Analysis of Massive Data Sets

Stream Data Model and Processing

Klemo Vladimir

Faculty of Electrical Engineering and Computing
Consumer Computing Laboratory

Stream Data Model and Processing

- Outline
  1. Introduction/Stream Data Model
  2. Sampling streams (*Random sample/Sliding windows*)
  3. Filtering streams (*Bloom filter*)
  4. Counting distinct elements (*Flajolet-Martin*)
  5. Estimating moments (*Alon-Matias-Szegedy*)
  6. Literature

The Stream Data Model

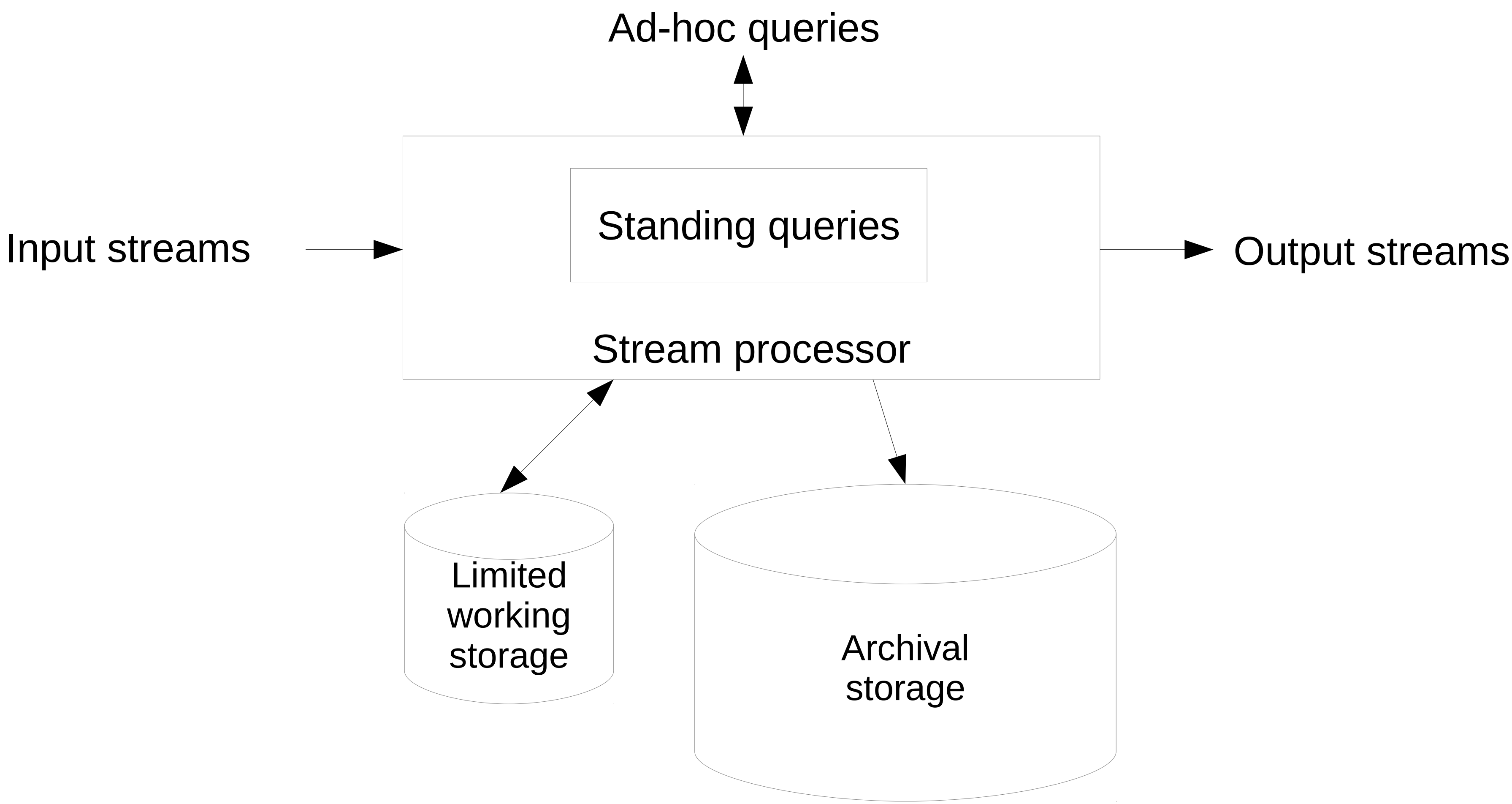
- Data arrives in *streams*
  - It must be processed immediately or stored
  - If not, then it is lost forever
- Data arrives *rapidly*
  - It is not feasible to store it all
- *Infinite* and *non-stationary* data
  - Controlled externally
    - Google queries, Twitter statuses...

