Trust the Cloud with Practical Interactive Proofs

Graham Cormode  
G.Cormode@warwick.ac.uk

Amit Chakrabarti (Dartmouth)  
Andrew McGregor (U Mass Amherst)  
Michael Mitzenmacher (Harvard)  
Justin Thaler (Harvard/Yahoo!)  
Suresh Venkatasubramanian (Utah)
There are no guarantees in life

“WE... MAKE NO REPRESENTATIONS OF ANY KIND... THAT THE SERVICE OR THIRD PARTY CONTENT WILL BE UNINTERRUPTED, ERROR FREE OR FREE OF HARMFUL COMPONENTS, OR THAT ANY CONTENT... WILL BE SECURE OR NOT OTHERWISE LOST OR DAMAGED.”

- From the terms of service of a certain cloud computing service...
- Can we obtain guarantees of correctness of the computation?
  - Without repeating the computation?
  - Without storing all the input?

YES!
Interactive Proofs

What’s the answer?
42
Prove it!

1010101001000110110101100010001

110101?

11010010001000110101010010001101

OK!
Albert and Bernard just become friends with Cheryl, and they want to know when her birthday is. Cheryl gives them a list of 10 possible dates.

May 15  May 16  May 19
June 17  June 18
July 14  July 16
August 14 August 15 August 17

Cheryl then tells Albert and Bernard separately the month and the day of her birthday respectively.

Albert: I don’t know when Cheryl’s birthday is, but I know that Bernard does not know too.
Bernard: At first I don’t know when Cheryl’s birthday is, but I know now.
Albert: Then I also know when Cheryl’s birthday is.

So when is Cheryl’s birthday?
**Streaming Interactive Proofs**

- Two party-model: **outsource** to a more powerful “prover”
  - Fundamental problem: how to be sure that the prover is honest?
- Prover provides “proof” of the correct answer
  - Ensure that “verifier” has very low probability of being fooled
  - Measure resources of the participants, rounds of interaction
  - Related to communication complexity Arthur-Merlin model, and Algebrization, with additional streaming constraints
Starter Problem: Index

- Fundamental (hard) problem in data streams
  - Input is a length $m$ binary string $x$ followed by index $y$
  - Desired output is $x[y]$
  - Requires $\Omega(m)$ space even allowing error probability

- Can we find a protocol to allow recovery of arbitrary bits
  - Without having the verifier store the entire sequence?
Real problem: Nearest neighbor

Data set: points $X$
from metric space $M$

$\otimes$: Query point $y$

$NN(y, X)$
Parameters

- \( m \) data points (\( m \) very large)
  - Verifier \( V \) processes data using small space \( << m \)
  - Prover \( P \) processes data using space at least \( m \)

- \( V \) and \( P \) have a conversation to determine the answer
  - If \( P \) is honest, 0.99 probability that \( V \) accepts the answer
  - If \( P \) is dishonest, 0.99 probability that \( V \) rejects the answer
  - Measure the space used by \( V, P \), communication used by both

\[ \text{Data Stream} \]

- Space \( v \)
- Space \( p \)
- Communication \( h \)

"Proof"
Index: 1 Round Upper Bound

- Divide the bit string into blocks of $H$ bits
- Verifier remembers a hash on each block
- After seeing index, Prover replays its block
- Verifier checks hash agrees, and outputs $x[y]$

**Cost:** $H$ bits of proof from the prover, $V = m/H$ hashes

- So $HV = O(m \log m)$, any point on tradeoff is possible
2 Round Index Protocol

Data indexed in Boolean hypercube $\{0,1\}^b$

Extended to hypercube $F^b$

Challenge line $\lambda$

Random point $r \in F^b$

1. $V$ picks $r$ and evaluates low-degree extension of input at $r$ to get $q$
2. $V$ sends $\lambda$ to $P$
3. $P$ sends polynomial $p'$ which is input restricted to $\lambda$
4. $V$ checks that $p'(r) = q$, and outputs $p'(y)$
Streaming LDE Computation

- Given query point $r \in F^b$, evaluate extension of input at $r$
- Initialize: $z = 0$
- Update with impact of each data point $y=(y_1, \ldots, y_b)$ in turn. Structure of polynomial means update causes

$$z \leftarrow z + \prod_{i=1}^{b} ((1-y_i)(1-r_i) + y_i r_i)$$

- Lagrange polynomial, can be evaluated in small space

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

- **Correctness of the protocol**
  - If $P$ is honest: $V$ will always accept
  - If $P$ is dishonest: $V$ only accepts if $p'(r) = q$
    This happens with probability $b/|F|$: can make $|F|$ bigger

- **Costs of the protocol**
  - $V$’s space: $O(b \log |F|) = O(\log n \log \log n)$ bits
  - $P$ and $V$ exchange $\lambda$ and $p'$ as $(b + 1)$ values in $F$,
    so communication cost is $O(\log n \log \log n)$ bits
  - Exponential improvement over one round

- **Consequences**: can do other computations via Index e.g. median
  - What about more complex functions?
Nearest Neighbour Search

- **Basic idea**: convert NNS into an (enormous) index problem
  - Work with input points in \([n]^d\)
  - Assume all distances are multiples of \(\varepsilon = 1/n^d\)
- Let \(B = \{\text{all distinct balls}\}\); note \(|B| \leq n^{2d}\)
  - Convert input points to virtual set of balls from \(B\):
    - point \(x\) → all balls \(\beta\) such that \(x \in \beta\)
- \(V\) processes virtual stream \(\sigma\) through index protocol
- For query \(y \in X\), \(P\) specifies point \(z \in X\), claiming \(z = \text{NN}(y,X)\)
  - Show \(\text{ball}(z,0) \in \sigma\) via Index Protocol
  - And \(\text{ball}(z, \text{dist}(y, z)-\varepsilon) \notin \sigma\) via Index Protocol
- Protocol allows correct demonstration of nearest neighbour
- **Drawback**: blow-up of input size costs \(V\) a lot!
Practical Proof Protocol

