BIG DATA

Streaming Algorithms

K V Subramaniam
Computer Science and Engineering

BIG DATA

Overview: Streaming Algorithms

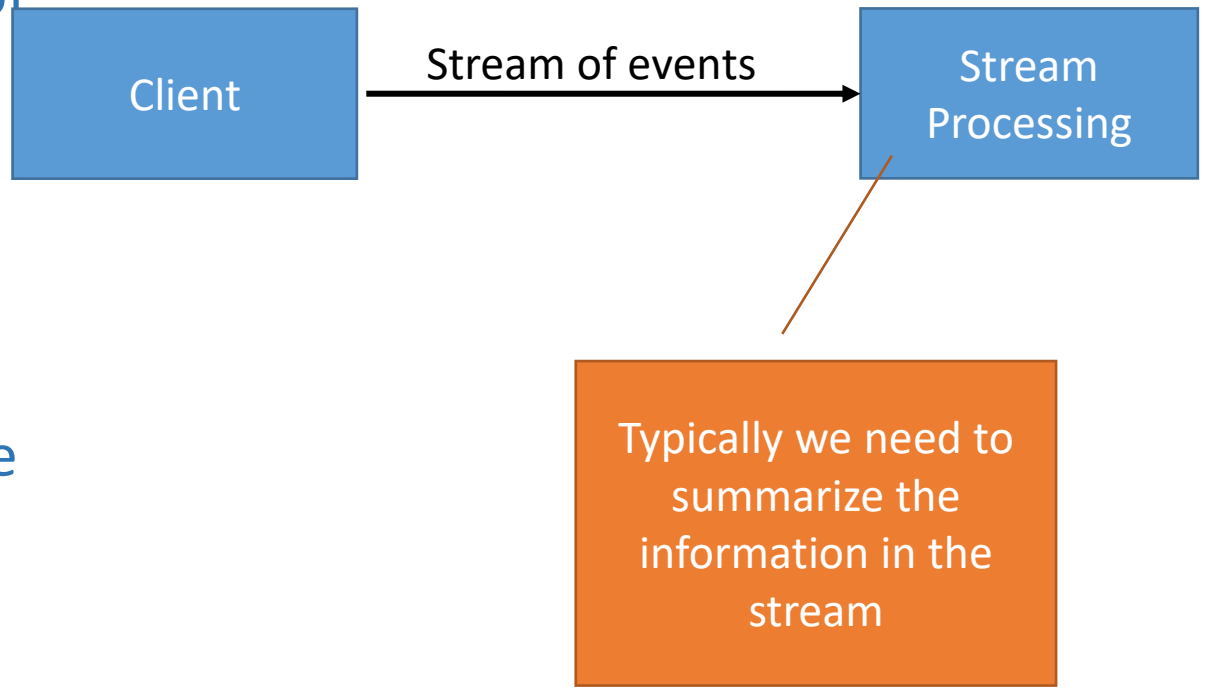
- Why study streaming algorithms?
- Sampling Algorithms

