Asymptotically Optimal Locally Private Heavy Hitters

via Parameterized Sketches

Hao Wu, Anthony Wirth

School of Computing and Information Systems

The University of Melbourne

POSTERA CRESCAM LAUDE

THE UNIVERSITY OF MELBOURNE

Frequency Estimation

- A set $\mathcal{U}$ of n users and a server.
- Each user $u \in [n]$ holds an element $v^{(u)}$ from some data domain $\mathcal{D}$ of size d.
- Each user reports their element.
- Two frequency estimation tasks for server:

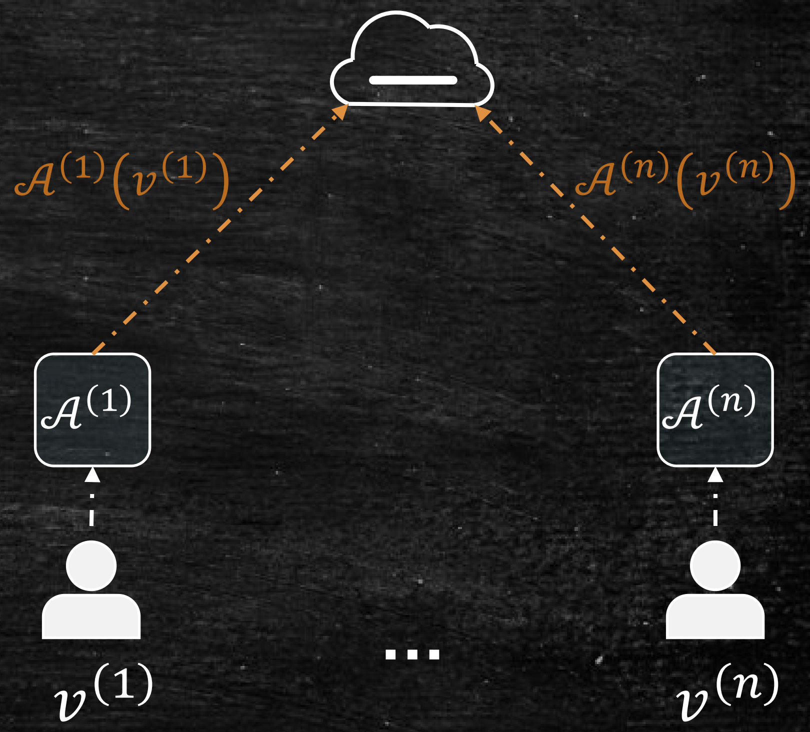
1. Frequency Oracle $\mathcal{A}_{oracle}$:
   - (Informally) given $v \in \mathcal{D}$, return an estimate of v's frequency in $\mathcal{U}$.
2. Succinct Histogram $\mathcal{A}_{hist}$:
   - (Informally) return a set of elements with high frequencies amongst $\mathcal{U}$, known as *heavy hitters*, together with their frequency estimates.

Local Differential Privacy

- To protect sensitive information, the users don't report their elements directly.
- Each user u perturbs $v^{(u)}$ with a local randomizer $\mathcal{A}^{(u)}: \mathcal{D} \to \mathcal{Y}$ before reporting.
- Server preforms the frequency estimation tasks based on the noisy reports.
- $\mathcal{A}^{(u)}$ is ε-local differentially private: for all $v, v' \in \mathcal{D}$ and all (measurable) $Y \subseteq \mathcal{Y}$,

$$\Pr[\mathcal{A}^{(u)}(v) \in Y] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}^{(u)}(v') \in Y]$$