The Stream Data Model

- Examples
  - Web traffic
    - Google search queries
    - Twitter public stream
  - Internet traffic
    - Routing IP packets
  - Sensor data
    - Data rate * Total number of sensors
  - Image data
    - Satellites, surveillance cameras

The Stream Data Model

- Data stream management system



The Stream Data Model

- Data stream management system
  - Analogy to DBMS
  - Archival storage
    - Not used to answer queries
  - Working storage
    - Limited – cannot store all the data from all the streams
    - Stores summarizes or parts of streams
    - Disk or main memory

The Stream Data Model

- Standing queries
  - Execute "permanently"
  - Example: temperature sensor
    - Alert on *x* degrees C
      - Depends only on most recent reading
    - Max temperature
      - Store current maximum
    - Average of *n* recent readings
      - First n readings
        - avg = sum/n
      - Next readings
        - x – new element, y – oldest element
        - new_avg = avg + (x – y)/n

The Stream Data Model

- Ad-hoc queries
  - Asked once about the current state of stream
  - Depend on stream summarization stored in working storage
    - *Sliding windows*
      - Store most recent elements of a stream
      - Stream management system keeps the window fresh
      - Removes oldest elements as new ones come in
    - *Random sample*
      - Store random, but representative sample of a stream

The Stream Data Model

- Online algorithms/learning *
  - In Machine Learning
  - Algorithm slowly adapts to the changes in data
    - For every sample in the data, the model is slightly updated
    - SVM, Perceptron ...

The Stream Data Model

- Conditions
  - Large number of streams
  - Stream elements enter at rapid rate
  - System cannot store the entire streams
- Challenge
  - *How to derive information from the stream using a limited amount of memory?*
- Accuracy/performance trade-off
  - It is much more efficient to get an *approximate* answer than an exact solution

2. Sampling streams

Sampling streams

- Motivating example: Search engine queries
  - Elements are tuples
    - (user, query, timestamp)
  - Need to study the behavior of typical users
    - Find *fraction of unique queries* in the past month
  - Sample 10% of the whole stream

Sampling streams

- Motivating example: Search engine queries
  - Obvious approach for 10% sample
    - For each search query, generate a random number in range 0 – 9
    - Store the query if number is 0
    - On average, 10% of the queries for each user after some time
      - the law of large numbers

Sampling streams

- Motivating example: Search engine queries
  - Obvious approach
    - What is the problem with this approach?
      - The fraction of **unique queries** in the sample will not be the fraction for the stream as a whole
      - Probability of a given query appearing to be unique in the sample is distorted by the sample
      - Example
        - Query appeared exactly two times in the whole stream
        - System sampled only first occurrence of the query
        - Query appears unique in the sample, but it is not unique in the stream

Sampling streams

- Motivating example: Search engine queries
  - Obvious approach: Random numbers 0-9
    - Finding unique queries
    - Suppose a query is unique
      - Query appears once in the whole stream

x

      - 10% chance of being in the sample
    - Suppose a query appears twice in the whole stream

x1 x2

x2 x1

10%*90%=9%

Total: **18%** chance of appearing **once** in the sample (looking unique)

