Mining Data Streams

Cam Tu Nguyen

阮锦绣

Software Institute, Nanjing University
nguyenct@lamda.nju.edu.cn
ncamtu@gmail.com

Data Streams

- In many data mining situations, we know the entire data set in advance

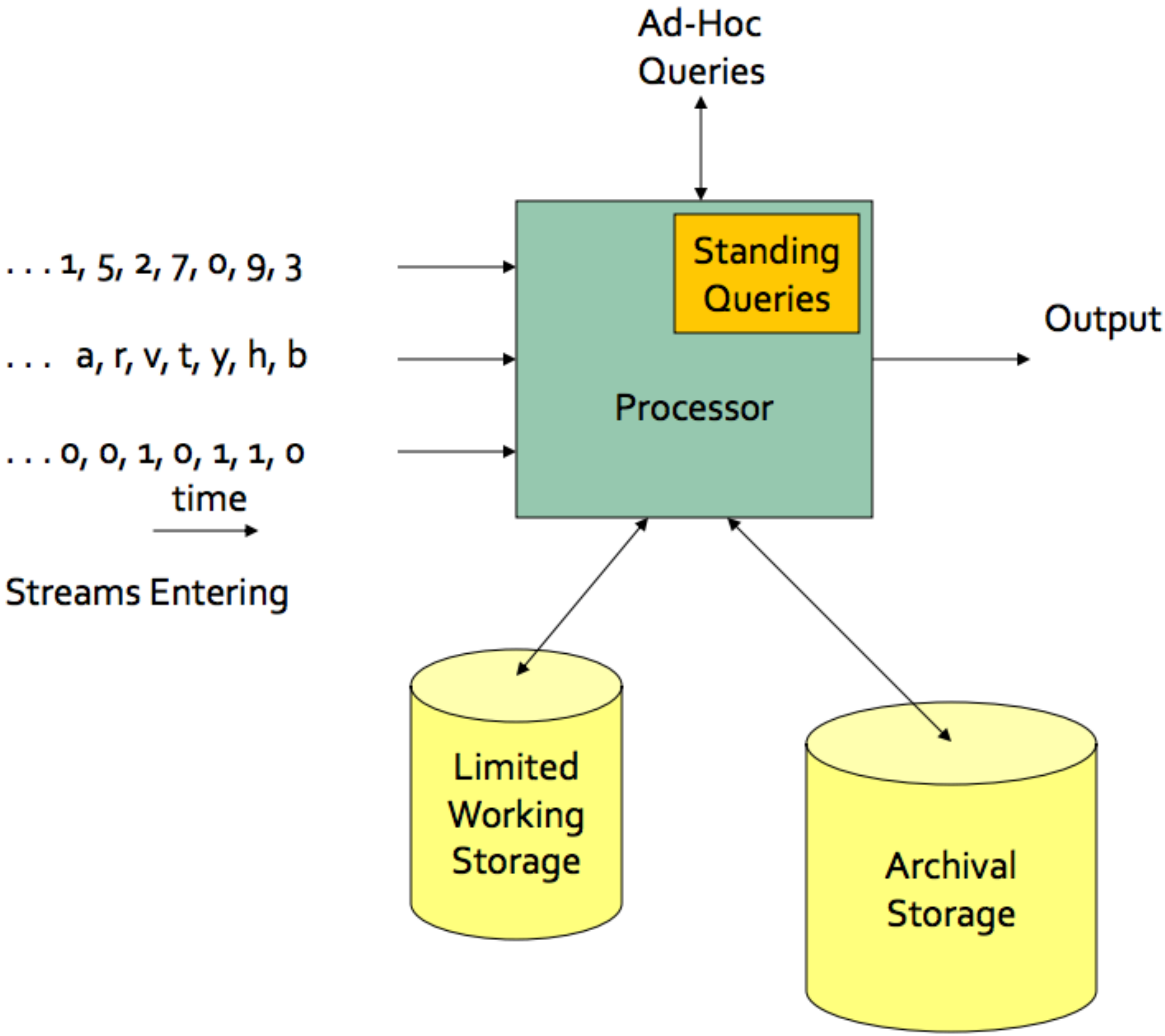
- Sometimes the input rate is controlled externally
  - Google queries
  - Twitter or Facebook status updates

Outline

- The Stream Data Model
- Sampling Data in a Stream
- Filtering Streams
- Counting Ones in a Window
- Clustering for Streams

The Stream Model

- Input tuples can enter rapid rate, at one or more number of streams
  - Streams need not have the same data rates
  - The time between elements of one stream need not to be uniform
- Systems can not store the entire stream accessibly
- How to make critical calculations about the stream using a limited amount of (primary or secondary) memory?



