# Streaming Verification of Outsourced Computation

**Graham Cormode**

G.Cormode@warwick.ac.uk

Amit Chakrabarti (Dartmouth)

Andrew McGregor (U Mass Amherst)

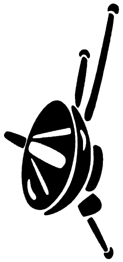
Michael Mitzenmacher (Harvard)

Justin Thaler (Harvard)

Ke Yi (HKUST)

# Big Data Streams

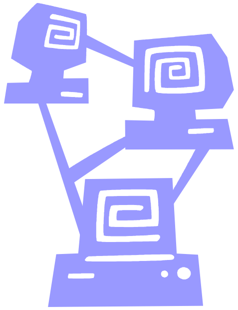
- The data stream model requires computation in small space with a single pass over input data
  - Models large network data, database transactions



- Fundamental challenge of data stream analysis: Too much information to store or transmit
- So process data as it arrives: one pass, small space: the *data stream* approach.
- Approximate answers to many questions are OK, if there are guarantees of result quality
  - Parameters: space needed, time per update as function of approximation accuracy, probability of error

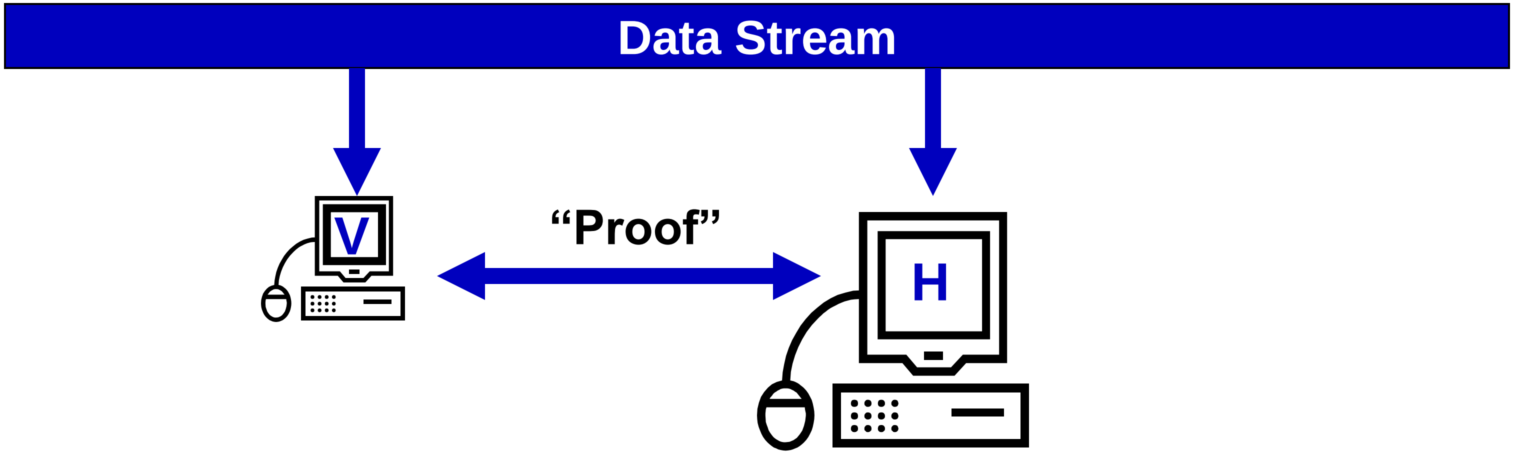
# Data Stream Algorithms

- Many problems solved efficiently in streaming model
  - $F_0$: How many distinct items (out of $10^{18}$ possible)?
  - HH: Which items occur most frequently?
  - H: What is the (empirical) entropy of the observed dbn?
- But many other natural problems are “hard” in this model
  - Hardness means large amount of space is needed
  - E.g. Was a particular item in the stream?
  - E.g. What is inner product of two vectors?
- Lower bounds proved via communication complexity
  - Independent of any assumptions on computational power



# Streaming Interactive Proofs

- "Practical" solution: outsource to a more powerful "prover"
  - Fundamental problem: how to be sure that the prover is being honest?
- Prover provides "proof" of the correct answer
  - Ensure that "verifier" has very low probability of being fooled
  - Related to communication complexity Arthur-Merlin model, and Algebrization, with additional streaming constraints

