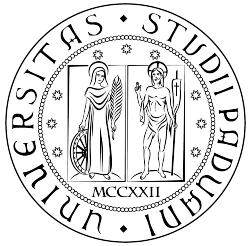


UNIVERSITA’ DEGLI STUDI DI PADOVA
Dipartimento di Ingegneria dell’Informazione

Corso di Laurea in Ingegneria Informatica

Relazione per la prova finale

Studio e analisi prestazionale dei Cuckoo Filter come alternativa efficiente ai Bloom Filter

Relatore: Prof. Francesco Silvestri

Laureando: Andrea Tridente
Matricola n.: 2109932

ANNO ACCADEMICO 2025/2026

Contesto e Obiettivi

Contesto

Test veloci per grandi dataset (Network & Storage):

- *Set Membership Test*.

Criticità

Limiti strutturali dei Bloom filter tradizionali:

- Architettura **intrinsecamente statica**.
- Impossibilità di cancellazioni sicure.

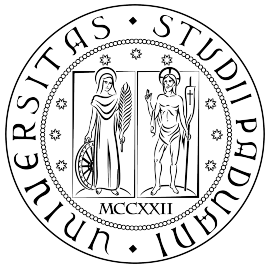
Soluzione Proposta

Adozione del Cuckoo Filter:

- Supporto nativo ad inserimenti e rimozioni.
- Mantenimento **alta efficienza spaziale**.

Obiettivo

- Analisi dei meccanismi operativi.
- Valutazione empirica rispetto ai benchmark di riferimento (*Fan et al.*, CoNEXT 2014).



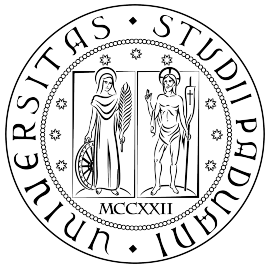
Limiti dei Bloom Filter

Struttura e Funzionamento

- Modello: Array di *m* bit condiviso e *k* funzioni di hash per il *Set Membership*.
- Assenza di falsi negativi: Se un bit è 0, l'elemento non è presente.
- Criticità nativa: Presenza di **falsi positivi** dovuti alla collisione nell'array.

Limiti e Necessità

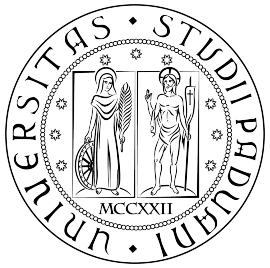
- **Impossibilità di rimozione sicura**: Azzerare un bit condiviso può corrompere altri elementi (**falsi negativi**).
- Compromessi falliti: Varianti esistenti con cancellazione sacrificano l'efficienza.
- Necessità: Passare ad una architettura nativamente dinamica.



Struttura del Cuckoo Filter

Struttura del Cuckoo Filter

- Memorizzazione di *fingerprint:* Sostituzione di array di bit condiviso con slot dedicati che **memorizzano *fingerprint*** compatte.
- *Multi-Way Cuckoo Hashing*: organizzazione della struttura in bucket *multi-entry*.
- ***Partial-Key Cuckoo Hashing***: permette di calcolare la posizione alternativa di un elemento **senza dover risalire alla chiave originaria**.



Cuckoo Filter: Operazioni Fondamentali

Insert

- Calcolo bucket candidati: $\begin{cases} i_1 = \mathrm{hash}(x) \\ i_2 = i_1 \oplus \mathrm{hash}(f) \end{cases}$

  Aspetto chiave è l'uso di $f = \mathrm{fingerprint}(x)$ **al posto della chiave originale**.
- Gestione Collisioni: Spostamento iterativo degli elementi esistenti per fare spazio ai nuovi inserimenti: $i_{\mathrm{alt}} = i \oplus \mathrm{hash}(f)$.