- Exploit structure of the metric space containing the points
  - Let $\Phi(\beta, x)$ be the function that reports 1 iff $x$ is in ball $\beta$
  - **Goal:** query the vector $v[\beta] = \sum_{x \in \text{input}} \Phi(\beta, x)$
  - $\Phi(\beta, x)$ has a simple **circuit** for common metrics (Hamming, $L_1$, $L_2$...)
  - “Arithmetize” the formula to compute distances

- Transform formula $\Phi$ to polynomial $\Phi'$ via
  $$G_1 \land G_2 \mapsto G'_1 G'_2 \quad \text{and} \quad G_1 \lor G_2 \mapsto 1-(1-G'_1)(1-G'_2)$$

- **Low-degree extension** of $v$: $v'(B_1 \ldots B_{2d \log n}) = \sum_x \Phi'(B_1 \ldots B_{2d \log n}, x)$
  - Can then apply Index protocol to $v'$ — $v$ never materialized by $P$ or $V$

- **Final costs of the protocol:**
  - Verifier can process each data point in time $\text{poly}(d, \log n)$
  - Communication cost and verifier space both $\text{poly}(d, \log m, \log n)$ bits
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 $i-1$ at $r_{i-1}$

Start with claimed output, end with LDE of input
  - Verifier can check against own calculated LDE
  - $V$ only needs LDE + compact circuit description
**Primitive: Sum Check Protocol**

Hold prover to a claim about a (low-degree) function of the input Index data using \(\{0,1\}^b\) in \(b = \log N\) dimensional space

- **Verifier picks one** \((r_1 \ldots r_b) \in \mathbb{F}^b\), and evaluates \(f^k(r_1, r_2, \ldots r_b)\)
- **Round 1:** Prover sends \(g_1(x_1) = \sum_{x_2 \ldots x_b} f^k(x_1, x_2 \ldots x_b)\), V sends \(r_1\)
- **Round i:** Prover sends \(g_i(x_i) = \sum_{x_{i+1} \ldots x_b} f^k(r_1, r_2 \ldots r_{i-1}, x_i, x_{i+1} \ldots x_b)\)
  Verifier checks \(g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)\), sends \(r_i\)
- **Round d:** Prover sends \(g_b(x_b) = f^k(r_1, \ldots r_{d-1}, x_b)\)
  Verifier checks \(g_b(r_b) = f^k(r_1, r_2, \ldots r_b) = \text{the LDE}\)
Putting GKR08 into practice

- Verifier needs LDEs of “wiring polynomials” of the circuit
  - E.g. $\text{add}(a, b, c) = 1$ 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 $\text{add}()$ and $\text{mult}()$ polynomials
  - Each gate contributes one term to the sum, so linear in circuit size
  - Going layer to layer keeps degree low: ensures probability holds

- 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.
Include some “shortcut” gates in addition to `add`, `mult`
- Wide-sum $\oplus$: 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

**(Relatively) efficient results for frequency moments \((F_2, F_0)\), pattern matching with wildcards \((PMwW)\)**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Gates</th>
<th>Size (gates)</th>
<th>P time</th>
<th>V time</th>
<th>Rounds</th>
<th>Comm</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F_2)</td>
<td>+, ×</td>
<td>0.4M</td>
<td>8.5 s</td>
<td>.01 s</td>
<td>986</td>
<td>11.5 KB</td>
</tr>
<tr>
<td>(F_2)</td>
<td>+, ×, ⊕</td>
<td>0.2M</td>
<td>6.5 s</td>
<td>.01 s</td>
<td>118</td>
<td>2.5 KB</td>
</tr>
<tr>
<td>(F_0)</td>
<td>+, ×</td>
<td>16M</td>
<td>552.6 s</td>
<td>.01 s</td>
<td>3730</td>
<td>87.4 KB</td>
</tr>
<tr>
<td>(F_0)</td>
<td>+, ×, (x^8), ⊕</td>
<td>8.2M</td>
<td>432.6 s</td>
<td>.01 s</td>
<td>1310</td>
<td>51.0 KB</td>
</tr>
<tr>
<td>(F_0)</td>
<td>+, ×, (x^{16}), ⊕</td>
<td>6.2M</td>
<td>441.2 s</td>
<td>.01 s</td>
<td>1024</td>
<td>56.8 KB</td>
</tr>
<tr>
<td>PMwW</td>
<td>+, ×, (x^8), ⊕</td>
<td>9.6M</td>
<td>482.2 s</td>
<td>.01 s</td>
<td>1513</td>
<td>56.1 KB</td>
</tr>
</tbody>
</table>
Concluding Remarks

- These protocols are truly practical
  - No, really, they are
- Also provide insight into the theory of Arthur-Merlin communication games

- Many open problems around this area
  - Extend to other data mining/machine learning problems
  - Prove lower bounds: some problems are hard
  - Evaluations on real data, optimization of implementations
  - Variant models: power of two provers...
Connections

- Full papers and connection to computational complexity

- **Streaming Computation**: what can be computed in one pass?
  - Typical focus on statistical security (no crypto assumptions)

- **Fully Homomorphic Encryption**: crypto security allows repetition
  - Make use of circuit garbling to provide secrecy as well as integrity

- **Argument Systems** (ginger/pepper/zaatar/pinnochio)
  - Need hefty preprocessing of computation

- **Other directions**: multiple provers, ZKP, public verifiability
Related Directions

- 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]
- Three party model (data owner, server, clients) [Cormode et al., SIGMOD 2013]