Web-Scale Information Analytics

Data Stream Processing

Prof. Wing C. Lau
Department of Information Engineering
wclau@ie.cuhk.edu.hk

Acknowledgements

- The slides used in this chapter are adapted from the following sources:
  - Stat 260 Scalable Machine Learning of UC Berkeley, by Alex Smola, CMU, http://alex.smola.org/teaching/berkeley2012/streams.html
  - Piotr Indyk, MIT, "Data Stream Algorithms," Open lectures for PhD Students in Computer Science 2012, http://phdopen.mimuw.edu.pl/index.php?page=z11w3#zal
  - Edith Cohen, Amos Fiat, Haim Kaplan, Paula Ta-Shma, Tova Milo, CS 0368.3239, Leveraging Big Data, Fall 2013/2014, TAU (Tel Aviv University) http://www.cohenwang.com/edith/bigdataclass2013
  - CS286 Implementation of Database Systems, UC Berkeley, Minos Garofalakis, Raghu Ramakrishnan, http://db.cs.berkeley.edu/cs286sp07/
  - IE3090 Advanced Networking Protocols and Systems, by D.M. Chiu, CUHK.


Data Streams

Roadmap

- Data Streams & Applications
- Data Streaming Models & Basic Mathematical Tools
- Summarization/Sketching Tools for Streams
  - Moments
    - Loglog Counting for Distinct Items via Flajolet-Martin (FM) Sketch
    - Alon-Matias-Szegedy (AMS) Sketch
  - Heavy Hitter (Frequent Item) Counting/ Detection in Streams
  - Bloom filter and Other Sketches

Streams

Website Analytics

- Continuous stream of users (tracked with cookie)
- Many sites signed up for analytics service
- Find hot links / frequent users / click probability / right now

Query Stream

- Item stream
- Find heavy hitters
- Detect trends early (e.g. Obsama bin Laden killed)
- Frequent combinations (cf. frequent items)
- Source distribution
- In real time

Network traffic analysis

- TCP/IP packets
- On switch with limited memory footprint
- Realtime analytics
- Busiest connections
- Trends
- Protocol-level data
- Distributed information gathering

Financial Time Series

- **real time prediction**
- **missing data**
- **metadata (news, quarterly reports, financial background)**

- time-stamped data stream
- multiple sources
- different time resolution

News

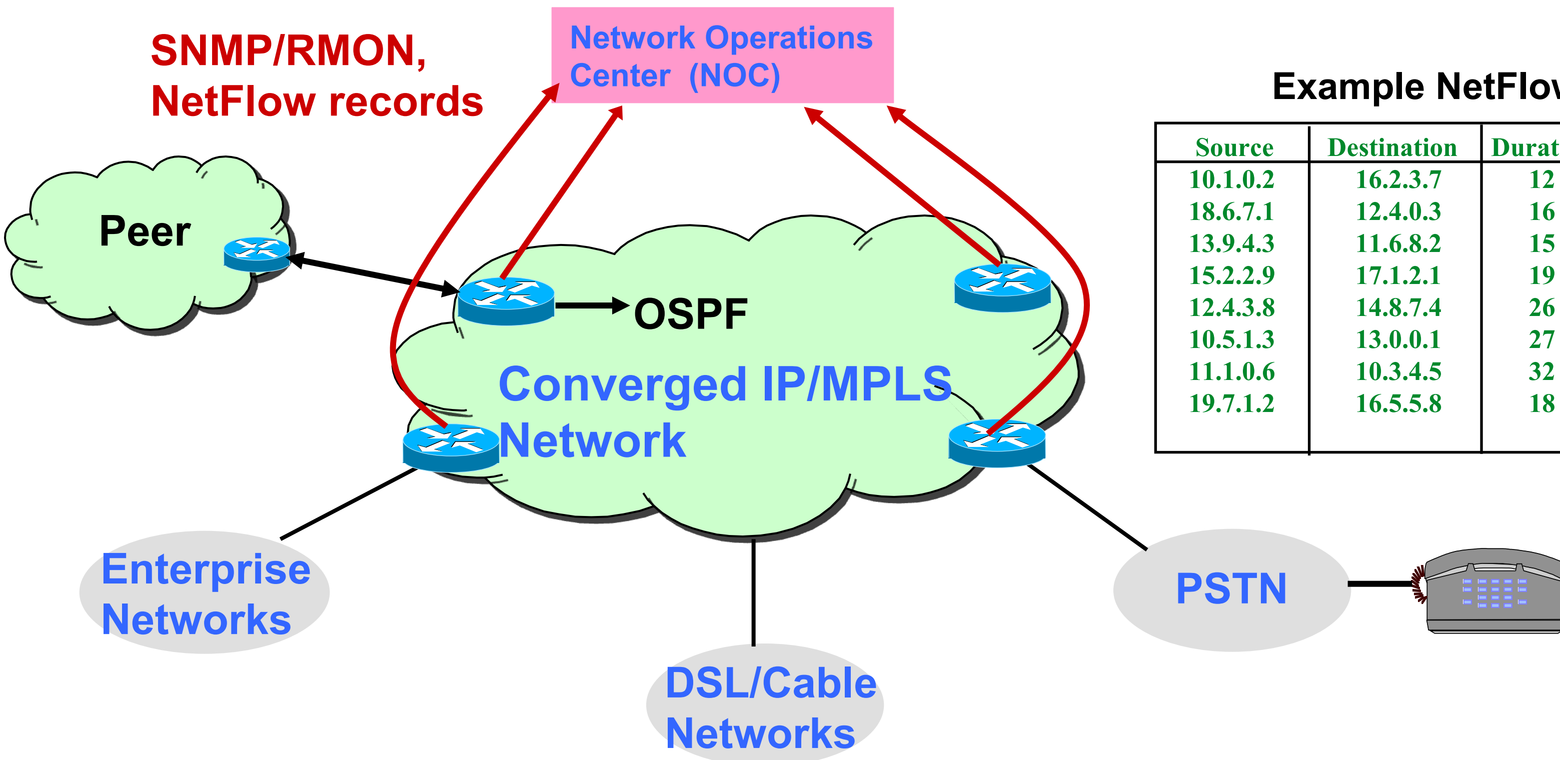
Realtime news stream

- Multiple sources (Reuters, AP, CNN, ...)
- Same story from multiple sources
- Stories are related

Data-Stream Management

- Traditional DBMS – data stored in finite, persistent data sets
- Data Streams – distributed, continuous, unbounded, rapid, time varying, noisy, . . .
- Data-Stream Management – variety of modern applications
  - Network monitoring and traffic engineering
  - Telecom call-detail records
  - Network security
  - Financial applications
  - Sensor networks
  - Manufacturing processes
  - Web logs and clickstreams
  - Massive data sets

Networks Generate Massive Data Streams



Example NetFlow IP Session Data

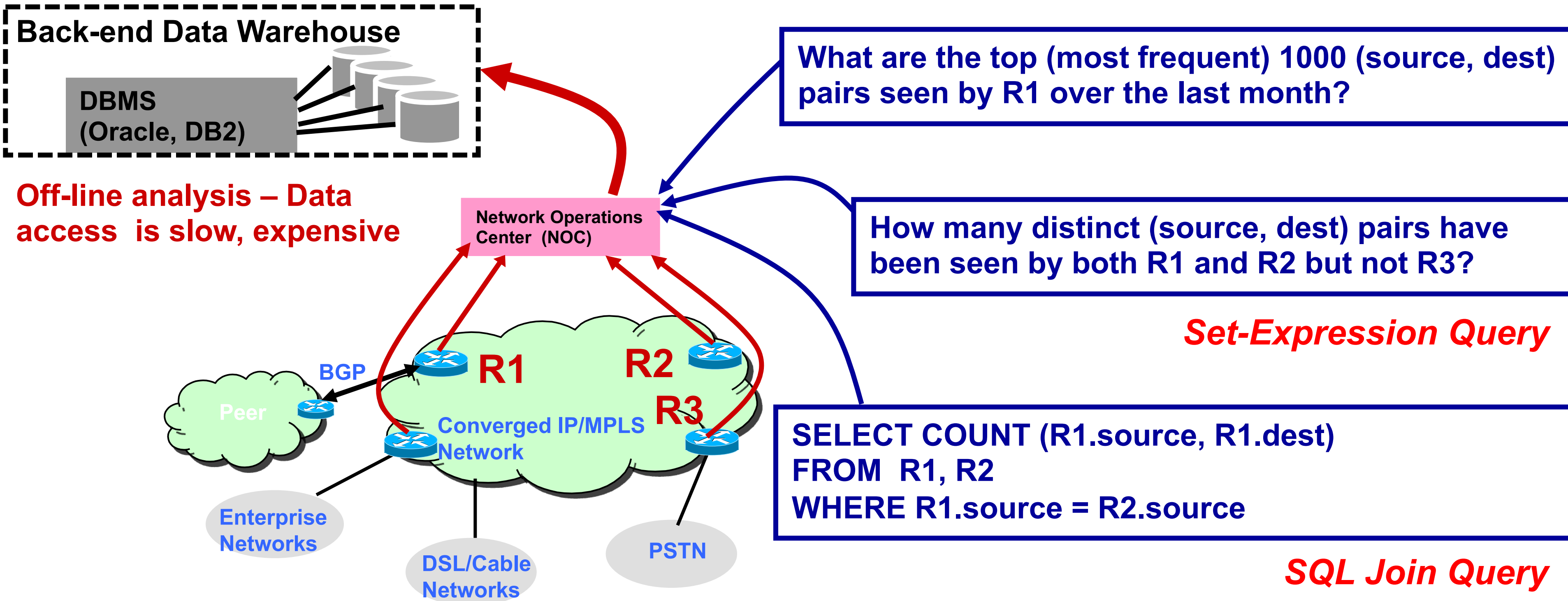
Source	Destination	Duration	Bytes	Protocol
10.1.0.2	16.2.3.7	12	20K	http
18.6.7.1	12.4.0.3	16	24K	http
13.9.4.3	11.6.8.2	15	20K	http
15.2.2.9	17.1.2.1	19	40K	http
12.4.3.8	14.8.7.4	26	58K	http
10.5.1.3	13.0.0.1	27	100K	ftp
11.1.0.6	10.3.4.5	32	300K	ftp
19.7.1.2	16.5.5.8	18	80K	ftp

- SNMP/RMON/NetFlow data records arrive 24x7 from different parts of the network
- Truly massive streams arriving at rapid rates
  - AT&T collects 600-800 GigaBytes of NetFlow data each day!
- Typically shipped to a back-end data warehouse (off site) for off-line analysis

Packet-Level Data Streams

- Single 2Gb/sec link; say avg packet size is 50bytes
- Number of packets/sec = 5 million
- Time per packet = 0.2 microsec
- If we only capture header information per packet: src/dest IP, time, no. of bytes, etc. – at least 10 Bytes.
  - Space per second is 50MB
  - Space per day is 4.5TB per link
  - ISPs typically have hundred of links!
- Analyzing packet content streams – whole different ballgame!!