Sampling streams

- Motivating example: Search engine queries
  - Obvious approach: Random numbers 0-9
    - **Overestimate** of the true fraction of unique queries
  - Problem
    - Independent selection
    - Each time query arrives, random number is generated
    - Based on query **position** in the stream
  - Solution
    - Sampling by **value**
    - Pick users, not searches

Sampling streams

- **Representative sample**
  - Sample by value: (user, query, t)
  - Keep list of all users with the membership information
    - if user is in the sample (True/False)
  - Procedure when new query arrives
    - User lookup
    - If membership and user is in the sample
      - add this query to the sample
    - If no membership yet
      - Generate random integer between 0 – 9
      - If 0: mark user as member of sample, else mark as not member
    - Problems
      - Not enough memory to keep the list of users
      - Expensive lookup for every query

Sampling streams

- Representative sample (hashing)
  - Sample by value: (user, query, t)
  - **b** = number of users
  - **a/b** = fraction of the users to store in the sample
  - Hash queries to **b** buckets (**0** – **b-1**)
    - Hashing as pseudobucketing
  - Store query in the sample if it is hashed to bucket with value less than **a**
    - All or none of specific user's queries are selected
    - Fraction of unique queries is the same as for the stream as a whole

Sampling streams

- Representative sample
  - Sample size
    - What if the total sample size is limited?
  - Bucket management
    - Use more buckets
    - Dynamically adjust the set of buckets
      - If sample gets too large:
        - Pick one bucket that is included in the sample
        - Delete elements from the selected bucket

Sampling streams

- Representative sample
  - Bucket management
    - Hash to 100 buckets
    - Sample: buckets 0 – 9
    - If the sample gets too big
      - get rid of bucket 9
      - 8, 7, …
  - Generalization: sampling key-value pairs
    - Stream elements are tuples
    - Sample is based on picking some set of keys

3. Filtering streams

Filtering streams

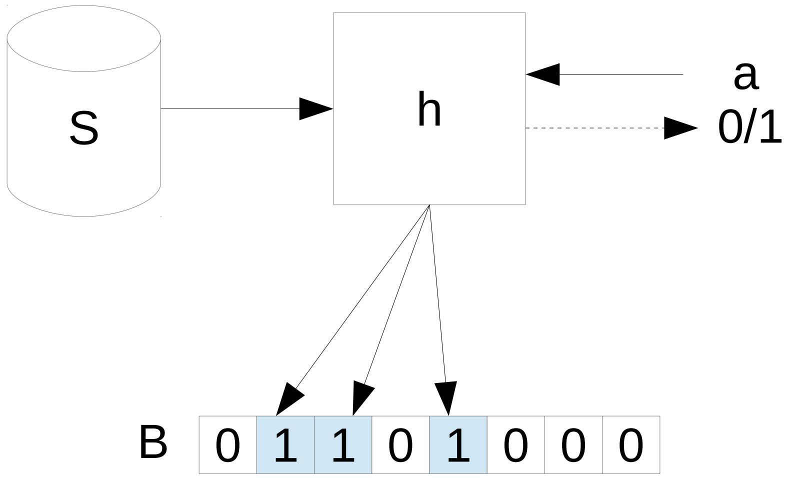
- Motivating example
  - Simple spam filter based on email addresses
  - Set S of 1 billion allowed email addresses
  - Not spam, if an email address is a member of S
- How to check membership of S?
  - Hash table
    - Naive solution
    - Works only if all of S can be stored in memory

Filtering streams

- **Bloom filter**
  - A Bloom filter placed on the stream of emails will answer with:
    - Not spam
      - Email address is member of S
    - Spam
      - Email address is not member of S
  - Reduced storage costs
  - Bloom filter can have false positives
    - And no false negatives

Filtering streams

- Bloom filter (simple)
  - Create a bit array **B** of **n** bits, initially all zeroes
  - Choose a hash function **h** with range **[0,n-1]**
  - Hash members of **S**, **h(s)**, and set bit **h(s)** to **1**
    - **B[h(s)] = 1**
  - Filtering
    - Emit **a** if **B[h(a)] = 1**



Filtering streams

- Bloom filter is:
  - An **array of n bits**, initially all 0's
  - A **collection of hash functions**
    - $h_1, h_2, \ldots, h_k$
    - Each hash function maps "key" values to **n** buckets, corresponding to the **n** bits of the bit-array
  - A **set S** of **m** key values.

