Mining Data Streams

Sommerakademie St. Johann im Ahrntal — AG 4
Introduction to streams and stream processing

Florian Rappl

Department of Theoretical Physics
University of Regensburg

1. September 2014


PROBABILITY PROBABILITY HASHING MANAGEMENT SAMPLING
MARTIN ESTIMATES ELEMENTS DISTINCT BIGDATA SENSORS FILTERING
SENSORS ALGORITHM HASHING ESTIMATES QUERIES FREQUENCY MOMENTS FILTERING TRAFFIC
DISTINCT PROBABILITY STREAMING WEB
DISTINCT BIGDATA WEB
STREAMING SAMPLING BIGDATA BIGDATA HASHING HASHING
FLAJOLET TRAFFIC QUERIES MARTIN MARTIN SENSORS
DISTINCT SAMPLING ELEMENTS SAMPLING TRAFFIC ALGORITHM MARTIN
DISTINCT ALGORITHM PROBABILITY MANAGEMENT MOMENTS QUERIES
BLOOM MOMENTS BLOOM STREAMING WEB SENSORS FILTERING TRAFFIC
TRAFFIC QUERIES WEB IMAGES QUERIES
ALGORITHM MANAGEMENT FLAJOLET BLOOM
SENSORS SAMPLING FILTERING FILTERING PROBABILITY IMAGES BIGDATA MARTIN
BLOOM HASHING BLOOM SENSORS FILTERING WEB BLOOM WEB QUERIES BLOOM TRAFFIC QUERIES
TRAFFIC IMAGES FREQUENCY QUERIES FLAJOLET
HASHING MOMENTS BLOOM DISTINCT MARTIN FLAJOLET WEB
FREQUENCY FLAJOLET DISTINCT ESTIMATES SAMPLING STREAMING FILTERING PROBABILITY
ALGORITHM PROBABILITY MARTIN BLOOM SENSORS PROBABILITY
FREQUENCY FLAJOLET SENSORS HASHING FILTERING DISTINCT BLOOM ELEMENTS FILTERING
STREAMING STREAMING HASHING ESTIMATES DISTINCT IMAGES FILTERING ALGORITHM FLAJOLET QUERIES
HASHING FREQUENCY SENSORS ELEMENTS FREQUENCY
DISTINCT BIGDATA FLAJOLET ESTIMATES MANAGEMENT STREAMING

Outline

Section 1

Introduction

Motivation

Problem Monitor network links for quantities such as

- Elephant flows (e.g., traffic engineering),
- Number of distinct flows or average flow size,
- Flow size distribution,
- Per-flow traffic volume,
- Entropy of the traffic,
- Traffic matrix estimation or others

Challenge

Network monitoring at high speed is challenging:

- Packets arrive every 25 ns on a 40 Gbps link
- DRAM cannot be used due to speed limitations
- We need to use SRAM for per-packet processing
- The per-flow state is too large for the SRAM
- Traditional solution not accurate enough

The stream data model

- Input rate is controlled externally

The stream data model

- Input rate is controlled externally
- Input records (tuples) enter at a rapid rate

The stream data model

- Input rate is controlled externally
- Input records (tuples) enter at a rapid rate
- We have one or more input ports (possible streams)

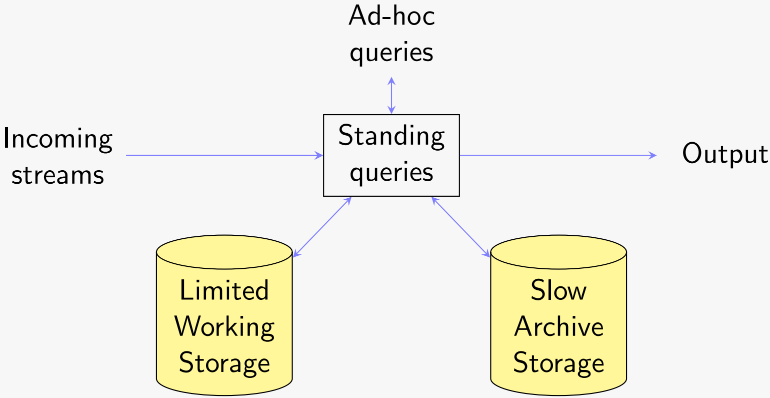
The stream data model

- Input rate is controlled externally
- Input records (tuples) enter at a rapid rate
- We have one or more input ports (possible streams)
- We are not able to store the entire stream

