VLDB 2025
WELCOME TO LONDON

Article | Poster | Code

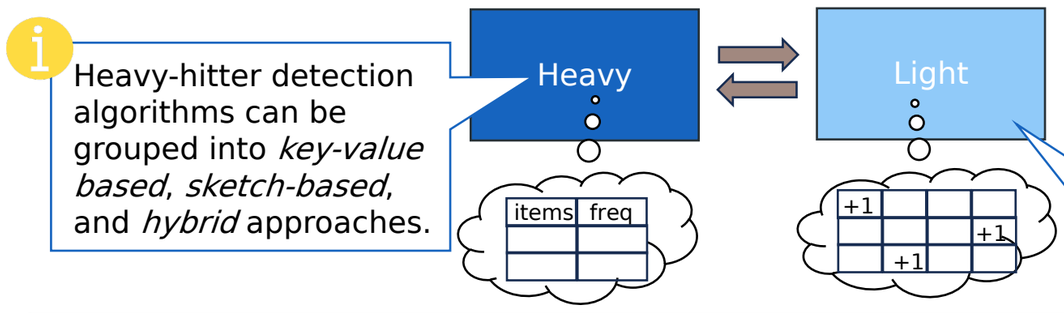
Cuckoo Heavy Keeper And The Balancing Act Of Maintaining Heavy Hitters In Stream Processing

Vinh Quang Ngo, Marina Papatriantafilou
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

Finding Heavy Hitters and State-of-the-art Algorithms

Finding heavy hitters is a fundamental problem with applications ranging from network monitoring to database query optimization, machine learning, and more.

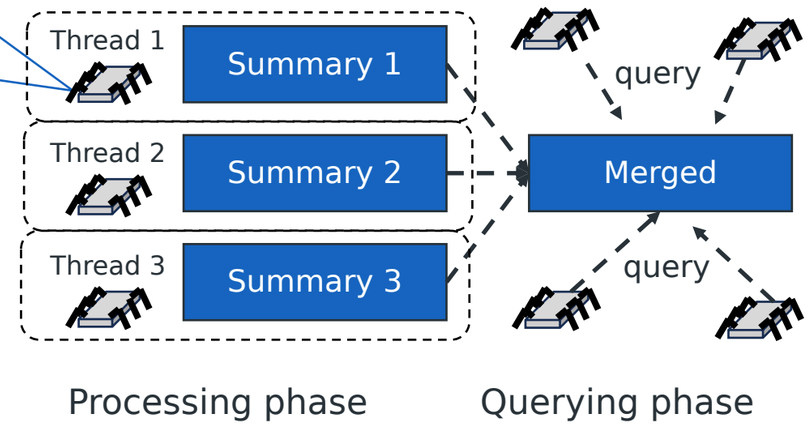
- Approximation algorithms offer practical solutions, but they present trade-offs involving **throughput**, **memory usage**, and **accuracy**.



- Modern applications often demand capabilities beyond sequential processing: **parallel scaling** and **support for concurrent queries and updates**.

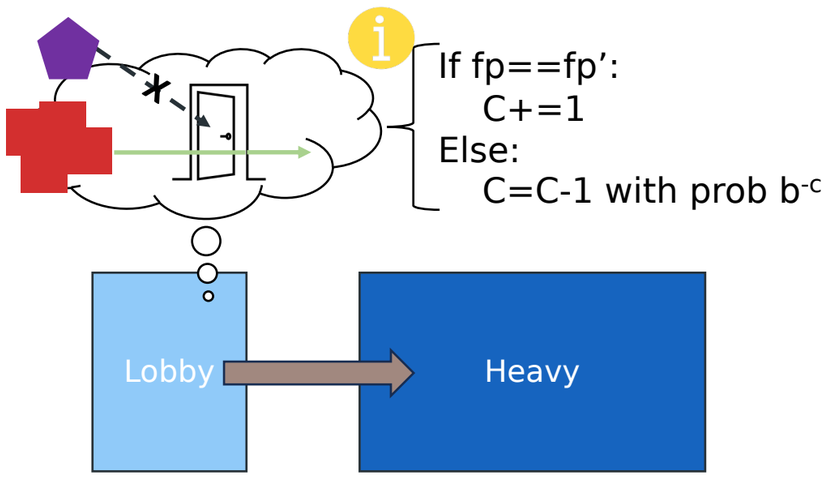
A common approach for parallelizing that requires frequent synchronization, mergeability and doesn't support concurrent queries and updates.

- Hash collisions among heavy hitters.
- Expensive data movement.
- Unused memory in light part.

