# Data Streaming Algorithms

Barna Saha

# Motivation

- Data arrives in a stream or streams
- If not processed immediately or stored, then data is lost for ever.
- Data arrives so rapidly that it is not feasible to store it all in active storage.
- We need new algorithmic paradigm to handle data streams.

# Example of Data Streams

**Sensor Data.**

- A temperature sensor in the ocean sending reading every one hour–Not an interesting stream since the data rate is low. It will not stress modern technology, and the entire stream can be kept in main memory.

# Example of Data Streams

**Sensor Data.**

- A temperature sensor in the ocean sending reading every one hour–Not an interesting stream since the data rate is low. It will not stress modern technology, and the entire stream can be kept in main memory.
- Suppose the sensor senses surface height information which changes rapidly. Now the sensor is sending data back every tenth of a second. If it sends a 4-byte real number each time, then it produces $4 * 10 * 3600 * 24 = 3456000 bytes = 3.5 Megabyte/$per day.–Still ok.

# Example of Data Streams

**Sensor Data.**

- A temperature sensor in the ocean sending reading every one hour–Not an interesting stream since the data rate is low. It will not stress modern technology, and the entire stream can be kept in main memory.
- Suppose the sensor senses surface height information which changes rapidly. Now the sensor is sending data back every tenth of a second. If it sends a 4-byte real number each time, then it produces $4 * 10 * 3600 * 24 = 3456000 bytes = 3.5 Megabyte/$per day.–Still ok.
- We may need to employ a million sensors to learn about ocean behavior.—3.5 terabytes of data per day, million of data arriving every tenth of a second.

# Example of Data Streams

**Image Data.**

- Satellites often send down to earth streams consisting of many terabytes of images per day.

# Example of Data Streams

**Image Data.**

- Satellites often send down to earth streams consisting of many terabytes of images per day.
- Surveilance cameras may produce images at every second. London is said to have six millions of such cameras.

# Example of Data Streams

**Internet and Web Traffic.**

- A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs: denial or service attacks.

# Example of Data Streams

**Internet and Web Traffic.**

- A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs: denial or service attacks.
- Google receives several hundred million search queries per day.

# Example of Data Streams

**Internet and Web Traffic.**

- A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs: denial or service attacks.
- Google receives several hundred million search queries per day.
- Yahoo! accepts billions of clicks per day on its various sites.

# Example of Data Streams

**Internet and Web Traffic.**

- A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs: denial or service attacks.
- Google receives several hundred million search queries per day.
- Yahoo! accepts billions of clicks per day on its various sites.
- Many interesting things can be learnt from these streams. An increase in queries like "sore throat" may help to track the spread of viruses. A sudden increase in the click rate for a link could indicate some news connected to that page etc.

# Which industries are deploying stream processors?

- Smart Cities - real-time traffic analytics, congestion prediction and travel time apps.
- Oil & Gas - real-time analytics and automated actions to avert potential equipment failures.
- Security intelligence for fraud detection and cybersecurity alerts. For example, detecting Smart Grid consumption issues, and SIM card misuse.
- Industrial automation, offering real-time analytics and predictive actions for patterns of manufacturing plant issues and quality problems.
- For Telecoms, real-time call rating, fraud detection and QoS monitoring from CDR (call detail record) and network performance data.
- Cloud infrastructure and web clickstream analysis for IT Operations.

# Few Stream Processing Systems

- SQLstream `http://www.sqlstream.com/blaze/`: use standards-compliant SQL for querying live data streams
- Spark Streaming: to build streaming applications in Apache Spark. Apache Spark is a general framework for large-scale data processing that supports concepts such as MapReduce, stream processing, graph processing or machine learning.
- IBM InfoSphere Streams: IBM's flagship product for stream processing.
- Apache Storm: an open source framework that provides massively scalable event collection.

# Developing Streaming Algorithms

- The main hurdle is the space.
- Often it is much more efficient to get an approximate answer than an exact answer.
- Often the algorithm uses randomization like hashing and sampling.

