BIG DATA ANALYTICS

Mining Data Streams

ESSEC BUSINESS SCHOOL

Data Streams

- We do not know the entire data set in advance:
    - Google queries
    - Twitter or Facebook status updates
- Stream Management: the input rate is controlled externally
- The data as "infinite"

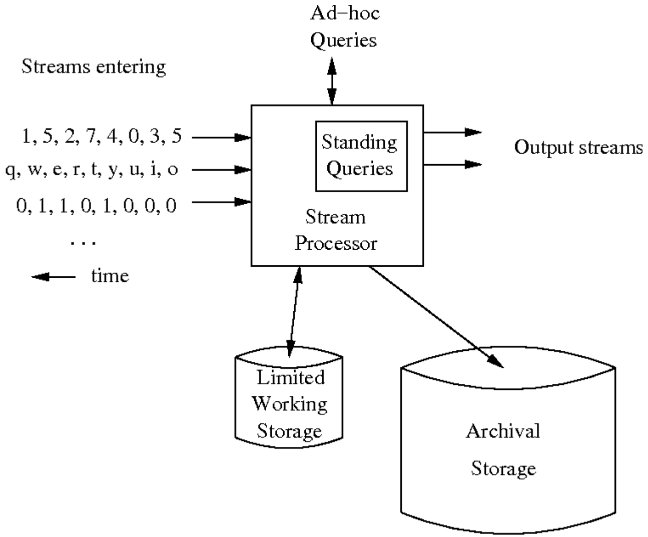
Outline

Outline

The Stream Model

- Stream: data arrives at a rapid rate
    - The elements of the stream: tuples
- It is not feasible to store all the data in active storage
- **Q: How to make critical calculations about the stream using a limited amount of memory?**
- **Summarization**

A Data-Stream-Management System



Mining of Massive Datasets, [Leskovec et al (2014)]

Sensor Data

- A temperature sensor sending back to a base station a reading of the surface temperature each hour
  - A stream of real numbers
  - The entire stream could be kept in main memory
- With a GPS unit it reports surface height: varies quite rapidly
- It produces ≈ 3.5 megabytes per day

Sensor Data

- A million sensors
- One for every 150 square miles of ocean
- We have 3.5 terabytes arriving every day:
    - **Q: what can be kept in working storage?**

Image Data

- Satellites:
  - send down to earth streams consisting of many terabytes of images per day
- Surveillance cameras:
  - many of them
  - each producing a stream of images at intervals $\sim$ one second
  - London: $\approx$ six million cameras

Internet and Web Traffic

- Web sites receive streams of various types
- Many interesting things can be learned from these streams:
  - An increase in queries like “sore throat” $\implies$ the spread of viruses
  - A sudden increase in the click rate for a link $\implies$
    - some news connected to that page
    - the link is broken

Types of Stream Queries

- *Standing queries*: permanently executed
  - E. g., an alert whenever the temperature exceeds 25 degrees
  - Depends only on the most recent stream element
  - Or each time a new reading arrives $\Longrightarrow$ the average of the 24 most recent readings
  - Need only to store the 24 most recent stream elements

Types of Stream Queries

- *Ad-hoc*: a question asked once about the current state of a stream or streams
- We cannot expect to answer arbitrary queries
- If we have some idea what kind of queries $\implies$ store appropriate parts or summaries of streams

Types of Stream Queries

- Types of queries one wants on answer on a data stream:
  - **Sampling data from a stream**
    - Construct a random sample
  - **Filtering a data stream**
    - Select elements with property x from the stream
  - **Counting distinct elements**
    - Number of distinct elements in the last k elements of the stream

Sliding Window

- Store a **sliding window** in the working store:
  - the most recent n elements of a stream
  - or all the elements that arrived within the last t time units
  - The stream-management system must keep the window fresh

Ad-hoc query: example

- **Q: Report the number of unique users over the past month**
- Each login = a stream element
- Window = all logins in the most recent month
- Associate the arrival time with each login:
  - we know when it no longer belongs to the window
- Maintain the entire stream of logins for the past month in working storage

Outline

Sampling Data in a Stream

- **Extracting reliable samples from a stream**
- Select a subset of a stream:
  - Queries about the selected subset
  - Problem: to have answers that are statistically representative of the stream as a whole
  - Allows the ad-hoc queries

A Motivating Example

- A search engine receives a stream of queries
  - **Problem:** study the behavior of typical users
- Stream of tuples: (user, query, time)
- **Answer questions such as:**:

  "What fraction of the typical user's queries were repeated over the past month?"
- Have space to store 1/10th of the stream elements.
- **Naive solution:**
  - Generate a random integer in $[0, \ldots, 9]$ for each query
  - Store the query if the integer is 0, otherwise discard

Solution: sample users

- This query cannot be answered by taking a sample of search queries!
- $\Longrightarrow$ pick 1/10th of the users, and take all their searches for the sample:
  - a search query arrives in the stream $\Longrightarrow$ is the user in the sample?
  - If yes, we add this search query to the sample
  - If no record of having seen this user before:
    - Generate a random integer between 0 and 9
    - If the number is 0, we add this user to our list

Generalized solution

- OK as long as we can keep the list of all users in main memory
- To avoid keeping the list of users: we hash each user name to one of ten buckets
    - If the user hashes to bucket 0, then accept this search query for the sample
    - Applied to the same user several times, we always get the same “random” number

The General Sampling Problem

- Our stream consists of tuples with n components
- A part of the components are the keys components
- To take a sample of size a/b:
    - we hash the key value for each tuple to b buckets
    - we accept the tuple for the sample if the hash value is less than a
- The selected key values will be approximately a/b of all the key values in the stream.