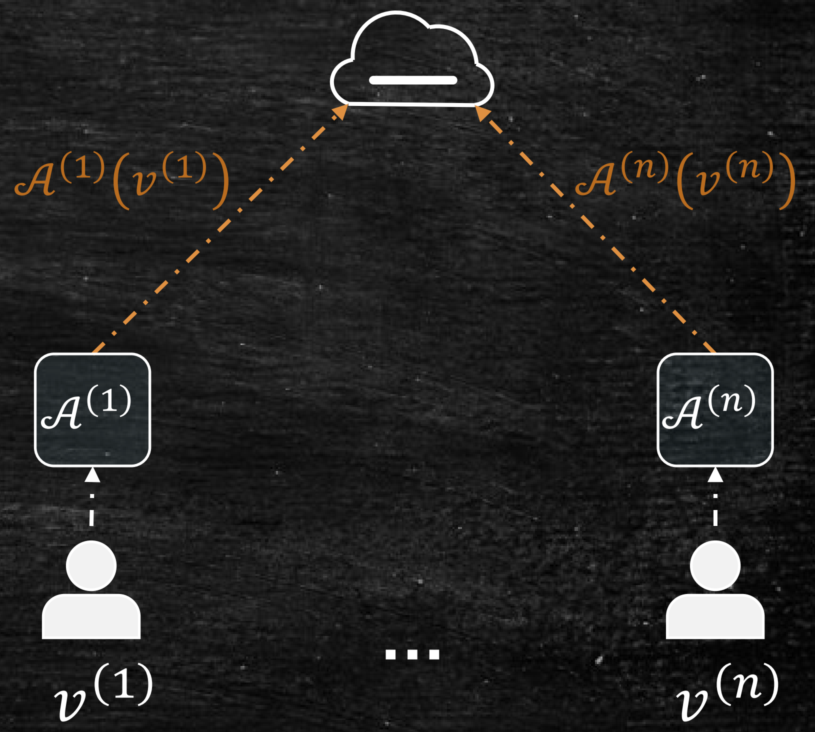
  - The output distribution of $\mathcal{A}^{(u)}$ varies little with the input.



Challenge

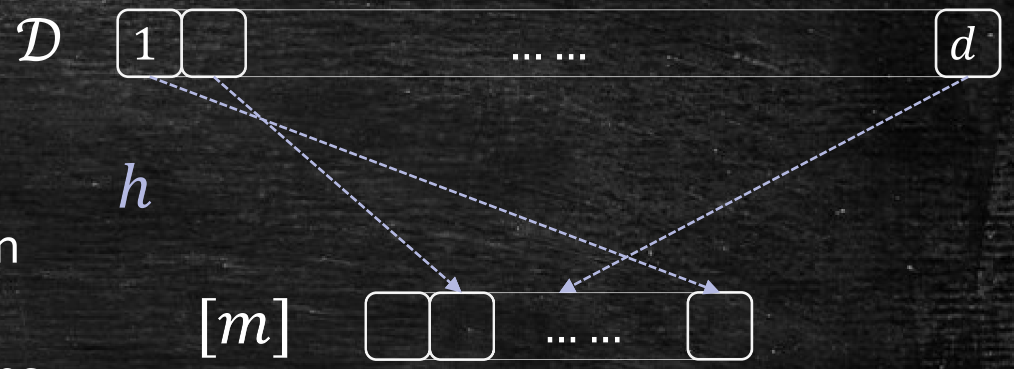
- The server-side algorithm's running time/memory usage may scale linearly with data domain size d.

- Example of concern: popular URLs with length up to 20 characters *(Fanti et al., 2016)*
  - URL characters can include digits (0-9), letters(A-Z, a-z), and a few special characters ("-" , "." , " " , "~ ").
  - Hence $d = 66^{20} \geq 10^{36}$

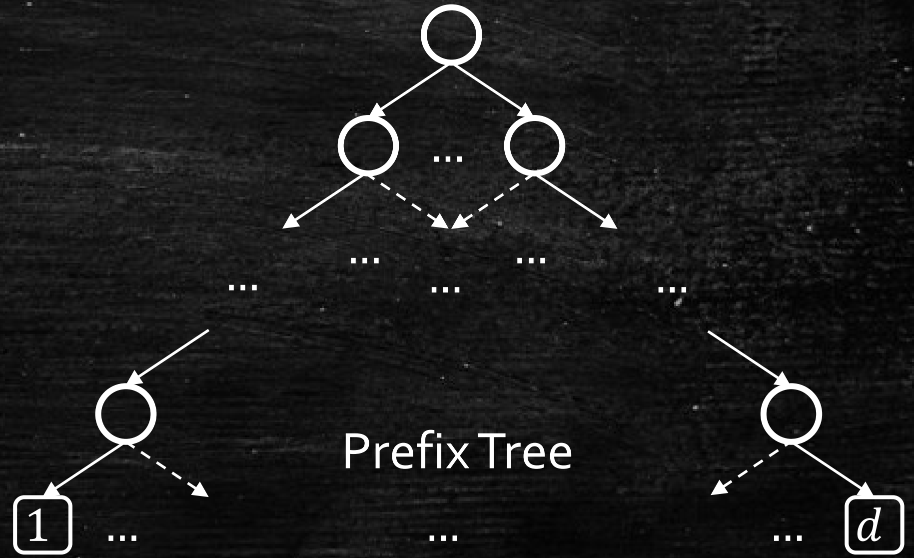


Existing Deployed Solutions

- *Sketching.*
  - Reduce the size of $\mathcal{D}$ for $\mathcal{A}_{oracle}$.
  - Pick a hash function h, which maps elements in $\mathcal{D}$ to a smaller domain $[m]$, for some $m \in \mathbb{N}^+$.
  - Repetitions may be required to handle collisions.