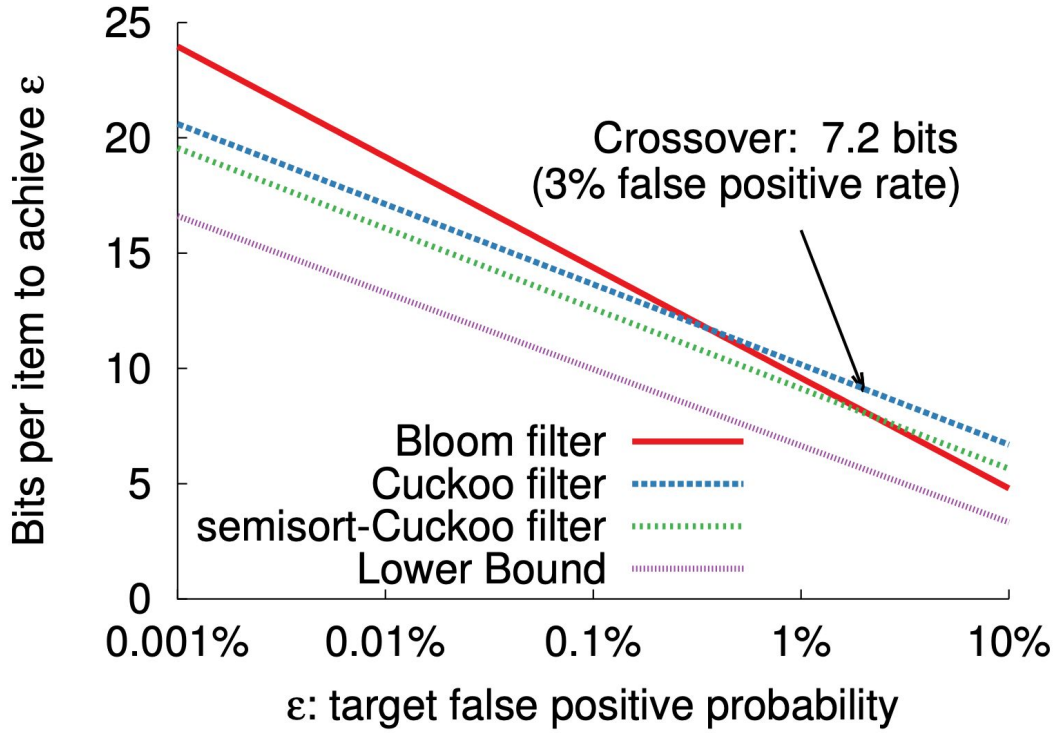
Lookup

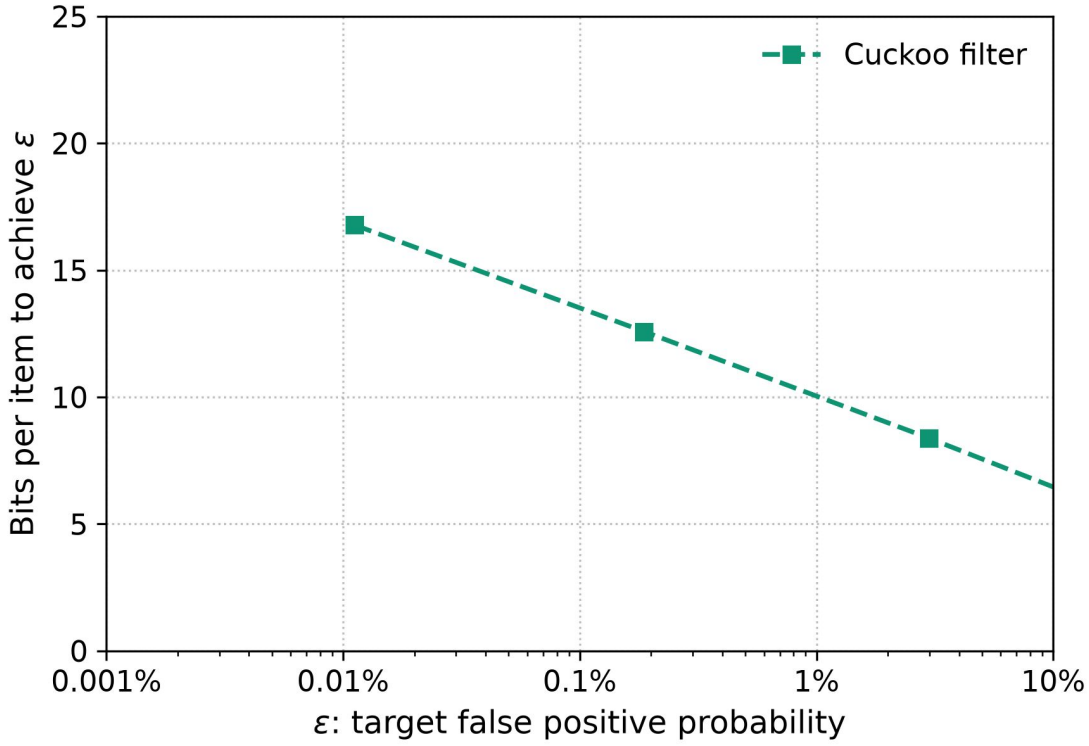
- Verifica della presenza del *fingerprint* nei due bucket calcolati.
- Costo in tempo $\mathcal{O}(1)$.