Real-Time Data-Stream Analysis



- Need ability to process/analyze network-data streams *in real-time*
  - As records stream in: look at records *only once in arrival order!*
  - Within resource (CPU, memory) limitations of the NOC
- Critical to important Network Management (NM) tasks
  - Detect and react to Fraud, Denial-of-Service attacks, SLA violations
  - Real-time traffic engineering to improve load-balancing and utilization

IP Network Data Processing

- Traffic estimation
  - How many bytes were sent between a pair of IP addresses?
  - What fraction network IP addresses are active?
  - List the top 100 IP addresses in terms of traffic
- Traffic analysis
  - What is the average duration of an IP session?
  - What is the median of the number of bytes in each IP session?
- Fraud detection
  - List all sessions that transmitted more than 1000 bytes
  - Identify all sessions whose duration was more than twice the normal
- Security/Denial of Service
  - List all IP addresses that have witnessed a sudden spike in traffic
  - Identify IP addresses involved in more than 1000 sessions

The Streaming Model

- Underlying signal: One-dimensional array *X[1…n]* with values *X[i]* all initially zero
  - Multi-dimensional arrays as well (e.g., row-major)
- Signal is implicitly represented via a stream of updates
  - *j-th* update is <*i, c[j]*> means:
    - The count of the *i-th* item in *X[ ]* changed by a value of *c[j]* during the *j-th* update, i.e.
    - $X[i] := X[i] + c[j]$ ($c[j]$ can be >0, <0)
- Goal: Compute functions on *X[ ]* subject to
  - Small space
  - Fast processing of updates
  - Fast function computation
  - …

Example IP Network Signals

- Number of bytes (packets) sent by a source IP address during the day
  - 2^{32}-sized 1-D array; increment only
- Number of flows between a source-IP, destination-IP address pair during the day
  - 2^{32}x2^{32} 2-D array; increment only, aggregate packets into flows
- Number of active flows per source-IP address
  - 2^{32}-sized 1-D array; increment and decrement

Streaming Models: Common Special Cases

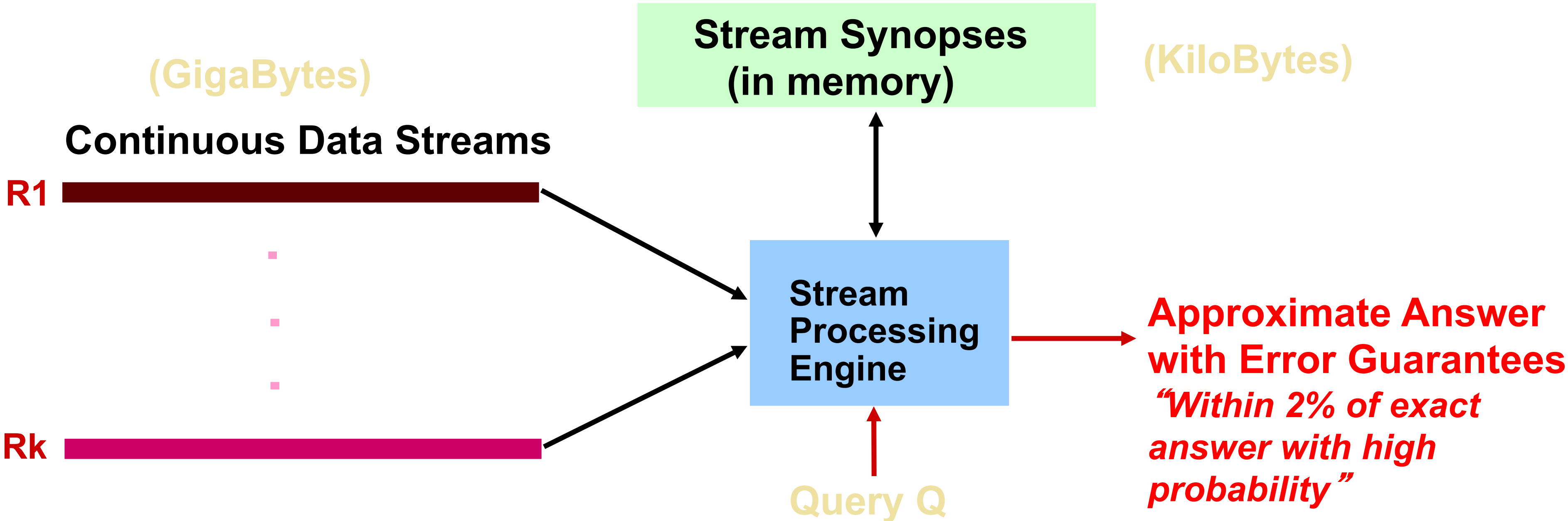
- Time-Series Model
  - Only *j-th* update updates $X[j]$ (i.e., $X[j] := c[j]$)

  e.g. stock ticker/index, velocity data, temperature, etc as functions of time
- Cash-Register Model
  - observed an item of type *i* with count $c[j]$ at the *j-th* update (or at time *j*): $<i, c[j]>$
  - $c[j]$ is always $>= 0$ (i.e., increment-only),

  e.g. a TCP packet (instead of UDP one) of 300 bytes arrives at time j

  Applicable for query stream, user activity, network traffic, revenue, clicks etc.
  - Often with $c[j]=1$, so we see a multi-set of items in one pass
- Turnstile Model
  - Most general streaming model
  - $c[j]$ can be >0 or <0 (i.e., increment or decrement, possibly require non-negativity, can consider even moving windowed statistics)
- *Problem difficulty varies depending on the model*
  - E.g., *MIN/MAX* in Time-Series vs. Turnstile!

Data-Stream Processing Model



- Approximate answers often suffice, e.g., trend analysis, anomaly detection
- Requirements for stream synopses
  - *Single Pass:* Each record is examined at most once, in (fixed) arrival order
  - *Small Space:* Log or polylog in data stream size
  - *Real-time:* Per-record processing time (to maintain synopses) must be low
  - *Delete-Proof:* Can handle record deletions as well as insertions
  - *Composable:* Built in a *distributed fashion* and combined later

Data Stream Processing Algorithms

- Generally, algorithms compute approximate answers
  - Provably difficult to compute answers accurately with limited memory
- Approximate answers - Deterministic bounds
  - Algorithms only compute an approximate answer, but bounds on error
- Approximate answers - Probabilistic bounds
  - Algorithms compute an approximate answer with high probability
    - With probability at least $1-\delta$, the computed answer is within a factor ε of the actual answer
- Single-pass algorithms for processing streams also applicable to (massive) terabyte databases!

Estimating Moments (Frequency Moments) of Data Streams

Frequency Moments

- Characterize the skewness of distribution
  - Sequence of instances
  - Instantaneous estimates

$$F_p = \sum_{i=1}^{|X|} X[i]^p$$

$X[i] =$ No. of times (i.e. repetitions) that the i-th type of items has been observed and $|X|$ is the no. of different types of items = dimension of array(vector) $X[\ ]$.

- Special cases
  - F_0 is number of distinct items
  - F_1 is number of items (trivial to estimate)
  - F_2 describes 'variance', the so-called *Gini's index of Homogeneity* (used e.g. for database query plans)
  - $F^*_\infty = \max_i \{ X[i] \}$

The Heavy Hitters Problem
(aka Frequent Items/ Hot Items/ Elephants Problem)

Frequent Elements

32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

- Elements occur multiple times, we want to find the type of elements that occur very often.
- Number of distinct elements is m
- Stream size = Total number of items in the stream = n

Note: We will use the term "element" and "item" interchangbly.

Frequent Elements

32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

Applications:

- Networking: Find "elephant" flows
- Search: Find the most frequent queries

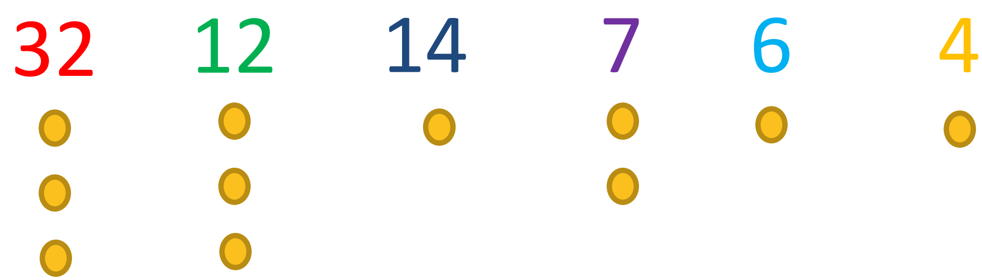
Zipf law: Typical frequency distributions are highly skewed: with few very frequent elements.
Say top 10% of elements have 90% of total occurrences.
We are interested in finding the heaviest elements

Find the Most Frequent Element
(The F_{∞}^{*} problem in [AMS 96])

32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

Exact solution:

- Create a counter for each distinct type of item on its first occurrence
- When processing an item, increment the counter for its type



*Consider the case where only some subset, say S, of element types occur exactly twice in the stream (and nothing else) before the last element, say x, arrives.

=> To determine the most frequent type of element, we must first

(i) maintain the exact membership of the subset S and then

(ii) compare with the type of x when it arrives.

In part (i), if you fail to maintain the membership of a single type, say y, when x turns out to be of type y, you won't be able to determine the most frequent one. Since you do not know the type of x in advance, you need to track the membership status of all possible element types in S

=> Need at least m bits of memory for part (i).

Top-k and Exact Heavy Hitters Problems

- Given a stream of *n* (possibly duplicate) numbers with values

in $[1, m]$, find the most frequent number

  - $\Omega(m)$ space is needed (This is the F_∞^* problem defined in [AMS96])
  - Using $\Omega(m)$ space is simple

Variations

- Top-k problem: finding the k most frequent numbers
  - At least as hard as above; still hard even if small freq. error is allowed.
  - Using $\Omega(m)$ space will be easy (with an additional heap)
- Exact Heavy-Hitters(HH) (aka Frequent or Hot Items) Problem
  - Finding the *exact list of numbers* which show up more than *n / k* times in the stream
  - $\Omega(m)$ space is also needed **[CH08]**

  e.g. For *k* = 2, i.e. HH w/ freq >50% of *n*, consider the following stream:

  1,2,3,……M, i, i, i,…. (i.e. *1,2,…,M* followed by *M* copies of *i* 's,)

  i is a HH if *i* in {1..*M*}, o.w. *i* is not a HH but Set Membership test is $\Omega(m)$

[CH08] G.Cormode, M.Hadjieleftheriou, "Finding Frequent Items in Data Streams," VLDB 2008.

Warm-up: The Majority Problem

The Problem:

Suppose we have a list of n numbers, representing the “votes” of n processors on the result of some computation.

We wish to decide if there is a majority vote (i.e. > 50%) and what the majority vote is.

- Assume the votes come in as a stream.
- Should use as little memory/storage as possible.
- Should allow the voting to be terminated at any time and the Algorithm should identify (if any) the majority vote up to the point of closing.

The Majority Algorithm

[BoyerMoore81], [FischerSalzburg82]

For each incoming item do:

```
if (currently there is no stored item)
    Store the incoming item and give it  a counter initialized to 1 ;
else if (incoming item == currently stored item)
    counter++ ;
else if (incoming item != currently stored item && counter > 1)
    counter - - ;
else /* incoming != currently stored item && counter == 1*/
    Delete the currently stored item and its counter ;
```

Outcome:

IF there is a majority vote up to this point, it will be the currently stored item. /* Note: If there is no majority vote, all bets are off. */

[BoyerMoore81] B.Boyer, J.Moore, "A fast majority vote algorithm," Tech Report ICSCA-CMP-32, ICS, U.of Texas, Feb. 1981.