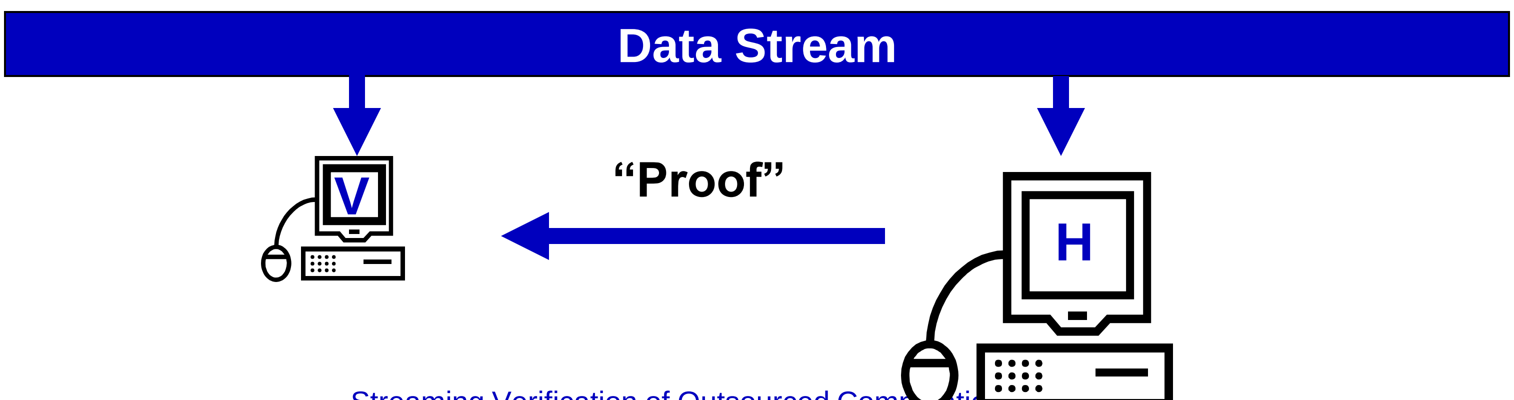


# Motivating Applications

- Cloud Computing
  - To save money, and energy, outsource data to a 3rd party
  - But want to know they are honest, without duplicating!
  - Use a streaming interactive proof to verify computation
- Trusted Hardware
  - Hardware components within a (distributed) system (e.g. video card, additional computing cores)
  - Use streaming interactive proofs for (mutual) trust

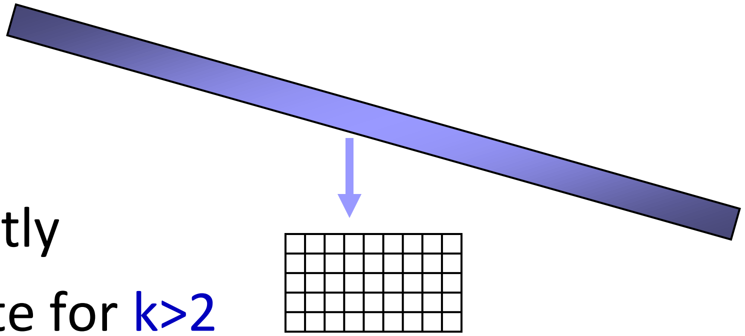
# One Round Model

- One-round model [Chakrabarti, C, McGregor 09]
  - Define protocol with help function h over input length N
  - Maximum length of h over all inputs defines *help cost*, H
  - Verifier has V bits of memory to work in
  - Verifier uses randomness so that:
    - For all help strings, $\Pr[\text{output} \neq f(x)] \leq \delta$
    - Exists a help string so that $\Pr[\text{output} = f(x)] \geq 1-\delta$
  - H = 0, V = N is trivial; but H = N, V = polylog N is not



# Frequency Moments

- Given a sequence of $m$ items, let $w_i$ denote frequency of item $i$
- Define $F_k = \sum_i |w_i|^k$
  - Core computation in data streams
  - Requires $\Omega(N)$ space to compute exactly
  - Need polynomial space to approximate for $k>2$



- Results: for $h,v$ s.t. $(hv) > N$, exists a protocol with $H = k^2\ h \log m$, $V = O(k\ v \log m)$ to compute $F_k$
  - Lower bounds: $HV = \Omega(N)$ necessary for exact, and $HV = \Omega(N^{1-5/k})$ for approximate $F_k$ computation