Delete

- Rimozione nativa e sicura eseguita azzerando direttamente lo slot associato al *fingerprint*.
- **No rischi di falsi negativi** o corruzione strutturale.

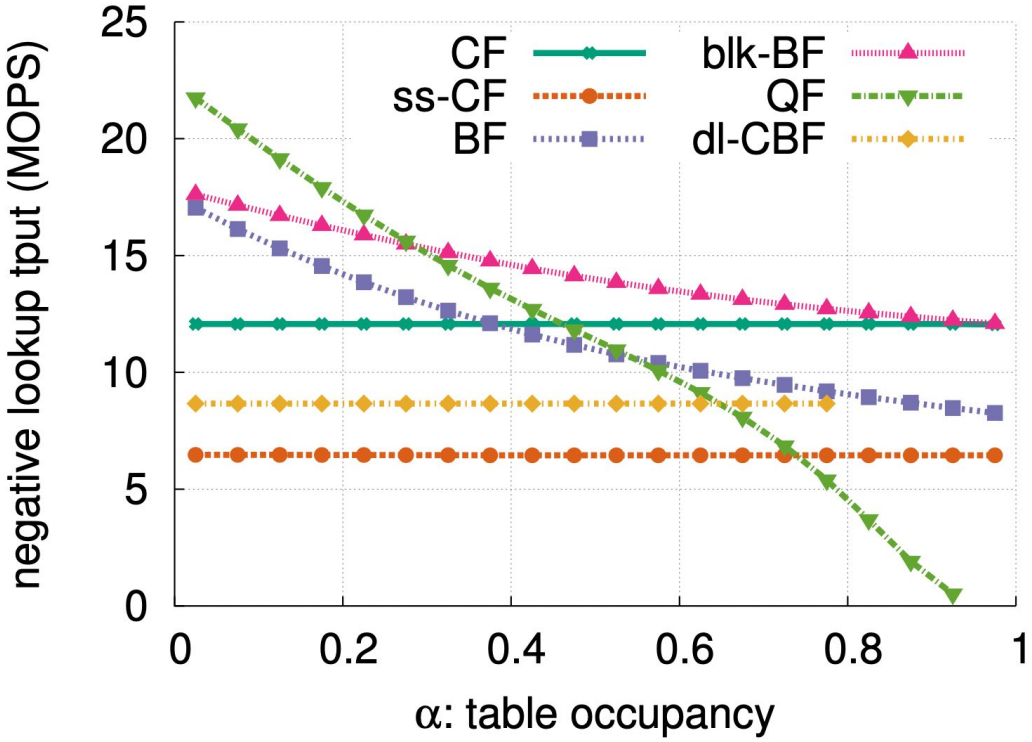
Efficienza Spaziale vs. Tasso Falsi Positivi