Ad-Hoc Queries
. . . 1, 5, 2, 7, 0, 9, 3
. . . a, r, v, t, y, h, b
. . . 0, 0, 1, 0, 1, 1, 0
time
Streams Entering
Standing Queries
Processor
Output
Limited Working Storage
Archival Storage

Two Forms of Query

- Ad-hoc queries: Normal queries asked one time about streams
  - Example: what is the maximum value seen so far in stream S?
- Standing queries: Queries that are, in principle, asked about the stream at all times
  - Example: Report each new maximum value ever seen in stream S.

Applications

- Mining query streams
  - Google wants to know what queries are more frequent today than yesterday
- Mining click streams
  - Yahoo! Wants to know which of its pages are getting an unusual number of hits in the past hour
- IP packets can be monitored at a switch
  - Gather information for optimal routing
  - Detect of denial-of-service attacks
- Mining surveillance data
  - Each surveillance camera produces a stream of images at intervals like one second.
  - Detect security threats

Outline

- The Stream Data Model
- Sampling Data in a Stream
- Filtering Streams
- Counting Ones in a Window
- Clustering for Streams

Sampling Data in a Stream

- Objective: extract reliable samples from a stream.
- **A Motivating Example:**
  - A search engine receives a stream of queries, and it would like to study the behavior of typical users.
  - The stream consists of tuples (user, query, time)
  - Answer queries such as "What fraction of the typical user's queries were repeated over the past month?"
  - Assume that we wish to store only 1/10th of the stream elements.
- **Obvious but not (quite) right approach**
  - Generate a random number between 0 and 9
  - Store the tuple if the generated number is 0
  - Counter example: suppose a user generates **s** queries in which **d** were queried twice, and none is queried more than twice.
    - Correct answer: d/(s+d)
    - Answer we get with the above solution is d/(10s+19d)

Sampling Data in a Stream

- **Solution**:
  - Stream of tuples with keys
    - Key is some subset of each tuple's components
    - E.g., tuple is (user, search, time); key is user
    - Choice of key depends on application
  - To get a sample of size a/b
    - Hash each tuple's key uniformly into b buckets
    - Pick the tuple if its hash value is at most a

Maintaining a fixed-size sample

- Suppose we need to maintain a sample of size exactly **s**
  - E.g., main memory size constraint
- Don’t know length of stream in advance
  - In fact, stream could be infinite
- Suppose at time t we have seen n items
  - Ensure each item is in sample with equal probability s/n

Solution

- Store all the first s elements of the stream
- Suppose we have seen n-1 elements, and now the n^{th} element arrives (n > s)
  - With probability s/n, pick the nth element, else discard it
  - If we pick the n^{th} element, then it replaces one of the s elements in the sample, picked at random

- Claim: this algorithm maintains a sample with the desired property
  - That is, on the stream with length n, all position have equal probability s/n of being chosen.

Proof: By induction

- Assume that after n elements, the sample contains each element seen so far with probability s/n
- When we see element n+1, it gets picked with probability s/(n+1)
- For elements already in the sample, probability of remaining in the sample is:

$$(1-\frac{s}{n+1})\frac{s}{n}+(\frac{s}{n+1})(\frac{s-1}{s})(\frac{s}{n})=\frac{s}{n+1}$$

If the (n+1)th position didn't get picked

If the (n+1)th position got picked

Sliding Windows

- A useful model of stream processing is that queries are about a **window** of length N – the N most recent elements received.
  - Alternative: elements received within a time interval T

- Interesting case: N is so large it can not be stored in main memory
  - Or, there are so many streams that windows for all do not fit in main memory

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Outline

- The Stream Data Model
- Sampling Data in a Stream
- Filtering Streams
- Counting Ones in a Window
- Clustering for Streams

Filtering Streams

- **Filtering**: accept those tuples in the stream that meet a criterion.

- **Criterion**:
  - A property of the tuple that can be calculated (easy)
  - Lookup for membership in a set, where the set is too large to store in the main memory (interesting)
    - **Bloom Filter**

- Motivating Examples
  - Filtered crawled Urls in Web Crawlers
  - Filtered emails in a blacklist.
  - Filtered malwares in a blacklist.