# Frequency Moments

3 7 1 2 0 8 5 9 1 1 1 0

- Map [N] to $h \times v$ array
- Interpolate entries in array as a polynomial $f(x,y)$
- Verifier picks random $r$, evaluates $f(r, j)$ for $j \in [v]$
  - Low-degree extension (LDE) of the input
- Prover sends $s(x) = \sum_{j\in[v]} f(x, j)^k$ (degree $kh$)
  - Verifier checks $s(r) = \sum_{j\in[v]} f(r,j)^k$
  - Output $F_k = \sum_{i \in [h]} s(i)$ if test passed
- Probability of failure small if evaluated over large enough field

| | | | |
|---|---|---|---|
| 3 | 7 | 1 | 2 |
| 0 | 8 | 5 | 9 |
| 1 | 1 | 1 | 0 |

12 -1 2 -90

# Streaming LDE Computation

- Must evaluate $f(r,i)$ incrementally as $f()$ is defined by stream
- Structure of polynomial means updates to $(a,b)$ cause

$$f(r,i) \leftarrow f(r,i) + p_{a,b}(r,i)$$

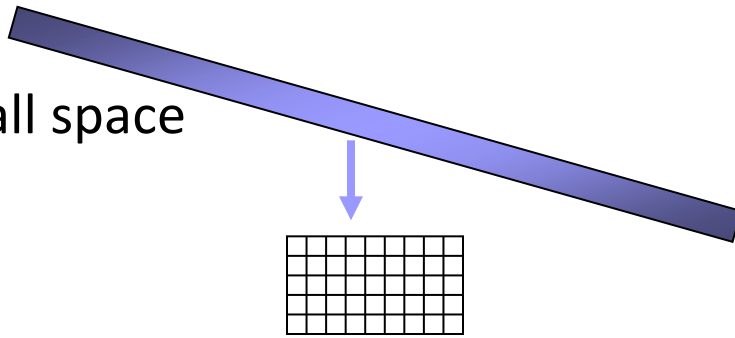
where $p_{a,b}(x,y) = \prod_{i \in [h]\setminus\{a\}} (x-i)(a-i)^{-1} \cdot \prod_{j \in [v]\setminus\{b\}} (y-j)(b-j)^{-1}$