- *Hierarchical Searching.*
  - Avoid inspecting each element in $\mathcal{D}$.
  - Encode elements in $\mathcal{D}$ as strings, over which a prefix tree is constructed.
  - If an element in $\mathcal{D}$ is heavy, so are its prefixes.
  - Heavy hitters are identified top down.



Existing Work Comparison

For failure probability β,

- *Sketching* $\mathcal{A}_{oracle}$:
  - *FreqOracle* (Bassily et al., 17)
  - Estimation error:

    $$O\left((1/\epsilon) \cdot \sqrt{n \cdot \ln(n/\beta)}\right)$$

  - Server running time: $\tilde{O}(n)$
  - Server memory: $\tilde{O}(\sqrt{n})$
- *Hierarchical Searching* $\mathcal{A}_{hist}$:
  - *TreeHist* (Bassily et al., 17)
  - Estimation error:

    $$O\left((1/\epsilon) \cdot \sqrt{n \cdot \ln(d) \cdot \ln(n/\beta)}\right)$$

  - Server running time: $\tilde{O}(n)$
  - Server memory: $\tilde{O}(\sqrt{n})$

- *Easy to implement; Low time complexity and memory usage.*
  - *Their variants are studied experimentally* (Cormode et al. 21).
  - *Algorithms following this vein perform well in practice.*
- *Sup-optimal errors.*

Existing Work Comparison

- *Theoretical optimal error.*
- *Running time and memory usage depend on d.*

- *Non-Sketching $\mathcal{A}_{oracle}$:*
  - *HRR (Nguyên et al., 16; Cormode et al., 19)*
  - Estimation error (matches lower bound):

    $$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(1/\beta)}\right)$$

  - Server running time: $O(n+d)$
  - Server memory: $O(d)$

- *Theoretical optimal error.*
- *Due to the sophistication of error-correcting codes, it has not been implemented.*

- *Error-Correcting Code $\mathcal{A}_{hist}$:*
  - *PrivateExpanderSketch (Bun et al., 19)*
  - Estimation error (matches lower bound):

    $$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(d/\beta)}\right)$$

  - Server running time: $\tilde{O}(n)$
  - Server memory: $\tilde{O}(\sqrt{n})$

Can We Close the Gaps?

1. Are theoretical error guarantees of existing approaches *sketching* $\mathcal{A}_{oracle}$ and *hierarchical searching* $\mathcal{A}_{hist}$ best possible?

2. Or can we obtain algorithms of this type that achieve optimal error guarantee?

New $\mathcal{A}_{oracle}$ **HadaOracle** & $\mathcal{A}_{hist}$ **HadaHeavy**

- *Sketching $\mathcal{A}_{oracle}$*:
  - *FreqOracle* *(Bassily et al., 17)*
  - Estimation error:
    $$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(n/\beta)}\right)$$
  - Server running time: $\tilde{O}(n)$
  - Server memory: $\tilde{O}(\sqrt{n})$

- *Non-Sketching $\mathcal{A}_{oracle}$*:
  - *HRR* *(Nguyễn et al., 16; Cormode et al., 19)*
  - Estimation error (matches lower bound):
    $$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(1/\beta)}\right)$$
  - Server running time: $O(n+d)$
  - Server memory: $O(d)$

- *Hierarchical Searching $\mathcal{A}_{hist}$*:
  - *TreeHist* *(Bassily et al., 17)*
  - Estimation error:
    $$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(d)\cdot\ln(n/\beta)}\right)$$
  - Server running time: $\tilde{O}(n)$
  - Server memory: $\tilde{O}(\sqrt{n})$

