Hierarchical Heavy Hitters with the Space Saving Algorithm

Michael Mitzenmacher, Thomas Steinke, Justin Thaler
Harvard SEAS

January 16, 2012
Overview

The Problem

Previous Work

Our Algorithm

Results
Motivating Problem

- Monitoring network traffic.
- Streaming problem. (Lots of data.)
- Want to find major patterns. (Identify big servers/server farms; connections; DDoS.)
The Hierarchy

- Data has structure.
- Subnets correspond to organisations, ISPs, geographical locations.
- Subnets are hierarchical: 123.234.*.* ⪯ 123.234.78.79
- Subnet masks form lattice.
- Level: 0: *.*.*.*, 1: 1.*.*.*, 2: 1.2.*.*, 3: 1.2.3.*, 4: 1.2.3.4
- Find frequent subnets or subnet pairs (Hierarchical Heavy Hitters).
- Condition counts: If 123.234.132.199 is frequent there is no need to output 123.234.132.*, 123.234.*.*, and 123.*.*.* as well.
Multidimensional Lattice
**HHH Definition**

**Definition**
(Exact HHHs) The set of exact Hierarchical Heavy Hitters are defined inductively.

1. $\mathcal{HHH}_L$, the hierarchical heavy hitters at level $L$, are the heavy hitters of $S$, that is the fully specified elements whose frequencies exceed $\phi N$.

2. Given a prefix $p$ from Level$(l)$, $0 \leq l < L$, define $\mathcal{HHH}^p_{l+1}$ to be the set $\{h \in \mathcal{HHH}_{l+1} \land h \prec p\}$ i.e. $\mathcal{HHH}^p_{l+1}$ is the set of descendants of $p$ that have been identified as HHHs. Define the conditioned count of $p$ to be $F_p = \sum_{(e \in S) \land (e \preceq p) \land (e \not\preceq \mathcal{HHH}^p_{l+1})} f(e)$. The set $\mathcal{HHH}_l$ is defined as

$$\mathcal{HHH}_l = \mathcal{HHH}_{l+1} \cup \{p : (p \in \text{Level}(l) \land (F_p \geq \phi N))\}.$$ 

3. The set of exact Hierarchical Heavy Hitters $\mathcal{HHH}$ is defined as the set $\mathcal{HHH}_0$. 
Definition

(Approximate HHHs) Given parameter $\epsilon$, the Approximate Hierarchical Heavy Hitters problem with threshold $\phi$ is to output a set of items $P$ from the lattice, and lower and upper bounds $f_{\min}(p)$ and $f_{\max}(p)$, such that they satisfy two properties, as follows.

1. **Accuracy.** $f_{\min}(p) \leq f(p) \leq f_{\max}(p)$, and $f_{\max}(p) - f_{\min}(p) \leq \epsilon N$ for all $p \in P$.

2. **Coverage.** For all prefixes $p$, define $P_p$ to be the set \( \{q \in P : q \prec p\} \). Define the conditioned count of $p$ with respect to $P$ to be $F_p = \sum_{(e \in S) \land (e \preceq p) \land (e \nprec P_p)} f(e)$. We require for all prefixes $p \notin P$, $F_p < \phi N$. 
Ancestry
Ancestry: 2D
Reduction to Counting

- When you receive 123.234.78.79, record 123.234.78.*, 123.234.*.*, and 123.*.*.* as well.
- Keep track of these counts separately (using approximate counting algorithm for Heavy Hitters problem).
- Use Space Saving (Metrally et al.).
- Recover hierarchy in postprocessing.
Results

We implemented all algorithms in C (gcc 4.1.2) and tested on AMD Opteron 850 (4 single cores, 64-bit, 2.4GHz, 1MB cache, 8GB memory) using real data (www.caida.org). Compare five aspects:

- Time
- Memory
- Output Size
- Accuracy
- Programmer Time
We have worst-case rather than amortized bounds on update time.
Results: Memory

Byte-granularity in one dimension with $\varepsilon = 0.001$ and $\phi = 0.002$

Byte-granularity in two dimensions with $\varepsilon = 0.01$ and $\phi = 0.02$

Our algorithm uses fixed-size tables, rather than dynamic memory.
Results: Output Size

We have good analytical bounds on output size.
Results: Accuracy

Byte-granularity in one dimension with $\varepsilon=0.001$ and $\phi=0.002$

Byte-granularity in two dimensions with $\varepsilon=0.001$ and $\phi=0.002$

We have improved worst-case and average-case analytical bounds on accuracy.
Results: Programmer Time

Not scientific, but our algorithm is significantly simpler.
Concluding remarks

- Our algorithm has better worst case and average case bounds.
- Our algorithm parallelizes well.
- Our algorithm suits TCAM implementations.
- Simplicity!