Filtering Streams: The Bloom Filter

- **A Bloom Filter consists of**
  1. A large array of **n** bits, initially all 0's
  2. A collection of hash functions h_1, h_2, …, h_k. Each hash function maps "key" values to n buckets, corresponding to the **n** bits of the bit array.
  3. A set S of m key values.
- **Lookup:**
  - Suppose element y appears in the stream, and we want to know if we have seen y before
  - Compute $h_i(y)$ for each hash function 1<=i<=k
  - If all the resulting bit positions are 1, say we have seen y before.
    - False positive is possible
  - If at least one of these positions is 0, say we have not seen **y** before
    - We are certainly right.

Filtering Streams: Bloom Filter Example

- Use n=11 bits for our filter
- Stream elements = integers
- Use two hash functions:
  - h1(x) =
    - Take odd-numbered bits from the right in the binary representation of x.
    - Treat it as an integer z
    - Result is z modulo 11
  - h2(x)= same as h1(x) but take even-numbered bits.

Filtering Streams: Bloom Filter Example

- **Lookup**:
  - Suppose we have the same Bloom Filter as before, and we have set the filter to 10100101010
  - Query: have we seen y=118?

- **Solution**:
  - y=118=1110110 (in binary)
  - h1(y)=14 modulo 11 = 3
  - h2(y)=5 modulo 11=5
  - Bit 5 is 1 but bit 3 is 0, so we are sure y is not in the set.

Filtering Streams: Bloom Filter Example

Stream element	h_1	h_2	Filter contents
			00000000000
25 = 11001	5	2	00100100000
159 = 10011111	7	0	10100101000
585 = 1001001001	9	7	10100101010

Note: bit 7 was already 1.

Analysis of Bloom Filtering

- Probability of a false positive depends on the density of 1's in the array and the number of hash functions.
  - =(fraction of 1's)$^{\text{\# of hash functions}}$.

- The number of 1's is approximately *the number of elements inserted* times *the number of hash functions.*
  - But collisions lower that number slightly.

Analysis of Bloom Filtering: Throwing Darts

- Turning random bits from 0 to 1 is like throwing **d** darts at **t** targets, at random.
- How many targets are hit by at least one dart?
- Probability a given target is hit by a given dart = 1/t
- Probability that none of **d** darts hit a given target is $(1-1/t)^d$
- Rewrite as $(1-1/t)^{t(d/t)}$ ~= $e^{-d/t}$.
  - Since $(1-1/t)^t$~=1/e for t is large

Analysis of Bloom Filtering: Example of Throwing Darts

- Suppose we use an array of 1 billion bits, 5 hash functions, and we insert 100 million elements.
- That is, $t=10^9$, and $d=5*10^8$.
- The fraction of 0's that remain will be $e^{-1/2}=0.607$
- Density of 1 = 0.393
- Probability of a false positive = $(0.393)^5=0.00937$

Outline

- The Stream Data Model
- Sampling Data in a Stream
- Filtering Streams
- Counting Ones in a Window
- Clustering for Streams

Counting Ones in a Window

- You can show that if you insist on an exact sum or count of the elements in a window, you cannot use less space than the window itself.
- But if you are willing to accept an approximation, you can use much less space.
- We'll consider the simple case of counting elements of a certain type as a special case.
- Sums are a fairly straightforward extension.

Counting Bits

- **Problem:** given a stream of 0's and 1's, be prepared to answer queries of the form "how many 1's in the most recent k bits?" where k <= N.
- Naïve solution: store the most recent N bits.
- But answering the query will take O(k) time.
  - Very possibly too much time.
- And the space requirements can be too great
  - Especially if there are many streams to be managed in main memory at once or N is huge.

Example: Bit Counting

- Count recent hits on URL's belong to a site
- Stream is a sequence of URL's
- Window size N=1 billion
- Think of the data as many streams – one for each URL
  - Bit on the stream for URL x is 0 unless the actual stream has x.

DGIM Method

- Name refers to the inventors
  - Datar, Gionis, Indyk, and Motwani

- Store only $O(\log^2 N)$ bits per stream, where N is the window size.

- Give approximate answer, never off by more than 50%.
  - Error factor can be reduced to any epsilon > 0, with more complicated and proportionally more stored bits.

Timestamps

- Each bit in the stream has a *timestamp*, starting 0, 1, ..
- Record timestamps modulo N (the window size), so we can represent any relevant timestamp in $O(\log_2 N)$ bits.