[FischerSalzburg82] M.Fischer, S.Salzburg, "Finding a majority among n votes:Sol. to Prob. 81-5," Journal of Algorithms, 1982

Frequent Elements: Misra & Gries 1982

(Generalization of the Majority Algorithm where *k=2*)

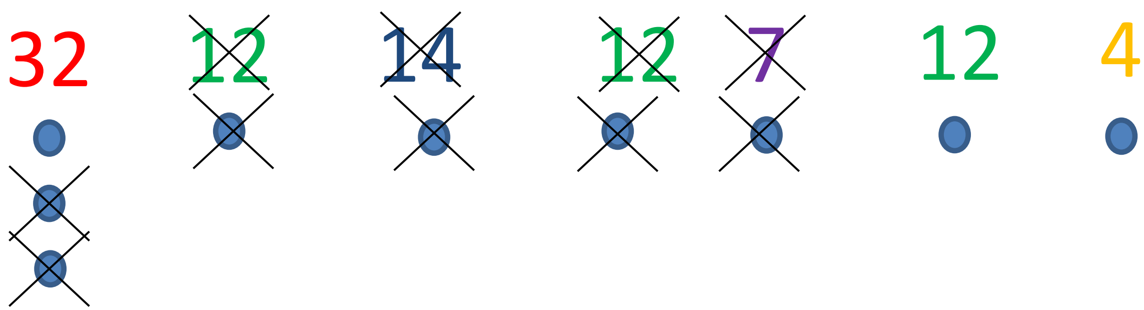
32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

Problem: For a stream with *n* elements, find and count elements which shows up more than *n/k* times

Solution:

<u>For each incoming element $\boldsymbol{i}$</u>

- If we already have a counter for $\boldsymbol{i}$, increment it
- Else, If there is no counter, but there are fewer than $k-1$ counters, create a counter for $\boldsymbol{i}$ initialized to $\mathbf{1}$.
- Else, decrease all counters by $\mathbf{1}$. Remove $\mathbf{0}$-value counters.



$$m = 6$$
$$k = 4$$
$$n = 11$$

Frequent Elements: Misra & Gries 1982

Generalization of the Majority Algorithm where *k=2*

32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

Processing an incoming element $\boldsymbol{i}$

- If we already have a counter for $\boldsymbol{i}$, increment it
- Else, If there is no counter for it, but there are fewer than $k - 1$ counters, create a counter for type $\boldsymbol{i}$ initialized to $\mathbf{1}$.
- Else, decrease all counters by $\mathbf{1}$. Remove $\mathbf{0}$ $-$value counters.

Query: How many times $\boldsymbol{i}$ occurred ?

- If we have a counter for $\boldsymbol{i}$, return its value
- Else, return $\mathbf{0}$.

The counter value for each element is clearly an under-estimate. What can we say precisely?

Misra & Gries 1982 : Analysis

How many decrements to element i's counter can we have ?

$\Leftrightarrow$ How many decrement rounds can we have ?

- Total number of items in the stream = n
- Let n' be the sum of final values of all counters.
- Each round of decrement results in removing $(k - 1 + 1)$ counts from the system (including the current occurrence of the input element.), i.e. k "uncounted" occurrences.

$\Rightarrow$ There can be at most $\frac{n-n'}{k}$ rounds of decrement

$\Rightarrow$ **The counter value of an element is smaller than its true count by at most $\frac{n-n'}{k}$!**

Misra & Gries 1982 : Analysis

Let $\widehat{f_i}$ be the final counter value for i. Take $\widehat{f_i}$ as an estimate of f_i, namely, the true number of occurrences of i in the stream, we have:

$$\left(f_i - \frac{n}{k}\right) \le \left(f_i - \frac{n-n'}{k}\right) \le \widehat{f_i} \le f_i$$

$$\Rightarrow \qquad \text{If } \left(f_i - \frac{n}{k}\right) > 0, \text{ then } \widehat{f_i} > 0$$

In other words, if an element actually occurs more frequent than $\frac{1}{k}$ of the stream size, it WILL have a counter bearing a +ve value at the end of the above process.

$\Rightarrow$ The list of elements having a counter value (> 0) will contain ALL "frequent" types of items. Here a "frequent" is defined as one which has a frequency greater than $\frac{1}{k}$ of the total number of items in the stream.

Note: there can be "false-positive" in this candidate list!

$\Rightarrow$ Can use a 2nd pass to verify (count exactly) if each candidate on the list is truly frequent.

Since there is at most $(k\text{-}1)$ non-zero counters, $O(k)$ memory suffices for this 2-pass alg.

Summary of Misra & Gries 1982

Estimate is smaller than true count by at most $\frac{n-n'}{k}$

$\Rightarrow$ Can get good estimates for f_i even with one single pass when the number of occurrences $\gg \frac{n-n'}{k}$

- By setting $k = 1/\mathcal{E}$, the estimation error will be bound (< $n\mathcal{E}$).
- The error bound can be readily computed: Can track n using simple count ; know n' (from list of final counter values) and k is a given parameter.
- Even using the 1st pass algorithm alone would work well in practice because typical frequency distributions have few very popular elements due to "Zipf law" ;
- Cannot handle "-ve" arrival though !

[MG82] J. Misra, D. Gries, "Finding repeated elements," Science of Computer Programming, No. 2, 1982, http://www.cs.utexas.edu/users/misra/scannedPdf.dir/FindRepeatedElements.pdf

Merging two Misra Gries Summaries [ACHPWY12]

Basic merge:

- If an element i is in both structures, keep one counter with sum of the two counts
- If an element i is in one structure only, keep the counter

Reduce: If there are more than $k-1$ counters

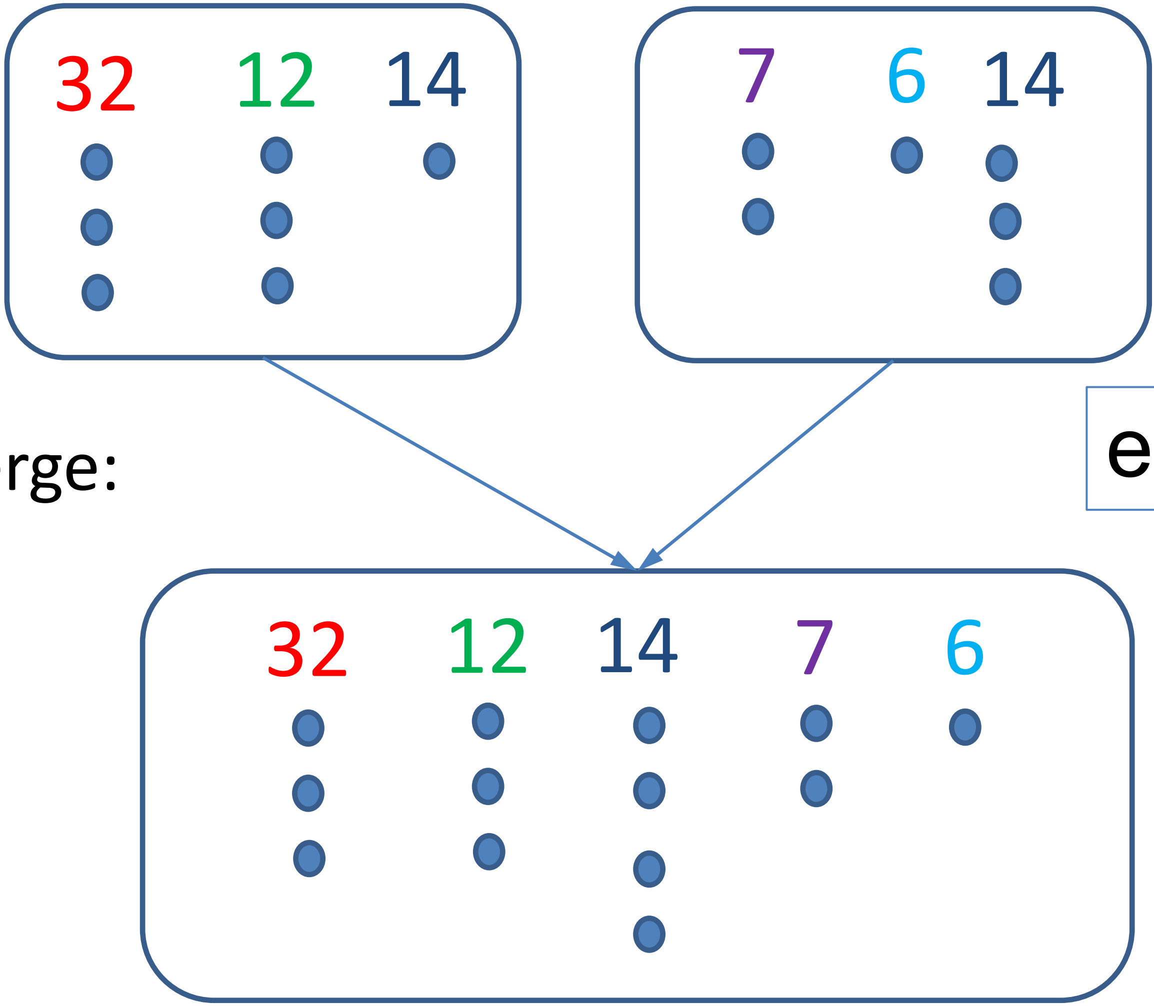
- Take the k^{th} largest counter
- Subtract its value from all other counters
- Delete non-positive counters

[ACHPWY12] Agarwal, Cormode, Huang, Phillips, Wei, and Yi, Mergeable Summaries, PODS 2012.

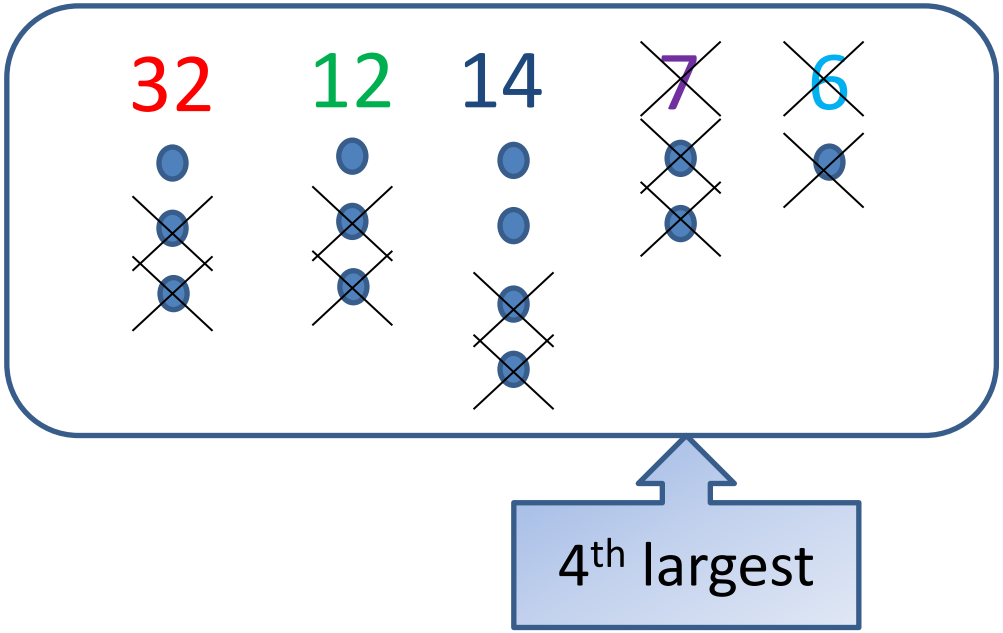
Merging two Misra Gries Summaries

Basic Merge:

e.g. $k = 4$



Merging two Misra Gries Summaries



e.g. $k = 4$

Reduce since there are more than $(k-1) = 3$ counters :

- Take the k^{th} = 4^{th} largest counter
- Subtract its value (2) from all other counters
- Delete non-positive counters

Merging MG Summaries: Correctness

Claim: Final summary has at most $(k-1)$ counters

Proof: We subtract the k^{th} largest from everything, so at most the $(k-1)$ largest counters can remain positive.

