Übung Bloom-Filter dist

Melvin Johner

Hochschule für Technik, Fachhochschule Nordwestschweiz FHNW

November 28, 2020

Idee und Funktionsweise des Bloom-Filters

In diesem Abschnitt werden die Idee und die Vor- bzw. Nachteile des Bloom-Filters erklärt.

Idee Der Bloom-Filter ist eine probabilistische Datenstruktur, die es erlaubt sehr effizient zu prüfen, ob Sie einen Wert bereits enthält oder nicht. Probabilistisch ist die Datenstruktur aus dem Grund, dass die Aussage ob ein Wert enthalten ist nur mit einer gewissen Wahrscheinlichkeit stimmt. Die Werte, welche ein Bloom-Filter "enthält", sind nicht als vollwertige Daten gespeichert. Sie werden mittels mehrerer Anwendungen von Hash-Funktionen auf einem linearen Bit-Vektor abgelegt. Dadurch können Daten, welche im Bloom-Filter abgelegt wurden, nicht mehr aus dem Bloom-Filter rekonstruiert werden.

Werden als Beispiel die Werte $\{A,\ B,\ C\}$ im Bloom-Filter abgelegt, kann der Filter mit Sicherheit sagen ob A, B, oder C im Filter enthalten ist. Wird der Filter nun abgefragt, ob er D enthält wird als Antwort eigentlich $FALSE$ erwartet. Mit einer gewissen Wahrscheinlichkeit gibt der Filter fälschlicherweise $TRUE$ zurück.

Dabei kann die Wahrscheinlichkeit für ein richtiges Resultat in einem Trade-off mit dem Speicherverbrauch und der Laufgeschwindigkeit bestimmt werden.

Vorteile

- Ein kleiner Bloom-Filer kann grosse Datenmengen abbilden
- Alle Operationen sind sehr schnell ($O(k)$ mit k = anzahl Hash-Funktionen)
- Die Fehlerquote ist nach Anwendungsbereich individuell konfigurierbar
- Einfach zu implementieren

Nachteile

- Aus der basic Implementierung lassen sich einmal hinzugefügte Elemente nicht mehr entfernen
- Kann falsche *TRUE* - Werte liefern
- Ist ein Filter erstellt, lässt sich nicht mehr herausfinden, welche Elemente er enthält

Beispielanwendung aus der Praxis

Der Bloom-Filter hat auf Grund seiner Tendenz, auch falsche Resultate zu liefern, ein eher kleinen Einsatzbereich. Hier eine kleine Auswahl von Einsatzzwecken, wobei auf den Einsatz im Google Chrome genauer eingegangen wird:

- Erkennen von schwachen Passwörtern gegen Wörterbuch-Angriffe
- Internet Cache Protokoll
- Erkennen von verdächtigen URLs im Google Chrome

- Bitcoin Wallet synchronisierung
- Viren Scanner

Safe Browsing Google Chrome Goggle führt eine Liste mit verdächtigen bzw. gemeldeten URLs. Diese Webseiten enthalten potentiell unerwünschte Inhalte. Damit nicht jeder Chrome Client bei jedem URL-Aufruf, bei Google anfragen muss, ob die URL "sauber" ist und nicht jeder Client die gesamte URL-Liste bei sich führen muss, wird mit dem Chrome ein Bloom-Filter mitgeliefert. Dieser Filter enthält alle schädlichen URLs. So kann der Client vor jedem URL-Aufruf prüfen, ob die URL potentiell schädlich ist. Sollte der Filter die URL als schädlich markieren, wird die URL zur exakten Prüfung an Google geschickt.
Durch diese Prüfung auf dem Client, kann Google viel Netzwerkverkehr und eigene Rechenleistung einsparen.

Resultate meiner Implementierung

Um meine Implementation zu prüfen, habe ich die zur Verfügung gestellte Lite mit englischen Wörtern (ca. 60k) als Filter-Werte verwendet. Als Testgruppe habe ich eine Wörterliste mit deutschen Wörtern(ca. 680k) verwendet. Aus dieser Testgruppe habe ich mit einer Java-Methode alle Wörter entfernt, welche in der englischen Wörterliste vorkommen.
Würde der Bloom-Filter fehlerfrei arbeiten, sollte er bei keinem der deutschen Wörter ein *TRUE* zurück gegeben. Da er das nicht tut, kann einfach der Anteil an "false-positive", der gesamt Menge an deutsch Wörtern gezählt werden. Ist dieser Anteil kleiner, als der im Bloom-Filter definierte Fehler-Wert und gibt der Bloom-Filter bei allen englischen Wörtern *TRUE* zurück, funktioniert er wie gewünscht.

Diese Tests wurden zusätzlich in einem Unit-Test definiert.

Als Demo wurde im Main ein Filter für erwartete 60'000 Elemente mit einer Fehlertoleranz von $< 1\%$ erstellt. Dieser Filter wurde mit 58'110 englischen Wörtern gefüllt. Danach wurden 683'937 deutsche Wörter gegen den Filter geprüft:

```
"C:\Program Files\ojdkbuild\java-14-openjdk-14.0.2-1\bin\java.exe" ...
Expected number of elements: 60000
Declared false positive tolerance: 0.01
Number of bits in filter vector: 575104
Number of hash functions: 6
Number of elements in filter: 58110

Total checked words: 683937
Total false positive: 6012.0
False positive rate: 0.00879028331556854

Process finished with exit code 0
```

Quellen

- http://matthias.vallentin.net/blog/2011/06/a-garden-variety-of-bloom-filters/
- https://github.com/google/guava/wiki/HashingExplained
- https://iq.opengenus.org/applications-of-bloom-filter/
- https://chromiumcodereview.appspot.com/10896048/