The stream data model

- Input rate is controlled externally
- Input records (tuples) enter at a rapid rate
- We have one or more input ports (possible streams)
- We are not able to store the entire stream
- The system may be required to scale (more streams, more frequent)

Illustration



Stream sources

- Sensors (GPS, IoT, ...)

Stream sources

- Sensors (GPS, IoT, ...)
- Network traffic (Web, TCP/IP, ...)

Stream sources

- Sensors (GPS, IoT, ...)
- Network traffic (Web, TCP/IP, ...)
- Data (Images, Videos, ...)

Stream sources

- Sensors (GPS, IoT, ...)
- Network traffic (Web, TCP/IP, ...)
- Data (Images, Videos, ...)
- Applications (Logging, Queries, ...)

Stream sources

- Sensors (GPS, IoT, ...)
- Network traffic (Web, TCP/IP, ...)
- Data (Images, Videos, ...)
- Applications (Logging, Queries, ...)
- Simulations (Results, Steps, ...)

Section 2

Sampling

Sliding window

- Usually the cheapest solution

Sliding window

- Usually the cheapest solution
- Only store the last N items (scales best)

Sliding window

- Usually the cheapest solution
- Only store the last N items (scales best)
- Alternatively keep the items of the last t seconds

Sliding window

- Usually the cheapest solution
- Only store the last N items (scales best)
- Alternatively keep the items of the last t seconds
- Only appropriate for certain queries

Sliding window

- Usually the cheapest solution
- Only store the last N items (scales best)
- Alternatively keep the items of the last t seconds
- Only appropriate for certain queries
- However, what if N is larger than the memory (volume)?

Sliding window

- Usually the cheapest solution
- Only store the last N items (scales best)
- Alternatively keep the items of the last t seconds
- Only appropriate for certain queries
- However, what if N is larger than the memory (volume)?
- Or if we have too many streams to handle (velocity)?

Random reservoir

- Estimation is key, exact choice may be irrelevant

Random reservoir

- Estimation is key, exact choice may be irrelevant
- Shrink incoming size to $1/n$-th of the original

Random reservoir

- Estimation is key, exact choice may be irrelevant
- Shrink incoming size to $1/n$-th of the original
- Generate a random number $r \in [1, n]$

Random reservoir

- Estimation is key, exact choice may be irrelevant
- Shrink incoming size to $1/n$-th of the original
- Generate a random number $r \in [1, n]$
- Consider the record if $r = 1$, otherwise discard

Random reservoir

- Estimation is key, exact choice may be irrelevant
- Shrink incoming size to $1/n$-th of the original
- Generate a random number $r \in [1, n]$
- Consider the record if $r = 1$, otherwise discard
- In practice not as simple, since the query might require more information

Example: Problem

- Incoming stream with tuples of (user, input, time)

Example: Problem

- Incoming stream with tuples of (user, input, time)
- Question: What fraction f of a typical user's input has been entered twice?

Example: Problem

- Incoming stream with tuples of (user, input, time)
- Question: What fraction f of a typical user's input has been entered twice?
- Assuming: s inputs occurred once, d twice, no input more often than twice, we find

$$f = \frac{d}{s + d} \tag{1}$$

Example: Problem

- Incoming stream with tuples of (user, input, time)
- Question: What fraction f of a typical user's input has been entered twice?
- Assuming: s inputs occurred once, d twice, no input more often than twice, we find

$$f = \frac{d}{s+d} \tag{1}$$

- Naively we would pick only the n-th record, e.g., $n = 10$

Example: Problem

- Incoming stream with tuples of (user, input, time)
- Question: What fraction f of a typical user's input has been entered twice?
- Assuming: s inputs occurred once, d twice, no input more often than twice, we find

$$f = \frac{d}{s+d} \tag{1}$$

- Naively we would pick only the n-th record, e.g., $n = 10$
- By considering each n-th record we get

$$\tilde{f} = n \frac{d}{ns + (2n-1)d} \leq f \tag{2}$$

Example: First correction

- Need more sophisticated way of picking a query

Example: First correction

- Need more sophisticated way of picking a query
- If user is known and tracked, then store the query