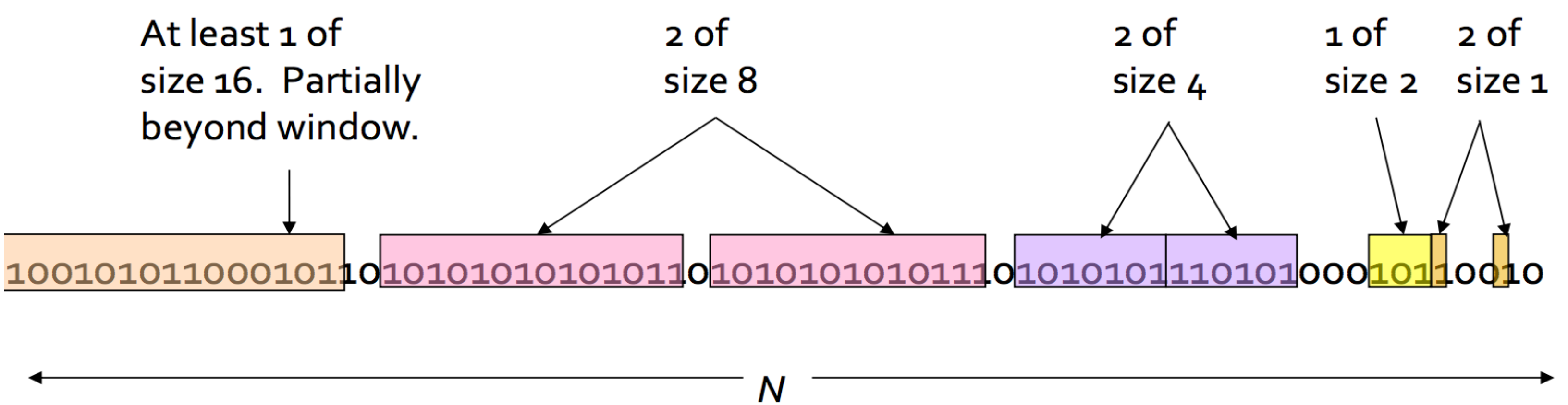
Buckets

- A **bucket** is a segment of the window; it is represented by a record consisting of
  - The timestamp of its end O(logN) bits
  - The number of 1's between its beginning and end
    - Number of 1's = **size** of the bucket

- Constraint on bucket sizes: number of 1's must be a power of 2.
  - Thus, only O(loglogN) bits are required for this count.

Representing a Stream by Buckets

- Bucket requirements:
  1. The right end of a bucket is always a position with a 1.
  2. Every position with a 1 is in some bucket
  3. Either one or two buckets any given size, up to some max size.
  4. All sizes must be a power of 2.
  5. Buckets do not overlap
  6. Buckets are sorted by size: older buckets are not smaller than newer buckets

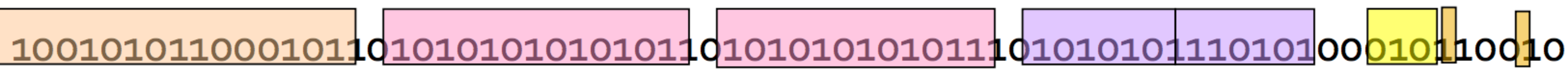


Updating Buckets

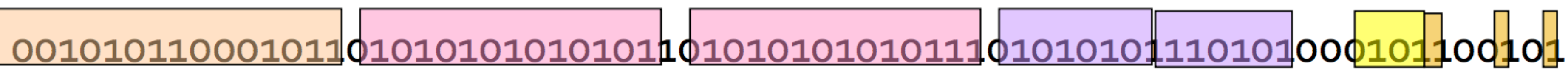
- When a new bit comes in, drop the last (oldest) bucket if its end-time is prior to N time units before the current time.
- If the current bit is 0, no other changes are needed.
- If the current bit is 1
  1. Create a new bucket of size 1, just for this bit
     - End timestamp = current time
  2. If there are now three buckets of size 1, combine the oldest two into a bucket of size 2.
  3. If there are now three buckets of size 2, combine the oldest two into a bucket of size 4.
  4. And so on….

