Verifying Computations with Streaming Interactive Proofs

Justin Thaler, Harvard University

Joint work with:

Graham Cormode (AT&T Labs – Research),

Ke Yi (Hong Kong University of Science and Technology)

Outsourcing

- Many applications require outsourcing computation to untrusted service providers.
  - Main motivation: Commercial cloud computing services.
  - Also, weak peripheral devices; fast but faulty co-processors.
  - Volunteer Computing (SETI@home, World Community Grid, etc.)

- User requires a guarantee that the cloud performed the computation correctly.

- One solution: require cloud to *prove* correctness of answer.

Goals of Verifiable Computation

- Provide user with a correctness guarantee, without requiring her to perform the requested computations herself.
  - Ideally user will not even maintain a local copy of the data.
  - User may have resorted to the cloud in the first place because she has more data than she can store.

- Minimize the amount of extra bookkeeping the cloud has to do to prove the integrity of the computation.

- Ideally our protocols will be secure against arbitrarily malicious clouds, but sufficiently lightweight for use in more benign settings.

Interactive Proofs

- Two Parties: Prover P and Verifier V.

- Think of P and powerful, V as weak. P solves a problem, tells V the answer.
  - Then P and V have a conversation.
  - P's goal: convince V the answer is correct.

- Requirements:
  - 1. Completeness: An honest P can convince V she's telling the truth.
  - 2. Soundness: V will catch a lying P with high probability no matter what P says to try to convince V (secure even if P is computationally unbounded).

Comparison to Standard Database Outsourcing Model

- There is a large body of work on *authenticating queries on outsourced databases* e.g. [HIM02, GTTC03, NT05,YPPK08, PYP09,YLCHKS09, …]

- In this model, there are three parties:
  1. A *data owner* who outsources work to:
  2. An untrusted *service provider*, who answers queries from:
  3. *Clients*.

Comparison to Standard Database Outsourcing Model

- There is a large body of work on *authenticating queries on outsourced databases* e.g. [HIM02, GTTC03, NT05, YPPK08, PYP09, YLCHKS09, …]

- In this model, there are three parties:
  1. A *data owner* who outsources work to:
  2. An untrusted *service provider*, who answers queries from:
  3. *Clients*.

- Goal: enable clients to verify correctness of query results returned by service provider.
  - Many existing solutions rely on the data owner *signing* the data set (e.g. Merkle Trees).
    - So only secure against computationally bounded service providers.
    - And most require data owner to retain a copy of the data.

- In comparison, the Interactive Proof model views the data owner and clients as a single entity.

Interactive Proofs

- IPs have revolutionized complexity theory in the last 25 years.
  - IP=PSPACE [Shamir 90].
  - PCP Theorem e.g. [AS 98]. Hardness of approximation.
  - Zero Knowledge Proofs.

- But IPs have had very little impact in real delegation scenarios.
  - Why?
  - Not due to lack of applications!

Interactive Proofs

- Old Answer: Most results on IPs dealt with hard problems, needed P to be too powerful.
  - But recent constructions focus on "easy" problems (e.g. "Interactive Proofs for Muggles" [GKR08]).
  - Allows V to run **very** quickly, so outsourcing is useful even though problems are "easy".
  - P does not need "much" more time to prove correctness than she does to solve the problem in the first place!

Interactive Proofs

- Why does GKR not yield a practical protocol out of the box?
  - Problem 1: Naively, V has to retain the full input.
  - Problem 2: P has to do a lot of extra bookkeeping (**cubic** blowup in runtime).
- Main focus of this work is addressing Problem 1. Can we allow V to be *streaming*?
- Follow-up work addresses Problem 2 in a general-purpose manner [CMT12, TRMP12].

Streaming Interactive Proofs: The Model

Data Streaming Model

- Stream: m elements from universe of size n
  - e.g., $S = \langle x_1, x_2, \ldots, x_m \rangle = 3,5,3,7,5,4,8,7,5,4,8,6,3,2, \ldots$

- Goal: Compute a function of stream, e.g., median, number of distinct elements, frequency moments, heavy hitters.

