Data Streams: *Introduction*

Mining Massive Datasets

Materials provided by Prof. Carlos Castillo — https://chato.cl/teach

Instructor: Dr. Teodora Sandra Buda — https://tbuda.github.io/

Sources

- Mining of Massive Datasets (2014) by Leskovec et al. (chapter 4)
  - Slides part 1, part 2
- Tutorial: Mining Massive Data Streams (2019) by Michael Hahsler

What is a data stream?

- A **potentially infinite sequence** of data points
  - Each data point can be a tuple or vector
- Examples:
  - web click-stream data → who clicks on what
  - computer network monitoring data
  - telecommunication connection data
  - readings from sensor nets
  - stock quotes

Do not confuse with "streaming," which typically means watching a video while it is being downloaded.

Example: web server log

```
tecmint@TecMint ~ $ tailf /var/log/apache2/access.log
127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET / HTTP/1.1" 200 729 "-" "Mozill
127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET /icons/blank.gif HTTP/1.1" 200
fox/56.0"
127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET /icons/folder.gif HTTP/1.1" 200
efox/56.0"
127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET /icons/text.gif HTTP/1.1" 200 5
ox/56.0"
127.0.0.1 - - [31/Oct/2017:11:11:38 +0530] "GET /favicon.ico HTTP/1.1" 404 500
127.0.0.1 - - [31/Oct/2017:11:12:05 +0530] "GET /tecmint/ HTTP/1.1" 200 787 "ht
0"
127.0.0.1 - - [31/Oct/2017:11:12:05 +0530] "GET /icons/back.gif HTTP/1.1" 200 4
01 Firefox/56.0"
127.0.0.1 - - [31/Oct/2017:11:13:58 +0530] "GET /tecmint/Videos/ HTTP/1.1" 200
101 Firefox/56.0"
127.0.0.1 - - [31/Oct/2017:11:13:58 +0530] "GET /icons/compressed.gif HTTP/1.1"
) Gecko/20100101 Firefox/56.0"
127.0.0.1 - - [31/Oct/2017:11:13:58 +0530] "GET /icons/movie.gif HTTP/1.1" 200
o/20100101 Firefox/56.0"
```

Key properties of data streams

- **Unbounded size**
  - Data cannot be persisted on disk
  - Only summaries can be stored
- **Transient**
  - Single pass over the data
  - Sometimes real-time processing is needed
- **Dynamic**
  - May require incremental updates
  - May require to forget old data
  - Concepts “drift”
- **Temporal order** is often important

Applications

- **Mining query streams**
  - A search engine wants to know what queries are more frequent today than yesterday
- **Mining click streams**
  - A newspaper wants to know when one of its pages starts getting an unusual number of hits per hour
- **Mining social network news feeds**
  - A social media platform wants to show trending topics

Applications (cont.)

- **Sensor Networks**
  - Many sensors feeding into a central controller
- **Telephone call records**
  - Data feeds into customer bills as well as settlements between telephone companies
- **IP packets monitored at a switch**
  - Gather information for optimal routing
  - Detect denial-of-service attacks

Why not simply use a relational DB?

Relational DBMS	DSMS (Stream)
persistent relations	transient streams
only current state is important	history matters
not real-time	real-time
low update rate	stream!
one time queries	continuous queries

