Invertible Bloom Lookup Tables and Applications

Michael Mitzenmacher
Joint work with
Michael Goodrich, Rasmus Pagh,
George Varghese
Stragglers Problem

• Consider data streams that insert and delete many items, such that most items inserted are subsequently deleted.

• Problem: Identify the stragglers

• Examples:
  – Flows through a router
  – people entering and leaving a building.
Stragglers Problem

- We want listing not at all times, but at “reasonable” or “off-peak” times, when the current working set size is bounded.
  - If we do all the N insertions, then all the N-M deletions, and want a list at the end, we want…

- Data structure size should be proportional to **listing size**, not maximum size.
  - Proportional to M, not to N!
  - Proportional to size you want to be able to list, not number of pairs your system has to handle.
Set Reconciliation Problem

- Alice and Bob each hold a set of keys/values, with a large overlap.
  - Example: Alice is your smartphone phone book, Bob is your desktop phone book, and new entries or changes need to be synched.
  - Example: Distributed databases.

- Want one/both parties to learn the (signed) set difference.

- Goal: communication is proportional to the size of the difference, not the size of the sets.
Invertible Bloom Lookup Tables Functionality

• IBLT operations
  – Insert (k,v)
  – Delete (k,v)
  – Get(k)
    • Returns value for k if one exists, null otherwise
    • Might return “not found” with some probability
– ListEntries()
  • Lists all current key-value pairs
  • Succeeds as long as **current** load is not too high
    – Design threshold

Possible Scenarios

• A **good** system
  – Each key has (at most) 1 value
  – Delete only items that are inserted

• A **bad** system
  – Deletions might happen for keys not inserted, or for the wrong value

• An **ugly** system
  – Key-value pairs might be duplicated
The Good System

(k,v) pair hashed to j cells

<table>
<thead>
<tr>
<th>Count</th>
<th>KeySum</th>
<th>ValueSum</th>
</tr>
</thead>
</table>

**Get:**
- If Count = 0 in any cell, return null
- Else, if Count = 1 in any cell and KeySum = key, return ValueSum
- Else, return Not Found

**Insert:**
- Increase Count,
- Update KeySum and ValueSum

**Delete:**
- Decrease Count,
- Update KeySum and ValueSum

SUM by XOR
The Good System : Listing

- **While** some cell has a count of 1:
  - Set \((k,v) = (\text{KeySum},\text{ValueSum})\) of that cell
  - Output \((k,v)\)
  - Call Delete\((k,v)\) on the IBLT
Listing Example

```
  3

  3

  1

  2

  2

  2

  3

  4

  0

  1
```
Possible Collision Scenario

No cell has only 1 entry, so listing terminates with an incomplete set of items listed.
The Good System : Listing Analysis

• **While** some cell has a count of 1:
  – Set \((k,v) = (\text{KeySum}, \text{ValueSum})\) of that cell
  – Output \((k,v)\)
  – Call Delete\((k,v)\) on the IBLT

• Peeling Process.
  – Keys = hyperedges, Cell = vertices.
  – This is peeling away degree 1 vertices on the hypergraph.
  – Same process finds the 2-core of a hypergraph.
  – Same process used to decode families of low-density parity-check codes.
Listing Performance

• Results on random peeling processes
• Theoretical thresholds for complete recovery depend on number of hash functions:

<table>
<thead>
<tr>
<th>J</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>m/n</td>
<td>1.222</td>
<td>1.295</td>
<td>1.425</td>
<td>1.570</td>
</tr>
</tbody>
</table>

• Interesting possibility: use “irregular” IBLTs
  – Different numbers of hash functions for different keys
  – Same idea used in LDPC codes
One-Sided Set Reconciliation

- Bob has a *subset* of Alice’s key-value pairs.
- Alice sends Bob an IBLT of her key-value pairs.
- Bob deletes his key-value pairs from the IBLT.
- What remains in the IBLT is the set difference, \( A - B \).
- Bob lists elements of the IBLT to find the items in the difference, \( A - B \).
- IBLT size only needs to be proportional to size of the set difference.
- Fast and space efficient solution.
Fault Tolerance: The Bad System

• Extraneous deletions
  – Now a count of 1 does not mean 1 key in the cell.
    • Might have two inserted keys + one extraneous deletion.
  – Need an additional check: hash the keys, sum into HashKeySum.
  – If count is 1, and the hash of the KeySum = HashKeySum, then 1 key in the cell.