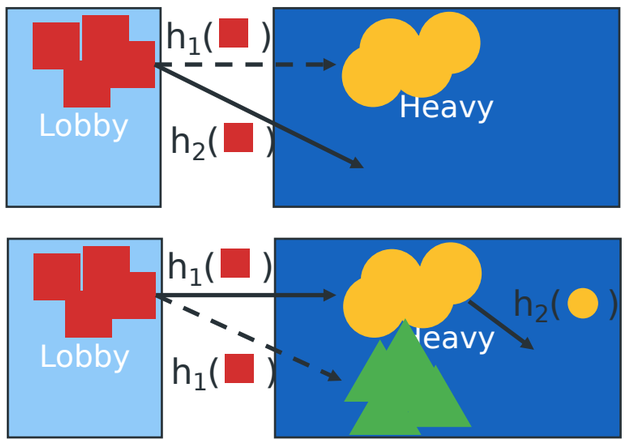


Cuckoo Heavy Keeper (CHK)

❶ *Inverts* the conventional flow by introducing *the lobby* data structure



❷ *Collision resolution* (cuckoo hashing) in the heavy part to avoid eviction



What makes CHK **different** from prior approaches:

- ❶ reduces data movement, improves throughput, makes performance predictable, and unlocks new algorithmic synergies that are inaccessible with prior approaches (e.g applying collision resolution).
- ❷ improves accuracy by giving colliding heavy hitters a second chance, *particularly under memory constraints where collisions are more likely to occur*.
- Not just asymptotic complexity, but also system aspects (e.g., cache, instruction cost) are considered.

Configurable approx bound: $\Pr[|\hat{f}(e) - f(e)| \geq \epsilon N\ ] \leq \frac{1}{\epsilon \mathcal{B}}$

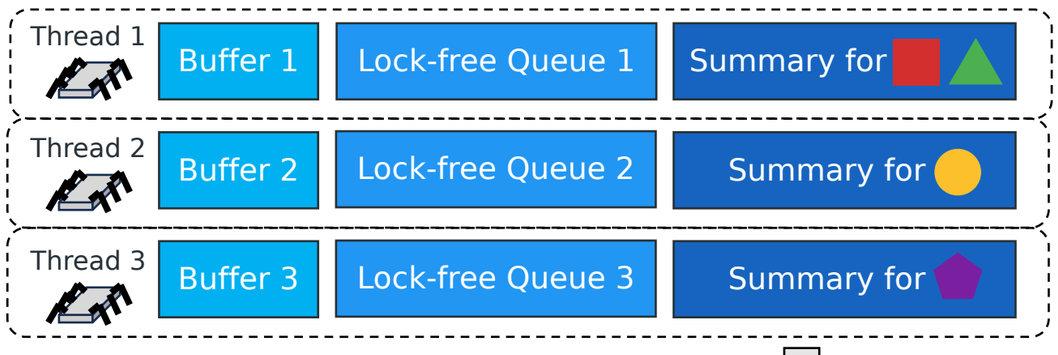
The Parallel Framework

Limitations of prior approaches	Our parallel framework
algorithm-dependent, require mergeability	**algorithm-agnostic** (can parallelize any algorithm)
mainly demanding synchronization	mainly **non-blocking, lock-free**
-	support concurrent insertions/queries
limited parallel speed-up	**high scalability**

Our parallel framework generalizes 2 key principles: ❶ *domain-splitting* and ❷ *delegated operations.* It offers two modes, depending on the workload:

- *Insertion-optimized* : for situations where insertions are predominant.
- *Query-optimized* : for scenarios where hh-queries are more frequent.

❶ Domain-splitting ❷ Operations are buffered and delegated to reduce synchronization

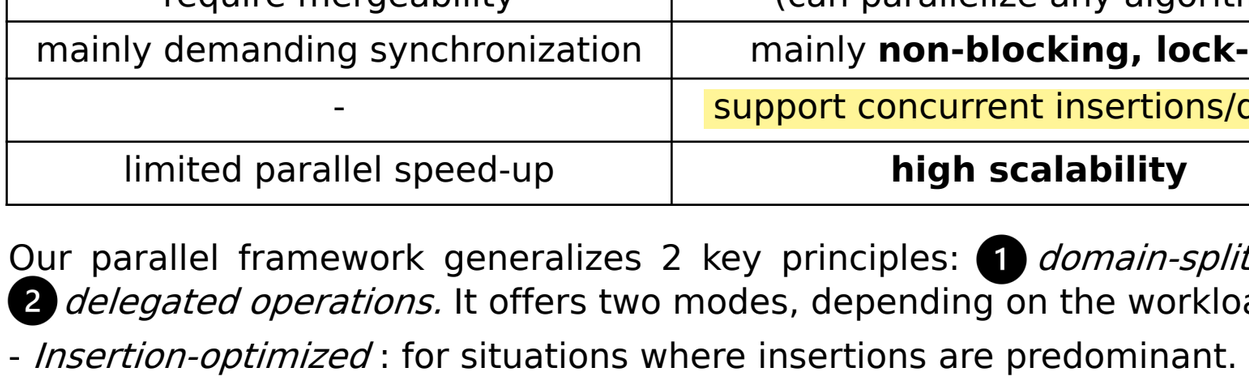


Evaluation

We performed an evaluation following the below experimental setup. Key findings are:

(1) Sequential Performance: CHK has 1.7x-5.7x higher throughput and up to 27,000x better accuracy compared to SOTA algorithms, even with *tight memory* and in *varying data skewness*.

(2) Parallel Performance: High, nearly linear scalability for any heavy-hitter algorithm, achieving billions of ops/s at 70 threads.



If you don't know about the skewness of your data, then CHK is your friend.

Parallel CHK can handle higher rates than others, especially with fewer threads, allowing you to save energy and retain processing capacity for other tasks.

CHALMERS UNIVERSITY OF TECHNOLOGY
Distributed Computing and Systems
Chalmers university of technology
relax
Funded by the European Union
UK Research and Innovation

VLDB 2025
WELCOME TO LONDON