  – Lagrange polynomial, can be evaluated in small space

- Can be computed quickly, using appropriate precomputed look-up tables

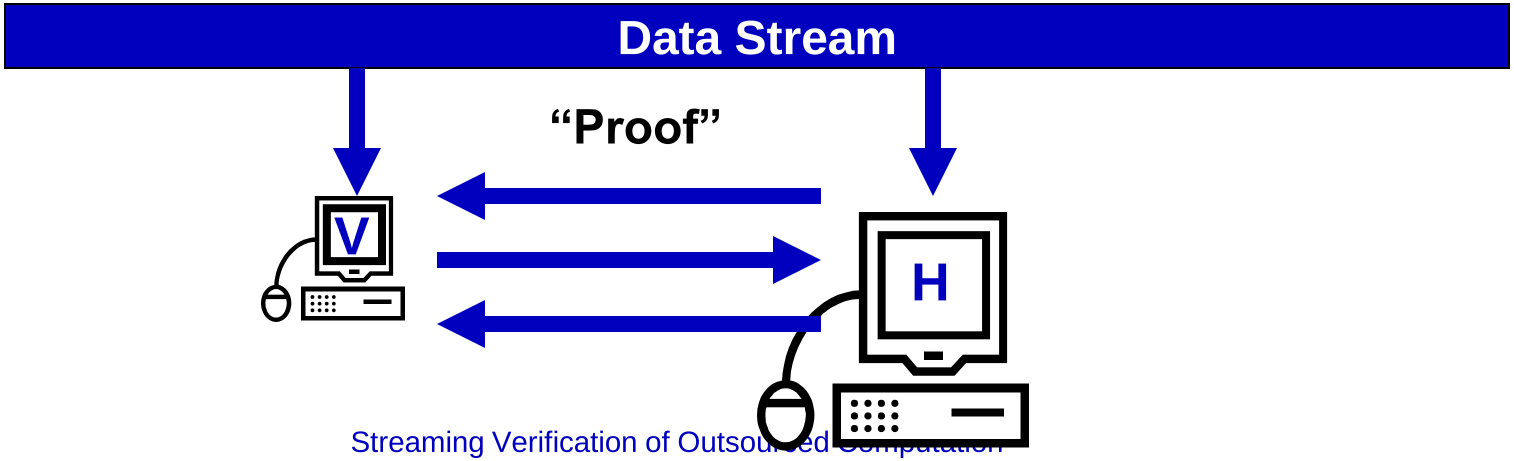
# Applications of Frequency Moments

- Inner products: $x \cdot y = ½ (F_2(x+y) – (F_2(x) + F_2(y)))$
  – Adapt previous protocol to verify directly
- Approximate $F_2$:
  – Methods known to $(1 \pm \varepsilon)$ approximate $F_2$ by computing $F_2$ of a random projection
  – Random projection computable in small space
  – Gives $HV = \Theta(1/\varepsilon^2)$ tradeoff
- Approximate $F_\infty = \max_i m_i$ :
  – Observe that $F_\infty{}^t \leq F_t \leq N\, F_\infty{}^t$
  – Pick $t = \log N/\log(1+\varepsilon)$ to get $(1+\varepsilon)$ approx to $F_\infty$
  – Gives $HV = \Theta(1/\varepsilon^3 \text{ poly-log } N)$ tradeoff



# Multi-Round Protocol

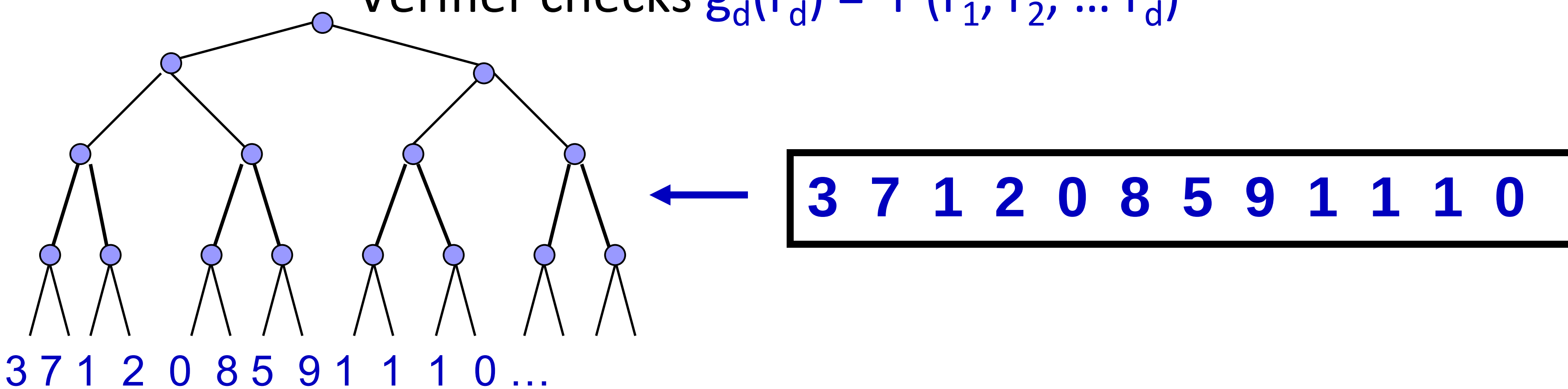
- Advantage of one-round protocols: Prover can provide proof without direct interaction (e.g. publish + go offline)
- Disadvantage: Resources still polynomial in input size
- Multi-round protocol improves exponentially [C, Thaler, Yi 12]:
  - Prover and Verifier follow communication protocol
  - H now denotes upper bound on total communication
  - V is verifier's space, study tradeoff between H and V as before



# Multi-Round Frequency Moments

Now index data using $\{0,1\}^d$ in $d = \log N$ dimensional space

- Verifier picks one $(r_1 \dots r_d) \in [p]^d$, and evaluates $f^k(r_1, r_2, \dots r_d)$
- Round 1: Prover sends $g_1(x_1)=\sum_{x_2 \dots x_d} f^k(x_1, x_2 \dots x_d)$, V sends $r_1$
- Round i: Prover sends $g_i(x_i) = \sum_{x_{i+1} \dots x_d} f^k(r_1, r_2 \dots r_{i-1}, x_i, x_{i+1} \dots x_d)$
  Verifier checks $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$, sends $r_i$
- Round d: Prover sends $g_d(x_d) = f^k(r_1, \dots r_{d-1}, x_d)$
  Verifier checks $g_d(r_d) = f^k(r_1, r_2, \dots r_d)$