Filtering streams

- Bloom filter
  - The purpose of the Bloom filter
    - allow through all stream elements whose keys are in S, while rejecting *most* of the stream elements whose keys are not in S
  - Procedure
    - Take each key value in S and hash it using each of the k hash functions
    - For some hash function h_i and some key value K in S
      - Set to 1 each bit that is $h_i(K)$

Filtering streams

- Bloom filter: Example
  - Stream elements: words
  - n=10 bits for the Bloom filter
  - Two hash functions:
    - $h_1(x)$ = len(x)%10
    - $h_2(x)$ = ord(x[0])%10

Filtering streams

- Bloom filter: Example
  - Input: “bloom filter”
  - Filter creation:
    - B = 0000000000
    - h_1(“bloom”) = 5, h_2(“bloom”) = 98%10=8
    - B = 0000**1**00**1**0
    - h_1(“filter”) = 6, h_2(“filter”) = 102%10=2
    - B = 00**1**00**11**0**1**0

Filtering streams

- Bloom filter: Example
  - Filter lookup:
    - We want to know if some element x was seen before
    - Compute h(x) for each hash function h
    - If all the resulting bit positions are 1
      - We have seen x before
      - Collisions: expect some number of false positives
    - If at least one of these positions is 0
      - We have not seen x before

Filtering streams

- Bloom filter: Performance
  - Probability of *false positive*
  - Depends on:
    - Density of 1's in the array (d_1)
    - Number of hash functions (k)
  - d_1=?
  - $\max(d_1)$ = num of inserted elements * k
  - In practice
    - collisions lower d_1
    - $d_1 < \max(d_1)$

$$d_1^k$$

Filtering streams

- Bloom filter: Performance
  - Model: throwing darts
  - **d** darts = number of elements * number of hash functions
  - **t** targets = number of bits in the bloom filter
  - d_1 = Number of targets hit by at least one dart

Filtering streams

- Bloom filter: Performance
  - Model: throwing darts
  - Probability of given target is hit by a given (one) dart
    - 1/t
  - Probability of one dart not hitting the given target
    - 1 – 1/t
  - Probability none of d darts hit a given target is
    - $d_0 = (1 - 1/t)^d$

Filtering streams

- Bloom filter: Performance
  - Model: throwing darts
  - Probability none of d darts hit a given target (d_0)

$$(1 - 1/t)^{d} = (1 - 1/t)^{t(d/t)} \sim= e^{-d/t}$$

$$(1 - 1/t)^{t} \rightarrow e^{-1}$$

Filtering streams

- Bloom filter: Performance
  - Model: throwing darts: Example
  - n = 1 billion, k = 5, m = 100 million
  - $t = 10^9$, $d = 5*10^8$
  - $d_0 = e^{-0.5} = 0.607$
  - $d_1 = 1 - d_0 = 0.393$ [$d_1 < 0.5$ (collisions)]
  - Probability of false positive
    - $d_1^k = 0.393^5 = 0.00937$ (<1%)

4. Counting distinct elements

Counting distinct elements

- Motivating example
  - Web site statistics
  - Number of unique users per month
  - Users are identified by IP addresses
  - 4 billion IP addresses
  - Obvious approach:
    - Keep list of all IP addresses with counts
    - Use efficient search structure (hash table, tree)