- Challenge:

  (i) Limited working memory, i.e., sublinear(n,m).

  (ii) Sequential access to adversarially ordered data.

  (iii) Process each update quickly.

Slide derived from [McGregor 10]

Models

- Prior work [CCM09/CCMT12, CMT10] introduced a more restrictive model for verifying streaming computations.
  - One message (non-interactive) protocols: P and V both observe stream. Afterward, P sends V an email with the answer, and a proof attached.

- Our model: Allow multiple rounds of interaction, i.e. P and V have a *conversation* after both observe stream.

AMERICA'S NEXT top model

Costs in Our Model

- Two main costs: words of communication h and V's working memory v.
  - We refer to (h, v)-protocols.
- Other costs: running time, number of messages.

Comparison of Two Models

- Pros of multi-round model:
  1. Exponentially reduces space and communication cost. Often (polylog n, polylog n) compared to ($\sqrt{n}$, $\sqrt{n}$).
  2. P often much faster than in single-round case.

- Cons of multi-round model:
  1. P must do significant computation *after each message*.
  2. More coordination needed; network latency might be an issue.

- Pros of single-message model:
  1. Space and communication still reasonable (< 1 MB).
  2. P can do all computation at once, just send an email with proof attached.

Streaming Interactive Proof Protocols

A Two-Pronged Approach

- Ideal: General purpose protocol allowing to verify arbitrary computation.
  - Based on general-purpose "Interactive Proofs for Muggles" construction [GKR08].

- Substantially improve on the GKR protocol for specific important problems.

A Two-Pronged Approach

- Ideal: General purpose protocol allowing to verify arbitrary computation.
  - Based on general-purpose "Interactive Proofs for Muggles" construction [GKR08].

- Substantially improve on the GKR protocol for specific important problems.
  - Reporting queries.
    - INDEX: What value is stored in memory location x of my database?
    - Range queries: List all employees whose income falls in a given range.
  - Aggregation queries.
    - Frequency Moments.
    - Inner Product
    - Distinct elements.
    - Range Sum.
    - Etc.

Prong 1: General-Purpose Result

- The GKR protocol can be modified to allow V to be streaming.
  - Reason: GKR protocol (and several others) only requires V to store a fingerprint of the data.
  - This fingerprint can be computed in a single, light-weight streaming pass over the input.
  - Fingerprint serves as a "secret" that V can use to catch the cloud in a lie.
- Fits cloud computing well: pass by V can occur while uploading data to cloud.
- V never needs to store entirety of data!
- The fingerprint is a few KBs in size, even if the input contains terabytes of data.

Prong 1: General-Purpose Result

- Theorem 1 ([GKR08] + previous slide):

(polylog n, polylog n) protocols for all problems in log-space uniform NC.

  - That is, any problem with an efficient parallel algorithm.
  - E.g. Median, MST, Determinant.

- Theorem 2 ([Kilian92] + previous slide):
  (polylog n, polylog n) *computationally sound* protocols for all problems in NP.

Prong 2: Special-Purpose protocols

- Despite powerful generality, [GKR08] is not optimal for many functions of high interest in streaming and database processing.

- We give improved protocols for these problems.
  - And argue that they are highly practical.

F_2 protocol

- Result: (log n, log n)-protocol requiring log n rounds.

- Moreover, we make P run in O(n) time.

F_2 protocol

- Result: (log n, log n)-protocol requiring log n rounds.

- Moreover, we make P run in O(n) time.

- [GKR08] yields ($\log^2 n$, $\log^2 n$) protocol requiring $\log^2 n$ rounds. P runs in $\Omega(n^3 \log n)$ time.

- [CCM09/CCMT12] shows that $\sqrt{n}$ space or communication is needed by any one-message protocol.
  - Exponential separation between one-message and multi-round models.