Claim: For each type of element, final summary count is smaller than true count by at most $\frac{n-n'}{k}$

Merging MG Summaries: Correctness

Claim: For each element, final summary count is smaller than true count by at most $\frac{n-n'}{k}$

Proof: "Counts" for element type i can be lost in part1, part2, or in the reduce component of the merge

We add up the bounds on the losses

Part 1:

Total occurrences: n_1

In structure: $n_1{}'$

Count loss: $\leq \frac{\boldsymbol{n_1 - n_1{}'}}{\boldsymbol{k}}$

Part 2:

Total occurrences: n_2

In structure: $n_2{}'$

Count loss: $\leq \frac{\boldsymbol{n_2 - n_2{}'}}{\boldsymbol{k}}$

Reduce loss is at most $\boldsymbol{X = k}^{\text{th}}$ largest counter

Merging MG Summaries: Correctness

$\Rightarrow$ "Count loss" of one element type is at most

$$\frac{n_1 - n_1'}{k} + \frac{n_2 - n_2'}{k} + X$$

Part 1:

Total occurrences: n_1

In structure: $n_1{}'$

Count loss: $\leq \frac{n_1 - n_1'}{k}$

Part 2:

Total occurrences: n_2

In structure: $n_2{}'$

Count loss: $\leq \frac{n_2 - n_2'}{k}$

Reduce loss is at most $X = k^{\text{th}}$ largest counter

Merging MG Summaries: Correctness

Counted occurrences in structure:

- After basic merge and before reduce: $n_1' + n_2'$
- After reduce: n'

Claim: $n_1' + n_2' - n' \geq k\, X$

Proof: X are erased in the reduce step in each of the k largest counters. Maybe more in smaller counters.

"Count loss" of each element type is at most:

$$\frac{n_1 - n_1'}{k} + \frac{n_2 - n_2'}{k} + X \leq \frac{1}{k}(n_1 + n_2 - n')$$

$$\Rightarrow \text{at most } \frac{n - n'}{k} \text{ uncounted occurrences}$$

Networking Applications of Heavy-Hitter Algorithms

Detection and Counting of:

- Super/Top Spreaders
  - Find hosts who are spreading a large number of flows
  - Scanning, worm spreading, under attack, P2P nodes, web server, proxy, …
- Super/Top Scanners
  - Find hosts who are spreading a large number of small flows
  - More suspicious than top spreaders
- Popular types of packets/flows/queries/search-words
- Flow Size Distribution/ Iceberg Histogram
  - How many flows are there having N >> 1 packets?
  - Traffic engineering, anomaly detection
  - Algorithm [Kumar04] extending the linear probabilistic counting algorithm [Whang90]

More Networking Applications of Data Stream Algorithms

- Elephant flow detection and counting
  - Find flow with large size
  - Billing and accounting
  - Sample and hold algorithm [Estan02] and
  - The Run-based schemes, e.g. co-incidence or 2-in-a-row, [Kodialam04], [Hao04], [Hao05].
- Flow entropy
  - Calculate the entropy of flows
  - Measures information randomness
  - Traffic engineering, anomaly detection, clustering
  - Algorithm [Lall06] based on estimating frequency moments algorithm [AMS96]
- OD flow entropy
  - Calculate the entropy of OD flows
  - Traffic engineering, network wide anomaly detection
  - Algorithm [Zhao07] based on estimating frequency moments algorithm [Indyk00]

Additional References

[FM85] Probabilistic Counting Algorithms for Data Base Applications, Phillippe Flajolet and G.Nigel Martin, Journal of Computer and System Sciences (JCSS), 1985.

[DF03] Loglog counting of large cardinalities, M. Durand and P. Flajolet, European Symposium on Algorithms 2003

[FFGM07] Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm, P. Flajolet, Eric Fusy, O. Gandouet, and F. Meunier, Conference on Analysis of Algorithms, 2007

[GKMS01] Surfing wavelets on streams: One pass summaries for approximate aggregate queries, A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss., VLDB Journal, 2001.

[Whang 90] A linear-time probabilistic counting algorithm for database applications, K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, ACM Transaction on Database Systems (TODS), 1990

[AMS96] The space complexity of approximating the frequency moments, Noga Alon, Yossi Matias and Mario Szegedy, ACM STOC 1996, JCSS 1999

[Indyk00] Stable distributions, pseudorandom generators, embeddings and data stream computation, P. Indyk, ACM FOCS 2000, JACM 2006

[CM05] What's hot and what's not: tracking most frequent items dynamically, Graham Cormode and S. Muthukrishnan, ACM TODS'05

[CCFC02] Finding frequent items in data streams, Moses Charikar, Kevin Chen, and Martin Farach-Colton3, ICAPL'02

[DLM02] Frequency Estimation of Internet Packet Streams with Limited Space, Erik D. Demaine, et al., ESA'02

[KSP03] A simple algorithm for finding frequent elements in streams and bags, Richard M. Karp, et al., ACM TODS'03

[MAA06] An integrated efficient solution for computing frequent and top-k elements in data streams, Ahmed Metwally, et al., ACM TODS'06

[Estan03] Bitmap algorithms for counting active flows on high speed links, Cristian Estan George Varghese Michael Fisk, ACM Internet Measurement Conference (IMC) 2003, ACM Transaction on Networking (TON) 2006

[ZKWX05] Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices, Q. G. Zhao, A. Kumar, J. Wang, and J. Xu., ACM Sigmetrics 2005

[Estan02] New directions in traffic measurement and accounting, Cristian Estan and George Varghese, ACM Sigcomm 2002

[Kodialam04] Run-bAsed Traffic Estimator (RATE): A Simple, Memory Efficient Scheme for Per-flow Rate Estimation, Murali Kodilam, T.V. Lakshman, S. Mohanty, Infocom 2004.

[Hao04] ACCEL-RATE: A faster memory efficient scheme for Per-flow Rate Estimation, Fang Hao, Murali Kodialam, T.V. Lakshman, ACM Sigmetrics 2004.

[Hao05] Fast, memory-efficient traffic estimation by co-incidence counting, Fang Hao, Murali Kodialam, T.V.Lakshman, Hui Zhang, Infocom 2005.

[Lall06] Data streaming algorithms for estimating entropy of network traffic, Ashwin Lall et al., ACM Sigmetrics 2006

[Zhao07] A Data Streaming Algorithm for Estimating Entropies of OD Flows, ACM IMC 2007

[Kumar04] Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution, Abhishek Kumar et al., ACM Sigmetrics 2007

Backup Slides

Counting Distinct Items in Data Streams (F_0)

Recap: Definition of Frequency Moments

- Characterize the skewness of distribution
  - Sequence of instances
  - Instantaneous estimates

$$F_p = \sum_{i=1}^{|X|} X[i]^p$$

$X[i] =$ No. of times (i.e. repetitions) that the i-th type of items has been observed and $|X|$ is the no. of different types of items = dimension of array(vector) $X[\ ]$.

- Special cases
  - F_0 is number of distinct items
  - F_1 is number of items (trivial to estimate)
  - F_2 describes 'variance', the so-called *Gini's index of Homogeneity* (used e.g. for database query plans)
  - $F^*_\infty = \max_i \{ X[i] \}$

Distinct Element Counting Problem (F_0)

- Given a stream of (possibly duplicated) elements, count the number of distinct elements
  - Example 1
    - a, b, c, a, d, e, f, c, g, a
    - Total number is 10, while distinct cardinality is 7
  - Example 2
    - Count the distinct records in a column of a large table
- Classical algorithms
  - Linear probabilistic counting (Hit test based) algorithm
  - Bit pattern based algorithms
  - Order statistics based algorithms

Linear probabilistic counting algorithm

[Whang[90] A linear-time probabilistic counting algorithm for database applications, Whang, et al., ACM Transaction on Database Systems, 1990

- Use a m bit bitmap $\mathbf{B}$
- Use a uniform hash function h to map each value v to $h(v) \in [0, m-1]$
- Set $\mathbf{B}[h(v)] = 1$ when *v* appears in stream
- After seeing all elements

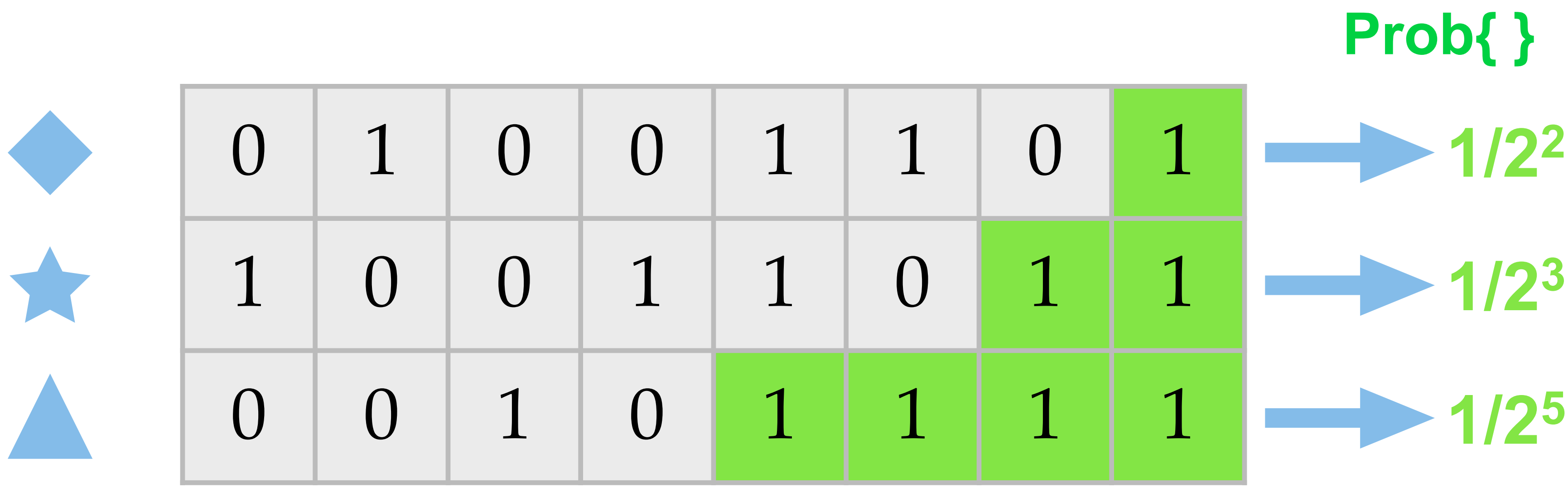
1	1	1	1	1	1	0	1	0	1	1	0	0	…	…	0	0	0	1	0	0

$$Pr(\text{bit } i \text{ is } 0) = (1 - \frac{1}{m})^n \approx e^{-\frac{n}{m}}$$

- The expected number of 0's is: $m \times e^{-\frac{n}{m}}$
- The distinct number can be estimated by $\hat{n} = m \times \ln \frac{m}{E}$ where E is the actual number of empty bits in $\mathbf{B}$
- Why it is called linear? $m \sim \Omega(n)$

Counting Number of Distinct Items (F_0) via Loglog-Counting by Flajolet et al

- Assume the no. of distinct items is at most N
- Use a uniform hash func. $h(v)$ to map each item v to a binary number in the range of $[0, 2^{\text{Log}_2(N)} - 1]$



- Consider the no. of consecutive "1"' s starting from the LSB before a "0" appears in the binary value output by the hash
- Equivalently, we can track the bit-position, say r, of the rightmost "0" in the hash output ; corresponding Prob. = $1/2^r$

LogLog Counting for Distinct Items

- Intuitively, the maximum number of consecutive "1"'s (starting from the LSB) among the hash output values of ALL of observed items indicates the magnitude of n, *i.e. # of distinct items observed* !
- More importantly, repetitions of same item do not matter !
- Let R be the maximum of the bit-position of the rightmost "0" among the hash outputs of n distinct items (where LSB = bit-position 1).
  - In other words, R = 1 + maximum # of consecutive "1"'s (starting from the LSB) observed over the hash output of n distinct items
- It can be shown that [DF03]: $E\left[R\right] \approx 1.3 + \log_2 n$
- As we only need to track the *maximum value of R* observed so far, the space complexity of the algorithm is merely LogLog *N* *!*
- Problem: Deviation of R from its expectation $E\left[R\right]$ may be very large
- In fact, $E\left[2^R\right] \rightarrow \infty$! How to reduce the deviation of R?

