Bloom-Filter

Marco Romanutti[1,2] und Dominik Fringeli[1,2]

[1] *Fachhochschule Nordwestschweiz FHNW, Brugg*
[2] `Diskrete Stochastik, Klasse 3Ia/5Ibb/5iCbb`

Ein Bloom-Filter ist eine Datenstruktur, welche eine schnelle Aussage darüber erlaubt, ob Daten in einer Datenbasis vorhanden sind oder nicht. Grundlage des Filters bilden Hash-Verfahren, welche den Elementen eines Datenstroms möglichst eindeutige Werte zuweisen. Ursprünglich wurden Bloom-Filter zur Rechtschreibkontrolle entwickelt - heute werden sie oft in Datenbanksystemen oder für das Routing in Netzwerken eingesetzt.

1 Funktionsweise

Der Bloom-Filter wird mithilfe eines m-stelligen Arrays und k unterschiedlichen Hashfunktionen umgesetzt. Im m-stelligen Array sind zunächst alle Werte mit Nullen gefüllt. Im folgenden Beispiel besteht ein Filter aus 10 Bits ($m=10$) und 3 Hashfunktionen ($k=3$):

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

Der Filter wird anschliessend mit Werten befüllt. Dazu werden die Hashfunktionen auf jedes Element der Datenbasis angewendet. Jeder daraus resultierende Wert entspricht einer Indexposition. Im m-stelligen Array werden die Werte jener Indexpositionen auf 1 gestellt. Angenommen, die 3 Hashfunktionen liefern die Werte 1, 6 und 7 für das Wort "klar". Dazu werden die Positionen 1, 6 und 7 im Array auf 1 gesetzt:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (2)$$

Falls nun überprüft werden soll, ob sich ein bestimmtes Element in der Datenbasis befindet, müssen dessen Hash-Werte ermittelt werden. Die resultierenden Werte entsprechen wiederum Indexpositionen im Array. Falls an jeder der k Hashfunktionen eine 1 steht, ist das Wort in der Datenbasis vorhanden. Das Wort "zentral" mit den Hashwerten 1, 4 und 9 befindet sich beispielsweise nicht in der Datenbasis. Für jedes weitere Wort, welches wir der Datenbasis hinzufügen, werden weitere Werte im Array auf 1 gestellt. Falls die Vorhandenseinsprüfung eines Worts positiv ausfällt, kann der Fall auftreten, dass das Element trotzdem nicht in der Datenbasis vorhanden ist (sog. "false positive"). In diesem Fall liefert die Prüfung wahr, weil zwar alle Werte der ermittelten Indexpositionen auf 1 stehen - diese jedoch von unterschiedlichen Elementen stammen.

2 Anwendungsbeispiel

Apache Cassandra ist ein verteiltes, skalierbares Datenbanksystem. Es baut auf einer Shared-Nothing-Architektur auf. Bei dieser Architekturform werden die Daten auf verschiedene Datenbank-Nodes in einem zusammengehörigen Datenbank-Cluster verteilt (sog. „Sharding“). Die Zuordnung zu einem Node erfolgt, indem der Schlüsselwert (engl. Partition Key) eines Datensatzes gehashed wird. Bei Apache Cassandra werden standardmässig Murmur3-Hashes eingesetzt. Die möglichen Hash-Werte sind in verschiedene Wertebereiche unterteilt - jeder Node ist für einen Wertebereich zuständig. Die folgende Abbildung 1 stellt das Verteilen der Daten auf den verschiedenen Datenbank-Nodes dar.

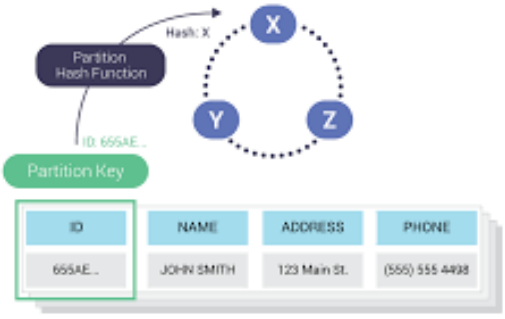

Figure 1: *Sharding in Apache Cassandra*

Bei einem Lesezugriff auf die Daten kann anhand des Wertebereichs ermittelt werden, auf welchen Nodes die Datenfiles (sog. "SSTables") gelesen werden müssen. In Apache Cassandra werden Bloom Filter eingesetzt, um zu ermitteln ob ein Datensatz in einem Datenfile vorhanden ist. Falls der Filter ein positive Ergebnis liefert und der Datensatz dennoch nicht in dem Datenfile vorhanden ist, handelt es sich um ein "false positive". Da Festplattenzugriffe auf die Disk in der Regel kostenintensiver sind als jene im Memory, wird der Bloom-Filter vollständig im Memory umgesetzt.