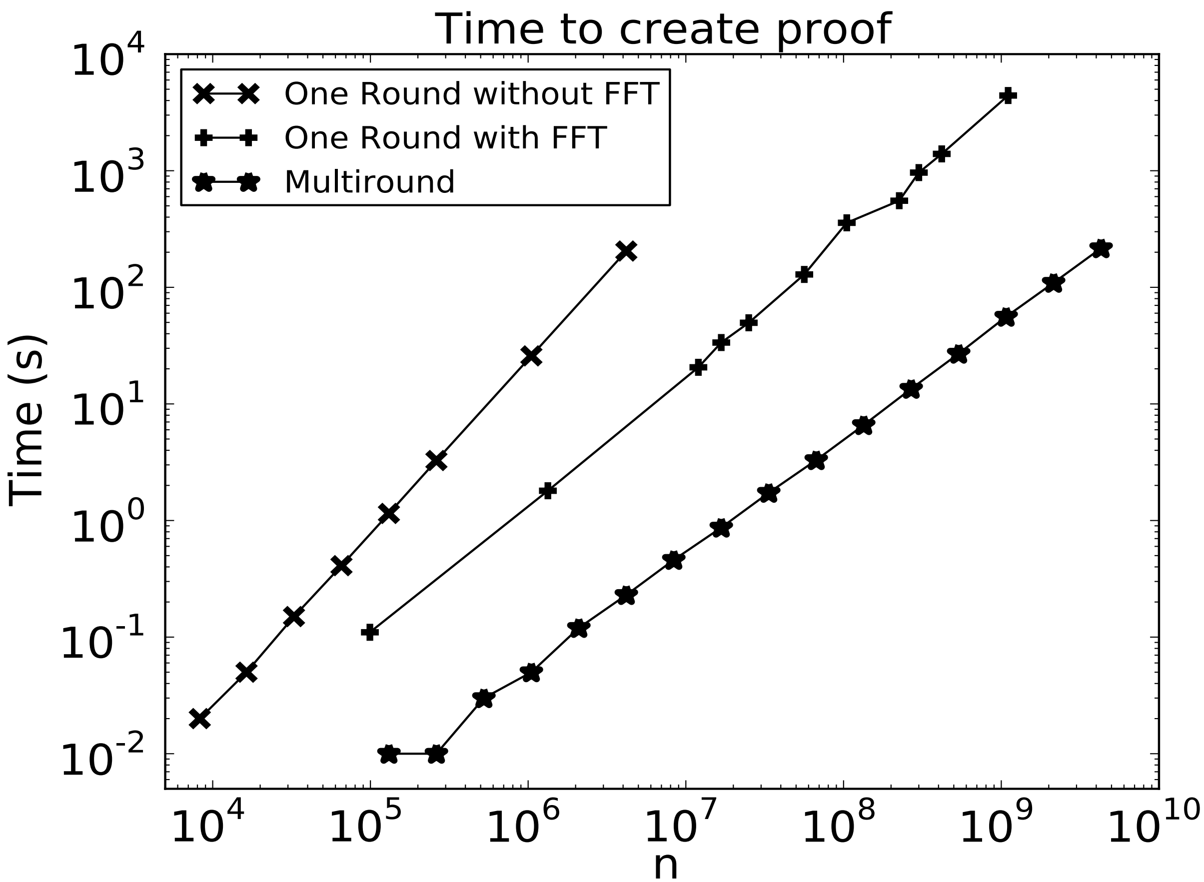
F_2 Experiments

- Implemented ($\sqrt{n}$, $\sqrt{n}$) one-message F_2 protocol from [CCM09] and our new (log n, log n) multi-round protocol.
  - One-message space and communication both ~ 1 MB for n=10 billion.
  - Multi-round space and communication always under 1 KB even when handling GBs of data.
- V highly efficient in both cases (20-40 million updates per second across all stream lengths).
- P much more efficient in multi-round case.