[DF03] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," European Symposium on Algorithms (ESA), 2003.

Loglog Counting (cont'd)

Stochastic Averaging

- Split the incoming streams into *m* sub-streams to obtain *m* different (random) values of R_j, one for each sub-stream.
- The split can be done by using another uniform hash function
  - What is the impact of potential temporal correlation in the original streams though ? e.g. items of the same type always arrive in a batch ?
- In [DF03], an estimate of *n* is obtained based on taking arithmetic average of the R_j's as follows:

$$\hat{n} := \alpha_m m \cdot 2^{\frac{1}{m}\sum_{j=1}^{m} R_j} \quad \text{with standard est. error} \approx 1.30/\sqrt{m}$$

$$\text{where } \alpha_m := \left(\Gamma\left(\frac{-1}{m}\right) \cdot \frac{1-2^{1/m}}{\log 2} \right)^{-m} \quad \text{and} \quad \Gamma(s) := \frac{1}{s}\int_0^{\infty} e^{-t} t^s \, dt$$

[DF03] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," European Symposium on Algorithms (ESA), 2003.
[FM85] P. Flajolet and G.N. Martin, "Probabilistic Counting for Database Applications," Journal of Computer and System Sciences Vol. 31, No. 2, 1985.

HyperLogLog and more…

- HyperLogLog
  - In [FFGM07], Harmonic Mean (H.M.) instead of Geometric Mean is used in the estimator to yield the so-called HyperLogLog estimator:

$$\hat{n} := \frac{\beta_m m^2}{\sum_{j=1}^{m} 2^{-R_j}} \text{ with } \beta_m := \left(m \int_0^{\infty} \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1} \text{ and std. est. error} \approx 1.03/\sqrt{m}$$

[FFGM07] P. Flajolet et al., “HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm,” 2007 Conference on Analysis of Algorithms, AofA 07.

  - Accuracy Comparison – for upto $N=10^9$ distinct input items:

Algorithm	*Cost*	*(units)*	*Accuracy*
Hit Counting [24]	$\frac{1}{10}N$	bits	$\approx 2\%$
Adaptive Sampling [12]	m	words (≈ 32 bits)	$1.20/\sqrt{m}$
Probabilistic Counting [15]	m	words (≤ 32 bits)	$0.78/\sqrt{m}$
MinCount [2, 6, 16, 18]	m	words (≤ 32 bits)	$1.00/\sqrt{m}$
LogLog [10]	m	bytes (≤ 5 bits)	$1.30/\sqrt{m}$
HyperLogLog	m	bytes (≤ 5 bits)	$1.04/\sqrt{m}$

  - Further Refinements and Practical Implementation in [HNH13].

[HNH13] S. Heule, M. Nunkesser, A.Hall, “HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm,” EDBT/ICDT 2013.

Other Order statistics based Algorithms for Distinct Item Counting

- Use a uniform hash function h to map each item (value) v to a real number $h(v) \in [0,1]$
- Find the minimum hashed result $X = \min(h(e[1]), h(e[2]), ...)$
- Routine Computation* can show that $E(X) = \frac{1}{n+1}$
- n can then be estimated from X by method of moments
  - Inverse [2002], square root [2005], logarithm [2006], …
- Stochastic averaging is needed
- Recently, Cohen et al [CKY] have proposed a Unified scheme to generalize "Extreme Order based" Distinct Item Counting schemes, including Hyperloglog, Loglog, MinCount, etc, to support Weighted Distinct Item Counting !

* https://research.neustar.biz/2012/07/09/sketch-of-the-day-k-minimum-values/

[CKY14] R.Cohen, L.Katzir, A.Yehezkel, "A unified scheme for generalizing cardinality estimators to sum aggregation," Information Processing Letters, 2014.

Problems and Algorithms in Networking

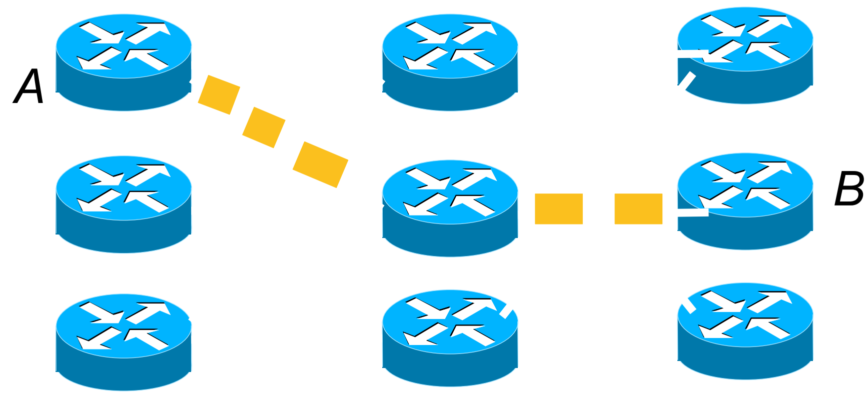
- Problems that use "Distinct element counting":
  - Flow Counting
  - Traffic (OD Flow) Matrices
  - Various Production systems in Google including Sawzall, Dremel and PowerDrill all require the estimation of the cardinality of some LARGE data sets [HNH13].
    - e.g., PowerDrill needs to estimate the number of distinct search queries sent to Google.com over a time period

Flow Counting

- A flow is defined as a combination:
  - f=<src address, dst address, src port, dst port, protocol>
- Total flow number can indicate
  - Link utilization
  - DDoS
  - Flash crowds
  - Port scan
  - Worm spreading
- Well captured by "distinct element counting”
- [Estan03] in SIGCOMM (networking) reconsiders this problem and extends Linear probabilistic Counting [Whang90]
  - Combine Linear probabilistic Counting with sampling to reduce memory consumption
  - Better than Probabilistic Counting, almost the same as LogLog/HyperLogLog ; N.B.: Loglog/ HyperLoglog estimates poorly for small *N*

Traffic (OD Flow) Matrices

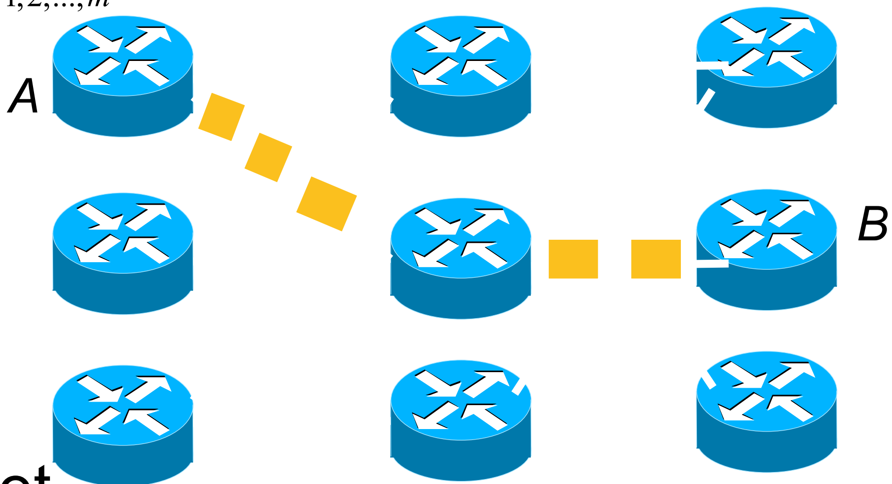
- Consider an ISP network with many POP's
- How much traffic is there from POP A to POP B?
- This information is useful
  - Traffic engineering
  - Network plan and provision
  - Network wide anomaly detection



- Traditional way
  - Collect link volume statistics at all (or at least a large portion of) POP's, combine with routing information, and use various traffic model assumptions (what are the disadvantages?)
  - Error around 20%
- Streaming way [ZKWX05]
  - Error around 3% (counting packets or flows)

Traffic Matrices

$B_A =$ the buckets (array) holding the m values of R_j's at node A, each denoted by R_j^A for $j = 1,2,\ldots,m$



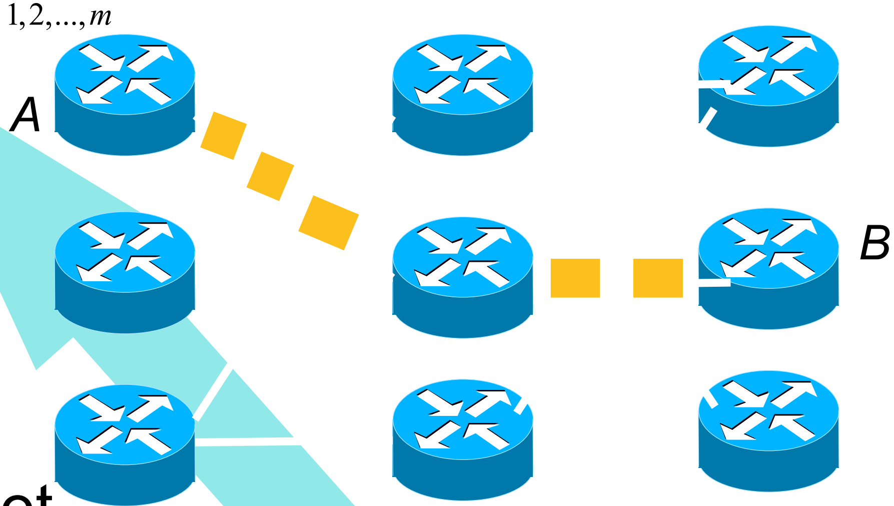
Let B_B be the buckets (array) holding the m values of R_j's at node B, each denoted by R_j^B for $j = 1,2,\ldots,m$

- We can get
  - P_A: the set of packets originating from A
  - P_B: the set of packets destined to B
- By tracking m R_j's at each node, we can estimate:
  - $N_A = ||P_A||$: number of pkts/flows originating from A, by $\mathbf{B}_A$
  - $N_B = ||P_B||$: number of pkts/flows destined to B, by $\mathbf{B}_B$
- How can we get $N_{A \cap B} = ||P_A \cap P_B||$?

