CS 7280: DSA for Scalable Computing Spring 2025

Lecture : Adaptive Filters — February 24, 2025

Prof. Prashant Pandey *Scribe: Griffin Ravo*

1 Overview

Recap of Range Filters

Given a range filter F on a dataset S, we want to determine whether any element exists within a given query range $[q_l, q_r]$.

- If there exists at least one element in S within the range $[q_l, q_r]$, then the filter always returns "Yes" with probability:
$$P(\text{Yes} \mid [q_l, q_r] \cap S \neq \emptyset) = 1$$
- If there are no elements in S within the range $[q_l, q_r]$, the filter may still return "Yes" with some false positive probability ϵ:
$$P(\text{Yes} \mid [q_l, q_r] \cap S = \emptyset) = \epsilon$$
- If the filter returns "No", then there are definitely no elements in S within $[q_l, q_r]$, with probability:
$$P(\text{No} \mid [q_l, q_r] \cap S = \emptyset) = 1 - \epsilon$$

To guarantee a maximum false positive rate of ϵ for range queries, the space required by the filter is:

$$O(n \log(R/\epsilon)) \text{ bits}$$

where n is the number of elements in the dataset S and R is the size of the queried range. This is the optimal lower bound.

Today

In this lecture we will cover Adaptive Filters.

2 Adaptive Filters

What is an Adaptive Filter?

An Adaptive Filter is a data structure that "learns" from past queries. Unlike traditional filters, which assume a uniformly random query distribution, adaptive filters dynamically adjust (reducing false positives).

Adaptive Filters satisfy the Strong Adaptivity Property:
For any sequence of n queries, the false positive rate is guaranteed to be bounded by:

$$O(n\epsilon)$$

irrespective of the query workload.

Worst-case space requirement:

$$O(n) \text{ words}$$

This is the lower bound needed to store any filter efficiently.

- $O(n \log(1/\epsilon))$ in-memory storage.
- $O(n)$ disk storage (external memory).

Making Fingerprint Filters Adaptive

2.1 How a Fingerprint Filter Works

1. Insertion: An element x is hashed into a table of size $2^{\log U}$.
2. Storage: Instead of storing x directly, an m-bit fingerprint of x is stored.
3. Querying: When querying an element y, we check if its hashed fingerprint matches an existing one.
4. False Positives: A false positive occurs if $h(x) = h(y)$ but $y \notin S$.

Let's explore how one might make this adaptive.

2.2 Idea 1: Rehash the Fingerprint Table After Every False Positive

An approach to making a fingerprint filter adaptive is to rehash the entire fingerprint table whenever a false positive occurs.

1. A query results in a false positive.
2. The filter rehashes all fingerprints.
3. Our hope is that new hashes will prevent repeated false positives.

Although this approach eliminates the immediate false positive, it performs poorly:

- Every false positive requires rehashing the entire table, leading to an overhead of O(n) for every rehash.

- Even after rehashing, some false positives may still persist.

2.3 Idea 2: Use Caching

We can try to improve an adaptive filter's performance using caching, which helps reduce disk access (saving time). Instead of querying the disk for every lookup, frequently accessed queries are stored in memory.

1. Queries are checked against a small in-memory cache before accessing the disk.
2. If the query exists in the cache, the result is returned instantly.
3. If not, the filter accesses disk storage and updates the cache for future queries.

Each word in the filter is stored in $\lg U$-bit entries. The cache holds $N\epsilon$ elements.

Although caching does reduce disk lookups, it still performs poorly due to some drawbacks:

- The cache only stores a fraction ($N\epsilon$) of total queries.
- Before the cache is populated, all queries must go to disk, leading to slow initial performance.
- As N grows, cache efficiency drops, since there are a growing number of unique queries.

With these considerations, caching is not an ideal solution to making fingerprint filters adaptive.

2.4 Idea 3: Cascading Filters

Another approach is to use cascading filters, where queries are passed through multiple levels of filters before reaching disk storage.

First, let's define Monotonic Adaptivity:

A filter is said to exhibit monotonic adaptivity if it never repeats a false positive and it's false positivity rate *never* increases.