F_2 Experiments

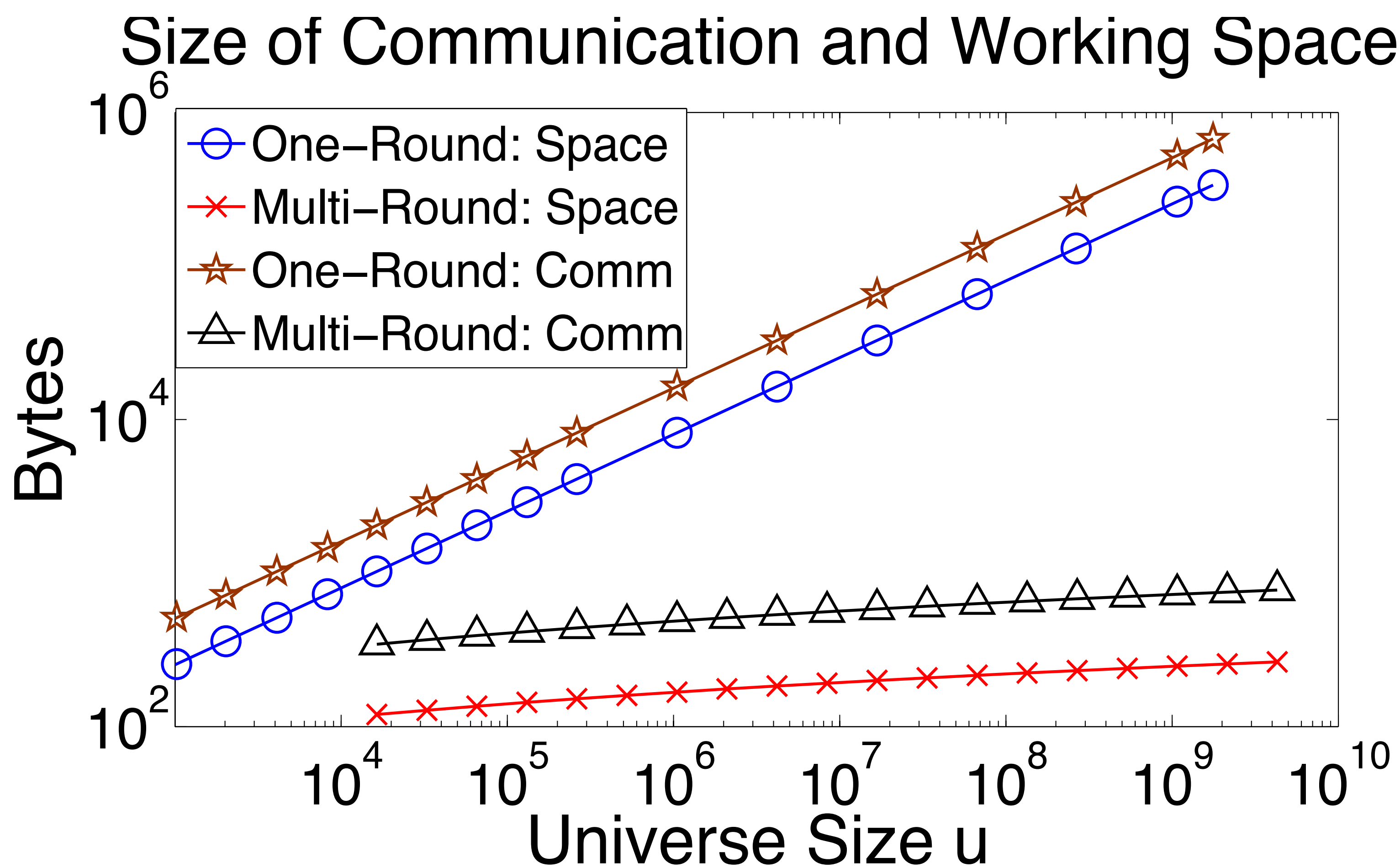
- P much more efficient in our multi-round protocol.
  - Multi-round case: P processes 20 million updates per second across all stream lengths.
  - Single-round case:
    1. Naïve implementation of P requires $\Omega(n^{3/2})$ time; doesn't scale to large streams.
    2. Follow-up work [CMT12] brings P's runtime down to O(n log n) using sophisticated FFT techniques, achieving 250,000-750,000 updates per second experimentally.

F_2 Experiments: P runtime



Multi-round P vs. Single-round P with and without FFT techniques

F_2 Experiments: Space & Communication



Range-Query Protocol + Experiments

- Result: (k+log n, log n)-protocol requiring log n rounds, where k is the number of items returned by the query.