if $\beta = n^{-c}$

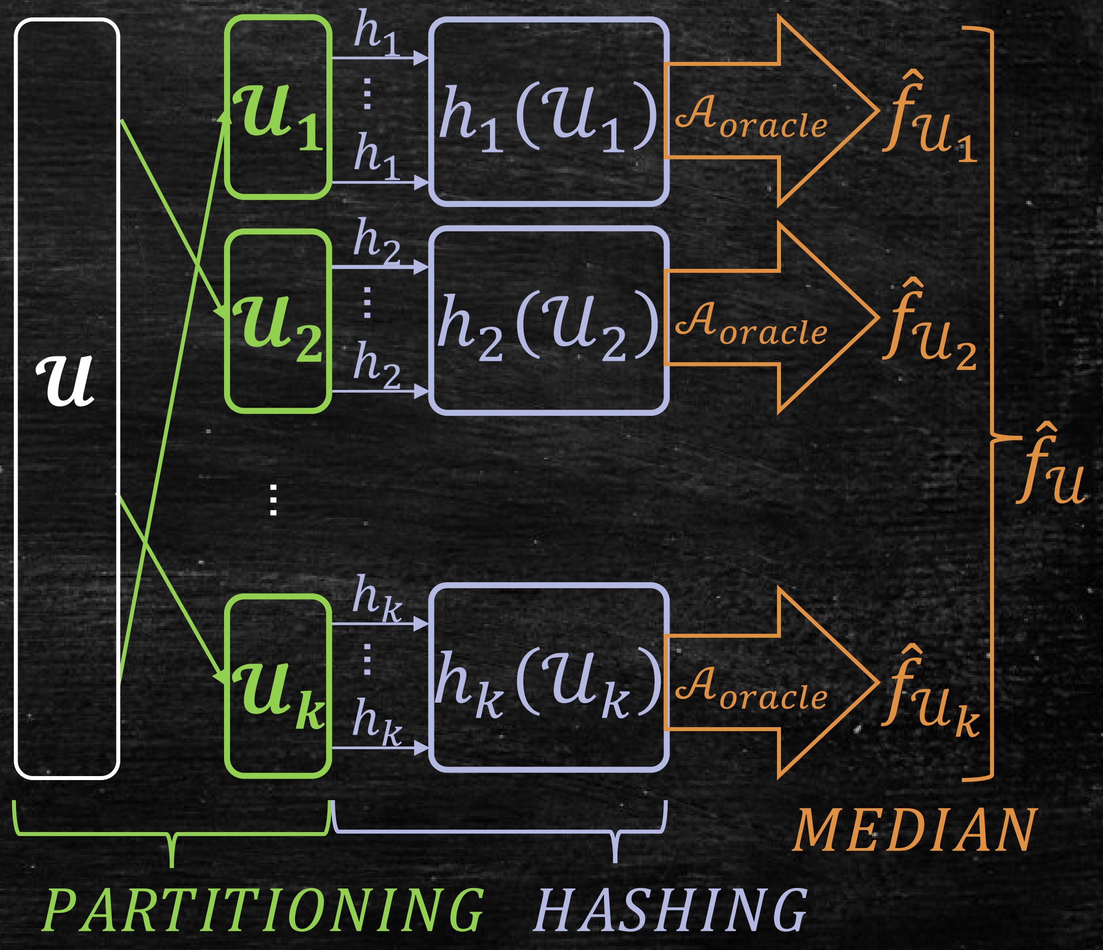
- *Error-Correcting Code $\mathcal{A}_{hist}$*:
  - *PrivateExpanderSketch* *(Bun et al., 19)*
  - Estimation error (matches lower bound):
    $$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(d/\beta)}\right)$$
  - Server running time: $\tilde{O}(n)$
  - Server memory: $\tilde{O}(\sqrt{n})$

Summary of Results

	Performance Metric	Server Time	Server Mem	Worst-Case Error	Lower Bound
FO	HadaOracle (*this work*)	$\tilde{O}(n)$	$\tilde{O}(\sqrt{n})$	$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(1/\beta)}\right)$	$\Omega\left(\frac{1}{\epsilon}\sqrt{n\ln\frac{1}{\beta}}\right)$
	HRR (*Nguyên et al., 16; Cormode et al., 19*)	$\tilde{O}(d)$	$\tilde{O}(d)$	$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(1/\beta)}\right)$	
	FreqOracle (*Bassily et al., 17*)	$\tilde{O}(n)$	$\tilde{O}(\sqrt{n})$	$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(n/\beta)}\right)$	
	Hashtogram (*Bassily et al., 17*)	$\tilde{O}(n)$	$\tilde{O}(\sqrt{n})$	$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(n/\beta)}\right)$	
S-Hist	HadaHeavy (*this work*)	$\tilde{O}(n)$	$\tilde{O}(\sqrt{n})$	$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(d)\cdot\left(1+\frac{\ln(1/\beta)}{\ln n}\right)}\right)$	$\Omega\left(\frac{1}{\epsilon}\sqrt{n\ln\frac{d}{\beta}}\right)$
	TreeHist (*Bassily et al., 17*)	$\tilde{O}(n)$	$\tilde{O}(\sqrt{n})$	$O\left((1/\epsilon)\cdot\sqrt{n\cdot(\ln d)\cdot\ln(n/\beta)}\right)$	
	PrivateExpanderSketch (*Bun et al., 19*)	$\tilde{O}(n)$	$\tilde{O}(\sqrt{n})$	$O\left((1/\epsilon)\cdot\sqrt{n\cdot\ln(d/\beta)}\right)$	