• What about a count of -1?
  – If count is -1, and the hash of -KeySum = -HashKeySum, then 1 key in the cell.
Symmetric Set Reconciliation

• Alice and Bob each builds an IBLT, S and T, for their sets, A and B, of key-value pairs and sends these to each other or a third party.
• The receiver computes S – T in an element-wise fashion.
• The receiver perform fault-tolerant listing on this IBLT.
• Each element removed with a count of -1 belongs to the set difference B – A, and each element removed with a count of 1 belongs to A – B.
• The size of the IBLTs need only be proportional to the size of the symmetric difference.
Fault Tolerance: The Ugly System

• Keys with multiple values
  – Need another additional check; HashKeySum and HashValueSum.

• Multiply-valued keys “poison” a cell.
  – The cell is unusable; it will never have a count of 1.

• Small numbers of poisoned cells have minimal effect.
  – Usually, all other keys can still be listed.
  – If not, number of unrecovered keys is usually small, like 1.
Experiments

- Plot (a) gives the percentage of key-value pairs recovered around the threshold. Slightly over the theoretical asymptotic threshold, we obtain full recovery of all key-value pairs. Plot (b) gives the percentage of trials with incomplete recovery with “damaged” keys that have multiple values. Each data point represents the average of 20,000 simulations.
Not Just for Theory: Bitcoin

O(1) Block Propagation

The problem

Bitcoin miners want their newly-found blocks to propagate across the network as quickly as possible, because every millisecond of delay increases the chances that another block, found at about the same time, wins the "block race."

Set reconciliation and Invertible Bloom Lookup Tables

Set reconciliation means finding the difference between two sets of data. To optimize new block broadcasts, the sets we care about are the set of transactions transactions in a new block (call that set "B") and the set of transactions that a node already knows about (call that set "P").

Ideally we want to transmit just the 80-byte block header and transaction data for B-P. Both B and P will grow in size as transaction volume and the size of blocks grows, but if transaction propagation is mostly reliable B-P will be a constant size (see the section below on incentives for more discussion).

Invertible Bloom Lookup Tables (IBLTs) are a new (described in 2011) data structure that can be used to solve the set reconciliation problem.

https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2
Not Just for Theory: Bitcoin

Scaling Bitcoin: Gavin begins work on invertible bloom lookup tables on Github. (twitter.com)

Submitted 5 days ago by platonicgap
124 comments share

All 124 comments - sorted by: best

[-] packersmate 36 points 4 days ago
Someone explain for us non techies?

[-] inappropriate_cliche 22 points 4 days ago
There is currently an incentive for miners to keep blocks small, because smaller found blocks can be transmitted around the network faster than large blocks, thus increasing the chance a miner can claim the prize for their found block. Gavin's solution would make all found block announcements a fixed-size blob, removing the small-size incentive. This in turn would likely mean miners would want to include more transactions in their blocks to collect more transaction fees.

[-] dingusbuttface 6 points 4 days ago
If this is accurate, it is the BEST explanation with the least technical/math jargon.
Thank You!!
Privacy-Enhanced Set Differences

• Has applications to privacy-enhanced methods for comparing DNA sequences.

• Main idea:
  – Bob and Alice encrypt their IBLTs with an additive homomorphic encryption scheme, using a public key for the trusted third party.
    • So we have an operation, “**”, such that E(X)*E(Y)=E(X+Y).
  – Alice and/or Bob performs the set difference using the operation “**” and sends result to the third party.
  – If the sets are close, the third party can list their difference. If they are far, the listing algorithm for the trusted third party will probably fail.
Biff Code: IBLTs for Codes

• Alice has message $x_1, x_2, \ldots, x_n$.
  – Creates IBLT with ordered pairs as keys, $(x_1,1), (x_2,2), \ldots (x_n,n)$.
  – Values for the key are a checksum hash of the key.
  – Alice’s “set” is $n$ ordered pairs.
  – Sends message values and IBLT.