$$||P_A \cap P_B|| = ||P_A|| + ||P_B|| - ||P_A \cup P_B||$$

Traffic Matrices

$B_A =$ the buckets (array) holding the m values of R_j's at node A, each denoted by R_j^A for $j = 1,2,\ldots,m$



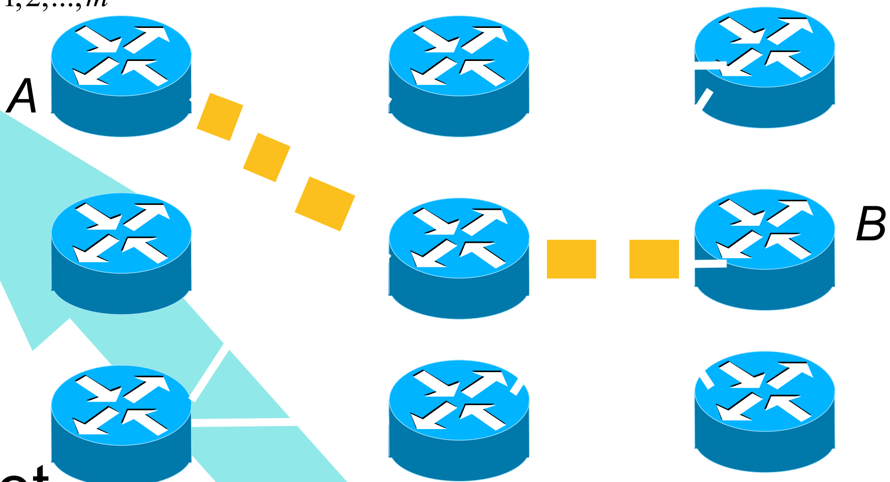
Let B_B be the buckets (array) holding the m values of R_j's at node B, each denoted by R_j^B for $j = 1,2,\ldots,m$

- We can get
  - P_A: the set of packets ...inating from A
  - P_B: the set of packets d... ...d to B
- By tracking m R_j's at eac... $\mathbf{B}_A$ ...de, we can est... $\mathbf{B}_B$ ...te:
  - $N_A = ||P_A||$: number of pkts/flo... ...iginating fro... by $\mathbf{B}_A$
  - $N_B = ||P_B||$: number of pkts/flows ... ...ned to B, ... $\mathbf{B}_B$
- How can we get $N_{A \cap B} = ||P_A \cap P_B||$?

$$||P_A \cap P_B|| = ||P_A|| + ||P_B|| - \boxed{||P_A \cup P_B||}$$

Traffic Matrices

$B_A =$ the buckets (array) holding the m values of R_j's at node A, each denoted by R_j^A for $j = 1, 2, \ldots, m$



Let B_B be the buckets (array) holding the m values of R_j's at node B, each denoted by R_j^B for $j = 1, 2, \ldots, m$

- We can get
  - P_A: the set of packet
  - P_B: the set of packets d

$\|P_A \cup P_B\|$ can be estimated via Loglog counting with $R_j^{A \cup B} = \max(R_j^A, R_j^B)$ for $j = 1, 2, \ldots m$

- By tracking m R_j's at eac $\mathbf{B}_A$ de, we can est $\mathbf{B}_B$ te:
  - $N_A = ||P_A||$: number of pkts/flo iginating fro by $\mathbf{B}_A$
  - $N_B = ||P_B||$: number of pkts/flows ned to B, $\mathbf{B}_B$
- How can we get $N_{A \cap B} = ||P_A \cap P_B||$?

$$||P_A \cap P_B|| = ||P_A|| + ||P_B|| - \boxed{||P_A \cup P_B||}$$

Bloom Filter

- http://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf

BLOOM.COM
SOCIAL BEAUTY STORE

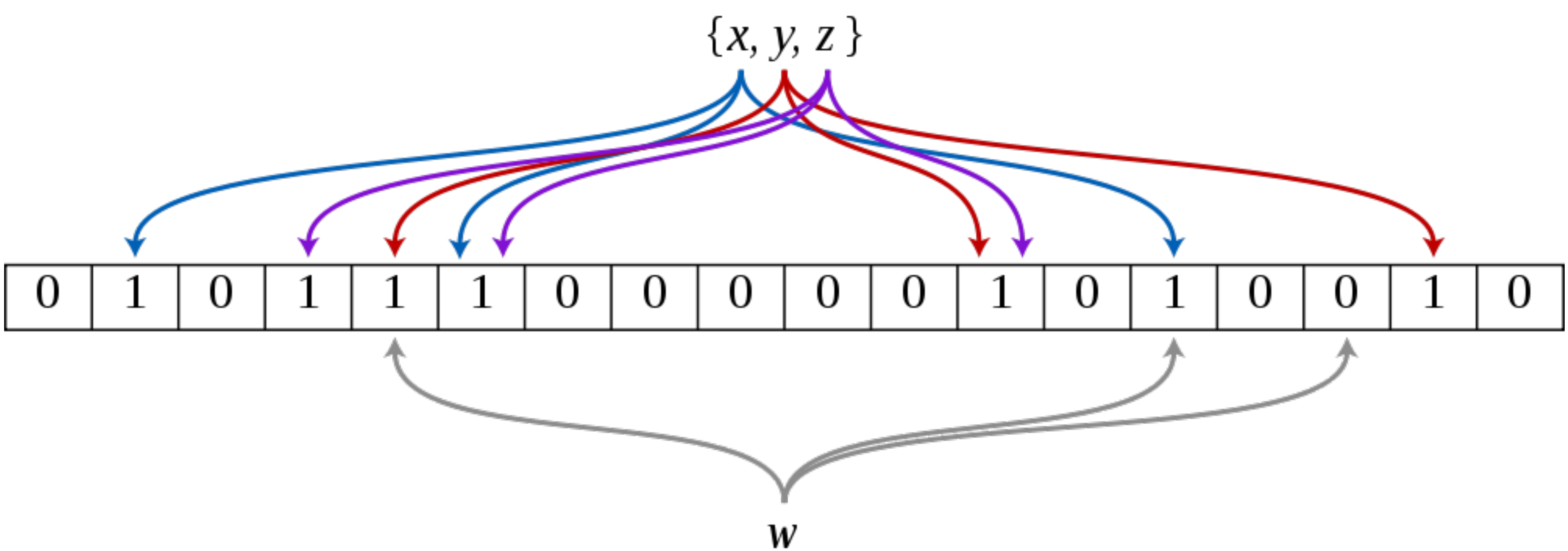
Beyond Heavy Hitters

- Check for previously seen items
  - but don’t need to have counts, just existence
- Check for frequency estimate
  - but don’t want to store labels
  - but want estimate for all items (not just HH)
  - but want to be able to aggregate
  - but want turnstile computation

Bloom filter, Count-Min sketch

Bloom Filter

- Bit array b of length n
  - insert(x): for all i set bit b[h(x,i)] = 1
  - query(x): return TRUE if for all i b[h(x,i)] = 1



Bloom Filter

- Bit array b of length n
  - insert(x): for all i set bit b[h(x,i)] = 1
  - query(x): return TRUE if for all i b[h(x,i)] = 1
- Only returns TRUE if all k bits are set
- No false negatives but false positives possible
  - Probability that an arbitrary bit is set

$$\Pr\{b[i]=1\} = 1-\left(1-\frac{1}{n}\right)^{mk} \approx 1-e^{-\frac{mk}{n}}$$

  - Probability of false positive (approx. indep.)

$$\Pr\{b[h(x,1)] = \ldots = b[h(x,k)] = 1\} \approx \left(1-e^{-\frac{mk}{n}}\right)^{k}$$

Bloom Filter

- Minimizing k to minimize false positive rate

$$\partial_k \left[k \log\left(1 - e^{-mk/n}\right)\right] = \log\left(1 - e^{-mk/n}\right) + \frac{mk}{n} \frac{e^{-mk/n}}{1 - e^{-mk/n}}$$

  This vanishes for $\frac{mk}{n} = \log 2 \text{ and hence } k = \frac{n}{m} \log 2$ with a false positive rate of 2^{-k}

- More refined analysis & details, e.g. in the Mitzenmacher & Broder 2004 tutorial.

- Use $1.44\log_2(1/\varepsilon)$ bits of space per inserted key where ε is the false positive rate of the BF.

Cool things to do with a Bloom Filter

- Bloom filter of union of two sets by OR

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1

1	0	1	0	1	1	1	0	1	0	1	1	0	1	0	1	1	1

- Parallel construction of Bloom filters
- Time-dependent aggregation
- Fast approximate set union
  (bitmap operation rather than set manipulation)
- Also use it to halve bit resolution of Bloom filter
  - by "OR"ing the 1st half of the BF with its 2nd half

Cool things to do with a Bloom Filter

- Set intersection via AND

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1

0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

  - No false negatives
  - More false positives than building from scratch
  - Use bits to estimate size of set union/intersection

$$\Pr\{b=1\} = \left(1-\left(1-\frac{1}{m}\right)^{k|S_1\cap S_2|}\right)$$

$$+\left(1-\frac{1}{m}\right)^{k|S_1\cap S_2|}\left(1-\left(1-\frac{1}{m}\right)^{k|S_1-(S_1\cap S_2)|}\right)\left(1-\left(1-\frac{1}{m}\right)^{k|S_2-(S_1\cap S_2)|}\right)$$

$$=\left(1-\left(1-\frac{1}{m}\right)^{k|S_1|}-\left(1-\frac{1}{m}\right)^{k|S_2|}+\left(1-\frac{1}{m}\right)^{k(|S_1|+|S_2|-|S_1\cap S_2|)}\right)$$

Cool things to do with a Bloom Filter

- Set intersection via AND

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1

0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

  - No false negatives
  - More false positives than building from scratch
  - Use bits to estimate size of set union/intersection

$$\Pr\{b=1\} = \left(1-\left(1-\frac{1}{m}\right)^{k|S_1\cap S_2|}\right) + \left(1-\frac{1}{m}\right)^{k|S_1\cap S_2|}\left(1-\left(1-\frac{1}{m}\right)^{k|S_1-(S_1\cap S_2)|}\right)\left(1-\left(1-\frac{1}{m}\right)^{k|S_2-(S_1\cap S_2)|}\right)$$

$$\approx 1-e^{-\frac{k|S_1|}{m}}-e^{-\frac{k|S_2|}{m}}+e^{-\frac{k|S_1\cup S_2|}{m}}$$

Cool things to do with a Bloom Filter

- Set intersection via AND

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1

0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

  - No false negatives
  - More false positives than building from scratch
  - Use bits to estimate size of set union/intersection

$$\Pr\{b=1\} = \Pr\{b=1|S_1\} + \Pr\{b=1|S_2\} - \Pr\{b=1|S_1 \cup S_2\}$$

$$\approx 1 - e^{-\frac{k|S_1|}{m}} - e^{-\frac{k|S_2|}{m}} + e^{-\frac{k|S_1 \cup S_2|}{m}}$$

Counting Bloom Filter

- Plain Bloom filter doesn't allow removal

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

  - insert(x): for all i set bit b[h(x,i)] = 1
    we don't know whether this was set before
  - query(x): return TRUE if for all i b[h(x,i)] = 1
- Counting Bloom filter keeps track of inserts
  - query(x): return TRUE if for all i b[h(x,i)] > 0
  - insert(x): if query(x) = FALSE (don't insert twice)
    for all i increment b[h(x,i)] = b[h(x,i)] + 1
  - remove(x): if query(x) = TRUE (don't remove absents)
    for all i decrement b[h(x,i)] = b[h(x,i)] - 1