- Moreover, we make P run in O(n) time.

- All experimental costs similar to those of F_2 protocol.

Range-Query Protocol Ideas

- Standard idea: have V keep a Merkle tree, so that the hash of the root is used as a “secret” to catch P in a lie.
  - Though this would only be secure against computationally bounded provers.
- But V cannot compute the hash of the root without storing the entire tree!

Range-Query Protocol Ideas

- Standard idea: have V keep a Merkle tree, so that the hash of the root is used as a "secret" to catch P in a lie.
  - Though this would only be secure against computationally bounded provers.
- But V cannot compute the hash of the root without storing the entire tree!
- We use a different hashing scheme that is similar in outline to a Merkle tree, but that can be computed incrementally by V as the stream updates arrive in arbitrary order.
  - To "cheat", P would have to find collisions under this hash function.
  - But P does not learn the hash function until she has already committed to an answer.
- Remaining engineering challenge: make P fast.

Conclusions

- IPs (and their relatives) represent some of the most celebrated results in complexity theory.
- They have the potential to mitigate trust issues in cloud computing, but were wildly impractical until recently.
- We modify known constructions to work with streaming verifiers.
- And improve on known constructions for specific, important problems.
  - Arguably obtaining the first practical interactive proof protocols.

Follow-up Work

- [CMT12] revisits the GKR protocol.
  - Brings the blowup in P's runtime down from **cubic** to logarithmic.
  - Develops a full, working implementation of the GKR protocol.
  - Demonstrates experimentally that V saves a lot of time and space (at least for problems with small-depth circuits).
  - The main remaining bottleneck is still P's runtime (P takes 27 minutes for 256 x 256 matrix multiplication).

- [TRMP12] describes a parallelized implementation of the GKR protocol that further reduces P's and V's runtimes by 40x-100x.

- Other recent general-purpose implementation work: [CRR11, SMBW12, SVPBBW12].

Thank you!

Second Frequency Moment

- The second frequency moment of a stream is defined as follows:
  - Let **X** be the frequency vector of the stream ($\mathbf{X}_i$ is number of occurrences of i in the stream)
  - $F_2(\mathbf{X})=\sum_i \mathbf{X}_i^2$

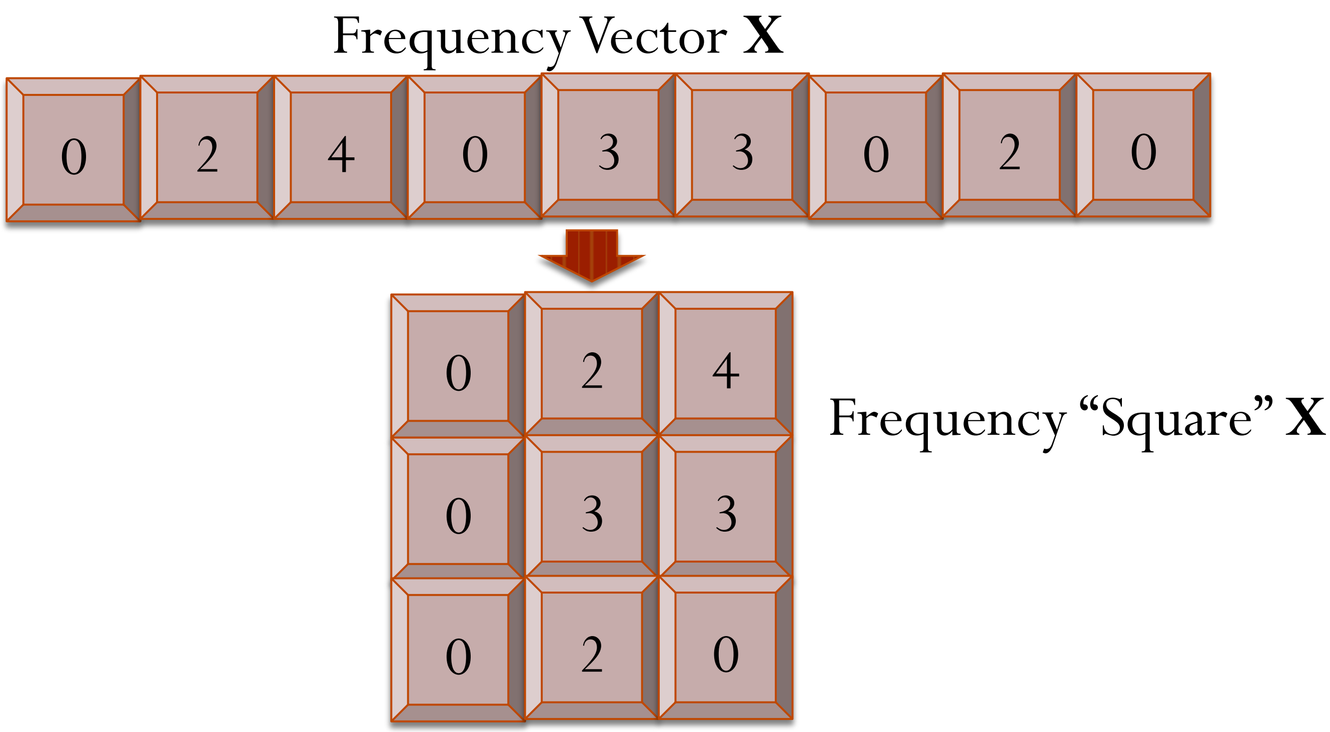
- [CCM 09/CCMT 12] ($\sqrt{n}$, $\sqrt{n}$)-protocol for F_2.
  - Terabytes of data translate to a few MBs of space and communication.