Varying the Sample Size

- The sample will grow as more of the stream enters the system
    - More searches for the same users
    - New users stream
- Limited budget for how many tuples from the stream can be stored
- $\Longrightarrow$ low the fraction of stored key values

Varying the Sample Size

- A hash function h from key values to a very large number of values $0, 1, ..., B-1$
- A threshold t (initially can be $= B-1$)
- The sample consists of tuples with $h(K) \leq t$
- If the number of stored tuples of the sample exceeds the allotted space: we lower t to $t-1$
- We can lower t by more than 1

Outline

Filtering Streams

- Selection, or **filtering**: we accept those tuples in the stream that meet a criterion
  - Accepted tuples are passed to another process
  - Other tuples are dropped
- Easy: the selection criterion is a property of the tuple that can be calculated.
- Harder when the criterion involves lookup for membership in a set
- Especially hard, when that set is too large to store in main memory
- **"Bloom filtering"**

Example: Email spam filtering

- $S =$ one billion "good" email addresses
- If an email comes from one of these, it is NOT spam
- The stream consists of pairs: an email address and the email itself
- The typical email address is 20 bytes or more
- We can not store S in main memory

First cut solution

- Given a set of keys S that we want to filter
- Create a bit array B of n bits, initially all 0s
- Choose a hash function h with range $[0, n)$
- Hash each member of S to a bit, and set that bit to 1
- All other bits of the array remain 0
- When a stream element arrives, we hash its email address
    - If the bit to which that email address hashes is 1 $\implies$ we let the email through
    - If the email address hashes to a $0 \implies$ we drop this stream element.

Creates false positives but no false negatives

- If the item is in S we surely output it, if not we may still output it

Email spam filtering

- One gigabyte of available main memory:
  - $|S| = 1$ billion email addresses
  - $|B| = 1GB = 8$ billion bits
- Approximately 1/8th of the bits will be 1 $\implies$
- $\approx$ 1/8th of the stream elements $\notin S$ will hash to a bit whose value is 1 $\implies$
- Some spam email will get through
- The majority of emails are spam $\implies$ eliminating 7/8th of the spam is a significant benefit
- If we want to eliminate every spam:
  - check for membership in S emails that get through the filter
  - use a cascade of filters

Bloom filter: Wrap-up

- **Bloom filters guarantee no false negatives, and use limited memory**
  - Great for pre-processing before more expensive checks
- Hash function computations can be parallelized

Outline

Counting Distinct Elements in a Stream

- Tricky in a reasonable amount of main memory
- Solution: use hashing and a randomized algorithm
- Approximate solution with little space needed per stream.

The Count-Distinct Problem

- **Problem:**
  - Stream elements are from some universal set of size N
  - How many different elements have appeared in the stream?
- Applications:
  - a Web site gathering statistics on how many unique users it has seen in each given month:
    - the universal set is the set of logins
    - a stream element is generated when someone logs in
  - Google: identify users by the IP address from which they send the query

The Count-Distinct Problem: solutions

- Obvious approach:
  - Keep in main memory a list of all the elements seen so far in the stream
  - Efficient search structure: quickly add new elements and check whether or not the element was already seen
  - Works if the number of distinct elements is not too great

The Count-Distinct Problem: solutions

- Real problem: **What if we do not have space to maintain the set of elements seen so far?**
  - Use more machines
  - Store most of the data in secondary memory and batch stream elements
  - Tests and updates on the blocks of data
- **Estimate the number of distinct elements**
  - Accept that the count may have a little error, but limit the probability that the error is large

Flajolet-Martin Approach

Idea:

- Hash the elements of the universal set to a bit-string
    - The bit-string should be large enough
- Pick many hash functions:
    - a hash function applied to the same element always produces the same result!
- More different elements in the stream $\Longrightarrow$ more different hash-values
- More different hash-values $\Longrightarrow$ it becomes more likely that one of these values will be "unusual":
    - **the value ends in many 0's**

Flajolet-Martin Approach: More formally

使用 Flajolet-Martin 算法，一个关键的观察是哈希函数的输出值可能会在其二进制表示中以多个0结束，而这个0的数量（即"尾部长度"或 r）可以用来估计流中不同元素的数量 m。

- Pick a hash function h that maps each of the N elements to at least $\log_2 N$ bits
- For each stream element a, let $r(a)$ be the number of trailing 0s in $h(a)$
  - $r(a)$ = position of first 1 counting from the right
    - E.g., say $h(a) = 12$, then 12 is 1100 in binary, so $r(a) = 2$
- Record R = the maximum $r(a)$ seen
- Estimated number of distinct elements $= 2^R$

Why it works: Rough Intuition

- $h(a)$ hashes a with equal probability to any of N values
- $h(a)$ is a sequence of $\log_2 N$ bits
- 2^{-r} fraction of all as have a tail of r zeros:
  - about 50% of as hash to $***0$
  - about 25% of as hash to $**00$
- We saw the longest tail of $r = 2$ (i.e., item hash ending $*100$) $\Longrightarrow$ probably, we have seen about 4 distinct items so far
- **It takes to hash about 2^r items before we see one with zero-suffix of length r**

Space Requirements

- It is not necessary to store the elements seen
- The only thing we need to keep in main memory: one integer per hash function
    - the largest tail length seen so far for that hash function and any stream element
- Only one stream: we could use millions of hash functions
- Limitation: the time to compute hash values for each stream element

References

- J. Leskovec, A. Rajaraman and J. D. Ullman *Mining of Massive Datasets* (2014), Chapter 4
- H.V. Jagadish, I.S. Mumick, and A. Silberschatz, "View maintenance issues for the chronicle data model", Proc. ACM Symp. on Principles of Database Systems, pp. 113 - 124, 1995.