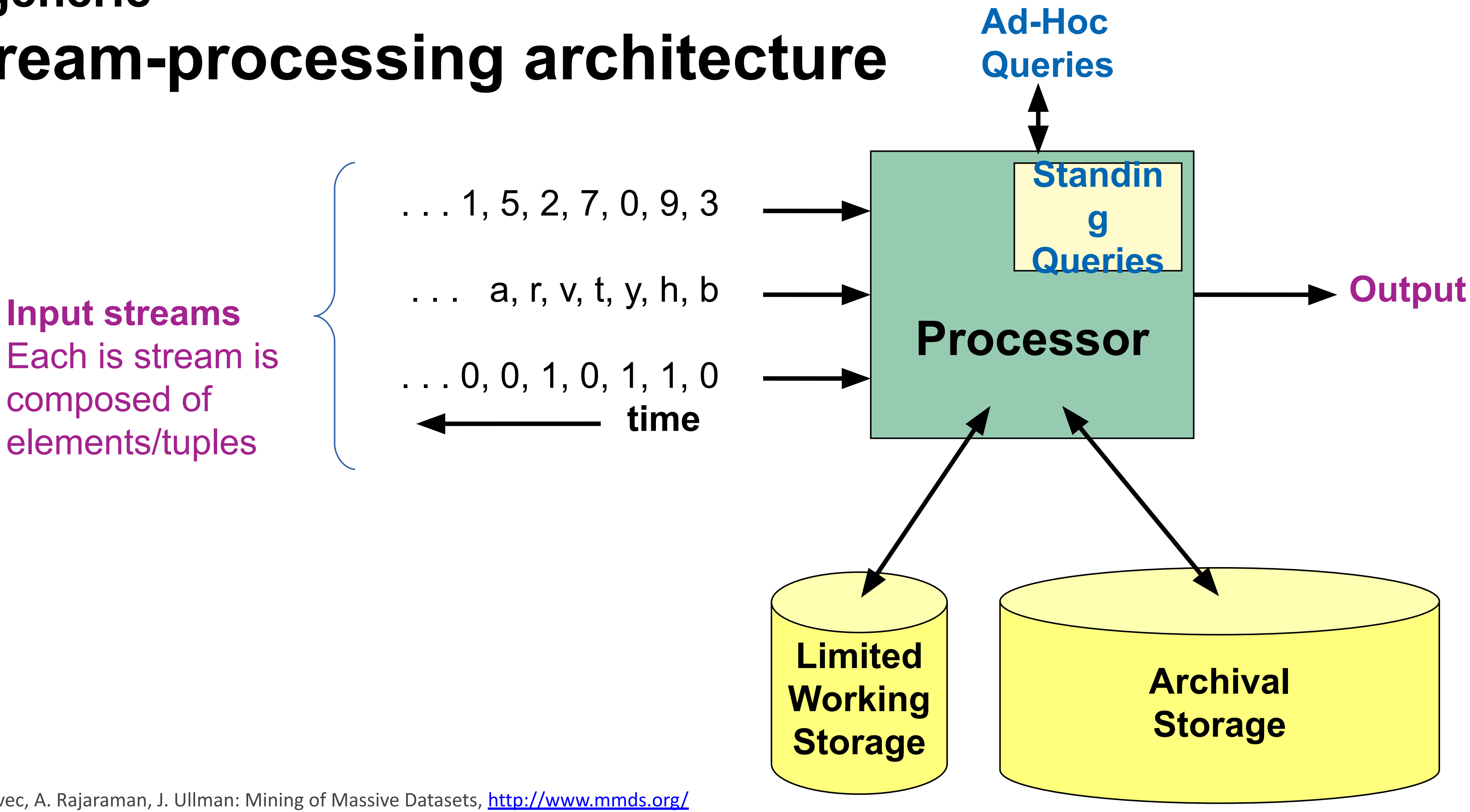
Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom (2002). Models and issues in data stream systems. In PODS ’02, pages 1–16, ACM Press.

Why do we need new algorithms?

	Traditional	Stream
passes	multiple	single
processing time	unlimited	restricted
memory	disk	main memory
results	typically accurate	approximate
distributed	typically not	often

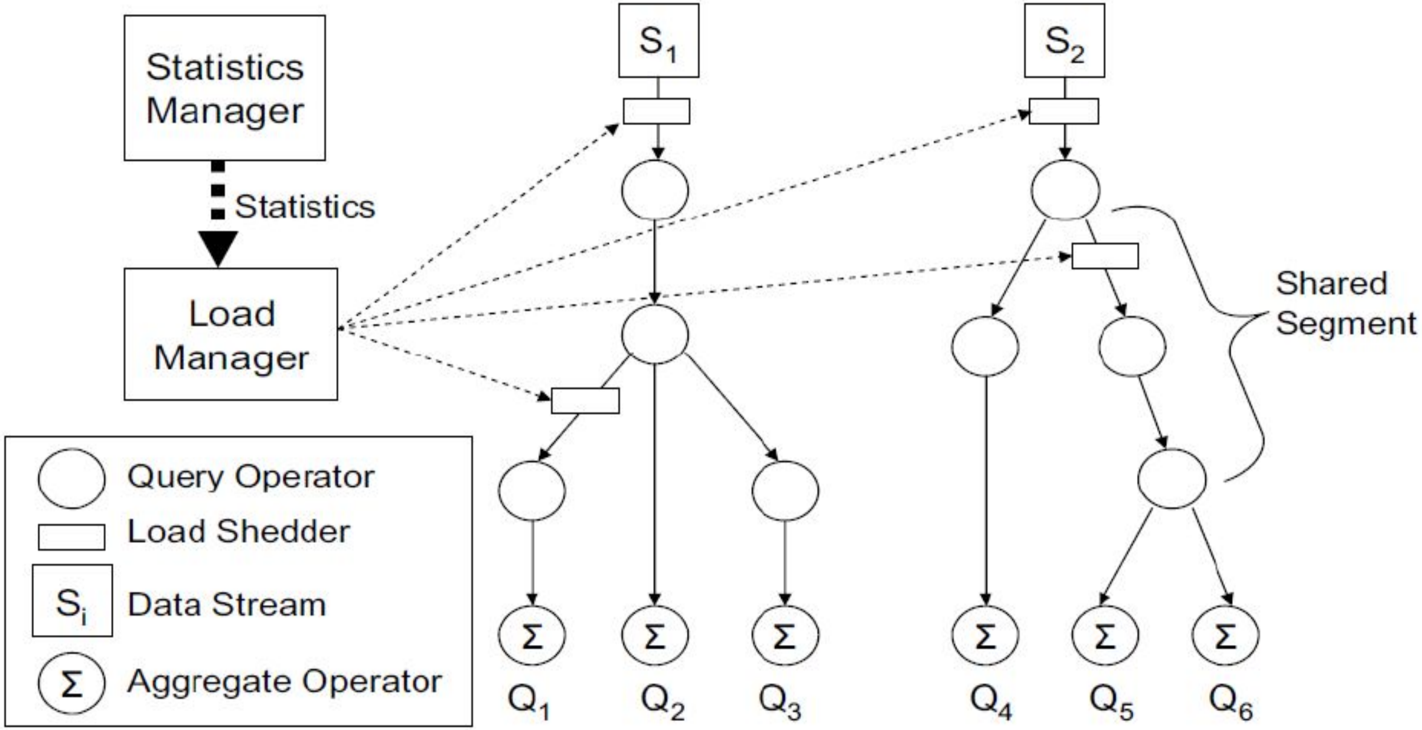
Source: Joao Gama, Data Stream Mining Tutorial, ECML/PKDD, 2007