Streaming Algorithms overview

BIG DATA

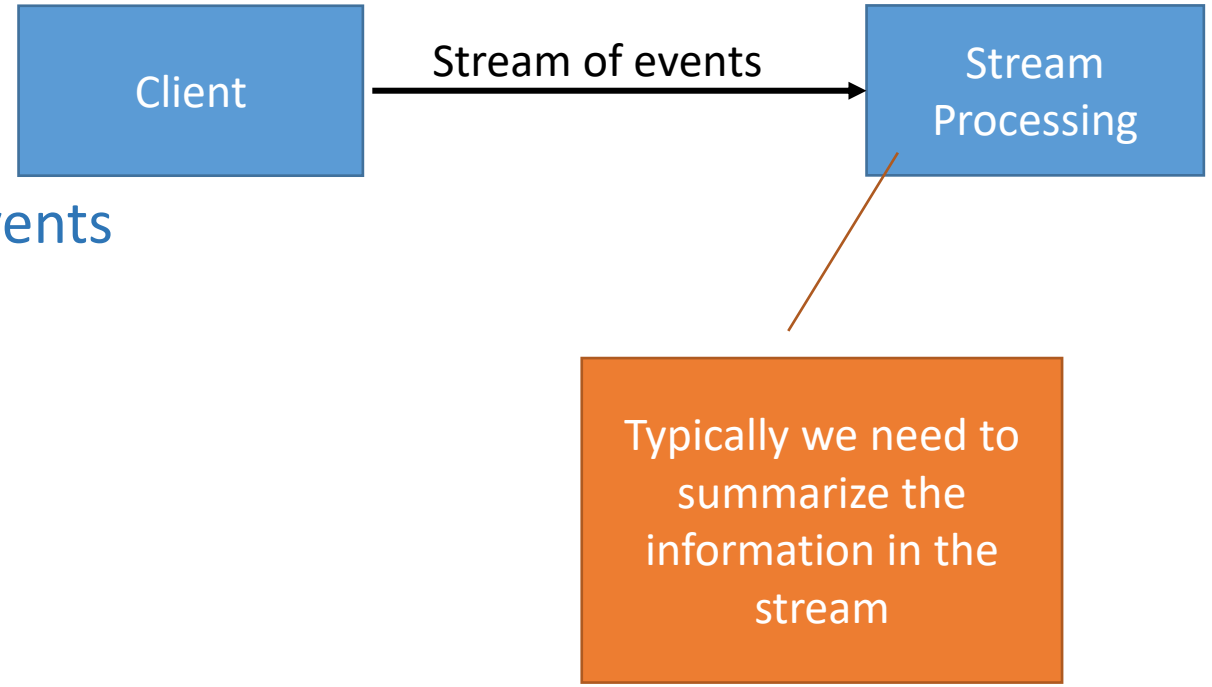
The need for processing events

- Stream processing requires processing of events in a never ending stream
- Need to look at a summaries
- For example
  - How many unique elements have we seen in the stream?
- In a relational data this is equivalent to counting the keys, but how to do it on a never ending stream



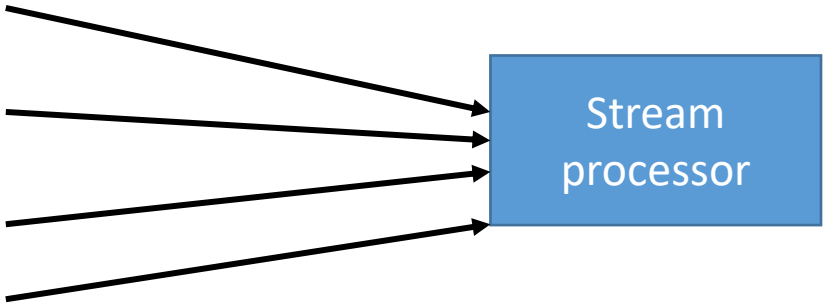
The need for processing events

- One approach
  - Breakup stream into a window of events
  - Window size *n*
  - Allows us to perform relational operations
  - Similar to Apache Spark model



BIG DATA

Issues in stream processing

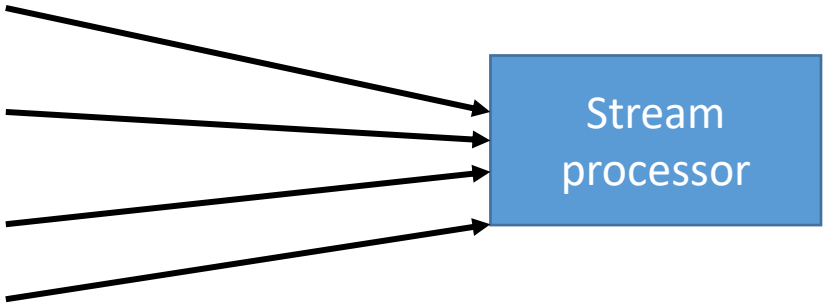


- Velocity of the stream
  - The rate at which data is being sent
  - Sometimes requires instantaneous decision.
  - Different streams may have different rates
  - For ex: mission critical systems

- #streams
  - Processing multiple streams each requiring a small amount of memory may stress memory system

BIG DATA

Stream processing algorithms



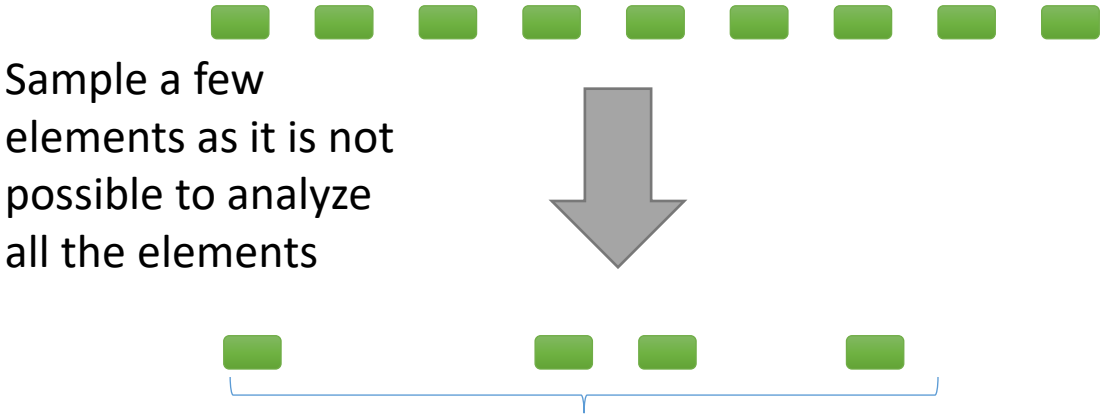
- Must process data in memory
  - Not off disk (too slow)
- More efficient to get *approximate* solution rather than an exact one.
- Often use *hashing* techniques to introduce randomness

