Los filtros Bloom y Cuckoo se usan en situaciones similares, pero hay muchas diferencias debajo que generalmente determinan cuál es una mejor opción.

Los filtros Bloom se utilizan internamente en los motores de bases de datos, especialmente Apache Cassandra. Las razones son, como dicen otros carteles, para reducir el costo de las operaciones lentas. Básicamente, cualquier operación de "si esto no existe o definitivamente no existe" con un alto costo puede usar un filtro Bloom para reducir la cantidad de comprobaciones realizadas.

Otro ejemplo común con el modelo SaaS actual sería un servicio REST remoto con un costo por llamada. Cualquier llamada API con una respuesta binaria como "es esta dirección NO VÁLIDA" puede usar un filtro de floración para eliminar más del 90% de las consultas duplicadas. Tenga en cuenta que, dado que los filtros Bloom y Cuckoo tienen falsos positivos, NO son útiles para la operación inversa "es esta dirección VÁLIDA"

Es importante recordar que los filtros Bloom y Cuckoo NO tienen falsos negativos. Esto hace que estos filtros sean útiles para comprobaciones como "definitivamente esto no es así o tal vez es correo no deseado", pero no es útil para operaciones en las que los falsos positivos son inaceptables, como verificar los permisos de los usuarios. En este aspecto, pueden conceptualmente considerarse lo opuesto a un caché. Tanto el filtro Bloom como el cuco se usan principalmente para reducir el costo de operaciones costosas con una respuesta booleana, excepto que las memorias caché no tienen falsos positivos y Bloom/Cuckoo no tiene falsos negativos.

diferencias notables entre cuco/Bloom incluyen:

* combinación. Los filtros Bloom se pueden fusionar eficientemente siempre que se creen con los mismos parámetros. Rápido y con poco ancho de banda. Esta es la razón por la que los ve con frecuencia en sistemas distribuidos masivamente, intercambiar filtros Bloom es rápido. Los filtros de cuco no son fácilmente compostables, lo que los hace menos útiles en estas circunstancias.
* Tasa de falsos positivos. Los filtros Cuckoo son más eficientes en cuanto a espacio. Muchos casos de uso para ambas estructuras se centran en redes de bajo nivel. En hardware débil, la eficacia ~ 40% mayor de los filtros Cuckoo para la misma tasa de falsos positivos puede ser importante. La implementación de referencia, en C++, ordena los elementos dentro de cada segmento para ahorrar espacio adicional, aprovechando la posición de un elemento dentro de un segmento para almacenar huellas dactilares más pequeñas. Las bibliotecas adicionales que mencionaré más adelante (incluida la mía) no parecen hacer esto. Si alguien alguna vez usa mi biblioteca, podría agregarlo.
* Constante tasa de falsos positivos. Los filtros Bloom tienen tasas asintóticamente peores de falsos positivos a medida que superan el tamaño diseñado. Puede seguir insertando elementos para siempre, pero finalmente su tasa de falsos positivos será casi del 100%. Los filtros de cuco, basados ​​en hashing de Cuckoo, tienen una capacidad establecida en la que las inserciones realmente fallarán. La repetición de la inserción de hashes de elementos no aleatorios puede hacer que los filtros Cuckoo fallen su inserción, posiblemente mucho antes de su nivel de llenado diseñado.
* Velocidad. Esto es subjetivo y depende mucho del hardware, pero los filtros Cuckoo generalmente son más rápidos en el caso promedio (según mi experiencia). La mayoría de los diseños de filtro de Bloom ejecutan una función hash dos veces. Al usar funciones hash seguras especialmente, esto puede ser una gran desventaja en comparación con los filtros Cuckoo que solo insertan elementos hash una vez. El código que he visto usa varias funciones de hashing para los filtros Bloom y Cuckoo. Google Guava Bloom utiliza Murmur3, muchas otras implementaciones usan SHA1 u otra cosa. Si las colisiones hash se pueden explotar para su caso de uso, asegúrese de que la biblioteca utilice un hash seguro. Es importante saber que los filtros Bloom tardan aproximadamente un tiempo constante en insertarse, mientras que los filtros Cuckoo tienen un caso PROMEDIO de tiempo constante. A medida que los filtros Cuckoo alcanzan un porcentaje de capacidad, las velocidades de inserción disminuyen considerablemente.Incluso entonces, solo se ralentiza la velocidad de inserción, todas las demás operaciones son tiempo promedio constante.
* Flexibilidad. Los filtros Bloom solo admiten inserción y contienen. Los filtros Cuckoo también son compatibles con la eliminación y el conteo limitado. En el diseño de referencia, los filtros Cuckoo pueden determinar cuántas veces se insertó un artículo, hasta 7 veces. Los filtros Bloom solo pueden determinar si-no. Los filtros Cuckoo también son compatibles con la eliminación de elementos insertados, un gran positivo en muchos casos de uso en comparación con Bloom. Cuando se utilizan filtros Bloom, es bastante normal volver a crear el filtro desde cero cuando está "lleno" (la tasa estimada de falsos positivos excede el umbral) ya que no puede eliminar elementos antiguos. Tenga en cuenta que la reconstrucción del filtro todavía ocurre con los filtros Cuckoo cuando las inserciones comienzan a fallar, por lo que dependiendo del caso de uso, esto podría ser discutible. En ciertas situaciones, los filtros Cuckoo son más útiles ya que puede eliminar elementos para mantenerse dentro de los límites de filtro en lugar de reconstruir.
* Asistencia. Los filtros de cuco son bibliotecas nuevas y estables para muchos idiomas, simplemente no existen.

La mayor ventaja de los filtros Bloom es que tienen un soporte de biblioteca más maduro en la mayoría de los idiomas. La matemática detrás de los filtros Bloom también es mejor entendida por los científicos. La mayoría de las características de los filtros Cuckoo han sido determinadas empíricamente, mientras que los filtros Bloom tienen una base numérica sólida. Esto excluye los filtros Cuckoo para sistemas críticos y en tiempo real que deben tener verificación de su rendimiento, aunque la evidencia experimental muestra que los filtros Cuckoo funcionan mejor en la mayoría de las circunstancias.

Shameless Plug: soy el desarrollador de una biblioteca de filtros Cuckoo para Java. . Falta el semiorcado de cubo utilizado en el documento, por lo que la eficiencia de espacio es algo menor que la implementación de referencia. En el archivo Léame del proyecto, tengo enlaces a otras implementaciones de las que soy consciente. Qué estructura es mejor depende de su caso de uso, pero sobre todo si existe una implementación sólida de filtro Cuckoo para su idioma.

Definitivamente debe echar un vistazo a la fuente antes de utilizar un filtro Cuckoo/Bloom en producción. Leí varias librerías antes de escribir las mías ... muchas de ellas tenían límites de tamaño silenciosos debido a arreglos subyacentes de 32 bits o problemas obvios de rendimiento. La mayoría tenía cero pruebas. La implementación de Google Guava Bloom tuvo la mejor calidad de código y pruebas (y admite límites de matriz de 64 bits). Las únicas deficiencias con Bloom de Guava es que no tiene una opción para usar una función de hash segura y no tiene múltiples subprocesos.

En un sistema de producción, es posible que desee varios subprocesos para la velocidad. La respuesta para Bloom de Guayaba es hacer un filtro diferente para cada hilo y combinarlos ocasionalmente. Como los filtros de Cuckoo no se pueden combinar, agregué un subproceso simultáneo a mi biblioteca de filtros de Cuckoo. El otro que conozco no es seguro para subprocesos o no es concurrente.