A generic
stream-processing architecture



Load shedding

Too much data? Ignore some of it



Brian Babcock, Mayur Datar, and Rajeev Motwani. Load shedding techniques for data stream systems. In Proc. MPDS, 2003.

Sampling a fixed proportion

Sampling a fixed proportion

- Example stream: `<user, query, timestamp>` from a search engine query log
- Suppose we have space to store 1/r of the stream
  - E.g.: 1/10th, 1/100th, 1/1000th,
- Naïve solution:
  - Generate uniform random number in 0...(r-1)
    - `numpy.random.uniform(0,r)`
  - If the number is 0, keep the item

What can we do with this sample?

- Approximate most frequent query
  - Pick the most frequent in the sample
- Approximate frequency of a query
  - Multiply observed frequency by r
- Do people ask query q?
  - Approximate answer (with some prob. of error)

Exercise: sampling at a fixed rate

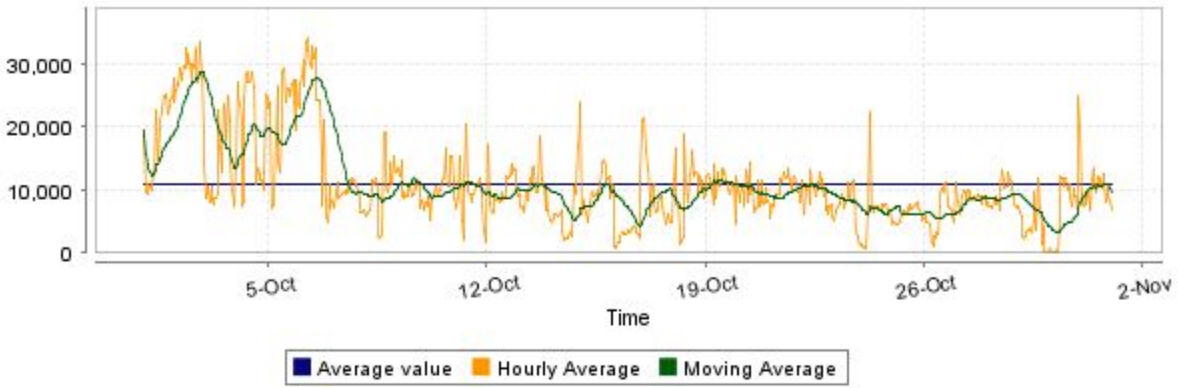
- We want to tell if we have seen item *q*
- Suppose we have seen *n* items
- Suppose we have sampled a fraction *1/r*
- Suppose item *q* appears with probability *p(q)*
- Can we observe a ...
  - False Positive? *(Item q <u>was not</u> in the stream but we said it <u>was</u>)*
  - False Negative? *(Item q <u>was</u> in the stream but we said it <u>was not</u>)*

Answer

- We want to tell if we have seen item *q*
- Suppose we have seen *n* items
- Suppose we have sampled a fraction *1/r*
- Suppose item *q* appears with probability *p(q)*
- Can we observe a ...
  - False Positive? NO. We cannot observe an item that is not.
  - False Negative? YES. We can miss an item that is there.

What can we do with this…? (cont.)