# Heavy Hitter Problem

- Problem. Given an array $A$ of length $m$, and a parameter $k$, find those values that occur at least $\frac{m}{k}$ times.
- Applications:
  1. **Computing popular products.** $A$ could be all of the page views of products on *amazon.com* yesterday. The heavy hitters correspond to frequently viewed items.
  2. **Computing frequent search queries.** For example, $A$ could be all of the searches on Google yesterday. The heavy hitters are then searches made most often.
  3. **Identifying heavy TCP flows.** Here, $A$ is a list of data packets passing through a network switch, each annotated with a source-destination pair of IP addresses. The heavy hitters are then the flows that are sending the most traffic. This is useful for, among other things, to identify denial-of-service attacks.
  4. **Identifying volatile stocks.** Here, $A$ is a list of stock trades.

# Finding Majority

- **Input.** An array $A$ of length $m$ with the promise that it has a majority element–a value that is repeated strictly more than $\frac{m}{2}$ times.
- **Problem.** Find the Majority element in linear time.

# Finding Majority

- Input. An array $A$ of length $m$ with the promise that it has a majority element–a value that is repeated strictly more than $\frac{m}{2}$ times.
- Problem. Find the Majority element in linear time.
- Compute median of $A$.

# Finding Majority

- Input. An array $A$ of length $m$ with the promise that it has a majority element–a value that is repeated strictly more than $\frac{m}{2}$ times.
- Problem. Find the Majority element in linear time in a single left to right pass in "constant" space.

# Finding Majority

- **Problem.** Find the Majority element in linear time in a single left to right pass in "constant" space.
- **Algorithm.**
  1. Set $count = 1$, $current = A(1)$.
  2. For $i = 2, 3, \ldots$
     2.1 If $count == 0$, set $current = A(i)$, $count = 1$,
     2.2 If $A(i) == current$, set $count = count + 1$
     2.3 Else set $count = count - 1$
  3. Return current

# Finding Majority

- Problem. Find the Majority element in linear time in a single left to right pass in "constant" space.
- Algorithm.
  1. Set $count = 1$, $current = A(1)$.
  2. For $i = 2, 3, \ldots$
     2.1 If $count == 0$, set $current = A(i)$, $count = 1$,
     2.2 If $A(i) == current$, set $count = count + 1$
     2.3 Else set $count = count - 1$
  3. Return current
- Exercise. Given there exists a majority element, show that the above algorithm correctly returns the majority.

# Heavy Hitter Problem

- Can we solve Heavy Hitter Problem in small space? Ideally in $\tilde{O}(k)$ space.

# Heavy Hitter Problem

- Can we solve Heavy Hitter Problem in small space? Ideally in $\tilde{O}(k)$ space.
- There is no algorithm that solves the Heavy Hitters problems in one pass while using a sublinear amount of auxiliary space.

# $\epsilon$-Approximate Heavy Hitter Problem

- Input is an array $A$ of length $m$ with two parameters $\epsilon$ and $k$.
- Output
  1. Every value that occurs at least $\frac{m}{k}$ times in $A$ is in the list.
  2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in $A$

# $\epsilon$-Approximate Heavy Hitter Problem

- Input is an array $A$ of length $m$ with two parameters $\epsilon$ and $k$.
- Output
  1. Every value that occurs at least $\frac{m}{k}$ times in $A$ is in the list.
  2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in $A$
- Why not set $\epsilon = 0$?

# $\epsilon$-Approximate Heavy Hitter Problem

- Input is an array $A$ of length $m$ with two parameters $\epsilon$ and $k$.
- Output
  1. Every value that occurs at least $\frac{m}{k}$ times in $A$ is in the list.
  2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in $A$
- Why not set $\epsilon = 0$?
- Space usage grows proportionately with $\frac{1}{\epsilon}$.

# $\epsilon$-Approximate Heavy Hitter Problem

- Input is an array $A$ of length $m$ with two parameters $\epsilon$ and $k$.
- Output
    1. Every value that occurs at least $\frac{m}{k}$ times in $A$ is in the list.
    2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in $A$
- Why not set $\epsilon = 0$?
- Space usage grows proportionately with $\frac{1}{\epsilon}$.
- If we take $\epsilon = \frac{1}{2k}$, space usage is $\tilde{O}(k)$, all elements with frequency $\frac{m}{k}$ is in the list and the elements in the list have frequency at least $\frac{m}{2k}$.