Counting distinct elements

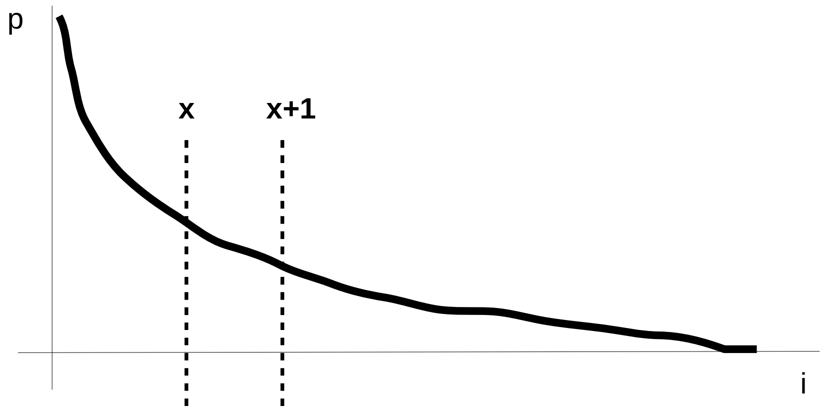
- Problem
  - Maintaining a count of the number of distinct elements in the stream
  - Not enough space to store the set of counts
- Applications
  - Number of unique visitors
  - Number of distinct products sold
  - Number of different words in a crawled web page

Counting distinct elements

- Flajolet-Martin algorithm
  - **n** = number of stream elements
  - **a** = stream element
  - Pick a hash function **h**
    - Maps stream elements to at least $\log_2 n$ bits
  - **r(a)** = number of trailing 0's in **h(a)**
  - **R** = **max(r(a))**, for all **a** in **S**
  - Distinct elements estimate:
    - 2^R

Counting distinct elements

- Flajolet-Martin algorithm
  - R depends only on distinct elements
    - If same element appears in the stream it will have r(a) the same
  - Probability that r(a) = i
    - Goes down exponentially



  - when *i* is increased by 1
    - Need to double the number of elements to reach prob. of x+1
    - Thus, 2^R is good estimate

Counting distinct elements

- Flajolet-Martin algorithm

r (trailing 0's)		Probability	Estimate of distinct elements	
1	******0	50%	$\frac{1}{2}$	2^1
2	*****00	25%	$\frac{1}{4}$	2^2
3	****000	.	.	
4	***0000	.	.	
5	**00000	.	.	
6	*000000	.	.	
		...		

Counting distinct elements

- Flajolet-Martin algorithm
  - Formal analysis
    - probability that hash ends in at least **r** zeroes (p_1)
      - 2^{-r}
    - Probability of **not** seeing a tail of **r** zeroes among **m** elements (p_0)
      - $(1-2^{-r})^m$

$$p_0 = (1-2^{-r})^m = (1-2^{-r})^{2\hat{}r * m * 2\hat{}-r} \sim= e^{-m * 2\hat{}-r}$$

Counting distinct elements

- Flajolet-Martin algorithm
  - Formal analysis
    - Probability of **not** finding hash with **r** tail
      - $m << 2^r$
        - $p_0 = e^{-m2\text{^}-r} \sim= 1 \rightarrow p_1 = 0$
      - $m >> 2^r$
        - $p_0 = e^{-m2\text{^}-r} \sim= 0 \rightarrow p_1 = 1$
    - Result
      - $\mathbf{2^R}$ is always around **m**!
        - Not too high, not too low

Counting distinct elements

- Flajolet-Martin algorithm
  - Problems
    - 2^R can be too large
  - Workaround
    - Use many hash functions h_i and obtain many samples R_i
    - Combine samples
      - Average: problem with large values
      - Median: all powers of 2
    - Solution
      - Partition samples into small groups
      - Count average in groups and take median of the averages

5. Estimating moments

Estimating moments

- Moments
  - Generalization of the *count distinct* problem
  - Stream has elements chosen from a set A of n values
  - m_i = number of occurrences of the *i*th element for any i
  - ***k*th-order moment** of the stream
    - Sum over all i of $(m_i)^k$

$$\sum_{i \in A} m_i^k$$

Estimating moments

- Special cases
  - 0th moment
    - Number of distinct elements
  - 1st moment
    - Length of the stream
  - 2nd moment
    - *Surprise number*
    - Measure of how uneven the distribution is