- Approximate num. queries per minute



- Peak frequency
  - Multiply observed peak by r

https://steemit.com/steemsql/@arcange/steemsql-update-14-infrastructure-upgraded-for-better-performances

There are questions we cannot answer with this sampling method

- **What fraction of queries by an average search engine user are duplicates?**
  - Suppose each user issues x queries once and d queries twice (total of x+2d queries)
  - **Correct answer: d/(x+d)**
- Proposed solution: We keep 1/10th of the queries (r=10)
  - Sample will contain x/10 of the singleton queries at least once
  - Sample will contain 2d/10 of the duplicate queries at least once
  - Sample will contain d/100 pairs of duplicates
    - d/100 = 1/10 · 1/10 · d
  - Of the d duplicates, 18d/100 will be seen once*
    - 18d/100 = ((1/10 · 9/10)+(9/10 · 1/10)) · d
- So the sample-based answer is

$$\frac{\frac{d}{100}}{\frac{x}{10}+\frac{18d}{100}+\frac{d}{100}} = \frac{d}{10x+19d}$$

* Copy A is in the selected part, copy B in the unselected part, or viceversa

There are questions we cannot answer with this sampling method (cont.)

- **What fraction of queries by an average search engine user are duplicates?**
  - Suppose each user issues x queries once and d queries twice (total of x+2d queries)
  - **Correct answer: d/(x+d)**
- Proposed solution: We keep 1/10th of the queries (r=10)
  - Sample will contain x/10 of the singleton queries at least once
  - Sample will contain 2d/10 of the duplicate queries at least once
  - Sample will contain d/100 pairs of duplicates
    - $d/100 = 1/10 \cdot 1/10 \cdot d$
  - Of the d duplicates, 18d/100 will be seen once*
    - $18d/100 = ((1/10 \cdot 9/10)+(9/10 \cdot 1/10)) \cdot d$
- So the sample-based answer is

WRONG!

$$\frac{\overbrace{\frac{d}{100}}^{\text{Observed duplicates}}}{\underbrace{\frac{x}{10}+\frac{18d}{100}}_{\text{Observed singletons}}+\underbrace{\frac{d}{100}}_{\text{Observed duplicates}}} = \frac{d}{10x+19d}$$

Sampling tuples at random by one attribute

Suppose we need to sample 1/r of users and all of their actions

How do we do this?

<user1, action, timestamp>
<user2, action, timestamp>
<user2, action, timestamp>
<user3, action, timestamp>
<user1, action, timestamp>
<user3, action, timestamp>
<user2, action, timestamp>
<user1, action, timestamp>
<user2, action, timestamp>
...

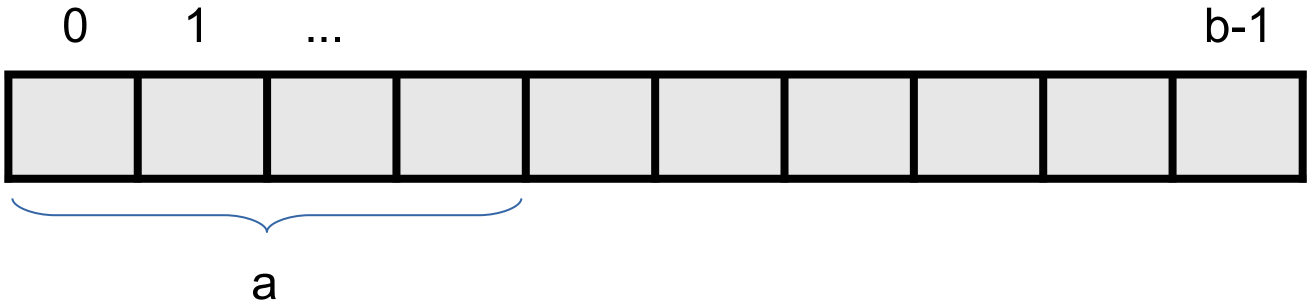
How do we solve it?

- We need to sample 1/r of users and all of their actions
- How do we do this?
  - **Hashing!**
  - Given <user, action, timestamp>
  - Compute h(user) → 0, 1, …, (r-1)
  - Keep tuple if hash value is 0

<user1, action, timestamp>
<user2, action, timestamp>
<user2, action, timestamp>
<user3, action, timestamp>
<user1, action, timestamp>
<user3, action, timestamp>
<user2, action, timestamp>
<user1, action, timestamp>
<user2, action, timestamp>
...

In general ...

- To sample a fraction a/b of a stream by key
- Compute h(key) → 0, 1, …, (b-1)
- Keep if h(key) < a



Summary

Things to remember

- What is a data stream
- How to sample a fixed percentage of values grouped by a key, using hashing

Exercises for TT22-T26

- Mining of Massive Datasets (2014) by Leskovec et al.
  - Exercises 4.2.5
  - Exercises 4.3.4
  - Exercises 4.4.5
  - Exercises 4.5.6