Example: First correction

- Need more sophisticated way of picking a query
- If user is known and tracked, then store the query
- If user is known and not tracked, discard

Example: First correction

- Need more sophisticated way of picking a query
- If user is known and tracked, then store the query
- If user is known and not tracked, discard
- Otherwise determine by random chance, *n*-th user

Example: First correction

- Need more sophisticated way of picking a query
- If user is known and tracked, then store the query
- If user is known and not tracked, discard
- Otherwise determine by random chance, n-th user
- Now we obtain

$$\tilde{f} = \frac{d}{n} \frac{n}{s + d} = f \tag{3}$$

Example: Solution

- Much better than taking a random value is ...

Example: Solution

- Much better than taking a random value is ...
- **Hashing** (in this case the user)

Example: Solution

- Much better than taking a random value is ...
- **Hashing** (in this case the user)
- Hash to *n* buckets

Example: Solution

- Much better than taking a random value is ...
- **Hashing** (in this case the user)
- Hash to n buckets
- Only consider the first bucket

Example: Solution

- Much better than taking a random value is ...
- **Hashing** (in this case the user)
- Hash to n buckets
- Only consider the first bucket
- No need to store user and status, only query

Section 3

Filtering

Conditional checking

- We only consider the current record if certain conditions are met

Conditional checking

- We only consider the current record if certain conditions are met
- The conditions may require more information than available

Conditional checking

- We only consider the current record if certain conditions are met
- The conditions may require more information than available
- It is sufficient to know if the conditions are not met

Conditional checking

- We only consider the current record if certain conditions are met
- The conditions may require more information than available
- It is sufficient to know if the conditions are not met
- The required information is too large for memory

Conditional checking

- We only consider the current record if certain conditions are met
- The conditions may require more information than available
- It is sufficient to know if the conditions are not met
- The required information is too large for memory
- The Bloom filter is the algorithm of choice

Bloom filter [Bloom (1970)]

- Basic idea: Use k hash functions h_i to reduce information

Bloom filter [Bloom (1970)]

- Basic idea: Use k hash functions h_i to reduce information
- The set of source elements S contains m entries

Bloom filter [Bloom (1970)]

- Basic idea: Use k hash functions h_i to reduce information
- The set of source elements S contains m entries
- We want to know if an incoming element $a \in S$

Bloom filter [Bloom (1970)]

- Basic idea: Use k hash functions h_i to reduce information
- The set of source elements S contains m entries
- We want to know if an incoming element $a \in S$
- Storing S or matching against every element is not possible

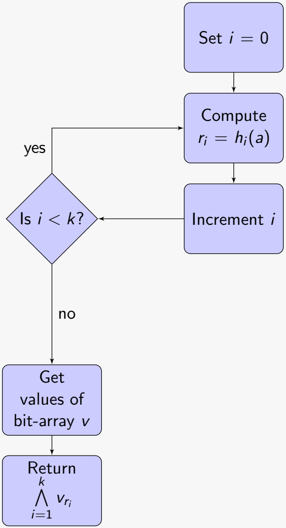
Bloom filter [Bloom (1970)]

- Basic idea: Use k hash functions h_i to reduce information
- The set of source elements S contains m entries
- We want to know if an incoming element $a \in S$
- Storing S or matching against every element is not possible
- Use a bit-array v with n entries as lookup table

Bloom filter [Bloom (1970)]