Estimating moments

- Special cases
  - 2nd moment
    - *Surprise number*
    - Example
      - Stream length: 100
      - 11 values appear
        - 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9 = 910
        - 90, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 = 8110

Estimating moments

- Alon-Matias-Szegedy
  - Works for all moments
  - Unbiased
  - Based on calculation of many random variables

Estimating moments

- Alon-Matias-Szegedy
  - 2^{nd} moment
    - $\sum_a m_a^2$
    - Suppose there is not enough space to count all the m's
    - Estimation using limited space
      - Compute some number of variables
      - For each variable **X** store
        - **X.element**
        - **X.value**

Estimating moments

- Alon-Matias-Szegedy
  - For each **X**
    - 1. choose position in the stream randomly
    - 2. set **X.element** to be the element found at that position
    - 3. initialize **X.value** to 1
    - 4. add **1** to **X.value** each time another occurrence of **X.element** is encountered

Estimating moments

- Alon-Matias-Szegedy
  - 2nd moment estimate

$$n(2X.value - 1)$$

  - For any X
  - Example
    - Stream: a, b, c, b, a, b
    - $n = 6$
    - $m_a = 2$, $m_b = 3$, $m_c = 1$
    - 2nd moment = $1^2 + 2^2 + 3^2 = 14$

Estimating moments

- Alon-Matias-Szegedy
  - Stream: a, b, c, b, a, b
  - 2 variables (X_1 and X_2)
  - Random positions: 1, 4
    - From position 1:
      - X_1.element = b, x_1.value = 3
    - From position 4:
      - X_2.element = a, x_2.value = 1
  - 2^{nd} moment estimate
    - $n(2X_1.\text{value} - 1) = 6(2*3 - 1) = 30$
    - $n(2X_2.\text{value} - 1) = 6(2*1 - 1) = 6$
    - Avg: 18

0	1	2	3	4	5
a	b	c	b	a	b

Estimating moments

- Alon-Matias-Szegedy
  - Why it works?
    - e(i) = stream element that appears at position i
    - c(i) = number of times element e(i) appears in the rest of the stream

$$E(n(2x-1)) = 1/n \sum_{1..n} n(2c(i)-1)$$

expected value of X

calculate average

n possible starting points

Estimating moments

- Alon-Matias-Szegedy
  - Why it works?
    - e(i) = stream element that appears at position i
    - c(i) = number of times element e(i) apperts in the rest of the stream

$$E(n(2x-1)) = 1/n \sum_{1..n} n(2c(i)-1)$$
$$= \sum_{1..n} (2c(i)-1)$$
$$= \sum_{1..n} 1+3+5+...+(2m_a - 1)$$

Last *a*: 2*1 – 1 = 1 Second last *a:* 2*2 – 1 = 3 First *a*: 2*m_a - 1

Estimating moments

- Alon-Matias-Szegedy
  - Why it works?
    - e(i) = stream element that appears at position i
    - c(i) = number of times element e(i) apperts in the rest of the stream

$$E(n(2x-1)) = 1/n \sum_{1..n} n(2c(i)-1)$$
$$= \sum_{1..n} (2c(i)-1)$$
$$= \sum_{1..n} 1+3+5+...+(2m_a - 1)$$
$$= \sum_{1..n} m_a^2$$

Literature

- J. Leskovec, A. Rajaraman, and J. D. Ullman, "**Mining of Massive Datasets**“, 2014, **Chapter 4. Mining Data Streams**

- P. Flajolet and G.N. Martin, “Probabilistic counting for database applications,” 24th Symposium on Foundations of Computer Science, pp. 76–82, 1983.
- N. Alon, Y. Matias, and M. Szegedy, “The space complexity of approximating frequency moments,” 28th ACM Symposium on Theory of Computing, pp. 20–29, 1996.