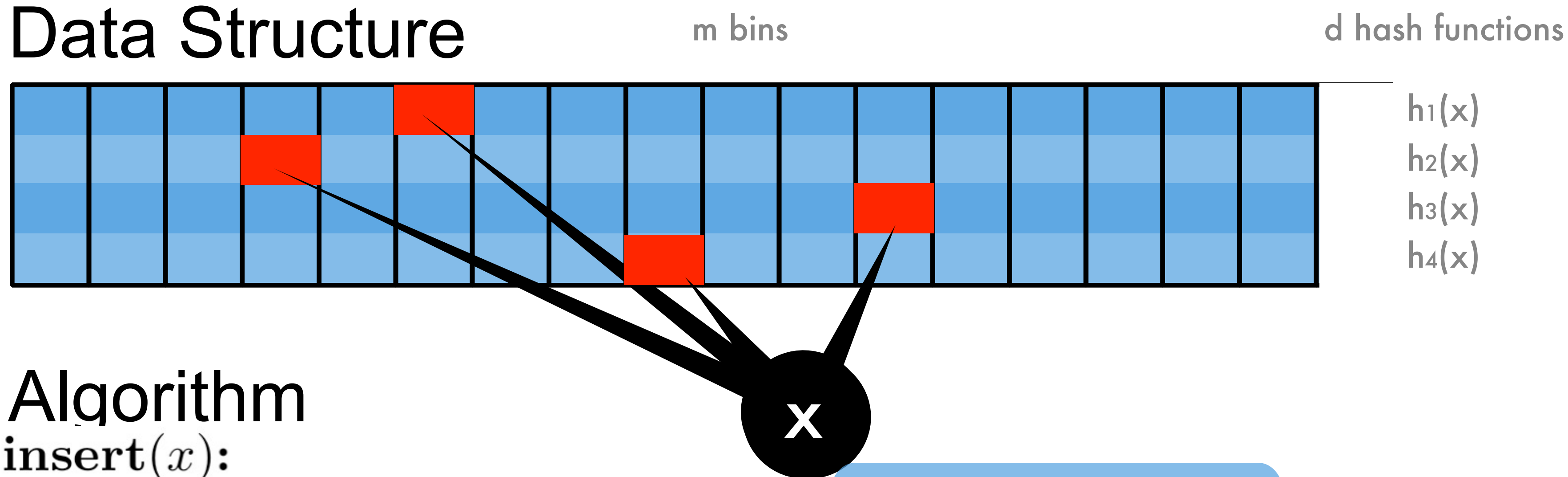
only needs
log log m bits

Count Min sketch

https://sites.google.com/site/countminsketch/

Count Min (CM) Sketch

- Data Structure



- Algorithm

insert(x)**:**
for $i = 1$ **to** d **do**
$\quad M[i, h_i(x)] \leftarrow M[i, h_i(x)] + 1$
end for

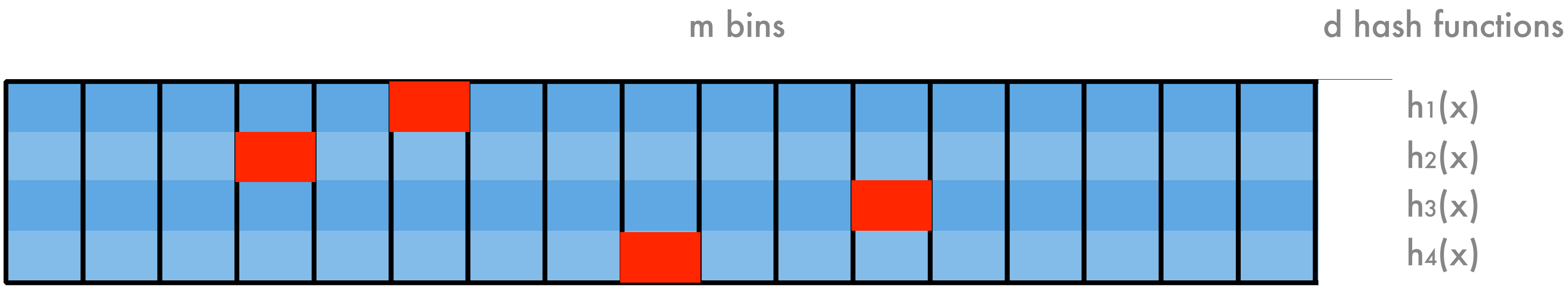
query(x)**:**
$c = \min\{h_i(x) \text{ for all } 1 \leq i \leq d\}$
return c

like Bloom filter but with counters

supports turnstile

Count Min (CM) Sketch

- Data Structure

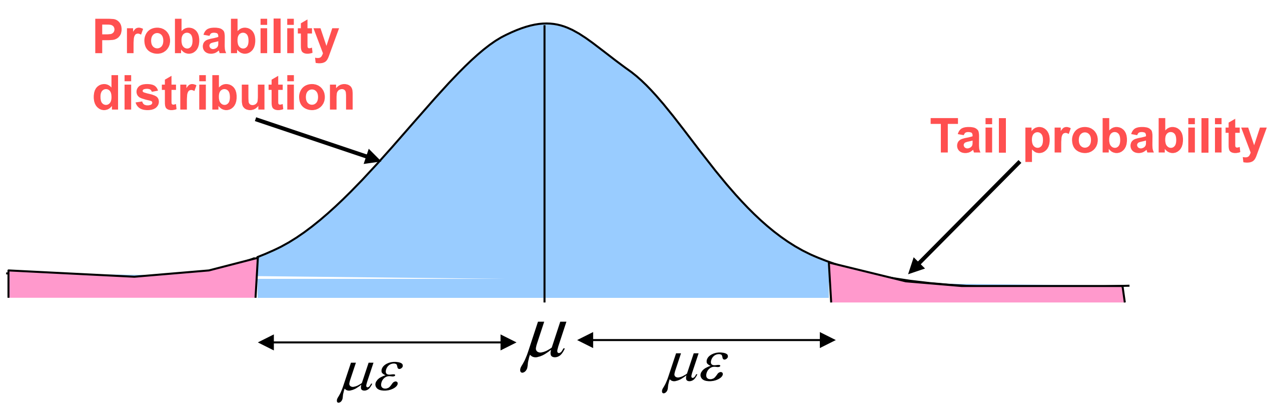


- Guarantees
  - Approximation quality is

$$n_x \leq c_x \leq n_x + \epsilon \sum_{x'} n_{x'} \text{ for } m = \left\lceil \frac{e}{\epsilon} \right\rceil \text{ with probability } 1 - e^{-d}$$

Basic Tools: Tail Inequalities

- General bounds on *tail probability* of a random variable (that is, probability that a random variable deviates far from its expectation)



- <u>Basic Inequalities:</u> Let X be a non-negative random variable with expectation μ and variance Var[X]. Then for any $\varepsilon > 0$

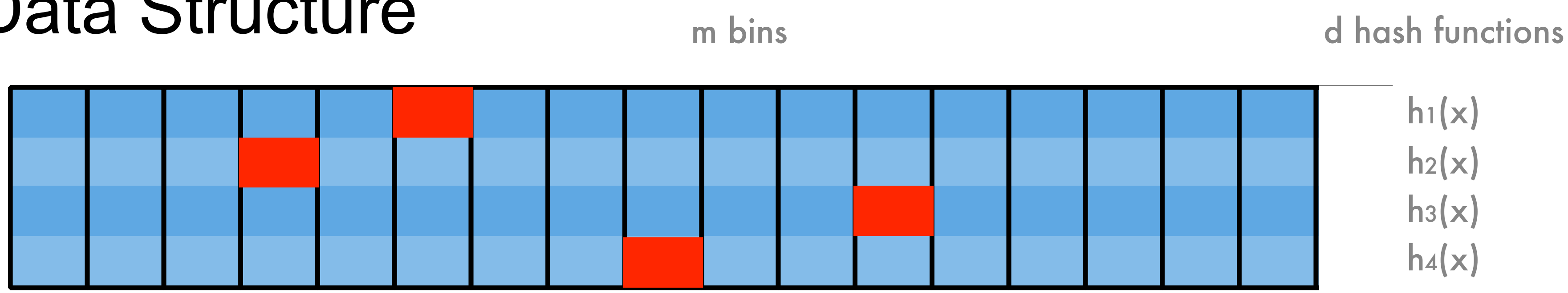
Markov: $\Pr(X \geq \varepsilon) \leq \frac{\mu}{\varepsilon}$

- Sub. $X = \left(Y - E(Y)\right)^2$ into the Markov Inequality, we have:

Chebyshev: $\Pr(|Y - \mu_Y| \geq \mu_Y \varepsilon) \leq \frac{Var[Y]}{\mu_Y^2 \varepsilon^2}$

Proof

- Data Structure



- Bin value lower bound by actual target item count
  - Each bin is updated whenever we see the target item
  - bin value >= no. of times the target item occurs
  - It is OK to take minimum among the d bins which the target item is mapped to
- Expectation of over-count
- Prob. of incrementing a bin at random (by other items) is 1/m
- => Expected overestimate is n/m.

Proof

- Markov inequality on random variable:

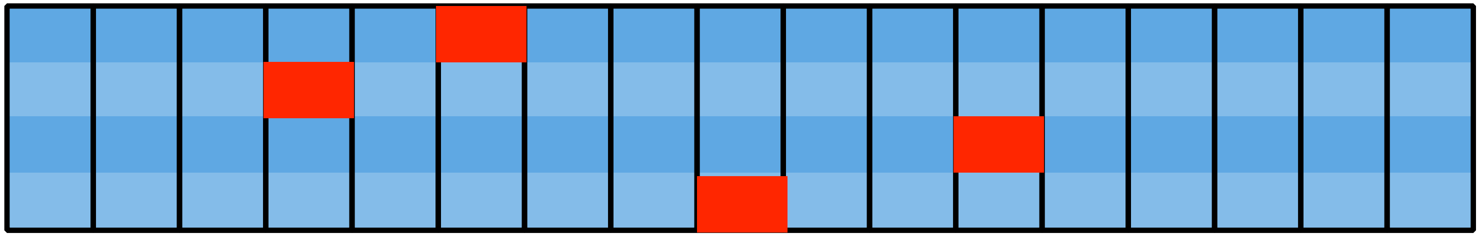
$$\mathbf{E}\left[w[i,h(i,x)] - n_x\right] = \frac{n}{m} \text{ hence } \Pr\left\{w[i,h(i,x)] - n_x > e\frac{n}{m}\right\} \leq e^{-1}$$

- Minimum boosts probability exponentially (only need to ensure that there's at least one random variable which satisfies the condition)

$$\Pr\left\{c_x - n_x > e\frac{n}{m}\right\} \leq e^{-d}$$

Properties of the count min sketch

- Linear statistics



- Sketch of two sets is sum of sketches
  - We can aggregate time intervals
- Sketch of lower resolution is linear function
  - We can compress further at a later stage

Estimating the 2nd moment (F_2) of Data Streams

Recap: Definition of Frequency Moments

- Characterize the skewness of distribution
  - Sequence of instances
  - Instantaneous estimates

$$F_p = \sum_{i=1}^{|X|} X[i]^p$$

$X[i] =$ No. of times (i.e. repetitions) that the i-th type of items has been observed and $|X|$ is the no. of different types of items = dimension of array(vector) $X[\ ]$.