- Basic idea: Use k hash functions h_i to reduce information
- The set of source elements S contains m entries
- We want to know if an incoming element $a \in S$
- Storing S or matching against every element is not possible
- Use a bit-array v with n entries as lookup table
- Initialize the bit-array: Enable entries at $h_i(a)$ for $a \in S$, $i \in [1, k]$

Existence check for incoming a



Optimizing parameters

- Probability for a false positive is given by

$$P \approx \left(1 - \exp\left(\frac{-mk}{n}\right)\right)^k \tag{4}$$

Optimizing parameters

- Probability for a false positive is given by

$$P \approx \left(1 - \exp\left(\frac{-mk}{n}\right)\right)^k \tag{4}$$

- The optimal number of hash functions is

$$k = \left(\frac{n}{m}\right) \ln 2. \tag{5}$$

Optimizing parameters

- Probability for a false positive is given by

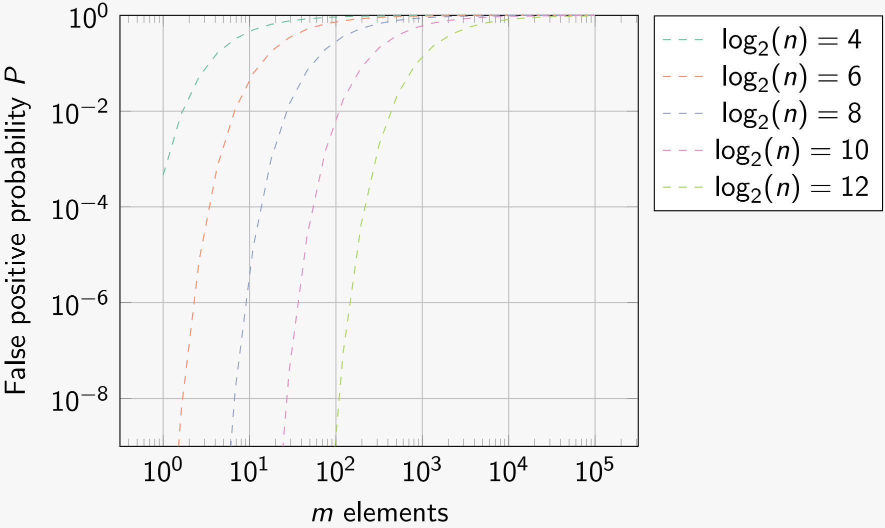
$$P \approx \left(1 - \exp\left(\frac{-mk}{n}\right)\right)^k \tag{4}$$

- The optimal number of hash functions is

$$k = \left(\frac{n}{m}\right) \ln 2. \tag{5}$$

- We can also try to optimize n, the length of the bit-array

False positive probability



Example

- Dictionary with w unique words

Example

- Dictionary with w unique words
- Select subset of size $m \leq w$

Example

- Dictionary with w unique words
- Select subset of size $m \leq w$
- Stream words of a text with $L \gg m$ words

Example

- Dictionary with w unique words
- Select subset of size $m \leq w$
- Stream words of a text with $L \gg m$ words
- Check for false positive rate P

Example

- Dictionary with w unique words
- Select subset of size $m \leq w$
- Stream words of a text with $L \gg m$ words
- Check for false positive rate P
- Use optimal number of hashing functions

Counting distinct elements

- The sum of the 0-th frequency moments is the number of distinct elements

Counting distinct elements

- The sum of the 0-th frequency moments is the number of distinct elements
- Idea similar to the Bloom filter

Counting distinct elements

- The sum of the 0-th frequency moments is the number of distinct elements
- Idea similar to the Bloom filter
- We use a hash function to reduce information

Counting distinct elements

- The sum of the 0-th frequency moments is the number of distinct elements
- Idea similar to the Bloom filter
- We use a hash function to reduce information
- The distribution of hash values yields information about the value distribution