Sampling Algorithms

BIG DATA

Extracting reliable samples

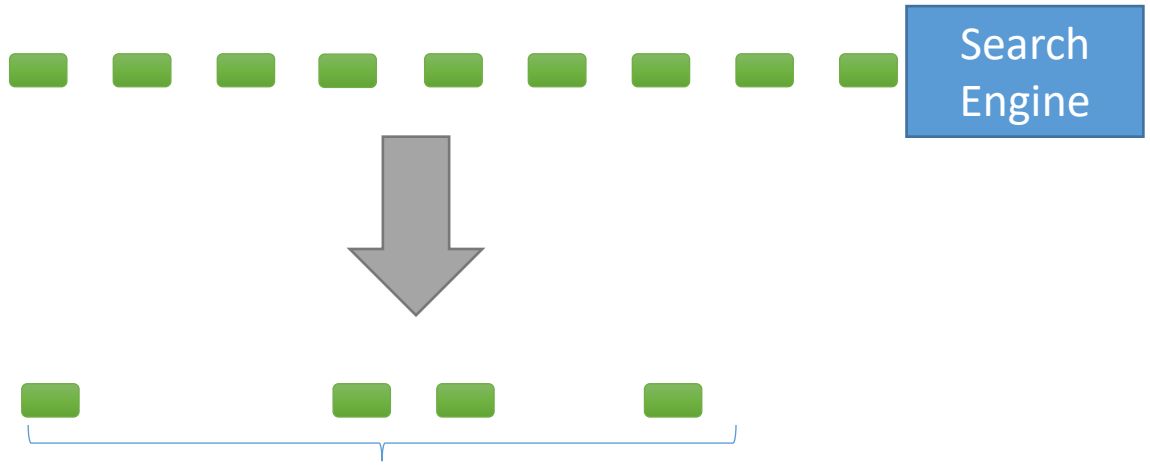
- Given a long stream of data
- How to create a representative sample?



BIG DATA

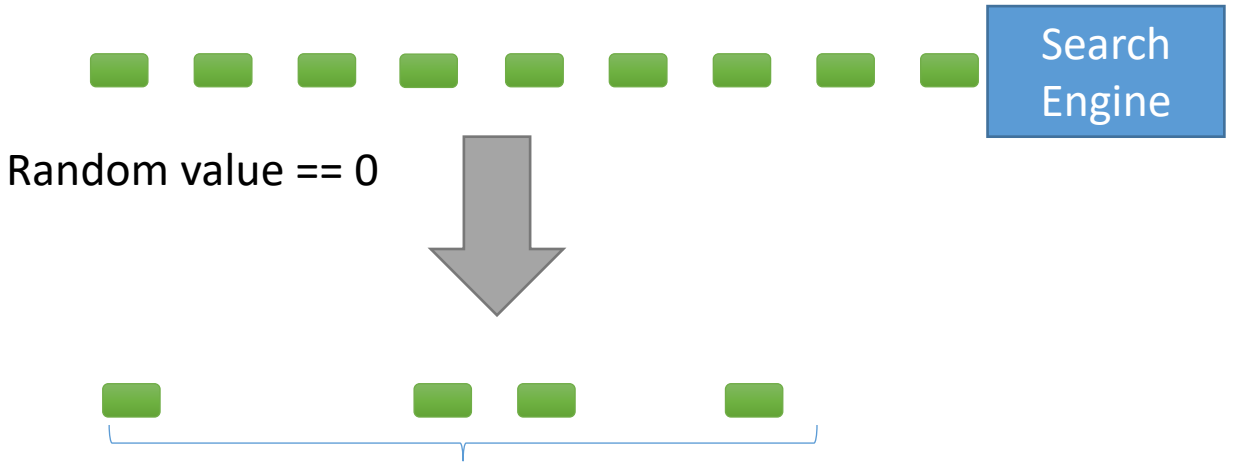
Motivating Example

- Each element of the query represents a query
- How to create a representative sample?
- We want to answer the question?
  - "What fraction of the typical user's queries were repeated over the past month?"
- What's obvious algorithm?



Obvious Algorithm

- For every stream tuple seen..
- Generate a random integer between 0..9
- If value is 0
  - Then store(use) the tuple
- Otherwise – discard.