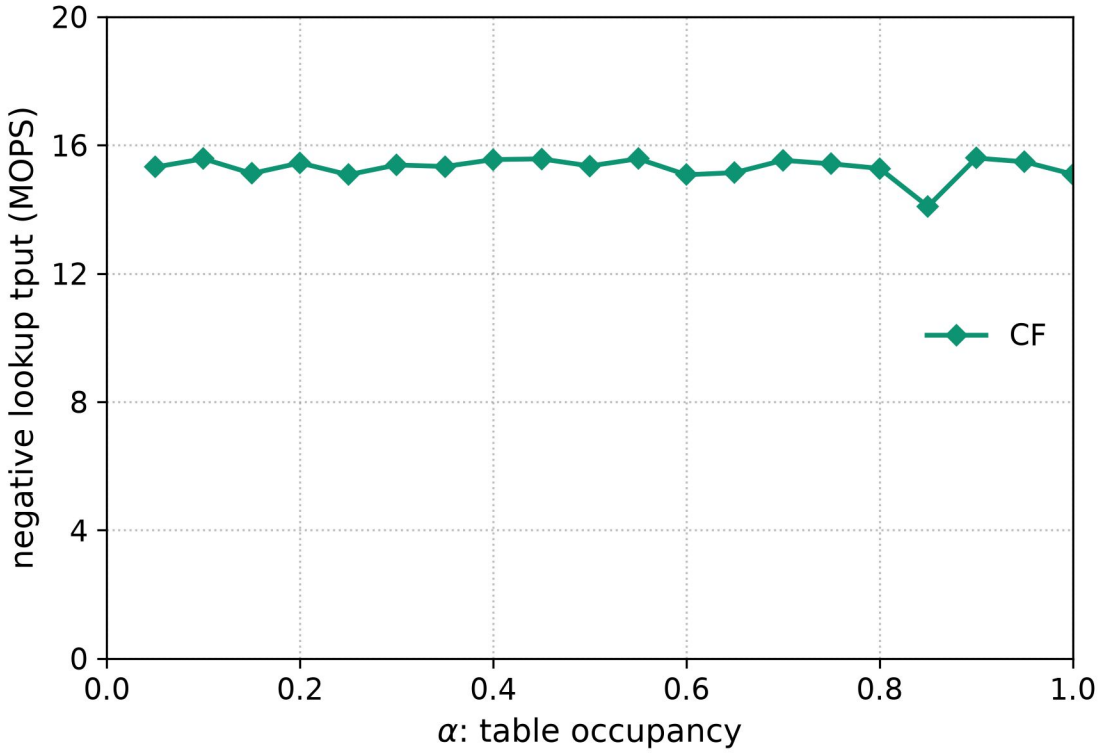




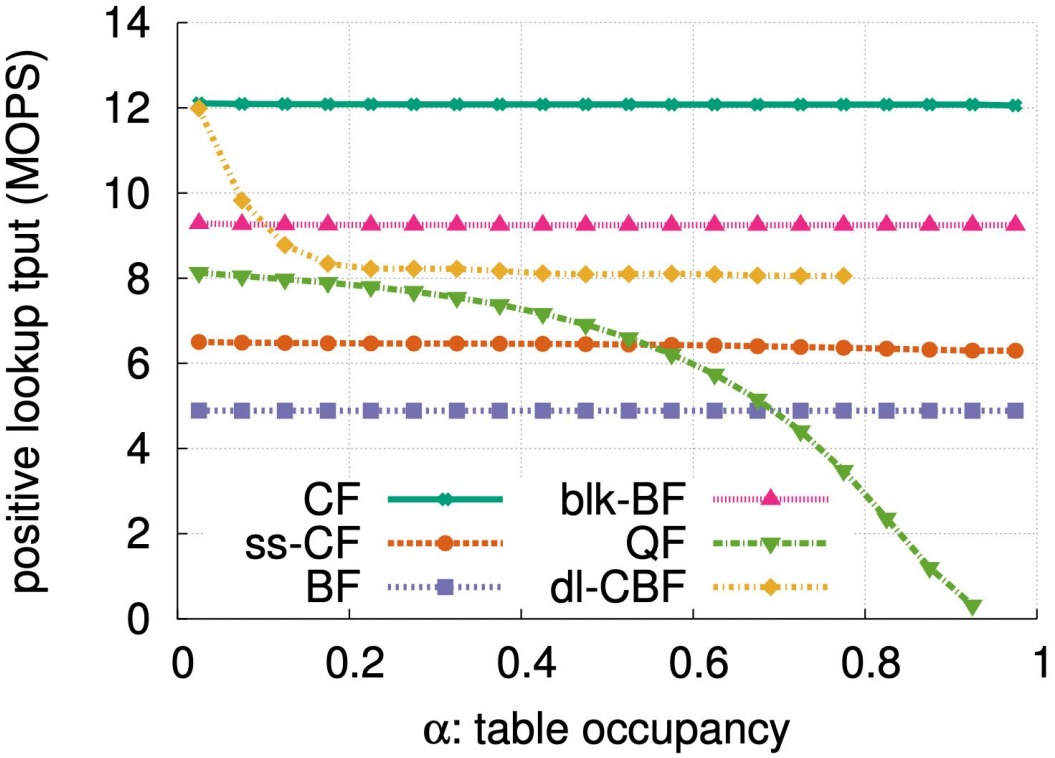
Quindi: $\varepsilon < 3\% \implies \text{Cuckoo} > \text{Bloom}$