Cascading filter structure consists of multiple filter layers which become increasingly smaller in size. Each layer reduces the numer of false positives. In a cascading filter structure with k layers:

1. Layer 1- The first filter has a total false positive count of:

$$O(n\epsilon).$$

2. Layer 2-

$$O(n\epsilon^2).$$

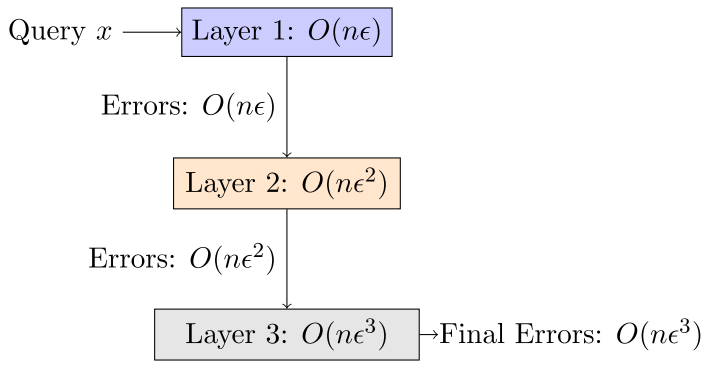
3. Layer 3-

$$O(n\epsilon^3).$$

Following this pattern, the false positive count at layer k is:

$$O(n\epsilon^k).$$

Below is a visualization of this cascading effect:


Query x
Layer 1: $O(n\epsilon)$
Errors: $O(n\epsilon)$
Layer 2: $O(n\epsilon^2)$
Errors: $O(n\epsilon^2)$
Layer 3: $O(n\epsilon^3)$
Final Errors: $O(n\epsilon^3)$

While cascading filters can significantly improve adaptivity and lower latency by resolving most queries before reaching disk, they introduce trade-offs in complexity and memory efficiency.

2.5 Idea 4: Variable-Length Fingerprints

An effective way to make an efficient adaptive filter is to use variable-length fingerprints. Instead of storing fixed-length hashes for all elements, we store only as many bits as needed to maintain an acceptable false positive rate.

By adapting fingerprint lengths, we achieve an optimal space complexity of:

$$O(n \log(1/\epsilon)) \text{ bits.}$$

How Variable-Length Fingerprints Work

1. In-Memory Storage: Stores only short fingerprints for most elements.
2. Filters: Adaptively store longer fingerprints with high false positive risk.
3. Reverse Map on Disk: A lookup table on disk holds full identifiers, using $O(n)$ words.

Querying works as follows:

1. Check In-Memory Filter: If a match occurs, proceed to the next step.
2. Check Adaptive Filters.
3. If necessary, perform a disk lookup using the reverse map.

While variable-length fingerprints provide near-optimal space complexity, they introduce some challenges:

1. Reverse map lookups can be slow
2. Balancing false positives vs. storage $\rightarrow$ Longer fingerprints reduce errors but use more space.

Nevertheless, this method provides the best balance between space efficiency and performance, making it an effective adaptive filter approach.

3 Existing Implementations of Adaptive Filters

- Adaptive Cuckoo Filters (2020): Achieve adaptivity by moving fingerprints or keys within the filter to reduce false positives, dynamically rearranging stored elements. [1]
- Telescoping Filters (2021): Adaptively adjust the number of fingerprint bits used for comparisons, ensuring that only the most relevant n bits are checked, effectively avoiding collisions. [2]
- Adaptive Quotient Filters (2025): Extend quotient filters by using variable-length fingerprints and a reverse map, storing extra metadata to track how many bits each fingerprint requires for accurate queries. [3]

References

[1] M. Mitzenmacher, S. Pontarelli, and P. Reviriego. *Adaptive Cuckoo Filters.* In *ACM Journal of Experimental Algorithmics*, Vol. 25, March 2020.

[2] D. J. Lee, S. McCauley, S. Singh, and M. Stein. *Telescoping Filter: A Practical Adaptive Filter.* In *CoRR*, arXiv preprint, 2021.

[3] R. Wen, H. McCoy, D. Tench, G. Tagliavini, M. A. Bender, A. Conway, M. Farach-Colton, R. Johnson, and P. Pandey. *Adaptive Quotient Filters.* In *Proceedings of the ACM on Management of Data (Proc. ACM Manag. Data)*, Vol. 2, No. 4, September 2024.