Flaw in obvious algorithm

Flaw in obvious algorithm

- Suppose user has issued a particular search query s twice
- There's a probability 1/100 (not 1/10) that both queries will show up in our sample
- There's only a probability 1/100 that we will know query s was repeated

So our sampling will be wrong

Query number *m* is *s*

p(m sampled) = 1/10

Query number *n* is also *s*

p(n sampled) = 1/10

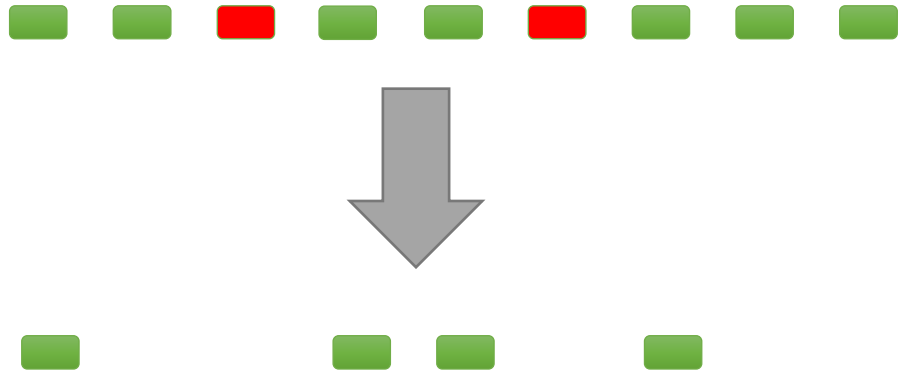
p(m and n sampled) = 1/100

Flaw in obvious algorithm

Flaw in obvious algorithm

- Suppose user has issued a particular search query s twice
- There's a probability 1/100 (not 1/10) that both queries will show up in our sample
- There's only a probability 1/100 that we will know query s was repeated

So our sampling will be wrong



Sampling Algorithms - refinement

Refined algorithm

- Sample 1/10th of the users
  - Not the transactions
- Details
  - When a query arrives
  - Look up user to see if they are in sample.
  - If so, add query to sample
  - If first time we have seen the user, generate a random number 0..9
    - Add user to sample if number is 0

Optimization to algorithm

- Checking if we have seen the user
- Requires a search through a data structure
- Not really required
- Just hash(username) → 0..9
  - If hash is 0, select the userid

Generalization of the algorithm

- We used “user” here to select
- How to extend this algorithm generically?
- Identify the *key components* of the query.
  - In previous example tuple is *<user, query, time>*
  - But *key component* is only user.
- Hash key components in the range *(0..b)*
- To get sample size (a/b)
  - Select query if *hash(key components) < a*

Exercise

- Suppose we want a sample dataset to debug a program that profiles transactions by user and country
- How would I generate a 1/20 sample?

Exercise - Solution

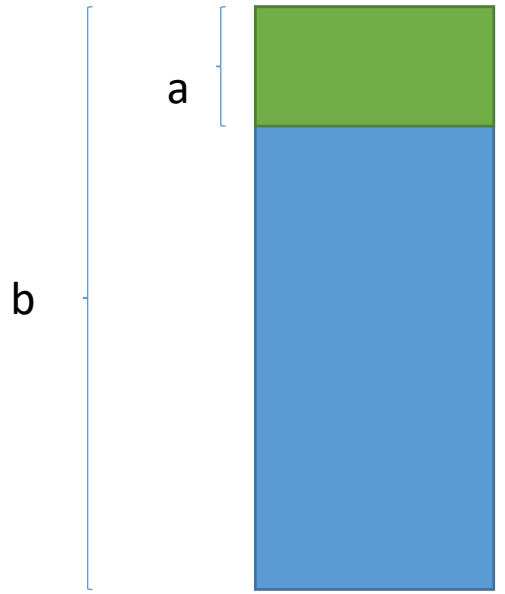
- There isn't a unique solution
- Suppose I want a sample dataset to debug a program that profiles transactions by user and country
- How would I generate a 1/20 sample? Hash userid and country in the range 0..19; select if the hash is 0
- The above method will give me 1/20 of all user-country pairs

Varying Sample size

BIG DATA

Varying the sample size

- We are selecting *a/b* samples from data arriving from outside.
- As more data is added to the system, the number of keys in the system also increases
  - Additional storage space required
- If there is a limit on #keys that can be stored, then how to handle it, so representatives is not lost
- Reduce *a* to *a-1.*



THANK YOU

K V Subramaniam, Usha Devi

Dept. of Computer Science and Engineering

subramaniamkv@pes.edu

ushadevibg@pes.edu