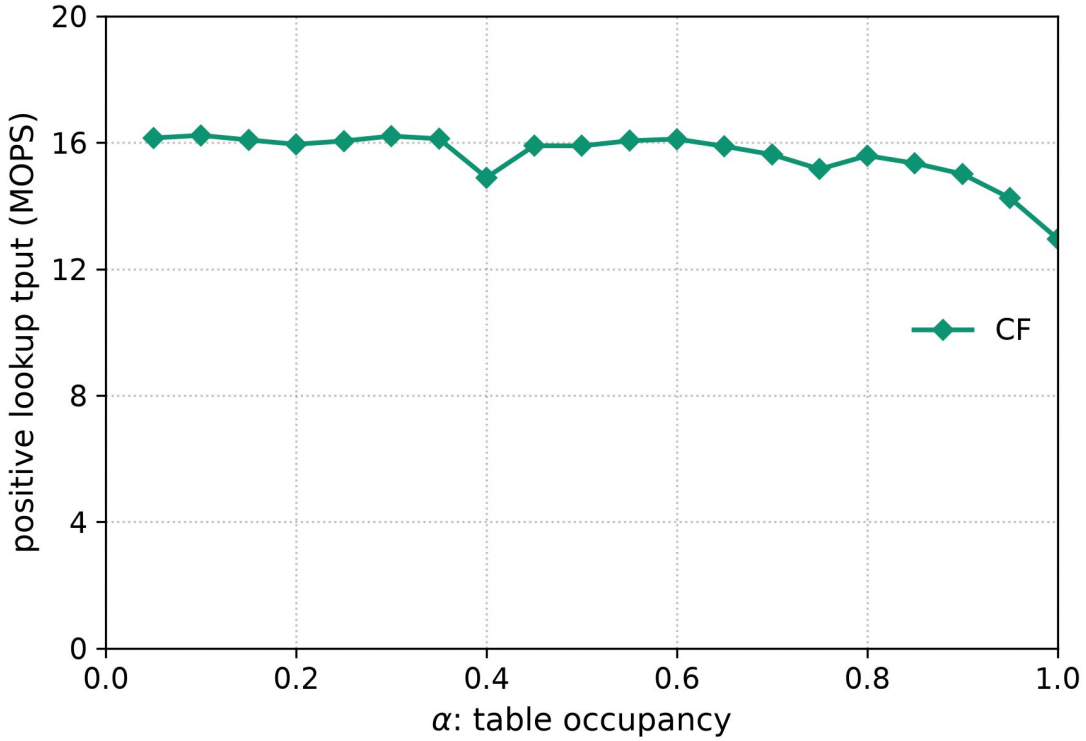
Prestazioni di *Negative Lookup*



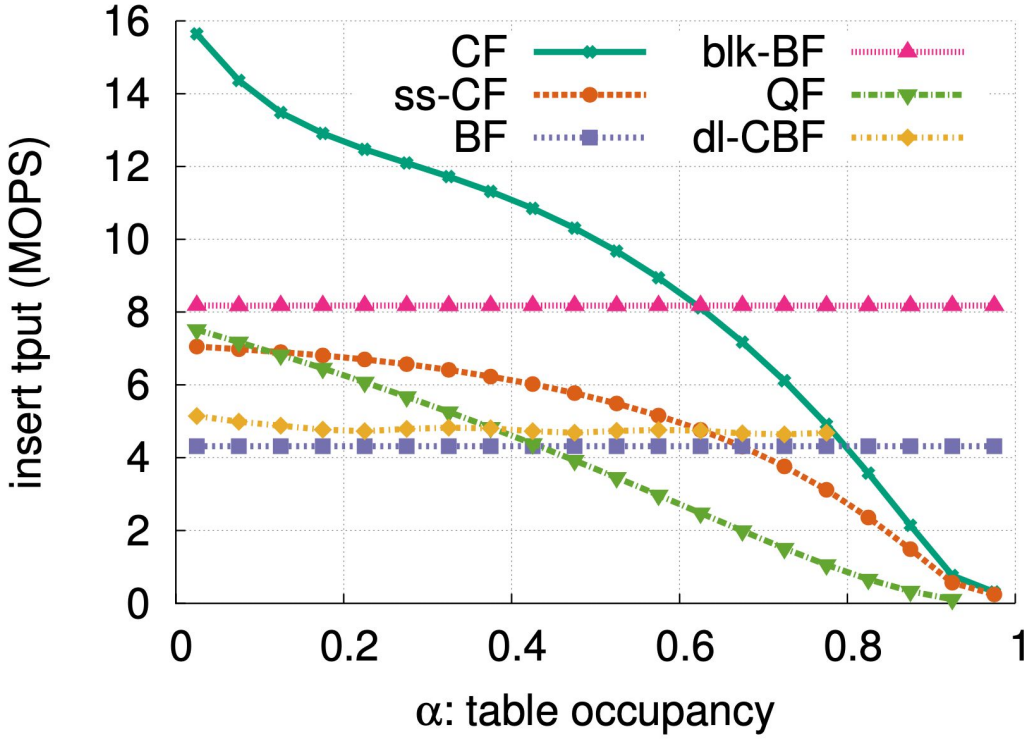


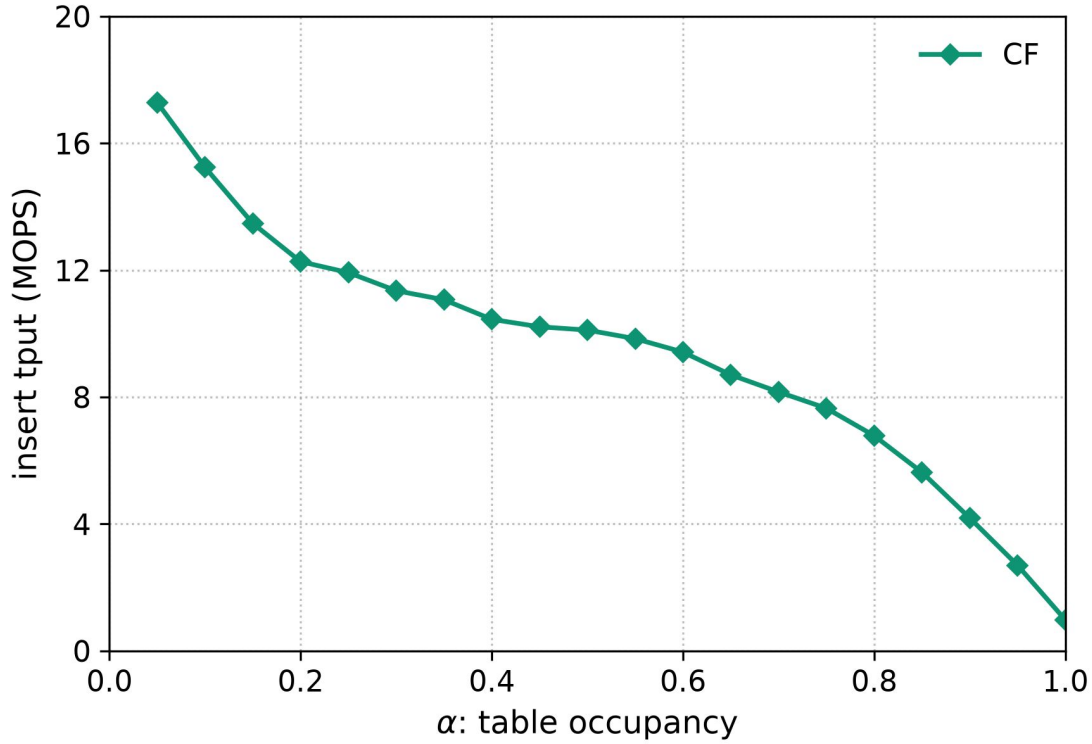
Prestazioni di *Positive Lookup*



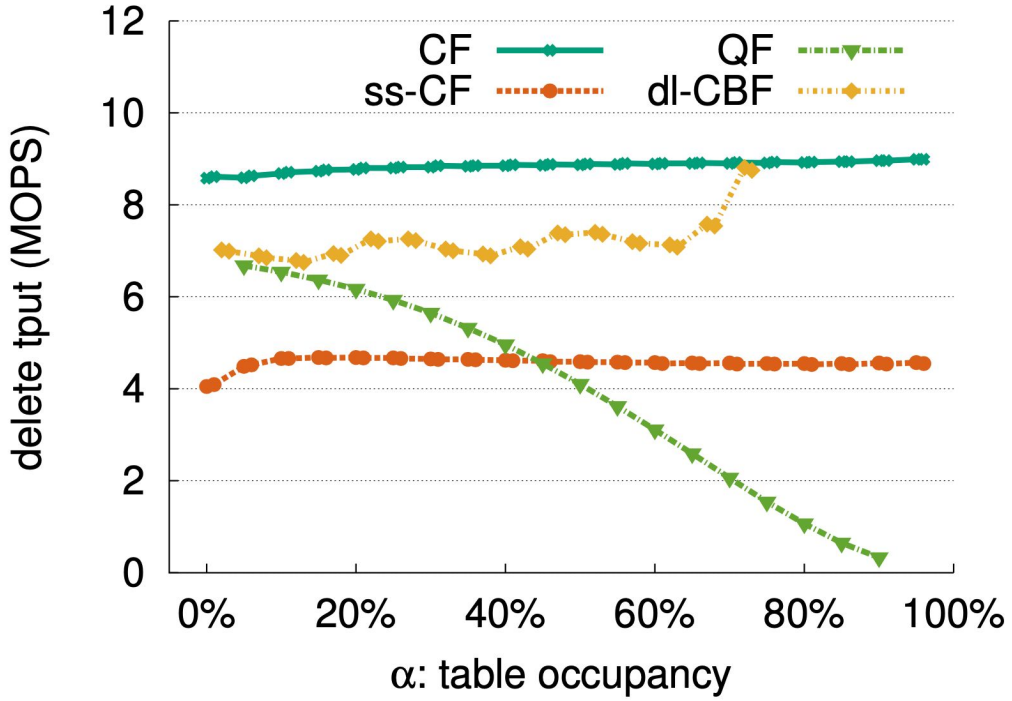