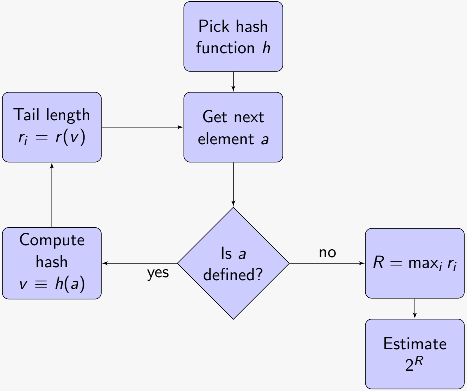
Counting distinct elements

- The sum of the 0-th frequency moments is the number of distinct elements
- Idea similar to the Bloom filter
- We use a hash function to reduce information
- The distribution of hash values yields information about the value distribution
- In the end we have a probablistic estimate

Counting distinct elements

- The sum of the 0-th frequency moments is the number of distinct elements
- Idea similar to the Bloom filter
- We use a hash function to reduce information
- The distribution of hash values yields information about the value distribution
- In the end we have a probablistic estimate
- The Flajolet-Martin algorithm describes this procedure

Flajolet-Martin algorithm [Flajolet, Martin (1985)]



Example

- Dictionary with w unique words

Example

- Dictionary with w unique words
- Select subset of size $m \leq w$

Example

- Dictionary with w unique words
- Select subset of size $m \leq w$
- Choose elements of this subset randomly with redraw

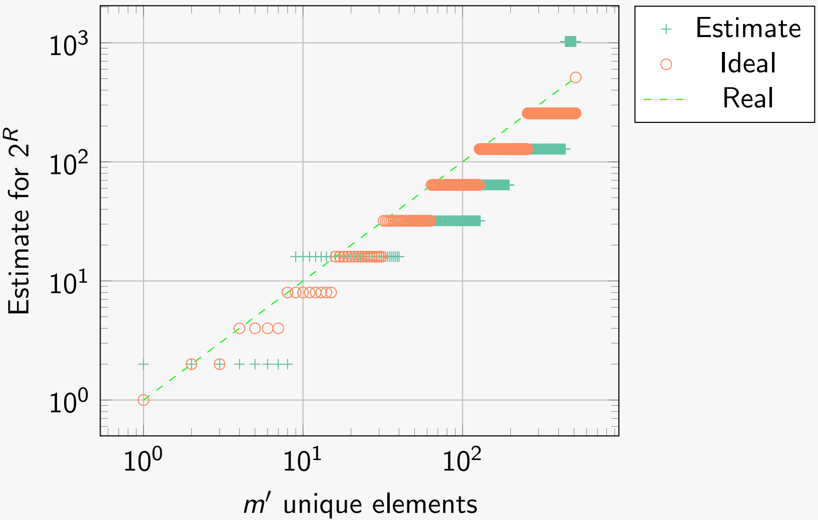
Example

- Dictionary with w unique words
- Select subset of size $m \leq w$
- Choose elements of this subset randomly with redraw
- Count the number of distinct elements

Example

- Dictionary with w unique words
- Select subset of size $m \leq w$
- Choose elements of this subset randomly with redraw
- Count the number of distinct elements
- Use ordinary string hashing function

Sample estimates



Section 4

Conclusion

Conclusion

- Evaluate queries in detail

Conclusion

- Evaluate queries in detail
- Keep sliding window for fast estimations if possible

Conclusion

- Evaluate queries in detail
- Keep sliding window for fast estimations if possible
- Use random reservoir only if probabilities are obvious

Conclusion

- Evaluate queries in detail
- Keep sliding window for fast estimations if possible
- Use random reservoir only if probabilities are obvious
- Embrace hash functions for randomness

Conclusion

- Evaluate queries in detail
- Keep sliding window for fast estimations if possible
- Use random reservoir only if probabilities are obvious
- Embrace hash functions for randomness
- Determine if approximations are sufficient

Conclusion

- Evaluate queries in detail
- Keep sliding window for fast estimations if possible
- Use random reservoir only if probabilities are obvious
- Embrace hash functions for randomness
- Determine if approximations are sufficient
- Always think about scaling

Thank you!