Example: Managing buckets

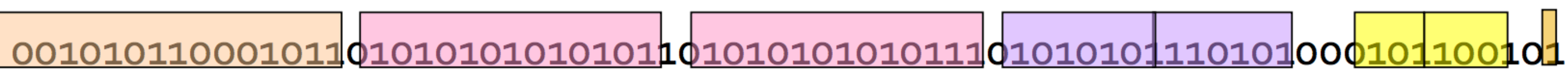
Initial



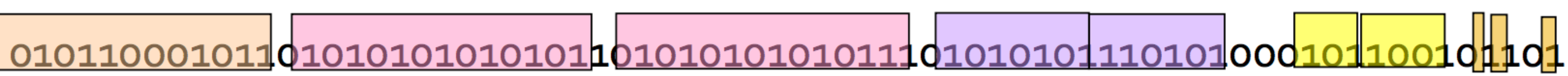
1 arrives; makes third block of size 1.



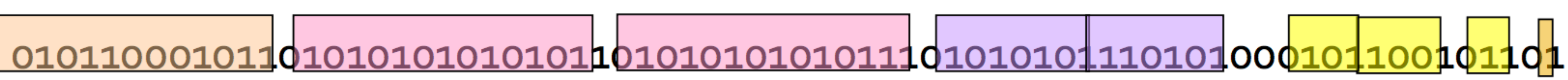
Combine oldest two 1's into a 2.



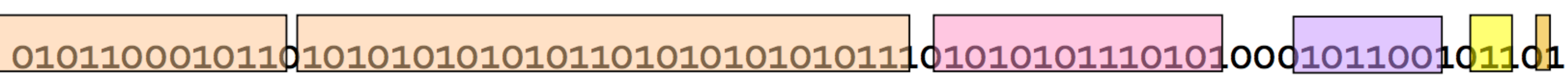
Later, 1, 0, 1 arrive. Now we have 3 1's again.



Combine two 1's into a 2.

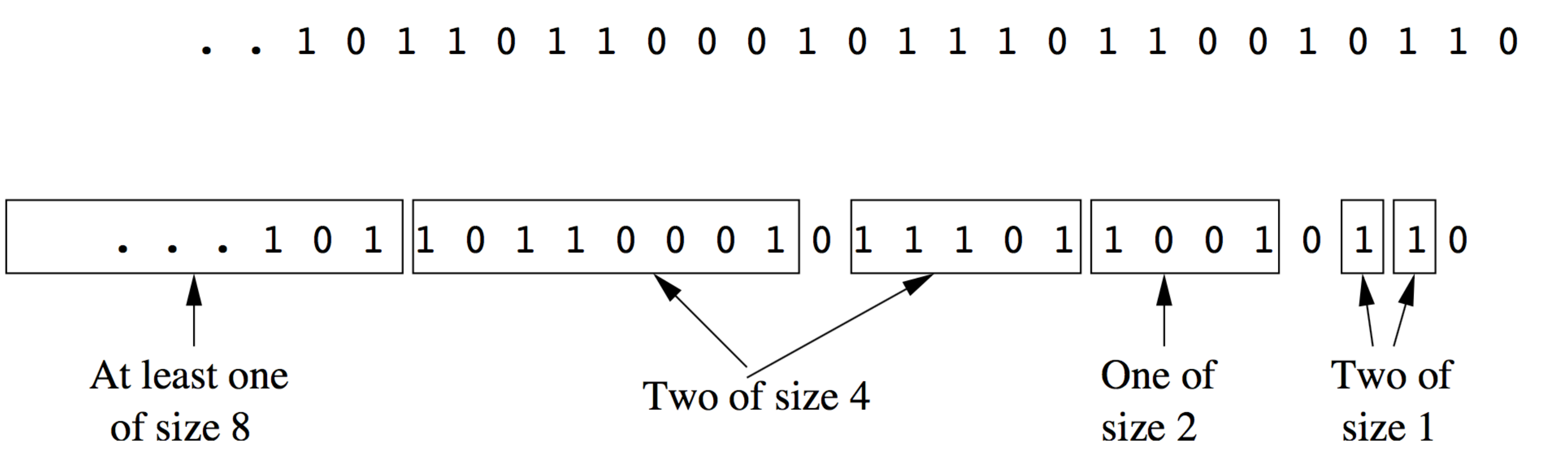


The effect ripples all the way to a 16.



Querying

- To estimate the number of 1’s in the most recent k<=N bits
  - Find the bucket **b** with the earliest timestamp that includes at least some of the **k** most recent bits.
  - Estimate the number of 1’s to be the sum of the sizes of all the buckets more recent than bucket **b**, plus half the size of b itself.
- Example: k=10



Error Bound

- Suppose the oldest bucket within the range has size 2^i.
- Then by assuming 2^{i-1} of its 1's are still within the window, we make the error at most 2^{i-1}.
- Since there is at least one bucket of each of the sizes less than 2^i, and at least 1 from the oldest bucket, the true sum is no less than 2^i.
- Thus the error is at most 50%.