# Multi-Round Frequency Moments

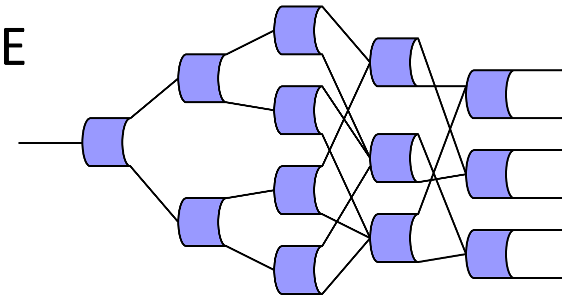
- Correctness: prover can't cheat last round without knowing $r_d$
- Then can't cheat round $i$ without knowing $r_i$…
  - Similar to protocols from "traditional" Interactive Proofs
- Inductive proof, conditioned on each later round succeeding
- Bounds: $O(k^2 \log N)$ total communication, $O(k \log N)$ space
- V's incremental computation possible in small space, via

$$\prod_{j=1}^{d} (r_j + \mathrm{bit}(j,i)(1-2r_j))$$

- Intermediate polynomials relatively cheap for helper to find

# General Computations

- Want to be able to solve more general computations
- Framework: “Interactive Proofs for Muggles”, STOC’08
  Goldwasser, Kalai, Rothblum [GKR08]
- Idea: computations modeled by arithmetic circuits
  - Arranged into layers of addition and multiplication gates
- (Super)Round $i$: Prover claims value of LDE of layer $i$ at $r_i$
  Run multiround IP to reduce to a claim about layer $i\text{-}1$ at $r_{i-1}$
- Start with claimed output, end with LDE of input
  - Verifier can check against own calculated LDE



# Putting GKR08 into practice

- Verifier needs an LDE of the "wiring polynomial" of the circuit
  - E.g. add(a, b, c) = 1 iff gate a at layer i has inputs b, c from layer i-1
  - Looks costly to evaluate directly, need to sum LDE over $n^3$ values?
  - Use the multilinear extension of the add() and mult() polynomials
  - Each gate contributes one term to the sum, so linear in circuit size
- Linear in circuit size is still slow – same as evaluating the circuit!
  - Take advantage of regularity in common wiring patterns
  - E.g. binary tree: compute contribution of all gates at once
  - Also holds for circuits for FFT, Matrix multiplication etc.

# Engineering GKR08

- Include some “shortcut” gates in addition to add, mult
  - Wide-sum $\bigoplus$ : add up a large number of inputs
    - Only needs a single sum-check protocol
  - Exponentiation: raise to a constant power ($x^8$, $x^{16}$)
    - More efficient than repeated self-multiplication
- Choose the right field size for computations
  - Work modulo a large Mersenne prime allows efficient arithmetic

# Experimental Results

| Problem | Gates | Size (gates) | P time | V time | Rounds | Comm |
|---|---|---|---|---|---|---|
| $F_2$ | $+, \times$ | 0.4M | 8.5 s | .01 s | 986 | 11.5 KB |
| $F_2$ | $+, \times, \oplus$ | 0.2M | 6.5 s | .01 s | 118 | 2.5 KB |
| $F_0$ | $+, \times$ | 16M | 552.6 s | .01 s | 3730 | 87.4 KB |
| $F_0$ | $+, \times, x^8, \oplus$ | 8.2M | 432.6 s | .01 s | 1310 | 51.0 KB |
| $F_0$ | $+, \times, x^{16}, \oplus$ | 6.2M | 441.2 s | .01 s | 1024 | 56.8 KB |
| PMwW | $+, \times, x^8, \oplus$ | 9.6M | 482.2 s | .01 s | 1513 | 56.1 KB |

- (Relatively) efficient results for frequency moments, pattern matching with wildcards (PMwW)

# Further Recent Enhancements

- Prover's work is data parallel: can take use of GPU for acceleration [Thaler et al. HotCloud 2012]
- Further tricks shave log factors off prover's effort [Thaler, Crypto 2013]
- Reduce dependency on domain size when data is sparse [Chakrabarti et al., 2013]
- Use crypto tools to handle three party model (data owner, server, clients) [Cormode et al., SIGMOD 2013]

# Open Questions

- Lower bounds for multi-round versions of the protocols
  - May need new communication complexity models
- Characterize problems that can be solved in this model
  - NP is known to be solvable with H = poly(N), V = log N [Lipton 90]
  - But we want H=O(N), and ideally H=o(N)
- Use these protocols
  - Protocols seem practical, but are they compelling?
  - For what problems are protocols most needed?