3 Vor- und Nachteile

Die folgende Tabelle beschreibt den Einsatz von Bloom-Filter gegenüber anderen Lösungsansätzen:

Pros	Cons
[1] Anstelle der eigentlichen Daten werden nur die Hashes der Elemente gespeichert. Dadurch wird weniger Speicherkapazität benötigt.	[1] Ein Bloom-Filter kann erkennen, ob ein Wort *nicht* im Filter vorhanden ist - ob ein Wort allerdings mit Sicherheit im Filter vorkommt kann nicht bestimmt werden.
[2] Das Hinzufügen und Prüfen von Elementen gehört zur Komplexitätsklasse *O(k)*. Weil die verschiedenen Hashverfahren voneinander unabhängig sind, kann die Anwendung ebendieser parallelisiert werden.	[2] Der Bloom-Filter ist einfach umzusetzen, falls Wörter nur hinzugefügt werden. Die eingangs beschriebene Funktionsweise eignet sich allerdings nicht, falls Wörter auch entfernt oder zu einem späteren Zeitpunkt geändert werden müssen.
[3] Durch die Anwendung von Hashfunktionen auf die einzelnen Werte kann der Nachteil von unregelmässig verteilten Daten verringert werden. In verteilten Datenbanken wird durch das Hashing-Verfahren eine gleichmässigere Verteilung der Daten auf den verschiedenen Datenbank-Nodes erreicht (vgl. Kapitel 2).	[3] Wird ein Bloom-Filter erstellt, so werden die damit verbundenen Daten nicht gespeichert. Dies führt zur Einschränkung, dass nicht auf die Daten zugegriffen werden kann, da nur deren Hash-Werte im Bloom-Filter gespeichert sind.

Table 1: *Gegenüberstellung Vorteile und Nachteile*

4 Implementierung

4.1 BloomFilter-Applikation

In der Applikation (Git-Repo: `https://gitlab.fhnw.ch/marco.romanutti/bloom-filter.git`) wurde ein einfacher Bloom-Filter implementiert und getestet. Die Konfiguration kann in der Klasse `BloomFilterTest` vorgenommen werden. Die Anzahl Testwörter wird über das Attribut `wordLimit` gesetzt. Die erwartete Fehlerwahrscheinlichkeit kann direkt beim Aufruf des Konstruktors `BloomFilter(double falsePositiveProbability)` als Argument mitgegeben werden. Um die Funktionsweise des Filters zu überprüfen, können die Tests im Verzeichnis `/src/test/` gestartet werden.

4.2 Vergleich erwartete und effektive Fehlerwahrscheinlichkeit

Im Verzeichnis `/src/test/` befinden sich zwei Test. Der Test `testContainedWords` stellt sicher, dass alle Wörter des Filters auch als möglicherweise im Filter klassifitiziert werden. Der Text `testNotContainedWords` vergleicht die erwarteten und die effektiven Fehlerwahrscheinlichkeiten. Nach Abschluss des Tests wird eine Übersicht ausgegegeben. Die Resultate sind in Tabelle 2 ersichtlich.

Table 2: *Erzielte Testresultate*

Parameter	Test(1)	Test(2)	Test(3)
Testwörter (in Mio.)	1.2	1.2	1.2
m	555888	362329	278494
k	7	5	4
p erwartet	0.01	0.05	0.1
p effektiv	0.0104	0.0504	0.1014

References

[1] Wikipedia: Bloom filter, `https://en.wikipedia.org/wiki/Bloom\_filter`

[2] ScyllaDB: Announcing ring architecture, `https://www.scylladb.com/2017/09/25/announcing-ring-architecture-docs/`

[3] Datastax: How data is distributed across a cluster, `https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archDataDistributeDistribute.html`