Space Requirement

- We can represent one bucket in O(logN) bits
- No bucket can be of the size greater than N
- There are at most 2 buckets of the same size (in range 1 to logN)
- There are at most 2logN buckets
- Thus, the space required is $O(\log^2 N)$

Outline

- The Stream Data Model
- Sampling Data in a Stream
- Filtering Streams
- Counting Ones in a Window
- Clustering for Streams

Clustering for Streams

- A stream of data points in some space (Euclidean or non-Euclidean)
- Problem:
  - The centroids or clustroids (representative points of clusters) of the best clusters formed from the last m of the points, for any m<=N.
- We will consider some possible solutions, depending on our assumptions about how clusters evolve in a stream.

BDMO algorithm

- BDMO (for the authors, B. Babcock, M. Datar, R.Motwani, and L.O'Callaghan)
- BDMO algorithm builds on the methodology for counting ones in the stream (DGIM algorithm). The key similarities and differences are:
  1. Like DGIM, the points of the stream are partitioned into, and summarized by **buckets**.
     - Here, the **size** of a bucket is the number of points it represents, rather than the number of stream elements that are 1.
  2. As before, the sizes of buckets obey the restriction that there are **one or two of each size**, up to some limit.
  3. We do not assume that the sequence of allowable buckets sizes starts with 1. Rather, they are required only to form a sequence where each size is twice the previous size., e.g. 3, 6, 12, 24…
  4. Bucket sizes are no decreasing as we go back in time.

BDMO algorithm

- The contents of a bucket:
  - The size of the bucket
  - The timestamp of the bucket
  - A collection of records that represent the clusters into which the points of that bucket have been partitioned.
    - The number of points in the cluster
    - The centroid or clustroid of the cluster.
    - Any other parameters needed to merge clusters and maintain approximations to the full set of parameters for the merged cluster.

Initializing Buckets

- Our smallest bucket size will be p times (a power of 2).
  - Thus, every **p** stream elements, we create a bucket, with the most recent **p** points.
  - The timestamp for each bucket is the timestamp of the most recent point in the bucket.
  - Cluster points in the buckets (or leave them as one point as one cluster)
    - Compute the centroids or clustroids for the clusters & count the points in each cluster.

Merging Buckets

- Similar to merging buckets in the problem of counting 1’s for streams.
  1. If some bucket has a timestamp that is more than N time units prior to the current time, drop it.
  2. If we have three buckets of size **p,** merge the oldest two of the three.
  3. If there are now 3 buckets of size **2p**, merge the oldest two of the three.
  4. If there are now 3 buckets of size 4p, …

Merging Buckets

- To merge two consecutive buckets
  1. The size of the new bucket is twice the sizes of the two buckets being merged.
  2. The timestamp for the merged bucket is the timestamp of the more recent of the two consecutive buckets.
  3. Consider merging clusters in 2 buckets (depending on the clustering algorithm)

Merging Buckets: Example

- Clustering algorithm: K-means in a Euclidean space.
- Clusters are represented by centroids, the #points in each cluster.
- We pick p=k
- Problem: merging 2 buckets, each containing K clusters.
- Solution
  - Find the best matching from K clusters of the first bucket and K clusters from the second bucket.
  - Best matching = minimum distance between centroids.
  - To merge two clusters (c1, n1) and (c2, n2), we create a new cluster with n=n1+n2 points and the new centroid

$$\mathbf{c} = \frac{n_1\mathbf{c}_1 + n_2\mathbf{c}_2}{n_1 + n_2}$$

Answering Queries

- Query: a request for the clusters of the most recent m points in the stream (m<=N).

- Solution:
  - Select the smallest set of buckets that cover **m** points.
    - Those buckets may not contain more than 2m points.
  - Assumption: the points between 2m and m+1 will not have radically different statistics from the most recent m points (for good approximation).
  - Pool all the clusters from the selected buckets.
    - For example: if you ask for K-clusters, we keep on merging until we reach K-clusters with the bucket-merging method in the previous slide.

Summary

- The Stream Data Model
- Sampling Data in a Stream
- Filtering Streams
- Counting Ones in a Window
- Clustering for Streams