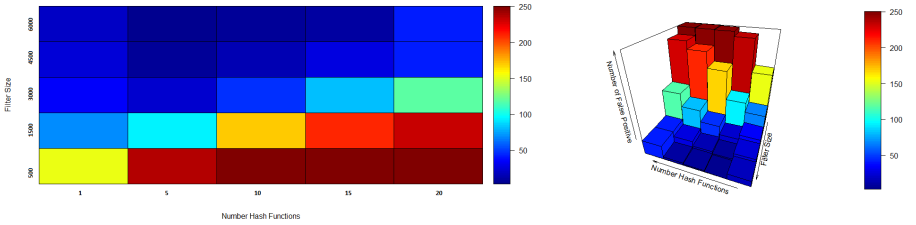

Figure 3: Gràfica relació nombre de falsos positius, mida del filtre de Bloom i nombre de funcions hash

La gràfica prèvia mostra el nombre de falsos positius amb 5 parelles de mides de filtres amb nombre de funcions de hash utilitzades. Podem observar com realment els dos paràmetres afecten significativament al nombre de falsos positius. A mesura que augmentem el tamany de la taula, com tenim més posicions que podem omplir significa que les claus que comprovem si formen part del conjunt o no tenen més posicions a les quals poden anar i llavors el nombre de falsos positius decrementa. Pel que fa a funcions de hash, com més funcions tenim són més les posicions a les que inserim i si el filtre no té el tamany suficient ocuparem un gran nombre de posicions i quan fem tests per comprovar si realment una clau és del conjunt o no obtindrem un gran nombre de falsos positius. Llavors si tenim el tamany suficient de filtre de Bloom a més funcions de hash aconseguirem un nombre menor de falsos positius ja que les claus que haguem inserit no ocuparan totes les posicions i les que probem en els tests tenir més funcions de hash ajudarà a determinar millor si realment està o no al conjunt.

Com inserim 500 claus al filtre, si el tamany del filtre és també de 500 com és obvi a més funcions de hash el nombre de falsos positius arriba al valor més elevat ja que totes s'omplen i al comprovar sempre dirà que el bit val 1 sigui o no del conjunt. A mesura que el tamany del filtre augmenta veiem com realment va disminuïnt el nombre de falsos positius i un nombre de funcions de hash pel voltant de les 10 o 15 dónen els millors resultats.

Amb el filtre de Bloom de conteig té un ritme de falsos positius decreixent en forma quasilogarítmica respecte el tamany del filtre de Bloom. Ara bé, A partir d'un filtre de mida 4.000, però, aquests falsos positius no pateixen una caiguda molt forta, quasi sense pendent, i no torna a caure amb major pendent fins als 4.500, que continua amb el ritme previ. A diferència del filtre bàsic, no comença amb molts falsos positius que cauen fins nivells similars quan la mida del filtre és igual a 1.000 (la realitat és que el bàsic té en una mida de filtre donat el doble de falsos positius que el de conteig a la mateixa mida de filtre), i que a partir d'allà el filtre bàsic és irregular, mentre que el de conteig té un ritme molt més regular en el seu decreixement. A la seva semblança en la seva versió encriptada, però, també té el petit pendent en l'intèrval de mides [4.000..4.500].

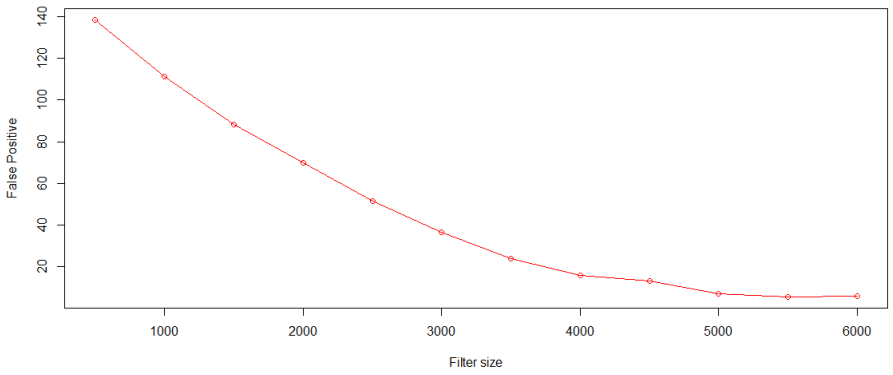

Figure 4: *Plot* del filtre de Bloom de conteig

6 Conclusions

Al fer les proves tant amb un filtre com amb un altre, ens hem trobat següents conclusions, les quals tenen les seves suposicions:

- En termes generals, tots els filtres tenen una proporció de falsos positius inversament proporcional al percentatge, és a dir, que a menor percentatge, major nombre de falsos positius, i viceversa. També començen a disminuir a major tamany del filtre de Bloom.

- Entre els filtres bàsics sense encriptar i amb encriptació SHA-256, tenim que les funcions encriptades solen tenir un major nombre de falsos positius que les que no ho estàn. Una possible raó és que en el procès d'encriptació es poden perdre una mica d'informació, i que a més també pot ser que dues claus que no s'assemblin en res tinguin funcions encriptades prou similars com per acabar en les mateixes posicions d'un filtre, tot i que creiem que aquest cas és força improbable.

- Entre els filtres de conteig sense encriptar i amb encriptació SHA-256, però, no passa el mateix: la diferència entre falsos positius entre funcions no acaba de quedar tan clara. Les úniques coses que podem dir que tenen en comú és que en la gran majoria de casos que no tenen cap positiu també ho són en l'altre, i si en un filtre no és zero, en l'altre hi hauràn molts pocs falsos positius. A més, es conserva totalment la proporció de falsos negatius entre els dos filtres.

- Entre els filtres bàsic i de conteig hi han notables diferències: el fet que el de conteig inclogui els falsos negatius ja n'és una, però a més el nombre de falsos positius és notablement menor en el cas de conteig que no pas en el cas del filtre bàsic. Això pot ser degut a que els esborrats invaliden posicions que ja estan posades per altres claus, i el que podria acabar sent una posició donada es torna en un fals negatiu perquè no la troba tot i saber que es troba allà.

- Dels filtres de conteig cal dir que aquestes tenen els falsos negatius, i a menys que hi hagin claus duplicades (i degut al funcionament del generador de claus, que per naturalesa és aleatori, seràn poques), el número de falsos negatius serà proper al $\%esborrats * \#claus$. No creiem que els nostres experiments funcionin adequadament, ja que, tal i com hem definit abans, un fals negatiu voldria dir que el filtre de Bloom nega un valor que sí existeix, i no és el cas! Com que els valors normalment surten un cop, la idea és que surtin pocs falsos negatius (aquells casos on surti alguna clau repetida).

References

[1] F. Bonomi et. al. "An Improved Construction for Counting Bloom Filters". In: *European Symposium on Algorithms* (2006). DOI: 10.1007/11841036_61.

[2] M. J. Serna et. al. *Data Structures: Hashing-1*. Online. 2017. URL: https://www.cs.upc.edu/~mjserna/docencia/grauA/P17/Hashing.pdf.

[3] *C++ sha256 function*. URL: http://www.zedwood.com/article/cpp-sha256-function.

[4] Judy Hankins. *Hash Tables Notes*. URL: https://cs.mtsu.edu/~jhankins/files/3110/notes/HashTables.pdf.

[5] Matthias Vallentin. *A Garden Variety of Bloom Filters*. URL: http://matthias.vallentin.net/blog/2011/06/a-garden-variety-of-bloom-filters/.