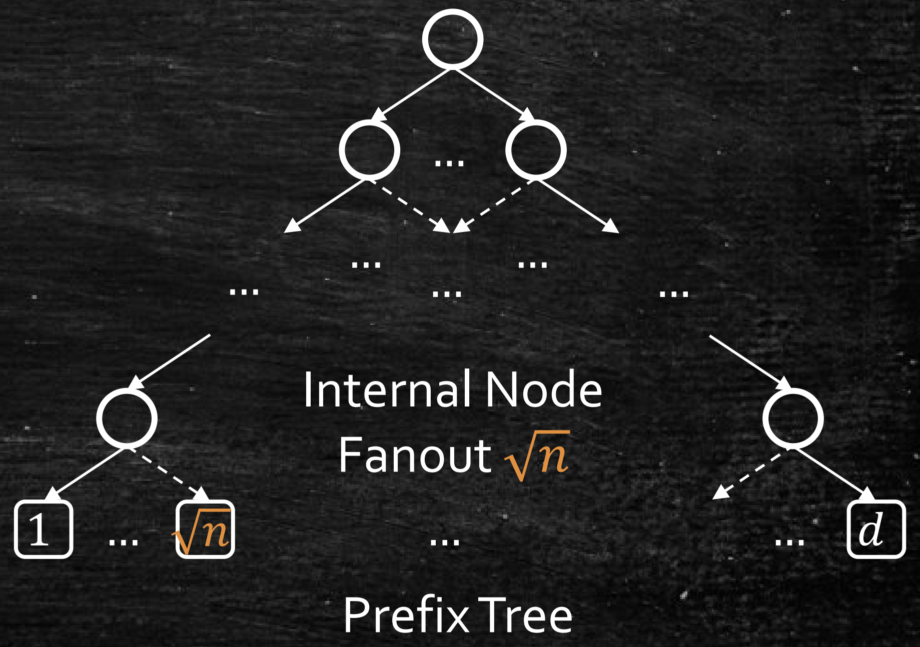
HadaOracle

- Randomly partition users $\mathcal{U}$ into subsets $\mathcal{U}_1, \ldots, \mathcal{U}_k$, where $k \in \Theta(\ln(1/\beta))$, analysis of which is based on *martingale concentration inequalities*.
- For each $i \in [k]$:
  - pick a pairwise independent hash function $h_i : \mathcal{D} \to [m]$, where $m \in \Theta(\sqrt{n})$.
  - for each $u \in \mathcal{U}_i$, replace their element $v^{(u)}$ by $h_i(v^{(u)})$.
  - denote the hashed elements of users in $\mathcal{U}_i$ by $h_i(\mathcal{U}_i)$.
- Construct $\mathcal{A}_{oracle}$(*HRR*) over all $h_i(\mathcal{U}_i)$:
  - for a query $v \in D$, return the scaled median of its frequency estimates.



HadaHeavy

- Encode elements in $\mathcal{D}$ as strings with alphabet size $\sqrt{n}$.
- Construct the prefix tree.
  - It has depth $L \doteq 2 \cdot \frac{\ln d}{\ln n}$.
  - Randomly split the users $\mathcal{U}$ into L groups, $U_1, U_2, \ldots, U_L$.
  - Build a *HadaOracle* for nodes with depth i, based on users U_i, for each $i \in [L]$.
- Heavy hitters are identified top down.
  - If a node is heavy, inspect its children to find possible heavy nodes.



THANK YOU

REFERENCES

- *G. C. Fanti, et.al, "Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries," Priv. Enhancing Technol., 2016.*
- *G. Cormode, et.al, "Answering range queries under local differential privacy," VLDB Endow., 2019.*
- *G. Cormode, et.al, "Frequency estimation under local differential privacy [experiments, analysis and benchmarks]," CoRR, 2021.*
- *M. Bun, et.al, "Heavy hitters and the structure of local privacy," ACM Trans. Algorithms, 2019.*
- *R. Bassily, et.al, "Practical locally private heavy hitters," NeurIPS., 2017,*
- *T. T. Nguyên, et.al, "Collecting and analyzing data from smart device users with local differential privacy," CoRR., 2016.*