- This is optimal. There is a lower bound that says for (h, v)-protocol for F_2, $hv=\Omega(n)$ lower bound.

- Notice (1, n) and (n, 1) protocols are trivial. What is non-obvious is how to trade off between h and v.

F_2 Protocol

- Recall: $F_2(\mathbf{X})=\sum_i \mathbf{X}_i^2$
- View universe [n] as [√n] x [√n].



- First idea: Have P send the answer "in pieces":
  - F_2(row 1). F_2(row 2). And so on. Requires $\sqrt{n}$ communication.
- V exactly tracks a row at random (denoted in yellow) so if P lies about any piece, V has a chance of catching her. Requires space $\sqrt{n}$.

Frequency Square **X**

			P sends
0	2	4	$20=2^2+4^2$
0	3	3	$18=3^2+3^2$
0	2	0	$4=2^2$

Slide derived from [McGregor 10]

- Problem: If P lies in only one place, V has small chance of catching her.

- We would like the following to hold: if P lies about even one piece, she will have to lie about many.

- Solution: Have P commit (succinctly) to second frequency moment of rows of an **error-corrected encoding** of the input.

- Need V to evaluate any row of the encoding in a streaming fashion. Can do this for "low-degree extension" code. Note: this code is *systematic*, meaning the first n symbols are just the input itself.

Error-corrected Encoding of Frequency Square X

Input is embedded in encoding (low-degree extension)

0	2	4
0	3	3
0	2	0
0	-1	-5
0	-6	-12
0	-13	-21

These values will all lie on low-degree polynomial s(X)

H sends

$20=2^2+4^2$

$18=3^2+3^2$

$4=2^2$

$26=(-1)^2+(-5)^2$

$180=(-6)^2+(-12)^2$

$610=(-13)^2+(-21)^2$

Multi-Round Protocol

- Replace "frequency square" with "frequency hypercube" i.e. view universe [n] as $[2]^d$ where d=log n.
- V's secret is now a *single* entry of the (encoded) frequency hypercube, rather than an entire row of the frequency square.
  - Requires space O(log n) rather than space O(√n).
- In Round 1, P sends the answer "in pieces", where piece j aggregates over all items of the form $i=(j, i_2, i_3, \ldots, i_d)$.
  - Then V tells P the first coordinate of her secret index, and the protocol iterates on the resulting subcube.
- Analysis: argue that if P sends a "wrong" polynomial in any round, then P will have to send a wrong polynomial in all subsequent rounds.