# Estimating Frequency of Elements

- Input Stream of $m$ elements from a universe $[1, n]$: $A(1), A(2), ..., A(m)$.
- Frequency of an element $i \in [1, n]$ in the stream is $f_i = |t \mid A(t) = i|$.
- Problem
  - For $i \in [n]$, estimate $f_i$ (Point Query)
  - For $\phi \in [0, 1]$, find all $i$ with $f_i \geq \phi m$. (Heavy Hitter)

# Count-Min Sketch

- Select an $\epsilon > 0$ and $\delta > 0$: $\epsilon$ denotes the error-parameter, and $\delta$ denotes our confidence.
- Select $d = \ln \frac{1}{\delta}$ hash functions $h_1, h_2, ..., h_d$ independently and randomly from a pair-wise independent hash family. Each $h_i : \{1, 2, ..., n\} \rightarrow \{1, 2, ..., w\}$ where $w = \frac{e}{\epsilon}$.
- Initialize a table $T$ of dimension $d \times w$ all with 0.
- Update: At time $t$, when $A(t)$ arrives, do the following.
  - $T(1, h_1(A(t))) = T(1, h_1(A(t))) + 1$
  - $T(2, h_2(A(t))) = T(2, h_2(A(t))) + 1$
  - .
  - .
  - $T(d, h_d(A(t))) = T(d, h_d(A(t))) + 1$

`http://research.neustar.biz/tag/count-min-sketch/`

## Count-Min Sketch:Point Query

- Problem For $i \in [n]$, estimate $f_i$
- Output An estimate $\hat{f}_i$ such that $f_i \le \hat{f}_i \le f_i + \epsilon ||\mathbf{f}||_1$
- Algorithm Construct Count-Min sketch. Return

$$\min_{l=1}^{d} T(l, h_l(i))$$

.

# Count-Min Sketch:Point Query

- Algorithm Construct Count-Min sketch. Return

$$\min_{l=1}^{d} T(l, h_l(i))$$

.

- Each $T(l, h_l(i)) \geq f_i$. Hence $\min_{l=1}^{d} T(l, h_l(i)) \geq f_i$.
- Define an indicator random variable $X_j^l$, $j = 1, 2, ..n$ and $l = 1, 2, .., d$.

$$X_j^l = 1 \text{ if } h_l(j) = h_l(i), \text{ else } X_j^l = 0$$

- Define $Y = \sum_{j \neq i} f_j X_j^l$. Then $T(l, h_l(i)) = f_i + Y$.

# Count-Min Sketch:Point Query

$$
\begin{aligned}
E[Y] &= \sum_{j \neq i} E[f_j X_j^l] = \sum_{j \neq i} f_j E[X_j^l] \\
&= \sum_{j \neq i} f_j Prob(h_l(j) = h_l(i)) \\
&= \sum_{j \neq i} \frac{f_j}{w} \text{ (} h \text{ is picked from a pair-wise family)} \\
&\leq \frac{||\mathbf{f}||_1}{w}
\end{aligned}
$$

# Count-Min Sketch:Point Query

$$\begin{aligned}
Prob(T(l, h_l(i))] > f_i + \epsilon||\mathbf{f}||_1) &= Prob(Y \geq \epsilon||\mathbf{f}||_1) \\
&= Prob(Y > w\epsilon E[Y]) \\
&\leq \frac{1}{w\epsilon} \quad \text{(By Markov Inequality)} \\
&= \frac{1}{e} \quad \text{(since } w = \frac{e}{\epsilon}\text{)}
\end{aligned}$$

# Count-Min Sketch:Point Query

$$
\begin{aligned}
& Prob\left(\min_{l=1}^{d} T(l,h_l(i))] > f_i + \epsilon||\mathbf{f}||_1\right) \\
& = Prob\left(\bigcap_{l=1}^{d} T(l,h_l(i))] > f_i + \epsilon||\mathbf{f}||_1\right) \\
& = \prod_{l=1}^{d} Prob\,(T(l,h_l(i))] > f_i + \epsilon||\mathbf{f}||_1) \leq \left(\frac{1}{e}\right)^{\ln\frac{1}{\delta}} = \delta
\end{aligned}
$$