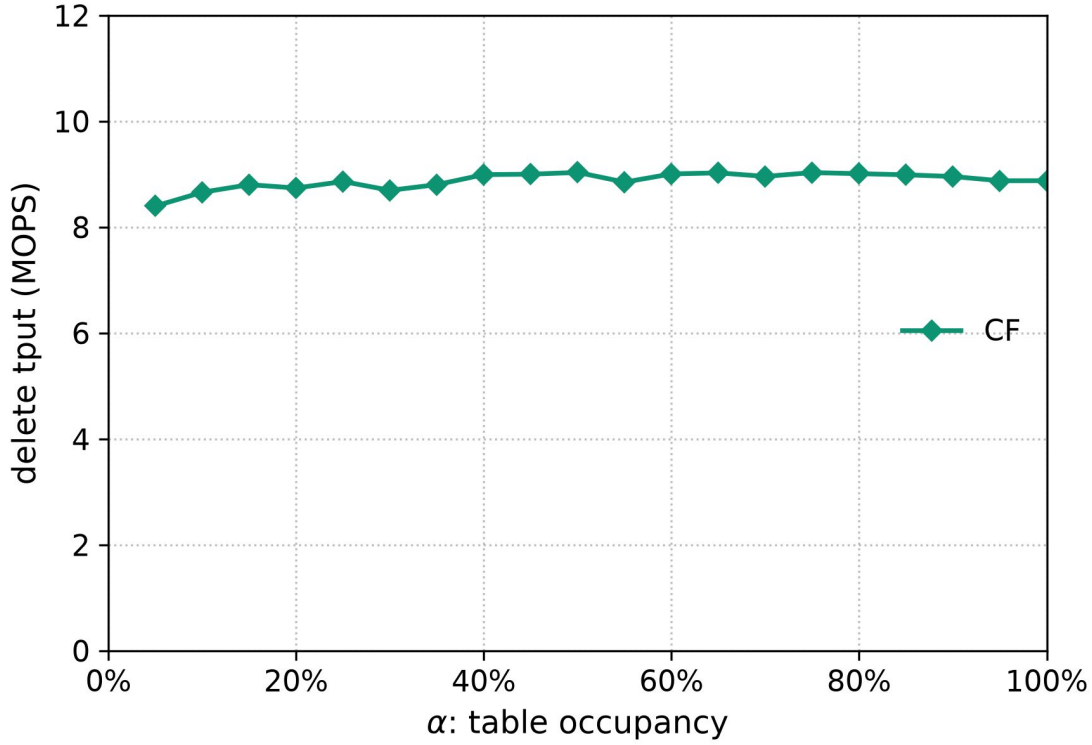
Prestazioni Inserimento

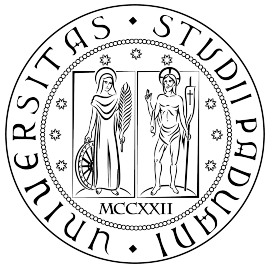




Efficienza e Stabilità di Cancellazione



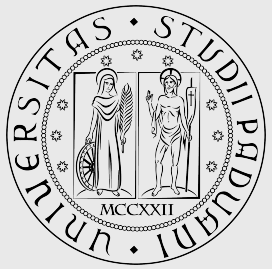




Conclusioni

Bilancio e *Trade-off*

- Vantaggi:
  - Elevata efficienza spaziale (superiore ai classici Bloom filter).
  - Cancellazione nativa e sicura.
- Compromessi:
  - Fattore di riempimento massimo limitato: $\alpha \approx 95\%$
  - Costo computazionale aggiuntivo dei rimpiazzi in condizioni di saturazione della struttura: $\alpha \to 1.0$



Bibliografia

[1] **B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher**, "Cuckoo Filter: Practically Better Than Bloom," in *In Proc. ACM CoNEXT*, 2014, pp. 75–87.