- Special cases
  - F_0 is number of distinct items
  - F_1 is number of items (trivial to estimate)
  - F_2 describes 'variance', the so-called *Gini's index of Homogeneity* (used e.g. for database query plans)
  - $F^*_\infty = \max_i \{ X[i] \}$

Why F_2 (i.e. *L2*-norm of the vector/array *X[ ]*) ?

- Database join (on A):
  - All triples (Rel1.A, Rel1.B, Rel2.B) s.t. Rel1.A=Rel2.A
- Self-join: if Rel1=Rel2
- Size of self-join:

  $\sum_{\text{val of A}} \text{Rows(val)}^2$
- Updates to the relation increment/decrement Rows(val)

Rel1

A	B
Lec1	distinct
Lec1	elements
Lec1	norm
Lec2	L2
Lec2	norm
….	

Rel2

A	B
Lec1	distinct
Lec1	elements
Lec1	norm
Lec2	L2
Lec2	norm
….	

A	Rel1.B	Rel2.B
Lec1	distinct	distinct
Lec1	distinct	elements
Lec1	distinct	norm
Lec1	elements	distinct
Lec1	elements	elements
	….	

The Alon-Matias-Szegedy (AMS) Sketch for F_2 [AMS96] - Godel Prize winner in 2005

Choose $r_1, r_2, \ldots, r_m$ to be i.i.d. random variables with:

$$\Pr[r_i = 1] = \Pr[r_i = -1] = \frac{1}{2}$$

Maintain $Z = \sum_{i=1}^{|X|} r_i X[i]$ under increments/decrements to $X[i]$,

e.g. when 3 new items of type i arrive, update $Z := Z + 3r_i$

Algorithm I: $Y = Z^2 = \left(\sum_{i=1}^{|X|} r_i X[i] \right)^2$

"Claim": Y approximates $F_2 \triangleq \sum_{i=1}^{|X|} X[i]^2$ with "good chances".

Error Analysis (1/3)

Approach:

Use Chebyshev inequality to bound the error for using Y to estimate F_2

=> Need to derive the Expectation and Variance of Y.

$$E[Y] = E[Z^2] = E\left[\left(\sum_i r_i X[i]\right)^2\right]$$

$$= E\left[\sum_i \sum_j r_i X[i] r_j X[j]\right] = \sum_i \sum_j X[i]X[j]E\left[r_i \cdot r_j\right]$$

We have:

For $i \neq j,\ E\left[r_i \cdot r_j\right] = E\left[r_i\right] \cdot E\left[r_j\right] = 0 \Rightarrow$ those terms will disappear

For $i = j,\ E\left[r_i \cdot r_j\right] = E\left[r_i \cdot r_i\right] = 1$

Therefore

$$E[Y] = E[Z^2] = \sum_i X[i]^2 = F_2$$

$\Rightarrow Y$ is an unbiased estimator of F_2

Error Analysis (2/3)

The 2nd moment of Y = The 2nd moment of $Z^2 = E\left[Z^4\right]$

But $\quad Z^4 = \left(\sum_i r_i X[i]\right)\left(\sum_j r_j X[j]\right)\left(\sum_k r_k X[k]\right)\left(\sum_l r_l X[l]\right)$

This can be decomposed into the sum of:

$\sum_i \left(r_i X[i]\right)^4 \Rightarrow \text{Expectation} = \sum_i X[i]^4;$

$\binom{4}{2}\sum_{i<j}\left(r_i \cdot r_j \cdot X[i] \cdot X[j]\right)^2 \Rightarrow \text{Expectation} = 6\sum_{i<j} X[i]^2 X[j]^2;$

The remaining terms involve single multiplier

$r_i \cdot X[i]$ (e.g., $r_1 X[1] r_2 X[2] r_2 X[2] r_3 X[3]$) $\Rightarrow \text{Expectation} = 0$

Therefore,

$$E\left[Z^4\right] = \sum_i X[i]^4 + 6\sum_{i<j} X[i]^2 X[j]^2$$

$$Var\left(Y\right) = Var\left(Z^2\right) = E\left[Z^4\right] - E^2\left[Z^2\right] = \sum_i X[i]^4 + 6\sum_{i<j} X[i]^2 X[j]^2 - \left(\sum_i X[i]^2\right)^2$$

$$= \sum_i X[i]^4 + 6\sum_{i<j} X[i]^2 X[j]^2 - \sum_i X[i]^4 - 2\sum_{i<j} X[i]^2 X[j]^2 = 4\sum_{i<j} X[i]^2 X[j]^2 \le 2\left(\sum_i X[i]^2\right)^2 = 2F_2^2$$

Error Analysis (3/3)

Thus, we have an estimator $Y = Z^2$ where $E\left[Y\right] = \sum_i X[i]^2$ and $Var\left[Y\right] = \sigma^2 \le 2\left(\sum_i X[i]^2\right)^2$

Recall the Chebyshev Inequality:

$$\Pr\left[\left|W - E\left[W\right]\right| \ge c\sigma_W\right] \le 1/c^2 \text{..................}(*)$$

Consider the following Algorithm:

1. Maintain $Z_1, Z_2, \ldots Z_K$ (and thus $Y_1,\ Y_2, \ldots, Y_K$); define $Y' = \sum_{k=1}^{K} Y_k / K$
2. Compute $E\left[Y'\right] = K \cdot E\left[Y\right] / K = \sum_i X[i]^2$
3. Compute $Var\left[Y'\right] = \sigma'^2 = \frac{1}{K} Var\left[Y\right] = \frac{1}{K}\sigma^2 \le \frac{2}{K}\left(\sum_i X[i]^2\right)^2$

Sub. $W = Y'$ into (*), we have the following guarantee on estimation error:

$$\Pr\left[\left|Y' - \sum_i X[i]^2\right| \ge c\sqrt{\frac{2}{K}} \sum_i X[i]^2\right] \le 1/c^2 \Leftrightarrow \Pr\left[\frac{\left|Y' - F_2\right|}{F_2} \ge c\sqrt{\frac{2}{K}}\right] \le 1/c^2$$

Setting c to some constant according to the desirable error requirement and $K = O\left(1/\varepsilon^2\right)$ to yield:

An $(1 \pm \varepsilon)$-approximation with probability $1/c^2$, e.g., with $c = 10$ and $\varepsilon = 0.05$, we can guarantee a less-than 5% estimation error at least 99% of the time by setting $K = \left(10\sqrt{2}/0.05\right)^2 = 80{,}000$.

More Networking Applications of Data Stream Algorithms

- Elephant flow detection and counting
  - Find flow with large size
  - Billing and accounting
  - Sample and hold algorithm [Estan02] and
  - The Run-based schemes, e.g. co-incidence or 2-in-a-row, [Kodialam04], [Hao04], [Hao05].
- Flow entropy
  - Calculate the entropy of flows
  - Measures information randomness
  - Traffic engineering, anomaly detection, clustering
  - Algorithm [Lall06] based on estimating frequency moments algorithm [AMS96]
- OD flow entropy
  - Calculate the entropy of OD flows
  - Traffic engineering, network wide anomaly detection
  - Algorithm [Zhao07] based on estimating frequency moments algorithm [Indyk00]

Source: Wikipedia (Japanese rock garden)

Further reading

- Muthu Muthukrishnan's tutorial http://www.cs.rutgers.edu/~muthu/stream-1-1.ps
- Alon Matias Szegedy http://www.sciencedirect.com/science/article/pii/S0022000097915452
- Count-Min sketch https://sites.google.com/site/countminsketch/
- Bloom Filter survey by Broder & Mitzenmacher http://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf
- Metwally, Agrawal, El Abbadi (space saving sketch) http://www.cs.ucsb.edu/research/tech_reports/reports/2005-23.pdf
- Berinde, Indyk, Cormode, Strauss (space optimal bounds for space saving) http://www.research.att.com/people/Cormode_Graham/library/publications/BerindeCormodeIndykStrauss10.pdf
- Graham Cormode's tutorial http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf
- Flajolet-Martin 1985 http://algo.inria.fr/flajolet/Publications/FlMa85.pdf

References

[FM85] Probabilistic Counting Algorithms for Data Base Applications, Phillippe Flajolet,G.Nigel Martin, Journal of Computer and System Sciences, 1985.

[DF03] Loglog counting of large cardinalities, M. Durand and P. Flajolet, European Symposium on Algorithms 2003

[FFGM07] Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm, P. Flajolet, Eric Fusy, O. Gandouet, and F. Meunier, Conference on Analysis of Algorithms, 2007

[HNH13] S. Heule, M. Nunkesser, A.Hall, "HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm," EDBT/ICDT 2013.

[CKY14] R.Cohen, L.Katzir, A.Yehezkel, "A unified scheme for generalizing cardinality estimators to sum aggregation," Information Processing Letters, 2014.

[GKMS01] Surfing wavelets on streams: One pass summaries for approximate aggregate queries, A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss., VLDB Journal, 2001.

[Whang 90] A linear-time probabilistic counting algorithm for database applications, K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, ACM Transaction on Database Systems (TODS), 1990

[AMS96] The space complexity of approximating the frequency moments, Noga Alon, Yossi Matias and Mario Szegedy, ACM STOC 1996, JCSS 1999

[Indyk00] Stable distributions, pseudorandom generators, embeddings and data stream computation, P. Indyk, ACM FOCS 2000, JACM 2006

[CM05] What's hot and what's not: tracking most frequent items dynamically, Graham Cormode and S. Muthukrishnan, ACM TODS'05

[CCFC02] Finding frequent items in data streams, Moses Charikar, Kevin Chen, and Martin Farach-Colton3, ICAPL'02

[DLM02] Frequency Estimation of Internet Packet Streams with Limited Space, Erik D. Demaine, et al., ESA'02

[KSP03] A simple algorithm for finding frequent elements in streams and bags, Richard M. Karp, et al., ACM TODS'03

[MAA06] An integrated efficient solution for computing frequent and top-k elements in data streams, Ahmed Metwally, et al., ACM TODS'06

[Estan03] Bitmap algorithms for counting active flows on high speed links, Cristian Estan George Varghese Michael Fisk, ACM Internet Measurement Conference (IMC) 2003, ACM Transaction on Networking (TON) 2006

[ZKWX05] Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices, Q. G. Zhao, A. Kumar, J. Wang, and J. Xu., ACM Sigmetrics 2005

[Estan02] New directions in traffic measurement and accounting, Cristian Estan and George Varghese, ACM Sigcomm 2002

[Kodialam04] Run-bAsed Traffic Estimator (RATE): A Simple, Memory Efficient Scheme for Per-flow Rate Estimation, Murali Kodilam, T.V. Lakshman, S. Mohanty, Infocom 2004.

[Hao04] ACCEL-RATE: A faster memory efficient scheme for Per-flow Rate Estimation, Fang Hao, Murali Kodialam, T.V. Lakshman, ACM Sigmetrics 2004.

[Hao05] Fast, memory-efficient traffic estimation by co-incidence counting, Fang Hao, Murali Kodialam, T.V.Lakshman, Hui Zhang, Infocom 2005.

[Lall06] Data streaming algorithms for estimating entropy of network traffic, Ashwin Lall et al., ACM Sigmetrics 2006

[Zhao07] A Data Streaming Algorithm for Estimating Entropies of OD Flows, ACM IMC 2007

[Kumar04] Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution, Abhishek Kumar et al., ACM Sigmetrics 2007