• Bob receives $y_1, y_2, \ldots, y_n$.
  – Bob’s set has ordered pairs $(y_1,1), (y_2,2), \ldots (y_n,n)$.
  – Bob uses IBLT to perform set reconciliation on the two sets.
Simple Code

Code is essentially a lot of hashing, XORing of values.

- **ENCODE**
  
  for $i = 1 \ldots n$ do
  for $j = 1 \ldots k$ do
    $T_j[h_j((x_i,i))].keySum = (x_i,i)$.
    $T_j[h_j((x_i,i))].valueSum = \text{Check}((x_i,i))$.
  
- **DECODE**
  
  for $i = 1 \ldots n$ do
  for $j = 1 \ldots k$ do
    $T_j[h_j((y_i,i))].keySum = (y_i,i)$.
    $T_j[h_j((y_i,i))].valueSum = \text{Check}((y_i,i))$.
  while $\exists a,j$ with $(T_j[a].keySum \neq 0)$ and $(T_j[a].valueSum == \text{Check}(T_j[a].keySum))$ do
    $(z,i) = T_j[a].keySum$
    if $z \neq y_i$ then
      set $y_i$ to $z$ when decoding terminates
  for $j = 1 \ldots k$ do
    $T_j[h_j((z,i))].keySum = (z,i)$.
    $T_j[h_j((z,i))].valueSum = \text{Check}((z,i))$. 
Result

• Fast and simple low-density parity-check code variant.
  – For correcting errors on q-ary channel, for reasonable-sized q.
• All expressed in terms of “hashing”
  – No graphs, matrices needed for coding.
• Builds on intrinsic connection between set reconciliation and coding.
  – Worth greater exploration.
• Builds on intrinsic connection between LDPC codes and hashing methods.
Extension: Multi-Party Set Reconciliation

• What if there are more than two entities, that need to reconcile?
  – Generally over a network.
  – Natural for distributed databases.

• Pairwise reconciliations possible.
  – But generally not efficient.

• IBLTs have been extended to multi-party set reconciliation.
  – Mitzenmacher/Pagh
  – Work over a “larger finite field” (instead of XORing)
Multi-Party

• Parties $A_1, A_2, A_3...$ with sets $S_1, S_2, S_3...$ of keys.
• Total set difference is
  $\left| \bigcup_j S_j - \bigcap_j S_j \right|$
• Goal: all parties at end have $\bigcup_j S_j$
• Additional goal: a party missing a value knows the ID of another party that has that value.
Multi-Party Via Relay

Looks like network coding
General Networks
Reconciliation via Gossip

• Reconciling over a network, intermediate nodes act as “relays”.
  – Combine and pass sketches.
• Use gossip algorithms (“PUSH-PULL”), but instead of passing rumors, pass sketches.
• Messages stay “fixed size”, proportional to total set difference $d$.
  – Messages are random linear combinations of IBLTs.
  – Just as in many network coding gossip algorithms messages are random linear combinations of messages.
• Time to reconcile = time for gossip algorithm to distribute all rumors.
Reconciliation Via Gossip

- Given an upper bound $d$ on the total set difference, $N$ parties on a network of $n$ vertices can reconcile sets using sketches of $O(d)$ values using a PUSH-PULL gossip algorithm, where reconciliation succeeds with high probability in $O(\Phi^{-1}\log n)$ rounds, where $\Phi$ is the conductance of the graph.
  - Operations on sketches are random linear combinations.
  - Each vertex need only maintain a “current sketch” that is a random linear combination of received sketches.
Conclusion

• Invertible Bloom Lookup Table a simple solution for reconciliation oriented problems.

• Allows insertions, deletions, listing.
  – Size proportional to target “listing size”.
  – Proportional to set difference for reconciliation problems.

• Can be made robust to various types of faults.

• New “multi-party” variations.

• Expect more uses forthcoming.