- Hence $Prob\left(\min_{l=1}^{d} T(l,h_l(i))] \leq f_i + \epsilon||\mathbf{f}||_1\right) \geq 1-\delta$.
- Therefore $f_i \leq \hat{f}_i \leq f_i + \epsilon||\mathbf{f}||_1$ with probability $\geq 1-\delta$.
- Space$= O(wd) = O(\frac{1}{\epsilon}\ln\frac{1}{\delta})$.

# Count-Min Sketch:Heavy Hitter

- Set $\delta' = \frac{\delta}{n}$, using space $O(\frac{1}{\epsilon}\ln\frac{n}{\delta})$ obtain estimates such that "For All *is* $f_i \le \hat{f}_i \le f_i + \epsilon m$.

- Use a min-heap to store the heavy-hitters.
  1. Keep a count on the total number of elements $m$ arrived so far.
  2. When item $A(i)$ arrives, compute its estimated frequency from the count-min sketch data structure.
  3. If the count is above $\frac{m}{k}$, insert it in the heap with key $Count(A(i))$, and delete any previous occurrence of $A(i)$ from the heap.
  4. If any element in the heap has count less than $\frac{m}{k}$ delete it through operations such as *Find-Min* and *Extract*-Min.
  5. Assuming no large error happens in the Count-Min sketch, the heap size is bounded by $2k$. Why? Therefore extra work per item to process the heap is $O(\log k)$.
  6. At the end, scan the heap, and for every item whose estimated frequency is $\geq \frac{m}{k}$ return it as a heavy hitter.

# Count-Min Sketch:Heavy Hitter

- Set $\delta' = \frac{\delta}{n}$, using space $O(\frac{1}{\epsilon} \ln \frac{n}{\delta})$ obtain estimates such that "For All $i$s $f_i \leq \hat{f}_i \leq f_i + \epsilon m$.
- Set $\delta' = \frac{\delta}{m*n}$, using space $O(\frac{1}{\epsilon} \ln \frac{m*n}{\delta}) = O(\frac{1}{\epsilon} \ln \frac{m}{\delta})$ obtain estimates such that "For All $t = 1, 2, .., m$s the estimated frequency is within the error-range.
- Use a min-heap to store the heavy-hitters.
  1. Keep a count on the total number of elements $m$ arrived so far.
  2. When item $A(i)$ arrives, compute its estimated frequency from the count-min sketch data structure.
  3. If the count is above $\frac{m}{k}$, insert it in the heap with key $Count(A(i))$, and delete any previous occurrence of $A(i)$ from the heap.
  4. If any element in the heap has count less than $\frac{m}{k}$ delete it through operations such as *Find-Min* and *Extract*-Min.
  5. Assuming no large error happens in the Count-Min sketch, the heap size is bounded by $2k$. Why? Therefore extra work per item to process the heap is $O(\log k)$.
  6. At the end, scan the heap, and for every item whose estimated frequency is $\geq \frac{m}{k}$ return it as a heavy hitter.

# Miscelleneous

- Implementation: `http://www.cs.rutgers.edu/~muthu/massdal-code-index.html`
- Twitter's algebird and ClearSpring's stream-lib offer implementations of Count-Min sketch and various other data structures applicable for stream mining applications.
- Application: Mostly a list of papers that use CM-sketch
  - `http://sites.google.com/site/countminsketch/cm-eclectics`
  - `http://sites.google.com/site/countminsketch/compressed-sensing`
  - `http://sites.google.com/site/countminsketch/databases`

# Mini Exercise [Due Oct 31st]

- Implement Count Min Sketch and plot the frequency of all elements as reported by the Count Min sketch data structure as well as their true frequencies using $\varepsilon=0.01$ and number of hash functions=25.
  - Data: consider a stream of size 1000000 where each element in the stream arrives from [1,1000] chosen uniformly at random.